CMSC 201 Spring 2016
Lab 07 – Functions

Assignment: Lab 07 – Functions
Due Date: During discussion, March 28th through March 31st
Value: 10 points

Part 1: Scope

Everything in Python has a ***scope*** – the places in the program in which it is accessible. For example, you can create a constant outside of **`main()`**.

```
MAX_VAL = 8

def main():
    # etc
```

That constant is now a ***global*** constant, which means it can be accessed by any code in the file. (Remember, for this course you are only allowed to have constants be global – regular variables should not be declared outside of functions.)

Local variables are only accessible to code within their same scope. If a variable is declared in **`main()`**, a separate function called **`printInfo()`** will not be able to access it. In the same way, a variable in **`printInfo()`** will not be accessible to the code in **`main()`**.

```
def printInfo():
    # this variable can't be accessed by main()
    varForPrintInfo = 5

def main():
    # this variable can't be accessed by printInfo()
    varForMain = 17

main()
```

Part 2: Functions

A function in Python is a way of compartmentalizing our code: a well-written function does *one* thing, and does it very well. A function allows us to write a piece of code once, and to then use, or "call," the function whenever we want to use that code.

A function has a few key parts:

1. Function name
   - This is how we call the function. It tells Python that we want it to use that function and execute its…
2. Function body
   - This is the code that makes up the function. This is what the function does when called.
3. Formal parameters (optional)
   - A function uses parameters to take in information from the code that called it. This is one of the ways that data is passed from one piece of code to another. A function can have no parameters, one parameter, or it could have a hundred!
4. Return statement (optional)
   - A function can also return data to the code that called it. This is the other way that data is passed around your program. A function can return multiple variables; a function with no **`return`** statement automatically returns **`None`**.

Let's take a look at some example functions and how they work:

```
def printName(name):
    print("Hello, my name is", name)
```

This function is called **`printName()`**; it takes in one formal parameter (**`name`**) and does not have a **`return`** statement. In order to use the code in this function, we must call the function and pass it an actual parameter. The actual parameter could be a variable, or it could be a literal string (one with quotation marks around it).

Here's the **printName()** function again, but this time we also have a **main()** that calls the function multiple times.

```
def printName(name):
    print("Hello, my name is", name)

def main():
    prezUMBC = "Hrabowski"
    prezUSA  = "Obama"
    printName(prezUMBC)
    printName(prezUSA)
    printName("John Jacob Jingleheimer Schmidt")

main()
```

Note that we have called the function with variables and with a string literal. Note also that the variable names we passed as actual parameters (**prezUMBC** and **prezUSA**) do not match the name of the formal parameter.

Here is the output for the code above:

```
Hello, my name is Hrabowski
Hello, my name is Obama
Hello, my name is John Jacob Jingleheimer Schmidt
```

Part 3: Returning from Functions

Here is another example of a function.

```
def attemptChange(num):
    num = 7 * num
    return num
```

This function has a **return** statement, which means that it returns a value to the code calling it. In order to "catch" or "save" this value, the code calling the function must use the ***assignment operator*** to assign the value returned to a variable for later use. Let's examine what that would look like.

Here is a **main()** function that calls the **attemptChange()** function two times. What do you think the output of the following code will be?

```
def attemptChange(num):
    num = 7 * num
    return num

def main():
    num = 5
    print("num was first", num)
    attemptChange(num)
    print("num is now...", num)
    num = attemptChange(num)
    print("num is now...", num)

main()
```

Before we look at the output, let's look at the two calls that are made to the **attemptChange()** function. In the first one, we pass in **num** as our actual parameter – but we don't use the assignment operator in this statement. What will happen to the **num** that is returned? Does **num** change in **main()**?

The second call to the **attemptChange()** function is identical, but this time we are using the assignment operator. What does that mean for the value of **num** in **main()**?

Here is the output:

```
num was first 5
num is now... 5
one last try: 35
```

From the output, we can see that the first call to the function didn't do anything to the value of **num**! We **must** use the assignment operator if we want to save the values returned by our function. Even though **main()** and **attemptChange()** both have variables named **num**, they are in different ***scopes***, and so a change to one does not mean a change to the other.

Part 4A: Writing Your Program

After logging into GL, navigate to the **Labs** folder inside your **201** folder. Create a folder there called **lab7**, and go inside the newly created **lab7** directory.

```
linux2[1]% cd 201/Labs
linux2[2]% pwd
/afs/umbc.edu/users/k/k/k38/home/201/Labs
linux2[3]% mkdir lab7
linux2[4]% cd lab7
linux2[5]% pwd
/afs/umbc.edu/users/k/k/k38/home/201/Labs/lab7
linux2[6]%
```

Once you're in the folder, you will need to copy the starter file from my public directory. Type (all on one line – don't forget the period at the end!):

```
cp /afs/umbc.edu/users/k/k/k38/pub/cs201/tv_shows_fxns.py .
```

To open the file for editing, type

```
emacs tv_shows_fxns.py
```

and hit enter.

