Computing with Algebra Course Document

Travis N. Miller February 12, 2018

More Variables

I am going to start out by introducing the concept of variable assignment. This is how variables get the data they are "pointed" to.

```
a=2
b=3
c=a*b
print(c)
```

Once again we can run the type on variable and the return will by the type of the data they are assigned.

This is fine if we want to use python as a calculator. But let us take a closer look at strings.

```
name = 'Travis Miller'

first = name[0:6]

last = name[7:]

middle = 'Neal'

name = first + ' ' + middle + ' ' + last

print(name)

print('wow'*3)
```

Note that once reassigned the variable name has no access to what is used to be assigned to. This can be very helpful with that are called recursive functions but we have to be sure we only want an endpoint and not all of the steps.

Take the two variables below an create a new variable named child using a mesh of their two names.
Assignment: I have a friend who is names Adrienne Sweetwater Her parents are Nancy Rainwater and Andy Sweet. Please create a name out of the to variables bellow feel free to add your own variable for a first name. Note: the child's name does not have to be Adrienne.

```
husband = 'Andy Sweet'

wife = 'Nancy Rainwater'
```

If Then Statements

If and then statements are a cornerstone of logic. In programing they allow our software to take different paths based on some information it is being fed. I believe it is good to look at if then statements as trees with branches. When nutrients (information) move up the tree they are analyzed and if they have specific qualities they go up a branch versus staying in the trunk.

```
if cond == met:
    something.is.done
```

Notice the == sign. This is to distinguish between the assignment of a value to a variable and testing if a variable is a certain value. Similarly greater than, greater than or equal to, less than, less than or equal to, and does not equal: >, >=, <, <=, != respectively. When looking at the first line of an if statement it should be noted that conditions are either met or not met. There is no gray area here. These results are called Booleans. They only have two cases True or False. The other thing to notice about the if statement is that the something.is.done is indented(4 spaces exactly) this is for readability and is something that make python unique. It is important to note that nothing indented is run unless the statement above it is satisfied.

Copy and paste the following program into your test.py and run it. Analyze the code as you go through the program in the terminal.

```
color = input('What is your favorite color:')

if color == 'Blue':
    print('Fine, off you go.')
```

Note:The above program includes a built in function in python that takes a string a user inputs is the terminal and in this case stores it in the variable color.

Test some colors. What are some of the issues with this program? The most notable is that if 'Blue' is not the users favorite color than nothing happens. the second is that this program is case sensitive. Meaning that 'Blue' is different than 'blue'.

Let us look at the first, this is where the branches of our tree gets interesting. We have an else function as well. It lies at the same level of the if.

```
if color == 'Blue':
    print('Fine, off you go.')
else:
    print('OFF THE CLIFF')
```

Now let us look at the case sensitivity. There are built in functions we can call on the 'class' of strings. These are called methods. More will be revealed on methods and creating our own when we get to classes. The method we need now is called lower() these methods are called on a string by 'string.lower()' this will return the lowercase of any given character in a string. Some characters do not have a lower such as numbers, @, or spaces.

```
color = input('What is your favorite color:')

if color.lower() == 'blue':
    print('Fine, off you go.')
else:
    print('OFF THE CLIFF')
```

Now we have some input error analysis. And a response for the errors. This error response may not be too helpful for a user but hey at least it is not silent. One of the tenants of the Zen of Python. Add the following into a blank python file and run it.

```
import this
```

Assignment: Create a simple compatibility test.

If someone passes the first compatibility question then they should get another. Embedded if statements these can get hairy so be careful with your whitespace.

```
name = input('What is your name?')

if name is not None:
    print('Good you have a name')
    outdoors = input('Do you like being outside? Y/N')
    if outdoors.lower() == 'y':
        print('Yes, I like the outdoors as well')
        improving = input('How important is improving to you. 1-Low, 10-High')
        if int(improving) > 6:
            print('You seem cool')
        else:
            print('Nobody is perfect. But i would rather go up than down')
    else:
        print('Sorry that is incorrect')
else:
    print('Sorry I do not trust people without names')
```

Listing 1: Example

Loops

The major way that computers beat out humans is they do not seem to get bored doing a task over and over again. Python is very good at going through data and sorting, modifying, verifying, and many more ings. Loops do a task over and over again until they have ran through what they are iterating over. To better understand loops let up look at this example:

```
lst = [1, 2, 3, 4, 5]
for elm in list:
    print(elm)
```

Listing 2: For Loop

As you can see this just prints all of the elements in a list. The below loop will produce the exact same outcome with a different type of loop.

```
i = 0
while i <= 5:
    print(i)
    i += 1
```

Each of these loops have there own strengths. The for is nice to go through elements of data. While loops are nice because they have a dynamic quality where they end when a value is reached.

```
import random
num = random.randint(1,10)
guess = int(input('Guess my number:'))
while guess != num:
    guess = int(input('Guess again!'))

print('YOU GOT IT!')
```

