Assignment 2

General Instructions

Note	**Read the instructions carefully and follow them exactly**
Assignment Weight	**As outlined in the syllabus, this assignment is worth 5 of your final grade**
Due Date	**This assignment is due at 8AM on Monday, Oct 16, 2017**
Important	**As outlined in the syllabus, late submissions will not be accepted**
	Any files with syntax errors automatically be excluded from grading. Be sure to test your code before you submit it
	For all functions, make sure you've written good docstrings that include the documentation, type contract, and the preconditions and postconditions

The goal of this assignment is to learn and practice the concepts covered thus far: function design, function calls, branching (i.e. if statements), strings, for-loops. We have a little more content to cover in class before you can finish the assignment, but you should be able to do most of it right now.

Note that the examples provided in this handout are just some examples with which you may test your code. These examples will be used to grade your submission, as well as other inputs not in the examples (so don't hard code your programs to work for just the examples here). Therefore, when you test your own code before submitting, be sure to test with different values which you think I might test it with. When you're satisfied with your code and your tests, copy your test cases and their outputs into the relevant text file, which also you are required to submit.

This assignment has two parts, each of which has to be addressed in a separate python file. The solution to Part 1 should be in `part1_xxxxxx.py` and the solution to Part 2 should be in `part2_xxxxxx.py` (where `xxxxxx` is your student number). Similarly, the tests for part 1 should be in `part1_tests.txt` and the tests for part 1 should be in `part2_tests.txt`. As practiced in Lab 1, put both these files in a folder/directory called `a2_xxxxxx` (where `xxxxxx` is your student number), and zip it into `a2_xxxxxx.zip`. You are required to

submit `a2_xxxxxx.zip`. Make sure that your files are named correctly - we will look for those exact filenames, and disregard all others (this means that if you name your file incorrectly, we will not consider it, and it will receive no marks).

If running a python file that you submit results in an error, then all questions answered in that file will get a zero.

For all questions that involve user input, your code should be flexible/robust enough to allow the user to input spaces/tabs before and after their actual input

Since some students had trouble with Assignment 1, there is a bonus question in Assignment 2. If you do the bonus question, add those files (see the description of the bonus question below) to the folder which you zip and submit.

1 Math Test Application - 20 points

Your little cousin has just started learning some math in grade school, and needs to practice for her tests. Being the ever helpful person that you are, you've taken it upon yourself to write a little program to help her practice. She already understands addition and multiplication. You are going to help her with subtraction and exponentiation.

Write all the code for your submission to this question in a file called `part1_xxxxxx.py` (where `xxxxxx` is your student number). Put the

1.1 `performTest(flag, n)` (The Core Function)

To do this, you are going to write a function called `performTest`. `performTest` takes two parameters, namely `flag` (which is supposed to be an `int`) and `n` (which is also supposed to be an `int`). If `flag` is `0`, `performTest` helps practice subtraction. But if `flag` is `1`, `performTest` helps practice exponentiation.

`performTest` then generates `n` math problems that your cousin must answer in turn. For each question, it generates two random positive, single-digit numbers (check out python's `random` module to see if there's a useful function in there) and asks your cousin for the answer to the math problem with those two numbers (either subtract the second number from the first, or raise the first number to the power of the second number). `performTest` then prompts your cousin for the answer, and checks if her answer is correct. At the end of `n` questions, `performTest` returns the number of questions answered correctly.

1.2 The User Interaction

Now that you have a function that performs the core functionality, you want to make it more user friendly for your cousin to use (after all, she doesn't know how to write code and call functions). While you are working on this, your neighbors hear about your program, and ask if they can use it for their kids as well (they bribe you with cookies - how could you possibly refuse?!). So you

decide to make it a little more user-friendly and ask for the user's name before calling `performTest`.

In the `main` part of your program, outside of the `performTest` function, write some code that asks the user for their name, and whether they would like to practice subtraction or exponentiation. Then ask how many practice questions they'd like (if they say `0`, then your code should not ask any math problems at all - they're ready for the test). Using their responses, call the `performTest` function with the appropriate values. When it returns the number of correct answers, display a message to the user (for the sake of the example, let's pretend the user's name is `Gamora`):

- If she did 90% or better, display `Congratulations Gamora! You'll probably get an A tomorrow. Now go eat your dinner and go to sleep. Good bye Gamora!` on screen.
- If she did 70% or better, but worse than 90%, display `You did well Gamora, but I know you can do better. Good bye Gamora!` on screen.
- If she did worse than 70%, display `I think you need some more practice Gamora. Good bye Gamora!` on screen.

