ICS4U Unit 3 Activity 7 - Unit Test Part 1

Phone Pyae Thet Khine
Teacher NEDA
ICS4U
14th May 2022

Knowledge/Understanding/Communication

Q1) Define, using examples where appropriate, the following computer terms:

1. Unified Modelling Language
   Unified Modeling Language (UML) is a standardization of maodelling languages created to help visualize a system or a program with the use of disgrams. This language is not a language per se, but a set of rules guiding to draw the diagrams in the standardized way.

2) Inheritance
Inheritance, like in the real world, allows a class to inherit all the attributes of the parent class such as the class-methods, private variables and the blueprint. After the class has inherited the attributes from the parent class, it (the child class) can then override and overload the methods to suit their specific needs.

```
class LivingThing:  # Parent Class
    def eat(self):
        print("Eating")

    def sleep(self):
        print("Sleeping")

    def die(self):
        print("Dead")

class Human(LivingThing):  # Child Class
    def eat(self, food):  # Method Overloading
        print(f"Human eating {food}")

    def sleep(self):  # Method Overriding
        print("Human sleeping")

deer = LivingThing()
david = Human()

deer.eat()
```

```
deer.sleep()
deer.die()

david.eat("deer meat")
david.sleep()
david.die()
```

Output

```
Eating
Sleeping
Dead
Human eating deer meat
Human sleeping
Dead
```

Even when the function `die()` is not defined in the `Human` class, it can still be called since `Human` class inherits all the attributes from the parent class (`LivingThing`). Every `LivingThing` will die someday. Unless `Human` class overrides the `die()` method and calls the `immortal()` method.

3) Modules
Modules are componentes (small sections of a program) written separately from the main script, in some languages, exported and the imported to the main program using keywords like `import`. This helps compartmentalize the program into many small files which are easier to maintain and is much more organized than programming everything inside one big file such as the main script.
Example:

```
# in the file greeting.py
def greet(name):
        print(f"Hi, {name}")
```

```
# in the file main.py
import greeting
greeting.great("William") # calling a function inside another file.
```

Output

```
Hi, William
```

4. Object Oriented Programming
   Object Oriented Programming (OOP) consists of data structures and made easy for programmers to access and model data by defining objects and blueprints which can then be called to make a new instance of the object. OOP makes development easier and more secured by features such as encapsulation, only giving out the necessary data/values by Abstraction, sharing methods by Inheritance and changing inherited methods to suit the needs by Polymorphism.

Thinking/Inquiry

Q2) Compute a program to show use of classes or methods in program.

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

    def printProfile(self):
        print(self.name, self.age, self.gender)

william = Student("William", 19, "Male")
william.printProfile()
```

Output

```
William 19 Male
```

Classes makes it easier for programmers to access or add the date into a datavase with a single function. All the data modelling is done here.

Application

Q3) Demonstrate the evidence to design sub program in any programming language.

Sub-programs are programs inside a larger program disigned to solve something and can be reused many times. Unlike modules which needs to be imported, sub-programs can be written directly in the main script and be called to solve a number of logical problems and calcualations which strays aray from the main call stack but returns to it eventually with the problem which it is meant to be handle, solved.
Sub-program Example:

```
def subProgram():
    print(0.1)
    print(0.2)
    print(0.3)

def mainProgram():
    print(1)
    print(2)
    subProgram()
    print(4)

mainProgram()
```

Output

```
1
2
0.1
0.2
0.3
4
```

Communication

Q4) Clarify and explain the following terms with examples.

- Abstract Data Types
  Abstract Data Types are special kind of data type which is unknown to the user and the user cannot see how the data is logically calculated. The outputed data type is of the main primative data types such as `string`, `integer` and etc. but the inner logic is unknown. Example of ADTs are `Stack()`, `push()`, `pop()`.

- Stack
  Stack is an Abstract Data Type which functions like a list of items that needs to be done with the behavior of a stack. Like in the real life, the last item added is the first one to be processed (Last in First out). The Stack takes the item on the top most of the stack and the first item to be added is the last one to be processed.

```
stack = []

def stacking(item):
    stack.insert(0, item)
    print(stack)
```

```
stacking(1)
stacking(2)
stacking(3)
```

Output

```
[1]
[2, 1]
[3, 2, 1]
```

- Queue

```
queue = []

def queing(item):
    queue.append(item)
    print(queue)

queing(1)
queing(2)
queing(3)
```

Output

```
[1]
[1, 2]
[1, 2, 3]
```

- Dictionary

```
williamkhine = {"name": "William Khine", "age": 19}
print(williamkhine["age"])
```

Output

```
19
```