Intentions and Operations: A Plan of Concentration

By Logan Davis

Graduation Date: May 2017

Marlboro College, Marlboro, Vermont

Sponsors:
Jim Mahoney - John Sheehy - Matan Rubinstein

Outside Examiner:
Brandt Kurowski

Plan Components:

- The design and implementation of an embedded systems language : 40%
- A modular, interactive memoir incorporating animated text, sound, and video : 30%
- A paper discussing the process of writing the memoir (AKA: glue/meta/intro/transition paper) : 10%
- Examinations in algorithms, and programming languages : 20%

All materials and included source code is under the following license:

Contents:

Introduction:

This body of work is a reflection, introspection and exploration of creating text based works as someone who is dyslexic. The contents of this paper will deal with language, both as a reader (of anything) and a writer (of code).

Acknowledgments:

I would like to thank my family for their support through my four years at Marlboro. From the first time I called in tears to now.

I would like to thank Jim Mahoney, John Sheehy, and Matan Rubinstein for their mentorship and kindness through my education.

I would like to thank my friends with a special acknowledgment to Krystal Graybeal and Sophie Gorjance for their help with *Release*, both as early testers and voice actors.

Lastly, I would like to thank the late Bruce Davis. He taught me what is means to be family; who allowed me to find confidence in the fact that I won't always know everything; helped me find and afford my love of learning; showed grace in acceptance of grief; and made sure that what I studied was also what I love.

Release: A Modular Memoir:

Due to the nature of the piece, it cannot be directly included in this paper.

- To see *Release*, please go to http://csmarlboro.org/ldavis
- All source code is maintained at https://github.com/OkayAlright/modular-paper

Why I Wrote Release:

Written words confuse me. I cannot make sense of Gravity's Rainbow. Latin gives me a head ache. I have never had the courage to read Moby Dick. I am dyslexic. I have been told to not identify with my disability, but if it affects how I intake the majority of information in my life, how could I not feel like it has become some kind of constant veneer.

In learning to program, my dyslexia constructed some massive hurdles that, frankly, almost made me quit computers three years ago. I have wanted to write about my experience since then, but I never had the forum to really dig into it.

My experience with programming eased as have my troubles with dyslexia. To say that it isn't a problem anymore would be akin to saying that I felt I have totally represented my experience in what you are about to read: a lie. I still have trouble with reading other people's code and I still struggle with typos. But I feel like this attempt to convey my experience has someone who struggles with reading and writing is adequate as a starting point.

A Reflection on Writing It:

In trying to deal with my troubles through coding, I wanted to write some personal reflection. I attempted a few drafts with minimal success. They were stories of my first class in programming, how I was a bad student, and how I broke my back teaching myself C++ over the summer to rejoin in the Fall semester as someone who could write a reasonable fizzbuzz program. They all fell flat. John Sheehy looked at my one day and said "don't just tell me, show me." I chewed on that for a while.

I have had a passing interest in Jorge Borges's **Library of Babel**.[1] The construction of a simple system and a complex result has always been interesting to me. I feel that Borges's short story is a pinnacle example of this kind of process. There was a website I found about two years ago that is dedicated to this book.What interested me most was an effect it uses on the landing page menu; the text increments in a constant stream upward through the ASCII values.[2] I had done some papers as straight HTML before, so making a paper to incorporate this as a way to "show" my experience felt fairly straight forward.

I brought a few examples to John the week later. That afternoon I was told to write text to incorporate the visual trick. At this time, the paper was still a linear project in my mind.

Around this time though, near my birthday, I got news that a friend from my childhood was found dead. I had to process it. I had been with him through what where some of the most trying and conflicted parts of my life. He left just as I entered high school, but by that point he was one of my dearest friends. We lost touch. Still, in thinking about how I ended up at Marlboro and how I dealt with challenges the way that I did, he proved inseparable from my internal dialogue. His laugh, his smile, and his fairness… it was a part of me that made its absence know. I drank about it. I cried about. I wrote about it. That became the first drafted section of this memoir.

Release was originally just going to be a linear project. However, when writing it I remembered a talk that I sat through, a talk on Matan Rubinstein's **Le Invisibili**, a modular music composition.[3] I though about modularity in terms of my own writing and if it had a place in **Release**. The dyslexic effect, though interesting, undersold the entire situation. Though a great many hours went into sorting out my feelings and coming to terms with my limits (at the moment), I had so many unresolved feeling about my childhood. Most of these came from the fact that I remember very little of them. Either from trauma or medication, most of my memory between ages 7 and 14 are gone. As some of the evaluations in the memoir even mention my memory was not a strong skill. This modularity in my memoir it expresses an unreliability to what happened or how I felt/feel.

Embedding this conflict in the text required me to leverage events in my life that I have not revisited in years. The modularity of the text is a literal representation of my own inability to settle on a feeling. Writing in this style was a much of an engineering hurdle as it was an emotional one. My emotions drove the writing which became restricted by the engineering which stretched to express my emotions which where fostered by my engineering. It was a mismatched Matryoshka doll flipping in between sets. One set was my ability to ask if something was feasible with my coding ability, the other was whether I felt okay putting this stuff down on paper.

I don't want to sell this as a deeply traumatic process, but to honestly talk about each part of my life this memoir touches upon, I had to think about questions I never answered. Was I happy during that time? Did my friends really enjoy me being around? Did I deserve their time? I won't ever have a solid answer to these questions and that is what best fed the modularity of this piece.

The first jab at making the memoir modular was to break apart each event I was writing about out as a discrete section of writing, rather than typing them all in a single document. Each of these sections were held as individual HTML pages that could be linked together in complex ways. These sections ended up falling into three different catagories:

- Present Tense: pieces written in present tense that would feature text switching and dyslexic effects.
- Past Tense: Reflections on themes brought up throughout the other pieces would be static text.
- Evaluations: Photo copies of education evaluations from my childhood.

The three types of sections would be three different reading styles and mechanics.

At this point, due to the Dyslexic text manipulations and the modular separation of the sections, the paper was unprintable (it would miss the point) so my freedom to leverage anything HTML and Javascript could do was unbounded. I quickly began creating a similar system to the dyslexic effect for the modular switching: every half second or so, an interval would trigger which would roll a probability to see if the effect took place. Where they differ is in what that effect does when applied.

The dyslexic effect scrambles any word that isn't blacklisted in the script (such as tags):

```javascript
function shuffleWord(word){
    // adapted from http://stackoverfl
    // ow.com/questions/6274339/how-can
    // -i-shuffle-an-array-in-javascript

    if((Math.random() > 0.98) &&
       (word.match("<br>|<p>|</p>") == null)){
        var j, x, i;
        var a = word.split("");
        for (i = a.length; i; i--) {
            j = Math.floor(Math.random() * i);
            x = a[i - 1];
            a[i - 1] = a[j];
            a[j] = x;
        }
        return a.join("")
    } else {
        return word
    }
}
```

In a different measure, the modular switching has intervals tied to each modular section. When one triggers, it randomly pulls out a choice from a list of phrases that could by interchanged with the one being switched. This new phrase in inserted instead:

```
JavaScript
function chooseFromList(list_of_choices){
    return list_of_choices[Math.floor(Math.random() * list_of_choices.length)];
}
function get_and_switch(list_of_choices,id_tag){
    if(Math.random() > 0.9){
        document.getElementById(id_tag).innerHTML = chooseFromList(list_of_choices)
    }
}
```

The **list_of_choices** that are used for switching text is stored as a collection of lists loaded and attached to specific tags in the text (this tags are often just <span> tags with some id fields). They are bound by a massive switch-case statement for setting intervals once a section is loaded.

So at this point the paper had modular sections of text that would switch out as one is reading and a dyslexic effect to scramble words in certain areas. The hyperlinked connection between pieces still felt wrong. Being able to consider each of these stories as discrete sections without reference to what it lead to and from, in a visual sense, weakened the connections. I decided to make the transitions between sections an expansion of text on a single webpage instead of a series of disjointed ones. This drastically complicated the systems that I had previously implemented.

The benefit of each sections being a separate HTML page is that they were isolated and couldn't stomp on each other. Their architecure allowed for a lot of code reuse between pieces, but to keep them all working on the same page, I had to change the tagging system for the switching and shuffling text to be unique in relation to the whole piece rather than just the section it was a part of. The way in which the text was expanded into the web page also was a complicated matter.

Originally I thought about using nested **iframes** to drop in new sections of text. The **iframes** would allow easy management of connections and avoid the entire problem of isolation mentioned in the previous paragraph. Unfortunately they proved to be greatly restricted (for security reasons) and under performant. I needed to treat the entire piece as a single, dynamic document.

Now there are many different frameworks for making single-page applications with Javascript, but given that everything else I had made worked without a framework and I wanted to keep this project as code-minimal as possible, I decided to code this functionality myself. Each connection between papers would need three tagged sections: a trigger (to insert the new piece), a gate (where to insert the piece), and a tag after the gate to mark all HTML after an inserted piece.

As far as getting the text to be injected, there was a little more creativity. I thought about AJAX calls to request the needed text from a server. But in my experience, if I am trying to read something and I have to wait a few moments for something to load on the screen (that should just be text), I bounce. If I am reading and that reading is broken by the simple structure of the piece, every second the page spends loading new content is a chance for my attention to be lost. It couldn’t be affected by network speeds.

Thankfully, rather recently, Javascript has gotten support for multi-line strings by use of *templates*. If you want to store a multi-line string is JS, you just use three back-ticks instead of quotes:

````JavaScript
```
this
is
a
multi-line
string
```
````

So I could store each section as a multi-line string with HTML tags and all. This allowed me to use a lot of the same code for switching modular text to insert new sections. While I was making the modular sections, I wanted to make use of the fact that, since the piece had to be viewed on a computer, I had the ability to project sound over the text. Originally I wanted this to be generative audio changing depending on the path you took through the piece and the shuffling/switching that occurred at a given moment. But after some discussion with John and Matan, we agreed that it just became too stressful of a noise while reading. I pivoted to recordings of readings of the piece. The evaluation pieces would be spoken over present tense pieces and the the past tense pieces would be played over evaluations. There was some question on how to do this because, as I decided to move forward on the idea, the Webkit repository was corrupted as a result of the SHA-1 collision reported a few days prior.[4] Webkit Audio was what most of my knowledge on web audio required, so I ended up just using the Javascript **Audio** object to store all the sound files. These would wait around in memory to wait for the same signal that sets all the switch/shuffle intervals for a new section.

What the code currently living at http://csmarlboro.org/ldavis represents is the evolution of me finding ways to articulate myself and my experience as a person who happens to have dyslexia. The project’s code base might be nicer if I used X, Y, or Z frame works. The rendering might be more consistent if I packaged it as a Python GUI application. It might last longer in an archive if I had made it printable material. But this was not a planned out architecture. I had no development model for the memoir. The code, the process, and product are a snap shot to the explorations I am taking in attempts to not just *say* what I experience, but to *show* it.

Future projects:

So this memoir, other than being a vessel for my reflections on being dyslexic, was an exploration of writing outside of the bound of paper. The level of modularity and unreliability that this piece attained would not have been possible on paper. However, the subject of this memoir and the time restraints of plan kept areas from being explored. Further writings in this form will look to leverage what this piece did not touch at all or leverage entirely.

Time is the least used aspect of this writing that could have been utilized. The memoir might be non linear and might cite dates from far apart times, but they were all released at the same time. Given the automation capabilities of a computer, there is no reason that the sections couldn't be time-independent in their release or their connections to each other.

Other aspects involve generation of entirely dynamic text, generative sound interfaces, and further non linearity of text. These are topics I plan to tackle in future writings.

Bibliography:

- [1]: Borges, Jorge Luis. "The Library of Babel." Collected Fictions. Trans. Andrew Hurley. NewYork: Penguin, 1998.
- [2]: Basile, Jonathan. "Library of Babel." *libraryofbabel.info*. Accessed: June, 2015. https://libraryofbabel.info
- [3]: Rubinstein, Matan. "Le Invisibili." Lecture, Family Day Faculty Lectures from Marlboro College, Marlboro, VT, October, 2013.
- [4]: Goodin, Dan. "Watershed SHA1 collision just broke the WebKit repository, others may follow." *Ars Technica*. Published: February 24, 2017. https://arstechnica.com/security/2017/02/watershed-sha1-collision-just-broke-the-webkit-repository-others-may-follow/

The Connection to The Compiler:

I will describe MinNo, the language I created, in a much more deeply technical manner shortly. But I feel some segue is required from the memoir to the compiler.

MinNo is meant to allow another avenue into Arduino programming. It is not some silver bullet in teaching someone how to program. It is not the end all be all of Arduino development. I won't even claim it is in any way an objective improvement over the languages already available to Arduino programmers. MinNo is the language that I wanted when I learned Arduino programming. It makes the assumptions that I wanted when I started. It is not the most universally readable. It is not the most logically simplistic. It is the language that makes the most sense to me. If the language serves me, than it should serve people like me. If that allows others to get into programming in an easier way than I did, then the project will have been worth it. I won't claim that this project brings a lot of new ideas to the table, but it is new in the fact that is it a language that makes intuitive sense to me.

MinNo: An Arduino Language Compiler

All writing related to the compiler will be included in this document. The source code for the compiler can be found here:

- https://github.com/OkayAlright/minno

There are three compiler documnets included in this paper:

- *"Why MinNo?"*: A more technically focuesed overview of the language and it's concepts.
- *"Making MinNo"*: A reflective piece in making the language and compiler.
- Documentation: Instructions and guides for the language.

Why MinNo?

I didn't have an easy time learning how to code when I first started. One of the largest reasons I had trouble was because of my first language: Python.[1]

Though Python has many strengths, I would not say consistency is not among them. This is a pervasive problem from the language's core mechanics (string methods being in place or returning new strings for example) to how any two given people try to teach the same idea (take these two answers to similar questions for example: 1 & 2.[2][3] One of Python's strengths is that it allows you to do one specific thing in ten different ways, but I am someone who likes to hear a single thing explained multiple ways. Though I was able to overcome my problems with Python, when I began toying around with the Arduino platform, I found the same problem with their home-grown language.[4][5]

The Arduino language (AKA Processing, Wiring, C/C++) is a language built on top of C++ but only implements a subset of the C standard library (but none of C++'s library, just using it's syntax and constructs).[6][7][8][9] There are also a fair amount of Arduino/AVR specific functions and global variables sprinkled about. Because of this mash up that is supposed to be a beginner platform and a collection of C functions resembling a K&R era implementation of the language, you get coders running the gamut in style and techniques. On one end you have some hobbyists making really high level sketches with sophisticated libraries to run wifi-shields for their Arduino Uno project in just a few lines of code.[10] On the other side you have some old school hackers writing some of the most PreProcessor-Macro'ed, bit twiddling, interrupt throwing files you could imagine.[11] If you aren't brought up to speed on C and Arduino by some guide that was provided by education companies like Sparkfun, you can find yourself in a hexadecimal-encoded hell within a matter of clicks.[12] MinNo is an attempt to create a more humane (for people like myself) entry point to the Arduino platform.

CPU Architecture: Harvard vs. Von Neumann

Most interest in processor architecture lies in standardized implementations (x86 and ARM). The conversation at this level typically is a discussion of instruction sets and bus implementation. AVR processors take it a step further back and discuss the basic organization of a computer. X86 and ARM architectures are organized in a manner originally specified by John Von Neumann: a central processing unit, memory, some form of input, some form of output, and long term storage across a set of shared buses.[13] AVR processors (generally) are organized in a manner consistent with Harvard specification (more accurately, the modified-Harvard spec).[14] The main difference between Von Neumann and Harvard is that Harvard separates memory into two different pools: instruction memory and data memory, which attach to the CPU via entirely separate buses. Harvard processors can simultaneously read and write to both memory pools because they do not share a bus. The two pools don't even need to be indexed and architectured the same. Often (as is the case with AVR processors), instruction memory is read/write accessible at runtime, data memory isn't. Data memory is often used for long-term, initialization data: tables, addresses, arrays, keys, and so on. Both the data and instruction memory pools are implemented as EPROM/Flash storage . AVR processors have general SRAM for data that needs to be accessed more quickly (in place of caches).

One of the problems with using C to program Harvard processors is that most C compiler implementations (and subsequently Arduino's language) have grown under the ideas of Von Neumann architecture. All memory accessors and management systems, property files, and object oriented idioms assume the computer they are running on has one shared pool of read/write memory. To allow the benefits of Harvard's memory architecture, we have to pollute the global name space with keywords to specifically access the data memory pool (such as declaration containing "const PROGMEM" is Arduino sketches).

MinNo is an attempt to embedded Harvard’s distinction in architecture at the syntactic level. By default, all declarations of variables are immutable unless they contain the “mutable” tag. If the compiler sees an immutable declaration, it will be put in “program memory” (Arduino’s name for data memory). This allows for a more concise memory placement syntax to counteract MinNo’s slightly more verbose declaration structure. This thought of embedding Harvards design choices in the syntax is a consideration throughout the languages idioms and structures.

Static Memory Allocation

Culturally, among novice programers, pointers and manual memory management stand as the mysterious and often frightening figure when writing software.[15] It is easy to misuse them and hard to figure out why they aren’t working (if you aren’t practiced in debugging them). Their inclusion in the Arduino language stands as an oddity to me for that reason, other than performance restrictions that prohibit a garbage collection system. Why would a platform bragging about beginner friendliness use one of the most notoriously confusing memory management systems? Given that garbage collection is out of the question due to the lower power CPUs found on Arduinos and that MinNo is supposed to be an alternative beginner language, I have decided (at least for now) to have no dynamically growing memory in MinNo: everything is statically allocated.

This may initially seem like a huge drawback, and in general programming I would agree. But in embedded systems, dynamic memory bugs can be subtle, incredibly hard to reproduce, and absolutely system crashing . Early in my time programming sketches, I swore off the use of malloc and free in favor of static memory declarations. From synthesizers, to controllers, to IR navigating robots, I have gotten away without using dynamic heap allocations. To anyone reading this, for your sanity, I would suggest a similar shift in coding style. Others also suggest minimizing dynamic allocation where you can.[16]

With purely static allocation, it also allows for far better utilization of the program memory pool. An added bonus in such a memory restricted system.

Template Operations vs. Stateful Procedures

MinNo is meant to emphasize a more template based form of thinking about code; the code represents templates operating on data. A hard separation of data and code should be taken where it can. Intermediary values and indexing values are, of course, held in read/write instruction memory, but these values can be thought of as markers and states in templates. Indexes are markers as to what templates are being applied to. Intermediate values are states in between templates operations. This mind set keep the separation of memory pools more consistently present and should lead one to minimize stateful code as to avoid unnecessary complexity.

Getting Under The Hood

Now there are absolutely times when this distinction of memory can be far more of a hinderance than a benefit. Sometimes we really need manual heap allocation. MinNo's compiler attempts to generate (subjectively) readable, tabbed C code. This will allow boiler plate functions and code to be generated, but the resulting code can be opened and optimized/altered to allow for malloc's, free's and a things C-like. It also allows for a bridge from those who have been using MinNo to transition into C in a low-stakes, take-it-at-your-pace manner.

For example, leveraging integer sizes to cut down of code to handle looping over collections is fairly common in Arduino programming, but something MinNo doesn't handle very well (yet). An example of this could be indexing through a waveform to be send out via an analog port on an Arduino board:

```
int squareWave[256] = {0};
byte i = 0; //8-bit integer type in Arduino

//some code to correctly assign the last 128 indexes of squareWave to 255

while(0){
    analogWrite(squareWave[i++]);
}
```

The reason this works is because whenever **i** equals 255 (the last index of **squareWave**), on the next increment it overflows back to 0 because it is an 8-bit value. This kind of trick is currently not directly possible in MinNo. To get a similar effect, your code would look like this:

```
let squareWave: array[int] = [0,0,0,0,0 ....Then the other 250 integer values];
let i : mutable int = 0;
while 0 {
    analogWrite squareWave[i] ;
    i = i + 1;
    if i >= 255 {
        i = 0;
    }
}
```

Though this is possibly more readable (less bit-length trickery), it is quite a few more instructions and would take quite a few more cycles to complete. Instead I can compile the above function which would produce the C code:

```
const PROGMEM int squareWave[] = { 0 , 0 , 0 , 0 , 0... the other 250 values} ;
int i = 0 ;
while(0 ){
    analogWrite(pgm_read_word(&squareWave[i])) ;
    i = i + 1 ;
    if(i >= 255 ){
        i = 0 ;
   }
}
```

Then I can get rid of the wrapping logic and change the type of **i** from **int** to **byte**:

```
const PROGMEM int squareWave[] = { 0 , 0 , 0 , 0 , 0... the other 250 values} ;
byte i = 0 ;
while(0 ){
    analogWrite(pgm_read_word(&squareWave[i++])) ;
}
```

Restrictions like this are something I know is a shortcoming of the language can I plan to address them in short order. But tricks like these, I would venture to say, are only in ~10% of the code I write. If MinNo allows me to more easily complete the other 90% and more mindfully consider when I really need to leverage tricks for performance over writing more understandable code, than it has done it's job.

State of The Language:

Currently MinNo is in an *Alpha* state and, though I find it useful for personal work, I would not call it production ready by any means. Some more consideration on both the ideas of the language and the construction of the compiler are subject to change. Wider support different bit-length integers; structs; optional safer-pointers; and Foreign Function Interfaces (for direct C code) are all being considered on the language side. The compiler needs a semantic verifier in the very near future. This would check arity, make sure no attempted re-assignments to constants occur and so. When you actually upload the compiled MinNo script, GCC does all of this for you, but all the messages reference the translated C and not the actual MinNo source file. This is workable for now but by no means is it ideal, and it will become a bigger problem with another goal for the compiler.

The LLVM community has been doing a lot of work on mainlining AVR support into their codebase. For those who don't know, LLVM is a compiler framework that allows for a single target language (the LLVM Intermediate Representation) to be optimized and compiled down to machine code for numerous platforms. If LLVM gets AVR support, MinNo will be able to benefit from the vast amount of tool for LLVM compiler and possibly pivot to supporting other platforms with minimal rewriting of core compiler components. This would be a pivot away from GCC which is why a proper semantic verifier is so important.

I don't plan to try to make MinNo more functional-oriented or construct some system to ensure type safety. I want a language to cut down on the code I have to write because the language I have been using makes the incorrect assumptions (like C's assumption of a single memory bus). Given that fact, expect the progression of the language to reflect as such.

Getting Involved:

Though I am not taking direct outside contributions to the project right now, I would love feed back on the language. MinNo, it's compiler, and the documentation are all under the MIT License, so if you like the idea but want to run a different direction with it, please feel free to fork the source code. If you have any questions or just want to talk about the language or anything I have mentioned here, please feel free to contact me:

- email: ldavis@marlboro.edu
- twitter: @Death_by_kelp

Bibliography:

- [1]: "Welcome to Python.org." *python.org*. Accessed: April 16th, 2017. https://www.python.org/.

- [2]: gregjor. "How to determine a variable's type?." *stackoverflow.com*. Published: December 31st, 2008. http://stackoverflow.com/questions/402504/how-to-determine-the-variable-type-in-python.

- [3]: Fredrik Johansson. "What's the canonical way to check for type in python?." *stackoverflow.com*. Published: September 30th, 2008. http://stackoverflow.com/questions/152580/whats-the-canonical-way-to-check-for-type-in-python.

- [4]: "Arduino - Home." *arduino.cc*. Accessed: April 16th, 2017. https://www.arduino.cc/.

- [5]: "Arduino - Reference." *arduino.cc*. Accessed: April 16th, 2017. https://www.arduino.cc/en/Reference/HomePage.

- [6]: "GitHub - arduino/Arduino: open-source electronics prototyping platform." *arduino.cc*. Accessed: April 16th, 2017. https://github.com/arduino/Arduino/.

- [7]: "Arduino Playground - Processing." *arduino.cc*. Accessed: April 16th, 2017. http://playground.arduino.cc/Interfacing/Processing.

- [8]: Mahmoud Saleh. "Introducing Arduino Wiring on Windows 10 IoT Core." *windows.com*. Published:

September 7th, 2016. https://blogs.windows.com/buildingapps/2016/09/07/introducing-arduino-wiring-on-windows-10-iot-core/#vYEu3yjHXGsS9ZXE.97.

- [9]: lloyddean. “What is the language you type in the Arduino IDE?.” *arduino.cc*. Published: December 8th, 2010. http://forum.arduino.cc/index.php?topic=45492.0.
- [10]: “Arduino - ArduinoWiFiShield101 .” *arduino.cc*. Accessed: April 16th, 2017. https://www.arduino.cc/en/Guide/ArduinoWiFiShield101.
- [11]: Can_I_Trade?. “Quick And Dirty Synth For Arduino Due.” *rcarduino.blogspot.com*. Published: November 30, 2012. http://rcarduino.blogspot.com/2012/11/quick-and-dirty-synth-for-arduino-due.html.
- [12]: “Sparkfun Inventor’s Kit Guide.” *Sparkfun*. Accessed: April 16th, 2017. https://cdn.sparkfun.com/datasheets/Kits/SFE03-0012-SIK.Guide-300dpi-01.pdf.
- [13]: Igor Kholodov. “The von Neumann Computer Model.” *Bristol Community College*. Accessed: April 16th, 2017. http://www.c-jump.com/CIS77/CPU/VonNeumann/lecture.html.
- [14]: Student Fredrick. “Processor architectures: Harvard, von Neumann and Modified Harvard architectures.” *embeddedknowledge.blogspot.com*. Published: February 05, 2010. http://embeddedknowledge.blogspot.com/2010/02/processor-architectures-harvard-von.html.
- [15]: jkerian. “What do people find difficult about C pointers.” *stackoverflow.com*. Published: October 26th, 2010. http://stackoverflow.com/questions/4025768/what-do-people-find-difficult-about-c-pointers.
- [16]: mdiaconescu. “Optimize Arduino Memory Usage.” *web-engineering.info*. Published: July 27th, 2015. http://web-engineering.info/node/30.

Making MinNo: Lexers, Parsers, and A Whole Mess of Parenthesis

By Logan Davis Last Updated: 4/17/17

Abstract:

MinNo is a compiled language targeting the Arduino platform. This paper, as part of my Plan of Concentration, is an overview of the technologies and techniques used in making MinNo's compiler as well as a collection of opinions of the Racket language and ecosystem (MinNo's implementation language).

Introduction & Overview:

In my junior year at Marlboro College, I wanted to make a synthesizer in one of my classes using Arduino development boards. The Arduino is a small, resource limited computer that allows written programs to directly control voltages put out by pins. They are cheap, readily available, and run on a 5 volt power supply. They seemed to fit the bill for making a custom signal generator for a larger synth.

Sadly, 16mhz, the frequency clock of the Arduino's processor, was slower than I could have anticipated. If I remember, the first version could only produce up to ~200 hertz (AKA lower than the 5th lowest string of a standard guitar on in other words "no where near acceptable"). So I began hacking away at the Arduino platform trying to learn every trick to get the signal generator to run faster. Look-up tables, magic hex values, and hidden registers. Ever shortcut I could think of was hacked into this thing. By the end, I had a signal generator that could put out ~20,000 hertz, but the code was painful to maintain by hand. One of the core problems, in my mind, was that most of the hacks I have to make were because the Arduinio's homebrew language was based off of C, an x*86/ARM centric language, and the Arduino doesn't just use a different implementation architecture (x*86 vs ARM), it belongs to a different architecture family (Harvard vs. Von Neumann). Most of the code I added was accessing some special register or some secret memory pool to x_86 programmers, but these were all simple and readily available parts of the Arduino (more accurately AVR) CPU (central processing unit). After the synth project, I realized that I didn't want to build on it for my Plan of Concentration. I wanted to make a better language to avoid so much hoop-jumping in the future. This is how MinNo began.

MinNo is a language and compiler targeting Arduino-C that is meant to handle some of the more common idiosyncrasies of AVR programming. For more on how MinNo attempts this, please see *Why MinNo*.

After toying with a few different language ideas, I had to actually get down to programming the compiler. This starts with making a *lexer*. A lexer classifies words and symbols in a source file and returns *tokens* for them. (it's the part of the compiler that says the symbol *1* is a number unless it's wrapped in string quotes). This classifying of words is in preparation for the *parser* which recognizes series of tokens based on a language grammar. At this step to compiler recognizes *for(int i = 0; i < 10 ; i++)…* as a loop but not that it will stop at *i == 9*. What the parser produces is an Abstract Syntax Tree (AST), which is a structured representation of some source code file as manipulatable data. From an AST, I needed some *translator* to turn my languages AST into one that better matches an Arduino-C equivalent AST. These are alterations such as cataloging and what can and cannot be put into the different memory modules on an Arduino board. To actually output it as text file while making a few formatting alterations, the translated AST is given to an *unpacker*. This paper will go over the creation of each of these parts, the tools that helped to create them, and a thoughts that I know have after getting this far with the project.

Racket: A LISP for Language Design

For the lexer, parser, translator, and various tools of MinNo, I chose to use Racket. Formerly known as PLT scheme, Racket is one of the many opinionated and highly idiosyncratic members of the LISP family of languages.[1] Where CLISP emphasizes out-of-the-box productivity, Clojure focuses on functional constructions, and Scheme strives towards simplicity, Racket has built itself around specificity.[2][3]

Originally, Racket was an education centric platform. It's homegrown IDE, ***DrRacket*** allows for you to turn parts of the language on and off to slowly introduce new coders to extended functionality in small steps. Ultimately, prior to my time using the language, they still wanted to keep it as an educational platform but they wanted also wanted to extend to a wider community. Racket made a concerted effort to shift it's strengths.

Now Racket markets itself as a "language platform." The entire ecosystem is built around the idea of embedding Domain Specific Languages (DSLs) into programs to simplify and better manage sub tasks of larger programs. To achieve this focus, the developers of the language have included lexers, parsers, and an extensive macro system.

With Racket's ability to handling syntax parsing and my previous experience with it in general programming, it came as a natural choice to use when implementing the MinNo compiler.

The Lexer Itself: The Matt Might Method

In creating this compiler, by far the most helpful resource was Matt Might's blog posts about his course on the subject.[4] The lessons are brief, but give an excellent overview of what one needs to do to make a compiler. I most heavily leaned on his course syllabus and class notes for MinNo's lexer.

A lexer is the section of a language compiler (or interpreter) that takes a source file and breaks it down into the languages smallest pieces (tokens). For example: if we were making a compiler for a basic calculator, the lexer would need to recognize “=”, “+”, “*”, “-”, “/”, and any integer or float. The lexer would need to return these tokens in some form that allows the parser (a later step) to easily recognize them and construct larger grammar objects in the language.

Regular Expressions and Their Use

The most typical way to construct a lexer for non-trivial syntax structures is to strap together a collection of regular expressions into some class or function that consumes a source file and returns a collection of tokens.

Racket, being a language platform, has an included lexer & regex library. Matt Might thinks highly of it and uses it in his own compiler course.[5] An example of how to construct a lexer in Racket, using our basic calculator language, may look something like the following:

```
(define calc-lex
  (lexer
    [#\+   '('ADD-OP lexeme)]
    [#\-   '('SUB-OP lexeme)]
    [#\*   '('MULT-OP lexeme)]
    [#\/   '('DIV-OP lexeme)]
    [(+ (char-range #\0 #\9)) ('INT lexeme)]
    [(: (+ (char-range #\0 #\9))
        "."
        (+ (char-range #\0 #\9))) (‘FLOAT lexeme)]))
```

Racket prefaces any character with a “#” to mean a *character literal*, so “#+” matches “+” in a source file. **lexeme** is a reserved word by the Racket lexer to reference the currently matched token. The last two entries in the lexer construct are examples of *s-expression regexes*. **+** recognizes 1 or more of the following regex. **char-range** is to match a range of character either in letters or in numbers (#\a to #\z or #\0 to #\9 for example). **:** concatenates all following expressions into one single pattern. Longer regular expressions are typically defined as lexer abbreviations to look a little more clean. With those abbreviations defined the lexer would look like this:

```
(define-lex-abbrev int? (+ (char-range #\0 #\9)))
(define-lex-abbrev float? (: (+ (char-range #\0 #\9))
                             "."
                             (+ (char-range #\0 #\9))))
(define calc-lex
  (lexer
    [#\+   '('ADD-OP lexeme)]
    [#\-   '('SUB-OP lexeme)]
    [#\*   '('MULT-OP lexeme)]
    [#\/   '('DIV-OP lexeme)]
    [int? ('INT lexeme)]
    [float? (‘FLOAT lexeme)]))
```

Given a source file containing “5 + 7.0 / 3” would produce a token stream of:

```
'('INT "5") '('ADD-OP "+") '('FLOAT "7.0") '('DIV-OP "/") '('INT "3")
```

The resulting lexer would be used to listen to an input-port and produce a token stream for the parser to consume as needed.

Token Construction and MetaData

The lexer can do more than just recognize tokens. It can help provide invaluable debugging information about the source file it is lexing. For parsing libraries like the one I used (which I will get to next), it expects the lexer to append the location of tokens in the source file being compiled to use when reporting a parsing error. The lexer can also be a place to catch stylistic warnings (such as rustc’s warning about anything other than *snake_case*).[6]

The State of MinNo’s Lexer:

MinNo’s lexer works on an architecture similar to that used by Matt Might to take advantage of Racket’s input ports.[7] The lexer consumes a file-port’s output and gathers it in a buffer. This is recursively done until the source file-port is empty. The resulting collection of tokens are passed to MinNo’s parser.

The lexer works as intended, though some more utility could easily find it’s way into that section of the compiler. constructing global tables of meta data about tokens (to help warnings in the translation step), or stylistic warnings (like I previously mentioned) will all be looked at when revising the lexer.

BRAG: How I Avoided Losing My Life to Making a Parser

Parsing is a topic of exploration large enough to the spend an entire dissertation on.[8] For MinNo, instead of spending the time making one from scratch, or learning the incredibly idiom-centric syntax recognizer in Racket I opted to use a popular Racket tool for grammar specification and parsing: *brag*.

brag is a *Backus Naur form* (BNF) grammar parser implemented as a *Domain Specific Language* within Racket. Before I get into using *brag*, some time should be taken to overview BNF.

Examples of Use and an explanation of BNF:

BNF grammars work using *symbols* and *expressions*. They are organized as such:

SYMBOL ::= EXPRESSION

an expression can be any collection of other symbols or terminals (tokens). If a symbol is in an expression, it is considered a non-terminal expression and must be parsed further. It is worth noting that, in use, some artistic merit is used with BNF notation. For instance, *brag* does not use "::=" for it's equal-sign, it just uses ":" so rules look like:

```
SYMBOL: EXPRESSION
```

Which can be confusing when first looking at it. *brag* allows for a common shim in BNF structures that allows expressions to be regular expression-like statements to allow concise grammar rules. So, to continue with the calc grammar, a *brag* rule to recognize numbers may look like:

```
num: (INT | FLOAT)
```

This rule states a "num" is a integer or a float (as tokenized by our lexer).

When a grammar file is given to the *brag* compiler, it produces a *parse* function. This function will consume a list of tokens and then, using the generated LALR parser, it will construct an abstract syntax tree (AST) abiding by the defined grammar. For instance, here is a snippet of grammar from the MinNo grammar spec for type signatures of variables and functions:

```
; Typing statements
type: TYPE | ARRAY-TYPE lSqBrac TYPE rSqBrac ;;; "int" | "array[int]"
signature: (((id colon type comma)* id colon type) | nonetype)
           arrow (type | nonetype)
```

(Keep in mind that anything is all upper case (like **TYPE**) is caught directly from the lexer while things in lower case are other rules in the grammar.)

The rule **type** is applied id a ‘TYPE token is passed from the lexer or if an **`ARRAY-TYPE** token is followed by a “[” some other 'TYPE and than a “]”. This means things like **int**, **float**, and **array[char]** are caught by this rule. The choice between these two different forms is denoted by the “|” in the rule, which just means “either the thing to my left or the thing to my right.”

The **signature** rules building off of the previous. It is saying that there can be any amount of **id**s followed by a **colon**, a **type**, and then a **comma** (including zero of them), but there must be a single id, followed by a colon and then a type to end the signature. Alternatively all of this can be replaced by a **nonetype**, but one of the two must by there. After this section of the signature (the arguments sections), an **arrow** (“->”) has to be next. This arrow can be followed by another **type** or **nonetype**.

The grammar produces a syntax-object which is handled by the translator.

The State of MinNo’s Grammar:

A few different versions of MinNo’s grammar were toyed around with. Originally, MinNo was going to be of a LISP grammar, but between the limitations a LISP interpreter would have and the shoe-horned-ness of compiling it, I decided against the project.[9] Later iterations took notes from F#, Scala, Java, Python, and, Pyret. The result of those revisions is what you see currently. The largest of the influences are from Scala and F#’s type notations:

F#:

```
let some_int = 5 //variable declaration
let foo x y = x * y //function definition
foo some_int 5 //function invocation
```

Scala:

```
val some_int : Int = 5 //variable declaration
def foo(x: Int, y: Int):Int = { // function definition
    x * y
}
foo(some_int, 6) //function invocation
```

MinNo:

```
let some_int: int = 5; //variable declaration
def foo x:int, y:int -> int { //function definition
    return x * y;
}
foo some_int 6; //function invocation
```

By no means does it reach in any of those directions as far as the language MinNo takes from (the type system is not even a fraction as complicated as Scala's). MinNo just takes some loose form from languages I have used and enjoyed in the past.

The Translator: A Handler Model

To further ease language creation using Racket, the language has a *syntax-object* data type. This is very similar to a quoted expression, but can hold some extra info like lexical bindings and source file location. Here is an example of how to create a quotes expression and a syntax-object:

```
 ; A quoted expression, note that "`" is shorthand for "quote"
 > `( + 1 2)
 '(+ 1 2)

 ; A syntax-object is also the same notation, but a "#" prepends the "`"
 > #`(+ 1 2)
 #<syntax:3:4 (+ 1 2)>
```

The two are very similar and their difference might not be immediately obvious, but in DrRacket, the syntax-object, under inspection, has this meta data attached to it:

```
General Info
Source
'|interactions from an unsaved editor|

Source module
#f

Position
100

Line
3

Column
4

Span
7

Original?
#t

Known properties
'errortrace:annotate
#t
```

Some of this data is a little funky because it is directly extracted from the Racket interpreter panel. But some things do make sense. The first line you type on is the third line of the interpreter because of the welcome message Racket prints out:

```
Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 256 MB.
```

And the syntax-object starts on the 4th column preceded by “> ” of the prompt and the “#`” the denote typing a syntax-object. This meta data what stands out about syntax-object when compared to quotes and quasiquotes in Racket.

Like syntax rules, Racket’s syntax-objects are highly specific to the crowd that is doing cutting edge language research using the platform. They have largely been constructed to make use of Racket’s *reader* system to make interpreted language. I had to make a choice of whether to figure out how to half-use this data type or do something a little more straight forward for my use case. To maintain my sanity, I decided to roll my own template-based method instead. Given MinNo’s rather strict syntax, predicting the grammar that can be used at any point is a programs AST is fairly straight forward.

The only thing that I lose from the not using syntax-objects is the source file location that would allow me to embed translation errors in the translation step. But given that I still have source location in from the lexer, I consider this a non-issue.

Avoiding BNF Hell

A word of warning, this is that area where you start to understand terrible mistakes you made in constructing your grammar or defining your lexer. Those steps can be made to consume and parse some pretty kludgy grammar structures. If they consume kludge they will also produce it. A complicated AST makes for a complicated project. It is worth the hours of revising your BNF grammar and lexer to save days of debugging further down the road.

Handlers:

The way I have decided to tackle MinNo’s AST is to define a collection of handlers for different syntax structures. There are handlers for let statements, block statements, function definitions, conditionals, returns, and so on. Each handler knows how to pass off any sub-syntax structure it contains (such as the statements in a block statement) by using a handler directory. The directory used to dispatch sections of the program to their respective handlers until the program is fully translated. When is left is a new AST closer to that of an Arduino Language program.