We've provided you with some starter code. Do not change this code, as our testing methodology depends on that code. If you change that code, we might become unable to test your code, which will result in you losing marks.

Keep in mind that your little cousin is very young and doesn't necessarily follow best practices for user input. So, she might include spaces before/after her name or any other input. Your code should be sufficiently flexible to accept these inputs

Hint check out `help(str.strip)`

2 A Library of Functions

For this part of the assignment, you are required two write several functions and save them in `part2_xxxxxx.py`.

2.1 `seriesSum(n)` - 5 points

(I don't have a story for this question. But if you do it right, you'll get marks)

Write a function called `seriesSum`. This function prompts the user for a non-negative integer, but the user might give you a negative integer. The function then returns $1000 + \sum_{i=1}^{n} \frac{1}{n^2}$ if the user inputs a non-negative integer. If the user inputs a negative integer, the function returns `None`.

2.1.1 Sample Testing

```
>>> series_sum()
Please enter a non-negative integer: -10
>>> series_sum()
Please enter a non-negative integer:
0
1000
>>> series_sum()
Please enter a non-negative integer:
5
1001.463611111111
```

2.2 `catch(side, x, y)` - 5 points

Your little brother has just gotten accepted to Carleton University (this is going to make for a fun family reunion at Thanksgiving this year), and you're helping him pack. The two of you are in his room - he's throwing his clothes at you, and you're trying to catch them in a box with a square bottom. But as most little brothers are, yours is a mischievous little $#*(&^!. He throws his clothes all over his room, and you have to run around with the box to make sure his clothes fall into the box. Being the clever person that you are, you've worked out a Cartesian coordinate system for his room, so that you can predict exactly where his clothes are going to fall. Now, you just need to figure out where to place the box, so that the clothes land exactly inside it.

Write a function called `catch`. `catch` takes three input parameters:

Parameter Name	Parameter Type	Parameter Description
`side`	`float`	the length of the side of the box you're carrying
`x`	`float`	the x coordinate of where your brother's clothes are going to land in his room
`y`	`float`	the y coordinate of where your brother's clothes are going to land in his room

If side is zero or a negative number, `catch` should return `None`. Else, catch should return the tuple `(x,y)`, the coordinates of the bottom left corner of the box, so that your brother's clothes land exactly in the center of the bottom of the box.

2.2.1 Sample Testing

```
>>> catch(2, 1.5, 0)
(0.5, -1.0)
>>> catch(0, 1, 1)
>>> catch(5, -1, 8)
(-3.5, 5.5)
```

2.3 `pell(n)` - 5 points

Pell numbers are a mathematical sequence of numbers that help approximate the value of $\sqrt{2}$ (check out the Wikipedia page on Pell Numbers: https://en.wikipedia.org/wiki/Pell_number). The sequence is defined recursively as shown in eq. 1.

$$P_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ 2P_{n-1} + P_{n-2} & \text{if } n > 1 \end{cases} \tag{1}$$

Write a function called `pell` that takes one integer parameter `n`, of type `int`. If `n` is negative, `pell` returns `None`. Else, `pell` returns the nth Pell number (i.e. P_n).

2.3.1 Sample Testing

```
>>> pell(0)
0
>>> pell(1)
1
>>> pell(2)
2
>>> pell(5)
29
>>> pell(-45)
>>> pell(-1)
>>> pell(6)
70
```

2.4 `countMembers(s)` - 5 points

(ok this is really lame, but I couldn't come up with a better story)

There is a list of *extraordinary* characters. These characters are the lower case letters between `e` and `j` (inclusive), the upper case letters between `Q` and

`X` (inclusive), numerals between `2` and `6` (inclusive), and the exclamation point (`!`), comma (`,`), and backslash (\)

You are required to count how many times these characters appear in a string.

Write a function called `countMembers` that takes a single input parameter `s`, of type `str`. `countMembers` then returns the number of characters in `s`, that are extraordinary. Therefore, if there are two `X`s in `s`, `countMembers` must count two extraordinary characters (one for each occurrence of `X` in `s`).

