Series and Sequences Coding Project ☺
B Precalc

The past few days, we have been working on understanding sequences and series from a mathematical point of view. This is going to help us create series and sequence themed computer programs as we learn how to code in python!!

Most important question: what is coding?

Great question! Coding is the process of writing directions for the computer to follow is a *language* that the computer can understand. Computers use *binary* to execute everything that it does. *Binary code* is simply a certain way of representing numbers through zeros and ones, where each digit represents a value double the digit to the right. We are familiar with the base-10 system of numbers where each digit represents a value equal to 10 times the value of the digit to the right. Let's learn by example. **Find the pattern and then fill in the rest of the table below**.

Base-10 representation	Binary representation
1	1
2	1 0
3	1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1
8	1 0 0 0
9	1 0 0 1
10	1 0 1 0
11	1 0 1 1
12	1 1 0 0
	1 0 0 1 0 0 1 1 1
100	
	1 1 0 0 1 0 1 1

This is the base language that computers use. How computers can translate binary into complex computer simulations, the Internet, your history paper, etc. is an act of building more complicated messages with binary as its basis.

Python is a *computer language* that you will learn for this project—there are many many more out there. Think of it as a middle ground between binary and how you think as a human being. In order to create complicated programs, it doesn't make sense to code in binary. Thus, python is a way of compromising with the computer to speak the same language and create cool things!

Some definitions:

**Below, the text that is in Arial font is just me talking and then the text that is in `Courier` font is mimicking python code. When I use the word *syntax*, I am referring to the specific way that a line or a command is written in python. Another note is that the computer does not read the text after a hashtag, #, as code. People use hashtags to write comments in their code. You'll see that I use a ton.

1. A *function* in python is very similar to a function in math, but it is simpler. A python function is a block of code that takes in an input and may or may not return an output. The syntax for a python function is as follows:

```
def function(input):     # This is a comment.

    code that creates an output
    ...[more code..blah blah blah
    be bop boop]

    return output
```

2. A *variable* is also just like a variable in math. It is a name that you use, as a coder, to store some value. Variables can have values of many types. Perhaps the easiest (for me, anyway) to comprehend is numerical variables. Here is an example of a numerical variable.

```
## The variable x, below, is set to 0

x=0
```

Variables can also be set to *strings*, or collections of characters. Strings are represented in quotations. Python recognizes single quotes ' ' and double quotes " " all the same.

```
## The variable adam_luv, below, is set to the string
#    'I love chocolate'

adam_luv= 'I love chocolate'
```

The value of a variable can always be changed as follows.

```
## If the variable is set to a number, the number can
#    be changed as such.

a=8.5          # The variable, a, is set to 8.5

a=a+2          # a is set to its previous value, 8.5,
               # plus 2, so a is now equal to 10.5
```

```
## If the variable is a string, the string can be
#    changed easily.

y='I like math'      # The variable y is set to the
                     # string 'I like math'

y='I love math'      # Here, it is changed to the
                     # value, 'I love math.'
```

3. A *conditional statement*, or an *if statement* means the same thing as an if statement in human language. “If I go to sleep early, then I will be at gather on time.” In computer science, it will look as follows:

```
if x==0:         # The conditional. Notice the double equal
                 # sign! This is the syntax required to say
                 # "if the variable x is equal to 0". Also
                 # notice the colon at the end of the
                 # statement.

     x=4         # In this case, if x is 0, we set x equal to
                 # 4. I am just using this as an example, but
                 # any block of code could exist here. Notice
                 # that the line after the if-statement is
                 # indented!

Code as normal...    # If the conditional is not true, then
                     # the computer skips the block of code
                     # inside the if statement and continues
                     # code as normal.
```

4. An *if-else statement* is a lot like an if statement, except it has an added `else:` block. It reads just like human language. “If I go to sleep early, then I will be at gather on time, or else, I won’t be at gather on time.”

```
if x==0 or x==1:     # You can add 'or' or 'and' in your
                     # conditional. The statement reads, "If
                     # x is equal to 0 or 1, then change x
                     # to 3. Or else, change x to 4."
     x=3

else:
     x=4
```