As an example of a handler in action, lets step through the translation of a variable declaration. Here is the relevant handler code to understand what is going to happen. Here is an example AST of a variable declaration:

```
(let-statement
  (let "let")
  (id "ledpin")
  (colon ":")
  (type "int")
  (eq "=")
  (statement (expr (term (factor (lit (int "-13"))))))
  (delimit ";"))
```

So this ast is a *let statement* for an immutable variable referred to by the ID *ledpin* which is an integer of “-13”. The first piece of code this AST will be handed to is the handler directory function *tree-transform* which sends off each expression to the correct handler to be translated:

tree-transform from *handerDirector.rkt*:

```
(define tree-transform
  (lambda (datum)
    (define tag (first datum))
    (cond
      ;;;----------------TOP LEVEL PROGRAM-----------------------;;;
      [(equal? tag 'program) (program-tag-handler datum)]
      ;;;----------------LET STATEMENT---------------------------;;;
      [(equal? tag 'let-statement) (let-tag-handler datum)]
      [(equal? tag 'relet-statement) datum]
      ;;;----------------DEFINE STATEMENT------------------------;;;
      [(equal? tag 'define-statement)
       (append (definition-tag-handler datum)
               (scope-statement-handler (fifth datum)))]
      ;;;----------------STATEMENT HANDLER-----------------------;;;
      [(equal? tag 'statment) (statement-handler datum)]
      [(equal? tag 'delimited-statement) (delimited-statement-handler datum)]
      [(equal? tag 'scope-statement) (scope-statement-handler datum)]
      ;;;----------------CONDITIONAL-HANDLER---------------------;;;
      [(equal? tag 'conditional) (conditional-handler datum)]
      ;;;----------------RETURN-HANDLER--------------------------;;;
      [(equal? tag 'return-statement) datum]
      ;;;----------------LOOP-HANDLER----------------------------;;;
      [(equal? tag 'while-loop) datum]
      [(equal? tag 'for-loop) (for-loop-handler datum)])))
```

So given than that first element of the example AST is **`let-statement**, the AST is passed of to **let-tag-hander** given the second conditional branch of the **cond** structure from **tree-transform.** So as an overview, before diving into the code, this is what needs to happen to the example AST to be ready for the unpacker: the variable need to be cataloged to distinguish it from a function ID when unpacking; if a mutable tag is not present, than “const PROGMEM” needs to be appended in the type-string; and if a variable is an array, the notation need to be translated (from something like “array[int]” to “int[]”). Here is the code:

lettypeHandler.rkt:

```
 #lang racket
 (require "state-roster.rkt")

; Handles a 'let-statement branch of an AST
 (define let-tag-handler
   ;;; add section for mutable lets
```

```
  (lambda (datum)
    (let ([mutable? (mutable-tag? datum)]
          [ast-result '()])
         (add-variables-defined (second (third datum))) ;;catalog it as variable
         (define is-array #f)
         (if mutable?                          ;; if immutable, catalog it
             (and (set! is-array (equal? (second (sixth datum)) "array"))
                  (set! ast-result (list  'declaration
                                         (list 'type
                                            (letType-handler-get-type
                                               (sixth datum))) ;;type
                                         (correct-id-if-array
                                            (third datum) is-array)  ;; id
                                         (seventh datum)  ;; equal symbol
                                         (eighth datum);;value
                                         (ninth datum))))
             (and (set! is-array (equal? (second (fifth datum)) "array"))
                  (set! ast-result (list  'declaration
                                         (list 'type (string-append
                                                      "const PROGMEM "
                                                      (letType-handler-get-type
                                                         (fifth datum)))) ;;type
                                          (correct-id-if-array
                                             (third datum) is-array)  ;; id
                                          (sixth datum) ;; equal symbol
                                          (seventh datum) ;;value
                                          (eighth datum))) ;; delimiter
                  (if in-scope-statement '() (add-prog-mem-variable
                                                  (third datum)))))
      ast-result)))

; Appends "[]" to the id tag for C's notation
(define correct-id-if-array
  (lambda (id-string is-array)
    (if is-array
        (list 'id (string-append (second id-string) "[]"))
        id-string)))


; Translates types from original-AST to target AST.
(define letType-handler-get-type
  ;;; TODO: add multi-dimensional array type handling.
  (lambda (datum)
    (cond [(equal? (second datum) "array") (fourth datum)]
          [else (second datum)])))
```

```
; Verifies that a mutable tag is present in a let-statement
(define mutable-tag?
  (lambda (let-statement)
    (define result (filter (lambda (x) (equal? (first x) 'mutable-tag))
                           (rest let-statement)))
    (if (equal? (length result) 1) #t #f)))
```

So the code looks a little verbose, but it is rather straight forward once broken down:

- *mutable-tag?* is just a predicate to see if some let-statement is marked as mutable in the course code.
- *letType-handeler-get-type* simply extracts the type from an AST for easier processing.
- *correct-id-if-array* covers the correction of array brackets in type declarations ("array[int]" -> "int[]").

  The only complicated function is *let-tag-handler* which glues all this stuff together. First, it catalogs the variable for the unpacker to correctly handle it later in a list from **state-roster.rkt**. Then, if the mutable tag is present, one of two different formatting procedures is used. This has to happen because the present of the mutable tag shifts the index of each element for the AST, restricting options in making a general purpose formatter. While formatting, each element of the array is either directly returns or passes through some minor correction (such as **correct-id-if-array**). After all of this, the translated AST is passed to the **unpacker**.

Unpacking the Handler

What the handler leaves us with is a list of lists that represent the program's AST; this is far from executable. This is where the ***Unpacker*** comes in. The Unpacker spelunks through the AST and does one of three things: * Returns contained inner string, * Hands off special cases to a correcter function, * or calls the Unpacker on sub lists for further unpacking.

The case of something that can just be returns is an atomic object of the AST such as:

```
'(lit "1")
```

There is no more information within this item and nothing needs to be formatted differently. However, in the case of certain AST objects require special unpacking to deal with some of the specific formatting of the Arduino language. Function calls, for instance, require special handling of arguments to make sure they are properly wrapped in parenthesis and separated by commas. Another would be variables declared in **PROGMEM**; the Unpacker will wrap each reference of them (as a pointer) in a Arduino built-in function to read from program memory. The compiler knows how to do these corrections using a directory similar to that of the handler’s directory function.

After the unpacking phase, the resulting Arduino sketch will be returned as a full string that is written out to a file for uploading.

Targeting AVR Processors:

Making Sure Things Stayed Fast

One of the goals of MinNo was to more easily handle the boiler plate code I use when writing Arduino scripts. Given the Arduino’s slow speed and limited resources, I needed to make sure that the generated code was not so slow as to entirely undermine this reason for using MinNo. Therefore I made the conscious decision to make MinNo more of a subtle semantic departure from C rather than something entirely different. It allowed me to emphasize certain points about Arduino programming that I find useful (like making immutability the default to make easy use of the architecture difference). Making it a shift in default behavior, rather than an entire paradigm shift allows MinNo to be closely and concisely mapped to a C program with only minor runtime overhead.

Letting People Under The Hood: Why Translator Result Readability Matters

One day I want MinNo to be a mature enough platform to write the majority of my Arduino sketches in. But the likelihood of achieving that level of maturity while also wanting the graduate within 4 months of writing this seems minuscule (to put it lightly).

To make up for this, the MinNo compiler focuses on making (subjectively) readable C code. Often compilers will leverage language constructs (such as lambdas) to make code generation easier however this can generate code that is hard to read. MinNo’s template approach lets the generated code be written so it can follow consistent styling and formatting for readability.

The code being readable allows the current restrictions of MinNo to be bypassed by editing the resulting C script. If you really want some kind of manual memory allocation, MinNo generated C will be totally accepting of that even if MinNo itself isn’t.

Disadvantages of My Approach:

A Very Insular Crowd:

Though I can say much to praise Racket, there are definitely some detractors. For instance, in trying to get into their syntax object I was assaulted with numerous conversation-specific words, phrases, and frameworks. Though the documentation gave a great deal of links on where to read further, it was just layers upon layers of conversations that I had no sane introduction to.

Most of these topics were manageable, but not without staring at docs and opening a an interactive Racket interpreter again and again. Racket is a small community with hyper-specific language to talk about what they do. Consequently there aren't very many resources available from people other than the ones responsible for the documentation itself.

Other languages have lexer and parser tools like BRAG. Java has ANTLR to construct compilers, and C/C++ has Lex/Flex to construct lexers and Yacc/Bison to compiler parsers. Both of these could have done what Racket's **lex** and BRAG accomplish in a similar manner. The lexer specification would be some series of regular expressions that produce tokens that would be consumed by a parser that follows form BNF-like grammar structure. The major benefit that I see in these over Racket is that they are far more used and subsequently have a much larger collection of tutorials and educational material surrounding them.

Handler are Both Great and Terrible:

Handlers are a wildly simple and easy framework to use in AST processing. It allows for very iterative design. Unfortunately, due to the way Racket does lexical parsing, it is very hard to make handlers that are separated at the file level. There are plenty of pieces in MinNo's handler system that can totally be separated out into another file for better organization, but this is not true for all of it. If a particular handler needs to call the directory function on some sub-section of the AST is handling, it needs to be in the same file as the directory. If you tried to separate it, there would need to be a cyclical dependency of the two files. Racket does not (and absolutely should not) allow cyclical imports and therefore makes it impossible to separate trampolining functions such as certain handlers and the handler directory.

What is Left to Be Done:

Targeting Closer to the Chip

Currently MinNo targets the Arduino programming language. A next step would be to target GCC (or LLVM) compatible C and AVR assembly directly.

This would be more verbose code for sure, but the speed increases and system controllability would be greatly increased. A major benefit would also be the ability to expand support for more embedded platforms. Targeting Arduino restricts me to boards within Arduino or it's community's interest to support. Boards like the Beagle Bone and the Raspberry Pi are outside of this list. Targeting GCC or LLVM (when the support for AVR is mainlined) will allow faster resulting code and wider target platform support.

Creating an Ecosystem & Library

MinNo can do plenty of common tasks on the Arduino, but it's current library of keywords and functions is sparse. In the interest of creating a useful language, a standard library needs to be properly fleshed out. String operations, mass pin manipulation, bit-shifting abilities, data-serializing functions, and more. All of this needs to be included to make MinNo a robust platform for embedded development.

It also needs to take advantage of the thousands of programming hours that have already been poured into the platform. Interoperability with C code directly in MinNo files is on the short list of features to tackle next. There are far too many libraries for Arduino add-ons, such as shields, to reimplement from scratch.

In managing the library, C code, and all of the chips to be targeted, better tooling needs to be developed. The rise of languages like Rust and Scala can be attributed solely to their innovation as languages, but that would be missing a key point that other languages (like D and Julia) neglected: tooling. SBT (Scala's build tool) and Cargo (Rust's version of SBT) helped root programmers in a central conversations of the respective languages. They allowed for immediately productive and understandable project structures (or at least allowed the authors of the language to direct where the communities style went). These common architectures allowed code to easily connect and communicate. MinNo needs to make a similar move into creating an entry point to allow coders to share what they make.

Opinions On The Ecosystem: Racket, raco, and General House Keeping

Package Management & Tooling

With Racket's niche being a platform to build more languages, they needed to have some way to share the tools that programmers produce so no one has to spend time reinventing the wheel. Therefore, Dr.Racket has a very intuitive graphical package manager built on top of their command line tool **raco**.

pkgs.racket-lang.org lists, (*start sarcasm*) with all the aesthetic mindfulness of Dr.Racket (*end sarcasm*), the packages available though **raco**. It contains tags, links to package documentation, and checksums (if you are into that).

There is a common moment of anxiety in development when a language or library releases an upgrade while you are using the now-out-of-date version. Some projects are stuck in a version lock (or rewrite) due to their reliance on libraries built in a previous version. While writing the MinNo Compiler, Racket came out with not 1, but 2 language releases.[10][11]

These were minor version changes (6.7 & 6.8), so I thought about not upgrading, but after reading their change-logs, I knew I had to. 6.7 greatly increased performance of strings in the language.[12] 6.8 improves optimization of equality statements.[13] Those two areas make up the vast majority of operations and data handled by a compiler; it would be a crime to not utilize these upgrades.

Of course I had that moment of panic when I downloaded the newest version of Racket. The raw terror of some feature of the Lex library not working and requiring me to 1. revert to the previous version or 2. rewrite some not trivial part of the program. After installation was finish, migrating my dependencies was next on the list.

In the graphical version of the Racket Package Manager, there is a very handy migration tool from previous language installations. It just checks your previous installation of racket for downloaded libraries and will automatically grab them from raco for the new installation. Once BRAG was migrated and updated, I decided to test how much a had broken my code base.

```
module: identifier already imported from a different source
whitespace
parser-tools/lex
brag/support
```

It seemed that one of the two libraries decided to now include the term “whitespace” which was already part of the other library, creating a naming conflict. After some very brief searching, some *import prefixing* did the trick.[14] After that, the upgrade was successful & the compiler felt much snappier then before (though I have no way of saying this objectively, since I never ran a numeric test).

One very surprising thing was that when I upgraded Racket, all of my preferences and settings carried over despite doing a “fresh” install. Binds, color schemes, and everything else. As someone who is color blind, not being bogged down for hours figuring out a color scheme that actually helps me was a really nice surprise.

Though I don’t have a great deal of praise of Dr.Racket (I don’t have deep criticism either, just a mild understanding that it is around and has a particular use), I have grown very accustomed to using it for Racket development.

All in all, it is not as comprehensive of a tooling infrastructure as **Cargo** or dynamically connected as something like **npm/bower** but there is something to be said for it’s simplicity.

Feelings about Where the Language is Going

Racket is unfortunately held back by it’s view as a tool for education and the lack of functional utility that it’s only development environment, Dr.Racket, exudes. When talking to a fellow college student during an internship, I mentioned that I used Racket. He looked at me with some confusion. “Racket, we used that in high school for the intro programming course. It was terrible.” Racket was initially created as an education tool.

There is a huge amount of resources poured into it to do as much. With the state of the default development environment, you are constantly affronted with what looks like software meant to educate you, not to do serious development. Being more open and accommodating with other setups for coding in Racket is absolutely required to attract & keep new developers.

It is a surprise that Racket hasn't broken out of their educational roots in regards to Dr.Racket like the rest of the language has. Though education is still a huge part of Racket, the language platform discussion has really taken off. By far one of the most interesting footholds I have witnessed Racket take is in game scripting as the main language for one of the most celebrated Sony developers.[15] Something about their process just seems to root around using LISP (considering they went through the trouble of maintaining their own sub language for a number of years).[16] For the last new (award winning) titles they have released, Racket has been their choice in scripting languages.

The movements into fields are far apart as language development and gaming scripting have done a great deal to lift Racket out of it's PLT reputation. If it could shed away from the last major reminder of those days, it's deep attachment to Dr.Racket, I think it could really be up there with Clojure as the new face of LISP.

Bibliography:

Directly cited:

- [1]: “Racket: From PLT Scheme to Racket.” *racket-lang.org*. Published: June, 2010. https://racket-lang.org/new-name.html.
- [2]: Graham, Paul. “LISP FAQ.” *paulgraham.com*. Accessed: February, 2017. http://www.paulgraham.com/lispfaq1.html.
- [3]: “Rationale.” *clojure.org*. Accessed: March, 2017. https://clojure.org/about/rationale.
- [4]: Might, Matthew. “Compilers :: CS 5470.” *matt.might.net*. Accessed: September, 2016. http://matt.might.net/teaching/compilers/spring-2015/.
- [5]: Might, Matthew. “Lexical analysis in Racket.” *matt.might.net*. Accessed: September, 2016. http://matt.might.net/articles/lexers-in-racket/.
- [6]: logicchains. “warning: function fooBar should have a snake case name such as foo*bar. # [warn(non*snake*case*functions)] on by default.” *reddit.com/r/rust*. Accessed February, 2017. https://www.reddit.com/r/rust/comments/29sumo/warning*function*foobarshould*have*a*snake*case/.
- [7]: Might, Matt. “derp2.” *github.com/mattmight*. Published: Mar 7, 2015. https://github.com/mattmight/derp2.
- [8]: Hall, David Leo Wright. “Refinements in Syntactic Parsing.” *digitalassets.lib.berkeley.edu*. Accessed: February, 2017 http://digitalassets.lib.berkeley.edu/etd/ucb/text/Hall*berkeley*0028E_15470.pdf.
- [9]: Johnson-Davies, David. “Tiny Lisp Computer.” *technoblogy.com*. Published: 8th September, 2016. http://www.technoblogy.com/show?1GX1.
- [10]: “Racket v6.7.” *racket-lang.org*. Published: October 26, 2016. http://blog.racket-lang.org/2016/10/racket-v67.html.
- [11]: “Racket v6.8.” *racket-lang.org*. Published: January 24, 2017. http://blog.racket-lang.org/2017/01/racket-v6-8.html.
- [12]: “Racket v6.7.” *racket-lang.org*. Published: October 26, 2016. http://blog.racket-lang.org/2016/10/racket-v67.html.
- [13]: “Racket v6.8.” *racket-lang.org*. Published: January 24, 2017. http://blog.racket-

lang.org/2017/01/racket-v6-8.html.

- [14]: Hendershott, Greg.“Overlapping module imports in Racket.” *stackoverflow.com*. Published: July 27, 2013. http://stackoverflow.com/questions/17894875/overlapping-module-imports-in-racket.

- [15]: Racket Lang. “RacketCon 2013: Dan Liebgold - Racket on the Playstation 3? It’s Not What you Think!” Filmed September 29, 2013. YouTube Video, Duration: 59:23. Posted: October 13, 2013. https://www.youtube.com/watch?v=oSmqbnhHp1c.

- [16]: Gavin, Andy. “Making Crash Bandicoot – GOOL – part 9.” *all-things-andy-gavin.com* Published: March 12, 2011. http://all-things-andy-gavin.com/2011/03/12/making-crash-bandicoot-gool-part-9/.

Background resources:

These were sources that helped inform me in how to build and program a compiler but had no direct quote or idea echoed in this paper.

- Politz, Joe. “Principles of Compiler Design.” *cs.swarthmore.edu*. Published: April 29, 2016. https://www.cs.swarthmore.edu/~jpolitz/cs75/s16/s_schedule.html.

- Norvig, Peter. “(How to Write a (Lisp) Interpreter (in Python)).” *norvig.com*. Published: August, 2016. http://norvig.com/lispy.html.

- Schrittwieser, Julian.“Scalisp - a Lisp interpreter in Scala.” *furidamu.org*. Published: March 23, 2012. http://www.furidamu.org/blog/2012/03/23/scalisp-a-lisp-interpreter-in-scala/

- Sarkar, Dipanwita, and Oscar W., and R. Kent D.. “A Nanopass Framework for Compiler Education∗.” *cs.indiana.edu*. Accessed: August, 2016. http://www.cs.indiana.edu/~dyb/pubs/nano-jfp.pdf.

- NewCircle Training. “The Value of Values with Rich Hickey.” Filmed: July, 2012. YouTube Video, Duration: 31:42. Posted: October 8, 2012. https://www.youtube.com/watch?v=-6BsiVyC1kM.

- GDC. “Programming Context-Aware Dialogue in The Last of Us.” Filmed: March, 2014. YouTube Video, Duration: 1:00:54. Posted: February 18, 2016. https://www.youtube.com/watch?v=Y7-OoXqNYgY.

MinNo Documentation:

1.0 Getting Started

Install The Compiler and Its Dependencies:

First off, start by grabbing the compilers source code and a distribution of Racket:

- Racket: https://racket-lang.org/download/
- The Compiler: https://github.com/OkayAlright/MinNo

After the compiler is downloaded, you need to install grab MinNo's single dependency: *BRAG*. BRAG can either be downloaded via DrRacket's graphical package manager or by using the Racket command line packaging tool *raco*:

```
$ raco pkg install brag
```

After that the compiler can be run by the Racket interpreter:

```
$ racket <path to compiler source>/minno/src/compiler.rkt <minno file> <output path>
```

Or you can compile it into a standalone executable using DrRacket or *raco*:

```
$ raco exe -o <some output name> <path to compiler source>//minno/src/compiler.rkt
```

This will create a system-specific executable that can be run with the typical *./* execution. The resulting executable can be run just as the non compiled file without the "racket" prefix:

```
$ <path to compiler source>/minno/src/compiler <some minno file> <output path>
```

After running the compiler on a source file, you can either upload it to an Arduino board using their IDE.

The Anatomy of A MinNo Script

Every MinNo file can be broken up into four sections: * Global Values, * a Setup function, * a Loop function, * and user defined functions.

This is very similar to the anatomy of a normal Arduino sketch. Global values are variables that you want to reference through out the sketch, the setup function runs once at boot, after the setup function the loop function will run until the board is reset or powered off. Any functions written other than those two will only be called if invoked by some other code.

A Quick Start:

“Blink” is the Arduino’s “hello world” program. It is a simple sketch that turns a pin on, waits, and than turns it off repeatedly. Here is a blink sketch in MinNo:

```
#/
blink.minno

The classic 'blink' sketch in MinNo.
/#

let outputPin : int = 13; //set pin for reference

def setup none -> none {
    //set mode to output
    pinMode outputPin OUTPUT;
}

def loop none -> none {
    // blink on and off with 1 sec intervals
    digitalWrite outputPin HIGH;
    delay 1000;
    digitalWrite outputPin LOW;
    delay 1000;
}
```

The first block of text is a multi-line. MinNo uses “#/” and “/#” to delimit these comments. A single line comment is started with a “//” similar to C. The next line:

```
let outputPin : int = 13;
```

This is an example of one form of variable declarations in MinNo. “let” is like Javascript’s “var” stating that you are about to reference a variable that will need to be allocated. “outputPin” is used as the values ID and it’s type is stated right after the colon separating it from the ID. This “colon type” marker is non optional. The rest of the line is fairly straight forward, noting that the literal value associated with this ID is the number 13. A semi-colon acts as a statement delimiter, similar to C.

After declaring the value, there are two function definitions. These are the required Setup and Loop functions in every MinNo script. The keyword “def” denotes that the following ID will be used for a function definition. After the ID is the type signature (arguments and return type). The Setup and Loop functions both take no arguments and return nothing, so “none -> none” says “I take nothing and I given nothing.” As mentioned earlier, the Setup function runs once and the Loop functions runs as long as the board is powered after that. In Setup the only line in it’s definition (delimited by “{” and “}”) calls the function “pinMode” to set up a pin for output. It is passed “outputPin” (the value we defined) and “OUTPUT” (a builtin value common in Arduino programming). MinNo function calls consume all whitespace-separated IDs and literals after the function name until a colon is found. No parenthesis needed. Section 2.3.0 goes over how to nest function calls within the same line (as a preview, you wrap inner function calls in parenthesis). The Loop function calls two functions twice: “digitalWrite” and “delay”. “digitalWrite” sets an output value to a pin (“outputPin” in this case). The value is set by the second arguement. This is case it flips between “HIGH” and “LOW”, other builtin values comparable to “on” and “off”. In between each call to “digitalWrite” there is a delay followed by “1000”. This will cause the Arduino board to stop everything for 1000 milliseconds (1 seccond) and then continue. So this script turns pin 13 on, waits a second, turns it off, and than waits a second before turning it back on again (and again and again).

Given all of that code, we end up with a basic MinNo script.

2.0 The Language

This is a more formal and thorough walk through MinNo’s builtin keywords and structures.

Brief Overview and Comparison to Arduino-C:

For a brief overview of MinNo and a comparison to Arduino-C please feel free to consult this poster:

- https://github.com/OkayAlright/MinNo/blob/master/extras/posterForConference.pdf

2.0.1 Types and Literals

MinNo’s types closely match C’s. Here are the builtin types in MinNo: * int : 1, 2, 3, -56 and so on. This is for negative or positive whole numbers.

```
let exampleInt : int  = 5 ;

let exampleInt2 : int  = -535 * 10 ;
```

- float : and decimal number, positive or negative.

  let exampleFloat: float = 3.14 ;

- char : this any single character wrapped in double quotes.

  let exampleChar : char = “a” ;

- array : a collection of any of the other types. This subtype is denoted by it’s keyword after the array keyword wrapped in square brackets. This is also how strings are made (as arrays of chars).

  let exampleIntArray : arrayt[int] = [1,2,3,4,5];

  let exampleString : array[char] = “This is how a string is declared.\n”;

  Arrays are index using zero-indexing by referencing the arrays ID with a integer corresponding the the index you want wrapped in square brackets after it.

2.0.2 Operators

The arithmetic operators in MinNo are the standard “+”, “-”, “*”, and “/”. They are addition, subtraction, multiplication, and division respectively. There are infix boolean operators as well:

- == : check to see if the numbers on both side are equal, in which case true, otherwise false.
- > : if the number of the right is greater than the left, true, otherwise false.
- < : like “>” but reversed.
- >= : like “>” but will also return true if both numbers are equal.
- <= : like “<” but will also return true if both sides are equal.
- && : a boolean AND. If both sides are true, than it is true, otherwise it is false.
- || : the boolean OR. If either side is true, than the whole thing is true, otherwise it is all false.

The order of these operations are all the same as they are in C. When in doubt, wrap the first thing that you want to resolve in the inner most parenthesis and work out from there.

2.1.0 Declaring Values

One of MinNo’s features is in it’s variable initialization and assignment. Given that MinNo targets Harvard

Architecture processors (which can handle variables in memory very differently from x86 and ARM processors), variable declaration is actually one of the more nuanced sections of the language. The main difference between declarations in MinNo will be broken down into two sections. Before getting into that here is some general information that is true for both major classes of data.

You have to give a value upon initializing a variable in MinNo. No null types here. This means that some sections of code might result in long lines, but it avoids null pointer errors. You must also give each variable a type. Speaking of which, here are the builtin types in MinNo: * int : 1, 2, 3, -56 and so on. This is for negative or positive whole numbers.

```
let exampleInt : int  = 5 ;

let exampleInt2 : int  = -535 * 10 ;
```

- float : and decimal number, positive or negative.

  let exampleFloat: float = 3.14 ;

- char : this any single character wrapped in double quotes.

  let exampleChar : char = “a” ;

- array : a collection of any of the other types. This subtype is denoted by it’s keyword after the array keyword wrapped in square brackets. This is also how strings are made (as arrays of chars).

  let exampleIntArray : array[int] = [1,2,3,4,5];

  let exampleString : array[char] = “This is how a string is declared.\n”;

  Arrays are index using zero-indexing by referencing the arrays ID with a integer corresponding the the index you want wrapped in square brackets after it.

The name of a variable can be any collection letters, underscores, and numbers (as long as they don’t start the ID name). Any re occurring name in the script must be the same type and mutability class as the other occurrences regardless of scope. This helps to stop “x” being reused as a throw-away name all over a single sketch.

2.1.1 Immutable Variables

By default all variables are immutable it MinNo. Specifically they act like a variable in C declares with the keyword “const”. You cannot reassign a new value after the variable is declared.

Within immutable declarations, there are two classes of data: stack and program data. Stack immutables are immutable values defined within function definitions. These exact exactly like C's "const" values. The only restriction they entail is that they cannot be reassigned.

The second group of immutable data structures are call "program data". This is to leverage the Harvard memory model. Any immutable value at the global level is stored in PROGMEM instead of flash storage and ram. variables in PROGMEM are quickly accessible and entirely separated from the other storage areas. The details of declaring and accessing these values are handled by MinNo, so you don't need to worry about it if you don't want to.

2.1.2 Mutable Variables

Now sometimes you want mutable variables for iterators or summing variables. MinNo allows the declaration of mutable variables with the "mutable" keyword placed before it's type. For example:

```
let exampleMutableInt: mutable int = 6;
```

This creates a mutable integer that allows you to reassign it's value like so:

```
exampleMutableInt = 10;
```

2.2.0 Defining Functions

All function definitions take a similar form in MinNo:

```
def <some name> <ID-type tuple for arguments or "none"> -> <type or "none"> {
    <Your code>
}
```

An example of a simple function definition would be the Setup function from the quick start section:

```
def setup none -> none {
    //set mode to output
    pinMode outputPin OUTPUT;
}
```

This follows the form specified above. A more complicated example could be this contrived function to multiply two numbers and add 2 to the result

```
def multAndAddTwo x:int, y:int -> int {
    return (x * y) + 2 ;
}
```

Now we have some arguments to parse. After the function ID, there are two arguments separated by a comma (as each argument needs to be). One is labeled “x” and the other as “y”. Both are integers.

2.3.0 Using Functions and Values Together

To avoid ambiguity in parsing, when you need to nest functions or operations within each other in MinNo, each sub expression need to be wrapper in parenthesis. Here is an example of a call to the previously defined “multAndAddTwo” being called and passed to another invocation of itself:

```
multAndAddTwo (multAndAddTwo 6 7) 3;
```

This calls the inner most “multAndAddTwo” passing “6” and “7”, takes the result and passes it along with “3” to another “multAndAddTwo”. The same is true for operations:

```
multAndAddTwo (5 * 9) 2;
```

2.4.0 Iteration

Though the leaning towards immutability might suggest a leaning towards recursive function as well (thusly leaning towards the functional programming camp), Arduino’s memory is far too limited to deal with recursion nicely. So MinNo has both “for” and “while” loops. It also supports recursive definitions if you feel like shooting yourself in the foot.

For most iteration, a “for” loop will do. A for loop in MinNo works like any other for loop in C or Javascript. It is made of a few basic parts:

```
 for <declaration or assignment of variable> ; conditional ; incrementor ;{
     <code>
 }
```

The declaration or assignment of a variable allows the creation or prep of a variable to track the for loops iteration. The conditional, when false, will stop the for loop to continue through your sketch, and the incrementor happens at the end of every pass through the for loop. Here is an example with some filler values that sum subsequent integers:

```
let sum: mutable int = 0;
for let i: mutable int = 1; i <= 10 ; i = i + 1;{
    sum = sum + i;
}
```

For more general iteration, “while” loops should suffice. The form for “while” loops looks like this:

```
while <conditional> {
    <your code>
}
```

The while loop will keep executing your code until the conditional statement is proven false. After that it will continue through your script.

2.5.0 Conditional Branching

Conditional branching in MinNo is handled by the “if” and “else” keywords. The “else” branch is entirely optional. After the if, some boolean expression must follow. A conditional branch has this form:

```
if <boolean expression> {
    <your code if true>
} else {
    <your code if false>
}
```

Your boolean expression can be nested boolean expressions within each other to get fairly complicated behavior. Currently there is no ability to chain “else-if” branches. This should be fixed soon.

3.0 The Builtins

3.1.0 Built In Constants

There are a few pre defined constants similar to Arduino’s.

HIGH and LOW

> *HIGH* and *LOW* are constants to act as *on* and *off* value respectively. They are most commonly used as values for *digitalWrite* or used to compare agains the output of *digitalRead*.

INPUT and OUTPUT

> *INPUT* and *OUTPUT* are used exclusively as values to pass to *pinMode* what setting up digital pins.

3.2.0 Built In Functions

MinNo's built-in functions act, effectively, as a super set to the Arduino built-ins. Any function you find in Arduino's own documentation will likely be present here. Though in practice only a small subset of those are used. This documentation will go over that subset.

Digital I/O

pinMode

> **pinMode** sets a digital pin (pin's 2 through 13) to be in either **INPUT** or **OUTPUT** mode. This should be called in a Setup function at the beginning of a script.
>
> arguments: * *pin*: An integer representing the pin you are setting. * *mode*: Either **OUTPUT** or **INPUT** depending on whether you want to read or write from a pin.

Example:

```
def setup none -> none {
    pinMode 13 OUTPUT; //sets pin 13 to write mode.
    pinMode 9 INPUT; //sets pin 9 to read values values
}
```

digitalWrite

> **digitalWrite** outputs a given value (typically **HIGH** or **LOW**) to a digital pin.
>
> arguments: * *pin*: An integer representing the pin you are setting. * *value*: Typically **HIGH** or **LOW** (on or off respectively). Example:

```
def loop none -> none {
    digitalWrite 13 HIGH; //turns pin 13 On
    digitalWrite 13 LOW; //turns pin 13 Off
}
```

digitalRead

> **digitalRead** reads and returns either **LOW** or **HIGH** from a specified digital pin.
>
> arguments: * *pin*: An integer representing the pin you want to read from

Example:

```
let exampleRead : mutable int = digitalRead 13;
//assigns the value read by 13 to exampleRead
```

Analog I/O

analogWrite

> **analogWrite** outputs a given value to any pin that is marked as analog or for Pulse Width Modulation (PWM). This value can be within a range of 0 to 255.
>
> arguments: * *pin*: An integer representing the pin you are setting. * *value*: Can be any 8-bit value for analog pins or any 8 bit value for digital PWM. Please reference your board to see which is which.

Example:

```
def loop none -> none {
    analogWrite 1 255;
    //turns analog pin 1 to full analog output, typically this is 5 volts.
}
```

analogRead

> **analogRead** reads and returns some value between 0 and 1024 from a specified analog pin.

> arguments: * *pin*: An integer representing the pin you want to read from

Example:

```
let exampleRead : mutable int = analogRead 4;
//assigns the value read by 4 to exampleRead
```

Time Related Functions

milliseconds

> Returns the amount of milliseconds since the arduino was turned on.
>
> Example:

```
//An example to measure time passed during code execution
let start: int = milliseconds ;
// <your code>
let end: int = milliseconds ;
let totalTime: int = end - start;
```

micros

> Like **milliseconds** but it returns microseconds instead.

delay

> **delay** halts all operations on the board for a specified number of milliseconds. After that time has passed, the board resumes execution.
>
> arguments: * *time*: the number of milliseconds you want to pause for.

delayMicroseconds

> Like **delay**, but using microseconds instead of milliseconds.

Math Utilities

min

> Returns the small of two passed numbers. This can be used to limit the largest number assigned to a variable.
>
> arguments: * *x*: One of the two values to possibly be returned. * *y*: The other value to compare it to.

max

> Like *min* but returns the larger of the two.
>
> arguments: * *x*: One of the two values to possibly be returned. * *y*: The other value to compare it to.

abs

> Calulates and return the absolute value of a passed integer.
>
> arguments: * *x*: some integer to calculate the aboslute value of.

map

> Takes a given number, it's possible range, and a desired range and constrains that given value from its original range into the desired one. This is useful for restricting the input from **analogRead** to some analog output value (a 10-bit to 8-bit map).
>
> arguments: * *valueToMap*: The value you want to constrain. * *originalMin*: the lower bounds of the original input. * *originalMax*: the upper bounds of the original input. * *newMin*: the lower bound of the desired input. * *newMax*: the upper bound of the desired input.

Example:

```
//This function will read an analog pin and
//automatically map it to a PWM writable range.
def readAndMap pin:int -> int {
    let pinread: int = analogRead pin ;
    return map pinread 0 1024 0 255 ;
}
```

---

pow

> Computes and returns a passed number raise to another passed number.
>
> arguments: * *value*: the number to raise. * *exponent*: the number to raise *value* by.

---

sqrt

> Computes and returns the square root of a passed number.
>
> arguments: * *x*: the integer to computer a square root for.

4.0 Contact Info and Licensing:

Feel free to get in touch with me either by email or twitter:

- email: ldavis@marlboro.edu
- twitter: @Death_by_kelp

  The compiler, examples, associated papers included in the GitHub repo, and this documentation are all freely available under the MIT License.

  This document was up-to-date as of 4/17/17

Exams:

The memoir and the compiler were labours to try to convey my experience. **Release** was a retelling of my education, socialization, and life. The compiler is an attempt at a tool that I wish I had when I learned to code. The exams will be the evidence that I have overcome the disabilities that inspired the previous components. The exams are here to demonstrate fluency in what gave me so much trouble my first two years at Marlboro: actually programming.

Exam 1 - Programming Languages:

Question 1 (40%)

As a way to demonstrate your understanding of programming ideas, discuss the concepts behind following programming buzz words across at least three languages that you're familiar with that allow different programming styles, perhaps Python, Racket, and C.

(a) First organize the terms into groups of concepts, showing which are variations of the same concept or idea across or within languages, or closely related concepts, or opposites.

(b) Then for each of these concept groups, discuss the ideas behind that group, and give concrete code snips across these languages to illustrate them.

Be clear that I *don't* want you to just define each of these words; instead, I want you to use them as the starting point for a conversation with examples about some of the fundamential [sic] notions of how programming languages work, and how those notions vary from language to language.

In alphabetic order, the words are

```
API
argument
array
bind
callback
class
closure
collection
comment
concurrent
compiled
data structure
dynamic
exception
fork
function
functional
global
hash
immutable
imperitive [sic]
inheritance
```

```
    interface
    interpreted
    iterate
    lambda
    lexical
    link
    list
    lazy
    lexical
    macro
    method
    name
    namespace
    object
    overload
    parse
    scope
    package
    pattern
    pass by reference
    pass by value
    pointer
    recursion
    side effect
    stack overflow
    static
    symbol
    syntactic sugar
    thread
    test
    throw
    type
    variable
    vector
```

Answer:

0 data structure:

Data structures are some of the fundamental architecture choices in a programming language that will define what the atomic pieces of a given program can do. How do I store multiple items? How do I reference the same piece of data repeatedly? How do I make sure a piece of data cannot be changed once I make it? These are questions on the level of data structures.

0.0 variable:

One of the most basic questions a programming language has to answer is how it will store data. Though there are plenty of instances where data is hard-coded in a program, more often these values need to be repeatedly referenced or changed during the run time of the program. A lot of pattern-centric thought can be applied to variables, their construction, and their usage, but I included it in data structures because the choices made about how to implement variables often informs what patterns are the best approach for working with it.

Though patterns can differ highly between languages, the data structures for variables rarely differ in a deep way in between languages. For example, here are some examples of variable declarations in C, Python, and Racket:

In C:

```c
int c_int_example = 1000;
double c_real_example = 7.0;
char c_string[] = "this is a string literal.";
```

In Python:

```python
integer_python = 1000
double_python = 7.0
string_python = "this is a string literal."
```

In Racket:

```
(define integer-racket 95439)
(define double-racket 5.67)
(define string-racket "this is a string literal.")
```

Though the syntax for each changes a little, the semantics are nearly identical. There are some different choices in typing (which I will talk about shortly), but in each example we are assigning some *value* to some *name*.

0.0.0 names:

A name is the symbolic representation of a repeatably referenceable value (the “c*int*example” at the very beginning of the code examples in the previous section). What a valid name is depends of the language. Most languages allow and character literal (“a” though “z”, lower or capital case), underscores, and some number (as long as it isn’t the first character of the name).

In languages like C, Python, and Java, the following are all valid names:

```
someVariable
this_is_okay_too
STILLFINE
th15_15_4l50_F1n3_but_y0u_5h0u1d_f33l_bad
```

Where something like this is not okay:

```
1_wrong
!nope, not going to work!
still-wrong
```

Some languages like Rust, Racket, and Pyret allow for a few more symbols (such as hyphens and “!” to be in names). Lisps often allows even more that other languages, such as the following:

```
this->is->fine!
still-okay?
YUP!
```

0.0.0.0 symbol:

The term “symbol” is a little bit overloaded. A symbol can refer to a token in lexers, a data type in some languages (such as Erlang and Elixir’s “atom”), or a very generic name for the reference of names of variables, functions, and classes. I will be discussing the latter of these options.

The usefulness and access of symbols is different from language to language. Python rarely uses any kind of symbol reference with the exception of function-bound symbols (function pointers). Other languages like Lisp make extensive use of their reference. In Lisp, to reference a symbol, you just “quote” the name. Racket, being a Lisp dialect, uses the following notation to do so:

```
> (quote a)
`a
> `a ;;shorthand notation for quotes
`a
> a ;;an example of not quoting a symbol, thus attempting to resolve it as a bound name
a: undefined;
cannot reference an identifier before its definition
> ;; this attempt to resolve "a" fails because it has no definition
```

The way to reference a function as a symbol is to not put parenthesis after the name:

```python
>>> def foo():
...     return 5;
...
>>> foo #note the lack of "()" after "foo"
<function foo at 0x101b789d8>
>>> #The returned address can be used as a symbol name for the function.
```

Languages that allow for a ML-style function invocation (such as Ocaml and Ruby) cannot utilize the paren-less reference as notation for a symbol. Sometimes this requires special notation such as Ruby:

```ruby
> def foo()
>     "this function does almost nothing."
> end
> foo
"this function does almost nothing."
> :foo
:foo
```

0.0.0.1 bind:

To *bind* is to connect a some form of data (a function of a variable) to some reference (a name or symbol) in a program. This is most often used to connect a value or function to a name. In many languages binding a variable vs. binding a function require entirely different forms (syntax) to do so. C and Python are excellent examples:

In C:

```C
// binding a name to a function
int squareNum(int x){
    return x*x;
}
//binding a name to a variable
int example_var = 10;
```

In Python:

```Python
# binding a function
def squareNum(x):
    return x**2

# binding a value to a name
example_var = 10
```

Other languages allow the forms for binding to be the same by use of lambdas (a topic I will explain) or stack operations. Racket and Javascript are examples of leveraging lambdas to bind in a form similar to variables:

In Racket:

```
(define square-it
    (lambda (x)
        (* x x)))

(define example-var 10)
```

In Javascript:

```JavaScript
//note that this style is not idiomatic, it is just for example

var square_it = function(x){ return x * x; }
var example_var = 10;
```

Languages such as Forth get away with similar binding by treating both functions and variables as stack operations:

```
: squareIt dup * ; ( a function )
: someVar 5 ; ( a variable )
```

0.0.1 types:

I have already brought up types in the C examples. Types categorize data into sets that inform the program what each piece of data can and cannot do. For instance, a variable of type “integer” likely cannot be added to something of type “function.” However something of type float and type double should be able to subtract from each other (both being a numeric type). Types can be integers, reals, doubles, strings, booleans, functions, and so on. What each language uses and how it treats them is very language dependant.