Challenge See if you can write the body of this function in one line

2.4.1 Sample Testing

```
>>> countMembers('One Two')
2
>>> countMembers('hahahahaha')
5
>>> countMembers('928375nsdjkohLh artALKGFm ald')
6
>>> countMembers('l\"sc')
0
>>> countMembers('l\\"sc')
1
>>> countMembers('\\')
1
>>> countMembers('\\Hello!')
3
```

2.5 `encrypt(s)` - 5 points

You want to send a note to your friend in class, and because you want to be respectful in class, you don't want to whip out your phone and send a text or an email (or any other digital communication). Instead, you choose to go old school and write it out on a piece of paper and pass it along to your friend. The problem is, you don't want anyone else to read the note as they pass it along. Luckily, your friend and you have come up with an encryption system so that nobody else can understand your message. Here's how it works: you write out your message backwards (so, `Hello, world` becomes `dlrow ,olleH`). But you don't stop there, because that's too easy to crack - anyone can figure that out! Now that you've written in backwards, Then you start on either side of the string and bring the characters together. So the first and the last characters become the first and the second character in the encrypted string, and the second and the second last characters become the third and the fourth characters in the string, and so on. Thus, `Hello, world` ultimately becomes

`dHlerlolwo ,` (notice how all punctuation, special characters, spaces, etc are all treated the same) and `0123456789` becomes `9081726354`.

Write a function called `encrypt`, which takes one parameter `s`, of type `str`. `encrypt` returns a `str`, which is the encrypted version of `s`.

Fun fact in cryptography, `s` is called "clear text" and the encrypted version you return is called "cipher text"

Challenge See if you can write the body of this function in one line

2.5.1 Sample Testing

```
>>> encrypt('Hello, world')
'dHlerlolwo ,'
>>> encrypt("0123456789")
'9081726354'
```

2.6 `alienNumbers(s)` - 5 points

Strange communication has been intercepted between two alien species at war. NASA's top linguists have determined that these aliens use a weird numbering system. They have symbols for various numeric values, and they just add all the values for the symbols in a given numeral to calculate the number. NASA's linguists have given you the following table of symbols and their numeric values. Since they have a lot of these numbers to compute, they want you to write a function that they can use to automate this task.

Symbol	Value
`T`	1024
`y`	598
`!`	121
`a`	42
`N`	6
`U`	1

Thus `a!ya!U!NaU` represents a value of `1095` (see table below for an explanation).

Numeral	Value	Occurrences	Total Value	Running Total
`T`	1024	0	$0 \times 1024 = 0$	0
`y`	598	1	$1 \times 598 = 598$	598
`!`	121	3	$3 \times 121 = 363$	961
`a`	42	3	$3 \times 42 = 126$	1087
`N`	6	1	$1 \times 6 = 6$	1093
`U`	1	2	$2 \times 1 = 2$	1095

Write a function called `alienNumbers` that takes one string parameter `s`, and returns the integer value represented by `s`. Thus, `alienNumbers(a!ya!U!NaU)` returns `1095`.

Challenge See if you can write the body of this function in one line

2.6.1 Sample Testing

```
>>> alienNumbers("a!ya!U!NaU")
1095
>>> alienNumbers("aaaUUU")
129
>>> alienNumbers("")
0
```

2.7 `alienNumbersAgain(s)` - 5 points

NASA is very pleased with your proof-of-concept solution in the previous question. Now, they've designed a small chip that runs a very restricted version of python - it doesn't have any of the string methods that you are used to. They want you to now rewrite your `alienNumbers` function to run without using any of those string methods you may have otherwise used. Basically, you can use a loop of some sort and any branching you'd like, but no string methods.

Write a function called `alienNumbersAgain`, that takes a single string parameter `s`, and returns the numeric value of the number that `s` represents in the alien numbering system.

Challenge See if you can write the body of this function in one line

2.7.1 Sample Testing

```
>>> alienNumbers("a!ya!U!NaU")
1095
>>> alienNumbers("aaaUUU")
129
>>> alienNumbers("")
0
```

2.8 `oPify(s)` - 5 points

(Sorry, no story for this one)

Write a function called `oPify`, that takes a single string parameter (`s`) and returns a string. This function looks at the given string and considers every pair of consecutive characters. It returns a string with the letters `o` and `p` inserted between these pairs of characters. If the first character is uppercase, it inserts an uppercase `O`. If however, it is lowercase, it inserts the lowercase `o`. If the second character is uppercase, it inserts an uppercase `P`. If however, it is lowercase,

it inserts the lowercase `p`. If either character is not a letter in the alphabet, it ignores the corresponding `o` or `p`, but keeps the character itself. If either character is not a letter of the alphabet (e.g. if it's a number, or any other special character), then ignore the `O`/`o` and `P`/`p` for that character (see examples below).

