Python Programming – Tutorial 04 and 05

Exercise 1 – Concept Revision in the Python Shell

Let's revise the concepts from this module by typing some simple statements into the **Python Shell**. Feel free to deviate from the workshop and experiment!

1. `nums = [1, 2, 4, 15]`
   Using the information in Lecture 3 and the Python documentation for data structures and built-in functions, write statements that manipulate the `nums` list in the following ways:
   (remember, you can check the content of a variable simply by typing its name)
   - ***a)*** *Change the last item in `nums` from `15` to `16`.*
   - ***b)*** *Insert a new item with a value of `8` at index `3` (i.e. between the `4` and the `16`).*
   - ***c)*** *Append a new item with a value of `32` to the end of `nums`.*
   - ***d)*** *Determine the `sum` of all of the items in `nums`.*
   - ***e)*** *Reverse the order of the items in `nums`.*

   - *Which of the previous tasks could be performed on `nums` if it was a tuple?*
     (remember, tuples are immutable)

2. `text = 'concatenation'`
   Using the information in the "Referring to Items" slides of Lecture 3, write statements that refer to parts of the `text` variable in the following ways:
   (remember, indexes start at 0 – the first letter of a string has an index of 0)
   - ***a)*** *The fifth letter of `text` ("a")*
   - ***b)*** *The third last letter of `text` ("i")*
   - ***c)*** *The first three letters of `text` ("con")*
   - ***d)*** *The fourth, fifth and sixth letters of `text` ("cat")*
   - ***e)*** *The last six letters of `text` ("nation")*

3. `range(1, 10)`
   While a range is similar to a list, it is not the same. Creating a range does not simply create a list of the appropriate numbers…

4. `list(range(1, 10))`
   …However you can convert a range to a list if you want a list of numbers that are easy to generate using a range.

5. `evenNumsBelow25 = list(range(2, 25, 2))`
   This creates a list containing [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24].

Exercise 2 – Simple Looping Program

Create a new file in the **Python Editor** and write the code to implement the following program. While this program does not achieve anything particularly useful, it practises your usage of a number of the concepts we've covered so far.

All of the information and examples needed are available in lectures and the Python documentation.

Before writing any code for the program described below, think about how you will approach it and *write pseudocode of your program design*.

Write a program that generates and displays random numbers between 0 and 10, only stopping when a 0 is generated. The output should include how many numbers it has generated. Below is an example of how the program's output should look, but it will change every time the program is run:

```
Number 1 was 8
Number 2 was 10
Number 3 was 3
Number 4 was 3
Number 5 was 9
Number 6 was 8
Number 7 was 2
Number 8 was 0
```

To generate random numbers, add the statement "**`import random`**" to the start of the program, and then use "**`random.randint(`***`min, max`***`)`**" to generate a random numbers. For example, this program would print a random number between 1 and 100 every time you run it:

Python

```
import random
num = random.randint(1, 100)
print(num)
```

Remember to write pseudocode of your program design before writing the code.

Once you have written and tested the program, modify it so that if the random number is a 7, the program prints "Lucky 7!" after the normal output. For example:

```
Number 1 was 7
Lucky 7!
Number 2 was 9
Number 3 was 6
Number 4 was 9
Number 5 was 7
Lucky 7!
Number 6 was 7
Lucky 7!
Number 7 was 6
Number 8 was 0
```

Task 3 – Number Guessing Game

Create a new file in the **Python Editor** and write the code to implement the following program.
As usual, you should write pseudocode of your program design before trying to write the code.

Write a program to implement a "guess the number" game that meets the following specifications:

- Generate a random number between 1 and 100 and prompt the user to guess what it is
- Use a while loop to repeatedly prompt the user to enter a guess
    - Each time the user enters a guess, add 1 to a "guessCounter" variable
    - If the guess is correct, print a congratulatory message which includes the number of guesses it took them to get the correct number, and end the program
    - If the guess is greater than the number, print a "Too high!" message
    - If the guess is less than the number, print a "Too low!" message

Below is an example of the program's output:

```
Guess the number between 1 and 100!
Enter your guess: 50
Too low!
Enter your guess: 75
Too high!
Enter your guess: 62
Too high!
Enter your guess: 57
Too low!
Enter your guess: 59
Too high!
Enter your guess: 58
58 is correct, you win!
You got it in 6 guesses.
```

Remember to write pseudocode of your program design before writing the code.