0.0.1.0 static:

Static typing requires any variable, once declared (not assigned) to have a value that can never change. Usually this also means that the programmer must manually specify what type a variable is (or a function returns) manually. This is the case in C as shown in the example for *bind*.

Some languages that use static typing allow the programmer to omit the types and the compiler will attempt to infer what they are, such as Scala or Rust:

In Scala:

```scala
Scala
//note that Scala only infers variable & return types, argument types must still be exp
def minutesToHours(minutes: Int) {
   val mod_val = 60 //note the lack of type annotation
   return minutes % mod_val
}

val minutes = 120
val hours = minutesToHours(minutes)
```

0.0.1.1 dynamic:

Many new languages don’t require you to type variables or functions at all. The examples of Python, Ruby, and Racket are all examples of this (the Python and Racket have typed variants). The compiler will attempt to deduce types by the the operations being applied to variables and it will allow single names to change types as needed. For example, the following is perfectly legal in Python:

```
Python
>>> someVal = 5
>>> type(someVal)
<class 'int'>
>>> someVal = "a different type"
>>> type(someVal)
<class 'str'>
```

0.0.2 immutable:

Most variables, be default, are mutable (they can be assigned new values). If you want something to remain the initial that you assign it, the variable must be immutable. Some languages require you to specify immutability, others treat it as a default, some don't have the option of immutability at all (such as Python, though this is at the programmer level. At the implementation level, strings are immutable so they just create a new string and throw out the old one when a string variable is reassigned).

An example of explicit immutability:

In C:

```
C
int a_mutable_variable = 5;
const an_immutable_var = 10; //note the keyword "const"

a_mutable_variable = 10; //okay, fine.
an_immutable_var = 7; this will cause a compiler time error!!!
```

An example of a language that assumes immutability:

In Rust:

```
Rust
let some_immutable_var = 7; //this is immutable by default
// If I want a mutable variable that I can mutate,
// Rust has a special word at declaration:
let mut some_mutable_var = 10;
some_mutable_var = 6; // totally legal
```

0.0.2 closure:

Across many different paradigms and patterns, the ability to partially define a function (I will cover function later) to rapidly create similar operations can prove very useful. This is the use case of a closure. A closure allows some data to be defined in a function, but not to have the function actually run (returning a partially applied function, usually).[1] Here is an example in Python that can create list filters from lambdas using closures:

```python
>>> def make_filter(x):
>>>     def fil(y):
>>>         return( list(filter( x, y )))
>>>     return fil
>>> zero_filter = make_filter((lambda x: x !=  0 ))
>>> zero_filter([0,0,3,4,0,5,6])
[3, 4, 5, 6]
```

0.1 collection:

Collections are a data structure that allows a programmer to encapsulate many values in a single place. This data could be "*n* odd numbers" or "people in a building". The implementation of these collections varies wildly depending of the language (often a single language will have more than one implementation).

0.1.0: list:

A list is a generic name similar to collections. Some languages use this phrase interchangeably with arrays or linked lists. Regardless of the implementation, a list is always a collection of values.

0.1.1 array:

Arrays are sequentially indexed collections. Each item is stored in a contiguous block of memory (typically) equal amounts of address space apart. This allows indexing through an array to be a constant time calculation because they are an equal, known distance from each other. The cost of an array is the reallocation of space. If you need to increase the size of an array by adding an item on the front, that requires the entire array to be reindexed. Arrays are the underlying implementation of C's array type and Python's list.

0.1.1 link:

The term link is overloaded, but given the presence of array and list, I will assume it is in reference to linked lists. Linked lists are like arrays though each item is linked to each other by reference of their individual addresses rather than some large indexed block. This allows for appending data to the front (or back if it is recorded) of a list to be done in constant time (and appends to the middle that have linear cost equal to the index appended rather than the entire list). The cost of linked lists are actually walking the list requires traversal of the links references because there is no way to calculate where the *nth* item is without knowing what's directly in front of it and behind it (if it is doubly linked).

0.1.3 hash:

Like “link,” hash is a common concept in programming. For instance a hash function takes some single value and should spit out a (fairly) unique value. This is the tech that backs MD5 and SHA. Though given the covering of linked lists and arrays, I will talk about hashes in terms of hash-arrays (also called dictionaries, record, and associative lists).

A hash-array is a special kind of collection that allows for uniquely named, (typically) constant access name-value pairs. This kind of data structure could be good for bank account totals (“checking” => $505, \text{“}savings\text{”} =>$800) and so on. Some languages like Lua and Javascript implement these kinds of collections as a special version of their array type. A hash-array is often implemented as an oversized array (to provide ample room for the next step). A hash function and hash collision function (when the hash function produces two similar hash values) are created to assist the array. The name of the name-value pair is hashed to provide a repeatable index in constant time calculations given the same name from the pair. The name is than stored at the hash in the array. The value is placed at the matching index in a second array of the same length. If two names collide in the hash function (they produce the same hash), than the hash collision function should be able to repeatably find the same alternative spot for the later name-value pair.

The reference in a hash-array, as said before, is constant. The cost of this data structure is in the amount of collision produced by the hash function (and the subsequent performance of the hash collision function) as well as when a hash-array gets too full and needs to allocate more space (which is the same problem arrays have).

0.1.4 vector:

Vectors are just about one of the most flexibly defined collections that I have found. So far, the most consistently true use of them that I have seen is simply using the term “vector” for some collection that is implemented in the exact opposite fashion as most common collection type of the language.[2][3][4] Some implementations are just special arrays of floats with some math-y attributes like calculating an angle in a multi-dimensional vector, like C#’s *vec2* and *vec3* types in the Unity 3D environment.

0.1.5 lazy:

Lazy evaluation, as opposed to *eager evaluation*, is when a value is not actually computed until it is called on for another computation. This is often implemented for handling collections to cut down on the performance hit of allocating a large list (hence being mentioned in this section). It can also be used to pull some clever tricks to assign completely impossible collections if one were to use eager evaluation:

```
> (define all-natural-number (in-naturals))
> (+ (stream-ref all-natural-number 10) 5)
15
```

Yes, all-natural-number was a collection of all the natural numbers. More accurately is was a stream that knows how to compute any natural number given an index. There is no way for Racket to compute that entire list up front so it just knows how to compute values that are asked for. That ability to not need to compute everything unless it is used is the benefit of Laziness.

0.2 function:

Though writing every step in sequential order is fine for a small system script here or there, once your code gets longer than a few lines or you start reusing the same bit of code again and again, you will want to encapsulate some sections of your program. Functions are one of the ways to do this.

A function is just a small section of code from a larger program. They often contain some isolated variables for their own operation and can call other functions in the same file (in most environments - Java's concept of privacy and things like it can make that statement more complicated).

I considered putting this section under *patterns* because the way functions are utilized can change drastically from paradigm to paradigm, but really it is a data structure for instructions. I will bring function up again when talking about paradigms. To see some general styles of function definitions, please feel free to refer back to the code examples in 0.1.1.0, 0.1.2, and 0.1.0.1.

0.2.0 argument:

There are a few methods for functions to get outside data to work with (such as globals, which we will get to), but by far the most common is to pass data in via arguments. A set of arguments and a return type from a function make what is called a *function signature*. I feel Scala does a good job at isolating function signatures for easy of readability:

```
Scala
def which_is_greater(x:Int, y:Int): Int {
    if(x < y){
        return y
    }else{
        return x
    }
}
```

The **(x:Int, y:Int): Int** section of that definition is the function's *signature*. The area in parenthesis are the arguments. Given that signature, you can see that **which_is_greater** takes two arguments, **x** and **y** (both of which are integer types). There is no reason that these arguments couldn't also be booleans, floats, objects (which will be covered later), or other functions (as shown in the *closure* example by passing in a lambda as an argument).

0.3.0.1: pass by value:

Some care should also be taken when taking about exactly how a function get access to the variable passed in as an argument. Often, in many languages, this is done by passing the argument's value, which is to say that the function copies the value of the passed argument for its own private use. Anything the function does to its copy does not affect the one that was passed in the first place. Here is an example of passed by value in Python:

```Python
>>> x = 5
>>> def add2(x):
...     return x + 2
...
>>> add2(x) #the function adds two
7
>>> x # but the original variable stays unaltered
5
```

0.2.0.2 pass by reference:

In contrast to passing by value, there is pass by reference. Instead of copying the value of the argument, the function copies that address of the original variable and applies changes there (which alters the original). This is done for many reasons, but most often it is for performance (both speed and memory footprint, not taking the time to carry over values). Python passed many arguments by value, but (to the dismay of new programmers), it passed lists by reference instead:

```Python
>>> x = [1,2,3,4] # make some list
>>> def mutating_list_concat(lst, to_add):
...     return list(map((lambda x: lst.append(x)),to_add))
...
>>> mutating_list_concat(x,[3,4]) # pass the original list and another small one
[None, None] #append returns nothing, modifying in place
>>> x
[1, 2, 3, 4, 3, 4] # the values of x
```

If **x** were passed by value, the join lists from the function would have died once the function exited. However, since it was passed by value, the in-place append's changes persisted outside the function in the original variable.

0.2.0.2.0 pointers:

On the nitty-gritty level of this passing business, it really comes down to whether the arguments are passing values or *pointers*. A pointer is the address of something in memory. Most languages hide these details because they are a notoriously great way to shoot yourself in the foot. Thankfully for this example, C has no issue with a programmer doing something possibly stupid:

```c
#include <stdio.h>

void doubleVal(int x){ x = x * 2; }

void doubleRef(int *x){ *x = *x * 2; }

int main(){
    int by_val_int = 5; //this is a non-pointer variable in C
    int by_ref_int = 5; //Though I have not declared this as a
                        //pointer, C allows me to access
                        // and pass it's address
    doubleVal(by_val_int); //pass by value
    doubleRef(&by_ref_int); //pass address (as noted by the '&')
    printf("The non-pointer int is still %d, while the\
    pointer int is now %d.\n", by_val_int, by_ref_int);
}
```

The above code's output is:

```bash
keep-away:~ ldavis$ gcc pointer.c
keep-away:~ ldavis$ ./a.out
The non-pointer int is still 5, while the pointer int is now 10.
```

As you can see, the function that used the variable's address mutated it in place where the other one did not.

0.2.0.3 lambda:

In certain languages, functions are implemented using a data structure called a lambda. A lambda is, in essence, a function that can be treated as a value like an integer. The example of Javascript in section 0.0.0.1 is a function assignment leveraging lambdas being assigned in the same way an integer is treated. Any function in Racket is a lambda, hence the assignment of both variables and functions being nearly identical. Lambdas are often used in more functional-leaning languages (Lisp, Haskell, and so on). Languages that don't promote functional style may have lambdas, but will cripple their usage to a very limited set of cases, such as Python's lambda limitation to single expression languages. But I am going to stop here because I will talk more about functional programming later.

1 pattern:

If data structures are the core elements of a language, patterns are the common ways to utilize data structures to improve readability, construct maintainable systems, or best leverage unique parts of the language. Do you pass by pointer or by value? Should a function reference variables outside of it or does everything have to be passed in? Should this be a class method or a global function? These are the questions of patterns.

1.0 scope:

Scope was hard to put in data structures or patterns, since I feel it kind of lives in both. A scope is some context which data can be referenced in. Though the kinds of scopes available are a question of data structure, the way that scope is used and thought about is heavily focused on patterns. In Python, you can reference a global variable (something I will cover shortly) in a function, but due the community's agreed patterns, this is often thought of as a terrible idea.

1.0.0 namespace:

The term namespace is often used when referring to one specific scope context. "the file namespace" or "the class namespace" are used to talk about a level of scope that entails certain access to certain variables. The name space tends to shrink in size as you get more specific. The "standard namespace" tend to just be the built-in language, which can be quite a bit. The "struct namespace" is quite small because it is very specific. Things in the same name space cannot have the same name, though over multiple name spaces a name can occur more than once. Namespaces allows for a programmer to make functions, files, and libraries without worrying about naming collisions.

1.0.1 global:

One of the larger namespaces is *global*. This means that everything (typically restricted to the file it's defined in) can see it. Functions can access it, variables outside of functions can access it, sometimes other files can access it. They can all do this as if it were defined in the current scope itself. This is often considered a bad design move because it becomes very hard to manage a large number of functions accessing the same variable without any way to tell who or when they are accessing it. Here is an example of a globally scoped variable being referenced in a function's scope:

```c
#include <stdio.h>

int THIS_IS_GLOBAL = 5;

int main(){
    printf("the value of THIS_IS_GLOBAL is %d\n", THIS_IS_GLOBAL);
    return 0;
}
```

The output:

```bash
keep-away:c_code ldavis$ gcc global_example.c
keep-away:c_code ldavis$ ./a.out
the value of THIS_IS_GLOBAL is 5
```

1.0.2 lexical:

Lexical scope is a scope marked by lexical structure, such as a function's code block. Two items can often have the same name as long as they are in separate code blocks (thus lexically separated). As an example, here is a small snippet of Python that contains two identically named variables with different values:

```python
>>> example = 5
>>> def foo():
...     example = 10
...
>>> example
5
```

The only reason the second definition of *example* doesn't stomp on the first is because they are lexically separated, thus having different name spaces.

1.2 comment:

Comments are a way to mark up code for others (and yourself). It can mention implementation details, HOWTOs, DOTOs, and even tests. The mindsets some get in about comments can be described as zealotry, but they are required if you want to pass around code between programmers.

1.3 test:

Though comments are a great way to tell someone what you meant code to do, often intention does not match action. Tests are a way to programmatically verify that code you wrote functions as intended. Many languages now include test libraries to aide in this process. Others go as far to embed test notation in the comments sections themselves.

I am not trying to say tests are better than comments. They both need to be present. Comments tell programmers further down the line what the code snippet should do. The tests verify that the snippet does just that.

1.4 interface:

Often though, when writing code for other projects, you don't need to comment every section or test every line. No other programmer needs to know what that one integer *i* is for that is buried in some helper function. What people want to use and know about is the code's interface, how to most simply integrate it into their own programs.

Some languages use the term interface rather abstractly, to refer to a set of functions in a library. Others take the term in a very literal manner, such as Java, which require special code structures called *interfaces* that your personal code must implement to interface with an external library.

1.4.0 package:

External code is often organized into a *package*. This is not always true; sometimes there are interfaces for sections of code that cannot do anything independently, one of the prime expectations with a packaged library. A package should either be entirely self contained or should tell you where to get the code it needs to run (these are known as dependencies). These package are also often set up in a manager such as *npm* for the node.js community) *pip* for Python, *cargo* for rust programmers, and *raco* for the Racketeer.

1.4.1 API:

An API is a more concrete version of the first interface definition I gave earlier. API's, typically, are a collection of functions meant to hide the dirtier work of external code. For instance, if there is a program to draw windows on a screen, I don't want to set up logic to correctly detect the operating system my program is running on to know which system calls to make for screen drawing. That logic is the work of the external library and I should just be able to call something like *screen.draw()* and have it work.

1.4.2 callback:

A common interface idiom that breaks the "self sufficient" trait of packages and libraries are *callback functions*. A callback function is a function that you need to define that the library will try to call on to get some information. These are very common in game and javascript programming and are not entirely dissimilar to Java's Interfaces.

1.5 macro:

In the most general sense, a macro is some method for expanding code from something relatively small or simple into something more complex. This could be simple code replacement (as it is in C's preprocessor) or dynamic expansion (as it is in Lisp) at runtime. Many languages shy away from macro-like behavior entirely. C's preprocessor has been claimed to be an ill-thought-out part of the language, shying programmers away from the concept when learning other languages. The reason macros have taken such a hold on Lisp is because of Lisp's trait of homoiconicity. Other languages that have this trait utilize macros as well (such as Red and Rebol). Here is an example of macro replacement in C:

```c
#include <stdio.h>

#define this_will_be_replaced_with_5 5

int main(){
    printf("the value of THIS_IS_GLOBAL is %d\n", this_will_be_replaced_with_5);
    return 0;
}
```

After stopping the compilation step at C's linker, (** -E**), we get the following macro text replacement:

```c
int main(){
    printf("the value of THIS_IS_GLOBAL is %d\n", 5);
    return 0;
}
```

Note that this output wat greatly truncated to exclude hundreds of lines of system definitions GCC injects.

1.6 syntactic sugar:

Often, over a language's life, people want a shorter way to write something that is common, like function definitions. Other times, language designers just forsee nice shortcuts to more easily write code. These layers of the syntax of a language are known as *syntactic sugar*. For instance, a function is Racket is a lambda, but most people don't like the write a lambda every time they define a function, they want a short hand:

```
(define (foo x)
    (* x x))
(define bar
    (lambda (x)
        (* x x)))
```

Both are equivalent, but one is markedly shorter than the other.

1.7 imperative:

Now we move onto paradigms. Paradigms are what turn discussions of data structures into discussions of patterns. In object-oriented programming, functions become methods and in function programming immutability becomes the default (if not only) operation. But first some time should be spent on imperative programming for chronology's sake and context for why the other two paradigms came to be.

Imperative languages are abundant. C, Python, Racket, Javascript, Rust, Kotlin, Bash, and Perl all support imperative styles of coding. The basic structure of an imperative program is a "to do list." The largest benefit of this style is, in my opinion, that it's conceptually easy. If you were to describe the sequence of tasks a program must complete, it very likely will mirror the structure of an imperative implementation of said program. Both of the other paradigms I will talk about rarely mirror this characteristic so readily.

The downside to imperative programming is that it doesn't emphasis or naturally provide good measures to modularize or encapsulate code. There are many C files that have magic variables defined at the global level to use within the file. It comes down to the language and the programmer to enforce good modularity and protection of data.

For a concrete example of imperative programming (vs the other two paradigms I am about to go over), please reference the answers to the second section of this test.

1.8 object:

An object is one of the fundamental concepts of Object Oriented Programming (OOP). The style was originally brought into the spotlight by languages like Smalltalk (though it never gained mass popularity due to it's slow speed). Languages like C++ and Java have brought OOP into the mainstream set of programming patterns.

The core idea of OOP is to encapsulate variables and functions (often called properties and methods in the paradigm) into *classes*. These classes can be instantiated as a variable in either a accessor function or object when the program is run. An instantiated class is called an *object*. These objects, in most cases, maintain their own copies of variables and methods, allowing easy protection of data a compared to imperative programming (though certain concepts such as the keyword *static* in Java make it so that things are shared across class instances).

This paradigm came out of the need to abstract many common operations across large code bases and to allow easier understanding of concepts that did not mesh well with newer technologies in computers (such as mice, multiple screens, and network sockets). The encapsulation of variable and function names also allowed code to more easily be shared in between programmers.

1.8.0 class:

As mentioned above, a class is the definition of an object to be used when it is instantiated.

1.8.1 method:

Methods are a special kind of function that is typically allowed unfettered access to an object's properties. I stop short of Larry Wall's position and the idea behind the *bless* keyword though. Largely a method acts the same as a function, though it is restricted to the namespace of it's parent and cannot be used until an object containing it is instantiated. Though, of course, Java allows you to get around that detail (with our friend *static*).

1.8.2 inheritance:

To fulfil OOP's goal of abstraction, that needs to be some way to build upon abstract classes into more and more specific use cases. Think of writing a generic file handler class. It can open and close some files, but that's it. You want to implement one for image files next. You want the opening and closing form the other class, but you also need methods to change the color mode of the image, rotate it, and so on. So instead of rewriting all of this, any OOP language worth it's salt has a thing called *inheritance*. Object inheritance allow a class to inherit all the properties and methods of a parent (sometimes called *super*) class. So instead of rewriting the open and close methods for files, you would just inherit them from the former class.

Some languages have implemented inheritance in markedly different ways. One of the many long standing points between between Java and C++ is their respective inheritance models. Java allows for only single class inheritance, which is to day that any class can only ever have one parent class to it. C++ on the other hand allows for as many parent classes as you could want in a single class definition. The Java fans claim that their model avoids the mass complexity that C++'s model has. The C++ people rave about how much more rich their classes can be from this open inheritance. I feel like both of these problems can be solved using wrapper methods around objects instead of inheritance. So the jury is still out on who got it right.

1.8.3 side effect:

A side effect is when a method or function gives some form of output that is not directly returned. This can be I/O, mutation of a object property, or the write-out of some file while returning a status code to say whether it is successful. Side effects are commonly critiqued as a downside of OOP because they are thought to add a large amount of complexity and state-dependant behavior to a project, making it harder to debug. Qualities like this are one of the largest factors to the resurgence of the next paradigm.

1.9 functional:

The only concrete thing I can say about functional programming is that there are a surprizing amount of articles trying to define functional programming. Consider function programming to be stateless/side-effectless programs, others chalk it up to simple immutability. Functional programming is actually one of the earliest paradigms, starting with Lisp is the 50s, but it lost popularity due to some conceptual overhead. Now with the rising popularity of languages like Clojure and Haskell and the fatigue the collective programmer's mind has with concepts like semaphores, functional trends are on the rise.

Though I cannot give a more of a concrete definition for functional programming than the vast number of articles on the internet, I will talk about traits found in programming languages that tend to more functional in style.

Tells of a functional language are functions like *map*, *filter*, *trampoline*, or having a *!* at the end of words (such as Racket's *set!*). Some other language concepts that tend to be function in nature are *expression based*, *monads*, and *recursion* (which I will mention later). Two common ideas among functional patterns are immutable variables and side effectless functions. It comes back down to the grade school definition of a function:

```
A function is a set of operations that have a set number of inputs and a set number o
The relation between inputs and outputs should be reproducible.
```

There can be no "change this global value if x is prime and then return true." That creates a side effect that will only change an outer variable on some inputs (a common practice in OOP, such a polling whether a file buffer has reached its end). Another thing that functional programming tries to limit is stateful patterns. Many languages (like Clojure and Haskell) don't allow you to reassign new values to declared variables. Other than limiting a programmer's ability to create side affects, it drastically cuts down the number of concerns in concurrent programming (which I will cover in a later section, but it deserves mention here.)

1.10 iterate:

So this is a fairly basic topic, but I felt it best to nest it after the talk about paradigms because the topics changes so drastically from paradigm to paradigm. Both OOP and imperative programming share many of the same forms of code iteration patterns. For loops, while loops, and do-while loops are fairly ubiquitous. These all are just ways to repeat the same block of code over and over again. Some languages have special iterators such as

1.10.0 recursion:

However, all of the major loop type (for, while, and do-while) are state-based patterns of iterating. Functional programming has to figure out a way around stateful iteration: recursion. A recursive function calls itself to pass data through it over and over again until some conditional check is satisfied. This hides the state of the iteration in the stack trace outside of the programmer's realm of control.

Some languages love recursion. Lisps often have an optimization call *tail-call recursion*. If the last executed part of a function is a recursive call to itself, Lisp will eliminate the previous stack frame in memory, thus making the memory foot print of a recursive function a fixed amount (to the frame size of a single function call).

Other languages hate it. Python actually limits the recursion depth of a function:

```Python
>>> def foo(x):
...     print(x)
...     foo("x = "+(str(x+1))
...
>>> foo()
#a very large stacktrace later:
RecursionError: maximum recursion depth exceeded while calling a Python object
x = 997
```

However, as mentioned in the section of lambdas, Python's hate of functional idioms (and crippling them) is nothing new.

1.11 concurrent:

Concurrent programming is coding in a way to solve multiple instances of the same subsection of a program at the same time. If you have a list of 100 numbers that you wanted to test to see if there was a prime number in the collection, you could go one by one or you could test all at the same time. As more and more cores have been crammed onto chips, this has become more and more of an option to increase the speed of a program.

OOP and imperative paradigms can do concurrent programming though you need to work about *race conditions*. A race condition is when two sections of code want to access the same variable at the same time. One changes it in the middle of another reading it and you end up with a wrong or corrupted value. To help with this, variables are *locked* to only allow code that holds a special piece of data called a *semaphore*. The passing of semaphores during variable mutation allows one to avoid race conditions, but also hurts the benefit of concurrent programming in the first place.

Functional programming tries to solve this issue by leveraging immutability. If a variable cannot change, how would a race condition occur? The data cannot be changed so no code can corrupt it.

1.12/1.13 threads & forks:

Threads and forks are two ways to implement concurrency at the process level. Their origins come from Window and Unix. Unix projects tackled the problem of concurrency by saying “let’s just split the concurrent process off from the calling process and give them their own memory and CPU time”. Though this is conceptually easier, it costs a lot of resources to do so.

Windows (more aptly NT) said “let’s keep all processes in the same OS process and just weave their tasks to in and out of the main program execution.” This model of concurrency does not require all of computational time to split data like a fork does, but it can require a much more complicated system to implement well. Both of these concepts came about around the time web servers started to become more seriously used. Forks proved too expensive for web traffic so that is why we have had to deal with NT servers for websites for all those years.

1.13 exception:

Failure is a guarantee in programming. Bugs are bound to happen, even if you are not the one to make them. User input, for instance, is a great way to cause errors when some user types “help” into your calculator program that parses input as integers. Many languages deal with this by using exceptions. Again this is a topic that could go into data structures because often an exception (and a throw) violate stack procedure, but the most common difference between language’s exceptions are how to use them, not how they are useful.

Two of the larger mind sets in exception handling are “fail fast and fix yourself” or “don’t do anything, log the error, and stop what you are doing.” Python is of the former and Java is the later.

1.13.0 throw:

When some operation goes really FUBAR, you might need to actually throw an error.

2/3 compiled & interpreted:

An area that use to be very clearly defined in programming languages, but has become much murkier in recent years is the distinction between interpreted and compiled languages.

In the clearest terms, a compiled language is one where source files are written and then passed to a *compiler*. The compiler takes the source language and translates it into some other language (often a native binary). Often, every time you want to change the program, the whole thing needs to recompiled in incorporate new changes or it needs to be run on a new computer. With technologies like *incremental builds* have made it so that smaller compiles are needed for just the sections of code bases that changed. Other facilities like the Win32/DLL API for Windows systems allows a single binary to run across multiple systems.

Interpreted languages are fed to a run-time interpreter that often holds the source code in some intermediate representation (IR). This IR is used to execute code in a virtual machine for the language. One of the largest benefits to a programmer using an interpreted language is that coding can often be done interactively (by feeding code lines to the VM directly), not needing to wait through long build times (like a compiled language), and very little system-specific code needs to be written on the level of the application (because they target the VM on the system). The cost is often in speed because what is effectively a smaller compilation has to happen for every new line and the entire system access has to be done through yet another level of abstraction on top of the Operating System (though languages like Forth have largely come to match compiled languages in speed).

Most languages are no longer purely interpreted or compiled anymore. As people wanted the runtime speed of compiled languages and the developer speed of interpreted languages, these two practices have merged. Java is both compiled and interpreted, as is Racket. The Java Virtual Machine (JVM) and the MZ Scheme Runtime both have native byte codes for their VM environments. When running Java, Scala, Kotlin, or Racket code, the source can be compiled into byte-code and run on the VM. This increases the speed of execution but saves many of the benefits on language interpreting. Another technology in this direction is JIT compilation. JIT ("Just in time") is the process of compiling languages as they are being read into an interpreter, recognizing common patterns in the source code, and streamlining their interpretation by making templates to just read in the changing parts of the pattern. This is a common technology in most Javascript engines in browsers.

4 lexical analysis:

(NOTE: I am assuming this is a topic because "lexical" appears twice in the original list of words.)

A lexical analyzer, more commonly called a "lexer," is a tool used to analyze and categorize each individual word (called a "token") in a source code file. The lexer does this to ease the process for the parser (the next step in compiling or interpreting a language). Lexers are usually made by strapping together a collection of regular expressions to match tokens with their token type.

5 parse:

Whether a language is interpreted, compiled, or some mixture of the two, all languages need to go through some parsing step. A parser's goal is to take a stream of tokens from a lexer and derive some semantic structure from it. This could be properly attributing statements to a code block of a function definition, separating statement lines, and so on. What a parser should return is some Abstract Syntax Tree (AST), a data representation of the the program's structure. Some languages make this process very simple (such as Forth or Lisp), others make it incredibly difficult (such as Scala or Perl). It's all a matter of their grammar and the of code you have to read to understand the context of the word you are currently focused on. Parsers themselves are often considered one of the more complicated parts of a compiler or interpreter, but I will give a brief overview and comparison of some of the ideas.

The two main classes or parsers are LL (left reading, left most derivation) and LR (left reading, right most derivation). Both can be followed by a number (such as "LL(4)") to denote how many tokens from the lexer the parser can look ahead when parsing. The difference between to the two classes are in how they recognizes larger forms of grammar.

An LL parser, sometimes called a "top down" parser, finds a word and guesses what grammar rule applies to it and tries to parse it further. An LL parser starts with the entire file and attempts to throw grammar rules at it until you get to tokens. This attempt to guess what rule applies can get LL parsers in some trouble. For instance, if a grammar has a left-recursive rule, the parser can get caught in a loop of parsing forever. A left-recursive rule is a grammar rule that the first pattern of its matching expression is a recursion reference to itself. This quality of LL parsers restricts them to a limited subset of grammars.

LR parsers are far more flexible though much harder to program from scratch. An LR parser has a *workspace* that takes in tokens, progressively generalizing them into larger and larger grammar forms. This quality of starting with tokens and working your way up to grammar forms with why LR parsers are often called "bottom up" parsers. Because LR parsers work with tokens first, they avoid the recursive confusion that LL parsers can get into, making them far more flexible. There are also many different sub catagories of LR parsers (such as LALR, SLR, and GLR), but they all still function on a very similar premise.

Where the two parsers come back together is in that fact that few people make them from scratch. Many different languages have tools to feed grammar specifications into another library and get a parser. The programs are called Parser Compilers. C has Yacc/Bison & Lex/Flex (for lexical analysis), Racket has BRAG (Beautiful Racket AST Generator), and Java has ANTLR.

6 stack overflow:

A stack overflow is the event that happens when the data assigned to a pointer in memory (written by the programmer or not) exceeds the actually allocated amount of space for that pointer, overwriting other data. This is often used to crash or exploit systems.

Bibliography:

- [1]: ynniv. "Closures in Python." *Blogger*. Published: August 7, 2017. http://ynniv.com/blog/2007/08/closures-in-python.html

- [2]: "4.11 Vectors." *racket-lang.org*. Accessed: April 11th, 2017. https://docs.racket-lang.org/reference/vectors.html

- [3]: "Vector (Java Platform SE 7)." *Oracle*. Accessed: April 11th, 2017.

https://docs.oracle.com/javase/7/docs/api/java/util/Vector.html

- [4]: Fleseriu, Gabriel.“C++ Tutorial: A Beginner’s Guide to std::vector, Part 1.” *CodeGuru*. Published: February 25, 2003 http://www.codeguru.com/cpp/cpp/cpp_mfc/stl/article.php/c4027/C-Tutorial-A-Beginners-Guide-to-stdvector-Part-1.htm

Question 2 (40%):

Write six programs implementing solutions to the following two problems across the three languages with different styles. (These may be the same three languages from question 1 but don't need to be.)

In each case, include docs and tests appropriate to the style of that language, including explicitly what verision of what language you ran, in what environment, what steps compiled and/or ran the code, and what the input and output looked like.

Use these programs to illustrate some different currently popular programming paradigms, as well as your mastery of the vernacular within these programming language communities.

Standard libraries and extensions are allowed, but the more you can do yourself the better.

As a postscipt, discuss which languages you found well suited to which problem, and why.

The two problems are

A) fraction sum search

What permutation of the digits from 1 to 9 will add to 1 when arranged in a form similar to 1/23 + 4/56 + 7/89 ? Find the answer with a brute force search.

B) web crawler visualization

Write a program which will first build a network of URLs and the links between them by starting at a given web page and following its links outwards, and then generate a visual representation of that network.

All the details are up to you, including which page to start at, how far to go, and how to display the result.

Answer:

The postscript will be included here, but for the answer's code, see the appendix.

Postcript on Question 2 part 1:

This problem was pretty straight forward in all three paradigms. Rackets functional idioms such as maps and filters made the entire process dead simple. This lines up with my opinion that functional languages are better equipt to tackle problems that are mathematical in nature. Pythons OOP structure felt like a lot of boiler plate for a such a simple task. Kotlin, unlike Python and Racket, did not include some way to generate permutations of a given list, so I needed to make one. I tried a few different methods, but they were all a little buggy or required a lot of parameters to know what to do. So I found a nice method for generating permutations quickly, which is cited in the source code. This problem was a lot more straight forward than the next one, so I don't have much to say about it.

Postcript on Question 2 part 2:

So this question was a little beefier than the first one. It was more direct in it's language of what it wanted, but what it wanted was less specific in how to do it. The hardest choice was what tool to use to visualize the web page links. Each language I used does have a GUI library which could have been used, but I am not experienced in any of those APIs. It would have been a lot to tackle in a week with the other questions. So I decided to have them all target the same visual framework: dot graphs. I have made .dot graph parsers before and the translation from lists to graphs put the solution a little bit more into muy wheel house with the compilation element of this solution. So each paradigm example produces a .dot spec to be read directly, compiled with Graphviz, or interactively rendered by something like Gephi. Racket and Python both had reasonable http request libraries, so I used them for the functional and OOP examples respectively. Originally I was going to use C for all the imperative examples, but than I got to this problem and changed my mind. Have you ever heard someone say "I love how C handles strings"? No? So I tried to switch tyo Node.js. I got the html parsers in place, but than I ran into a snag using their http request library. It is all asynchronous. I couldn't just grab everything and than process it sequentially. So I looked around so a way to synchronously call for web pages from Javascript and was largely met with phrases like "embrace asynchronous programming."[1] So I also dropped Node.js. In panic, I tried to think of an imperative language that I knew which included a sane http library. I had done some programming in Scala and Java, which are both object oriented, which isn't useful, but they use a lot of the same type and notation of an imperative JVM language: Kotlin. Kotlin had access to all of Java's http libraries and can be as imperative as you want as you want it to be. It actually was fairly pleasant to program in as well. So I implemented the web grapher in Kotlin and redid my imperative answer to the fraction sum finder to match my new choice in languages.

Bibliography:

- [1]: Crowder, T.J. "How to get around the asynchronous Node.js behaviour?" *Stack Overflow*. Accessed: April 12th, 2017. http://stackoverflow.com/questions/28930034/how-to-get-around-the-asynchronous-node-js-behaviour

Question 3 (20%):

Discuss the strengths and weaknesses of these programming languages as you see them. Feel free to discuss a few other languages that you're familiar with as well. What sorts of problems or situations are good fits to these languages, and why? Which do you personally like, and why? Be specific, giving examples that justify your comparisons and conclusions. (This may well cover some ground you've already discussed in the previous two problems. If so, you don't have to repeat any of that, just refer back to it and bring up anything that you feel hasn't yet been brought forward.)

Answer:

The Tools for the Job:

First and foremost, I don't think a single language or paradigm has complete and total advantage over every other competitor in their category. I would not want to write a physics engine in Java, and I would not want to write an input handler in Racket. But flip the languages for each others problems and I am more than happy to work with them.

I like LISP but the drone of people like Paul Graham saying it's the end-all be-all frustrate me. The problem you want to solve should inform the language you want to use, not the other way around. In a slightly different, but related perspective to Leslie Lamport's statements of "silver bullet math" being an unproductive venture, I feel that the hurdles and twists we put in languages so they are more generally productive often complicate and fragment out ability to understand them.[1]

Monads are not the best way to deal with print statements, they are a construct to deal with decisions that languages like Haskell made further up stream in their development cycle. Java's need to have setter and getter does not clean away all of the woes related to accessing properties of an object, they are needed boilerplate methods to deal with Java's privacy model.

Clojure's immutability makes it an excellent choice for concurrent programs. Rust's low-level access makes it a great choice when experimenting with kernel development. Racket's macro system makes it a great candidate for language compiler implementation.

Programming should not be about some choice abstraction that tries to solve everything at the cost of intensely idiomatic constructs. If you understand the problem and it's qualities, the language, the paradigm, and the architecture will all fall into place.

C, The Unending:

Despite *Hacker News* being covered in articles about the new “C killer” language on, what seems like, a daily basis, C is not going anywhere soon. Too many core technologies are heavily written in it. The linux kernel, for example, is about 15 million lines of straight C code.[2] Any new language that wants to get into the system-level space also needs to interop with it as well (such as Rust).[3] Otherwise, the loss of not being able to quickly interface with the 40 years of development done in C is just too costly.

I avoided C in this test for reasons I stated in the postscripts for question two, but I still think it is a language that programmers should know. The problems that C has are not all its fault. It’s a language developed before the internet was a concern and before GUI’s were wide spread. String manipulation, network programming, and graphics can all be done in C, but the programmer needs to be skilled enough to do it without causing damage to those who use it down the line. You can make safe programs in C, but few programmers have the discipline to actually do it. This problem is solved by either adopting a safer language or coming up with a better way to find these kinds of subtle bugs (to get around the problem of the programmer who doesn’t “waste” the keystrokes to check for null pointers).

Racket & LISP:

I talk about Racket to some extent in my paper of making the MinNo compiler (which is %100 Racket code), but I will echo some of that here as well. Racket has done a great deal of work trying to be more than it’s education-centric roots. Its reputation for being a language platform is improving and its use in video games is getting it noticed. It sadly still sucks for command line programming. Its process library feels lacking when matched up against Python and Scala/Java’s, and its error handling is incredibly verbose given its “contract” system of error detection. I would really like to see these two areas improve in the short term.

Languages like Clojure are appealing to me more and more given that they understand the academic interest of LISP, but they also understand want a language needs to do in order to be generally useful. Their models of concurrency also leave Racket’s in the dust in term of simplicity.

Kotlin, Scala, Java, and the JVM:

The JVM language space is a maelstrom of death and birth. Old Java alternatives like Groovy are getting a second wind.[4] Newcomers like Scala and Clojure are getting their time in the spot light. Oddities like Ceylon are dying before they even leave the gate, and others like Kotlin are watching the carnage from a safe but stagnant distance.[5] All of this is happening under the looming shadow of Java 9, which promises to be a rather large update with its jigsaw packaging system.[6]

Each language has it’s own appeal, but largely you still have to deal with Java and the JVM once you get to a large scale project. Clojure will get all the have of a LISP and a JVM language at once. Kotlin will always have the reputation of coming from a company that is only known for making Java IDEs. Scala will continually implode as it reinvents itself due to some new research.[7]

I am excited to work with these languages more, but on a platform that is so widespread and so heavily researched (as the JVM is), I feel like they are a sub-field of programming languages entirely.

Python and Why I Want to Leave It:

Python is largely a fine language. I am not going to make some crazy statement like, “Python 3 is not turing complete”.[8] It does the job, and for most people it does that job well. I like how easy it is to set up an object. I enjoy how quickly I can test my code. Their doctest model is actually pretty cool. But I need a language that I don’t need to fight with.

I fight with the community. The adoption of Python 3 is nothing short of a water wheel running through a giant stew of terrible ideas, flinging them in every single direction.[9][10] Think if someone approached you and said, “Hi, I’m a Java 6 developer.” That’s what it is like.