Hint check out `help(str.isalpha)` and `help(str.isupper)`

2.8.1 Sample Testing

```
>>> oPify("aa")
'aopa'
>>> oPify("aB")
'aoPB'
>>> oPify("ooo")
'oopoopo'
>>> oPify("ax1")
'aopx1'
>>> oPify("abcdef22")
'aopbopcopdopeopf22'
>>> oPify("abcdef22x")
'aopbopcopdopeopf22x'
>>> oPify("aBCdef22x")
'aoPBOPCOpdopeopf22x'
>>> oPify("x")
'x'
>>> oPify("123456")
'123456'
```

2.9 `battleOutcome(s)` - 10 points

Diplomatic relations between the Jedi and the Sith are very tense right now, and a battle is imminent. As the two armies march towards each other, the Jedi High Council has asked you to predict the outcome of the battle. You note that the Jedi use blue light sabers and the Sith use red light sabers, and that each Jedi is on average 1.5 times as skilled as a Sith, so you can do some simple algebra to figure out which side is going to win. You also realize that it's been a while the light saber technology has been open sourced, so civilians who do not participate in the battle have green light sabers.

Write a function named `battleOutcome`, that takes a single string parameter `s`. The string lists the colors red, green, and blue multiple times, separated by spaces. `battleOutcome` has to count the number of occurrences of each color, and report on the outcome of the battle.

- If the combined skill level of the Jedi army is at least twice the combined skill level of the Sith army, `battleOutcome` returns the string **``The Jedi army destroys the Sith army''**
- If the combined skill level of the Jedi army is at least 1.5 times (but not quite twice) the combined skill level of the Sith army, `battleOutcome` returns the string **``The Jedi army defeats the Sith army''**
- If the two armies are within 1.5 times each other's skill level, `battleOutcome` returns the string **``Ooh! this is going to be interesting''**
- If the combined skill level of the Sith army is at least twice the combined skill level of the Jedi army, `battleOutcome` returns the string **``The Sith army destroys the Jedi army''**
- If the combined skill level of the Sith army is at least 1.5 times (but not quite twice) the combined skill level of the Jedi army, `battleOutcome` returns the string **``The Sith army defeats the Jedi army''**

Since the Jedi High Council are not familiar with python, write a function called `battleInterface`, that takes no parameters.Use it to ask the user to input a space-separated string of the three light saber colors mentioned. Then, pass that string to `battleOutcome` and obtain the results. Finally, display the string **``This is what's going to happen: ``**, followed by the results from `battleOutcome`.

2.9.1 Sample Testing

```
>>> battleOutcome(' red    red blue green
red blue green     ')
'Ooh! this is going to be interesting'
>>> battleOutcome('red    red blue green
red blue green blue blue blue')
'The Jedi army destroys the Sith army'
>>> battleOutcome('red    red blue green    red green')
'The Sith army destroys the Jedi army'
>>> battleOutcome('red blue red blue')
'The Jedi army defeats the Sith army'
>>> battleOutcome('red blue red blue red red red')
'The Sith army defeats the Jedi army'
```

3 Bonus - 10 points

This is ninja-level programming for this course, which is why it's worth twice as much as a normal question. The actual solution is quite easy, but requires a lot of thought and effort to realize

Write a function called `callfunc` that accepts two parameters, namely `func` and `args`. `func` is a function (which itself takes three arguments) that you can call with arguments, much like any other function you write. `args` is a tuple with three elements in it. `callfunc` then calls the function `func` using the elements in the tuple `args` as arguments to this function, returning its return value.

Therefore, if there is a function named `add3` that takes three numbers and returns their sum, then `callfunc(add3, (1,2,3))` should call the function `add3`, capture its return value, and return that captured return value.

3.1 Sample Testing

```
>>> def add3(a,b,c):
    return a+b+c

>>> callfunc(add3, (1,2,3))
6
>>> callfunc(add3, ())
TypeError: add3() missing 3 required positional arguments: 'a', 'b', and 'c'
```