2807/7001ICT Programming Principles (I), Trimester 3, 2019
Workshop 6

School of Information and Communication Technology
Griffith University

October 31, 2019

Module	2
When	Day 6
Goals	This workshop focusses on lists, indexing, slices, list methods, and/or tuples.
Marks	5
Due	Pre-workshop questions at the start of the workshop; problems by the *beginning* of the next workshop.

1 Preparation

Before your workshop class:

- Read all of this document.
- Review the lecture notes sections 1 to 17.
- Bring some paper (a print-out of this document is best) and writing implements.
- Bring a storage device, such as a portable hard drive and cable, or a USB drive.

2 Pre-workshop questions (1 mark)

Complete these questions in writing *before* the start of the workshop. They will be marked early in the workshop.

1. For each of these code snippets, how many times will the print statement execute?

   (a)
   ```
   for i in range(10):
       print(i)
   ```
   ______

   (b)
   ```
   for i in range(10):
       for j in range(3):
           print(i * j)
   ```
   ______

   (c)
   ```
   for i in range(5):
       for j in range(i):
           print(i + j)
   ```
   ______

   (d)
   ```
   i = 0
   while i < 10:
       print(i)
       i += 1
   ```
   ______

   (e)
   ```
   i = 0
   while i < 10:
       print(i)
       if i % 2 == 0:
           i += 1
   ```
   ______

2. What will each of these code snippet print? (Show it, don't describe it in words.)

   (a)
   ```
   for i in range(5):
       print('*')
   ```
   (b)
   ```
   for i in range(5):
       print('*', end = "")
   print()
   ```
   (c)
   ```
   for i in range(5):
       print('*' * i)
   ```
   (d)
   ```
   for i in range(5):
       print(' ' * i, '*', sep = "")
   ```
   (e)
   ```
   for i in range(5):
       print(' ' * (5 - i), '*', sep = "")
   ```

3 Workshop activities

3.1 Marking last workshop's problems

If you have problems that still need marking from the previous workshop, get them marked at the *start* of this one.

3.2 Problem 1 (1 mark)

Problem: Write a program that reads strings typed by the user until an empty string in entered, then prints all of the strings sorted into ascending (actually, non-descending, lexicographic) order.

```
Enter a string: On a little piece of wood,
Enter a string: Mr. Spikky Sparrow stood;
Enter a string: Mrs. Sparrow sate close by,
Enter a string: A-making of an insect pie,
Enter a string: For her little children five,
Enter a string: In the nest and all alive,
Enter a string: Singing with a cheerful smile
Enter a string: To amuse them all the while,
Enter a string: Twikky wikky wikky wee,
Enter a string: Wikky bikky twikky tee,
Enter a string: Spikky bikky bee!
Enter a string:
A-making of an insect pie,
For her little children five,
In the nest and all alive,
Mr. Spikky Sparrow stood;
Mrs. Sparrow sate close by,
On a little piece of wood,
Singing with a cheerful smile
Spikky bikky bee!
To amuse them all the while,
Twikky wikky wikky wee,
Wikky bikky twikky tee,
```

Hint: use the sentinel to save all the input strings into a list, then sort it with a method.

3.3 Problem 2 (1.5 marks)

Problem: The *median* of a sequence of numbers is the value in the middle, that is, if the numbers are sorted, as many numbers in the sequence are above the median as below. If there are an odd number of values, the median is just the middle one. If there is an even number of values, the median is half way between the two values closest to the middle.

Write a program that allows the user to enter numbers until a blank line is entered instead of number, and then prints the median. For example:

```
Enter a number: 2
Enter a number: 3
Enter a number: 1
Enter a number:
Median = 2.0
```

```
Enter a number: 2
Enter a number: 4
Enter a number: 1
Enter a number: 3
Enter a number:
Median = 2.5
```

The Python standard library has a `statistics` module, with a `median` function. You are not permitted to use that module for this problem.

3.4 Problem 3 (1.5 marks)

Problem: A *palindrome* is a string that reads the same backwards as forwards. It is usual to remove all non-letters from the string first and to ignore case.

Write a *function* that accepts a string as its argument and returns `True` if and only if the string is a palindrome.

Your main program should allow the user to test input strings until they enter an empty one. For example:

```
Enter a string: abc
It is not a palindrome.
Enter a string: Abba
It is a palindrome!
Enter a string: canoe
It is not a palindrome.
Enter a string: Kayak
It is a palindrome!
Enter a string: Madam, I'm Adam.
It is a palindrome!
Enter a string: Was it a car or a cat I saw?
It is a palindrome!
Enter a string:
```

Hints: The function might use a for loop to build up a list of only the letters in upper case. A second for loop can check that the first letter equals the last, the second equals the second last, etc. Python makes this easy with its negative indices.

For your mild amusement.

4 After the workshop

- You have created programs that might be useful to refer back to in future workshops. Make sure that you will have that work in the future. One copy is not enough for at IT professional. You should have at least 2 copies:

  1. on your Griffith network storage drive; and
  2. on your portable storage device.