I fight with the runtime constantly. The python runtime is not meant to handle recursion and poorly handles concurrency. It might be nicer to install than something like the Scala or Clojure runtime, but that means little if it’s less useful once it’s working.

I fight with the syntax constantly. This is a large undertone of my Plan. As a dyslexic reader, Python’s syntax provides all the same challenges that read english text does. It doesn’t matter if I am reading or writing it, I have used the syntax because it is the *lingua franca* of the Computer Science department here. Once I move on from Marlboro, I hope to move on from Python as well.

Looking Ahead:

In 10 years the Python engine could be a masterfully engineered concurrent beast, Groovy could kill Java 10, and Racket might finally ditch the cartoon-y input bar when reading user input in Dr.Racket. I don’t know what things will look like then. The languages I use now could very well vanish entirely by then. I know what I know now is not what I will need to know in 10 years. Thing will be different.

When learning to code I think I identified that learning language families is far more beneficial than learning any particular syntax within them. If you know ML, why it came to be, and why people use it Ocaml, Scala, subsections of Ruby, and newer languages like Elm immediately make more sense. Knowing C and Smalltalk make Java a far more comprehensible beast. Learning the semantics and the intentions behind them will have a much further reach than learning any particular syntax based on them.

Between these two facts, I feel I am a reflexive programmer. If a new language needs to be learned, point me towards an example and I will figure it out. If a new paradigm surfaces in popularity, I will spend a few days reading about the conditions from which it emerged to prepare for the wave of languages following the craze.

This understanding of the constant change of technologies and the history of developments that backs them, I feel like I will be able to tackle each new project I work on in a similar manner stated at the start of the paper: Not relying on some multi-tool I learned to live by, but by analyzing the problem, it's qualities, and finding out what techniques & technologies could best be used to tackle it.

Bibliography:

- [1]: Meijer, Erik. “Erik Meijer and Leslie Lamport - Mathematical Reasoning and Distributed Systems”. Channel 9 video, 37:20. Posted March 9th, 2010. https://channel9.msdn.com/Shows/Going+Deep/E2E-Erik-Meijer-and-Leslie-Lamport-Mathematical-Reasoning-and-Distributed-Systems

- [2]: “The Linux Kernel Open Source Project on Open Hub.” *Blackduck Software*. Published: April 27th, 2017. https://www.openhub.net/p/linux

- [3]: “Foreign Function Interface.” *rust-lang.org*. Accessed: April 11th, 2017. https://doc.rust-lang.org/book/ffi.html

- [4]: Krill, Paul. “Ruby, Groovy post surprise gains in popularity” *IDG*. Published: March 7th, 2016.http://www.infoworld.com/article/3041333/application-development/ruby-python-groovy-post-surprise-gains-in-popularity.html

- [5]: McAllister, Neil. “Red Hat's Ceylon language is an unneeded tempest in a teapot.” *IDG*. Published: April 22nd, 2011. http://www.infoworld.com/article/2624027/java/red-hat-s-ceylon-language-is-an-unneeded-tempest-in-a-teapot.html

- [6]: Hanson, Jeff. “Modularity in Java 9: Stacking up with Project Jigsaw, Penrose, and OSGi”. *IDG*. Published: February 2nd, 2015. http://www.javaworld.com/article/2878952/java-platform/modularity-in-java-9.html

- [7]: “Dotty.” *École Polytechnique Fédérale de Lausanne* Accessed: April 11th, 2017. http://dotty.epfl.ch

- [8]: Shaw, Zed. “The Case Against Python 3 (For Now).” *Shavian Publishing*. Accessed: April 11th, 2017. https://learnpythonthehardway.org/book/nopython3.html

- [9]: Harris, Naftali. “Why I'm Making Tauthon” (previously “Why I'm Making Python 2.8” as shown by the URL). Published: November 30, 2016. https://www.naftaliharris.com/blog/why-making-python-2.8/

- [10]: cpeterso “Why I'm Making Python 2.8 (naftaliharris.com)”. *ycombinator*. Published: December 10th,

2016. https://news.ycombinator.com/item?id=13144713

Exam 2 - Algorithms:

Question 1 (40%):

Explain, implement, explore numerically the O() behavior either Prim's or Kruskal's algorithm, which finds the minimum spanning tree of a weighted graph.

The core of this work should be a numerical experiment in which you randomly generate graphs of different sizes, find either the time or number of steps the algorithm takes, use those to create plots of its behavior, and compare the shape of the plot with the expected result.

As usual with my assignments, your code should be clearly documented with explicit tests.

Be clear that you should choose one of these to do, not both.

Answer:

Chosen algorithm: Prim's

Description of the algorithm: Given a weighted graph *T*, construct a tree that shows the minimally costful traversal of the entire graph. Find this by:

- 1) Either choose a root for the tree or have some method for choosing a root (such as finding the least costful edge of the entire graph and starting from some node it leads from).
- 2) Choose the lowest costing edge from any node actively connected to the root node. If an edge is found, the node that it leads to is now active. Any further consideration of edges will include this connection.
- 3) repeat step 2 until all nodes of the graph have been visited.

Prep Work:

To implement this algorithm, we first need some way to represent a weighted graph. For this we will need some class to maintain information about each node of a given graph and the class to manage all of the graph's nodes.

(note: fully commented versions of this code will be included with this paper.)

```python
import math, itertools, random, copy, time

class node(object):
    def __init__(self, name="none"):
        self.name = name
        self.connections = {}

class directed_graph(object):
    def __init__(self):
        self.start = None
        self.node_pool = {}
```

The class **node** handles our graph nodes, their names and each node's connections. The manager class **directed_graph** will handle things like starting nodes and all of the instantiated **node** objects related to one another.

Now this is all we need to start representing graphs, but I don't feel like writing these objects and their properties out by hand, but I also want control over what is attached to what. So I will make a Domain Specific Language (DSL) that compiles to these directed graphs. The DSL syntax is a stripped down and slightly modified version of the dot graph syntax. A directed graph in this language that connects node "A" to node "B" by weight "5" would look like:

```
A ->(5) B;
```

This will produce a **directed_graph** object containing nodes "A" and "B". Node "A"'s **.connections** will conatin the tuple **("B", 5)**. This will also make my life easier when running tests. While compiling the usable Python object, the compiler should also produce a dot graph file so we can see how it parsed out spec. Here is the compiler:

Python

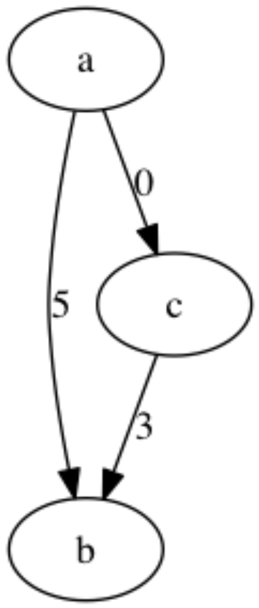
```python
class gen_graph(object):
    def __init__(self, spec):
        self.graph_spec = spec
        self.graph = directed_graph()
        self.dot_spec = ""


    def parse_spec(self):
        rules = list(filter((lambda x: x != ''),\
                            self.graph_spec.replace("\n", "")\
                                           .split(";")))
        for rule in rules:
            parts = rule.split(" ")
            origin = parts[0]
            edge_weight = self.get_weight_val(parts[1])
            dest = parts[2]
            self.create_node(origin, edge_weight, dest)

    def get_graph(self):
        return self.graph

    def display_dot_spec(self):
        print("digraph viz{\n"+self.dot_spec+"}")

    def create_node(self, start, weight, end):
        if start in self.graph.node_pool:
            if end in self.graph.node_pool:
                self.dot_spec += "{} -> {} [label=\"{}\"];\n"\
                                     .format(start, end, weight)
                self.graph.node_pool[start].connections[end] = \
                              (self.graph.node_pool[end], weight)
            else:
                self.graph.node_pool[end] = node(end)
                self.create_node(start, weight, end)
        else:
            self.graph.node_pool[start] = node(start)
            self.create_node(start, weight, end)

    def get_weight_val(self, weight_part):
        return int(weight_part.split("->")[-1][1:-1])
```

Here is an example of its use:

```
test_graph = """
a ->(5) b;
a ->(0) c;
c ->(3) b;
"""
graph_compiler = gen_graph(test_graph)
graph_compiler.parse_spec()
graph = graph_compiler.get_graph()

print("Test graph has nodes {}".format(list(graph.node_pool.keys())))

for parentnode in graph.node_pool:
    for subnode in graph.node_pool[parentnode].connections:
        print("\t Node {} has connection {} of weight {}"\
              .format(parentnode,
                      subnode,
                      graph.node_pool[parentnode]\
                           .connections[subnode][1]))

print("\ngraphViz spec:")
graph_compiler.display_dot_spec()
```

```
Test graph has nodes ['c', 'b', 'a']
     Node c has connection b of weight 3
     Node a has connection c of weight 0
     Node a has connection b of weight 5

graphViz spec:
digraph viz{
a -> b [label="5"];
a -> c [label="0"];
c -> b [label="3"];
}
```

Upon compiling the dot graph output (usage: $ dot -Tpng -O), we get:



So the graph representation and the DSL compiler work.

The Search Code:

Now that we have something to traverse, we can get onto implementing Prim’s algorithm. This will have a search method that taks a graph to search and an anchor node to start the tree from. There will also be a method to start by finding the node with the cheapest edge in the graph. These two things combined will allow us to choose whether to specify the root of our min-span tree or whether to let the searcher find the cheapest, greedy traversal of the graph. The searcher should also produce a dot-spec to represent the min span tree so we can visually compare it to the original graph. We can actually get both the Python usable version of the graph and the dot graph by leveraging the DSL that we already have. The search just needs to contruct the DSL encoding of the min-span graph and call the DSL compiler on it. Here is the source:

Python

```
class prim_search(object):
    def __init__(self):
        self.currently_considered_nodes = []
        self.visited = []
        self.graph = None
        self.dot_spec = ""
        self.tree_spec = ""

    def search_anchored_min_span(self, graph, starting_node_id):
        self.currently_considered_nodes = [starting_node_id]
        self.visited.append(starting_node_id)
        while len(self.visited) < len(graph.node_pool):
            best_move = ["none", "none", math.inf]
            for current_node in self.currently_considered_nodes:
                for subnode in graph.node_pool[current_node].connections:
                    weight = graph.node_pool[current_node].connections[subnode][1]
                    if subnode in self.visited:
                        continue
                    if best_move[2] > weight:
                        best_move = [current_node, subnode, weight]
            self.currently_considered_nodes.append(best_move[1])
            self.visited.append(best_move[1])

            self.tree_spec += "{} ->({}) {};\n".format(best_move[0],
                                                       best_move[2],
                                                       best_move[1])

        compiler = gen_graph(self.tree_spec)
        compiler.parse_spec()
        self.graph = compiler.graph
        self.dot_spec = compiler.dot_spec
        return self.graph

    def display_dot_spec(self):
        print("digraph viz{\n"+self.dot_spec+"}")

    def search(self, graph):
        node_with_cheapest_edge = (None, math.inf)
        for node in graph.node_pool:
            for connection in graph.node_pool[node].connections:
                weight = graph.node_pool[node].connections[connection][1]
                if weight < node_with_cheapest_edge[1]:
                    node_with_cheapest_edge = (node, weight)

        return self.search_anchored_min_span(graph, node_with_cheapest_edge[0])
```

The weight of the work is done by **prim*search.search*anchored*min*span**. Given a graph and a starting node, it will search the list **self.currently*considered*nodes** until **self.visited** contains the same number of nodes as the passed graph. It will find the cheapest edge on all considered nodes, follow it, and update the lists of what is being considered and visited accordingly. As it is doing this, the search also compiles the min-span tree in the graph DSL. Once the search is done, it passes the results DSL spec to a **gen_graph** object and returns the usable version of the min-span graph while assigned the dot spec to **self.dot_spec**.

The method **search** wraps **search*anchored*min_span**. It will find the cheapest edge of the graph and pass that to **search*anchored*min_span**, effectively acting as our "non specificied root" search.

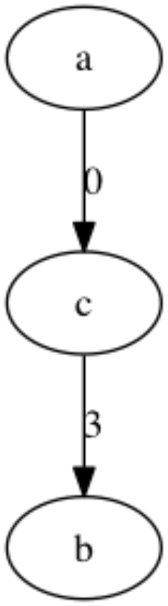
To verify that this works, let's use the tiny graph from earlier. The result should be a tree that is simply "a -> c ->

b”. The connection from “a -> b” is far more expensive than “a -> b” or “c -> b”.

Python

```python
search = prim_search()
result = search.search(graph)

print("Test graph has nodes {}".format(list(graph.node_pool.keys())))

for parentnode in result.node_pool:
    for subnode in result.node_pool[parentnode].connections:
        print("\t Node {} has connection {} of weight {}"\
                  .format(parentnode,
                          subnode,
                          result.node_pool[parentnode]\
                          .connections[subnode][1]))

print("\ngraphViz spec:")
search.display_dot_spec()
```

```
Test graph has nodes ['c', 'b', 'a']
     Node c has connection b of weight 3
     Node a has connection c of weight 0

graphViz spec:
digraph viz{
a -> c [label="0"];
c -> b [label="3"];
}
```



So, as expected, the min-span version of our earlier graph is “a -> c -> b”.

Testing Big-O Behaviour:

To measure the big-O growth of Prim’s algorithm, we will need some way to generate a random graph of a given density. This can be done by generating a graph on N nodes that are all symmetrically directed to every other node with random weights for each edge. This will give us a consistently dense graph given some *n* that will produce different min-span trees because the weights throughout the graph are always different. To do this, we can just generate specs for the DSL language:

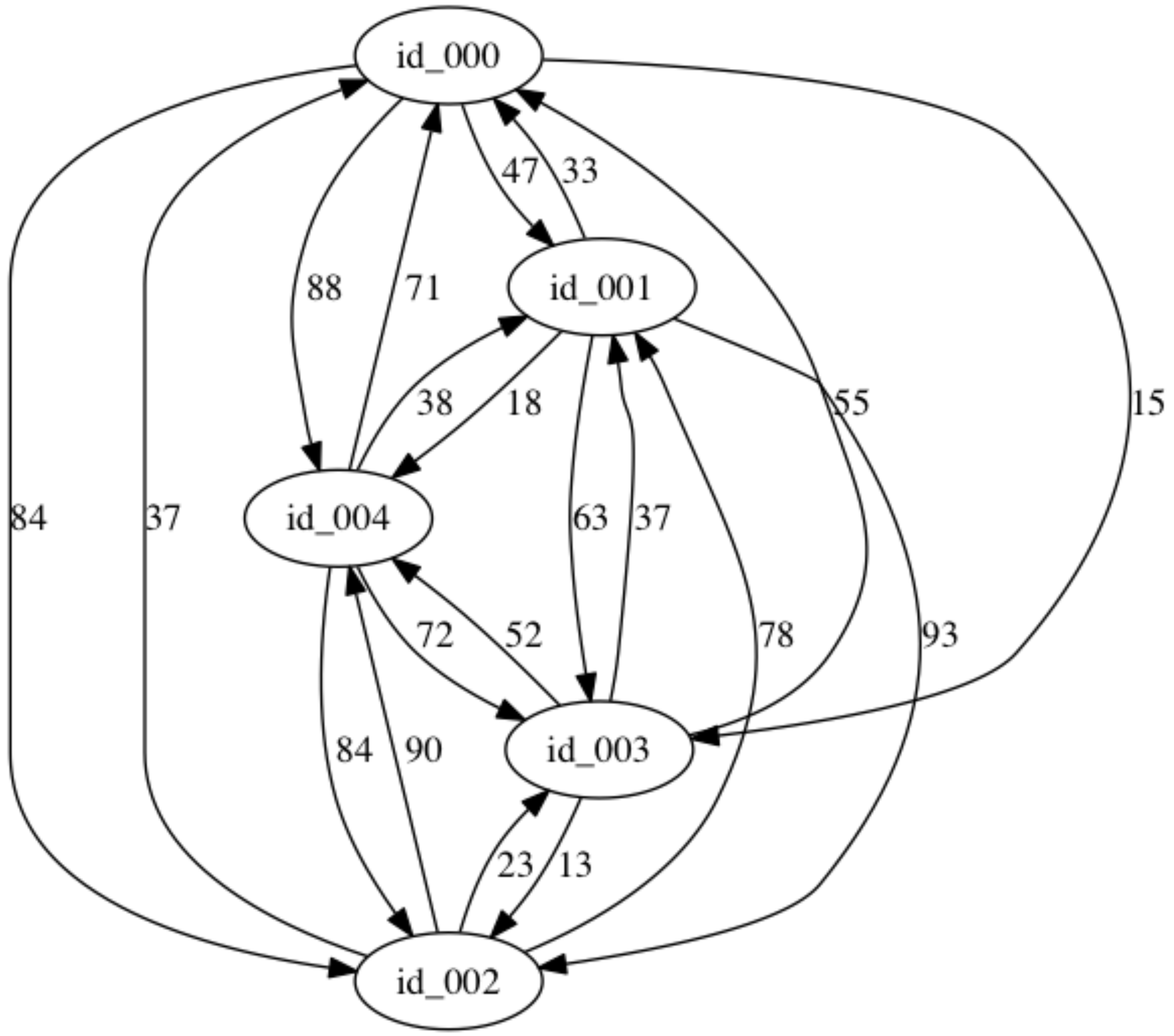
```Python
class tangled_directed_graph_spec_gen(object):
    def __init__(self, nodes):
        self.nodes_num = nodes
        self.symbol_table = []

    def gen_symbol_table(self):
        i = 0
        for x in range(0, 100):
            for y in range(0, 100):
                for z in range(0, 100):
                    if i >= self.nodes_num:
                        return
                    else:
                        self.symbol_table.append("id_{}{}{}".format(x, y, z))
                    i += 1

    def run(self):
        self.gen_symbol_table()
        return self.generate_graph()

    def generate_graph(self):
        if self.symbol_table == []:
            self.gen_symbol_table()
        spec = ""
        for i in range(0, self.nodes_num):
            for x in range(0, self.nodes_num):
                if x == i:
                    continue
                else:
                    spec += "{} ->({}) {};\n".format(self.symbol_table[i],
                                                     random.randint(0, 100),
                                                     self.symbol_table[x])
        return spec
```

To show that this works, lets generate a symetrically directed graph of 5 nodes:

```python
graph_maker = tangled_directed_graph_spec_gen(5)
graph = graph_maker.run()

compiler = gen_graph(graph)
compiler.parse_spec()
compiler.display_dot_spec()
```

```
digraph viz{
id_000 -> id_001 [label="47"];
id_000 -> id_002 [label="84"];
id_000 -> id_003 [label="15"];
id_000 -> id_004 [label="88"];
id_001 -> id_000 [label="33"];
id_001 -> id_002 [label="93"];
id_001 -> id_003 [label="63"];
id_001 -> id_004 [label="18"];
id_002 -> id_000 [label="37"];
id_002 -> id_001 [label="78"];
id_002 -> id_003 [label="23"];
id_002 -> id_004 [label="90"];
id_003 -> id_000 [label="55"];
id_003 -> id_001 [label="37"];
id_003 -> id_002 [label="13"];
id_003 -> id_004 [label="52"];
id_004 -> id_000 [label="71"];
id_004 -> id_001 [label="38"];
id_004 -> id_002 [label="84"];
id_004 -> id_003 [label="72"];
}
```

Here is the compiled specs result:



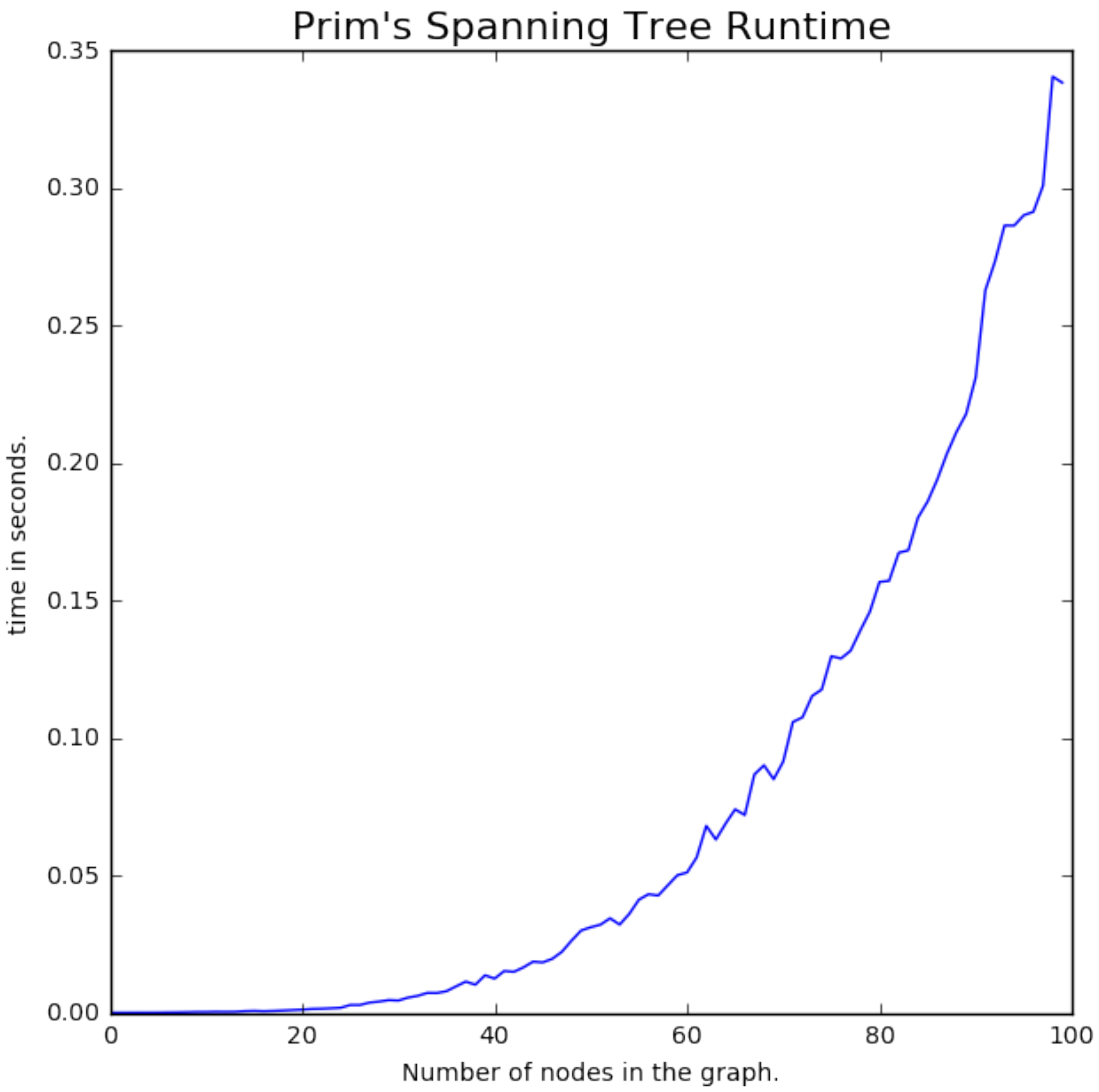
As the gut-feeling-assumption of how Prim's works, given the current implementation of the Prim search, this spec generator will cause it to perform its worst case growth pattern. Every time the search looks for a new edge to traverse, it has to look over every considered node, which will be *n*. At each node check it also has to search over every node it is connected to. Since this graph is entirely semetric in it's connections (every node it attached to every other node), this subnode search is also *n*. Therefore this implementation is n^2. To test this, we can time the search over graphs 0 to *n* in size (of node count). Each *n* will be tested 5 times and their results will be averaged.

Python

```python
def get_big_O_for_prim(n=100, step=1):
    times = []
    for i in range(0, n, step):
        subtimes = []
        for x in range(0,5):
            graph_gen = tangled_directed_graph_spec_gen(i)
            starting_graph_spec = graph_gen.run()
            compiler = gen_graph(starting_graph_spec)
            compiler.parse_spec()
            starting_graph = compiler.get_graph()
            searcher = prim_search()

            start = time.time()
            searcher.search(starting_graph)

            end = time.time()
            subtimes.append(end-start)
        times.append([i, (sum(subtimes)/10)])
    return times

timing = get_big_O_for_prim()
```

Python

```python
#graph code
plt.figure(figsize=(7,7),dpi=600)
plt.xlabel('Number of nodes in the graph.')
plt.ylabel('time in seconds.')
plt.title("Prim's Spanning Tree Runtime", fontsize=16)
plt.plot(list(map ((lambda x: x[0]), timing)),
         list(map ((lambda x: x[1]), timing)))
```



So the graph supports the gut-assuption of a n^2 growth pattern (minus the few system pausing-related stops). Now this can be improved by altering the search for cheap edges among nodes. If connections are held in some structured or ordered list (such as a sorted array or a heap) the search time can be cut down to some nicer growth pattern.

Question 2 (30%):

Illustrate a depth-first and breadth-first tree search, preferably using a stack and queue, with the tree of possible moves in a triangular peg solitaire game as the tree.

The game I have in mind starts with this configuration

```
    .
   * *
  * * *
 * * * *
* * * * *
```

where each * is a peg in a hole, and the . is an empty hole. The goal is to remove pegs by jumping as in checkers, leaving one in the center as the final position.

Answer:

In a practical sense, the difference between depth-first and breadth-first searches of game trees is whether you are focusing on a single game's progression at a time (progressing through it more quickly), or you are considering all games and the sub games in equal amounts (more evenly considering each possible option). The both of these search traversal techniques can be implemented in the same game solver with a simple toggle in the search-queue system. If we have a list to hold all game states to process, how we access that list will determine whether we are doing a breadth-first or a depth-first traversal. Say we have a game state (abstractly). Let us call this state "A". After processing it we see that there are two valid moves in some defined ruleset. We store these game states an "AA" and "AB". Each of those have two moves ("AA" goes to "AAA" and "AAB" while "AB" goes to "ABA" and "ABB") and so on. As these new states are found, we append them to the list of new game states to be considered. If we treat this list of game states as a queue (first in, first out), we will construst a tree in the following manner:

```
NOTE: symbols in angle-brackets are nodes currently being
considered. Symbols in square brackets are new nodes
connceted to the node currently under consideration.

step 1:       <A>       step 2:     <A>
             /                      / \
           [AA]                    AA [AB]



Being considered: A     Being considered: A
Found Move: AA          Found Move: AB
States to process:[AA]  States to process:[AA, AB]



step 3:      _A_             step 4:        _A_
            /   \                          /   \
          <AA>  AB                       <AA>  AB
          /                              /  \
       [AAA]                           AAA [AAB]

Being considered: AA         Being considered: AA
Found Move: AAA              Found Move: AAB
States to process:[AB, AAA]  States to process:[AB,AAA,AAB]

step 5:      __A__                  step 6:     __A__
            /     \                            /     \
          _AA    <AB>                        _AA    <AB>
         /  |     |                         /  |     | \
       AAA  AAB [ABA]                     AAA  AAB ABA [ABB]

Being considered: AB                Being considered: AB
Found Move: ABA                     Found Move: ABB
States to process: [AAA, AAB, ABA]  States to process: [AAA, AAB, ABA,ABB]


 And so on...
```

By moving towards the oldest entry in the “states to be processed” we are just progressing down the game tree on move at a time in all possible move sets. This allows us to computer the game tree in its “breadth” first, considering all avenues of the game equally. By constrast, if we treat the “states to be processed” as a stack (first in, last out), we end up with a depth-first search:

```
step 1:       <A>       step 2:     <A>
             /                      / \
           [AA]                    AA [AB]



Being considered: A     Being considered: A
Found Move: AA          Found Move: AB
States to process:[AA]  States to process:[AA, AB]



step 3:      _A_             step 4:        _A_
            /   \                          /   \
           AA  <AB>                       AA  <AB>
          /                              /  \
       [ABA]                           ABA  [ABB]

Being considered: AB         Being considered: AB
Found Move: ABA              Found Move: ABB
States to process:[AA, ABA]  States to process:[AA,ABA,ABB]

step 5:      __A__                       step 6:     __A__
            /     \                                 /     \
           AA     AB_                              AA     AB_
                 /   \                                   /   \
                ABA <ABB>                              ABA  <ABB>_
                    /                                       /     \
                 [ABBA]                                   ABBA   [ABBB]

Being considered: ABB                    Being considered: ABB
Found Move: ABBA                         Found Move: ABBB
States to process: [AA, ABA, ABA, ABBA]  States to process: [AA, ABA, ABA,
                                                             ABBA, ABBB]

And so on...
```

This is effectively considering one game at a time, until it terminates because it found a win or lose condition. If the termination of the branch was because the solver determined a loss, it will go back to the nearest unconsidered node and continue from there.

To illustrate a solver for triangular peg solitaire that does both depth and breadth searches, I will present implementations of a solver with a togglable (between a stack and a queue) list to store game states to compute. It will also inlcude a dot-graph compiler to create graphs of each search to show the difference between breadth and depth searches in practice.

Constructing The Helpers For The Solver:

(Before getting into the code, for the sake of neatness, the comments have been stripped from the code presented here, but a commented version will be included with this PDF.)

First we will need to import a few libraries to give us some utilities we will need later.

Python
```
import copy, math
```

The library **copy** will be needed because we will need to **deepcopy** board objects in the solver. We will also want the option of limiting the solver's number of iterations (like when we want a graph that isn't gigantic). To accomplish this we will just have a **step_limit** property in the solver, but when we want it to run however long it needs to, we will assign it the value **math.inf**, hence importing **math**. This will allow us to tune the solver to a specific iteration amount or just let it exhaust a game tree if need be.

Next we will need some way to encapsulate information about different game states:

Python
```
class peg_board(object):
    def __init__(self,rows):
        self.rows = rows
        self.board = self.construct_board(rows)
        self.history = []


    def construct_board(self, rows):
        board = []
        for i in range(1, rows+1):
            board.append(list("."*i))
        return board

    def init_board(self):
        for i in range(1,self.rows):
            for x in range(0,len(self.board[i])):
                self.board[i][x] = "*"

    def print_board(self):
        offset = self.rows
        for i in self.board:
            print((" "*offset)+" ".join(i))
            offset -= 1

    def print_history(self):
        mock_board = peg_board(self.rows)
        mock_board.init_board()
        turn_counter = 0
        for move in self.history:
            y, x = move[0]
            y_off, x_off = move[1]
            y_kill, x_kill = move[2]

            mock_board.print_board()
            print("TURN {}: move (x:{}, y:{}) to (x:{}, y:{}) \
            and remove (x:{}, y:{}).".format(turn_counter,
                                             x, y,
                                             x+x_off,y+y_off,
                                             x+x_kill,y+y_kill))
            mock_board.board[y][x] = "."
            mock_board.board[y+y_off][x+x_off] = "*"
            mock_board.board[y+y_kill][x+x_kill] = "."
            turn_counter += 1
        mock_board.print_board()
        print("Done!")
```

So we now have a way to store a game state. **peg_board.board** is a 2d array that will hold our trangular board of **peg_board.rows** size. The board has a **print_board** function to *pretty print* the currently contained board to ease some output later. Lastly, each board as a **history** property that will contain every move made on the board. This allows us to display the winning set of moves later by using the **print_history** method.

Next we will need a symbol generator for the dot graph compiler to name nodes:

```python
class symbol_table(object):
    def __init__(self):
        self.symbols = range(0,100).__iter__()
        self.cap = 100
        self.current = 0

    def next(self):
        self.check_if_end()
        self.current += 1
        return str(self.symbols.__next__())

    def check_if_end(self):
        if self.current == self.cap-1:
            self.symbols = range(self.cap, self.cap*2).__iter__()
            self.cap = self.cap*2
```

The dot graph compiler will need a stream of unqiue symbols to give each game state. This object, when instantiated, will give us a unique symbol (from any other string it produces) by calling **.next()**. The **.check*if*end** method will extend the table of symbols if we reach the end of the obejct’s internal stream.

The last thing we need to get in place for the solver is a dot graph compiler. This can be incredibly special purposed to just this solver, so it won’t be anything fancy:

```python
class peggame_dot_writer(object):
    def __init__(self):
        self.dot_spec = ""

    def add_dot_edge(self, origin, dest):
        self.dot_spec += "\""+origin +"\""+ "->" + "\""+dest+"\";\n"

    def get_graph(self):
        return "digraph peggame {\n"+self.dot_spec+"}"
```

The compiler just accumulates graph rules by calling **.add*dot*edge** and passing is a start and end label (from the **symbol_table** class). So if I pass **“a”** and **“b”** the resulting rule would be **“"a” -> “b”;\n"**, making a directed edge from **“a”** to **“b”**. To get the full graph **.get_graph** appends the correct header and footer the graph spec for compiling.

Now we can get onto the solver:

```python
class peg_board_solver(object):
    def __init__(self):
        self.boards_to_do = []
        self.search_mode = "breadth"
        self.step_limit = math.inf
        self.make_dot_spec = False
        self.dot_writer = peggame_dot_writer()
        self.symbol_table = symbol_table()
        self.solved = False


    def get_pegs_moves(self, board, x, y):
        offsets = [[(-2, -2), (-1, -1)],
                   [(-2,  0), (-1,  0)],
                   [( 0, -2), ( 0, -1)],
                   [( 0,  2), ( 0,  1)],
                   [( 2,  0), ( 1,  0)],
                   [( 2,  2), ( 1,  1)]]
        valid_moves = []
        for off_set in offsets:
            if (y+off_set[0][0] < 0) or (x+off_set[0][1] < 0):
                continue
            try:
                if (board[y][x] == "*") and\
                   (board[y+off_set[0][0]][x+off_set[0][1]] == ".") and\
                   (board[y+off_set[1][0]][x+off_set[1][1]] == "*"):
                    valid_moves.append([(y,x)]+off_set+[id(board)])
            except IndexError: #off the board
                continue
        return valid_moves

    def realize_board_step(self, board, move):
        new_board = copy.deepcopy(board)
        y_origin, x_origin = move[0]
        y_dest_off, x_dest_off = move[1]
        y_kill_off, x_kill_off = move[2]
        new_board[y_origin][x_origin] = "."
        new_board[y_origin+y_dest_off][x_origin+x_dest_off] = "*"
        new_board[y_origin+y_kill_off][x_origin+x_kill_off] = "."
        return new_board

    def step_board(self, board_obj):
        moves = []
        for row in range(0,len(board_obj.board)):
            for peghole in range(0,len(board_obj.board[row])):
                valid_moves = self.get_pegs_moves(board_obj.board, peghole, row)
                if  valid_moves != []:
                    moves.append(valid_moves)
        for origin_set in moves:
            for move in origin_set:
                new_board_state = self.realize_board_step(board_obj.board, move)
                new_board = peg_board(len(new_board_state))
                new_board.name = self.symbol_table.next()
                new_board.board = copy.deepcopy(new_board_state)
                new_board.history = copy.deepcopy(board_obj.history+[move])

                if self.make_dot_spec:
                    self.dot_writer.add_dot_edge(board_obj.name, new_board.name)

                if self.winner_check(new_board):
                    self.winning_path_found(new_board)
                else:
                    self.boards_to_do.append(new_board)

    def winner_check(self, board_obj):
        if board_obj.board[0][0] != "*":
            return False
        for i in range(1,board_obj.rows):
            for peghole in board_obj.board[i]:
                if peghole == "*":
                    return False
        return True

    def winning_path_found(self, board_obj):
        print("A winner was found:")
        board_obj.print_history()
        self.solved = True

    def get_mode(self):
        if self.search_mode == "depth":
            pop_index = -1
        elif self.search_mode == "breadth":
            pop_index = 0
        else:
            print("Unrecognized searchmode state {}.\
                   Defaulting to breadth-first.".format(self.search_mode))
            pop_index = 0

        return pop_index

    def find_winner(self, board):
```

```
        print("Starting search with board:")
        board.print_board()
        print("Searching...")
        self.boards_to_do.append(board)

        pop_index = self.get_mode()
        step = 0

        while not self.solved:
            if self.boards_to_do == []:
                print("No winning path was found. Sorry.")
                self.solved = True
                continue
            if self.step_limit < step:
                print("Halted due to step limit\
                       of {} being reached.".format(self.step_limit))
                self.solved = True
                continue

            current_board = self.boards_to_do.pop(pop_index) #stack
            self.step_board(current_board)

            step += 1

    def breadth_first_solve(self, board):
        self.search_mode = "breadth"
        self.find_winner(board)

    def depth_first_solve(self, board):
        self.search_mode = "depth"
        self.find_winner(board)

    def play_depth(self, size):
        start_board = peg_board(size)
        start_board.init_board()
        start_board.name = self.symbol_table.next()
        self.depth_first_solve(start_board)

    def play_breadth(self, size):
        start_board = peg_board(size)
        start_board.init_board()
        start_board.name = self.symbol_table.next()
        self.breadth_first_solve(start_board)
```

So that is a lot of code in one run, but I will break it down. So first we define a few properties for the class:

- self.boards*to*do: this is a queue/stack to store game states for later consideration.
- self.search_mode: this will contain a string of either “depth” or “breathe”.
- self.step_limit: the limit we can set to our iterations through the game tree.
- self.make*dot*spec: a toggle of whether to output a dot spec.
- self.dot*writer: an instance of the peggame*dot_writer class to compile dot graphs.
- self.symbol*table: an instance of the symbol*table class to name peg_board objects for graphing.

After the initial properties, the first thing (to me) is to deal with the manipulation of boards. Specifically, the first thing to do is get this solver to be able to identify a valid move for a given peg. Given a **peg_board**, an x index, and y index, **get*pegs*moves** will return all valid moves given peg on the board at *x,y* co-ordinates. It does this by applying a set of off sets to make sure that there is no peg where it wants to move and that there is a peg between it and the new space. If a move is valid, the origin, destination off set, and inbetween offset are all put into a list. The list of all valid moves for a given peg are what is returned.

Next is to apply **get*pegs*moves** to every peg on a given board, saving all new valid moves for later consideration. This is the work of **step_board**. **step_board** will go through each peg and call **get*pegs*moves** on each. All of the results will be gathered into a single list of all valid moves across the entire board. After all of that is processed, if **self.make*dot*spec** is true, the new state’s edges are appended to the dot graph for the search and the new board that it is processing is checked to see if it is in a win state. After all of this, it generates new board objects that represent each possible move and those are appended to **self.boards*to*do**.

To see if a board is in a winning state, **winner_check** makes sure that only a single peg is left in the board. This peg needs to be a the 0,0 index of the board (the top-single peg). This is a slight alteration from the parameters given in the initial question, but since the peg board can be made to be any size, this felt like a better way to handle the variable size. If **winner_check** returns **True**, than **step_board** will call **winning*path*found**. **winning*path*found** will print that a winning board was found, call **print_history** on the board, and output **self.dot_spec** for the search (if the dot writer was turned on).

All of this is wrapped in **find_winner**. Given some initial board, **find_winner** will manage getting the next board to feed **step_board**. How is does this is where the **depth-first/breadth-first** search comes in. A **pop_index** is set by **get_mode** which, depeding of what **self.search_mode** is set to, will return either **0** or **-1**. When used to pop a value from **self.boards*to*do**, an index of **0** will pop from the front of the list, where **-1** will pop from the back. This is how we treat of to-do-list as a queue or a stack, thus getting out **breadth-first** and **depth-first** searchs. **find_winner** will also handle stopping the search if either:

- A) We run out of possible game states, meaning there is no solution for that size board
- B) We reach self.step_limit

To close out the class there are four methods that are mostly wrapping things already defined.

breadth*first*solve and **depth*first*solve** both set **self.search_mode** to either “breadth” or “depth” respectively. These are in turn wrapped in **play_depth** and **play_breadth**, which both generate initial boards of *n* size and use those to start the game.

Lets see if it works.

