FIT1008 Introduction to Computer Science (FIT2085 for Engineers)

Tutorial 6

Semester 2, 2018

Objectives of this tutorial

- To understand how to work with ADTs.
- To understand Classes and Objects.
- To understand namespaces and scoping.

Exercise 1 *

This exercise and the next one are about scoping and namespaces. Examine these nested functions:

```python
def silly():
    x = 'ping'
    def g():
        print(x)
    x = 'pong'
    def f(x):
        print(x)
    g()
    f(x)

silly()
```

It will print:
pong
pong

When g() function is called, the print(x) searches for x in the local namespace of the g(), but when x is not initialized in the local namespace, it searches in the enclosing namespace (which is the local namespace of silly()) and prints the "ping".

When f(x) is called, the variable x = "pong" is passed as the argument into f(x) function and prints out the "pong" string.

Exercise 2 *

More scoping and namespaces. Now, with objects! A student named Juan is asked to make a Member class for a community pool. The only thing we care about is our pool's name, and our members' name, age and gender. So Juan writes:

```python
class PoolMember:
    poolname = "FIT1008␣students␣community␣pool"
    name = ""
    age = 0
    gender = None

    def __init__(self, name, age, gender):
        PoolMember.name = name
        PoolMember.age = age
        PoolMember.gender = gender
```

Juan makes himself the first member of the pool, and also its admin. It all seems to go so well!

```
>>> admin = PoolMember('Juan', 18, 'male')
>>> admin.name
'Juan'
>>> admin.age
18
>>> admin.gender
'male'
>>> admin.poolname
'FIT1008␣students␣community␣pool'
>>>
```

But an order comes from above. The pool has to be open to any Monash students and staff, not just FIT1008 students. And we need to change the name before other students come! So we tell Juan to change the pool name before we make the new membership card for a well-respected professor, Emilia, who will be our supervisor:

```
>>> admin.poolname = "MONASH␣all-inclusive␣community␣pool"
>>> admin.poolname
'MONASH␣all-inclusive␣community␣pool'                # it looks allright
>>> supervisor = PoolMember(Emilia, 26, 'female')    # let's make Emilia's user
```

Phew! Disaster averted! All seems alright again, until Emilia wants to check that the pool's name was really changed

```
>>> supervisor.poolname
'FIT1008␣community␣pool'          # what, this can't be!
>>> PoolMember.poolname           # let's check the global name
'FIT1008␣community␣pool'          # it's the same!
```

It's still the old name! Emilia wants to find who is responsible for the mistake, so she looks at the details for the admin:

```
>>> admin.name
'Emilia'
>>> admin.age
26
>> admin.gender
'female'
```

Juan did wrongly by changing the poolname for admin. What she did was changing the poolname for the admin and not changing the poolname for the class. This is because the poolname is associated as a class variable and not instance variable and hence if a permanent change is requested, she needs to change it via the class to access the class variable "poolname". It can be illustrated as "PoolMember.poolname = "MONASH all inclusive community pool".

- What? Is Juan framing Emilia for his mistakes? Or is it just an unfortunate series of bugs?
- What did Juan do wrong? Can you write the correct class and the correct way to change the pool's name?

Exercise 3 *

The function computes the sum of the elements in a_list2 from 0 to length of a_list2. If the element of a_list1[i] is larger than element of a_list2[i], then the function stops computing the elements and returns the latest sum (temp) result.

Consider the following function, which again uses the abstract `List` data type defined in week 5:

```
def mystery(a_list1, a_list2):
    temp = 0
    n1 = length(a_list1)
    n2 = length(a_list2)
    for i in range(n1):
        if i < n2:
            item2 = get_item(a_list2,i)
            temp +=  item2
            if get_item(a_list1,i) > item2:
                return temp
    return temp
```

For example:
a_list1 = [1,2,3,4,5,7,8,9]
a_list2 = [1,2,3,4,5,6,7,8]

The returned result will be 1+2+3+4+5+6 = 21, because the value of a_list1[5] > a_list[5] as 7 > 6, so it stops calculating the subsequent elements and immediately returns the latest sum, which is 21.

Worst-case is O(n1), it executes for n1 times for the for loop.
Best-case is O(1), it executes the inner return statement.

(a) Again without executing the code, indicate what the code does and illustrate with a few different examples.

(b) What is the best and worst Big O complexity of this function for two lists of strings? Explain.

Exercise 4 *

Given below is a snippet of a problem description:

A coffee store in a popular shopping centre is planning to introduce a customer loyalty system. This program is designed to offer rewards for their frequent customers, as well as make their ordering process easier. Each customer has an ID (assigned automatically by the system), name, phone number, and a count of loyalty points. The system will record points earned, accumulated by each customer when they order their coffee. These points can then be redeemed (used) for free coffee on subsequent visits. The system also manages up to 5 different coffees that the store makes. Each coffee has a name and price (in whole dollars). Each coffee is also half price on one day of the week. This day is different for each coffee, and is checked when an order is placed.

1. Describe the ADT's you would require for a program to implement this customer loyalty system.