Engineering 13300 HW7 PY3
Python 3: Loops in Python

Individual Tasks

This assignment has one section. This 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.

Guidelines for Task 5:

1. These two tasks are individual assignments. 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.
2. Each individual is responsible for submitting their own individual assignment to Gradescope.

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, **end the program** and print an appropriate error message to the screen as "Error 1: Input integer or floating-point number".
- The number of decimal places for convergence and the maximum number of terms must both be positive integers only. If non-positive integers are entered, **end the program** and print an appropriate error message to the screen as "Error 2: Input a positive integer".

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:

1. The built-in round() function can be helpful when comparing the sum between iterations.

2. Handling exceptions – How to get your program to end if the user doesn't input an integer or floating-point number:
   Python is flexible enough to handle selected exceptions. In the example code below, which asks the user for input, until a valid floating-point number is entered (note – floating-point accepts integers as well). If any other datatype is entered – say a string, ValueError is returned from the input statement (line 2) and thus, leading to the except clause to kick in, which prints the following error message. And, in the end, having the code to "return" will end the program and you get to rerun the program from the top.

```
try:
    a = float(input("Enter the lower limit of integration: "))
except ValueError:
    print("Error 1: Input integer or floating-point number")
    return
```

   For more information on Handling exceptions – refer to section "8.3 – Handling Exceptions" of the Python official documentation - https://docs.python.org/3/tutorial/errors.html

   Save your file as:

   `Py3_Task5_`*`username`*`.py`

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

   `Py3_Task5_functions_`*`username`*`.py`

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: -1

Enter the upper limit of integration: 4

Enter the number of decimal places for convergence: 3

Enter the maximum number of terms: 30

Approximations:
n = 0: sum = 21.667
n = 1: sum = -368.452
n = 2: sum = 2809.051
n = 3: sum = -11393.883
n = 4: sum = 28474.003
n = 5: sum = -48173.189
n = 6: sum = 58972.743
n = 7: sum = -54789.574
n = 8: sum = 40044.223
n = 9: sum = -23661.791
n = 10: sum = 11556.417
n = 11: sum = -4745.073
n = 12: sum = 1664.716
n = 13: sum = -502.759
n = 14: sum = 134.254
n = 15: sum = -29.962
n = 16: sum = 7.471
n = 17: sum = -0.128
n = 18: sum = 1.255
n = 19: sum = 1.028
n = 20: sum = 1.061
n = 21: sum = 1.057
n = 22: sum = 1.057
n = 23: sum = 1.057

The integral from -1.0 to 4.0 is estimated to be 1.057.
Total number of terms: 24
```

Sample Input and Output 2:
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: 2.75

Enter the upper limit of integration: 5.3

Enter the number of decimal places for convergence: 5

Enter the maximum number of terms: 10

Approximations:
n = 0: sum = 42.69337
n = 1: sum = -2725.9188
n = 2: sum = 67442.51733
n = 3: sum = -899928.66685
n = 4: sum = 7470028.08875
n = 5: sum = -42127647.70403
n = 6: sum = 171572287.20207
n = 7: sum = -527771388.8259
n = 8: sum = 1269107790.44061
n = 9: sum = -2451378308.40785

Error: The approximation did not converge to 5 decimal places with
only 10 terms.
```

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 to the respective assignment links on Gradescope 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