pythonIn-class Task

This Lab is more of a Tutorial to Reinforce some challenges for prior topics. Don't forget to Change the numbers and you do not need to assign – every problem in section 1 & 2. The main focus is the green font section

Objective: The purpose of this lab is to get you comfortable with using the Python functions and modules. Python has a lot of modules that come with the language, which collectively are called the Python Standard Library. The Python documentation contains the specifications for all the functions in the library. You can get to it from the link on the course website, or by going here:

The Python Standard Library — Python 3.10.7 documentation

However, be warned that the Python library documentation is not written for beginners. While it will make more sense later in the course, most of it will be hard to read now. The purpose of this lab is to guide you around some simpler parts of the Python library.

Getting Credit for the Lab. This lab is very similar to the previous one, in that you will be typing commands into the Python interactive shell and recording the results. All of your answers should be written down on a sheet of paper (or on the sheet provided to you in the lab). When you are finished, you should show your written answers to this lab to your lab instructor, who will record that you did it.

You should always do your best to finish during lab hours. Remember that this lab task are graded on effort, not correctness.

With that said, it is very important that you complete the String section + last section, **Writing MyGeeUp Function**. If you find yourself running out of time in the lab, you should jump to this section. And if you are not able to finish it today, please consult with the TA even office hours if need

1. Strings

We talked about strings last week in class. Strings have many handy methods, whose specifications can be found at the following URL:

http://docs.python.org/2/library/stdtypes.html#string-methods

For right now, look at section 5.6.1 "String Methods," and do not worry about all the unfamiliar terminology. You will understand it all by the end of the semester.

Using a method is a lot like using a function. The difference is that you first start with the string to operate on, follow it with a period, and *then* use the name of the method as in a function call.

For example, the following all work in Python:

```
s.index('a')                    # assuming the variable s contains a string
'CS 1301.index('1') # you can call methods on a literal value
```

```
s.strip().index('a') # s.strip() returns a string, which takes a method
```

We will learn more about methods in lectures, but this syntax is enough for now. Strings are a built-in type, so although there is a module named string, you do not need it for basic string operations.

Before starting with the table below, enter the following statement into the Python shell:

```
s = 'Hello World!'
```

Once you have done that, use the string stored in s to fill out the table, just as before.

Expression	Expected Value	Calculated Value	Reason for Calculated Value
s[1]	"e"	"e"	The second element of the string is e.
s[15]	"IndexError"	"IndexError"	There is no 16th element in the string! The max it can do is s[11]
s[1:5]	"ello"	"ello"	It prints eveything starting from the 2nd element to the 6th element, not incuding the 6th element.
s[:5]	"Hello"	"Hello"	It prints everything until the 6th element, not including the 6th element.
s[5:]	" World!"	" World!"	It prints everything starting from the 6th element, including the 6th element till the end of the string.
'e' in s	True	True	Since the element/character 'e' does lie in the string.
'x' in s	False	False	Because there is no element 'x' in the string.
s.index('e')	1	1	Since the element e, is the second element of the string. Also, because python is 0 indexed.
s.index('x')	ValueError	ValueError	Since there is no element 'x' in the string. We are trying to index something that is nonexistant.
s.index('l', 5)	9	9	Since .index() takes in a second optional argument which indicates the program from where they should start indexing the required element. And since there is only one 'l' after the 5th element, we print the location of that element.

s.find('e')	1	1	.find() necessarily does the same task as .index(), and since 'e' is the second element of the string, we print out 1.
s.find('x')	-1	-1	The difference between .index and .find is that, .find returns '-1' if the element to find is nonexistant in the string. Meanwhile .index returns an Error. Since there is no 'x' in the string, it prints out '-1'.
s.lower()	'hello world!'	'hello world!'	.lower() function turns all uppercase characters in the string to its lowercase equivalent.
s.islower()	False	False	.islower() function returns a boolean answering the question, 'are all the characters in the string lowercase?' Since 'H' and 'W' of our string is in uppercase, it returns False.
s[1:5].islower()	True	True	As the elements between the 2nd item and the 5th item are all in lowercase.

As we saw in the last lab, even though single and double quotes are used to delimit string literals, they also are characters that can occur in strings, just like any other character. The simplest way to get a quote character into a string is to use the other kind of quotes to delimit the string:

```
q = "Don't panic!"
```

When both kinds of quotes need to appear, we need to use the *escape sequences* we saw in class. An escape sequence consists of a backslash followed by a quote:

```
q1 = 'The phrase, "Don\'t panic!" is frequently uttered by consultants.'
```

You could also write a string like this using double quotes as the delimiters. **Rewrite the assignment statement for** `q1` **above using a double-quoted string literal**:

q1 = "The phrase, 'Don\'t panic' is frequently uttered by consultants."

2. Writing Your First Function

In addition to the modules provided by Python, you can make your own. To try this out, we return to our string `q1` above. As you can see, this string has double quotes inside of it. We would like to extract the substring inside those quotes (which is "Don't panic!"). But we want to do it in a way that is *independent* of `q1`. That means, whatever Python statements we use, they should still work if we used a different value of `q1`, provided that it still has a pair of double-quote characters somewhere.

In the box below, write a sequence of one or more assignment statements, ending by assigning to a variable inner. The contents of inner should be the two substrings inside the double-quote characters (but not including the double quotes themselves).

```
def inner(sting):
    left = sting.find('"')
    right = sting.find('"', left+1)

    return sting[left+1:right]

q1 = input("Enter the test case: ")
print (inner(q1))
```

To test that your statements are correct, do the following. First, type in q1 = 'The phrase, "Don\'t panic!" is frequently uttered by consultants.'

