FIT1008 – Intro to Computer Science
Tutorial 1

Semester 1, 2019

Objectives of this tutorial

- For you to get to know each other and your tutor.
- To understand the organisation of this unit.
- To revise the material you covered in FIT1045 or equivalent.

Important

Before you do this tute, please read the document downloadable from the Moodle link "Lecture-Tute-Prac Expectations" that appears in Week 0. Make sure you understand what a tute is, what to expect from tutes, what to expect from tutors, and (importantly) what is expected from you.
Note that the purpose of tutorials is not to give you the answers to these questions. Solutions will be released on Moodle after all tutorials have finished, so if all you want is the answers, you can simply wait. **The purpose of tutorials is to make you think further** about the material covered during the lectures and, in particular, to **generate discussion about some of the more interesting and challenging concepts**.
If you have read the Unit Guide you might already know that **you need to prepare answers to the starred tute questions before** you come to the tute, and **bring the written answers** to the tute. These answers do not need to be correct (as you might still have questions about how to do them); they just need to show you put enough effort in preparing for the tutorial (about 1 to 1.5 hours per week) to be able to contribute meaningfully to the discussion. If you do not do this, you will negatively affect the learning of others and, therefore, from week 2 you will not be allowed in the tutorial.
Also note that tutes are often long and there will not be enough time to go through the entire tutorial, so make sure you tell your tutor if you are unsure about a particular question and would like to discuss it.

*Exercise 1 **

Consider the following pseudo code:

```
/* An algorithm to perform a MYSTERY action */

    count <- 0
    sum <- 0
    i <- 0

    while i < length(aList) {
        print "Outside"
        count <- count+1
        sum <- sum + aList[i]
        j <- 0

        while j < length(aList) {
            print "␣␣␣␣␣Inside"
            count <- count+1
            sum <- sum + aList[j]
            j <- j + 1
        }
        i <- i + 1
    }

    print "Sum:␣" + sum
    return count
```

Discuss the answers to the following questions:

(i) For a list of length 7, how many times does the `mystery` function print "`Outside`" and print "   `Inside`"?

7 times for “Outside” and 49 times for “   Inside”

(ii) What is the value returned in `count` for a list of length 7?

56

(iii) What value is printed for `sum` when the list contains numbers 1,2,3,4,5,6 and 7?

28 + (28 * 7) = 224

(iv) Discuss the answers to the above four questions for a list of length N.

“Outside” will print N times and “   Inside” will print N^2 times
count value will be N + N^2
sum value will be sum(N) + (sum(N) + N), where sum(N) is the sum of the elements of the list with length N

*Exercise 2 **

A string can be considered as a list of characters, and we could use the notation **w[k-1]** to denote the **k**th character in the string **w**.
A palindrome is a string which is spelt the same ways forwards as backwards. For example, **ab1ba** is a palindrome but **a3bbbaba** and **321236** are not.
Write a function which takes as a parameter a string and returns **True** if the string is a palindrome, and **False** otherwise. Assume all characters are lowercase alphanumerics and, thus, you do not need to deal with issues brought by matching uppercases to lowercases.

```
def palindrome(aString):
    for index in range(len(aString) // 2):
        if aString[index] != aString[len(aString) - 1 - index]:
            return False
    return True
```

*Exercise 3 **

Write a function which takes as parameters two lists, **list1** and **list2**, and prints out all the items that belong to both lists. For example, for lists $[2, 4, 8]$ and $[9, 3, 2]$, it should print 2. Assume no element appears twice in the list.

```
def identical(list1, list2):
  for i in range(len(alist1)):
    for k in range(len(alist2)):
      if alist1[i] == alist2[k]:
        print(alist[i])
```

Exercise 4

Answer the following questions:

- How does a heap differ from a binary search tree?
- How does a stack differ from a queue?
- How does recursion differ from iteration?

A heap is structured such that the root node is larger than the children (max-heap) or smaller than children (min-heap).
A binary search tree is structured in a way where the left node is always smaller than the parent node and right node is always larger than the parent node.

Both stack and queue are data structures.
Stack’s operation is based on Last-In-First-Out principle.
Queue’s operation is according to First-In-First-Out principle.

Both recursion and iteration repeatedly executes a set of instructions.
Recursion occurs when the function in a function calls itself recurrently, it doesn’t usually involve conditions.
Iteration occurs as a loop that executes continuously as long as the condition is true.