COMP2152 LAB MANUAL

OPEN SOURCE DEVELOPMENT

This booklet will help the reader understand the concepts, principles, and implementation of the Python programming language. By the end of the booklet, the reader will be able to code comfortably in Python.

Prepared by Ben Blanc

TABLE OF CONTENTS

CHAPTER 6

INHERITANCE

BASE CLASS

Base classes are what were created in Chapter 5. They take the following structure:

class CLASS_NAME :

//constructor

def __init__(self [, optional parms to pass to object])
statements

//methods

def NAME_OF_METHOD (self [,parameters]) :
statements

//accessors and mutators

@property
def accessor_property_name(self) :
(statements)

@accessor_property_name.setter
def accessor_property_name(self, newValueParam) :
(statements)

Let us create an Animal class in a file called **animalclass**:

```
class Animal :

    def __init__(self, Name='', Gender='', Weight=0, Height=0):

        self.name = Name
        self.gender = Gender
        self.__weight = Weight
        self.__height = Height
```

We will be using this base class to derive another class in the next section.

CHAPTER 6 AT A GLANCE

In this chapter you will learn how to define a class in relation to another class. You will learn about the terms base class and derived class. You will learn the importance of property and method permission levels and about constructors of derived classes.

DERIVED CLASS

A derived class is a class that is created from another class. The syntax of a derived class is the following:

class DERIVED_CLASS_NAME (BASE_CLASS_NAME) :

statements

We will add this class in our **animalclass** file

```
class Dog (Animal) :
    pass
```

We will create a **runner** file and construct objects of Animal and Dog.

```
animal_one = Animal()

animal_two = Animal("Fourth", "M", 150, 200)

dog = Dog()
```

CONSTRUCTORS OF DERIVED CLASSES

Constructors of derived classes inherit the constructors of the base class. You may override any method (including the constructor) in the derived class.

If you would like to override a constructor, you may do so in the derived class by adding the following code:

```
class Dog (Animal) :

    def __init__(self, Name='', Gender='', Weight=0, Height=0, Teeth=0):

        Animal.__init__(self, Name, Gender, Weight, Height)
        self.teeth = Teeth
```

In the above code, we have added an additional property to the Dog class. In the Dog class constructor, we call the Animal constructor class.

Here, the **self** keyword refers still refers to the current class, but includes all the public properties of the Animal and Dog class, as well as all of the methods of the Animal and Dog class.

The structure of overriding a constructor of the base class and calling the base class in the derived class is

def __init__(self [, optional parms to pass to object])
BASE_CLASS_NAME.__init__(self [, optional parms to pass to object])

Now we can create Animal and Dog objects in our **runner** file in the following ways:

```
from animalclass import Animal, Dog

animal_one = Animal()

animal_two = Animal("Fourth", "M", 150, 200)

animal_three = Animal("Sixth", "F")

animal_four = Animal("Tenth")

dog_one = Dog()

dog_two = Dog("Second", "F", 175, 100, 50)

dog_three = Dog("Seveth", "M", 90)

dog_four = Dog("Ninth")
```

METHOD SCOPES / OVERRIDING

Private or protected methods are not possible in python. This means that all methods are public and can be overridden in a derived class. Consider the following code in the Animal class

```
def GetName(self):
    return "The name of the "+__class__.__name__+ " is "+self.name

def IsFemale(self):
    if len(self.gender)==1 and ord(self.gender) == 70 or ord(self.gender) == 102 :
        return True
    return False
```

We can access the method GetName() or IsFemale() in our **runner** file for either the Animal or Dog class.

```
animal = Animal("AnyAnimal", "M", 120, 240)

dog = Dog("MyDog", 'F', 60, 180)

print("Animal GetName result is \n", animal.GetName())

print("Animal IsFemale result is \n", animal.IsFemale())

print("Dog GetName result is \n", dog.GetName())

print("Dog IsFemale result is \n", dog.IsFemale())
```

We can access the Animal method IsFemale() in the Dog class to create a new method in the Dog class

```
def CanBeMother(self):
    if self.IsFemale() :
        return self.name + " can be mother"
    return self.name + " cannot be a mother"
```

To access a base class method in the derived class method, we use the structure:

self.METHOD_NAME();

We can use this newly created Dog method in our **runner** file

```
first_dog = Dog("My First Dog", 'F', 60, 180)

second_dog = Dog("My Second Dog", 'M', 30, 90)

print(first_dog.CanBeMother())

print(second_dog.CanBeMother())
```

PROPERTY SCOPES

Public properties are accessible to derived classes. However, private properties are not accessible to derived classes.

When we try to access a private property in our derived class of Dog, we actually set a new private property that is only available to the Dog class and DO NOT the private property of the Animal class (the base class).

```
@property
def weight(self):
    return self.__weight

@weight.setter
def weight(self, Weight):
    self.__weight = Weight
```

When we go to our **runner** file and try to output the weight property, we get an error.

```
print(second_dog.weight)

    return self.__weight
AttributeError: 'Dog' object has no attribute '_Dog__weight'
```