File: worksheet-blank

Life Conference Class - Introduction to Programming

February 1, 2019

Andrew Gallant
jamslam@gmail.com

Kaitlyn Brady
kbrady39@gmail.com

Vocabulary

If you don't understand all of these at first, that's OK! We'll see examples of them throughout the class.

programming:

> The process of designing and building a program for solving a specific problem.

program:

> A specific sequence of instructions that precisely instruct a computer to do something. Software consists of one of more programs.
>
> Programs are normal files on your computer, just like anything else. The only difference between a program file and other files is that a program file can be turned into a process.

process:

> A single instance of a running program on your computer.

programming language:

> A set of instructions with which to write a program. Source code is made up of these instructions, and the specific way in which one writes the instructions is called the programming language's syntax.

text editor:

> A program that one uses to manipulate the source code for a program.

shell:

> A tool for issuing commands directly to the computer.

What are we building?

Today, we are going to build a spell checker!

A spell checker is a program that reads some text and prints out all of the misspelled words in that text.

Before we get started on our spell checker, we will need to learn how to write and execute programs.

Getting Started

For today, we will be working exclusively in your computer's shell. To open a shell on macOS, do the following:

1. Click on a blank space in the desktop.
2. Make sure it says "Finder" in the top left corner.
3. Click "Go" and select "Utilities" from the menu.
4. Double click on the "Terminal" icon. A shell will open.
5. Move and resize the shell so that it uses the left half of your screen.
6. Open a second shell by clicking "Shell" near the top left corner, and then select "New Window" from the menu. A second shell will now open.
7. Move and resize the second shell so that it uses the right half of your screen.

At this point, you can now issue commands to your computer! Test it out by typing "ls" (without the quotes) and pressing enter. What happens? Type in something else (anything) and press enter. What happens?

Tip: use Command+` to switch between shells quickly using only the keyboard.

Basic commands

In order to build our spell checker, we'll need some text to spell check and a dictionary of words. Let's setup a directory for our project and download these files.

```
$ mkdir spell-checker
$ cd spell-checker
$ curl -O https://burntsushi.net/stuff/life-conference/dictionary
$ curl -O https://burntsushi.net/stuff/life-conference/text
```

If you run

```
$ ls
```

You should see two files in your current directory: dictionary and text.

What do these commands do?

ls -

cd -

mkdir -

curl -

Let's explore the data files we downloaded by learning about more commands:

cat -

wc -

Creating our first program

In order to write a program, we first need to learn how to use a text editor.

There are many different text editors to choose from. Some of them are very advanced and actually help you write programs. In this class, however, we will be using a very simple text editor called nano. You can run it right from your shell:

```
$ nano
```

Once you're in nano, you can start typing your program. So let's write our first one.

Once you've typed in the program, you now need to write the program to a file. You can do that by pressing Control+O. You will be asked to give the name of the file you want to write. Type "program.py" (without the quotes) and press enter. Your program is now saved!

Now all we have to do is run the program. Switch to your other shell, and change into the directory you created before:

```
$ cd spell-checker
```

and now list the contents of the directory:

```
$ ls
```

In addition to the data files you downloaded previously, you should now see a new file "program.py". This means the program has been saved as a file and is ready to run. To run the program, use the following command:

```
$ python program.py
```

What happens?

Syntax

When telling a computer what to do, it is very important to be precise. One small slip-up or one small typo can cause your program to fail. This may happen to you a lot. The process of figuring out what's wrong with your program and how to fix it is called debugging, and it is something that programmers spend a lot of time doing.

Let's see an example of how a program can misbehave. Switch back over to the shell that contains the program you wrote in the previous step. Once there, change "print" to "pint". Save the file by pressing Control+O and hitting enter (you do not need to type the file name again). Now switch back over to your other shell and run your program again:

```
$ python program.py
```

What happens? Experiment with other changes to the program. What is allowed and what isn't?

Variables

Variables are an important aspect of almost all programs. Variables allow you to associate values with a name, and then use that name to reference that value later.

Once you have the example written in your text editor, save the file and run

the program. What do you see?

What happens if you change the variable name? What about the contents of the variable?

If statements

"if" statements are another important aspect of almost all programs. An "if" statement lets you execute something only if a particular condition is true or not.

Loops

Loops are a way to execute something more than once without having to explicitly repeat it. For example, let's say we wanted to print a message 5 times along with the message's number. You could do this:

```
message = "Hello, world!"
print 1, message
print 2, message
print 3, message
print 4, message
print 5, message
```

In this program, we assign our message to a variable, and then print that message 5 times along with its number. With a loop, we can avoid repeating ourselves!

Reading files

Before writing a spell checker, we need to first learn how to read data from other files. Let's try the simplest possible thing: read data from a file and then print it back out again. (Does this sound familiar? This is what the "cat" program does!)

Compare the output of your program with the output of

```
$ cat text
```

What's different? How can we fix it?

Hint: line.strip("\n") will give you a line without the end of line terminator.

Counting words

There's one last thing we need to learn before we can write a spell checker, and that's how to look at words. To start with, let's try writing a version of the "wc" program that counts the total number of words in a file. Start by writing out an algorithm:

1.

2.

3.

4.

5.

Here's a twist. What if instead of counting the total number of words, we wanted to count the total number of unique words? Before trying to write the code let's try to sketch out an algorithm for it:

1.

2.

3.

4.

5.

Now translate the algorithm above into code. Below are a couple hints that you'll need to write the code.

This will create an empty set and assign it to the variable "words":

```
words = set()
```

And this will add a word to the set:

```
words.add("school")
```

This will add a word in a variable to the set:

```
word = "school"
words.add(word)
```

Building a spell checker

Finally, you have everything you need to build a spell checker. As with previous examples, let's try to sketch out an algorithm for our spell checker. Your spell checker should scan the text and print out words that it believes are misspelled.

Hint: To check whether a word is in a set or not, use this:

```
    if word not in dictionary:
```

CHALLENGE PROBLEM

If you've managed to write a spell checker successfully and are looking for an additional challenge, consider modifying the spell checker to print not just the misspelled words, but also print one or more suggestions for a spelling correction. For example, if I write "destinguish", then your program should suggest "distinguish" as a correction.