A for loop is a good option for checking elements in a object. Sting, dictionary, list. Let us look at an example of this.

```
lst = [3, 6, 9, 12, 15, 18, 21]
time_lst = []
for hour in lst:
    if hour == 12:
        noon = str(hour) + ' PM'
        time_lst.append(noon)
    elif hour > 12:
        hour = hour - 12
        PM = str(hour) + ' PM'
        time_lst.append(PM)
    else:
        AM = str(hour) + ' AM'
        time_lst.append(AM)
```

```
print(time_lst)
```

Assignment:

Create a countdown timer that takes an input from the console and produces a printed bomblike countdown. You will need to use the sleep method. Run the example below to see how this method works.

```
import time
print("1 and ...")
time.sleep(1)
print("2 and ...")
time.sleep(3)
print("5")
```

Definitions:

Iterate: Each repetition of the process is also called an "iteration", and the results of one iteration are used as the starting point for the next iteration. In the context of mathematics or computer science, iteration (along with the related technique of recursion) is a standard building block of algorithms.

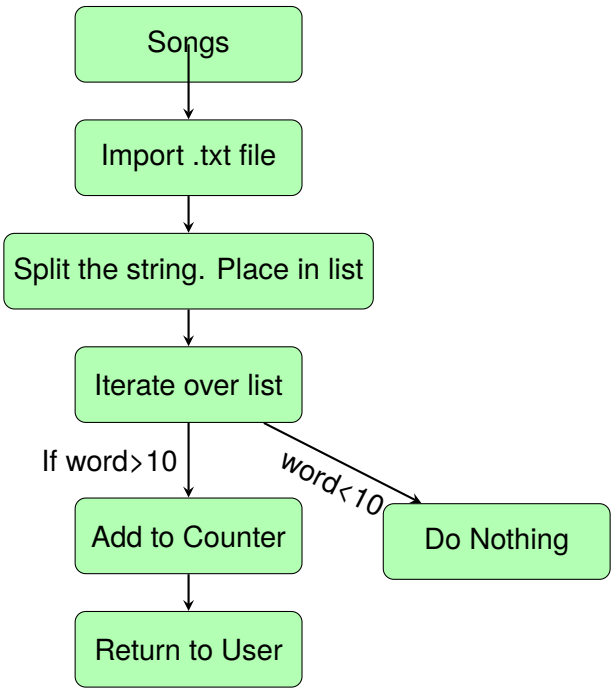
Designing a Program

In this course we will be talking about problem solving. Much in the world of problem solving comes down the pipe so to speak. Life hands you a problem and you have to solve it. Problem solving in computer science starts with picking a problem. Once the problem has been identified we then can focus on finding a solution. Once a procedure has been identified then comes the task or writing the code that executes the steps outlined in the solution.

First thing to do is ask who is this program for and what do they want. When we are starting coding these coding projects are often for ourselves. This is not selfish but natural as we need to improve before anyone else will want what we can do.

This class's first project will be for the inquisitive reader. We will be creating a program that gathers data from .txt files. Mine will be looking at the number of words over the length of 10 letters in songs(in .txt files, or maybe .csv files) it is given.

Now we have a problem and an desired results. Songs as .txt files go into our software and numbers come out. Specifically these numbers are the amount of words that are over 10 characters long. Now the path from Start to Finish is up to us. This is where we draw.



Once we have our diagram we can start looking at what processes we know will achieve our overall project goal. It is rare if we do not have to learn something to solve a problem. So do not be worried if there are massive differences in your skillset and what will need to be done in order to solve this problem.

It is often smart to start writing the code for the aspect of the project we know how to achieve. The UNIX philosophy tells us to write code that does one thing well and to make them pluggable into other processes of our software. While this might be a little overzealous in counting words that are over 10 characters in songs it translates to that we should be able to tell which process the sections of code we are writing are working on.

Assignment:

Create a problem or question where the data that needs to be analyzed is text. Come up with an outcome that when the program is run what should be extracted from the text. Then draw a diagram with the steps it would take to get there. (Note: copy and pasting data from the internet into a .txt file is not something you need to automate)

Function Introduction

Functions are the building blocks of projects. Similar to mathematics they take inputs and produce outputs. Unlike mathematics there is no worrying about one-to-one or onto. Functions can take many functions variables or no variable. According to Allen Downey their are two types of functions; fruitful functions "return" a value, void functions do not. Void functions may print an output of move graphics but they are the end they are not to pass into another function.

That concept brings us the UNIX philosophy. Two of the tenants are:

1. Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new "features".

2. Expect the output of every program to become the input to another, as yet unknown, program. Don't clutter output with extraneous information.

We will be trying to build our programs as small pieces that plug into each other. This allows us to change things easily and because we just change on piece of the functionality not some interconnected function that does many things. Think about these functions as as little pieces of a big machine. You want them to work independently and they and do their job well. If they fail to do their task it is obvious where the ball was dropped and easy to fix it.

Lets take a look at one:

```
#example of a void function
def hello_world():
    print('Hello World')

#example of fruitful functions
def mult(a, b):
    val = a * b
    return val
```

While the second function not print anything to the console it can quickly do so if we call print on the function mult with two function values.

In the exercise of creating a pyg latin translator we need to call this function on a string function variable.

Today's goal is to create a single world pyg latin translator.