5. Last but not least is the beautiful and wildly useful *for-loop*. The for-loop will loop through a block of code the number of times you tell it to. It is incredibly useful when you are, say, summing a sequence. Here is an example.

```
def example_function():              # Here, we define a
                                     # function called
                                     # example_function and
                                     # it has no input.

     var=9          # Set variable, var, equal to 9.

     for m in range(5):         # The way this for-loop is
                                # created is by setting a
                                # variable, m, to increase
                                # from 0 to 4, increasing by
                                # 1 every time the computer
                                # loops back. Thus, this loop
                                # iterates 5 times,
                                # (0,1,2,3,4). More
                                # generically, when writing x
                                # in range(n), the computer
                                # will create a variable
                                # called x and will change
                                # its value every time the
                                # for-loop iterates. It will
                                # start at the value, 0, and
                                # loop until it hits x-1,
                                # thus iterating x times.

          var=var+1 # In each iteration of the loop, var is
                    # changed to its previous value plus 1.

     return var #The output of the function is the final
                #value of var. In other words, once the
                #for-loop occurs its final time, the value
                #of var is returned.
```

6. In math, functions have an input. In computer science, functions also have inputs (which you can see in the example above :). These inputs are called *parameters*. The difference between a *parameter* and a *variable* is one of the most important distinctions to understand in computer science. For me, it was the most confusing. I will try to explain my understanding to you.

In math, you can say f(x)=2x+3. If this were computer science, the x inside the parentheses would be the parameter. I know that in math you would call it a variable, but in computer science, this is a parameter. If you plug in a value for x, it would look like this: f(1)=2(1)+3=5. In computer science terms, you are *calling on the function* f, and in order to do that you need to plug in a value. The function returned the value 5 when the value 1 was passed into the function. **A parameter is the input of a function.**

A variable does not have to be associated with the input or the output of a function. Remember the function from the last example? I'll show it again without the comments just so you know what I'm talking about.

```
def example_function():
     var=9
     for m in range(5):
          var=var+1
     return var
```

If I were to *call* on the function, `example_function()`, I would always get the output 15. Every time. The function does not take in any parameters! There is no input! However, the variable, `var`, is being used throughout the function. In computer science, a variable is merely a way to store information.

7. Python understands mathematical operations in a similar way that you do. When we write 1 + 1, we know that that is equal to 2. So does python. However, there are some mathematical operations that are useful in computer science that you may not have ever encountered. The following table shows common mathematical operators, their symbols used in python code, and a brief description of what they mean.

Operator	Symbol used in Python	Brief description	Example:
Addition	+	Adds the values to the left and right of the operator.	`4+3 = 7`
Subtraction	-	Subtracts the value to the right of the operator from the value to the left of the operator.	`2-5 = -3`
Multiplication	*	Multiplies the values to the left and right of the operator.	`5*2 = 10`
Division	/	Divides the value to the left by the value to the right. In Python 2, this operation rounds the value down to the nearest whole number. 4 / 5 = 0.	`6/3 = 2` `4/5 = 0`
		To get a decimal, you can do the following:	`4 /(5 * 1.0) = 0.8`
Exponent	**	Raises the value on the left to the power of the value on the right.	`2**3 = 8`
Modulus	%	Divides the value on the left by the value on the right and returns the remainder.	`7%3 = 1`

If all the information above is really confusing, don't sweat it. We'll talk about all of this in class AND the point of this project is to allow you to practice this concept.

Read up to here by Wednesday ☺

Part 1: Getting to know terminal and python ☺

**For the entire project, you must type everything *exactly* as you see it in quotes. Computer programming is sensitive to every space, semicolon, colon, capitalization, etc. So if it says to call a folder "Python_Project", your folder name must be identical.

1. Go to your email inbox, find the email from me with the subject 'Python Project Files' and download all the attached files. In your Documents, make a new folder called 'Python_Project'. Save all of the files into the 'Python_Project' folder. Do not change their names. They should be:

   series_project_intro.py
   series_project_arithmetic.py
   series_project_geometric.py
   series_project_fibonacci.py

2. Go to your applications and open 'Terminal.' It should look like box that looks like it's from a SciFi movie, and it should say "bash" somewhere at the top. There also should be a blinking rectangle thingy. This is called the 'Terminal'! It is a program from which you will launch the programs that you write.

3. Type in

   ```
   cd ~/Documents/Python_Project
   ```

   and hit enter. You just told Terminal to change directory, and it is now inside of your Python_Project folder. Type in the following lines, hitting enter after every single line.

