Lab 4 – CSE 101 (Fall 2019)

Objectives

The primary objectives are to learn the following:

i. Python “magic incantation”
ii. Command line arguments
iii. While loop
iv. Generating random numbers
v. Using the break statement
vi. Nested For loop

1. Python Magic Incantation

Refer to the following program while going through Part 1 and Part 2.

Celsius.py program:

```
from sys import argv
def celsius(f):
  "Convert temperature f to Celsius."
  return (f - 32) * 5 / 9
if __name__ == "__main__" :
  # print(argv)
  t = int(argv[1])
  print(celsius(t))
```

We import Python scripts to be able to use Python functions or classes defined in other files (e.g such as the math library). In general, if you import a Python program into your Python program, the imported program will be executed completely at the time of import.

Sometimes we want to be able to run a Python program by itself, while also being able to import that program into another Python program. However, we need a way to tell the Python interpreter what code should run in each situation. Using the `if __name__ == "__main__":` magic incantation

allows a Python script to run the code that follows ONLY if the file is run directly (and not run when imported into another file).

Demo: Save the above program and call it “Celsius.py” Using the command line, navigate to the folder where you saved the file and type the following commands to import and use the celsius function:

```
>>> python
>>> from Celsius import celsius
>>> celsius(90)
```

Remember: type python3 instead of python if you are using a Mac.

Your output should look something like this:

```
lab 4>python
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from Celsius import celsius
>>> celsius(90)
32.22222222222222
```

2. Using Command Line Arguments

Now let’s walk through what is happening in the Celsius.py program if you run it directly (NOT importing it into another file). First it reads an input string from the command line. Then, it converts that string to an integer, passes the integer to the celsius function, and then finally prints the result.

The other thing to notice about the program is this statement at the top:

```
from sys import argv
```

This is called an “import statement”, and what it is doing is bringing in functionality from another class into your program, so you can use that functionality without rewriting all the code. In this example, Celsius.py is using a standard library named `sys`. This library includes a variety of items used to connect a program to the operating system. In this case, the statement on line 1 imports `argv`, which represents the collection of arguments the user typed on the command line as a list. The name stands for “argument vector”.

Demo: Assuming you are still in the same folder you saved Celsius.py in (if not, navigate back to that folder), type the following commands into the command prompt to run the Celsius.py program without and with command line input.

```
python Celsius.py
python Celsius.py 90
```

Your output should look something like this:

```
lab 4>python Celsius.py
Traceback (most recent call last):
  File "Celsius.py", line 7, in <module>
    t = int(argv[1])
IndexError: list index out of range


lab 4>python Celsius.py 90
32.22222222222222
```

Notice that you can expect an error the first time you try to run Celsius.py *without* an argument — this is because you have included `argv`, which indicates to your program that it is expecting inputs from the command line to pass into the celsius(f) function.

For a more detailed and in-depth explanation, follow the tutorial project on Page No. 83 of the textbook Explorations in Computing.

Exercise: Add the `argv` import statement and "magic incantation" lines to the sieve.py program (Posted online as part of the Chapter 3 programs). This will turn it into a command line application which you can then call from the command line and pass arguments to. After you do this you should be able to:

1. Make it so you can run the program from the command line to make a list of primes up to 100.
2. Run the Python interactive console in the program's directory and import the sieve program to call the sieve function directly from the console.

3. How to Construct While Loops

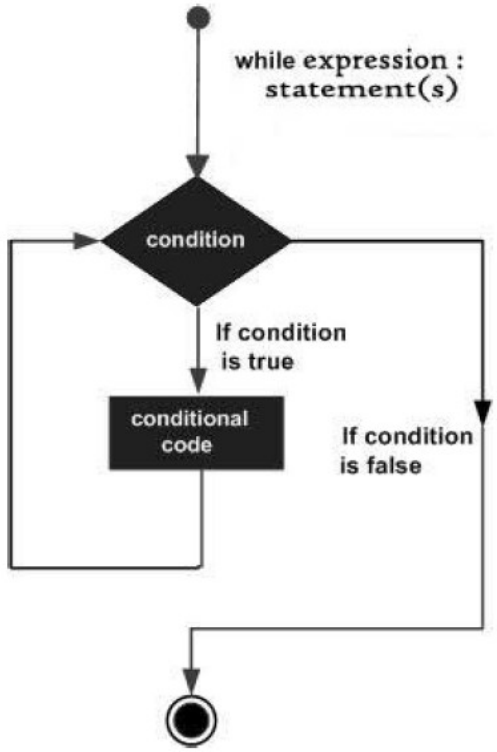

Figure 1: Illustration of While loop

A while loop implements the repeated execution of code based on a given Boolean condition. The code that is in a while block will execute as long as the while statement evaluates to True. You can think of the while loop as similar to a for loop, except that it repeats indefinitely rather than a set amount of times. In a while loop, the program will jump back to the start of the while statement after it finishes executing whatever is in the while block, until the original condition is False. While loops are especially useful if you don't know how many times you'll need to repeat something when you write the code.

In Python, while loops are constructed like so:

```
while [a condition is True]:
     [do something]
```

Example: In the following program, we will ask for the user to input a password. After the user enters a password there are two possible outcomes: If the password is correct, the while loop will exit. If the password is not correct, the while loop will continue to execute and ask for the password again.

