Unit 4

Object-oriented Development using Python

(Seminar)

Seminar 2 – Python Programming Exercises

MODULE 9 CHALLENGE: CLASSES AND OBJECTS

1. In this challenge, you will create five (5) Person objects with appropriate values for first name, last name, weight, and height.
2. For this exercise, you can use the names p1, p2, p3, etc. for the names of your instantiated objects.
3. Once you have all five objects, create a list and store the objects in the list. Recall, an object is simply an instance of a user-defined type, so the syntax is the same as adding any other variable to a list.
4. Using a for loop, iterate over your list and print out the first names of each of your Person objects that you created. The output should look something like this:

```
class Person:
  def __init__(self, fname, lname, height, weight):
    self.firstname = fname
    self.lastname = lname
    self.height = height
```

```
    self.weight = weight


p1 = Person("A", "B", 1, 2)
p2 = Person("C", "D", 3, 4)
p3 = Person("E", "F", 5, 6)
p4 = Person("G", "H", 7, 8)
p5 = Person("I", "J", 9, 10)

persons = [p1, p2, p3, p4, p5]

for x in range(1, 5):
  print(persons[x-1].firstname)
```

MODULE 10 CHALLENGE: CLASSES AND METHODS

1. Add an instance method called print_self() that, when called, will output the instance object's first name, last name, weight, height, and BMI. Call that method and validate the output is correct for each object instance you create.
2. Create another method in your Person class that returns a value indicating if the person is underweight, good weight, or overweight, according to the CDC ranges. You already have the ranges available to you from previous labs. Call this method to output the result. HINT: You will need to make a method call inside a method call for this to function because we are not storing the BMI value in our Person class.

```
class Person:
    count = 0
```

```
    """Represents a generic Person."""

    def __init__(self, first, last, weight, height):
        self.first_name = first
        self.last_name = last
        self.weight_in_lbs = weight
        self.height_in_inches = height
        Person.count = Person.count + 1

    def calc_bmi(self):

        return (self.weight_in_lbs * 703) / self.height_in_inches ** 2

    @classmethod
    def print_count(cls,):
      return cls.count

    def print_self(self):
      print("%s, %s, %s, %s, %s" % (self.first_name, self.last_name, self.weight_in_lbs, self.height_in_inches, self.cal
c_bmi()))

    def is_underweight(self):
      bmi =self.calc_bmi()

      if bmi < 16:
        print("severely underweight")
      elif bmi >= 16 and bmi < 18.5:
         print("underweight")
      elif bmi >= 18.5 and bmi < 25:
         print("Healthy")
      elif bmi >= 25 and bmi < 30:
```

```
            print("overweight")
        elif bmi >=30:
            print("severely overweight")

p = Person('Tom', 'Thumb', 145, 64)
p.print_self()
p.is_underweight()
```

This challenge makes use of the “@classmethod” decorator to indicate a **static** method. To define static methods in Python, requires a parameter called “cls,” whereas an instance method requires a parameter called “self”.

MODULE 10 STRETCH CHALLENGE:

Stretch Challenge: This is optional but, see if you can work out how to create a static print_class() method that will output the class attributes, rather than the print_self() instance method that outputs values for each member attribute.

```
class Sample:
  @classmethod
  def print_class(cls,):
    print (cls.__dir__(cls))


p = Sample()
p.attr1 = "hello"
p.attr2 = 33333
print(dir(p))
```

To output the class attributes, I referenced the "__dir__()" class method through the static "cls" parameters. The "__dir__()" method attempts to retrieve a class's list of attributes and methods via any custom "__getattr__" method, failing which the method will retrieve data from the class's "__dict__" property.

MODULE 11 CHALLENGE: INHERITANCE

1. Create a new class called Teacher that inherits from Adult. Provide additional attributes around teacher type (such as kindergarten, secondary, or higher education) and BMI risk factor (kindergarten = 1, secondary = 2, and higher ed = 3). Assign BMI risk factors in the initializer based on the type of teacher that is supplied when creating the object.

```
class Person:
    count = 0

    """Represents a generic Person."""
    def __init__(self, first, last, weight, height, age=0, gender=''):
        self.first_name = first
        self.last_name = last
        self.weight_in_lbs = weight
        self.height_in_inches = height
        self.bmi = ''
        self.this_age = age
        self.this_gender = gender
        Person.count = Person.count + 1

    def calc_bmi(self):
        return (self.weight_in_lbs * 703) / self.height_in_inches ** 2
```

```
    @classmethod
    def print_count(cls,):
        return cls.count

class Adult(Person):
    def calc_bmi(self):
        bmi_tmp = (self.weight_in_lbs * 703) / self.height_in_inches ** 2

        print('BMI number is: ' + str(bmi_tmp))

        if bmi_tmp < 18:
            self.bmi = 'Underweight'
        elif bmi_tmp > 18 and bmi_tmp < 25:
            self.bmi = 'Normal'
        elif bmi_tmp > 25 and bmi_tmp < 30:
            self.bmi = 'Overweight'
        elif bmi_tmp > 30:
            self.bmi = 'Obese'
        return self.bmi

class Child(Person):

    def get_male_bmi(self, age, bmi_temp):
        bmi_class = ''
        if self.this_age > 2 and self.this_age < 9:
            if bmi_temp < 14:
                bmi_class = 'Underweight'
            elif bmi_temp > 14 and bmi_temp < 17:
                bmi_class = 'Normal'
            elif bmi_temp > 17 and bmi_temp < 20:
```

```
                bmi_class = 'Overweight'
            else:
                bmi_class = 'Obese'

        elif self.this_age > 9 and self.this_age < 16:
            if bmi_temp < 17:
                bmi_class = 'Underweight'
            elif bmi_temp > 17 and bmi_temp < 23:
                bmi_class = 'Normal'
            elif bmi_temp > 23 and bmi_temp < 25:
                bmi_class = 'Overweight'
            else:
                bmi_class = 'Obese'

        elif self.this_age >= 16:
            if bmi_temp < 19:
                bmi_class = 'Underweight'
            elif bmi_temp > 19 and bmi_temp < 23:
                bmi_class = 'Normal'
            elif bmi_temp > 23 and bmi_temp < 25:
                bmi_class = 'Overweight'
            else:
                bmi_class = 'Obese'

        return bmi_class

    def get_female_bmi(self, age, bmi_temp):
        return 'Not implemented'

    def calc_bmi(self):
        bmi_tmp = (self.weight_in_lbs * 703) / self.height_in_inches ** 2
```

```
        if self.this_gender == 'M':
            self.bmi = self.get_male_bmi(self.this_age, bmi_tmp)
        elif self.this_gender == 'F':
            self.bmi = self.get_female_bmi(self.this_age, bmi_tmp)
        return self.bmi


class Teacher(Adult):

  def __init__(self, first, last, weight, height, age=0, gender='', level='kindgergarten'):
    super().__init__(self, first, last, weight, height, age, gender)
    self.level = level
    self_risk_factor = self.calc_risk_factor(level)

  def calc_risk_factor(self):
    if self.level == "kindergarten":
      return 1
    if self.level == "secondary":
      return 2
    if self.level == "higher":
      return 3

    return 0
```

Python inheritance is written as <class_name>([parent_1, …]) where “parent_1” is an optional comma-separated list of classes to derive from.