First the depth search:

```python
depthsolver = peg_board_solver()
depthsolver.play_depth(5)
```

```
Starting search with board:
     .
    * *
   * * *
  * * * *
 * * * * *
Searching...
A winner was found:
     .
    * *
   * * *
  * * * *
 * * * * *
TURN 0: move (x:2, y:2) to (x:0, y:0) and remove (x:1, y:1).
     *
    * .
   * * .
  * * * *
 * * * * *
TURN 1: move (x:2, y:4) to (x:2, y:2) and remove (x:2, y:3).
     *
    * .
   * * *
  * * . *
 * * . * *
TURN 2: move (x:4, y:4) to (x:2, y:4) and remove (x:3, y:4).
     *
    * .
   * * *
  * * . *
 * * * . .
TURN 3: move (x:1, y:4) to (x:3, y:4) and remove (x:2, y:4).
```

```
     *
    * .
   * * *
  * * . *
 * . . * .
TURN 4: move (x:3, y:3) to (x:1, y:1) and remove (x:2, y:2).
     *
    * *
   * * .
  * * . .
 * . . * .
TURN 5: move (x:0, y:2) to (x:2, y:4) and remove (x:1, y:3).
     *
    * *
   . * .
  * . . .
 * . * * .
TURN 6: move (x:3, y:4) to (x:1, y:4) and remove (x:2, y:4).
     *
    * *
   . * .
  * . . .
 * * . . .
TURN 7: move (x:0, y:4) to (x:2, y:4) and remove (x:1, y:4).
     *
    * *
   . * .
  * . . .
 . . * . .
TURN 8: move (x:1, y:1) to (x:1, y:3) and remove (x:1, y:2).
     *
    * .
   . . .
  * * . .
 . . * . .
TURN 9: move (x:0, y:0) to (x:0, y:2) and remove (x:0, y:1).
     .
    . .
   * . .
  * * . .
 . . * . .
TURN 10: move (x:0, y:3) to (x:0, y:1) and remove (x:0, y:2).
     .
    * .
   . . .
   . * . .
  . . * . .
TURN 11: move (x:2, y:4) to (x:0, y:2) and remove (x:1, y:3).
     .
    * .
   * . .
  . . . .
 . . . . .
TURN 12: move (x:0, y:2) to (x:0, y:0) and remove (x:0, y:1).
     *
    . .
   . . .
  . . . .
 . . . . .
Done!
```

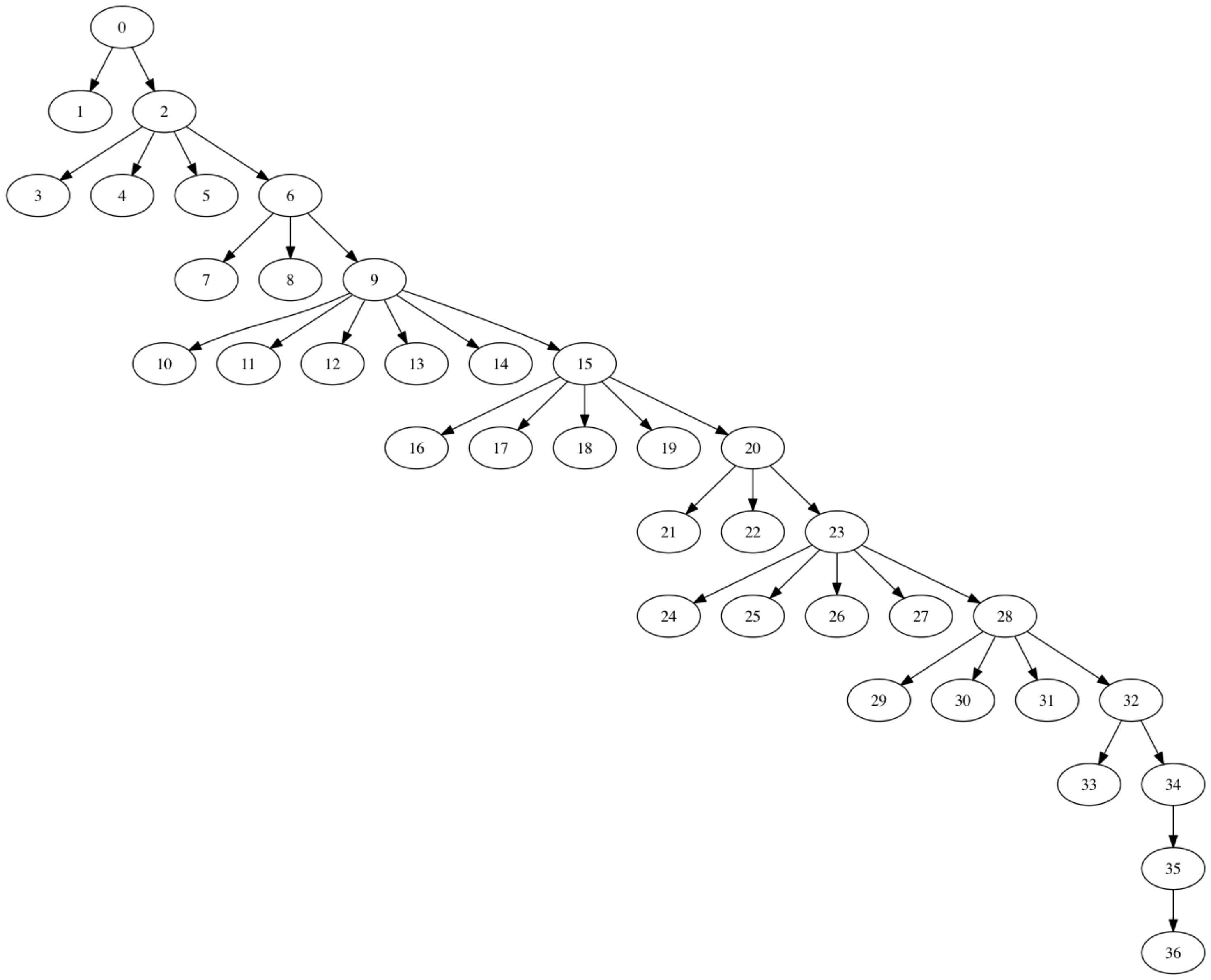
Next the breadth. Due to the explosion of game states as this game plays out, breadth search is prohibitively slow. I will cap this example at 20000 game state evaluations:

```python
breadthsolver = peg_board_solver()
breadthsolver.step_limit = 20000
breadthsolver.play_breadth(5)
```

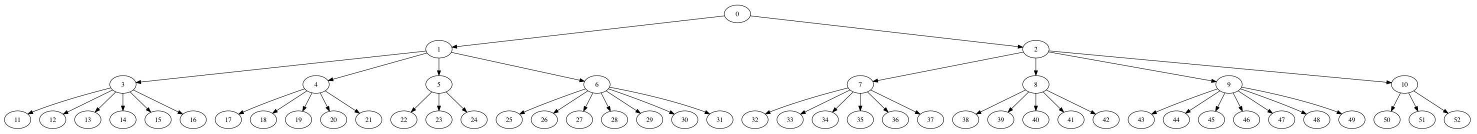
```
Starting search with board:
     .
    * *
   * * *
  * * * *
 * * * * *
Searching...
Halted due to step limit of 20000 being reached.
```

To show this solver work in both a **depth** and **breadth** method, I have them locally (capping their iteration to 10). The resulting dot graph spec has been compiled using GraphViz’s commandline tool. Each node’s label represents in which order the solver found that game state:

Depth:



Breadth:



As you can see, the solver, depending on which side it pops the newly considered game from the solver's to-do-list, can solve trianglular peg solitaire in either a depth-first or breadth-first manner.

Question 3 (15%):

The task of finding a given string from a collection of strings can be solved by algorithms that fit into several paradigms, including brute force and (if the list is sorted) divide-and-conquer.

- Given an example of an algorithm that fits each of these descriptions, each of these, and describe their O() behavior.
- Can you come up with a third algorithm design technique that can be applied to this problem, a representative algorithm, and its O() behavior?

(You don't need to code these - just discuss what's going on.)

Answer:

The brute force approach to collection searches is simply going through the collection in its entirety while checking each item (in this case a string) to see if it matches a pattern. A pseudo code algorithm for this would look like:

```
function naive_search( collection_of_strings,
                       match_pattern):
    index = 0
    for item in collection_of_strings:
        if item == match_pattern:
            return index
        index++

test_collection = ["aba","ccc","bbc"]
test_pattern = "bbc"
naive_search(test_collection, test_pattern)
2
```

The big-O of this algorithm would be nm where *n* is the length of the collection and *m* is the length of the match pattern. This behavior is fine in most cases, but in certain applications, like comparing passages of a book held in a list, this can cause a great deal of overhead.

A divide and conquer algorithm attempts to break up the problem into smaller pieces to more efficiently handle whatever job it needs to do. An example of this in the sorting algorithm space is *merge sort* which breaks apart collections into smaller lists, sorts them and combines the results to achieve better than n^2 behavior (the growth pattern of more naive sorting algorithms). A common cost of divide and conquer algorithms is memory foot print. Breaking up the problems into smaller pieces often requires multiple index variables, sub lists, ect. Thus the constant battle between speed and memory size rears its head.

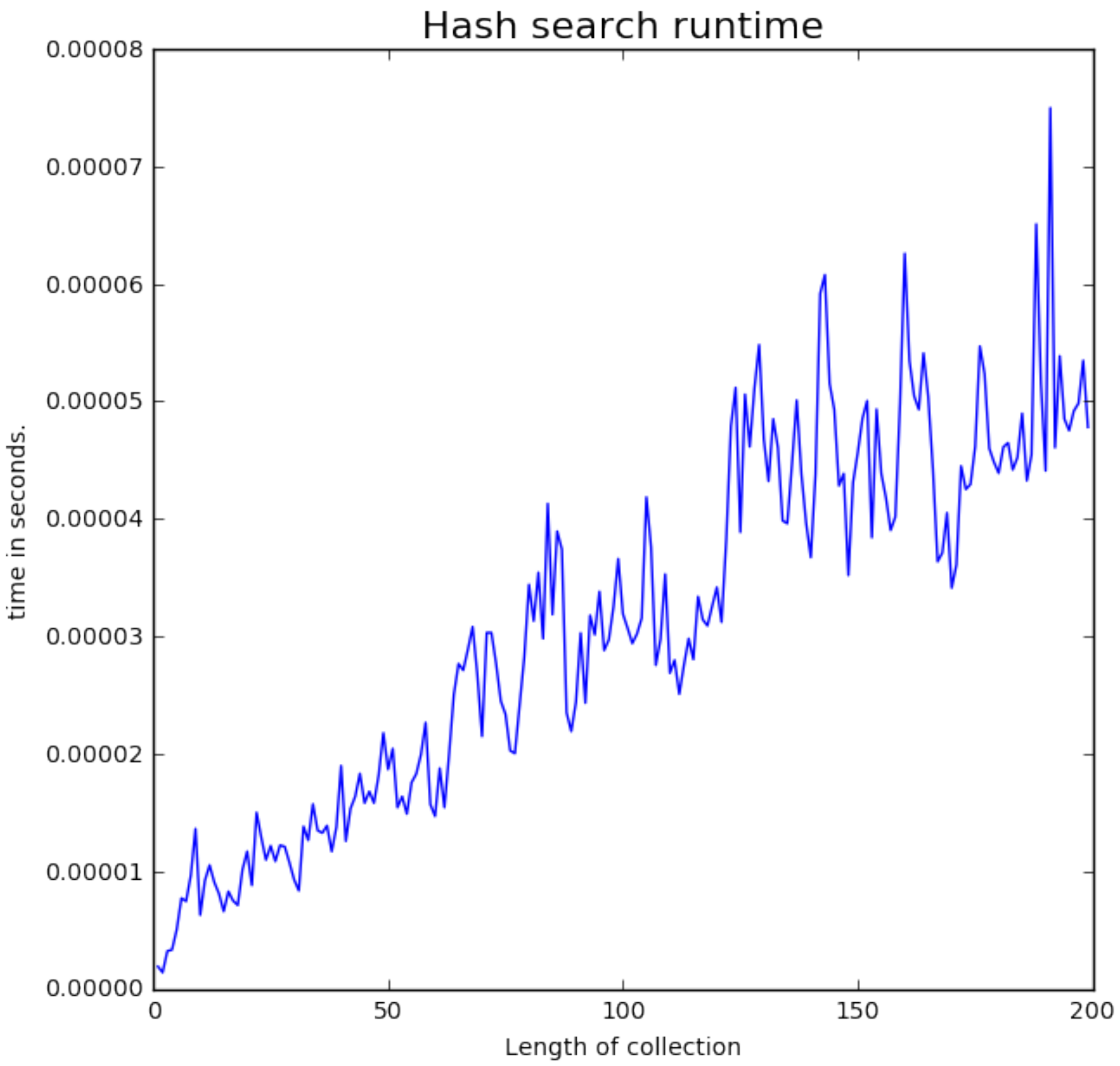
In the search algorithms, a divide and conquer technique can take two main flavors: optimizing either the traversal of the search space or the efficiency of checking a match. In numerical searches (finding a number in a collection), the strategy of optimizing traversals is a fairly easy. Binary trees can be constructed efficiently and greatly shorten the number of comparisons that need to be done. In consideration of strings, this becomes trickier. A tree-centric data structure could be used to ease traversal, but what is the pattern for splitting branches? length? What happens if there are more than one word of the same length (or worse, all of them are the same length)? A list iteration step will still be needed. Okay, what about lexicographical ordering? Alright, but to construct the tree a traversal of each string is still needed to differentiate the strings from one another. That still leaves us with a nm growth. To me (though this is probably deeply affected by the work I have done over my last two internships), the more natural way to optimize this in my head is to work on the comparison of values rather than their traversal.

If you were to iterate through the collection and use some fixed length hashing algorithm (that appended strings under that length), a creation of a hash table in average-linear time is trivial. The pattern you search for could be hashed and used as an index in the table. A single verification step would be needed to make sure the unhashed values match. The reason this is better than the naive search is because (on average) the comparison traversal of the search pattern is done only once, not on every item. This turns our nm to a $n + m$, or more tersely (to match the idioms of big-O notation *n*. This algorithm, of course, is more complicated once you factor in hash collisions (when two strings produce the same hash). Than you are also bound to the growth of the hash-collision algorithm which could be another indexing of the array or another hashing step. It would also increase our single comparison to however many are done before an empty index is found. But on average, with most hash centric algorithms, the access should be constant-time if done well which just gives us the *n* of constructing the table in the first place. This is simple enough to test in python, so I am just going to demonstrate this algorithm:

Python

```
import matplotlib
import numpy as np
import matplotlib.pyplot as plt
## generic graphing libs ^
import random, time

def search_via_hash(collection, pattern):
    hash_table = {}
    for index in range(0,len(collection)):   #construct hash_table
        hash_table[collection[index]] = index

    try:                               #return index
        return hash_table[pattern]
    except KeyError:                   #if hash isn't there, say so
        return -1

def get_big_O(n=200, step=1):
    times = []
    for i in range(1, n, step):   #from collection length 0 to 199
        subtimes = []
        for x in range(0,10):  # average timing over 10 tests
            items = []
            for x in range(0,i):   #construct collection of n length
                string = ""
                #make random string
                for z in range(0,random.randint(0,100)):
                    string += chr(random.randint(65, 110))
                items.append(str(string))

            start = time.time()
            search_via_hash(items, items[-1]) #find the last item
                                              #in the collections
            end = time.time()
            subtimes.append(end-start)
        times.append([i, (sum(subtimes)/10)])
    return times

timing = get_big_O() #get timing


#Make graph
plt.figure(figsize=(7,7),dpi=600)
plt.xlabel('Length of collection')
plt.ylabel('time in seconds.')
plt.title("Hash search runtime", fontsize=16)
plt.plot(list(map ((lambda x: x[0]), timing)),
         list(map ((lambda x: x[1]), timing)))
```

The results are:



So there is a pretty chaotic fluctuation between each length increment in the test, but the general trend of the search is linear. The likely reason for the jumps between *n*s is either collisions (like I mentioned earlier) or the dictionary (hash table) needing to extend to fit more entries (requiring time to allocate memory).

The divide and conquer aspect of this is that we are preprocessing each individual part in some way that aggregates to a larger, more easily solvable version of the problem we had before.

For a third algorithm, I will take about finite automata & regular expressions. Finite automata represent a state based machine for pattern recognition. A string could be fed through the finite automata and if it finds the pattern that you want (which can just be the sequence of characters of the string you want), it will end in an *accept state*. The key difference between this and the naive search is that all indexing in the string being searched and the pattern you are searching for is happening concurrently. This means that finite automata's big-O is *n* (linear) where *n* is the length of the input. To utilize this in searching a collection of strings, we just need to concatenate all the strings in the collection separated by some reserved character. The pattern we want to find will become the finite automata pattern or, more practically, regular expression (the coder's equivalent) in our search. The resulting regular expression will be run over the concatenated collection marking each reserved character in a counter. If we successfully match the regular expression, return the reserved character counter, otherwise, return **-1**, denoting the patterns absence.

```
function regex_index_search(collection, pattern){
    reserved_character_counter = -1
    aggregate_string = concatenate collection
                       using "#" (example reserved character)
    string_stream = aggregate_string //some iterator with
                                     //an internal index (like
                                     //Racket's ports or Java's
                                     //streams)

    lexer = lexer {            //some pseudo lexer
                               //construct similar to rackets
       search_regex -> match
       example_reserved_char -> inc
       else -> skip
    }

    while string_stream not done{
       match lexer(string_stream){
           case match:
               return reserved_character_counter
           case inc:
               reserved_character_counter++
           case skip:
               continue
       }
    }
   return -1
}
```

The catch about this pseudo code is that is uses both a lexer construct and a stream object. These are common constructs in certain groups, but if you aren't in one, the example might look like black magic. Streams are a collection of symbols (often characters). When you pass a stream to something that consumes it, it will increment further through the stream. It is effectively an isolate iterator that keeps track of its own index regardless of scope. A lexer construct straps together a number of regular expressions and (in the case of Racket's lexer) will smartly consume stream input until a regex is matched. Depending on which one is matched, it will return different things (such as symbols in this example).

The benefit of this approach is that it is linear in its growth pattern, but it would be far more consistent in that linearity than the hash table approach.

Question 4 (15%):

You're given a list of N names and told that you'll need to search the list for a given name M times. The following are suggested:

- using a hash table
- using a heap
- sequential search
- sorting followed by binary search

Discuss the time efficiency of these approaches as the size of N and M vary. Which one would you suggest and why? Would your answer change if you needed to change the list of names by adding and deleting names (say, P times) between searches?

(Again, no code is required here.)

Answer:

The Initial Question:

In the initial question (*n* name *m* times), the properties we are measuring between the different data structures suggested are:

- How long it takes to traverse the data structure (affected by *n*),
- and the behavior of that structure on successive accesses?

The ending note, deleting/inserting new names, added a third metric:

- How does the data structure have to restructure itself if there is a change?

Hash Table:

Hash tables, in this instance would likely perform very well. On average, using a constant-time hashing function, the access time would be constant. There are catches though. If many of the names are similar, you could run into hash collisions (when two key values in a collection produce the same hash). This could easily happen if the hashing function only uses 8 characters of the key and you have to entire 50 “Jonathan"s. Than the access of the hash table would be bound to whatever your hash collision function does. Often these take the flavour of "go *x* number of spots to the right and just try to place it there, if occupied keep going right…” which implies linear access based off of how “far right” these things have to go. This also could complicated successive accesses (our *m* term). If you need to find 40 of those 50 “Jonathan"s, the access would could act more like a linear search (with a much bigger memory footprint).

In considering the circumstance where I also have to delete and add names in between searches, the hash table still should perform reasonably well. Unless I just add names and never delete anything, the likelihood of needed to request another block of memory for the hash table is low. If the situation does only add names though, a hash table’s memory footprint could become rather expensive.

Heaps & Binary Search:

A general heap, with out specifying an particular scheme in which to connect nodes to their parents doesn’t given you much of anything. Whats worse is the need to fix the heap on any insertion or deletion that displaces a parent node. Every time this happens the cost of shifting nodes around will affect successive access times.

If it is a binary heap (I.E. we have sorted the names in some way), than there would be gains in traversal time, but if there are many similar names, the tree traversal will suffer and the reconstruction of the tree on insertion or deletion could be problematic. How to partition the names on branches and the possible behavior given insertion/ deletion could get messy. Sorting the list of names each time will also add whatever that sorting method is to the overhead of access the binary heap/search. If this is something like merge sort, it will have a large memory footprint; if it’s quicksort, you could have some unpredictable behavior if a poor pivot value is chosen; and anything like insertion sort has far too costly.

Sequential Search:

Sequential search has to things going for it: it’s conceptually simple and really predictable. The iteration through the list of names is going to be have a growth pattern based off of the length of the list and the length of the name it is matching. This is result in a *nlm* big-O behavior (*l* being the length of the name searched for). So the behavior isn’t great but it will require the shortest preparation for the search (no restructuring), and the memory footprint is only the search’s stack frame and the collection of names itself.

In terms of the "add or delete” situation, it depends on how the underlying array is implemented that the search is operating on. If it’s an array, the deletion and addition of names will be super costly (in linear time at least). If a name is added or deleted in the middle or beginning of the array, the whole block after the deletion/insertion has to be shifted to compensate. If the list is actually a linked list, appending to the list should be constant. If the deletion or addition is in the middle, it only has a linear cost of traversing the list to the node it wants to operate on. Given this situation, I am not sure I would recommend arrays, but linked lists still make a fine data structure.

My Opinion:

I think it really depends on the context a choice like this comes up in. If I am working with a few names in a system, sequential search would largely be fine and easy enough to implement (I am a frequent user of Python "if *x* in *z* syntax for list member ship). If I was passing around connected information to each name, a hash table makes perfect sense. In fact, if I wasn't in a memory restricting environment, I would probably just use hash tables be default. On average, they offer great access and assignment performance and it forces me to think in terms of membership instead of index. If I new the collection of names well and there was some sane pattern to use, the heap or binary search might be fun options, but I try to abide by the "keep it simple,stupid" (K.I.S.S.) mentality whenever I code. Though these heap data structures may be well studied, hash table and sequential searches are as simple as it gets. In terms of what I would recommend, and in the mind set of least headache later, I would recommend a sequential search in a simple, non-performance critical system, or hash tables otherwise.

Bibliography:

- "Heap (data structure)." *wikipediahttps://en.wikipedia.org/wiki/Heap*(data*structure)

Appendix:

For the masochists in the crowd, here is all the source code from my Plan:

Release Source Code:

index.html:

Markup

```html
<!DOCTYPE html>
<html>

<head>
    <title id="title">Dyslexia</title>
    <link rel="stylesheet" href="css/normalize.css">
    <link rel="stylesheet" href="css/skeleton.css">
    <link rel="stylesheet" href="css/custom_landing.css">
    <script src="js/switch.js"></script>
    <script src="js/text.js"></script>
    <script src="js/shuffle.js"></script>
    <script src="js/audio.js"></script>
</head>

<body>
      <div class="container">  <h1>Release</h1>
        <hr>
        <p>by Logan H. G. Davis</p>

        <p>
            This project is an attempt to recreate and convey the anxieties and
            feelings I have and have had as a person with dyslexia.
        </p>
        <p>
            Black underlined words will progress you through the piece.
        </p>
        <a href="./Dyslexia.html">Start.</a></div>
</body>

</html>
```

Dyslexia.html:

Markup

```html
<!DOCTYPE html>
<html>

<head>
    <title id="title">Dyslexia</title>
    <link rel="stylesheet" href="css/normalize.css">
    <link rel="stylesheet" href="css/skeleton.css">
    <link rel="stylesheet" href="css/custom_landing.css">
    <script src="js/switch.js"></script>
    <script src="js/text.js"></script>
    <script src="js/shuffle.js"></script>
    <script src="js/audio.js"></script>
</head>

<body>
    <video playsinline autoplay muted loop>
    <source src="vid/IMG_0160.MOV">
   </video>
    <div class="container">
        <div id="badfood_gate"></div>

        <script>
        document.getElementById("badfood_gate")
        insertAdjacentHTML("beforeend",badfood);
        activate_audio(audio_for_badfood);
        setSwitches(badfood);
        </script>
    </div>
</body>

</html>
```

custom_landing.css:

```css
.gate{
    text-decoration: underline;
}
video {
    /* adapted from https://codepen.io/tyrus/pen/twpuK */
    position: fixed;
    min-width: 1080px;
    min-height: 1080px;
    z-index: -100;
    background-color: black;

    filter:blur(10px);
      -o-filter:blur(10px);
      -ms-filter:blur(10px);
      -moz-filter:blur(10px);
      -webkit-filter:blur(10px);
}
body{
    background-color: black;
}

.container{
    background-color: rgba(255, 255, 255, 0.5);
    padding: 15px;
}

a{
    color: black;
    text-decoration: underline;
}

a:visited{
    color: black;
}
```