   ```
   chmod +x series_project_intro.py
   chmod +x series_project_arithmetic.py
   chmod +x series_project_geometric.py
   chmod +x series_project_fibonacci.py
   ```

   This allows you to run all of these files through terminal. It makes the file "executable." Now, you're all set to execute your programs! You never have to do this again with these specific files. When you make new files, you will always have to run these lines or your code won't work!

4. Open Sublime Text 2. Go to File -> Open -> then find 'series_project_intro.py'. This is the file where you will be writing all your code! Notice how there is a lot of text that is grey. A '#' denotes that the computer should not execute the text written after it. This allows the coder to write comments in the code, which is very important when collaborating so that someone can understand the lines of code being written.

5. Read all the comments in the file. Try to understand what each line is doing. Pay attention to the *syntax*, or the way each line is written. Where are there colons? Where are there equal signs? Commas? Parentheses? Indents? How big are the indents? These things all matter and any mistake will throw the computer into oblivion…just kidding. But it definitely won't work. Also, try to point out where there are parameters? Where are there variables?

6. This is the first question where you will actually write code! In the function `print_message(str)`, whose purpose is to print the string that is passed into the function unless the string is 'Math is boring!'. Your job is to write an if-statement that says that if the string passed through is equal to 'Math is boring!', then you must change it to something else. You can choose what message you would like to change it to. Comment each line and explain in very few words what the line does. **Test your if statement** by first **saving** the file once you feel that your if statement is correct. Next, go to terminal and type in

   ```
   ./series_project_intro.py
   ```

   and see if it works. If it doesn't work, then you typed something incorrectly and you need to go back and fix it. Work with each other on this if you're stuck but try to work through it!

7. Once you have found that your function works, test your function even further by changing the message that is passed into the function below the mainline, where it says `a = 'Math is boring!'`. Does it print the correct message? Once you feel like your function is correct, write down all the strings you passed into the function below and record their output below.

Part 2: You're ready for the real deal

1. In Sublime Text 2, go to File -> Open -> then find 'series_project_arithmetic.py'. In this file, you will find a function called `sum_arithmetic` that takes in three parameters. It takes in an initial value, an increment, and a number that represents the length of an arithmetic series. It returns the value of the sum of the arithmetic series of the given length. Write code that accomplishes this (hint: you will need a for-loop). *Make sure you write comments next to each line to explain in very few words what the purpose of that line is*. **Test** this by plugging in at least 10 different sets of values for initial, increment, and n in the lines below the mainline. You can run this code in terminal by **saving** the file every time you change something and writing

   ```
   ./series_project_arithmetic.py
   ```

   Write down all the values for initial, increment, and n as well as the output in the space below in a table. Make sure you test negative values and fractions. Show me that your code works for every possible arithmetic sequence.

2. Now, go to File -> Open then find ‘series_project_geometric.py’. In this file, you will find a function called `sum_geometric` and it has the same parameters and purpose of the function in question #1, except it takes in a constant ratio (multiplicative factor) and sums a geometric series of length n. Write a function that does this successfully. *Make sure you write comments next to each line to explain in very few words what the purpose of that line is*. This time I haven’t given you ANY code, but you have built the skills you need to write the function from scratch! Run the program in terminal by writing the following.

```
./series_project_geometric.py
```

Similar to the question above, change the values of initial, ratio, and n to test the function. Create 10 different sets of values and record them below in a table. Make sure to test negative values and fractions to show me that your code works for every possible geometric sequence!

3. Open the file 'series_project_fibonacci.py'. In this file, you will find a function called `fibonacci` and it takes in a number. This number, n, represents the n'th number in the Fibonacci sequence. In case you have forgotten, the Fibonacci sequence begins with 1,1 and then each element is the sum of the two previous elements: (1, 1, 2, 3, 5, 8, 13, …). The function's purpose is to spit out the value of the n'th number in the Fibonacci sequence when only given n. This can be done in 2 lines of code ;). Again, comment your code. You don't need to make a table this time.

Part 3: Handing everything in!!

First of all, make sure everything is saved! Make sure that all four files are completed and that each line is commented. Once you are sure everything is good and saved, go ahead and email the files to me at agilbert@chewonki.org. Also hand me this packet.

Yay! You're all done ☺