Then type in your statements from the box above. Finally, print the value of inner. You should see Don't panic, without the quotes (printing always removes the quotes from a string value).

Now, try the process again with q1 = 'The question "Can you help me?" is often asked in consulting hours.'

Type in the assignment statement above, then type in your statements from the box, and finally print the value of inner. You should see Can you help me?, without the quotes. If you had to modify your answer in the box for the second q1, you have done it incorrectly.

The statements in the box are the start of your first interesting computer program. Given a string q1 with a value in quotes, we can always extract the substring inside the quotes. Of course, we have to retype it every time we want to change q1, which can be really annoying.

To keep ourselves from retyping these all the time, you are going to write a new function and put it in a ***module***. To create a module, open up Vs Code Edit. Select "New File" from the File menu and save it (even though it is still empty) as lab05.py. Inside of this module you will write the body of a function called first inside quotes (); this function takes a string and returns the substring inside the first pair of double-quote characters.

Here is a Video –> Click here

Your function should look something like this:

```
def first _inside _quotes(q1): # Your
    assignment statements here

    return inner
```

Replace the comment with your assignment statements in the box on the previous page.

It is now time to try out your function. Navigate the command line to the folder containing this file. Start up the Python interactive shell and import the module. **Remember to omit the** .py **suffix when you use the import command**. Call the function lab05.first inside quotes('The instructions say "Dry-clean only".')

What Happens Next?

It prints out - Dry-clean only

To check off this portion of the lab you should demo your function to the course staff with a few different arguments. Make sure that each argument always has a pair of double-quote characters in it. When you have completed all this, you are done with the lab.

3. Calling functions

Built-in functions are those that do not require you to *import* a module to use them. You can find a list of them in the Python documentation:

http://docs.python.org/library/functions.html

Note that the casting and typing operations are listed as functions. While this is true in Python, this is not always the case in other programming languages. That is why we treated those functions especially in the last lab.

The table on the next page uses several built-in functions. Fill out the table just like you did in last week's lab. For each expression, first **compute the expression in your head, without Python**. Write the result in the second column, or "?" if you have no idea. Next, use Python to compute the expression. If the answers are different, try to explain why in the last column.

Expression	Expected Value	Calculated Value	Reason for Calculated Value
min(25, 4)	4	4	The min() function gives the minimum value. Here 4 is smaller than 25 and so it is the output.
max(25, 4)	25	25	Logic is similar to the min() function
min(25, max(27, 4))	25	25	First it gets the max value between 27 and 4, which is 27 and then compares it with 25 for

			the minimum value. Hence 25 is printed.
abs(25)	25	25	The abs() function prints the absolute value of an element. The absolute value of 25 is 25 itself.
abs(-25)	25	25	The absolute value of any negative number is its positive equivalent.
round(25.6)	26	26	The round() function rounds off the floating value to the nearest integer.
round(-25.6)	-26	-26	Same logic as above.
round(25.64, 0)	26.0	26.0	The round function has an optional argument which allows the user to round off the number to n digits precision after the decimal point.
round(25.64, 1)	25.6	25.6	Same logic as above.
round(25.64, 2)	25.64	25.64	Same logic as above.
len('Truth')	5	5	The len() function gives us the length of a string. Since the string 'Truth' contains 5 characters, 5 is printed.
len('Truth ' + 'is ' + 'best')	12	12	After concatinating all the strings, we have 11 characters but since the function len() also counts whitespace as a character, we get an output of 12.

4. Using a Python Module

One of the more important Python library modules is the math module. It contains essential mathematical functions like sin and cos. It also contains mathematical constants such as π. These values are stored in *global variables*; global variables are variables in a module (created via an assignment statement) that are not part of any function. To learn more about this module, look at its online documentation:

http://docs.python.org/library/math.html

To use a module, you must *import* it. Type the following into the Python interactive shell:

import math

You can now access all of the functions and global variables in math. However, they are still in the math *namespace*. That means that in order to use any of them, you have to put "math." before the function or variable name. For example, to access the variable pi, you must type math.pi.

Fill out the table on the next page, using the same approach as the previous table.

Expression	Expected Value	Calculated Value	Reason for Calculated Value
math.sqrt(9)	3.0	3.0	The .sqrt() function gives the square root of an element.
math.sqrt(-9)	ValueError	ValueError	Since we cannot calcluate the square root of negative numbers, the program gave us an Error.
math.floor(3.7)	3	3	The .floor() function rounds off the number to the lowest whole number.
math.ceil(3.7)	4	4	The .ceil() function rounds off the number to the highest whole number.
math.ceil(-3.7)	-3	-3	-3 is grerater as a number than -4.
math.copysign(2,-3.7)	-2.0	-2.0	The .copysign() function takes two arguments, x and y; and takes the sign of y and returns it with x, regardless of the original sign of x.
math.trunc(3.7)	3	3	The .trunc() function returns the integer part of a number, removing any fractional digits.

math.trunc(-3.7)	-3	-3	Same logic as above.
math.pi	3.141592653589793	3.141592653589793	The .pi function returns the value of π .
math.cos(math.pi)	-1.0	-1.0	The .cos() function returns the cosine of value passed as argument.

In addition to the above expressions, type the following code into the Python interactive shell:

```
math.pi = 3 math.pi
```

What Happens?
We get an Error. SyntaxError to be specific. We get this because the syntax of the code doesn't make sense and hence is invalid.