Note all other CSS in *Release* is from SkeletonCSS: http://getskeleton.com

text.js:

```javascript
/*
text.js
by Logan Davis
```

```
Description:
    A storage for modular memoir text sections.
    This code is very case specific to best serve
    text and audio handling for *Release*

4/29/17 | MIT License
*/

function open_gate(gate_id,template_name,audio_name){
    /* inserts a text template based on template_name
     * at tag with id = gate_id.
     *
     * Starts audio object named audio_name.
     * Also starts switches and shuffles based on
     * template_name.
     */
     var prev_height = document.body.scrollHeight;
     window.scrollTo(0,document.body.scrollHeight);
     document.getElementById(gate_id)
     .insertAdjacentHTML("beforeend",template_name);
     document.getElementById(gate_id+"_trigger").onclick = "";
     document.getElementById("post_"+gate_id).innerHTML = "";
     window.scrollTo(0,prev_height+50);
     activate_audio(audio_name);
     setSwitches(template_name);
}
eval1 = `<img src="img/strengths.jpg" height="100%" width="100%"></img>
        <span class="gate" id="seq_gate_from_eval1_trigger"
         onclick='open_gate("seq_gate_from_eval1",sequences,audio_for_sequences);'>
        Sequences.
        </span>
        <div id="seq_gate_from_eval1"></div>
        <div id="post_seq_gate_from_eval1"></div>`
eval2 = `<img src="img/summary1.jpg" height="100%" width="100%">
         <img src="img/summary2.jpg" height="100%" width="100%">
         <img src="img/summary3.jpg" height="100%" width="100%">
         <img src="img/summary4.jpg" height="100%" width="100%">
         <span class="gate" id="python_from_eval2_trigger"
          onclick='open_gate("python_from_eval2",python,audio_for_python);'>
         Python.
         </span>
         <div id="python_from_eval2"></div>
         <div id="post_python_from_eval2"></div>`
eval3 = `<img src="img/momsletter.jpg" height="100%" width="100%">
         <span class="gate" id="death_gate_from_eval3_trigger"
```

```
          onclick='open_gate("death_gate_from_eval3",death,blank);'>
         Death.
         </span>
         <div id="death_gate_from_eval3"></div>
         <div id="post_death_gate_from_eval3"></div>`
eval4 = `<img src="img/youngeval1.jpg" height="100%" width="100%">
         <img src="img/youngeval2.jpg" height="100%" width="100%">
         <img src="img/youngeval3.jpg" height="100%" width="100%">
         <span class="gate" id="busride_gate_from_eval4_trigger"
          onclick='open_gate("busride_gate_from_eval4",busride,audio_for_budride);'>
         Busride.
         </span>
         <div id="busride_gate_from_eval4"></div>
         <div id="post_busride_gate_from_eval4"></div>`

badfood=`<h1>Bad Food:</h1>
    <hr>
    <p>
        So this is what food poisoning feels like. <br>
        I think the volume that I have thrown up is <br>
        greater than the food that made me sick. <br>
        We are totally going to get charged extra for <br>
        this and the technician that needed to open that <br>
        safe. Why did Jake put his wallet in there? <br>
        <span id="badfood_switch1"> Why did I let him put it
        in there</span>? <br>
        <span id="badfood_switch2"> I just want to go home.</span>
         <span class="gate" id="seq_gate_from_food_trigger"
        onclick='open_gate("seq_gate_from_food",sequences,audio_for_sequences);'>
        Fuck this. <br>
    </p>
    <p>
        I know this is supposed to be for Jake and all. <br>
        He has been clean for a year and he wants to celebrate <br>
        it with me and Brad. But why the fuck did he want to go to <br>
        botanical gardens in Delaware? Why Delaware? <br> We
        could have gone to gardens in<span id="badfood_switch3">
        Jersey</span>, but no. <br>
        Fucking Delaware. <br>
    </p>
    <p>
        Never once have I heard someone
        <span class="gate" id="eval4_gate_from_food_trigger"
        onclick='open_gate("eval4_gate_from_food",eval4,audio_for_eval_4);'>
        advocating </span> for a visit, <br>
        move, road trip, or exodus to Delaware. Why would <br>
        Jake choose here? Him and Brad both drove up into Pennsylvania <br>
        tonight because they actually wanted to do something other than <br>
        look at flowers. I however, having ordered the chicken for dinner, <br>
        am now stuck vomiting my brains out in hotel toilet. <br>
    </p>
    <p>
        Now those poor bastards have to drive back down here just to <br>
        drive ride back through to Amish state because I live about <br>
        20 miles from where they are currently shooting off fireworks. <br>
        Or they don't come back for me and then I am really fucked. <br>
        Also, why did they drive out of state to shoot fireworks? They <br>
        are legal here. They BOUGHT them here, why would they not <br>
        realize it's okay to set them off without <span class="gate"
        id="busride_gate_from_food_trigger"
        onclick='open_gate("busride_gate_from_food",busride, audio_for_budride);'>
        driving <br>
        200 miles north?</span><span id="badfood_switch4">
        Why did they leave when I am sick</span>? <br>
    </p>
    <p>
        I know they popped into Philly to go see some strippers, which <br>
        would explain the initial venture northward. Though even I <br>
        could have guessed no strip joint would be open on a Monday. <br>
        They also mentioned something about people  <span class="gate"
        id="eval1_gate_from_food_trigger"
        onclick='open_gate("eval1_gate_from_food",eval1,audio_for_eval_1);'>
        lying in the <br>
        streets.</span> <span id="badfood_switch5"> Did they see a
        dead guy?</span> <br>
    </p>
    <p>
        When are they going to get back? It's 3 in the morn...ugh. <br>
        <span class="gate" id="goodbye_gate_from_food_trigger"
        onclick='open_gate("goodbye_gate_from_food",goodbye,blank);'>
        I should get up, I think I am going to be sick again.</span> <br>
    </p>
    <div id="goodbye_gate_from_food"></div>
    <div id="post_goodbye_gate_from_food">
    <div id="eval1_gate_from_food"></div>
    <div id="post_eval1_gate_from_food">
    <div id="busride_gate_from_food"></div>
    <div id="post_busride_gate_from_food">
    <div id="eval4_gate_from_food"></div>
    <div id="post_eval4_gate_from_food">
    <div id="seq_gate_from_food"></div>
    <div id="post_seq_gate_from_food"></div>
    </div>
    </div>
    </div>
    </div>
    </div>
    </div>
    </div>`
goodbye = `<h1>Goodbye:</h1>
        <hr>
        <p>
            My mother always tells me that when she dropped me off at
            college, I said "It's going to be okay, right?" <br />
            She replied to me in a calm, measured tone "of course." After
            that she started her drive back to Jersey, <br /> she cried for a
            few hours. I nervously wandered around trying to find my trip
            leaders for orientation. <br />
        </p>
        <p>
            The longest paper I had ever written prior to Marlboro was
            about 6 pages. I wrote them once, never proof read <br />
          them, and now I couldn't tell you what a single one
          was about. Most of my high school writing was in the <br /> form of
            single-page short stories that used words from a book of
            insults because our teacher was <br /> passive aggressive
            towards the course evaluators. <br />
        </p>
        <p>
            I had to write papers about medieval history my first semester.
            They were terrible. All of them rambled on, were <br />
          full of typos, and had no particular claim. I failed that
          semester's writing portfolio with a 2.25 out of 4. <br />
        </p>
        <p>
            There were people who tried and help with my time management,
            but I had no one to go to for my papers (or at least that <br/>
          was how I felt, given that I knew how far behind I was).

          <span class="gate" id="csharp_gate_from_goodbye_trigger"
          onclick='open_gate("csharp_gate_from_goodbye",csharp,audio_for_csharp);'>
          It took me a fair deal of time and a huge amount of time and
          stress to get somewhere
          </span> with writing. <br />
        </p>
        <p>
            Anxiety was also a factor. I was surrounded by new people in a
            place that I felt I could have been kicked out <br /> in
            a moments notice.

            <span class="gate" id="python_from_goodbye_trigger"
             onclick='open_gate("python_from_goodbye",python, audio_for_python);'>
             I was failing introductory courses
            </span> while my classmate was getting his second book published. <br />
        </p>
        <p>
            I remember crying to my father over the phone in the middle of a panic
            attack. I also recall when I was talking <br />
          my mother about the fact that the meds I was given for anxiety were
          heavily affecting my memory. The day <br /> after,
            I could be found vomiting in a ditch along South Road after drinking
            too much. <br />
        </p>
        <p>
            I never use to drink when I was young. I took too many medications.
            The doctors always told me my liver was <br /> in a
            fragile state. Given how often I had blood tests, I believed them.
            I was able to stop taking medication <br /> though.
            Not so much by choice, but because I would forget to take it in
            my all-night-writing-binges. <br /> Before I knew it,
            I hadn't taken anything in a month. <br />
        </p>
        <p>
            I didn't feel any different. Though the writing I produced
            became clearer. <br />
        </p>
        <p>
            My mother was mostly right.
          <span class="gate" id="grandparents_gate_from_goodbye_trigger"
          onclick='open_gate("grandparents_gate_from_goodbye",grandparents,
          audio_for_grandparents);'>
          It is okay now, but it wasn't for a time. </span> The late
          nights of responding to Town Meeting <br>
          emails; the stress of possibly losing financial aid; and nights
          when I missed dinner and had to <br />
          write until 5AM on a collapsing stomach were not okay. But
          this, right now, is. <br />
        </p>

            <div id="grandparents_gate_from_goodbye"></div>
            <div id="post_grandparents_gate_from_goodbye">
            <div id="python_from_goodbye"></div>
            <div id="post_python_from_goodbye">
            <div id="csharp_gate_from_goodbye"></div>
            <div id="post_csharp_gate_from_goodbye">
            </div>
            </div>
            </div>`
busride = `<h1>Bus Ride:</h1>
    <hr>
    <p>
    Given how I'm leaning against this window, I hope we don't <br>
    hit a speed bump.<span id="bus_switch1"> I am always afraid my head will
    do through the glass</span>.
    <br> I have spent too much time on this bus. <br>
    </p>
    <p>
    One hour drives, each way, one hundred and eighty days a <br>
    year for nine years; I have spent 3960 hours in this bus. <br>
    The ride is about thirty five miles each way. Which means I <br>
    have traveled at least 138600 miles over that period of time. <br>
    <span id="bus_switch2"> I feel gross thinking about it
    that way.</span> <br>
    So many hours spent. <br>
    </p>
    <p>
    I think the ride would be better if there was someone <br>
    else on bored other than the bus driver. Most kids rides <br>
    are an hour long, but at least they have someone to talk to. <br>
    <span id="bus_switch3"> I just have to make friends with the
    rotating drivers.</span> <br>
    Why can't I live closer to my friends? <br>
    </p>
    <p>

    <span class="gate" id="seq_gate_from_bus_trigger"
     onclick='open_gate("seq_gate_from_bus",sequences,audio_for_sequences);'>
     ... <br>

    </p>
    <p>
    Oh look, another accident. Fuck, that person's car on fire. <br>
    Where are the police? Where is anyone? Is that car just <br>
    on fire on the side of the road? It looks like it has been <br>
    burning for awhile. <br>
    </p>
    <p>
    At least people are freaked out enough by it that there don't <br>
    seem to be many rubber-neckers...Other than me, of course. <br>
    There have to be two or three accidents a day on this highway. <br>
    <span id="bus_switch4"> It's a fucking miracle that I haven't
    been in more.</span> <br>
    My neck still isn't right from that last one. <br>
    </p>
    <div id="seq_gate_from_bus"></div>
    <div id="post_seq_gate_from_bus">
    </div>`

sequences = `<h1>Sequences</h1>
            <hr>
            <p>
            I just want to go to sleep.
            <span class="gate" id="sound_from_seq_trigger"
             onclick='open_gate("sound_from_seq",sound,audio_for_sound);'>
            My head hurts.<br/>
             </span>
            </p>
            <p>
                <span id="seq_switch1">We have to learn cursive</span>.
                The teacher says it<br/>
                is something everyone needs to learn, it's<br/>
                just what you do. I can't read cursive...<br/>
                well, I can't properly read print either,<br/>
                but I really can't cursive. I have spent the last<br/>
                thirty minutes writing fifty lower case "a"s. Then<br/>
                I get to spend thirty more minutes writing fifty lower<br/>
                case "b"s...then I get to do it with the next 24 letters.<br/>
                Oh, wait, then it's uppercase. I am
                <span id="seq_switch2">going to be up all<br/>
              night doing</span> this.<br/>
            </p>
            <p>
                <span id="seq_switch3">Who came up with the cursive letter
                for uppercase "G"s? What
              is wrong with them?</span><br/>
            </p>
            <p>
                Why am I not learning how to read? Why <span id="seq_switch4">
                am I spending</span> time learning<br/>
                an <span class="gate" id="csharp_from_seq_trigger"
                onclick='open_gate("csharp_from_seq",csharp, audio_for_csharp);'>
                entirely new and even more confusing </span> <br>
             way to write a language that<br/>
                I already struggle with? <span id="seq_switch5">Why are
                they</span> doing this?<br/>
            </p>

            <p>
                <span class="gate" id="goodbye_gate_from_seq_trigger"
                 onclick='open_gate("goodbye_gate_from_seq",goodbye,blank)'>
                <span id="seq_switch6">I want to cry.</span>
                </span>
            </p>
            <div id="csharp_from_seq"></div>
            <div id="post_csharp_from_seq">
            <div id="goodbye_gate_from_seq"></div>
            <div id="post_goodbye_gate_from_seq">
            <div id="sound_from_seq"></div>
            <div id="post_sound_from_seq">
            </div>
            </div>
            </div>`

grandparents = `<h1>Pops and Grandma:</h1>
      <hr>
      <p>
      <span id="grandparents_switch1"> I don't want to be here.</span><br>
      The family is all gathered around at this old inn again, eating <br>
      and laughing. <span id="grandparents_switch2"> I guess we wouldn't all
      be here if they were around.</span><br>
      I don't think we will ever come back. <br>
      </p><p>
      <span id="grandparents_switch3"> Now pops and grandma both overlook
      the Delaware.</span><br>
      <span class="gate" id="death_gate_from_grandparents_trigger"
       onclick='open_gate("death_gate_from_grandparents",death,blank)'>
      One died as I started school, the other dies as I finish it. </span>
      I wonder <br>
      what they would say if they were here. They would probably demand <br>
      to know why my food hasn't been served yet. I know the tavern is <br>
      busy, but everyone else's food has been out for an hour. <br>
      <span id="grandparents_switch4"> They must have forgotten about it.
      </span><br>
      </p><p>
      Actually, I don't know what grandma would say. <br>
      <span id="switch5"> I don't have a good memory of her.</span><br>
      <span class="gate" id="eval3_gate_from_grandparents_trigger"
       onclick='open_gate("eval3_gate_from_grandparents",eval3,audio_for_eval_3);'>
      I have heard enough </span> stories that they are all I really know
        about her. <br>
      All I can clearly recall is her in a red bandana on her head, sitting in <br>
      the Old Farms kitchen. She died not too long after that. <br>
      </p><p>
      Pops, on the other hand, I have numerous memories with. <br>
      <span id="grandparents_switch6"> Watching him build a bar and how proud
      he was when he finished. </span> <br>
      <span id="grandparents_switch7"> All of those Christmases and
        Thanksgivings. </span> <br>
      <span id="grandparents_switch8"> All of the symphonies we went to
        together. </span> <br>
      I got the chance to know him. <br>

      <div id="eval3_gate_from_grandparents"></div>
      <div id="post_eval3_gate_from_grandparents">
      <div id="death_gate_from_grandparents"></div>
      <div id="post_death_gate_from_grandparents">

      </div>`
death = ` <h1>Death:</h1>
        <hr>
        </p>
        <p>
            I have never seen death. I have seen someone dying and I have <br>
            seen someone dead, but never the moment in between the <br>
            two. What is described as a "look in their eyes," or a <br>
            "sound of a resigned sigh" that tells you that they realize <br>
            they are about to die is something I have never witnessed. <br>
        </p>
        <p>
            The earliest person I can remember that I saw dying was <br>
            my grandmother. I was too young to know what the <br>
            bandanna on her head meant, but I knew it wasn't good. <br>
            She is also the earliest person I saw dead, I believe. I don't <br>
            remember much from the few years following her death. <br>
        </p>
        <p>
            What I do remember is that I typically have a delayed <br>
            response when I find out someone has died. For a day or <br>
            two after the news I am sad, but after a dat or two I move <br>
            into a deeper stage of mourning. I'll fall into bouts of <br>
            crying for a few minutes and then be fine for the next few <br>
            hours. There are only two deaths I can remember crying <br>
            immediately after: my aforementioned grandmother and <br>
            her husband, my grandfather. <br>
        </p>
        <p>
            My grandfathers death was particularly strange; it <br>
            was the day before my other grandmother's birthday. <br>
            Though a cried when I heard he died, I sobbed when I <br>
            wished my grandmother a happy birthday the next day. <br>
            She cried with me. <br>
        </p>
        <p>
            My mother's side of the family talks about death plainly. <br>
            We talk about those who passed. We talk about those who <br>
            are passing. We talk about our own death as well. It's <br>
            never sad. Typically it has a gallows-humor tone to it. To <br>
            my fathers family death is a topic to be discussed <br>
            amongst the men behind a closed door. Never should you <br>
            speak of it at near the dinner table. <br>
        </p>
        <p>
            I have never cared to give death much of a thought past <br>
            the brief, honest evaluation of possibility before I do <br>
            something stupid. I have talked about my death in front <br>
            of family members to mixed response.

            <span class="gate" id="phil_from_death_trigger"
             onclick='open_gate("phil_from_death",phil,blank)'>
            It is because I am <br> too young </span>
            , not because of what I say, which surprises me. <br>
            What reasonable thing could I say about it? I know <br>
            nothing on the subject. I have never seen death. <br>

            <div id="phil_from_death"></div>
            <div id="post_phil_from_death">
        </p>
        <p>`
phil = ` <h1>"Phillip is filling up his cup:"</h1>
        <hr>
    <div id='shuffle_phil_1'>
    <p>You walked in and wouldn't introduce yourself. Neck craned forward
    and hands uncomfortably pressed
    against your sides like you were mimicking a tall bird. We couldn't stop
    staring at your bowl cut. I am sorry.</p>
```

```
<p>You wanted us to call you Phil and you wanted us to laugh. Never once did
we not laugh at a joke of
yours, unless you were laughing to much at it to get to the punch line. You
seemed addicted to
laughter, both yours and others.</div> I will always remember, clear as day,
when you walked to the back
of a quiet classroom and poured yourself water:</p>

    <p>"Phillip is filling up his cup."</p>

<div id='shuffle_phil_2'>
<p>You laughed hard enough that you needed an inhale. We looked at you
like you were a freak. You were a freak.
We were all freaks. Eli couldn't focus, Billy threw chairs, Aaron
couldn't add, I couldn't read, and
you tried to kill yourself. We didn't know that until later. </p>

<p>You left for district. You left us in Center. We didn't talk for
years. I got updates periodically through
Billy, but he was too strung out to recall what clean clothes looked
like most of the time, or maybe his
parents missed a water bill. I don't know. But he left soon after you
did, so I lost all contact with you.
I am sorry. </p>

<p>What happened? How did you choose to become an engineer? Where did
you go to school? Why did you go to that
specific school? I don't know. I won't know. Why should I know? You
left and I stayed. You made it out.
You made it out. You made it out.</p>

<p>What the fuck does that even mean? Austin went to prison, Nathan
is peddling shit, Jake went in and out
of programs for addiction, and you died. Why did we want to get out?</p>

<p>You were told to get out enough by teachers. You would interrupt a
class with a joke or do something stupid
and Mr Z. would yell "PHIL." You would turn to him with a blank stare
of confusion and then throw another
pencil into a spinning ceiling fan because you were bored.</p>

<p>You were always bored. You never had enough work to do. Anything we
were assigned you would have done before
teacher finished explaining the prompt, so you often did the wrong thing.
```

```
    You didn't redo it though, you
    already did work.</p> </div>


    <p>My parents still have the picture of us from seventh grade. The picture
    where you stepped forward right before
    it was taken. You did it to look taller than the rest of us; you just
    looked out of place.  Sometimes people
    say "I can't picture them as an adult" when young people die. I think
    you were always out of place.
    I am sorry.</p>`

csharp = `<h1>C#:</h1>
      <hr>
      <p>
      I guess it is time to get back to work. <br>
      <span id="csharp_switch1"> Mr O. wants me to copy more code </span>. <br>
      <span class="gate" id="sound_from_csharp_trigger"
      onclick='open_gate("sound_from_csharp",sound, audio_for_sound);'>
      <span id="csharp_switch2"> I want to enjoy this, but I don't <br>
      understand what is happening.</span></span>
      <span class="gate" id="color_from_csharp_trigger"
       onclick='open_gate("color_from_csharp",color,blank);'>
      I am not sure </span> how <br>
      I am supposed to make this into a class <br>
      for them to teach, but maybe <span class="gate" id="eval2_from_csharp_trigger"
      onclick='open_gate("eval2_from_csharp",eval2,audio_for_eval_2);'> that
      is the reason they are <br>
      having me do it</span>:<span id="csharp_switch3"> they don't know
      how to either</span>. <br>
      </p>
      <p>
      Let me look at the example code: <br>
      </p>
      <p id="shuffle_csharp">
      var target : Transform; //the enemy's target <br>
      var moveSpeed = 3; //move speed <br>
      var rotationSpeed = 3; //speed of turning <br>
      <br>
      var myTransform : Transform; //current transform data of
      this enemy <br>
      <br>
      function Awake() <br>
      { <br>
          myTransform = transform; //cache transform data for easy
```

```
        access/performance <br>
    } <br>
    <br>
    function Start() <br>
    { <br>
        target = GameObject.FindWithTag("Player").transform;
        //target the player <br>
    }<br>

    function Update () { <br>
        //rotate to look at the player <br>
        myTransform.rotation = Quaternion.Slerp(myTransform.rotation, <br>
        Quaternion.LookRotation(target.position - myTransform.position),
         rotationSpeed*Time.deltaTime); <br>
    <br>
        //move towards the player <br>
        myTransform.position += myTransform.forward * moveSpeed *
        Time.deltaTime; <br>
    } <br>
    </p>
     <p>
    Yup, about as <span id="csharp_switch4"> understandable </span>as ever. <br>
    Well, lets put it in the game and see what it does. </p><br>
     <p>
    ... <br>
     </p><p>
    <span id="csharp_switch5"> Oh, the zombies are chasing me</span>. <br>
    This was only, like, 10 lines of code. <br>
    <span id="csharp_switch6"> How</span> are they doing that? <br>
    <span id="csharp_switch7"> I am really lost now. </span>  <br>
    <span id="csharp_switch8"> How does the code know who
    the playeris? </span><br>

    <span class="gate" id="python_from_csharp_trigger"
     onclick='open_gate("python_from_csharp",python,audio_for_python);'>
    How do I study this?
    </span>

    <div id="python_from_csharp"></div>
    <div id="post_python_from_csharp">
    <div id="color_from_csharp"></div>
    <div id="post_color_from_csharp">
    <div id="eval2_from_csharp"></div>
    <div id="post_eval2_from_csharp">
    <div id="sound_from_csharp"></div>
```

```
       <div id="post_sound_from_csharp">
       </div>
       </div>
       </div>`
python = `<h1>Python:</h1>
        <hr>
        <p> It's one in the morning on a Tuesday. I have class
        tomorrow and I cannot even finish the first assignment.
        All my program has to do is display "Hello, world." Instead
        I am staring at black screen with a small sentence in
        harsh white text:</p>

       <p>"What the fuck did I get myself into..." </p>

       <p>
       <span class="gate" id="color_from_python_trigger"
        onclick='open_gate("color_from_python",color,blank);'>
         I don't get it.
        </span>
        I am supposed to just write "print" before "Hello, World." But
        how do I run the program?
        Why does that work? <span id="python_switch_1">I really spent
        eleven years reading nonsense words to figure out how to spell
        to get stuck on this.</span></p>


        <p id="python_switch_2"> Maybe they were right, I am going to be a
        gas station attendant. They told me otherwise, but I know.</p>

        <p>No, no, no. <span id="python_switch_3">I can figure this out
        </span>. Let me look at the class notes.</p>
        <p>
        <div id="python_shuffle_1"
        meta="from: http://cs.marlboro.edu/courses
        /fall2013/python/notes/getting_started">
        Here's the "hello world" program in python : <br>
        # This file is hello.py - a first short program in python. <br>
        print "Hello world" <br>
        From a terminal program (the $ is the prompt waiting for you
        to type), in the same folder as the file (see below for more) <br>
        $ python hello.py <br>
        Hello world! <br>
        You should have this working this week. Again - you need help,
        ask, don't wait and get behind. <br>
        </div>
        </p>
        <p>I am absolutely fucked.</p>

        <p id="python_switch_4">I mean, I know I didn't do much reading
        in highschool nor did I do very much math, but this should be simple.
        Maybe this is why Mr. O said he couldn't teach me programming, he
        never was able to learn.</p>

        <p>What am I going to tell <span id="python_switch_5">Jim</span>.
        I just got to college and I am already hitting a wall.</p>

        <p id="python_switch_6">I am no better than I was.</p>

        <p>Let me try this again: <br>
            >>> print "Hello world."<br>
            Hello, world <br>
        </p>

        <span class="gate" id="joy_from_python_trigger"
         onclick='open_gate("joy_from_python",joy,blank);'>
        <p id="python_switch_7">Wait, it worked...it actually worked.
        Let me try it again.</p>
        </span>

        <p>
        <div id="python_shuffle_2">
            $ Print "Hello world." <br>
            NameError: name 'Print' is not defined <br>
        </div>
        </p>

        <p id="python_switch_8">Fuck, why didn't that work.</p>



        <div id="joy_from_python"></div>
        <div id="post_joy_from_python">
        <div id="color_from_python"></div>
        <div id="post_color_from_python">
        </div>
        </div>`
joy = ` <h1>Joy:</h1>
    <hr>
    <p>
    It was once told to me, when I studied music, <br>
```

```
the feeling of conducting an orchestra. The <br>
adrenaline rush when, with the flick of a wrist <br>
brass bellows, strings hum, drums crash, and <br>
woodwinds whistle.
<span class="gate" id="sound_from_joy_trigger"
 onclick='open_gate("sound_from_joy",sound,audio_for_sound);'>
All on time. All in harmony. <br>
</span>
</p>
<p>
When writing a program, there can be what feels <br>
like an eternity of fixing, tuning, and rewriting code. <br>
It is like some old comedy skit where someone plugs <br>
a hole in a leaky roof to only find two more leaks created <br>
because of the fix. Sometimes the answer is to just rebuild <br>
the entire roof. Sometimes you just keep tacking gum to <br>
the ceiling hoping that it doesn't cave in on you. <br>
</p>
<p>
After this period of debugging though, there is a moment. <br>
It can be brief, it can be half founded, and it can lead you <br>
into a whole new stage of debugging, but it is there. <br>
It's when something works. <br>
</p>
<p>
Your debugging tools are running. You just finished a <br>
new line of code that hopefully stops whatever the hell <br>
was throwing some system-crashing exception. Yet <br>
another step in what seems to be a never ending hike <br>
through stack trace after stack trace, you hit enter and <br>
you watch your code run. <br>
</p>
<p>
There is something different this time. No 78-line trace <br>
of "function this" handed context over to "function that". <br>
It is just a blank screen. Stress begins to make its presence <br>
known by the sound of a held breath. It's silent, but just for <br>
a moment. Finally, there it is: the output you expected. <br>
</p>
<p>
It could just be some text on the screen; a series of data points; <br>
rendered images; an active server connection; or some results <br>
after folding proteins. It doesn't matter. It is there, you wanted it <br>
to be there, and you figure out how to make the computer put it <br>
there. <br>
```

```
    </p>
    <p>
    For a brief moment you have hundreds of billions of transistors, <br>
    just nanometers in size, processing bits at a rate between 1 and 3 <br>
    billions times a second under your control. You knew how to tell a <br>
    machine that can only subtract by using strange tricks in addition <br>
    how to carry out what could be some of the complex tasks imaginable. <br>
    You computed numbers is mere seconds that,
    if considered as days on a calender, <br>
    would span far past your lifetime. It didn't
    require years of hardened practice <br>
    or many thousands of mathematicians at your
    disposal. It took a text file <br>
    and a tap of a key. </p>

    <div id="sound_from_joy"></div>
    <div id="post_sound_from_joy">
    </div>`
sound = `<h1>Sound:</h1>
      <hr>
      <p>
      I don't <span id="sound_switch_1">get
      </span> music. It is <span
       id="sound_switch_2">overwhelming</span>.
      </p>
      <p>
      The worst is during holidays. With all of my <br />
      loud aunts and uncles speaking, I already get <br />
      a head ache, but music just adds mud to the air. <br />
      I can't <span id="sound_switch_3">think straight</span>. <br />
      </p>
      <p>
      It feels a lot like reading. I look at a page <br />
      and <span id="shuffle_sound">the text starts to
      move</span>, I cannot <br />
      tell the words apart. Sound is the same way. I <br />
      <span id="sound_switch_4">get lost in</span> the wall of noise.
      </p>
      <p>
      My sister likes music, she also likes the city. <br />
      I get too flustered by it, though I like skyscrapers. <br />
      I want to be an <span id="sound_switch_5">
      architect</span> when I am <br />
      older. Buildings make sense, noise doesn't. <br />
      </p>
```

```
       <p>
       Architecture is a system, there are parts to put together <br />
       and if they go up in the wrong order, <br />
       <span id="sound_switch_6">it doesn't work</span>. Sound can be in <br />
       any order you want it to be. A <span id="switch7">car horn</span> <br />
       here and there, some <span id="sound_switch_8">
       yelling</span> in the distance <br />
       sprinkled throughout. Sound doesn't make sense. <br />
       </p>
       <p>
       Music is just sound that has been <span id="sound_switch_9">
       reigned</span> in. <br />
       It can sound like it is about to end but then <br />
       it just gets louder. <br />
       </p>
       <p>
       I can turn away from text. <span id="sound_switch_10">Music</span> <br />
       is surrounding <span id="sound_switch_11">you</span> all the time. <br />
       </p>`

color = ` <h1>Color:</h1>
         <hr>
         <p>
         My first memory of my colorblindness was in preschool. <br>
         We were playing a game to learn the names of different colors <br>
         by sorting bears into piles. The bears were red, blue, green, yellow, <br>
         and purple. My kind of colorblindness allows me to see red and green <br>
         if they are the only colors on an object, but if they are mixed with <br>
         anything else (such as blue, in the case of purple) I see the <br>
         secondary color only. <br>
         </p>
         <p>
         I made my piles, the four I could distinguish (red, blue, green, <br>
         and yellow), and I presented it to my teacher. She told me it <br>
         was wrong and to try it again. I went back and couldn't <br>
         figure out what I had done wrong. After repeatedly saying it <br>
         was wrong, the teacher helped me. She separated my blue pile into <br>
         two groups. For the life of me I could not figure out what she was <br>
         doing. She said "these ones are blue and these ones are purple." <br>
         I thought  she was lying and making fun of me so I cried. I was <br>
         roughly three years old. <br>
         </p>
         <p>
         Colorblindness was been one of my two longest standing sore points <br>
         (only comparable to someone making fun of when I stutter).  I think it <br>
```

```
is because the jokes made about it are jokes I can never be "in on." <br>
When someone makes fun of dyslexia, I can get it because I have the <br>
ability to read correctly; I know the contrast that creates the <br>
joke. If someone makes fun of me not being able to see green, I can <br>
never find the humor. Color is akin to a mathematical definition to <br>
me: I know purple is blue mixed with red, but it is not <br>
something that naturally occurs to me, only an idea. Since it is <br>
impossible to just imagine a new color, it just lives in the words of <br>
its definition. <br>
</p>
<p>
Given my inability to really engage with color, it was never a large <br>
part of my process. I never really liked coloring books once I found <br>
out I was colorblind. I never really used things like sticky notes for <br>
organization. My memories are never really in color unless an items <br>
color has great importance to the event I am remembering. <br>
</p>
<p>
One of the more recent problems I have run into is syntax highlighting <br>
when writing code. Text editors for programmers often color text <br>
differently base on what they types. Variables may be one color while <br>
function definitions might be another. This highlighting has no effect <br>
on my and I often don't turn it on if it is off by default. When <br>
programming with other people it can become cumbersome for them to <br>
look at code on my computer because it becomes this wall of black and <br>
white colors. There is absolutely no visual assistance for them to <br>
break up the syntax. Considering the inverse, some popular color <br>
schemes make it entirely impossible for me to see certain kinds of <br>
words. If someone is using a blue background (common in the very <br>
popular "cobalt" editor scheme) and some word is colored purple, there <br>
is no chance in hell I will be able to see it. <br>
</p>
<p>
I came to acquire Enchroma glasses within the last year. They are <br>
supposed to assist in correcting colorblindness. They won't totally fix <br>
it, they lessen the blindness. They work. I got to see purple for the <br>
first time in my life when a friend googled it and turned the computer <br>
screen to me as I put them on for the first time. It was totally <br>
overwhelming. I have only wore them a handful of times. It isn't just <br>
like correcting a lisp or stutter. It isn't like improving your <br>
ability to consistently identify words. It is the process of unveiling <br>
something that was

<span class="gate" id="dreams_from_color_trigger"
 onclick='open_gate("dreams_from_color",dreams,blank);'>
```

```
        totally out of my grasp
         </span>

        for the majority of my <br>
        life. Taking things you have known for years and making you aware of <br>
        an aspect you have been so plainly and fundamentally missing. I spent <br>
        four or five hours just staring at things in my room one day. Photos <br>
        of family, small wooden boxes, and notes from my god daughter looked <br>
        alien in through those lenses. They were covered in colors I had never <br>
        seen before. They didn't feel like they were mine. <br>
        </p>

        <p>
        I don't wear the glasses in my house anymore. <br>
        </p>

        <div id="dreams_from_color"></div>
        <div id="post_dreams_from_color">
        </div>`
dreams = `<h1> Dreams: </h1>
        <hr>
        <p>
        Most of my dreams are fairly average.

        <span class="gate" id="dreamsII_from_dreams_trigger"
         onclick='open_gate("dreamsII_from_dreams",dreamsII,audio_for_dreams2);'>
        Not average
        </span>

        as in the average <br>
        depiction of a dream (flying through the air of some visual mishmash <br>
        of the last book I read, TV show I watched, and week of my life).
        Average <br>
        as in they play out like an average day.

         <span class="gate" id="gender_from_dreams_trigger"
          onclick='open_gate("gender_from_dreams",gender,blank);'>
        I wake up in the dream
         </span>

        , I go get a <br>
        drink, and then I go back to my bed
        and fall asleep. The only difference <br>
        is that some room will have a single
         weird corner (maybe it has some <br>
        kind of inexplicable geometry, or it's
        on fire or some other thing), or <br>
        everyone will have lost their hair, but
        never anything major (like there is <br>
        no gravity). They are minor differences,
        but other than that, the dream just <br>
        feels like an average day. Every once and
        a while I have a different kind of <br>
        dream though, always similar but slightly
         different. I have had this series <br>
        of dreams since I was in elementary school. <br>
        </p>
        <p>
        I couldn't tell you how each starts, because
        I can never remember a beginning <br>
        to them. The first consistent memory I have
        from each is walking under an <br>
        overpass. If you have ever been to New Jersey,
        the roads around me resemble <br>
        to spaghetti of ramps connecting I-78 and 287
        where they intersect. There are <br>
        fences, poorly held up, guarding small
        construction vehicles telling me that a <br>
        night crew is working on a nearby section
        of pavement. It couldn't be this <br>
        junction though, there aren't enough
        cars to be Jersey. <br>
        </p>
        <p>
        At this point the dreams all take a similar path:
        I am trying to get to a city, I <br>
        need to get something, I meet someone on the way.
        Some times these people <br>
        are in cars and give me a ride, other times
        they are other walkers. It is always <br>
        day time when I start, then it is night, but
        by the end the sun is up again. <br>
        </p>
        <p>
        I talk to the person I meet. I don't know their
        name, I cannot recall their face, but <br>
        I know they did all the talking. Sometimes they
        retell stories that I know are actually <br>
        mine, but I don't interrupt them to say so (either
        out of respect for giving me a ride <br>
```

```
or not wanting to drive welcome company away).
Other times they tell me stories I <br>
have never heard before, I always feel awkward
being so personal with a complete <br>
stranger. Before I know it we are on a bridge
with an upward incline. They tell me that <br>
it is my stop and I get out. It is always night
by this point. I see the city back-lighting <br>
a larger bridge being built to the right of
the one I was on. The scene is not terrible <br>
dissimilar from the current state of the Tappan Zee. <br>
</p>
<p>
I have to backtrack off the bridge at this
point because I cannot walk across it. Under <br>
the bridge is an old rail yard. I walk while
listening the engines and tires above. By this  <br>
point the city is in view. I don't know its
name. I don't know why I am going there. I <br>
don't how much longer it will take to it. <br>
</p>
<p>
I arrive at the cities edge; it's morning
again. Now I am standing on a platform of a New <br>
York-esque dock, with water between me and
the city. It is not New York though, the <br>
buildings aren't high enough. The overpass
curves into the city, leaving support pillars <br>
on both sides on either side of the docks inlet.
The only way I can cross is by a few <br>
rusted pipes or a log. There are people at the
other side yelling for me to get over <br>
before someone sees. I get on the thin beam to
realize that I am about 30 feet above <br>
the water. I hate heights. Someone always
kicks the beam and I always go down. <br>
</p>
<p>
The water is cold and smells of filth. I barely
make my way to a ladder on the other side. I <br>
manage to get on to the other side of the dock,
soaking wet, and the person who kicked <br>
me over is gone. Then the dream ends. It always
ends there. I never get to enter the city. <br>
I never remember why I am there. <br>
```

```
        </p>


        <div id="gender_from_dreams"></div>
        <div id="post_gender_from_dreams">
        <div id="dreamsII_from_dreams"></div>
        <div id="post_dreamsII_from_dreams">
        </div>
        </div>`
gender = ` <h1>Gender:</h1>
        <hr>
        <p>
        I wish I were a woman. <br>
        Every day that I look at my body I think that. <br>
        I have thought that since I was a kid, but I never told <br>
        anyone because I already felt like my connection with <br>
        those around me was strained enough. I didn't want to <br>
        stress those ties anymore than they already were. <br>
        </p>
        <p>
        I played with barbies as a kid. My mothers father <br>
        wasn't very happy about that fact. A few other people <br>
        in my non-immediate family took objection to it as well. <br>
        I didn't know about it then. Or maybe I just forgot. <br>
        </p>
        <p>
        My mother wondered if I was asexual growing up. <br>
        I dated someone briefly in my freshman year of <br>
        highschool. I could never really talk to girls in a <br>
        calm manner. Not just because of the awkwardness <br>
        of teenage years (though I am sure that didn't help), <br>
        but because I was made so keenly aware of what I was <br>
        not when I spoke to them: a girl. What I felt <br>
        during those years is what I can only <br>
        describe as a lighting-depression <br>
        </p>
        <p>

        <span class="gate" id="dreamsII_from_gender_trigger"
         onclick='open_gate("dreamsII_from_gender",dreamsII,audio_for_dreams2);'>
        I didn't
        </span>
         wallowed in deep depression for weeks or <br>
        months like what my bipolar did to me. But when I looked <br>
        in a mirror, began to undress, or sat in a doctors office, <br>
        it would strike. <br>
        </p>
        <p>
        The creaking of my knees feel louder. <br>
        I would get a cold snap up my lower back. My hands <br>
        would feel heavier. My skin looked blotchier. <br>
        I would be depressed until something happened to take <br>
        my mind to other things. <br>
        </p>
        <p>
        I still think about it frequently, (not) being a woman. <br>
        Though I dressed "clash" for a little while, that didn't <br>
        feel right to me. And my build and my height are far <br>
        too masculine to pull off androgyny. <br>
        </p>
        <p>
        So I dress in clothes that fit me and are comfortable, <br>
        as to try and bring as little (self) attention to my <br>
        body as possible. I still get into funks about it though. <br>
        </p>
        <p>
        I once told my mom that if I had the choice to be born <br>
        again but as a woman, I would take it. I still stand by that  <br>
        statement. <br>
        </p>

        <div id="dreamsII_from_gender"></div>
        <div id="post_dreamsII_from_gender">
        </div>`
dreamsII = ` <h1> Dreams II: </h1>
        <hr>
        </p>
        <p>
        Why am I having these dreams. I never
        used to have these kinds of  <br>
        <span id="dreamsII_switch_1"> unusual
        </span> dreams before. I mean, <br>
        what even was that? <br>
        </p>
        <p>
        <span id="dreamsII_switch_2"> I was
        standing on a patio, only looking to my right. I couldn't <br>
        seem to turn my head. There as grass just beyond my feet heavily <br>
        dotted with yellow pansy flowers. They rested on a downward slope. <br>
        At the bottom of the small hill was another patio covered in wrought <br>
        iron table frames with wood tops. Above each table was a yellow <br>
        umbrella. From my viewpoint, the umbrellas were about the same size <br>
        as the adjacent pansies. It all washed into a single, colorful <br>
        movement the hills that lay even further down, creating these flumes <br>
        with the small rivers at their bases. I blinked and it was over. </span><br>
        </p>
        <p>
        I have been losing so much sleep. I
        think I got<span id="dreamsII_switch_3"> 12</span> hours this week  <br>
        including tonight. How am I going to
        code today if I can barely focus <br>
        enough to sleep?<span id="dreamsII_switch_4">
        Why has this started happening now?</span> <br>
        Whatever, I am too cold to get out of these covers anyway, I will fall <br>
        back asleep at some point. I wonder if this is because I wrote about <br>
        my dreams being so mundane. I guess that piece is a lie now. <br>
        Maybe this is my brain saying "I can throw you into some  <br>
        Alice and Wonderland scenario if I want to...I just don't always feel  <br>
        like it." Maybe it's <span id="dreamsII_switch_5">
        my depression coming back</span>. <br>
        Whatever, as long as I get those bugs fixed <br>
        tomorrow, I will be fine. <br>
        </p>
        <p>
        ... <br>
        </p>
        <p>
        Maybe it is because of all the cars and city noises. I have never <br>
        really had to deal with such loud nights. Typically it's <br>
        silent, but those guys at the police station were indefinitely <br>
        practicing Amazing Grace on those bagpipes when I <br>
        first got in bed. The motorcycles roaring by probably aren't helping <br>
        either. <br>
        </p>
        <p>
        It doesn't matter, nothing I can do about it.  <br>
        Just got to<span id="dreamsII_switch_6"> go back to sleep</span>. <br>
        </p>`
```

shuffle.js:

```javascript
/*
shuffle.js
by Logan Davis
```

```
Description:
    Handles shuffling of given text
    in an HTML page.

4/29/17 | MIT License
*/

function isBlank(item){
    /* Checks to see if an item isn't a blank string,
     * if not, returns True.
     */
    return item != "" ;
}

function shuffleWordsFromText(text){
    /* returns a string of a section of text with the
     * shuffleWord function mapped to each word.
     */
    return text.split(" ").filter(isBlank).map(shuffleWord)
                                          .join(" ");
}

function shuffleWord(word){
    /* Has a 2% chance of shuffling a
     * word unless it is <br>|<p>|</p>
     *
     * If the shuffling happens, the new word
     * is returned, otherwise, the old one
     * is returned.
     *
     */
    // adapted from http://stackoverflow.com/
    // questions/6274339/how-can-i-shuffle-an-array-in-javascript

    if((Math.random() > 0.98) &&
       (word.match("<br>|<p>|</p>") == null)){
        var j, x, i;
        var a = word.split("");
        for (i = a.length; i; i--) {
            j = Math.floor(Math.random() * i);
            x = a[i - 1];
            a[i - 1] = a[j];
            a[j] = x;
        }
        return a.join("")
    } else {
        return word
    }
}

//Sets a special interval for the title
setInterval(function(){
    document
    .getElementById("title")
    .innerHTML = shuffleWordsFromText(document
                            .getElementById("title").innerHTML)
},100)
```

switch.js:

```JavaScript
/*
switch.js
by Logan Davis

Description:
    Defines specific functions for text switching
    in *Release* while also handling all switch/shuffle
    interval setting.

    All choices for the modular text switching are
    defined here.

4/29/17 | MIT License
*/

function chooseFromList(list_of_choices){
    /* Returns a random item from list_of_choices
     */
    return list_of_choices[Math.floor(Math.random() * list_of_choices.length)];
}
function get_and_switch(list_of_choices,id_tag){
    /* Given an id_tag and a list of choices,
     * will randomly choose one item from the
     * list and assign it to the content of
     * the tagged element.
     */
    if(Math.random() > 0.9){
        document.getElementById(id_tag).innerHTML = chooseFromList(list_of_choices)
    }
}

function setSwitches(id){
    /* Given an id, sets intervals for
     * switching and shuffling related to the
     * piece.
     */
    switch(id){
        case badfood:
            setInterval(function(){get_and_switch(
                badfood_switch1,"badfood_switch1")},500);
            setInterval(function(){get_and_switch(
                badfood_switch2,"badfood_switch2")},500);
            setInterval(function(){get_and_switch(
                badfood_switch3,"badfood_switch3")},500);
            setInterval(function(){get_and_switch(
                badfood_switch4,"badfood_switch4")},500);
            setInterval(function(){get_and_switch(
                badfood_switch5,"badfood_switch5")},500);
            break;
        case busride:
            setInterval(function(){get_and_switch(bus_switch1,"bus_switch1")},500);
            setInterval(function(){get_and_switch(bus_switch2,"bus_switch2")},500);
            setInterval(function(){get_and_switch(bus_switch3,"bus_switch3")},500);
            setInterval(function(){get_and_switch(bus_switch4,"bus_switch4")},500);
            break;
        case sequences:
            setInterval(function(){get_and_switch(seq_switch1,"seq_switch1")},500);
            setInterval(function(){get_and_switch(seq_switch2,"seq_switch2")},500);
            setInterval(function(){get_and_switch(seq_switch3,"seq_switch3")},500);
            setInterval(function(){get_and_switch(seq_switch4,"seq_switch4")},500);
            setInterval(function(){get_and_switch(seq_switch5,"seq_switch5")},500);
            setInterval(function(){get_and_switch(seq_switch6,"seq_switch6")},500);
            break;
        case grandparents:
            setInterval(function(){get_and_switch(
                grandparents_switch1,"grandparents_switch1")},500);
            setInterval(function(){get_and_switch(
                grandparents_switch2,"grandparents_switch2")},500);
                        setInterval(function(){get_and_switch(
                grandparents_switch3,"grandparents_switch3")},500);
            setInterval(function(){get_and_switch(
                grandparents_switch4,"grandparents_switch4")},500);
            setInterval(function(){get_and_switch(
                grandparents_switch5,"grandparents_switch5")},500);
            setInterval(function(){get_and_switch(
                grandparents_switch6,"grandparents_switch6")},500);
            setInterval(function(){get_and_switch(
                grandparents_switch7,"grandparents_switch7")},500);
            setInterval(function(){get_and_switch(
                grandparents_switch8,"grandparents_switch8")},500);
            break;
        case csharp:
            //switch
            setInterval(function(){get_and_switch(
                sharp_switch1,"csharp_switch1")},500);
            setInterval(function(){get_and_switch(
                sharp_switch2,"csharp_switch2")},500);
            setInterval(function(){get_and_switch(
                sharp_switch3,"csharp_switch3")},500);
            setInterval(function(){get_and_switch(
                sharp_switch4,"csharp_switch4")},500);
            setInterval(function(){get_and_switch(
                sharp_switch5,"csharp_switch5")},500);
            setInterval(function(){get_and_switch(
                sharp_switch6,"csharp_switch6")},500);
            setInterval(function(){get_and_switch(
                sharp_switch7,"csharp_switch7")},500);
            setInterval(function(){get_and_switch(
                sharp_switch8,"csharp_switch8")},500);
            //shuffle
            setInterval(function(){
                document
                .getElementById("shuffle_csharp")
                .innerHTML = shuffleWordsFromText(
                    document.getElementById("shuffle_csharp").innerHTML)
            },500)
            break;
        case python:
            //switch
            setInterval(function(){get_and_switch(
                python_switch_1,"python_switch_1")},500);
            setInterval(function(){get_and_switch(
                python_switch_2,"python_switch_2")},500);
            setInterval(function(){get_and_switch(
                python_switch_3,"python_switch_3")},500);
            setInterval(function(){get_and_switch(
                python_switch_4,"python_switch_4")},500);
            setInterval(function(){get_and_switch(
                python_switch_5,"python_switch_5")},500);
            setInterval(function(){get_and_switch(
                python_switch_6,"python_switch_6")},500);
            setInterval(function(){get_and_switch(
                python_switch_7,"python_switch_7")},500);
            setInterval(function(){get_and_switch(
                python_switch_8,"python_switch_8")},500);
            //shuffle
            setInterval(function(){
                document
                .getElementById("python_shuffle_1")
                .innerHTML = shuffleWordsFromText(
                    document.getElementById("python_shuffle_1").innerHTML)
            },500)

            setInterval(function(){
                document
                .getElementById("python_shuffle_2")
                .innerHTML = shuffleWordsFromText(
                    document.getElementById("python_shuffle_2").innerHTML)
            },500)
            break;
        case sound:
            //switch
            setInterval(function(){get_and_switch(
                sound_switch_1,"sound_switch_1")},500);
            setInterval(function(){get_and_switch(
                sound_switch_2,"sound_switch_2")},500);
            setInterval(function(){get_and_switch(
                sound_switch_3,"sound_switch_3")},500);
            setInterval(function(){get_and_switch(
                sound_switch_4,"sound_switch_4")},500);
            setInterval(function(){get_and_switch(
                sound_switch_5,"sound_switch_5")},500);
            setInterval(function(){get_and_switch(
                sound_switch_6,"sound_switch_6")},500);
            setInterval(function(){get_and_switch(
                sound_switch_7,"sound_switch_7")},500);
            setInterval(function(){get_and_switch(
                sound_switch_8,"sound_switch_8")},500);
            setInterval(function(){get_and_switch(
                sound_switch_9,"sound_switch_9")},500);
            setInterval(function(){get_and_switch(
                sound_switch_10,"sound_switch_10")},500);
            setInterval(function(){get_and_switch(
                sound_switch_11,"sound_switch_11")},500);
            //shuffle
            setInterval(function(){
                document
                .getElementById("shuffle_sound")
                .innerHTML = shuffleWordsFromText(
                    document.getElementById("shuffle_sound").innerHTML)
            },500)
            break;
        case dreamsII:
            setInterval(function(){get_and_switch(
                dreamsII_switch_1,"dreamsII_switch_1")},500);
            setInterval(function(){get_and_switch(
                dreamsII_switch_2,"dreamsII_switch_2")},500);
            setInterval(function(){get_and_switch(
                dreamsII_switch_3,"dreamsII_switch_3")},500);
            setInterval(function(){get_and_switch(
                dreamsII_switch_4,"dreamsII_switch_4")},500);
            setInterval(function(){get_and_switch(
                dreamsII_switch_5,"dreamsII_switch_5")},500);
            setInterval(function(){get_and_switch(
                dreamsII_switch_6,"dreamsII_switch_6")},500);
            break;
        case phil:
            setInterval(function(){
                document
                .getElementById("shuffle_phil_1")
                .innerHTML = shuffleWordsFromText(
                    document.getElementById("shuffle_phil_1").innerHTML)
            },500)

            setInterval(function(){
                document
                .getElementById("shuffle_phil_2")
                .innerHTML = shuffleWordsFromText(
                    document.getElementById("shuffle_phil_2").innerHTML)
            },500)
            break;
        default:
            break;

    }
}

/* Lists of Choices for modular text switching */
```

```
//BADFOOD
var badfood_switch1 = [" Why did I let him put it in there",
                       " Why did he close it without checking"];
var badfood_switch2 = [" I just want to go home.",
                       " Where are they?",
                       " Why did I agree to come here?"];
var badfood_switch3 = [" Jersey"," Pennsylvania"," Virginia"];
var badfood_switch4 = [" Why did they leave when I am sick",
                       " Why didn't they invite me",
                       " Why did they do that"];
var badfood_switch5 = [" Did they see a dead guy?",
                       " Were they both actually getting high?",
                       " Philly is so fucking strange."];

//BUSRIDE
var bus_switch1 = [" I am always afraid my head will do through the glass",
                   " My head hurts enough as it is"];
var bus_switch2 = [" I feel gross thinking about it that way.",
                   " Why do I have such a long ride?",
                   " At least it gives me time to catch up on my sleep."];
var bus_switch3 = [" I just have to make friends with the rotating drivers.",
                   " I get so board on that ride. ",
                   " It gives me time to listen to my music, I guess."];
var bus_switch4 = [" It's a fucking miracle that I haven't been in more.",
                   " Why don't we actually fix what is causing the accidents?",
                   " Only a matter of time before I am in another one."];

//SEQUENCES
seq_switch1 = ["We have to learn cursive","They say we have to learn cursive"];
seq_switch2 = ["going to be up all night doing","never going to be able to do",
               "going to figure out how to do"];
seq_switch3 = ["Who came up with the cursive letter for uppercase 'G's? What\
 <br/>is wrong with them?","Who in their right mind thought cursive-uppercase \
 's' as still okay when <br/> the ampersand was created?"]
seq_switch4 = ["am I spending","do I have to spend"];
seq_switch5 = ["Why are they","Why am I"];
seq_switch6 = ["I want to cry.", "I think I am going to cry.", "I am crying."];

//GRANDPARENTS
var grandparents_switch1 = [" I don't want to be here.",
                            " Pops should have been here."];
var grandparents_switch2 = [" I guess we wouldn't all be here if they were around.",
                            " Though I guess we are only here because of them."];
var grandparents_switch3 = [" Now pops and grandma both overlook the Delaware.",
                            " Now they are both in the ground."];
var grandparents_switch4 = [" They must have forgotten about it. ",
                            " I am so hungry.",
                            " In fact, that is what I am wondering.",
                            " Whatever it doesn't matter.",
                            " Actually, no, they would probably ask about \
                            why my sister hasn't settled down yet."];
var grandparents_switch5 = [" I don't have a good memory of her.",
                            " She died before I got to know her.",
                            " The medicine wiped my memory.",
                            " I was too young when she passed."];
var grandparents_switch6 = [" Watching him build a bar and how proud \
he was when he finished. ",
" The car crash we were both in together.",
" The dinners I had with him."];
var grandparents_switch7 = [ " All of those Christmases and Thanksgivings.",
                             " When I was a messager between him and my \
                             father during an argument."," How we would discuss\
                             college and my what classes I was taking."];
var grandparents_switch8 = [" All of the symphonies we went to together. ",
                            " How kind he was when my father was in the hospital.",
                            " When I was his pallbearer..."];

//C#
var csharp_switch1 = [" Mr O. wants me to copy more code ",
                      " I have to write more code"]
var csharp_switch2 = [" I want to enjoy this, but I don't <br> understand \
what is happening.",
" I'm kind of lost right now, <br> there is a lot here."]
var csharp_switch3 = [" they don't know how to either",
                      " If I can figure it out, anyone should be able to"]
var csharp_switch4 = [" understandable "," unreadable "]
var csharp_switch5 = [" Oh, the zombies are chasing me",
                      " Holy shit, the zombies are moving"]
var csharp_switch6 = [" How"," Why"]
var csharp_switch7 = [" I am really lost now. "," Oh, I get it."]
var csharp_switch8 = [" How does the code know who the player is? ",
                      " How do they know to get around walls? ",
                      " Why does this really work though? ",
                      " Why does this work on all of them at once? ",
                      " Why do they seem so fast? ",
                      " How did the tutorial come up with this? ",
                      " Do AI in other games work this way? ",
                      " How do I get them to be animated so they \
                      don't just slide towards me?"]
```

```
//PYTHON
var python_switch_1 = ["Did I really spend eleven years reading nonsense \
                        words to figure out how to spell to get stuck on this?",
                        "I really spent eleven years reading nonsense words to \
                        figure out how to spell to get stuck on this."]
var python_switch_2 = ["Maybe they were right, I am going to be a gas \
                        station attendent. They told me otherwise, but \
                        I know.","Maybe I was right, I am going to be a \
                        gas station attendent. They knew."]
var python_switch_3 = ["I can figure this out","I cannot figre this out"]
var python_switch_4 = ["I mean, I know I didn't do much reading in highschool \
                        nor did I do very much math, but this should be simple. \
                        Maybe this is why Mr. O said he couldn't teach me \
                        programming, he never was able to learn.","I mean, I \
                        know I did not read in highschool nor did I do very \
                        much math, but this is hard. Maybe this is why Mr. \
                        O said he couldn't teach me programming, he didn't want \
                        to waste the time."]
var python_switch_5 = ["Jim","Mom","Dad","Matan","Pops","Grandma","Emily"]
var python_switch_6 = ["I am worse than before.","I am no better than I was."]
var python_switch_7 = ["Wait, it worked...it actually worked.Let me try it again.",
                       "Wait, it worked...why did it work? Let me try it again."]
var python_switch_8 = ["Fuck, why didn't that work.","Fuck, why can't I do this."]


//SOUND
var sound_switch_1 = ["get","like"]
var sound_switch_2 = ["is overwhelming","has no point","is just orderly noise"]
var sound_switch_3 = ["think straight","stand it"]
var sound_switch_4 = ["drown in","get lost in","feel choked by"]
var sound_switch_5 = ["architect","engineer"]
var sound_switch_6 = ["it doesn't work","you did something wrong"]
var sound_switch_7 = ["car horn","bang","crash","scream"]
var sound_switch_8 = ["yelling","shattering","trains","hammering"]
var sound_switch_9 = ["rained", "reigned"]
var sound_switch_10 = ["Music","Sound","Noise"]
var sound_switch_11 = ["you","me"]

//DREAMS II
var dreamsII_switch_1 = [" unusual"," strange"," actually fictitious"];
var dreamsII_switch_2 = ["I was standing on a patio, only looking to my right. \
I couldn't <br>seem to turn my head. There as grass just beyond my feet heavily \
<br>dotted with yellow pansy flowers. They rested on a downward slope. <br>At the\
 bottom of the small hill was another patio covered in wrought <br>iron table \
 frames with wood tops. Above each table was a yellow <br>umbrella. From my \
 viewpoint, the umbrellas were about the same size <br>as the adjacent pansies. \
 It all washed into a single, colorful <br>movement the hills that lay even \
 further down, creating these flumes <br>with the small rivers at their bases.\
 I blinked and it was over."," There was just fire over a small hill to my \
 right. I was running <br>along side it, never quite what caused it. It was \
 night and the pathway <br>was lit by street lights. To my left was the longest \
 parking garage I <br>have ever seen. Just past the fire skyscrapers could be \
 seen. I ran for <br>what felt like hours, but then tripped and woke up."," I \
 just walked for a long time in a dark room. I started off slowly,  <br>trying \
 to find a wall of some sort to ground myself. After what felt <br>like an \
 eternity, I gave up. Once I gave up on the wall, I just kind <br>of walked \
 forward. There was no light. No sound. Nothing. I couldn't <br>even tell you \
 if it were hot or cold. I suddenly was woken up."];
var dreamsII_switch_3 = [" 12"," 5"," 20"," jack-shit"];
var dreamsII_switch_4 = [" Why has this started happening now?",
                         " Well, it hasn't stopped me before.",
                         " I am just going to have a head ache."];
var dreamsII_switch_5 = [" my depression coming back",
                         " just my luck",
                         " living in this fucked up house",
                         " because this job is so tiring"];
var dreamsII_switch_6 = [" go back to sleep"," get out of here"];
```

audio.js:

```javascript
/*
audio.js
by Logan Davis

Description:
    Instantiates and defines audio objects and
    audio-specific functions for *Release*

4/29/17 | MIT License
*/

var audio_for_eval_1 = new Audio("sound/Color.m4a");
var audio_for_eval_2 = new Audio("sound/Death.m4a");
var audio_for_eval_3 = new Audio("sound/Dreams.m4a");
var audio_for_eval_4 = new Audio("sound/Goodbye.m4a");
var audio_for_budride = new Audio("sound/Krystal Weakness.m4a");
var audio_for_badfood = new Audio("sound/Krystal Clinical.m4a");
var audio_for_sequences = new Audio("sound/Sophie - Strengths.m4a");
var audio_for_grandparents = new Audio("sound/Krystal Medical.m4a");
var audio_for_csharp = new Audio("sound/Sophie - Social 2.m4a");
var audio_for_python = new Audio("sound/Krystal Language 1.m4a");
var audio_for_sound = new Audio("sound/Krystal Language 2.m4a");
var audio_for_dreams2 = new Audio("sound/Start.m4a");
var blank = new Audio();

var active_audio = audio_for_eval_1; //stored current/last played audio

function activate_audio(name){
    /* resets currently playing or last
     * played audio object to beginning,
     * switches to a new object corresponding
     * to *name* , and begins to play it.
     */
    active_audio.pause();
    active_audio.currentTime = 0;
    active_audio = name;
    setTimeout(function () {
        active_audio.play();
    }, 200);
}
```

