Engineering 13300
Python 3: Loops in Python

This assignment has three sections.

Team Hand Tracking - The first section has Team Exercises that are intended to be done with your team without actually using Python. Work the problems together and answer the questions and present your answer to one of the instructional team. After you find the correct answers, you can move to the second section. There are no files to turn in but you must complete this section before advancing to the next tasks.

Team Tasks - The second section has four team tasks (Tasks 1-4) that you may work together and develop code together as long as you document who contributed.

Individual Tasks - The third section has one task (Tasks 5) that is an individual task. You can work together to conceptually help each other but each person should work on their own code.

Team Hand Tracking

A) What is the output of the following code when run after all the variables have been cleared with the command reset?

```
x=3
y=6
while i<10:
    for z in [x,y]:
        i+=z
    if x<3:
        x+=1
print(i)
```

B) What is the output of the following code when run after all the variables have been cleared with the command reset?

```
i=1
x=3
y=6
while i<10:
    for z in [x,y]:
        i+=z
    if x<3:
        x+=1
print(i)
```

C) The following commands were entered into the console window of Python. What appears on the screen after they are executed?

```
x=[3,6,-2,1]
y=0

for index in x:
    if index <= 3:
        y=y+index
print(f'y={y}')
```

D) The following commands were entered into the console window of Python. What appears on the screen after they are executed?

```
x=[3,6,-2,1]
y=0

for index in x:
    if index <= 3:
        y=y+index
    print(f'y={y}')
```

Team Tasks

Recall the guidelines for team activities:

1. You should work as a team; **all** team members will be held responsible for all material. You may work together and contribute to one program and submit similar codes as long as the contributors to the development of the solution are documented.
2. Each student is responsible for submitting their own assignment.

Task 1 (of 5) [Team]

Learning Objective: Use for and while loops to conduct repetitive operations in Python.

Background: Many mathematical discoveries represent patterns observed in nature. For example, the number of curved rows of seeds in a sunflower can be defined by a Fibonacci number. Fibonacci numbers are a series of numbers that begin with $F_0 = 0$, then $F_1 = 1$. From that point the Fibonacci numbers are defined by the linear recurrence equation:

$$F_n = F_{n-2} + F_{n-1}$$

Therefore, $F_2 = 0 + 1 = 1$ and $F_3 = 1 + 1 = 2$. Likewise, $F_4 = 3$ and $F_5 = 5$. Determining numbers for large values of *n* would require a long series of repetitive computations based on this pattern. Repetitive loop structures are perfect methods for performing these computations. The following set of tasks explores various mathematical series as a method to practice loop constructs in Python.

Part A

In a PDF, save a flow chart of an algorithm that calculates the **Fibonacci numbers** to the n^{th} term. Then, write a Python program that calculates **Fibonacci numbers** to the n^{th} term.

1) Input value is a positive integer value '`n`'.
2) Output is the Fibonacci sequence to the n^{th} term using the general convention that the first Fibonacci term will be 0 followed by 1.

Example with an input value of n=12:

```
0,1,1,2,3,5,8,13,21,34,55,89
```

Save your files as:

1. `Py3_Team_`*`teamnumber`*`.pdf`
2. `Py3_Task1A_`*`teamnumber`*`.py`

Part B

Modify your flow chart and Python program from **Part A** such that the user inputs a maximum value for the series and the series will be computed until that value has been exceeded.

Example with a maximum value for the series initialized to 122

`0,1,1,2,3,5,8,13,21,34,55,89,144`

Save your modified flow chart as a separate page in the previously created PDF file. Additionally, save the modified Python program as:

`Py3_Task1B_`*`teamnumber`*`.py`

As a separate page in the PDF, include answers to the following questions:

1. When do you use `for` and `while` loops in programming?
2. What is the syntax of `for` and `while` loop in Python?
3. How do `for` and `while` loops work in Python?
4. When do `for` loops and `while` loops terminate?
5. What is the output of the piece of code given below? Identify the number of times the code in the 'for' statements will run.

```
units = []
for x in range(0, 6):
    print("Adding", x, "to the list.")
    units.append(x)

for x in units:
    print("List consists of:", x)
```

Task 1 Files:

1. `Py3_Team_`*`teamnumber`*`.pdf`
2. `Py3_Task1A_`*`teamnumber`*`.py`
3. `Py3_Task1B_`*`teamnumber`*`.py`

Task 2 (of 5) [Team]

Learning Objective: Predict the output of a complete `for` and `while` loop in Python.