Password.py program:

```
password = 'secret'
userInput = ''
while userInput != password:
     print('What is the password?')
     userInput = input()
print('Yes, the password is ' + password + '. You may enter.')
```

4. Generating and guessing random numbers

Using the `random` module, we can generate a random numbers. The function `random()` generates a random floating point number between zero and one [0, 0.1 .. 1]. As an aside, note that these numbers are not truly random and thus called “pseudorandom”, but they are enough random for most purposes.

Generate a random number between 0 and 1.

We can generate a (pseudo) random floating point number with the following code. Note that the line `from random import *` means, from the `random` module, import *all classes and functions.* The * means “everything”.:

```
from random import *
# Generate a pseudo-random number between 0 and 1.
print (random())
```

Generate a random number between 1 and 100

To generate a whole number (integer) between one and one hundred use the following slightly different function, still imported from the `random` module:

```
from random import *
# Pick a random number between 1 and 100.
print (randint(1, 100))
```

This will print a random integer. If you want to store it in a variable to use in a later part of your code, you can do so like this:

```
from random import *
# Get a random number between 1 and 100 and store in the ‘x’
variable
```

```
x = randint(1, 100)
print (x)
```

Random number between 1 and 10 [what's the difference between random() and uniform() ??]

To generate a random floating point number between 1 and 10 you can use the uniform() function.

```
from random import *
print (uniform(1, 10))
```

You can use the `random()` and `randint()` functions in many different ways in your code. Let's look at some examples of how you might use randomization in real scenarios:

Mixing up a list, like you are shuffling a deck of cards:

```
from random import *
items = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
shuffle(items)
print (items)
```

Picking an option from a list of options, like picking somewhere to eat for lunch:

```
from random import *
items = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
# Pick a random item from the list
x = sample(items, 1)
print (x[0])
# Pick 4 random items from the list
y = sample(items, 4)
print (y)
```

You can use the random functions with lists of strings too:

```
from random import *
items = ['Alissa','Alice','Marco','Melissa','Sandra','Steve']
```

```
# Pick a random item from the list
x = sample(items, 1)
print(x)
# Pick 4 random items from the list
y = sample(items, 4)
print(y)
```

5. Python break statement

In Python, `break` and `continue` statements can alter the flow of a normal loop. The break statement terminates the loop in which you call break. This causes the program to flow immediately to the next statement after the body of the loop. If you call `break` inside a nested loop (a loop inside another loop), break will stop the innermost loop, but keep executing the outer loop.

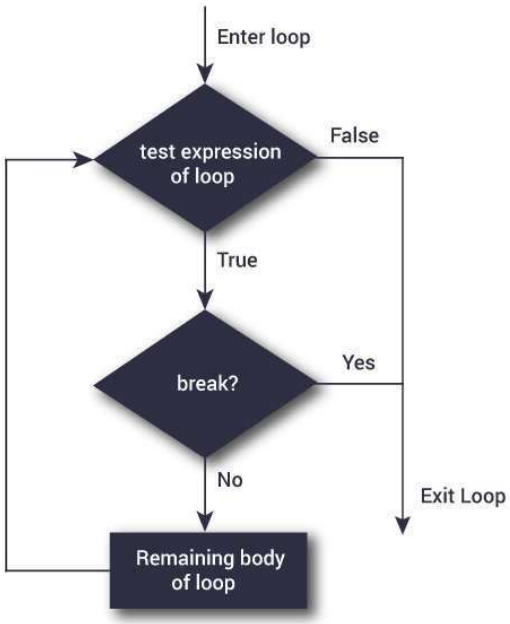

Figure 2: Illustration of break statement

Exercise: What does the following program guess.py do? Write your answer in the file explanation.txt and submit it on Blackboard.

guess.py program

```
import random
number = random.randint(1, 25)
number_of_guesses = 0
while number_of_guesses < 5:
```

```
    print('Guess a number between 1 and 25:')
    guess = input()
    guess = int(guess)
    number_of_guesses += 1
    if guess == number:
        break
```

6. Nested For Loops

Loops can be nested in Python, as they can with other programming languages. A nested loop is a loop that occurs within another loop, structurally similar to nested if statements. These are constructed like so:

```
 # Outer Loop
for [first iterating variable] in [outer loop]:
     [do something]
     # Nested Loop
     for [second iterating variable] in [nested loop]:
          [do something]
```

Here are some code examples of nested loops:

nestedloop.py program

```
num_list = [1, 2, 3]
alpha_list = ['a', 'b', 'c']
for number in num_list:
     print(number)
     for letter in alpha_list:
          print(letter)
```

pyramid.py program

```
for i in range(1,6):
     for j in range(i):
          print("*",end=' ')
     print("\n",end='')
```

Exercise: Create the pyramid similar to the above pyramid.py program using a while loop. Name your program whilepyramid.py and submit it on blackboard.

Exercise: Write a program named factorial.py that contains the following two functions:

```
def while_factorial(num)
def for_factorial(num)
```

These should calculate the factorial of a given number represented by the argument `num` using a while loop and a for loop respectively.

7. Submit the following files on blackboard.

1. sieve.py (modified to include the magic incantation and take command line arguments) – 2 pts
2. explanation.txt (explanation of guess.py program) – 1 pt
3. whilepyramid.py (Pyramid printing using while loop) – 1 pt
4. factorial.py (factorial program that include factorial functions using for loop and while loop) – 1 pt

Note: The topics for this lab are taken from following textbooks:

1. Explorations in Computing: An Introduction to Computer Science and Python Programming by John S. Conery. Chapman and Hall/CRC, 2014. ISBN 978-1466572447.
2. How to Code in Python 3 by Lisa Tagliaferri, Digital Ocean, New York, NY.