MinNo Source Code:

compiler.rkt:

```
#lang racket
#|
compiler.rkt
Logan Davis

DESCRIPTION:
    The centeral file to wrap file opening and compiling
    for MinNo.

TO USE:
    Run from the command like and provide two arguements:
     - source_file: the MinNo file that you want to compile.
     - output_file: the path and name you want to give the
                    resulting C program.

3/24/17 | Racket 6.8 | OS: MacOS
|#

(require (prefix-in brag: brag/support))
(require "lexer.rkt")
(require "grammar.rkt")
(require "handlerDirector.rkt")
(require "unpacker.rkt")


(define token-stream '()) ; accumulator



;recursively lexs a file-port accumulating tokens in token-stream.
(define reader
    (lambda (file)
      (if end-of-file
                  '()
                  (and (set! token-stream
                            (append token-stream (list (tokenize file)))
                       (reader file)))))

(define source-code (open-input-file (vector-ref
                (current-command-line-arguments) 0))) ;;example file
```

```
(port-count-lines! source-code)  ;;enable line counting on port
(printf "File Found.\nTokenizing...\n")

(if (equal? (reader source-code) '())
    (printf "File has been tokenized.\nParsing...\n")
    (and (printf "Error tokenizing file!\n")
         (exit)))

(define tree-raw (syntax->datum (parse token-stream)))

;;(printf "\nResult:\n-------\n~a" tree-raw)

(define ast-translated (tree-transform tree-raw))
;; (printf "\nResult:\n-------\n~a" ast-translated)

(define output-string (unpack ast-translated)) ;; parse/return a datum



(printf "\nResult:\n-------\n~a" output-string)

(with-output-to-file (vector-ref
                       (current-command-line-arguments) 1)
                       #:exists 'replace
  (lambda ()
    (printf "~a" output-string)))

(printf "Done!\n")
(exit)
```

definitionHandler.rkt:

```
#lang racket
#|
definitionHandler.rkt
By Logan Davis

Responible for translating function defition
ASTs.

3/24/17 | Racket 6.8 | MacOS
|#
```

```racket
(require "state-roster.rkt")

; The parent handler function to be called on a definition AST
(define definition-tag-handler
  (lambda (datum)
    (define id (third datum))
    (define return-type (get-return-type  (fourth datum)))
    (define args (get-args (fourth datum)))
    (list 'definition return-type id args)))

; Gets the return type of the def AST
(define get-return-type
  (lambda (signature)
    (define return-type (last signature))
    (cond [(equal? (first return-type) 'nonetype) '(void-type "void")]
          [else (list 'type (last return-type))])))

; Gets arguements from the definition AST
(define get-args
  (lambda (signature)
    (define args (get-args-helper signature))
    (define properly-formed-args '(args))
    (for ([item args])
      (set! properly-formed-args (append properly-formed-args
                                 (list (last item) (first item)))))
    properly-formed-args))

; Assists get-args in parsing sub-AST elements
(define get-args-helper
  (lambda (signature)
    (define full-arg-list '())
    (define arg-accumulator '())
    (for ([item (rest signature)]
          #:final (equal? (first item) 'arrow))
      (cond [(or (equal? (first item) 'comma)
                 (equal? (first item) 'arrow))
             (and (set! full-arg-list (append full-arg-list
                                             (list arg-accumulator)))
                  (set! arg-accumulator '()))]
            [else
             (and (set! arg-accumulator
                        (append arg-accumulator (list item)))
                  (if (equal? (first item) 'id)
                    (add-variables-defined (second item)) '()))]))
    full-arg-list))
```

```
(provide (all-defined-out))
```

formatter.rkt:

```racket
#lang racket
#|
formatter.rkt

A set of utils for string correction in the MinNo compiler

3/24/17 | Racket 6.8 | MacOS
|#

(define correct-over-commaing
  (lambda (code-string)
    (string-replace (string-replace code-string ",)" ")")
                    ", )" ")")))

(define correct-over-tabbing
  (lambda (sketch)
    (string-replace sketch "\t }" " }")))

(define remove-tabs
  (lambda (sketch)
    (string-replace sketch "\t" "")))

(define remove-newline
  (lambda (sketch)
    (string-replace sketch "\n" "")))

(define remove-tabs-and-newlines
  (lambda (sketch)
    (remove-tabs (remove-newline sketch))))

(define correct-over-spacing
  (lambda (sketch)
    (string-replace (string-replace sketch "      " "")
                    "  " " ")))

(define remove-delimit
  (lambda (delimited-string)
    (string-replace delimited-string ";\n\t" "")))

(define make-one-line
  (lambda (multi-line-string)
    (string-replace multi-line-string ";\n\t" ";")))

(provide (all-defined-out))
```

grammar.rkt:

```
#lang brag
; grammar.rkt
; by Logan Davis
;
; The grammar spec of MinNo to be consumed by BRAG.
;
; 3/24/17 | Racket 6.8 | MacOS


; main structure
program: (let-statement | define-statement )+

let-statement: let id colon [mutable-tag] type eq statement delimit
relet-statement: id eq statement delimit

define-statement: def id signature scope-statement


func-call: id (lit | id | lparen expr rparen)*
statement: expr
delimited-statement: statement delimit

return-statement: return delimited-statement

;code blocks
scope-statement: lbrac (delimited-statement| let-statement |
                        relet-statement | while-loop|
                        for-loop | conditional|return-statement)*
                 rbrac

; Typing statements
type: TYPE | ARRAY-TYPE lSqBrac TYPE rSqBrac ;;; "int" | "array[int]"
signature: (((id colon type comma)* id colon type) | nonetype)
             arrow (type | nonetype)

; control-flow
while-loop: while (comparison | statement) scope-statement
for-loop: for (let-statement | relet-statement)
              comparison delimit
              relet-statement scope-statement
conditional: if comparison scope-statement
             [else scope-statement]
```

```
comparison: (statement | lparen comparison rparen)
            bool-comp
            (statement | lparen comparison rparen);;; "x == y"


; Seperating characters
lbrac: LBRAC   ;;; {
rbrac: RBRAC   ;;; }
lparen: LPAREN   ;;; (
rparen: RPAREN   ;;; )
lSqBrac: LSQBRAC   ;;; [
rSqBrac: RSQBRAC   ;;; ]
comma: COMMA    ;;; ,
colon: COLON    ;;; :
delimit: SEMI-COLON ;;; ;
eq: EQUAL ;;; =
return: RETURN

; Ops taken from http://math.purduecal.edu/
; ~rlkraft/cs31600-2012/chapter03/syntax-examples.html

expr: (func-call | term) [(add | sub) term]*
term: factor [(mult | div) factor]*
factor: (lparen expr rparen | id | lit)

add: ADD-OP
sub: SUB-OP
mult: MULT-OP
div: DIV-OP

; data structure
lit: (int | float | string | array)
array: lSqBrac (((lit| id) comma)* (lit | id)) rSqBrac
int: INT
float: FLOAT
nonetype: NONE-TYPE
mutable-tag: MUT-TAG
string: STRING
while: WHILE           ;;; "while"
for: FOR
if: IF                 ;;; "if"
else: ELSE             ;;; "else"
bool-comp: BOOL-COMP   ;;; "==" | ">" ... ect

;;; Type info
```

```
arrow: ARROW   ;;; "->"

; other keywords
id: ID[lSqBrac (int | id) rSqBrac]
def: DEF
let: LET
```

handlerDirectory:

```
#lang racket
#|
handlerDirector.rkt
Logan Davis

Description:
    The director for specific AST structures and handing them
    to handlers for translating.

To Use:
    Import into another racket file and call (tree-transform)
    on a syntax->datum list of a MinNo AST.


3/24/17 | Racket 6.8 | MacOS
|#
(require "definitionHandler.rkt")
(require "lettypeHandler.rkt")
(require "state-roster.rkt")


; Handles a 'program branch of an AST
(define program-tag-handler
  (lambda (datum)
    (let ([content (rest datum)]
          [new_program '(program)])
         (for ([item content])
           (set! new_program (append new_program
                                     (list (tree-transform item)))))
         new_program)))

; handles block statements by breaking them into one
; line statements and concatting them
(define scope-statement-handler
  (lambda (datum)
    (set-in-scope-statement-flag #t)
    (let ([block '('scope-statement)])
      (for ([item (rest datum)])
        (if (or (equal? (first item) 'lbrac)
                (equal? (first item) 'rbrac))
            (set! block (append block (list item)))
            (set! block (append block (list (tree-transform item))))))
      (set-in-scope-statement-flag #f)
      (list block))))

; processes oneline statements
(define statement-handler
  (lambda (datum)
    datum))

; handles conditions
(define conditional-handler
  (lambda (datum)
    datum))

; handles one line statements with a ";" delimiting it
(define delimited-statement-handler
  (lambda (datum)
    (append (second datum) (list (last datum)))))

; handles for loop structures
(define for-loop-handler
  (lambda (datum)
    (list (first datum)
          (second datum)
          (tree-transform (third datum))
          (fourth datum)
          (fifth datum)
          (tree-transform (sixth datum))
          (first (tree-transform (seventh datum))))))




; Takes some syntax->datum list of original
; AST and returns translated AST
(define tree-transform
  (lambda (datum)
    (define tag (first datum))
    (cond
      ;;;----------------TOP LEVEL PROGRAM-----------------------;;;
      [(equal? tag 'program) (program-tag-handler datum)]
      ;;;----------------LET STATEMENT---------------------------;;;
      [(equal? tag 'let-statement) (let-tag-handler datum)]
      [(equal? tag 'relet-statement) datum]
      ;;;----------------DEFINE STATEMENT------------------------;;;
      [(equal? tag 'define-statement)
       (append (definition-tag-handler datum)
               (scope-statement-handler (fifth datum)))]
      ;;;----------------STATEMENT HANDLER-----------------------;;;
      [(equal? tag 'statment) (statement-handler datum)]
      [(equal? tag 'delimited-statement) (delimited-statement-handler datum)]
      [(equal? tag 'scope-statement) (scope-statement-handler datum)]
      ;;;----------------CONDITIONAL-HANDLER---------------------;;;
      [(equal? tag 'conditional) (conditional-handler datum)]
      ;;;----------------RETURN-HANDLER--------------------------;;;
      [(equal? tag 'return-statement) datum]
      ;;;----------------LOOP-HANDLER----------------------------;;;
      [(equal? tag 'while-loop) datum]
      [(equal? tag 'for-loop) (for-loop-handler datum)])))


(provide (all-defined-out))
```

lettypeHandler.rkt:

```
#lang racket
#|
lettypeHandler.rkt
By Logan Davis

Responsible for prepping and translating let declarations
given by a handlerDirector.

3/24/17 | Racket 6.8 | MacOS
|#
(require "state-roster.rkt")

; Handles a 'let-statement branch of an AST
(define let-tag-handler
  ;;; add section for mutable lets
  (lambda (datum)
    (let ([mutable? (mutable-tag? datum)]
          [ast-result '()])
         (add-variables-defined (second (third datum))) ;;catalog it as variable
         (define is-array #f)
         (if mutable?                          ;; if immutable, catalog it

             (and (set! is-array (equal? (second (sixth datum)) "array"))
                  (set! ast-result (list  'declaration
                      (list 'type (letType-handler-get-type (sixth datum)))
                      (correct-id-if-array (third datum) is-array)  ;; id
                      (seventh datum)  ;; equal symbol
                      (eighth datum);;value
                      (ninth datum))))
             (and (set! is-array (equal? (second (fifth datum)) "array"))
                  (set! ast-result (list  'declaration
                      (list 'type (string-append
                                   "const PROGMEM "
                                   (letType-handler-get-type (fifth datum))))
                      (correct-id-if-array (third datum) is-array)  ;; id
                      (sixth datum) ;; equal symbol
                      (seventh datum) ;;value
                      (eighth datum))) ;; delimiter
                  (if in-scope-statement '()
                       (add-prog-mem-variable (third datum)))))
      ast-result))

; Appends "[]" to the id tag for C's notation
(define correct-id-if-array
  (lambda (id-string is-array)
    (if is-array
        (list 'id (string-append (second id-string) "[]"))
        id-string)))


; Translates types from original-AST to target AST.
(define letType-handler-get-type
  ;;; TODO: add multi-dimensional array type handling.
  (lambda (datum)
    (cond [(equal? (second datum) "array") (fourth datum)]
          [else (second datum)])))


; Verifies that a mutable tag is present in a let-statement
(define mutable-tag?
  (lambda (let-statement)
    (define result (filter (lambda (x) (equal? (first x) 'mutable-tag))
                           (rest let-statement)))
    (if (equal? (length result) 1) #t #f)))

(provide (all-defined-out))
```

lexer.rkt:

```
#lang racket
#|
possible_grammar_lex.rkt
By Logan Davis

    Description:
        A lexer for some non-trivial grammar.

3/24/17 | Racket 6.8 | MacOS
|#
(require parser-tools/lex)
(require parser-tools/lex-sre)
(require (prefix-in brag: brag/support))

(define-lex-abbrev
     int? (: (? "-") (+ (char-range #\0 #\9))))

(define-lex-abbrev float? (: (? "-")
                             (+ (char-range #\0 #\9))
                             "."
                             (+ (char-range #\0 #\9))))

(define-lex-abbrev string? (: #\" (* (char-complement #\")) #\"))

(define-lex-abbrev identifier?
    (: (+ alphabetic)
       (* (or alphabetic #\_ #\! #\? (char-range #\0 #\9)))))

(define-lex-abbrev bool-comp? (or "==" "&&" "||" "<"
                                  ">" ">=" "<=" "!="))
(define-lex-abbrev type? (or "int" "char" "float"))

(define-lex-abbrev comment?
    (or (: "//" (* (char-complement #\newline)))
        (: "#/" (complement (: any-string "/#" any-string)) "/#")))

(define end-of-file #f) ; are we done yet?
```

```
; define file marker position
(define line 0)
(define column 0)
(define offset 0)


;Takes a file port and returns a single token-struct per call.
(define tokenize
  (lambda (file)
    (set!-values (line column offset) (port-next-location file))
    (define find-token
      (lexer
       [comment?    '(COMMENT lexeme #:skip? #t)]
       [int?        (list 'INT lexeme)]
       [float?      (list 'FLOAT lexeme)]
       [string?     (list 'STRING lexeme)]
       [type?       (list 'TYPE lexeme)]
       ["none"      (list 'NONE-TYPE lexeme)]
       ["mutable"   (list 'MUT-TAG lexeme)]
       ["array"     (list 'ARRAY-TYPE lexeme)]
       ["return"    (list 'RETURN lexeme)]
       [#\*         (list 'MULT-OP lexeme)]
       [#\/         (list 'DIV-OP lexeme)]
       [#\-         (list 'SUB-OP lexeme)]
       [#\+         (list 'ADD-OP lexeme)]
       [#\{         (list 'LBRAC lexeme)]
       [#\}         (list 'RBRAC lexeme)]
       [#\(         (list 'LPAREN lexeme)]
       [#\)         (list 'RPAREN lexeme)]
       [#\[         (list 'LSQBRAC lexeme)]
       [#\]         (list 'RSQBRAC lexeme)]
       [#\,         (list 'COMMA lexeme)]
       [#\:         (list 'COLON lexeme)]
       [#\;         (list 'SEMI-COLON lexeme)]
       [bool-comp?  (list 'BOOL-COMP lexeme)]
       [#\=         (list 'EQUAL lexeme)]
       ["while"     (list 'WHILE lexeme)]
       ["for"       (list 'FOR lexeme)]
       ["if"        (list 'IF lexeme)]
       ["else"      (list 'ELSE lexeme)]
       ["->"        (list 'ARROW lexeme)]
       ["def"       (list 'DEF lexeme)]
       ["let"       (list 'LET lexeme)]
       [identifier? (list 'ID lexeme)]
       [(or #\newline #\space #\tab nothing) '(WHITESPACE lexeme #:skip? #t)]
      [(eof) (set! end-of-file #t)]))
    (define result (find-token file))
  (cond [(equal? result (void)) (void)]  ;;;EOF
        [(equal? (length result) 2) (brag:token
                  (first result) (second result) #:line line #:column column)]
        [else (brag:token (first result) #:skip? #t)])))

(provide (all-defined-out))
```

state-roster.rkt:

```
#lang racket
#|
A collection of stateful trackers (with setters and getters)
for the compiler and the source code it is compiling.

3/24/17 | Racket 6.8 | MacOS
|#
(define prog-mem-variables '())
(define variables-defined '())
(define arrays-defined '())
(define in-def-unpacking #f) ;; a flag for if in a func definition
(define in-func-call #f) ;; a flag to properly wrap prog-mem variables
(define in-scope-statement #f)
(define tab-level 0)

;; appending setters
(define add-prog-mem-variable
  (lambda (item)
    (set! prog-mem-variables (append prog-mem-variables (list item)))))

(define add-variables-defined
  (lambda (item)
    (set! variables-defined (append variables-defined (list item)))))

(define add-arrays-defined
  (lambda (item)
    (set! arrays-defined (append arrays-defined (list item)))))

;; Toggle setters
(define set-in-def-flag
  (lambda (state)
    (set! in-def-unpacking state)))
```

```
(define set-in-func-call-flag
  (lambda (state)
    (set! in-func-call state)))

(define set-in-scope-statement-flag
  (lambda (state)
    (set! in-scope-statement state)))

(define set-tab-level
  (lambda (level)
    (set! tab-level level)))
    ;(if (< tab-level 0) (set! tab-level 0) '()))))

(provide (all-defined-out))
```

unpacker.rkt:

```
#lang racket
#|
unpacker.rkt
By Logan Davis

DESCRIPTION:
    Unpacks an Arduino-C AST from the MinNo
    compiler's (tree-trasform) into a raw
    string.

TO USE:
    call (unpack) on an Arduino-C AST.

3/24/17 | Racket 6.8 | MacOS 10.11
|#
(require "state-roster.rkt")
(require "formatter.rkt")

; unpacks an Arduino-C AST into a string
(define unpack
  (lambda (program-datum)
    (let ([program-structure (rest program-datum)]
          [sketch ""])
      (for ([item program-structure])
        (if (equal? (first item) 'definition) (set-in-def-flag #t) '())
        (set! sketch (string-append sketch " " (correcter item) "\n")))
        (if in-def-unpacking (set-in-def-flag #f) '())
      sketch)))

; checks if a datum needs to be corrected or is ready to be unpacked.
(define correcter
  (lambda (datum)
    (define needs-correcting '(lSqBrac
                               rSqBrac
                               func-call
                               delimit
                               lbrac
                               rbrac
                               args
                               id
                               while-loop
                               conditional
                               for-loop))
    (let ([output ""])
      (if (member (first datum) needs-correcting)
          (set! output (string-append output (correction-handler datum)))
                    (if (string? (second datum))
              (set! output (string-append output (second datum) )) ;;ready to go
              (let ([remaining-data (rest datum)]) ;;needs to be unpacked further
                (for ([item remaining-data])
                  (set! output (string-append output (correcter item) " "))))))
      (correct-over-tabbing (correct-over-spacing output)))))

; Either returns a corrected string or passes off the datum to a handler
(define correction-handler
  (lambda (datum)
    (cond [(equal? (first datum) 'delimit) (string-append ";\n"
                                           (make-string tab-level #\tab))]
          [(equal? (first datum) 'lSqBrac) "{"]
          [(equal? (first datum) 'rSqBrac) "}"]
          [(equal? (first datum) 'lbrac) (increase-tab-level-and-start-block)]
          [(equal? (first datum) 'rbrac) (lower-tab-level-and-delimit-block)]
          [(equal? (first datum) 'args) (arg-corrector datum)]
          [(equal? (first datum) 'func-call) (func-call-or-id datum)]
          [(equal? (first datum) 'id) (correct-if-immutable-or-array datum)]
          [(equal? (first datum) 'while-loop) (while-loop-corrector datum)]
          [(equal? (first datum) 'for-loop) (for-loop-corrector datum)]
          [(equal? (first datum) 'conditional) (conditional-corrector datum)]
          [else "PLACEHOLDER"])))

(define lower-tab-level-and-delimit-block
  (lambda ()
    (set-tab-level (- tab-level 1))
    (string-append "}\n" (make-string tab-level #\tab))))

(define increase-tab-level-and-start-block
  (lambda ()
    (set-tab-level (+ tab-level 1))
    (string-append "{\n" (make-string tab-level #\tab))))



(define while-loop-corrector
  (lambda (datum)
    (string-append "while(" (correcter (third datum)) ")"
                    (correcter (fourth datum)))))

(define for-loop-corrector
  (lambda (datum)
    (string-append (make-one-line (string-append "for("
                      (remove-tabs-and-newlines
                      (string-append (correcter (third datum))
                                     (correcter (fourth datum))
                                     (correcter (fifth datum))
                                     (remove-delimit (correcter
                                                      (sixth datum)))))
                                                 ")"))
                                                 (correcter
                                                 (seventh datum)))))
;; corrects conditional if statements
(define conditional-corrector
  (lambda (datum)
    (string-append "if(" (correcter (third datum)) ")"
                   (correcter (fourth datum))
                   (if (member '(else "else") datum)
                       (string-append "else" (correcter (sixth datum)))
                       ""))))


;; deals with wrapping of values and progmem variables.
(define correct-if-immutable-or-array
  (lambda (datum)
    (let ([result (second datum)])
      (if (member (second datum) arrays-defined)
          (set! result (string-append result "[]"))
                '())
      (if (member '(lSqBrac "[") datum)
          (set! result (string-append result "[" (second (fourth datum)) "]"))
          '())
      (if (and (member datum prog-mem-variables) in-func-call)
          (set! result (string-append "pgm_read_word(&" result ")"))
          '())
      result)))

; check if it is not just a variable
(define func-call-or-id
  (lambda (datum)
    (if (member (second (second datum)) variables-defined)
        (correct-if-immutable-or-array (second datum))
        (func-call-corrector datum))))

; adds parens and commas to a func-call datum and unpacks
(define func-call-corrector
  (lambda (func-call-datum)
    (set-in-func-call-flag #t) ;; start in context
    (let ([func-call-string (string-append
                                (second (second func-call-datum)) "(")]
          [call-ast (rest (rest func-call-datum))])
      (for ([item call-ast])
        (set-in-func-call-flag #t) ;; correct context for nested func calls
        (cond [(equal? (first item) 'lparen) ""]
              [(equal? (first item) 'rparen) ""]
              [(set! func-call-string
                     (string-append func-call-string (correcter item) ","))]))
      (set-in-func-call-flag #f)
      (correct-over-commaing (string-append func-call-string ")")))))

; adds parens and commas to a definition's sub-datum "args" and unpacks
(define arg-corrector
  (lambda (arg-datum)
    (if (equal? arg-datum '(args (nonetype "none") (nonetype "none")))
        "()"
        (let ([arg-string ""]
              [args (rest arg-datum)])
          (for ([item args])
            (if (equal? (first item) 'type)
                (set! arg-string (string-append arg-string (second item) " "))
                (set! arg-string (string-append arg-string (second item) ", "))))
          (correct-over-commaing (string-append "(" arg-string ")"))))))

(provide (all-defined-out))
```

Programming Language Test Source Code:

fractionSumFinder.kt:

```kotlin
/*
fractionSumFinder.kt
By Logan Davis

The program is to answer the following test question:

   What permutation of the digits from 1 to 9 will add to 1
   when arranged in a form similar to 1/23 + 4/56 + 7/89 ?
   Find the answer with a brute force search.

TO-USE:
 Compile:
 $ kotlinc webscraper.kt -include-runtime -d <some output name>.jar
 After it is done compiling:
 $ java -jar fractionSumFinder.jar <Some sequence of integers except 0>

4/11/17 | Racket 6.8 | MIT license
 */
package fractionSumFinder

fun main(args: Array<String>){
    /*
     * parses the user's inputted set and runs the permutations check.
     */
    val set: MutableList<Int> = mutableListOf()
    for(arg in args){
        try{
            set.add(arg.toInt())
        }catch (e: Exception){
            println("Error: Invalid argument.\nUsage: java -jar fraction
            SumFinder.jar <Some sequence of integers except 0>")
        }
    }

    perm(mutableListOf(),set)
    val permutations: MutableList<MutableList<Int>> = sets
    test_for_sum(permutations)
}
```

fractionSumFinderUtils.kt:

```kotlin
/*
fractionSumFinderUtils.kt
By Logan Davis

This collection of functions is to be interfaced
through fractionSumFinder.kt

Please refer to its documentation.

4/11/17 | Racket 6.8 | MIT license
*/
package fractionSumFinder

var sets : MutableList<MutableList<Int>> = mutableListOf(mutableListOf())

fun test_for_sum(sets: MutableList<MutableList<Int>>): Unit{
    /*
     * given the equation scheme in the question, this tests
     * a list of sets to see if any sum to 1.
     *
     * @param sets: a list of all permutations of a set for testing
     */
    for(set in sets){
        if(set.isEmpty()){
            continue
        }
        var sum = (set[0].toFloat()/((set[1]*10)+set[2]))+
                  (set[3].toFloat()/((set[4]*10)+set[5]))+
                  (set[6].toFloat()/((set[7]*10)+set[8]))
        if(sum == 1.0.toFloat()){
            println("Set "+set.toString()+" sums to 1.")
        }
    }
}

fun perm(prefix: MutableList<Int>, s: MutableList<Int>): Unit{
    /*
     * computes and appends a permutation of a given list
     * to the global variable "sets"
     *
     * @param prefix: the currently computed permutation
     * @param s: the intial set to permute
     */
    // adapted from here: http://introcs.cs.princeton.edu/java/23recursion/
    // Permutations.java.html
    val n = s.count()
    if (n == 0)
        sets.add(prefix)
    else {
        for (i in 0..n - 1)
            perm((prefix + s[i]).toMutableList(), (s.subList(0, i) +
                    s.subList(i + 1, n)).toMutableList())
    }
}
```

webscraper.kt:

```kotlin
/*
webscraper.kt
By Logan Davis

This program is to answer the following question:

B) web crawler visualization

   Write a program which will first build a network of URLs
   and the links between them by starting at a given web page
   and following its links outwards, and then generate a visual
   representation of that network.

   All the details are up to you, including which page to start
   at, how far to go, and how to display the result.

DESCRIPTION:
   Constructs a .dot graph spec from a base url tracing the
   links from that base page to a specified depth.

   The produced .dot file can be made into a visual graph
   using either graphviz or gephi.

   For an example, here is how to make the output'ed spec
   into a PNG:

   $ dot -Tpng <the output from this script> -O

TO-USE:
   Compile:
    $ kotlinc kotlinc webscraperUtils.kt webscraper.kt \
     webscraperTests.kt -include-runtime -d <some output name>.jar
   After it is done compiling:
    $ java -jar <the compiled webscraper.jar> <some url to start>\
        <an integer for spelunking depth> <an output name>


   This file can either be run directly, which will start a
   prompt to construct the graph.

4/11/17 | JDK 1.8/Kotlin 1.0.4 | MIT license
 */
package webscraper

import java.io.File

fun main(args: Array<String>){
    testsuite()

    val base_url: String
    val depth: Int
    val output_path: String
    try{
        base_url = args[0]
        depth = args[1].toInt()
        output_path = args[2]
    }catch(e : NumberFormatException ){
        println("Error: Invalid depth number.\nUsage: $ java -jar
        webscraper.jar <some full url> <some integer> <output name>")
        return
    }catch(e: ArrayIndexOutOfBoundsException){
        println("Error: Malformed input.\nUsage: $ java -jar
        webscraper.jar <some full url> <some positive integer> <output name>")
        return
    }
    val graph_spec: String = " digraph webGraph{\n"+
                             draw_graph(base_url, depth) +"}"
    val file = File(output_path+".dot")
    file.writeText(graph_spec)
}
```

webscraperTest.kt

Kotlin

```kotlin
/*
webscraperTests.kt
By Logan

A very small set of tests for webscraper.kt

Please refer to it's documentation for use.
These tests are run at the beginning of every
invokation of webscraper.main

11/4/17 | JDK 1.8/Kotlin 1.0.4 | MIT license
*/

package webscraper

fun testsuite():Unit {
    correctUrlReturn()
    correctLinkParse()
}

fun correctUrlReturn():Unit{
    //http://stackoverflow.com/questions/1075895/
    //how-can-i-catch-all-the-exceptions-that-will-
    //be-thrown-through-reading-and-writi
    var response: String = ""
    try{
        response = get_url_content("http://csmarlboro.org/ldavis/webscrape_test.html")
    } catch (e: Exception ){
        println("Error testing get_url_content: "+e)
    }
}

fun correctLinkParse():Unit{
    val url: String = "http://csmarlboro.org/ldavis/webscrape_test.html"
    val response: String = get_url_content(url)
    var links: MutableList<String> = mutableListOf()
    val expectedLinks: MutableList<String> = mutableListOf("http://www.test-url.com")
    try{
        links = get_exact_links(get_anchors(response), url)
        if(links != expectedLinks){
            throw Exception()
        }
    } catch( e :Exception){
        println("Error testing get_exact_links.\nExpected:\n"+
                              expectedLinks+"\nBut got:\n"+links)
    }
}
```

webscraperUtils.kt:

```kotlin
/*
webscraperUtils.kt
By Logan Davis

These are all the functions used by webscraper.kt

Please read it's documentation.

11/4/17 | JDK 1.8/Kotlin 1.0.4 | MIT license
*/
package webscraper

import java.io.InputStream
import java.io.FileNotFoundException
import java.lang.NumberFormatException
import java.lang.ArrayIndexOutOfBoundsException
import java.net.*
import kotlin.collections.*

var graph_spec: String = ""

fun get_url_content(url_string: String): String{
    /*
     * Gets a url's content and returns it as a string.
     *
     * @param url_string: the full url to get html from
     * @return the html content from the page request
     */
    //coverted from code: http://stackoverflow.com
    // /questions/31462/how-to-fetch-html-in-java
    val url = URL(url_string)
    val stream : InputStream
    try {
        stream = url.openStream()
    } catch (e: FileNotFoundException){
        return ""
    } catch (e: UnknownHostException){
        return ""
    }
    var stream_status = 0
    val buffer = StringBuffer()
    while ( stream_status != -1) {
        stream_status = stream.read()
        buffer.append(stream_status.toChar())
    }
    return buffer.toString()
}

fun get_anchors(content: String): MutableList<String>{
    /*
     * Gets all anchor tags from an html string
     *
     * @ content: a string containing html
     * @ a list of anchor tags from the html
     */
    val broken_content: List<String> = content.split("<")
    var anchors: MutableList<String> = mutableListOf()
    for(item in broken_content){
        if(item.length < 2){
            continue
        }else if(item.subSequence(0,2) == "a "){
            anchors.add(item.replace(" ","").replace("\n",""))
        }
    }
    return anchors
}

fun get_exact_links(anchors: MutableList<String>,
                    base_url: String): MutableList<String>{
    /*
     *  Gets the links from an anchor tag and
     *  appends the rest of the url if the link
     *  is a relative path.
     *
     *  @param anchors: a list of anchor tags as strings
     *  @param base_url: the url that the anchors came from
     *  @return a link of urls from the anchor tags as strings
     */
    var links: MutableList<String> = mutableListOf()
    for(anchor in anchors){
        var parts: List<String> = anchor.split('"')
        for(i in parts){
            if(i.length < 4){
                continue
            }else if((i.subSequence(0,2) == "a ") or
                     (i.subSequence(0,1) == ">") or
                     (i.subSequence(0,4) == "mail") or
                     (i == "ahref=")) {
                continue
            }else if ((i.subSequence(0,1) == "/") or
                      (i.subSequence(0,4) != "http")){
                links.add(base_url+"/"+i)
            } else {
                links.add(i)
            }
        }
    }
    return links
}

fun draw_graph(base_url: String, depth: Int): String {
    /*
     * Constructs the .dot graph spec from a url to a specific depth
     *
     * @param base_url: the url to start at
     * @param depth: some integer for the spelunking distance of the graph
     * @return the global variable of the graphs spec for easy passing
     */
    var current_set_of_urls: MutableList<MutableList<String>> = \
    mutableListOf(mutableListOf(base_url))
    var next_set_of_urls: MutableList<MutableList<String>> = \
    mutableListOf(mutableListOf())
    var life: Int = depth
    while(life > 0){
        for(urls in current_set_of_urls){
            next_set_of_urls = link_urls(urls)
            current_set_of_urls = next_set_of_urls
        }
        life--
    }
    return graph_spec
}

fun link_urls(urls: MutableList<String>): MutableList<MutableList<String>> {
    /*
     * Runs through a list of urls, formats from in .dot graph, and
     * find all of new urls links
     *
     * @param urls: a list of urls
     * @return a list of lists of urls from each sub url that is parsed.
     */
    var links_to_do_next: MutableList<MutableList<String>> = \
    mutableListOf(mutableListOf())
    for(url in urls){
        var sub_urls = get_exact_links(get_anchors(get_url_content(url)), url)
        links_to_do_next.add(sub_urls)
        for(link in sub_urls) {
            val rule: String = '"' + url + "\" -> \"" + link + "\";\n"
            if(graph_spec.contains(rule)){
                continue
            }else{
                graph_spec += rule
            }
        }
    }
    return links_to_do_next
}
```

fracingSumFinder.py:

```python
"""
fracingSumFinder.py
By Logan Davis

The program is to answer the following test question:

   What permutation of the digits from 1 to 9 will add to 1
   when arranged in a form similar to 1/23 + 4/56 + 7/89 ?
   Find the answer with a brute force search.

TO-USE:
   interactive_mode: $python fracingSumFinder.py
       This mode will prompt you for a list of integers and
       a target sum. The answers will be printed.
   programmatic_mode:
       This code can be imported like any other library, but
       you will need to set the target and base set of numbers
       manually. See the documentation of the fracingSumFinder
       class for more information.

4/11/17 | Python 3.6 | MIT license
"""
```

```
import itertools

class fractionSumFinder(object):
    """
    This class implements a solution to the fraction sum
    problem specified at the top of this file.

    Either the program can be run as is in interactive mode
    (with user prompts to set the target and base collection
    of integers to use) or imported and invoked
    programmatically.

    ARGS:
    -----------------------------------------------------------------------
     - target: the target some of the fraction sums
     - set_to_use: the set of integers to generate
                   permutations for constructing
                   fractions. Must be 9 in length
    """
    def __init__(self, target=None, set_to_use=None):
        self.target = target
        self.set = set_to_use

    def find_target(self):
        """
        Computes all unique sets of self.set and
        attempts to find some permutation that,
        in the scheme specified in the the docstring
        for self.nineSetDivide, sums to self.target

        >>> sumFinder = fractionSumFinder(1,[1,2,3,4,5,6,7,8,9])
        >>> sumFinder.find_target()
        TARGET OF 1 FOUND: (5, 3, 4, 7, 6, 8, 9, 1, 2)
        TARGET OF 1 FOUND: (5, 3, 4, 9, 1, 2, 7, 6, 8)
        TARGET OF 1 FOUND: (7, 6, 8, 5, 3, 4, 9, 1, 2)
        TARGET OF 1 FOUND: (7, 6, 8, 9, 1, 2, 5, 3, 4)
        TARGET OF 1 FOUND: (9, 1, 2, 5, 3, 4, 7, 6, 8)
        TARGET OF 1 FOUND: (9, 1, 2, 7, 6, 8, 5, 3, 4)

        """
        if (self.set == None) or (self.target == None):
            print("Please set both self.target and self.set and try again.")
            quit()
        target_found = False
        perm = itertools.permutations(self.set)
        for permutation in perm:
            answer = self._nineSetDivideSum(permutation)
            if answer == self.target:
                print("TARGET OF {} FOUND: {}".format(self.target, permutation))

    def _nineSetDivideSum(self, nine_set):
        """
        Takes a set of length 9 and returns the
        collection after running the following
        equation on it:

        - separate the set into three sets of length 3
        - computer the following for each subset of 3:
            - <first element>/<second ele in the
            ten's place><third ele in the singles place>
            I.E. (3 4 5) becomes 3/45
        - sum the results of the three subsets

        The sum of the three subsets is what is returned.

        ARGS:
        ----------------------------------------------
         - nine_set: a set of nine integers

        >>> sumFinder = fractionSumFinder()
        >>> known_set1 = [5, 3, 4, 7, 6, 8, 9, 1, 2]
        >>> sumFinder._nineSetDivideSum(known_set1)
        1.0

        """
        return (nine_set[0]/(nine_set[1]*10+nine_set[2]))+\
               (nine_set[3]/(nine_set[4]*10+nine_set[5]))+\
               (nine_set[6]/(nine_set[7]*10+nine_set[8]))

    def get_user_set(self):
        """
        Gets the set of integer to test as user input that has to
        consist of exactly 9 integers separated by spaces. If the user input
        not of length 9 or solely made up of integers, the program will complain
        and exit.

        The target for the set to sum to is some gathered and parsed. If parsing fails
        (I.E. the user put in something that was not an integer), the program quits.
        """
        set_as_string = input("Please enter 9 integers (no commas for separation): ")
        target_as_string = input("Please enter a target sum (as some integer): ")
        set_as_ints = []
        try:
            set_as_string = set_as_string.replace(" ", "")
            for char in set_as_string:
                set_as_ints.append(int(char))
            if len(set_as_ints) != 9:
                raise AssertionError
        except AssertionError:
            print("Please enter a list of 9 integers. you only entered {}"\
                  .format(len(set_as_ints)))
            quit()
        except ValueError:
            print("Please only enter integers")
            quit()
        self.set = set_as_ints

        try:
            self.target = int(target_as_string)
        except ValueError:
            print("Please enter some integer for the target.")
            exit()




if __name__ == "__main__":
    import doctest
    doctest.testmod()

    sumfinder = fractionSumFinder()
    sumfinder.get_user_set()
    sumfinder.find_target()
```

webgraph.py:

```python
"""
webgraph.py
By Logan Davis

This program is to answer the following question:

B) web crawler visualization

