Codes:

As an exercise, write a function called `distance_between_points` that takes two Points as arguments and returns the distance between them.

```
1  import math
2  def distance_between_points(p1,p2):
3      distance = math.sqrt(((p1[0]-p2[0])**2)+((p1[1]-p2[1])**2))
4      print(distance)
5
6  distance_between_points([3,4],[5,6])
7
8
```

As an exercise, write a function named `move_rectangle` that takes a Rectangle and two numbers named dx and dy. It should change the location of the rectangle by adding dx to the x coordinate of corner and adding dy to the y coordinate of corner.

```
class Point:
    """a blueprint for all point objects.
    It represent a point.
    It's attributes include: x, y.
    """

class Rectangle:
    """a blueprint for all rectangle objects.
    It represents a rectangle.
    It's attributes include: width, height, corner.
    """

# an object made of Rectangle class and it's attributes.
rectangle1 = Rectangle()
rectangle1.width = 7.0
rectangle1.height = 4.0
rectangle1.corner = Point()
rectangle1.corner.x = 0.0
rectangle1.corner.y = 0.0


def print_point(p):
    print(f"The x and y coordinates of the point are {p.x} and {p.y}.")


def move_rectangle(rect, dx, dy):
    # an object made of Point class and it's attributes.
    point1 = Point()
    point1.x = rect.corner.x + dx
    point1.y = rect.corner.y + dy
    return point1


movement = move_rectangle(rectangle1, 10, 12)
print_point(movement)

```

As an exercise, write a version of `move_rectangle` that creates and returns a new Rectangle instead of modifying the old one.

```
import copy

class Point:
    """a blueprint for point objects.
    It represents a point.
    It's attributes include: x, y"""


class Rectangle:
    """a blueprint for rectangle objects.
    It represents a rectangle.
    It's attributes includes: width, height, corner).
    """
rectangle1 = Rectangle()
rectangle1.width = 7.0
rectangle1.height = 4.0
rectangle1.corner = Point()
rectangle1.corner.x = 0.0
rectangle1.corner.y = 0.0


def print_point(p):
    print(f"The x and y coordinates of the point are {p.x} and {p.y}")


def move_rectangle(rect, dx, dy):
    new_rect = copy.deepcopy(rect)
    new_rect.x = rect.corner.x + dx
    new_rect.y = rect.corner.y + dy
    return new_rect


movement = move_rectangle(rectangle1, 10, 12)
print_point(movement)
```

Write a function called `draw_rect` that takes a Turtle object and a Rectangle and uses the Turtle to draw the Rectangle. See Chapter 4 for examples using Turtle objects.

Write a function called `draw_circle` that takes a Turtle and a Circle and draws the Circle.

```
import turtle
turtle1 = turtle.Turtle()

class Rectangle():
    "Represents a rectangle"

rectangle1 = Rectangle()

def draw_rect(ttle,rect):
    for i in range(2):
        ttle.forward(100)
        ttle.right(90)
        ttle.forward(60)
        ttle.right(90)

draw_rect(turtle1,rectangle1)

turtle.mainloop()
```

```
import turtle
turtle1 = turtle.Turtle()

class Circle():
    "Represents a circle"

circle1 = Circle()

def draw_circle(ttle,circle):
    ttle.circle(60)

draw_circle(turtle1,circle1)

turtle.mainloop()
```

As an exercise, write a function called `print_time` that takes a Time object and prints it in the form hour:minute:second. Hint: the format sequence `'%.2d'` prints an integer using at least two digits, including a leading zero if necessary.

```
class Time:
    """Represents a time.
    attributes: hour, minute, second"""

time1 = Time()
time1.hour = 11
time1.minute = 59
time1.second = 3

# %.2d shows the second in two digits: 03
def print_time(t):
    "Prints a representation of the time"
    print('%.2d:%.2d:%.2d' % (t.hour, t.minute, t.second))

print_time(time1)

```

Write a boolean function called `is_after` that takes two Time objects, t1 and t2, and returns True if t1 follows t2 chronologically and False otherwise. Challenge: don't use an if statement.

```
def time_to_int(t):
    "Converts time into integer"
    minutes = t.hour * 60 + t.minute
    seconds = minutes * 60 + t.second
    return seconds


class Time:
    """Represents the time
       Attributes: hour, minute, second"""

    def is_after(t1, t2):
        if time_to_int(t1) > time_to_int(t2):
            return True
        else:
            return False


time1 = Time()
time1.hour = 11
time1.minute = 59
time1.second = 30

time2 = Time()
time2.hour = 12
time2.minute = 34
time2.second = 20

print(time_to_int(time1))
print(time_to_int(time2))
print(Time.is_after(time1, time2))
```

This challenge will make some changes to the previous challenge. You will make use of your knowledge of classes, in addition to looping and lists. Lists can store objects so this challenge will require you to create a list of Person objects, that you will instantiate and add to the list.