Background: Another common mathematical series is a factorial, which is often used in figuring out combinations of events. For example, the number of ways to arrange the letters A, B, C without repeating the letters is determined using a factorial. The factorial of a positive integer, n, is defined as:

$$n! = n * (n-1) * (n-2) * \cdots * 2 * 1$$

(By convention, 0! = 1 and the factorial calculation cannot be performed with a negative number) Therefore, the three letters A, B, and C can be arranged in 3! = 3*2*1 = 6 ways; specifically, the arrangements are [ABC], [BAC], [BCA], [CBA], [CAB], and [ACB]. As n increases it becomes difficult to determine these permutations. By using a computer and loop structure, however, computing the factorial becomes a snap.

Create a flow diagram of an algorithm programmed as a sub-process that will calculate the factorial of a number passed as an argument. The function should return an error flag value (such as -999) if the number passed is negative, since the factorial function can only be calculated for non-negative integers. Save the flow chart in a separate page in the PDF file.

Next, write a Python program that defines a function `MyFactorial` to compute the factorial of a number using your flow chart.

Example 1 with 5 passed as the value for n:

```
The Factorial of 5 is 120.
```

Example 2 with -5 passed as a value of n:

```
Error[Negative input].
```

Save your main program as:

`Py3_Task2_`*`teamnumber`*`.py`

In a separate page in the PDF, answer the following questions:

1. Is it more appropriate to use a `for` loop or `while` loop for this task? Justify this answer.
2. Could you have used the other form of loop to solve this problem? Justify this answer

Task 2 Files:

1. `Py3_Team_`*`teamnumber`*`.pdf`
2. `Py3_Task2_`*`teamnumber`*`.py`

Task 3 (of 5) [Team]

Learning Objectives: Predict the output of a complete `for` and `while` loop in Python; Create, manipulate, and read from arrays, lists, and dictionaries in Python; Manipulate and extract information from lists, arrays, and dictionaries in Python; Use loops to compare and evaluate associated elements in data sets in Python.

Background:
The use of functions reduces duplication of code and decomposes complex problems into simpler pieces. For example, we often need to compute values for a large set of data inputs. Therefore, we define a function to perform the basic operation, then repeatedly call that function based on how many values we have to process. This saves time and effort of having to retell the computer what to do every time it does a common task. This next task provides a basic example of this situation.

Part A
Generate a flow chart that illustrates the process of accepting a list of values from the user and computing the factorial for each value using the sub process developed in **Task 2**. Save the flow chart in your previously created PDF.

Part B
Copy only the user defined function `MyFactorial` you created in **Task 2**, and save it into a new file named `Py3_Task3_factorial_`*`teamnumber`*`.py`. Now write a main program which will use the function `MyFactorial` contained in module `Py3_Task3_factorial_`*`teamnumber`*`.py`, to calculate the factorials of a list of numbers (i.e. when your main program runs, it should import module `Py3_Task3_factorial_`*`teamnumber`*`.py` and call the `MyFactorial` function). The list of numbers must be pre-defined in the main program, and must be in the form of a space delimited string (Hint: You may use `split()` to convert a string to list). Make sure to write your program such that it computes factorials for any number of values that are to be provided. Your main program must be named in the following format.

`Py3_Task3_`*`teamnumber`*`.py`

Example:
```
Numbers whose factorials will be evaluated:
-5 2 3
Error: factorial routine requires positive integers.
The factorial of 2 is 2.
The factorial of 3 is 6.
```

As a separate page in the previously created PDF file, answer the following questions:

1. How did loops help you in performing this task?
2. Provide an example problem that an engineer would use a loop to solve.

Task 3 Files:

1. `Py3_Team_`*`teamnumber`*`.pdf`
2. `Py3_Task3_factorial_`*`teamnumber`*`.py`
3. `Py3_Task3_`*`teamnumber`*`.py`

Task 4 (of 5) [Team]

Learning Objectives: Use for loops to conduct repetitive operations in Python; use while loops to conduct repetitive operations in Python; use loops to evaluate corresponding data sets (i.e. lists, arrays, and dictionaries) in Python.