```

```
   Write a program which will first build a network of URLs
   and the links between them by starting at a given web page
   and following its links outwards, and then generate a visual
   representation of that network.

   All the details are up to you, including which page to start
   at, how far to go, and how to display the result.

DESCRIPTION:
   Constructs a .dot graph spec from a base url tracing the
   links from that base page to a specified depth.

   The produced .dot file can be made into a visual graph
   using either graphviz or gephi.

   For an example, here is how to make the output'ed spec
   into a PNG:

   $ dot -Tpng <the output from this script> -O

TO-USE:
   This file can either be run directly, which will start a
   prompt to construct the graph, or imported so you can utilize
   it in some other script.

TO-DO:
   Some parsing problems happen with newlines are in urls which
   are used as a url, and fails to be reached.

4/11/17 | python 3.5 | MIT license
"""
import urllib.request

class page_grabber(object):
    """
    Requests a webpage and returns the response as a string.
    """

    def __init__(self):
        self.request_sent = False
        self.response = ""

    def _send_get_request(self, url):
        """
        Gets a webpage and parses it as a str. Prints out if a
        request is successful or whether there was an error.

        >>> page = page_grabber()
        >>> page._send_get_request("http://csmarlboro.org/\
        ldavis/webscrape_test.html")
        [GOOD]: url http://csmarlboro.org/ldavis/webscrape_test.html \
        responded properly
        >>> page.response
        'b\\'<a href="http://www.test-url.com"> this page is for \
        testing webscrapers </a>\\\\n\\''

        """
        try:
            response_object = urllib.request.urlopen(url)
            self.response = str(response_object.read())
            print("[GOOD]: url {} responded properly".format(url))
        except urllib.error.HTTPError as err:
            print("[WARNING]: {} response error -> {}".format(url, err))
        except urllib.error.URLError as err:
            print("[ERROR]: {} url error -> {}".format(url, err))

    def get_page_from_url(self, url):
        """
        A wrapper for _send_get_request that also
        returns the result for assignment.

        >>> page = page_grabber()
        >>> result = page.get_page_from_url("http://csm\
        arlboro.org/ldavis/webscrape_test.html")
        [GOOD]: url http://csmarlboro.org/ldavis/webscrape_test.html \
        responded properly
        >>> result
        'b\\'<a href="http://www.test-url.com"> this page is for \
        testing webscrapers </a>\\\\n\\''

        """
        self._send_get_request(url)
        return self.response


class html_link_parser(object):
    """
    Parses html requests and extracts any h-ref
    starting with http, http, ftp, and will
    correct any any partial url (such as a relative
    path on a server).
    """

    def parse(self, response, url_base):
        """
        Returns all links from a given html-as-string.

        ARGS:
        ---------------------------------------------
         - response: some html page as a string
         - url_base: the url that the response was from

        >>> url_grabber = page_grabber()
        >>> parser = html_link_parser()
        >>> url = "http://csmarlboro.org/ldavis/webscrape_test.html"
        >>> content = url_grabber.get_page_from_url(url)
        [GOOD]: url http://csmarlboro.org/ldavis/webscrape_test.html \
        responded properly
        >>> parser.parse(content, url)
        ['http://www.test-url.com']
        """
        anchors = self._get_anchors(response)
        return self._extract_urls(anchors, url_base)

    def _get_anchors(self, response):
        """
        Gets the actual anchor tags from an html-as-string.

        ARGS:
        --------------------------------------------
         - response: some html as a string
        """
        anchor_list = []
        tmp = response.split("<")
        for item in tmp:
            if item[0:2] == "a ":
                anchor_list.append(item)
        return anchor_list

    def _extract_urls(self, anchor_list, url_base):
        """
        Gets the actual url from an anchor tag's href
        attribute.

        ARGS:
```

```
        ----------------------------------------------
         - anchor_list: a list of anchor tag string
         - url_base: the base url that all the anchors are
                     from.
        """
        links = []
        for anchor in anchor_list:
            tmp = anchor.split('"')
            for i in tmp:
                if len(i) < 2 or (i[0:4] == "mail"):
                    continue
                if i[0:2] != "a " and i[0] != ">":
                    if i[0:4] != "http" and\
                       i[0:3] != "ftp" and\
                       i[0:5] != "https" and\
                       i[0:3] != "www":
                        links.append(url_base+i)
                    else:
                        links.append(i)
        return links


class link_grapher(object):
    """
    Creates a .dot graph pec from a url that
    follows and marks all links from each parsed page
    to a given depth.
    """
    def __init__(self):
        self.url_grabber = page_grabber()
        self.parser = html_link_parser()
        self.graph_spec = ""

    def run(self, url, depth, output_name):
        """
        A wrapper that will go from grabbing all the urls
        to writing out the spec.

        ARGS:
        ---------------------------------------------------
         - url: the base url that you want to start at
         - depth: the depth that you want to go through web links
         - output_name: the path and name of the output'ed .dot file
        """
        life = depth
        urlsets = [[url]]
        urlset_next = []
        while life >= 0:
            for urls in urlsets:
                urlset_next = self._link_urls(urls)
            urlsets = urlset_next[::]
            life -= 1
        self.write_out_graph(output_name)

    def _link_urls(self, urls):
        """
        Consumes a list of urls, gathers the next set of links to
        parse, and appends the .dot graph statements to self.graph_spec

        ARGS:
        ----------------------------------------------------------
         - urls: a list of urls to parse and format

        >>> linker = link_grapher()
        >>> linker._link_urls(["http://csmarlboro.org/ldavis/\
        webscrape_test.html"])
        [GOOD]: url http://csmarlboro.org/ldavis/webscrape_test.html \
        responded properly
        [['http://www.test-url.com']]
        """
        links_to_do_next = []
        for url in urls:
            content = self.url_grabber.get_page_from_url(url)
            links = self.parser.parse(content, url)
            links_to_do_next.append(links)
            for link in links:
                graph_rule = '"'+url+'" -> "'+link+'";\n'
                if not graph_rule in self.graph_spec:
                    self.graph_spec += graph_rule

        return links_to_do_next

    def get_full_graph(self):
        """
        Gets the generated graph spec with the proper header and footer for
        .dot graph files.
        """
        return " digraph webGraph{\n" + self.graph_spec + "}"

    def write_out_graph(self, output_name):
        """
        Writes self.graph_spec to a file.

        ARGS:
        -----------------------------------------------------------------
         - output_name: the name and path of the output'ed .dot file.
        """
        output = open(output_name, "w")
        output.write(self.get_full_graph())
        output.close()

class grapher_input(object):
    """
    Handles getting user input for the link_grapher
    """
    def get_graph_spec(self):
        """
        Prompts the user for a url, a depth, and an output path.
        All of these values are returns in a triple return as strings,
        with the exception of depth, with is parsed to an integer.
        """
        url = input("Please enter a full url (protocol included): ")
        depth = input("Please enter the depth that you want the grapher to explore: ")
        outputname = input("Please enter the output name \
        you want to give the .dot file: ")

        try:
            depth = int(depth)
        except ValueError:
            print("Please enter an integer for depth...")
            quit()

        return url, depth, outputname


if __name__ == "__main__":
    import doctest
    doctest.testmod()

    grapher = link_grapher()
    input_grabber = grapher_input()
    url, depth, output = input_grabber.get_graph_spec()
    grapher.run(url, depth, output)
```

fracSumFinder.rkt:

```
#lang racket
#|
fracSumFinder.rkt
By Logan Davis

The program is to answer the following test question:

   What permutation of the digits from 1 to 9 will add to 1
   when arranged in a form similar to 1/23 + 4/56 + 7/89 ?
   Find the answer with a brute force search.

TO-USE:
   Feel free to import this into a script and use (find-fraction-sum)
   or run fracSumFinderInteractive.rkt for an prompted version.

4/11/17 | Racket 6.8 | MIT license
|#

; Finds permutations of a set (when used in the calulcation
; from the question) sum to a specified target.
(define find-fraction-sum
  (lambda (target set-to-permute)
    (define permutation-list (permutations set-to-permute))
    (define results (map test-permutation
                         (build-list (length permutation-list)
                                     (lambda (x) 1)) permutation-list))
    (define answers (filter non-empty? results))
    answers))

; returns true if a passed collection is empty, false otherwise
(define non-empty?
  (lambda (item)
    (not (empty? item))))

; test to see if a set, when computed as the problems states,
; sums to a target number. If is does, return to permutation,
; otherwise, return an empty cons
(define test-permutation
  (lambda (target permutation)
    (if (equal? (get-fraction-sum permutation) target)
        permutation
        '())))
```

```
; implements the set-fraction-sum specified in the
; question. Returns the result.
(define get-fraction-sum
  (lambda (permutation)
    (+ (/ (first permutation) (+ (* (second permutation) 10) (third permutation)))
       (/ (fourth permutation) (+ (* (fifth permutation) 10) (sixth permutation)))
       (/ (seventh permutation) (+ (* (eighth permutation) 10) (ninth permutation))))))

; Parses a char value to an integer.
(define parse-char-as-int
  (lambda (item)
    (- (char->integer item) 48)))

(provide (all-defined-out))
```

fracSumFinderInteractive.rkt:

```
#lang racket
#|
fracSumFinderInteractive.rkt
By Logan Davis

The program is to answer the following test question:

   What permutation of the digits from 1 to 9 will add to 1
   when arranged in a form similar to 1/23 + 4/56 + 7/89 ?
   Find the answer with a brute force search.

TO-USE:
   interactive_mode: $ racket fracSumFinderInteractive.rkt
       This mode will prompt you for a list of integers and
       a target sum. The answers will be printed.

4/11/17 | Racket 6.8 | MIT license
|#
(require "fracSumFinder.rkt")

(printf "Please enter the target sum you want: ")
(define target (string->number (read-line)))
(printf "Please enter 9 integers separated by spaces: ")
(define permute-set (map parse-char-as-int
      (string->list (string-replace (read-line) " " ""))))
(define answers (find-fraction-sum target permute-set))

(printf "The answers for target ~a are ~a.\n" target answers)
```

fracSumFinderTest.rkt:

```
#lang racket
#|
fracSumFinderTest.rkt
By Logan Davis

A test for frackSumFinder.rkt

TO-USE:
    $ racket fracSumFinderTest.rkt

4/11/17 | Racket 6.8 | MIT License
|#
(require rackunit)
(require "fracSumFinder.rkt")

; math tests
(check-equal? (get-fraction-sum '(9 1 2 5 3 4 7 6 8)) 1 "Didn't get a\
 sum of 1 on a known 1-sum set.")
(check-equal? (test-permutation 1 '(9 1 2 5 3 4 7 6 8)) '(9 1 2 5 3 4 7 6 8))
(check-equal? (test-permutation 1 '(1 1 1 1 1 1 1 1 1)) '())

;logic tests
(check-equal? (non-empty? '(1)) #t)
(check-equal? (non-empty? '()) #f)

;translation test
(check-equal? (parse-char-as-int #\1) 1)
(check-equal? (parse-char-as-int #\9) 9)

(printf "Done testing!\n")
```

webScraperInteractive.rkt:

```
#lang racket
#|
webscraper.rkt
By Logan Davis

This program is to answer the following question:
   web crawler visualization

   Write a program which will first build a network of URLs
   and the links between them by starting at a given web page
   and following its links outwards, and then generate a visual
   representation of that network.

   All the details are up to you, including which page to start
   at, how far to go, and how to display the result.

DESCRIPTION:
  This is a wrapper to interact with webscraper.rkt through interactive
  prompts instead of raw code. The .dot spec that webscraper.rkt produces
  (see its documentation) is saved to an output file.

  The resulting .dot graph spec can be made into a visual graph
  by programs like CL tools like graphviz or GUI tools like Gephi

  To compile the spec to a PNG with graphviz, issue the following command:
  $ dot -Tpng <the .dot spec> -O

TO-USE:
  $ racket webScraperInteractive.rkt
  # than follow the prompts.

4/11/17 | Racket 6.8 | MIT license
|#
(require "webscraper.rkt")

; get user sepcs
(printf "Please enter a fully qualified url (protocol and all) that you\
 would like to start at: ")
(define url (read-line))
(printf "Now enter the number of pages you want to crawl through: ")
(define depth (string->number (read-line)))
(printf "Please enter where you want to .dot to be saved: ")
(define output-path (read-line))

; check if "depth" parsed correctly
(if (not depth)
  (and (printf "please enter an integer for depth: ") (exit))
  (printf "Okay, starting to generate graph...\n"))

; run and save
(with-output-to-file  output-path #:exists 'replace
  (lambda ()
    (printf (build-web-graph url depth))))

(printf "Done!\n")
```

webscraper.rkt:

```
#lang racket
#|
webscraper.rkt
By Logan Davis

This program is to answer the following question:

   web crawler visualization

   Write a program which will first build a network of URLs
   and the links between them by starting at a given web page
   and following its links outwards, and then generate a visual
   representation of that network.

   All the details are up to you, including which page to start
   at, how far to go, and how to display the result.

DESCRIPTION:
  This is a series of functions to walk through webpage
  links and constructs a .dot graph specification
  (https://en.wikipedia.org/wiki/DOT_(graph_description_language))
  to represent the links from different webpages.

  The resulting .dot graph spec can be made into a visual graph
  by programs like CL tools like graphviz or GUI tools like Gephi

  To compile the spec to a PNG with graphviz, issue the following command:
  $ dot -Tpng <the .dot spec> -O

TO-USE:
  Import into another script and call build-web-graph passing it
  a url and a depth to walk. The .dot graph will be returned.


4/11/17 | Racket 6.8 | MIT license
|#
(require net/url)
```

```
(require html-parsing)


(define non-empty?
  (lambda (item)
    (not (empty? item))))

; returns a flat-mapped collection of all h-refs including
; complete urls from a cons-parsed webpage
(define get-hrefs
  (lambda (datum)
    (cond [(or (string? datum) (symbol? datum)) '()]
          [(equal? (first datum) 'href) (second datum)]
          [else (filter-out-incomplete-urls
                (flatten (map get-hrefs (rest datum))))])))

; filters all non-http and non-https, completely specified urls
; IE: not "../dir/some.html" or "ftp://somedownload.com/example"
;     but it will return "http://cs.marlboro.edu"
(define filter-out-incomplete-urls
  (lambda (urls)
    (filter (lambda (x) (string-contains? x "http")) urls)))

; Consumes an http-pipe and returns all href elements.
(define parse-hrefs
  (lambda (html-pipe)
    (filter non-empty? (get-hrefs (html->xexp html-pipe)))))

; spelunks all complete h-refs from a given url to a
; specified tree depth and returns a .dot graph spec
; corresponding to the h-ref walk.
(define make-node-and-continue
  (lambda (url depth)
    (if (> depth 0)
        (let ([node-name url]
              [nodes (parse-hrefs (get-pure-port (string->url url)))])
          (define branch (string-join
                          (map (lambda (origin dest)
                          (string-append "\"" origin "\" -> \"" dest "\";\n"))
                               (make-list (length nodes) node-name) nodes)))
          (string-append branch (string-join (map make-node-and-continue
                                                  nodes (make-list (length nodes)
                                                  (- depth 1))))))
        "")))
```

```
; A wrapper for made-node-and-continue
; take a url and depth, walks h-refs on the url
; and returns a .dot graph spec for that link walk.
(define build-web-graph
  (lambda (base-url depth)
    (make-node-and-continue base-url depth)))

(provide (all-defined-out))
```

webscraperTest.rkt:

```
#lang racket
#|
webscraperTest.rkt
By Logan Davis

A test for webscraper.rkt

TO-USE:
    $ racket webscraperTest.rkt

4/11/17 | Racket 6.8 | MIT License
|#

(require rackunit)
(require "webscraper.rkt")

;logic tests
(check-equal? (non-empty? '(1)) #t)
(check-equal? (non-empty? '()) #f)

; parsing tests
(check-equal? (filter-out-incomplete-urls '("../example"
    "http://example.come" "https://example.come"))
              `("http://example.come" "https://example.come"))

(printf "Done with testing.\n")
```

pegSolver.py:

```python
"""
pegSolver.py
by Logan Davis
```

```
This script is to answer the following question:
-------------------------------------------------------------
Illustrate a depth-first and breadth-first tree search,
preferably using a stack and queue, with the tree of
possible moves in a triangular peg solitaire game as the tree.
The game I have in mind starts with this configuration
     .
    * *
   * * *
  * * * *
 * * * * *
where each * is a peg in a hole, and the . is an empty
hole. The goal is to remove pegs by jumping as in
checkers, leaving one in the center as the final position.
-------------------------------------------------------------

This file is full of utilities and classes to solve any
sized game of triangular peg solitaire. Here is an example
being used to some a game board of size 5:

>>> solver = peg_board_solver()
>>> solver.play_depth(5)
Starting search with board:
     .
    * *
   * * *
  * * * *
 * * * * *
Searching...
A winner was found:
     .
    * *
   * * *
  * * * *
 * * * * *
TURN 0: move (x:2, y:2) to (x:0, y:0) and remove (x:1, y:1).
     *
    * .
   * * .
  * * * *
 * * * * *
TURN 1: move (x:2, y:4) to (x:2, y:2) and remove (x:2, y:3).
     *
    * .
   * * *
  * * . *
 * * . * *
TURN 2: move (x:4, y:4) to (x:2, y:4) and remove (x:3, y:4).
     *
    * .
   * * *
  * * . *
 * * * . .
TURN 3: move (x:1, y:4) to (x:3, y:4) and remove (x:2, y:4).
     *
    * .
   * * *
  * * . *
 * . . * .
TURN 4: move (x:3, y:3) to (x:1, y:1) and remove (x:2, y:2).
     *
    * *
   * * .
  * * . .
 * . . * .
TURN 5: move (x:0, y:2) to (x:2, y:4) and remove (x:1, y:3).
     *
    * *
   . * .
  * . . .
 * . * * .
TURN 6: move (x:3, y:4) to (x:1, y:4) and remove (x:2, y:4).
     *
    * *
   . * .
  * . . .
 * * . . .
TURN 7: move (x:0, y:4) to (x:2, y:4) and remove (x:1, y:4).
     *
    * *
   . * .
  * . . .
 . . * . .
TURN 8: move (x:1, y:1) to (x:1, y:3) and remove (x:1, y:2).
     *
    * .
   . . .
  * * . .
 . . * . .
```

```
TURN 9: move (x:0, y:0) to (x:0, y:2) and remove (x:0, y:1).
     .
    . .
   * . .
  * * . .
 . . * . .
TURN 10: move (x:0, y:3) to (x:0, y:1) and remove (x:0, y:2).
     .
    * .
   . . .
  . * . .
 . . * . .
TURN 11: move (x:2, y:4) to (x:0, y:2) and remove (x:1, y:3).
     .
    * .
   * . .
  . . . .
 . . . . .
TURN 12: move (x:0, y:2) to (x:0, y:0) and remove (x:0, y:1).
     *
    . .
   . . .
  . . . .
 . . . . .
Done!

The solver can either do a breadth or depth search for the solution.
Which one is used depends on whether you start the searcher by either
calling "play_depth" or "play_breadth".

The Solver can also be given a state iteration limit and made to
produce a dot-graph spec for visual reference:

>>> solver = peg_board_solver()
>>> solver.make_dot_spec = True
>>> solver.step_limit = 10
>>> solver.play_depth(5)
Starting search with board:
     .
    * *
   * * *
  * * * *
 * * * * *
Searching...
Halted due to step limit of 10 being reached.
```

```
>>> print(solver.dot_writer.get_graph())
digraph peggame {
"0"->"1";
"0"->"2";
"2"->"3";
"2"->"4";
"2"->"5";
"2"->"6";
"6"->"7";
"6"->"8";
"6"->"9";
"9"->"10";
"9"->"11";
"9"->"12";
"9"->"13";
"9"->"14";
"9"->"15";
"15"->"16";
"15"->"17";
"15"->"18";
"15"->"19";
"15"->"20";
"20"->"21";
"20"->"22";
"20"->"23";
"23"->"24";
"23"->"25";
"23"->"26";
"23"->"27";
"23"->"28";
"28"->"29";
"28"->"30";
"28"->"31";
"28"->"32";
"32"->"33";
"32"->"34";
"34"->"35";
"35"->"36";
}

This resulting spec can be compiled using a command like the following:

        $ dot -Tpng <source spec> -O
```

```
4/26/17 | Python 3.5 | MIT license
"""
import copy, math

class symbol_table(object):
    """
    Continually generates any number of integers-as-string
    for node naming purposes.
    """
    def __init__(self):
        self.symbols = range(0,100).__iter__()
        self.cap = 100
        self.current = 0

    def next(self):
        """
        returns the next symbol in the stream
        of labels.
        """
        self._check_if_end()
        self.current += 1
        return str(self.symbols.__next__())

    def _check_if_end(self):
        """
        Checks to see if the symbol stream is nearing its
        end. If so, extend the symbol range.
        """
        if self.current == self.cap-1:
            self.symbols = range(self.cap, self.cap*2).__iter__()
            self.cap = self.cap*2

class peggame_dot_writer(object):
    """
    Compiles a dot-graph spec
    for a pegboard game.
    """

    def __init__(self):
        self.dot_spec = ""

    def add_dot_edge(self, origin, dest):
        """
        adds a rule to the compiled dot spec
        of <origin> -> <dest>.

        ARGS:
        ------------------------------------
         - origin: the origin node id of the edge
         - dest: the destination of the edge
        """
        self.dot_spec += "\""+origin +"\""+ "->" + "\""+dest+"\";\n"

    def get_graph(self):
        """
        returns the dot spec as a string with
        proper file heading and footer.
        """
        return "digraph peggame {\n"+self.dot_spec+"}"

class peg_board(object):
    """
    A pegboard object for triangular peg solitaire.
    Contains game state and move history.

    ARGS:
    -----------------------------------------------
     - rows: the number of rows you want the peg board
             to have
    """

    def __init__(self, rows):
        self.rows = rows
        self.board = self.construct_board(rows)
        self.history = []


    def construct_board(self, rows):
        """
        Constructs an empty board

        ARGS:
        --------------------------
         - rows: The number of rows
                 to generate.
        """
        board = []
        for i in range(1, rows+1):
            board.append(list("."*i))
```

```python
        return board

    def init_board(self):
        """
        Fills the board with pegs except the
        very top peg
        """
        for i in range(1,self.rows):
            for x in range(0,len(self.board[i])):
                self.board[i][x] = "*"

    def print_board(self):
        """
        Pretty prints the board.
        """
        offset = self.rows
        for i in self.board:
            print((" "*offset)+" ".join(i))
            offset -= 1

    def print_history(self):
        """
        Given a history, the board with
        print out each stage of the game
        stored in self.history along
        with coordinates to show movement
        of pegs.
        """
        mock_board = peg_board(self.rows)
        mock_board.init_board()
        turn_counter = 0
        for move in self.history:
            y, x = move[0]
            y_off, x_off = move[1]
            y_kill, x_kill = move[2]

            mock_board.print_board()
            print("TURN {}: move (x:{}, y:{}) to (x:{}, y:{}) and remove (x:{}, y:{}).'
                                                  .format(turn_counter,
                                                          x, y,
                                                          x+x_off,y+y_off,
                                                          x+x_kill,y+y_kill))
            mock_board.board[y][x] = "."
            mock_board.board[y+y_off][x+x_off] = "*"
            mock_board.board[y+y_kill][x+x_kill] = "."
            turn_counter += 1
        mock_board.print_board()
        print("Done!")

class peg_board_solver(object):
    """
    Solves a game of triangular peg solitaire
    If a winning strategy is found, the moves to
    make such a play are printed out.
    """

    def __init__(self):
        self.boards_to_do = []
        self.search_mode = "breadth"
        self.step_limit = math.inf
        self.make_dot_spec = False
        self.dot_writer = peggame_dot_writer()
        self.symbol_table = symbol_table()
        self.solved = False


    def get_pegs_moves(self, board, x, y):
        """
        Returns a list of the pegs origin,
        each valid offset to move to, and the
        offset to the peg that would need to
        be removed to make the aforementioned
        jump.

        ARGS:
        --------------------------------------
         - board: the board to look at.
         - x: the number of peg holes to the
              right that you want to look at.
         - y: the row you want to look at.
        """
        offsets = [[(-2, -2), (-1, -1)],
                   [(-2,  0), (-1,  0)],
                   [( 0, -2), ( 0, -1)],
                   [( 0,  2), ( 0,  1)],
                   [( 2,  0), ( 1,  0)],
                   [( 2,  2), ( 1,  1)]]
        valid_moves = []
        for off_set in offsets:
            if (y+off_set[0][0] < 0) or (x+off_set[0][1] < 0):
                continue
            try:
                if (board[y][x] == "*") and\
                   (board[y+off_set[0][0]][x+off_set[0][1]] == ".") and\
                   (board[y+off_set[1][0]][x+off_set[1][1]] == "*"):
                    valid_moves.append([(y,x)]+off_set+[id(board)])
            except IndexError: #off the board
                continue
        return valid_moves

    def realize_board_step(self, board, move):
        """
        Given a board and a move, actually
        creates a new board based off the old one
        and commits the move to the new boards
        game state.

        ARGS:
        ------------------------------------------
         - board: the board to commit moves to
         - move: a set of origin and offsets to
                 use to commit the move.
        """
        new_board = copy.deepcopy(board)
        y_origin, x_origin = move[0]
        y_dest_off, x_dest_off = move[1]
        y_kill_off, x_kill_off = move[2]
        new_board[y_origin][x_origin] = "."
        new_board[y_origin+y_dest_off][x_origin+x_dest_off] = "*"
        new_board[y_origin+y_kill_off][x_origin+x_kill_off] = "."
        return new_board

    def step_board(self, board_obj):
        """
        Processes a considered board and catalogs any
        valid game state that could be made from a move
        available on the given board.

        ARGS:
        -----------------------------------------------
         - board_obj: the board to find all move
                      from.
        """
        moves = []
        for row in range(0,len(board_obj.board)):
            for peghole in range(0,len(board_obj.board[row])):
                valid_moves = self.get_pegs_moves(board_obj.board, peghole, row)
                if  valid_moves != []:
                    moves.append(valid_moves)
        for origin_set in moves:
            for move in origin_set:
                new_board_state = self.realize_board_step(board_obj.board, move)
                new_board = peg_board(len(new_board_state))
                new_board.name = self.symbol_table.next()
                new_board.board = copy.deepcopy(new_board_state)
                new_board.history = copy.deepcopy(board_obj.history+[move])

                if self.make_dot_spec:
                    self.dot_writer.add_dot_edge(board_obj.name, new_board.name)

                if self.winner_check(new_board):
                    self.winning_path_found(new_board)
                else:
                    self.boards_to_do.append(new_board)

    def winner_check(self, board_obj):
        """
        Checks to see if a board is
        in a winning state.

        ARGS:
        ---------------------------
         - board_obj: the board to
                      check if in a
                      winning state.
        """
        if board_obj.board[0][0] != "*":
            return False
        for i in range(1,board_obj.rows):
            for peghole in board_obj.board[i]:
                if peghole == "*":
                    return False
        return True

    def winning_path_found(self, board_obj):
        """
        Handles win condition

        ARGS:
        ----------------------
```

```
         - board_obj: A winning board
        """
        print("A winner was found:")
        board_obj.print_history()
        self.solved = True

    def get_mode(self):
        """
        returns a pop index
        depending on when search
        mode the solver is set
        to.
        """
        if self.search_mode == "depth":
            pop_index = -1
        elif self.search_mode == "breadth":
            pop_index = 0
        else:
            print("Unrecognized searchmode state {}. Defaulting to breadth-first.".forr
            pop_index = 0

        return pop_index

    def find_winner(self, board):
        """
        Given an initial board, this wraps
        board steps, iteration limit checks,
        and handles non-winnable boards.

        ARGS:
        ------------------------------------
         - board: a starting board to try to
                  find a winning set of moves
                  for.
        """
        print("Starting search with board:")
        board.print_board()
        print("Searching...")
        self.boards_to_do.append(board)

        pop_index = self.get_mode()
        step = 0

        while not self.solved:
            if self.boards_to_do == []:
                print("No winning path was found. Sorry.")
                self.solved = True
                continue
            if self.step_limit < step:
                print("Halted due to step limit of {} being reached.".format(self.step_
                self.solved = True
                continue

            current_board = self.boards_to_do.pop(pop_index) #stack
            self.step_board(current_board)

            step += 1

    def breadth_first_solve(self, board):
        """
        Sets the solver to a breadth first search
        and than call .find_winner

        ARGS:
        -----------------------------------------
         - board: the initial board to try and find
                  a solution for.
        """
        self.search_mode = "breadth"
        self.find_winner(board)

    def depth_first_solve(self, board):
        """
        Sets the solver to a depth first search
        and than call .find_winner

        ARGS:
        -----------------------------------------
         - board: the initial board to try and find
                  a solution for.
        """
        self.search_mode = "depth"
        self.find_winner(board)

    def play_depth(self, size):
        """
        Generates an initial board of
        a given size and than called
        .depth_first_solve
        ARGS:
        ----------------------------
         - size: The size of the board
                  you want to generate
                  and find an answer for
        """
        start_board = peg_board(size)
        start_board.init_board()
        start_board.name = self.symbol_table.next()
        self.depth_first_solve(start_board)

    def play_breadth(self, size):
        """
        Generates an initial board of
        a given size and than called
        .breadth_first_solve

        ARGS:
        ----------------------------
         - size: The size of the board
                  you want to generate
                  and find an answer for
        """
        start_board = peg_board(size)
        start_board.init_board()
        start_board.name = self.symbol_table.next()
        self.breadth_first_solve(start_board)


if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

prims.py

```
"""
prims.py
by Logan Davis

This script is to answer the following question:
-------------------------------------------------------------
Explain, implement, explore numerically the O() behavior
either Prim's or Kruskal's algorithm, which find the minimum
spanning tree of a weighted graph.
```

```
The core of this work should be a numerical experiment in
which you randomly generate graphs of different sizes, find
either the time or number of steps the algorithm takes, use
those to create plots of its behavior, and compare the shape
of the plot with the expected result.

As usual with my assignments, your code should be clearly
documented with explicit tests. Be clear that you should
choose one of these to do, not both.
-------------------------------------------------------------

So this code contains classes to make directed graphs,
symmetric directed graph generators, and generate prim
min-span graphs.

Graphs are available as both directed_graph objects
and dot graph specs for visualization.

The graph spec generator is made to compile a domain
specific language to construct directed, weighted graphs.
The general form of the language is:

        <origin node> ->(<weight>) <destination node> ;

An example of a cycling triangle graph is as follows:

    a ->(1) b ;
    b ->(2) c ;
    c ->(5) a ;

which is to say that node "a" goe to "b" by weight "1", "b"
goes to "c" by weight "2", and "b" goes to "c" by weight "5".
Here is an example of a hand writtern spec being parsed and
searched:
>>> graph_spec = "a ->(1) b \;b ->(2) c \;"
>>> usable_graph = gen_graph(graph_spec)
>>> usable_graph.parse_spec()
>>> usable_graph.display_dot_spec()
digraph viz{
a -> b [label="1"];
b -> c [label="2"];
}

This resulting dot spec an be compile to a visual graph:
        $ dot -Tpng <source spec> -O

But you can just get a randomly weighted, symmetrically directed
graph of N nodes by using tangled_directed_graph_spec_gen()

>>> test_spec_gen = tangled_directed_graph_spec_gen(5)
>>> test_graph = test_spec_gen.generate_graph()
>>> usable_graph = gen_graph(test_graph)
>>> usable_graph.parse_spec()
>>> searcher = prim_search()
>>> min_span_graph = searcher.search(usable_graph.graph)
>>> len(min_span_graph.node_pool) == 5
True

Due to the random weighting when the graph is generated,
no doctest can be made to concretely show the generated graph
or corresponding dot spec.

4/26/17 | Python 3.5 | MIT license
"""
import math, itertools, random, copy

class node(object):
    """
    A super generic node object to
    construct graphs

    ARGS:
    ----------------------------
     - name: the name used to identify the node
    """
    def __init__(self, name="none"):
        self.name = name
        self.connections = {}
        self.order = []

class directed_graph(object):
    """
    An object to contain a directed graph
    and all of its nodes.
    """
    def __init__(self):
        self.start = None
        self.node_pool = {}
```

```
class gen_graph(object):
    """
    Generates a graph based off of
    a graphing DSL spec
    """
    def __init__(self, spec):
        self.graph_spec = spec
        self.graph = directed_graph()
        self.dot_spec = ""


    def parse_spec(self):
        """
        Parses the input'ed graph spec
        and passes it to self.create_node
        to be constructed.
        """
        rules = list(filter((lambda x: x != ''), self.graph_spec.replace("\n", "").spli
        for rule in rules:
            parts = rule.split(" ")
            origin = parts[0]
            edge_weight = self._get_weight_val(parts[1])
            dest = parts[2]
            self.create_node(origin, edge_weight, dest)

    def get_graph(self):
        """
        Gets the compiled graph object
        """
        return self.graph

    def display_dot_spec(self):
        """
        Prints the compilers dot graph
        spec of the compiler graph
        """
        print("digraph viz{\n"+self.dot_spec+"}")

    def create_node(self, start, weight, end):
        """
        Commits a graph rule to both the graph
        object and the dot spec of the compiler.

        ARGS:
```

```
        ----------------------------------------
         - start: The origin node
         - weight: the weight of this edges connection
         - end: the destination for this added edge
        """
        if start in self.graph.node_pool:
            if end in self.graph.node_pool:
                self.dot_spec += "{} -> {} [label=\"{}\"];\n".format(start, end, weight
                self.graph.node_pool[start].connections[end] = (self.graph.node_pool[er
            else:
                self.graph.node_pool[end] = node(end)
                self.create_node(start, weight, end)
        else:
            self.graph.node_pool[start] = node(start)
            self.create_node(start, weight, end)

    def _get_weight_val(self, weight_part):
        return int(weight_part.split("->")[-1][1:-1])

class tangled_directed_graph_spec_gen(object):
    """
    Generates a symmetric, directed, and randomly
    weighted graph of N nodes.

    ARGS:
    ---------------------------------------------
     - nodes: The number of nodes you want to
              have in the generated graph
    """
    def __init__(self, nodes):
        self.nodes_num = nodes
        self.symbol_table = []

        self.gen_symbol_table()

    def gen_symbol_table(self):
        """
        generates up to 1000 symbols to
        be used in constructing nodes.
        Please be kind to you computer and
        don't try to make a graph larger
        than 1000 nodes.
        """
        i = 0
        for x in range(0, 10):
            for y in range(0, 10):
                for z in range(0, 10):
                    if i >= self.nodes_num:
                        return
                    else:
                        self.symbol_table.append("id_{}{}{}".format(x, y, z))
                    i += 1

    def generate_graph(self):
        """
        Generates the graph DSL
        spec to be parsed and
        returns it.
        """
        if self.symbol_table == []:
            self.gen_symbol_table()
        spec = ""
        for i in range(0, self.nodes_num):
            for x in range(0, self.nodes_num):
                if x == i:
                    continue
                else:
                    spec += "{} ->({}) {};\n".format(self.symbol_table[i],
                                                    random.randint(0, 100),
                                                    self.symbol_table[x])
        return spec


class prim_search(object):
    """
    Implements a Prim's Min Span Tree Search algorithm.
    Either a search based on a starting node can be done
    (.search_anchored_min_span) or the search can start
    at the node with the cheapest edge (.search).

    With either search, the min-span version of the
    original graph is returned as a directed-graph
    object.

    An internal DSL spec and dot spec are stored & accessable
    after a search is done.
    """
    def __init__(self):
        self.currently_considered_nodes = []
        self.visited = []
        self.graph = None
        self.dot_spec = ""
        self.tree_spec = ""

    def search_anchored_min_span(self, graph, starting_node_id):
        """
        Searchs a given graph for a min-span traversal starting
        at a given node. What is returned is a min-span version
        of the original graph object.

        ARGS:
        -------------------------------------------------------
         - graph: the graph to run Prim's on
         - starting_node_id: the name of the node in the
                             graph's node_pool to start at
        """
        self.currently_considered_nodes = [starting_node_id]
        self.visited.append(starting_node_id)
        while len(self.visited) < len(graph.node_pool):
            best_move = ["none", "none", math.inf]
            for current_node in self.currently_considered_nodes:
                for subnode in graph.node_pool[current_node].connections:
                    weight = graph.node_pool[current_node].connections[subnode][1]
                    if subnode in self.visited:
                        continue
                    if best_move[2] > weight:
                        best_move = [current_node, subnode, weight]
            self.currently_considered_nodes.append(best_move[1])
            self.visited.append(best_move[1])

            self.tree_spec += "{} ->({}) {};\n".format(best_move[0],best_move[2],best_r

        compiler = gen_graph(self.tree_spec)
        compiler.parse_spec()
        self.graph = compiler.graph
        self.dot_spec = compiler.dot_spec
        return self.graph

    def display_dot_spec(self):
        """
        prints the min-span tree as a dot graph spec
        with the proper header & footer.
        """
        print("digraph viz{\n"+self.dot_spec+"}")
```

```python
    def search(self, graph):
        """
        Identifies the cheapest edge of a graph
        and than calls .search_anchored_min_span
        on that node and the passed graph.

        ARGS:
        -----------------------------------------
         - graph: the graph to Prim search.
        """
        node_with_cheapest_edge = (None, math.inf)
        for node in graph.node_pool:
            for connection in graph.node_pool[node].connections:
                weight = graph.node_pool[node].connections[connection][1]
                if weight < node_with_cheapest_edge[1]:
                    node_with_cheapest_edge = (node, weight)

        return self.search_anchored_min_span(graph, node_with_cheapest_edge[0])


if __name__ == "__main__":
    import doctest
    doctest.testmod()
```