Computer Project 2
Get To Know Python!

DUE: 10/26/22

Instructions: Create a Python file that solves the problems given below. For a nice overview of the language, take a look at the free *A Byte of Python* tutorial which can be found at the link below.

`https://python.swaroopch.com/`

You don't need to look through all of the topics covered in the tutorial. Going through the section on Data Structures is probably enough for our purposes.

1) **Variables and Lists**

   a) Create a list which consists of the numbers from 0 to 100 (including 100). Assign this list to a variable.

   b) Print out the last 20 elements of the list. You may need to use the syntax `print(*list_name)` depending on how you defined your list!

   c) Print out the even elements of the list.

   d) Print out the elements of the list that are divisible by 3.

   e) Create a list of the first 100 perfect squares (starting with 0) using list comprehension.

2) **Create a Function**

   a) Create a function in Python called `C2F` which converts degrees Celsius to degrees Fahrenheit. The output should be a real number.

   b) Create a function in Python called `F2C` which converts degrees Fahrenheit to degrees Celsius. The output should be a real number.

   c) Create a function in Python called `degreeConvert` which takes two arguments: a real number and either the letter `'C'` or the letter `'F'`. If the letter is `'C'`, you should convert the given real number to degrees Fahrenheit; if the letter is `'F'`, you should convert the given real number to degrees Celsius. If any other letter is supplied, you

should print a message reporting that fact. The output should look like the following.

```
>>> degreeConvert(45, 'C')
45 degrees C = 113 degrees F
>>> degreeConvert(59, 'F')
59 degrees F = 15 degrees F
>>> degreeConvert(30, 'G')
Usage Error:  Expected either F or C for temperature scale!
```

Notice that your function only has to print messages! You do not have to specify a return value.

3) **Recursively Defined Functions**

Like most modern computer languages, Python allows you to define functions recursively. That is, you are allowed to call the function you are defining in the body of the definition! Care must be taken to avoid getting into an infinite recursion, however.

A classic example is the factorial function. We know that $n! = n \cdot (n-1)!$ which allows us to reduce computing factorials to an easier case. In order to stop the process, we need a *base case*. For factorials, we can use $0! =$. The following code is a recursive implementation of factorial.

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

a) Write a recursive implementation of the function `new_pow(x,n)` which returns x^n where n is a non-negative integer. For the base case, use `new_pow(x,0) = 1`.

b) Write a recursive implementation of the function `Fibonacci(n)` which returns the n-th Fibonacci number. This function requires two base cases.

```
Fibonacci(0) = 0
Fibonacci(1) = 1
```

4) **For Loops in Python**

a) Rewrite the `factorial` function above iteratively. That is, compute `factorial(n)` by using a for loop to accumulate the final answer.

b) You can directly iterate over any list in Python as follows.

```
myList = ['Robert', 'Thomas', 'Kathy', 'Brent']
for name in myList:
    print(f"{name} is in the Differential Equations course.")
```

Create a list of the first 100 perfect squares as in problem 1e). Iterate over this list printing out only the perfect squares that are divisible by both 3 and 5.

5) **Importing Modules in Python**

There are a huge number of modules developed for Python that add new functions and tools once they are imported. An extremely important one is the Random Module which can be imported with the following code.

```
import random
```

This module defines a number of functions that generate (pseudo)-random numbers.

a) The function `random.randint(a,b)` selects a random integer N in the range $a \leq N \leq b$ (note that b is included here). Use this function to create a list of 100 random integers between 1 and 100 (including 100 as a possibility). Iterate through this list counting how many are in the ranges $1 \leq N < 25$, $25 \leq N < 50$, $50 \leq N < 75$, and $75 \leq N \leq 100$.

b) The function `random.uniform(a,b)` selects a random floating point number r in the range $a \leq r \leq b$ (note that b is also included here). The special case `random.uniform(0,1)` is equivalent to the function `random.random()`. Use this function to create a list of 100 random floating point numbers between 0 and 10 (including 10 as a possibility). Iterate through this list counting how many are in the ranges $0 \leq r < 2$, $2 \leq r < 4$, $4 \leq r < 6$, $6 \leq r < 8$, and $8 \leq r \leq 10$.

c) The function `random.choice(lst)` selects a single random element from the list `lst`. Create a function `chooseLetters(n)` that returns a list of length n containing randomly selected letters from the list

`letters = ['A','B','C','D','E','F','G','H','I','J']`.

Repeats are allowed (and necessary if $n > 10$)!

6) **The `Pyplot` Module and Graphing in Python**

The `Pyplot` module is a submodule of `Matplotlib` which will allow us to make decent plots in Python. We can import this submodule using the following command.

```
import matplotlib.pyplot as plt
```

Notice that the `as` keyword allows us to alias the longer name of the submodule as the more manageable `plt`. If you leave off that portion of the `import` statement, you would need to type `matplotlib.pyplot.command` instead of `plt.command` for any commands you want to enter from the module. A nice overview of the `Pyplot` module can be found at the site below.

`https://matplotlib.org/stable/tutorials/introductory/pyplot.html`

a) Plot a graph of $y = x^2$ on the interval $-5 \leq x \leq 5$. In order to do this in Python, you need to create a list `X` which consists of a number of points uniformly chosen over the interval $[-5, 5]$. For this problem, use at least 1000 points (and make sure the last point is 5). You then create a list `Y` which consists of `x**2` for each `x` in the list `X`. You can then plot the graph with the following code.

```
plt.plot(X,Y)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Graph of y = x^2')
plt.show()
```

There are many other options you can include before displaying the graph with the command `plt.show()`. Notice that once you close the pop–up window with the graph, the entire plot is gone from memory!

b) Repeat part a), but display the graphs of $y = \sin(x)$ and $y = \cos(x)$ on the same axes in two different colors. Reuse your list of x-values, `X`, from part a), but you will need two lists for the y-values (say `Y1` and `Y2`). Note that you will need to include `import math` and call the trig functions by `math.sin(x)` and `math.cos(x)`. The code is very similar to the previous example.

```
plt.plot(X,Y1,color='blue')
plt.plot(X,Y2,color='red')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Graphs of Sine and Cosine')
plt.show()
```

The total list of named colors can viewed at

`https://matplotlib.org/stable/gallery/color/named_colors.html`.

Also note that common colors like blue, red, black, etc. can be specified without the `color = 'colorname'` syntax. The tutorial mentioned at the beginning of this section should have the details.