Background: A **Maclaurin series** is an n^{th} degree polynomial that can be used to approximate a function around $x = 0$. The exponential function, e^x, can be approximated as follows:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \cdots$$

Where the infinite series approximation converges with the actual function over all real numbers. Practically, for a finite approximation, the fewer terms that are used, the less accurate the approximation will be, particularly as x gets further from 0.

Part A

Using your `MyFactorial` function as in **Task 3**, write a main program which will use the Maclaurin series defined above to approximate e^x with specified values of n and x. The program should also print to the screen the percent error of the estimated value with respect to a calculation of the actual value. Save the flow diagram as a separate page in the previously created PDF file. Note that ask user to input n, and x in this order.

Example with an initialized value of n = 5 and x = 3:

```
Approximate value:    18.40
Actual value:         20.09
Error:                -8.4%
```

Save your files as:

`Py3_Task4A_`*`teamnumber`*`.py`

Part B

This time, write a main function that will generate as many terms as is necessary to approximate the function to a specified level of accuracy (i.e., below an percent error threshold) at a specified value of x. Additionally, print to the screen the number of terms the series must contain before the target accuracy is achieved. Save the flow diagram as a separate page in the previously created PDF. Note that ask user to input x, and target error threshold in this order.

Example with an initialized value of x = 3 and target error threshold of 5%:

```
Target error threshold:    5%
Actual value:              20.09
Terms needed:              7
Approximate value:         19.41
```

Save your file as:

`Py3_Task4B_`*`teamnumber`*`.py`

As a separate page in the previously created PDF, answer the following questions:

1. How did loops help you in performing this task?
2. Could you solve this problem without using loops?

Task 4 Files:

1. `Py3_Team_`*`teamnumber`*`.pdf`
2. `Py3_Task4A_`*`teamnumber`*`.py`
3. `Py3_Task4B_`*`teamnumber`*`.py`

Individual Task

Guidelines for Task 5:
This task is an individual task. You may seek help from classmates, the instructional team or others but the work you submit should be your own. If you collaborate with others and use information developed together or by someone else, ALWAYS document and reference that material.

Task 5 (of 5) [Individual]

Objectives: Predict the output of a complete `for` and `while` loop in Python; Use loops to conduct repetitive operations to perform numerical operations in Python; Use loops to create corresponding data sets (i.e. lists, arrays, and dictionaries) in Python; Create, manipulate, and read from arrays, lists, and dictionaries in Python; Manipulate and extract information from lists, arrays, and dictionaries in Python; Use loops to compare and evaluate associated elements in data sets in Python.

Background:
It is common that an engineer must evaluate or perform operations on a variety of functions. One such operation is to find the area under the function—what is known in Calculus as taking an anti-derivative or integral of the function. Many common functions can be integrated to produce another function, leading to relatively simple calculations (e.g., the antiderivative of $\sin x$ is $-\cos x$). However, there are occasionally functions that do not integrate into another common function (e.g., the antiderivative of $\sin(x^2)$ is $\int \sin(x^2)\,dx$).

When we want to evaluate a definite integral (i.e., the area under the curve from one *x* value to another) of a function that has a simple antiderivative, we can apply the Fundamental Theorem of Calculus. This theorem says:

$$\int_a^b f(x)\,dx = F(b) - F(a)$$

where $F(x)$ is the antiderivative of $f(x)$. For the example of a simple function, $\sin x$,with antiderivative $-\cos x$, this is a straightforward task:

$$\int_0^{\pi} \sin x\,dx = -\cos\pi - (-\cos 0) = 1 + 1 = \boxed{2}$$

However, when the function does not have a nice integral, numerical methods are necessary. Maclaurin series, like those you studied in **Task 4** and their more generalized counterpart, **Taylor series**, can be used to estimate the function using a polynomial, which will always integrate easily.

The Maclaurin series for $\sin(x^2)$:

$$\sin(x^2) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{4n+2}}{(2n+1)!}$$

The antiderivative of this series can be found by increasing the exponent of each term by 1 and dividing each term by the new exponent:

$$\int_a^b \sin(x^2)\,dx = \left[\sum_{n=0}^{\infty}(-1)^n \frac{x^{4n+3}}{(4n+3)\times(2n+1)!}\right]\Bigg|_a^b$$

Where the vertical line after the summation tells you the limits when applying the fundamental theorem of calculus. That is, you plug b in for x and evaluate, then plug a in for x and evaluate, then subtract the second from the first.