If you look at the "Break Example 1" slide of Lecture 3, you can see that there are two ways to
approach the while loop in this task:

1. Initialising the `guess`variable to `None`and using a loop condition of `guess != number`
2. Using an endless loop (`while True`) and using `break`to end it when the guess is correct

Once you have written the program using one approach, modify it to use the other approach. Remember to test your code!

Task 4 – Lap Time Recorder

Create a new file in the **Python Editor** and write the code to implement the following program.

Write a program to implement a "lap time recorder" that allows the user to input the times taken for a car to complete multiple laps of a racetrack, and see some basic statistics about the lap times. The program must meet the following specifications:

- Prompt the user to enter the number of lap times that will be entered
- Use a for loop to repeatedly prompt the user to enter that many lap times
    - The prompt to enter a lap time must specify the current and total number of lap times to be entered, e.g. "Enter lap time 3 of 5:"
- Once the specified number of lap times have been entered, the program should display the fastest (minimum) lap time, the slowest (maximum) lap time and the average lap time
    - Use the `round()` function to round the average lap time to two decimal places

There are two approaches which can be used to calculate the fastest, slowest and average lap time. This approach does not use any data structures, and does most of the processing during the loop. The pseudocode for this version of the program is below:

Pseudocode

```
Set total to 0
Set fastest to 9999
Set slowest to 0

Prompt user to specify how many laps

Loop for number of laps
    Prompt user to enter lap time

    If lap time is less than fastest
        Set fastest to lap time

    If lap time is more than slowest
        Set slowest to lap time

    Add lap time to total

Display fastest
Display slowest
Display total / number of laps (average)
```

Write the code for this version of the program, and test it to ensure that it is working correctly.

Next let's write a version of the program that uses a list. As you can see by the pseudocode below, it is much more efficient:

Pseudocode

```
Create an empty list named lapTimes

Prompt user to specify how many laps

Loop for number of laps
    Prompt user to enter lap time
    Append lap time to lapTimes list


Display minimum item in lapTimes (fastest)
Display maximum item in lapTimes (slowest)
Display sum of items in laptimes / length of lapTimes (average)
```

In this version, the only thing that occurs in the loop is to prompt the user for a lap time and append it to the lapTimes list (see the List Functions slide of Lecture 4).

After the loop, use the built in functions `min()`, `max()`, `sum()` and `len()` to determine the minimum (fastest), maximum (slowest) and average lap time. Much more efficient when using a list!

Task 5 – Enhancing the Lap Time Recorder

Now let's enhance the lap time recording program so that instead of needing to specify how many lap times will be entered, it prompts for lap times until the user enters a lap time of "x":

- Make sure this is mentioned in the prompt, e.g. "Enter lap time 3 ('x' to end):"
    - Note that the prompt cannot include the total lap count anymore, since we don't know what it will be – it should still show the current lap count
- You will no longer need the initial prompt to specify a number of laps, and since we no longer know how many times the loop needs to run, it will become a while loop
- The while loop will be endless, using `break`to end if an "x" is entered at the prompt

The Break and Continue slides of Lecture 3 should give you an idea of how to implement this.
As usual, you should write pseudocode of your program design before trying to write the code.

Below is an example of the program's output:

```
Enter lap time 1 ("x" to end): 23.4
Enter lap time 2 ("x" to end): 25.1
Enter lap time 3 ("x" to end): 27.3
Enter lap time 4 ("x" to end): 22.0
Enter lap time 5 ("x" to end): x
Fastest Lap Time: 22.0
Slowest Lap Time: 27.3
Average Lap Time: 24.45
```