The first thing you should do in your new file is create and fill out the comment header block at the top of your file. Here is a template:

```
# File:          tv_shows_fxns.py
# Author:        YOUR NAME
# Date:          TODAY'S DATE
# Section:       YOUR DISCUSSION SECTION NUMBER
# E-mail:        USERNAME@umbc.edu
# Description:
#   This file contains python code that implements lab5
#   (a TV show voting system) using functions to:
#   1) Get a choice
#   2) Find the name of the winner
```

Now you can start writing your code for the lab, following the instructions in Part 3B.

Part 4B: Creating Functions

At this point, if you try to run the file, you will get an error. That is because the file is only partially completed for you.

You will need to update the file to complete the two function definitions and two function calls. If you open the file, you should see comments boxed in by **#** signs – these are where you need to write new code. Read the function header comments to see the details about the two functions.

The **`getVote()`** function should be similar to your code from the last lab. The major difference is that you should let the user know they made an incorrect choice before prompting again. (Feel free to use your code from last lab to complete it, but be careful to pay attention to what you're doing – don't just copy and paste!)

The **`getWinner()`** function is new code for this lab, and you will have to code it up from scratch. Instead of simple presenting the results, you will write code that determines the name of the winning show, based on the votes given.

PRO-TIP: Both of these functions should have a return statement.

You will also need to write calls to both of these functions; the places where these calls need to happen in **`main()`** are indicated for you. You shouldn't need to write any other code.

Sample output is on the next page;

Hints are on the page after that, if you need them.

Sample output for this lab, with the user input in blue.

(Note that in the event of a tie, the show that comes first in the list wins.)

```
bash-4.1$ python tv_shows_fxns.py
1 - Daredevil
2 - Fargo
3 - Limitless
4 - Elementary
5 - Brooklyn 99
6 - Empire
7 - Supergirl
You and your friends are voting on a show to watch.
Which show would you like to vote for?
Enter '0' to stop voting: 4
Enter '0' to stop voting: 9
Invalid vote -- try again
Enter '0' to stop voting: 3
Enter '0' to stop voting: 5
Enter '0' to stop voting: 5
Enter '0' to stop voting: 2
Enter '0' to stop voting: 3
Enter '0' to stop voting: 7
Enter '0' to stop voting: 0

Here are the final votes:
Daredevil has    0 votes
Fargo has        1 votes
Limitless has    2 votes
Elementary has   1 votes
Brooklyn 99 has  2 votes
Empire has       0 votes
Supergirl has    1 votes
Limitless wins!
```

Try to solve Part 4B on your own before you turn to these hints!

Are you stuck on how to get started?

Open up the **`tv_shows_fxns.py`** *file and read the code, including the comments. Do your best to understand what the code needs to do – worry about how to do it afterwards.*

Are the votes not updating?

You might have one of two problems. First, are you remembering to save the **`choice`** *the user made by using the assignment operator? Second, are you remembering to add that vote to the* **`votes`** *list?*

Stuck on how to start writing `getWinner()`?

You'll need to use a **`for`** *loop to iterate through the contents of the* **`votes`** *list, and find the show that got the most votes.*

Not sure how to find the name of the winner?

You'll need to find two things: the current max number of votes, and where in the list that current max was found. You can do it in a single **`for`** *loop, but it may be easier to use two* **`for`** *loops: one to find the max, and a second to find where in the list the max is located.*
(If you need to use two loops, you are highly encourage to implement it with one for loop on your own time.)

Did you get an error like the following?

```
TypeError: unorderable types: str() > int()
```

When a function is called, the program needs to pass in actual parameters in the same order as the function's formal parameters. Try switching the order of the two parameters when you call the **`getWinner()`** *function.*

Part 5: Completing Your Lab

To test your program, first enable Python 3, then run **`tv_shows_fxns.py`**. Try a few different inputs to see how well your program works.

```
bash-4.1$ python tv_shows_fxns.py
1 - Daredevil
2 - Fargo
3 - Limitless
4 - Elementary
5 - Brooklyn 99
6 - Empire
7 - Supergirl
You and your friends are voting on a show to watch.
Which show would you like to vote for?
Enter '0' to stop voting: 6
Enter '0' to stop voting: 6
Enter '0' to stop voting: 2
Enter '0' to stop voting: 3
Enter '0' to stop voting: 0
[etc]
```

Since this is an in-person lab, you do not need to use the **`submit`** command to complete your lab. Instead, raise your hand to let your TA know that you are finished.

They will come over and check your work – they may ask you to run your program for them, and they may also want to see your code. Once they've checked your work, they'll give you a score for the lab, and you are free to leave.

IMPORTANT: If you leave the lab without the TA checking your work, you will receive a **zero** for this week's lab. Make sure you have been given a grade before you leave!