Because the series above is an *alternating series* (i.e., its terms approach 0 and switch between being positive and negative), the series will converge to a value. If you only care about the sum being accurate to a given decimal place, you can stop the sum once you've reached stability to that level of rounding (i.e., the sum no longer changes at that level of rounding).

Task:

Write a program that uses a Maclaurin series for $\sin(x^2)$ to compute the integral over a specified interval from $[a, b]$. You will specify the decimal place to which you would like the sum to converge. However, because more terms are needed to converge to more decimal places, you will set an upper limit for the maximum number of terms that will be calculated. You will ask the user to input (in this order, use the `input()` function):

1. The lower limit of integration, a
2. The upper limit of integration, b
3. The number of decimal places for convergence
4. The maximum number of terms to calculate

Your program will need to handle situations when the user enters an unacceptable value for any of the inputs:

- The lower and upper limits of integration should be either an integer or a floating-point number (i.e., it cannot be a string or complex number). If the user does not input an integer or floating-point number, **terminate the method** and print an appropriate error message to the screen beginning with "Error 1:".
- The number of decimal places and the maximum number of terms must both be whole numbers (i.e., they must be positive integers). If non-positive integers are entered, **terminate the method** and print an appropriate error message to the screen beginning with "Error 2:".

Your program should output the estimated value for the integral after each term is added. The program should stop iterating (i.e., adding new terms) once the rounded value of integral does not change after **three** consecutive iterations. If the program reaches the maximum number of terms specified before converging, then the program should output an appropriate error message to the user.

Develop a flow diagram to represent the program described above. Include the checks on the input values as a subroutine and show its logic in a separate flowchart. Save your file in a PDF called:

`Py3_Ind_username.pdf`

Translate your flow diagrams into Python program.

Hints:

- The built-in function `round()` might be useful.

Sample Input and Output 1:
This example is to show the expected format of the output (inputs bold). Please note that these are the actual values you should obtain for the specified inputs.

```
Enter the lower limit of integration: -2
Enter the upper limit of integration: 3
Enter the number of decimal places for convergence: 3
Enter the maximum number of terms: 20

Approximations:
n = 0: sum = 11.667
n = 1: sum = -43.452
n = 2: sum = 92.302
n = 3: sum = -97.932
n = 4: sum = 70.717
n = 5: sum = -31.835
n = 6: sum = 13.521
n = 7: sum = -1.716
n = 8: sum = 2.303
n = 9: sum = 1.449
n = 10: sum = 1.598
n = 11: sum = 1.576
n = 12: sum = 1.579
n = 13: sum = 1.579
n = 14: sum = 1.579

The integral from -2.0 to 3.0 is estimated to be 1.579.
Total number of terms: 15
```

Sample Input and Output 2:

```
Enter the lower limit of integration: 3
Enter the upper limit of integration: 4
Enter the number of decimal places for convergence: 6
Enter the maximum number of terms: 9

Approximations:
n = 0: sum = 12.333333
n = 1: sum = -325.690477
n = 2: sum = 2717.610281
n = 3: sum = -11295.523542
n = 4: sum = 28403.78919
n = 5: sum = -48140.859548
n = 6: sum = 58959.716583
n = 7: sum = -54787.362814
n = 8: sum = 40042.414756

Error: The approximation did not converge to 6 decimal places
with only 9 terms.
```

Save your file as:

Py3_Task5_*username*.py

For user-defined functions, define them within the following file:

Py3_Task5_functions_*username*.py

Task 5 Files:

1. Py3_Ind_*username*.pdf
2. Py3_Task5_*username*.py
3. Py3_Task5_functions_*username*.py

Summary of Submit Files: Submit *all* files with required filenames electronically via gradescope to the appropriate box on time.

1. Py3_Team_*teamnumber*.pdf
2. Py3_Task1A_*teamnumber*.py
3. Py3_Task1B_*teamnumber*.py
4. Py3_Task2_*teamnumber*.py
5. Py3_Task3_*teamnumber*.py
6. Py3_Task3_factorial_*teamnumber*.py
7. Py3_Task4A_*teamnumber*.py
8. Py3_Task4B_*teamnumber*.py
9. Py3_Ind_*username*.pdf
10. Py3_Task5_*username*.py
11. Py3_Task5_functions_*username*.py