Engineering 14100
Python 3 ACT: Loops in Python

Recall the guidelines for activities:

1. You should work as a team; **all** team members will be held responsible for all material.
2. You should work on this Task using one computer for the entire team unless otherwise directed.
3. The time estimate given is approximately three times the amount required for an experienced user to complete the task. If you are not making progress, take action to get unstuck.
4. Do not write on the activity sheets and be sure to return them at the end of class.

Task 1 (of 4)

Objectives: Use for and while loops to conduct repetitive operations in Python.
Computer Operator: Person whose family lives the furthest away.

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:

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

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

Example with an initialized value of n=12:

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

Save your files as:

1) `Py3_ACT_`*`login`*`.pdf`
2) `Py3_ACT_Task1A_`*`login`*`.py`

Part B

Modify your flow diagram 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 and n=12

`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_ACT_Task1B_`*`login`*`.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 while 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_ACT_`*`login`*`.pdf`
2) `Py3_ACT_Task1A_`*`login`*`.py`
3) `Py3_ACT_Task1B_`*`login`*`.py`

Task 2 (of 4)

Objective: Predict the output of a complete for and while loop in Python.
Computer Operator: Person with the next birthday.

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

$$n! = n * (n - 1) * (n - 2) * \ldots . * 2 * 1$$

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; namely, the permutations are [ABC], [BAC], [BCA], [CBA], [CAB], and [CBA]. As n increases it becomes difficult to determine these permutations. By using a computer and loop structure, however, computing the factorial becomes a snap.

Part A

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 value (such as -1) if the number passed is negative, since the factorial function can only be calculated for non-negative integers. Save the flow diagram 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 diagram.

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_ACT_Task2_`*`login`*`.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 File:

1) `Py3_ACT_`*`login`*`.pdf`
2) `Py3_ACT_Task2_`*`login`*`.py`

Task 3 (of 4)

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.
Computer Operator: Person with the most pets.

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 accepting a list of values from the user and computes the factorial for each value using the sub process developed in Task 2. Save the flow chart in your previously created PDF.

Part B
Modify the Python program written for Task 2 to reflect the changes described in the flowchart above. The input values can be initialized in a main program contained in the file called:

`Py3_ACT_Task3_`*`login`*`.py`

which will call the function `MyFactorial` contained in module `Factorial.py` (i.e. when file `Py3_ACT_Task3_`*`login`*`.py` runs, it imports module `Factorial.py` and calls `MyFactorial`). The initialization of the list of values in the main program will be given in one line and space delimited, and any number of values can be provided.

Example:
```
Please enter numbers whose factorials should 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 File:

1) `Py3_ACT_`*`login`*`.pdf`
2) `Factorial.py`
3) `Py3_ACT_Task3_`*`login`*`.py`

Task 4 (of 4)

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.
Computer Operator: Person who has been on a plane most recently.

Background: The **Perrin numbers** are defined by the linear recurrence equation:

$$P(n) = P(n-2) + P(n-3)$$

Where the following initial conditions are given:

$$P(0) = 3, P(1) = 0, P(2) = 2$$

Part A

Repeat the same steps as **Task 1 – Part A**, but instead of the **Fibonacci sequence**, output the **Perrin sequence** to the n^{th} term. Save the flow diagram as a separate page in the previously created PDF file.

Example with an initialized value of n=11:

```
3, 0, 2, 3, 2, 5, 5, 7, 10, 12, 17
```

Save your files as:

`Py3_ACT_Task4A_`*`login`*`.py`

Part B

Perform the same steps as **Task 1 - Part B**, but instead iterate the **Perrin sequence** until a target value is exceeded. Additionally, print to the screen the number of terms the series must contain before the target value is exceeded. Save the flow diagram as a separate page in the previously created PDF.

Example with an initialized value of n=15:

```
3, 0, 2, 3, 2, 5, 5, 7, 10, 12, 17
The series contain 11 terms before the target of 15 is exceeded.
```

Save your file as:

`Py3_ACT_Task4B_`*`login`*`.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_ACT_`*`login`*`.pdf`
2) `Py3_ACT_Task4A_`*`login`*`.py`
3) `Py3_ACT_Task4B_`*`login`*`.py`

Bonus Activity Submissions

Instructions: Complete and submit **ALL** Task materials associated with this Activity (see 'Submit Files' below). You are allowed to combine the work you and your team completed during the Activity with materials you individually (or as a team) complete outside of class. The Bonus Activity Submission will not be graded and returned to you like a typical assignment. Instead, it will be reviewed, and the bonus point awarded, for its completeness, i.e., for completing ALL the Tasks associated with the Activity. Submitting an incomplete Bonus Activity (something less than all of the Tasks) will be considered an act of **Academic Dishonesty** for which the penalty will be forfeiture of the opportunity to earn future Bonus Activity Submission points.

There are two options for completing the materials for the Bonus Activity Submission:

As an Individual: Combine the work you and your team completed in class with materials you have individually completed outside of class. When submitting an individual Bonus Activity Submission you will append your electric signature (i.e., your typed name) at the top of the file that represents your individual work. Your electronic signature indicates that this is your individual work and you have not collaborated with other individuals (other than the teaching team) to obtain the final materials being submitted – working with other individuals/groups (e.g., discussing ideas and concepts, helping find errors, talking about potential solutions to errors) is permissible up to the point where the work represents a coloration (i.e., working with another person or group to achieve an answer). Any work previously completed by your team should include each team member's electronic signature. The significance of an electronic signature by an individual for team work is stated below.

As a Team or Ad Hoc Group: Combine the work you and your team completed in class with materials your team (or ad hoc group) completed outside of class **(For the Bonus Activity Submission ONLY**: you are allowed to work with any other members of the class to complete the assignment). However, you should exercise care when appending your electronic signature to ensure you are in full compliance). When submitting a Bonus Activity that has been worked on as a team (or ad hoc group) each person will append his/her electronic signature (i.e., his/her typed name) at the top of the file that represents the collaborative work. The electronic signature of each individual implies he/she: was an active participant in the preparation of the materials; and has a general understanding of **ALL** the materials being submitted. Even for work submitted as a team, each individual who wishes to receive credit must submit the team's file (with all appropriate signatures) to their own individual assignment drop box.

Submit Files: Submit *all* files electronically via Blackboard to the appropriate box on time.

1) `Py3_ACT_`*`login`*`.pdf`
2) `Py3_ACT_Task1A_`*`login`*`.py`
3) `Py3_ACT_Task1B_`*`login`*`.py`
4) `Py3_ACT_Task2_`*`login`*`.py`
5) `Py3_ACT_Task3_`*`login`*`.py`
6) `Factorial.py`
7) `Py3_ACT_Task4A_`*`login`*`.py`
8) `Py3_ACT_Task4B_`*`login`*`.py`