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:

Tom
Fred
George
Tanya
Mary

5. Step 4 will require you to think about how you will access each member of the list and then how to access the correct attribute of that

```
class Person:
  def __init__(self,first,last,weight,height):
    self.first_name = first
    self.last_name = last
    self.weight = weight
    self.height = height

person1 = Person("Tom","Jones",150,70)
person2 = Person("Fred","Dawn",155,75)
person3 = Person("George","Mosley",160,80)
person4 = Person("Tanya","Derrik",165,85)
person5 = Person("Mary","Jaws",170,90)

person_list = [person1,person2,person3,person4,person5]
for i in person_list:
  print(i.first_name)

```

As an exercise, rewrite `time_to_int` (from Section 16.4) as a method. `

```
class Time:
    """Represents the time.
    Attributes: hour, minute, second"""

    def __init__(self, hour, minute, second):
        self.hour = hour
        self.minute = minute
        self.second = second

    def time_into_int(self):
        "Converts time into integer"
        minutes = self.hour * 60 + self.minute
        seconds = minutes * 60 + self.second
        return seconds

time1 = Time(12, 35, 45)
print(time1.time_into_int())

```

As an exercise, write a str method for the Point class. Create a Point object and print it.

```
class Point:
    """Represents a point.
    attributes: x, y"""

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        "Returns the string representation of the object"
        return '({0}, {1})'.format(self.x, self.y)


# creating an object from Point class
point1 = Point(4, 6)
print(point1)

```

As an exercise, write an add method for the Point class.

```
class Point:
    """Represents a point.
    attributes: x, y"""

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        "Returns the string representation of the object"
        return '({0}, {1})'.format(self.x, self.y)


    # operator overloading (__ __): let you use operators on a class and its components
    def __add__(self, other):
        x = self.x + other.x
        y = self.y + other.y
        return Point(x, y)


# creating an object from Point class
point1 = Point(4, 6)
point2 = Point(7, 8)

# prints based on __add__
print(point1 + point2)
```

Ok, we have given you some guidance and practice creating and using methods in your classes, both as instance methods and as static methods. In this challenge, you will add on to your Person class by building in some additional functionality.

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

  def calc_bmi(self):
    return (self.weight * 703) / (self.height ** 2)

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

  def print_self(self):
    return self.first, self.last, self.weigth, self.height, self.calc_bmi()
```

```
20    def print_weight_status(self):
21      if self.calc_bmi() < 18.5:
22        return f"{self.calc_bmi()} Underweight"
23      elif 18.5 < self.calc_bmi() < 24.9:
24        return f"{self.calc_bmi()} Healthy Weight"
25      elif 25.0 < self.calc_bmi() < 29.9:
26        return f"{self.calc_bmi()} Overweight"
27      else:
28        return f"{self.calc_bmi()} Obesity"
29
30  person = Person('Tom', 'Thumb', 150, 62)
31  person2 = Person('Fred', 'Flint', 225, 57)
32
33  print(person.calc_bmi())
34  print(person2.calc_bmi())
35  print(Person.print_count())
36  print(person.print_weight_status())
37  print(person2.print_weight_status())
```

Define the `Country` class which has attributes for `name`, `capital`, `population`, and `continent`. Please use the Python convention for making these attributes private.

Expected Output

Initialize an object of the `Country` class as follows:

```
my_country = Country('France', 'Paris', 67081000, 'Europe')
```

Task	Output
Print the name attribute	`France`
Print the capital attribute	`Paris`
Print the population attribute	`67081000`
Print the continent attribute	`Europe`

```
class Country:
  def __init__(self, name, capital, population, continent):
    self._name = name
    self._capital = capital
    self._population = population
    self._continent = continent

my_country = Country('France', 'Paris', 67081000, 'Europe')

print(my_country._name)
print(my_country._capital)
print(my_country._population)
print(my_country._continent)
```

Define the `Artist` class which has attributes for `name`, `medium`, `style`, and `famous_artwork`. Do **not** use the Python convention to make these attributes.

Expected Output

Initialize an object of the `Artist` class as follows:

```
my_artist = Artist('Bill Watterson', 'ink and paper', 'cartoons', 'Calvin a
```

Task	Output
Print `_Artist__name`	`Bill Watterson`
Print `_Artist__medium`	`ink and paper`
Print `_Artist__style`	`cartoons`
Print `_Artist__famous_artwork`	`Calvin and Hobbes`

```python
class Artist:
  def __init__(self, name, medium, style, famous_artwork):
    self.__name = name
    self.__medium = medium
    self.__sytle = style
    self.__famous_artwork = famous_artwork

  def helper1(self):
    return self.__name

  def helper2(self):
    return self.__medium

  def helper3(self):
    return self.__sytle

  def helper4(self):
    return self.__famous_artwork

my_artist = Artist('Bill Watterson', 'ink and paper', 'cartoons', 'Calvin and Hobbes')

print(my_artist.helper1())
print(my_artist.helper2())
print(my_artist.helper3())
print(my_artist.helper4())
```