G12_11_Term3-Assignment1

January 28, 2020

1 Computer Science G12/11

2 Term 3

3 Assignment 1

This is a review assignment on the basic of objects. While graded, the aim is for you to see where you stand. Approach it without losing sight of the tight scheduled for submitting it on time. This means, you should try to solve it on your own keeping a balance with an effective problem-solving-and-learning strategy: (1) getting as many done within the alloted time and (2) taking note of things that aren't clear yet! Effectiveness should be you main goal here.

There are two groups of questions. The first amounts to 85% of the mark. The second consists on three questions amounting to 15% of the mark of this assignment.

The first group must be fully answered -no questions left blank. Ask if you get stuck; discuss it with your peers, etc.

3.0.1 Due Date: Thu. Jan. 30

4 Required Questions (85%)

4.1 Select the result obtained if we'd run the following program

```
class Point:
    def __init__(self,x,y):
        self.x = x
        self.y = y

origin = Point(0,0)
origin.x = 12
print(origin.x)
```

1. 0
2. "0"
3. 12
4. There will be an error message

4.2 Select the result obtained if we'd run the following program

```
class Point:
    def __init__(self,xcoord,
    ycoord):
        self.x = xcoord
        self.y = ycoord
    def __str__(self):
        return "(" + str(self.xcoord)
        + " , " + str(self.ycoord) + ")"

origin = Point(0,0)
print(origin)
```

1. Point(0,0)
2. "(0,0)"
3. There will be an error message due to an error in **init**
4. There will be an error message due to an error in **str**

4.3 Select the result obtained if we'd run the following program

```
class Point:
    def __init__(self,xcoord,ycoord):
        self.x = xcoord
        self.y = ycoord
    def __str__(self):
        return "(" + str(self.x)
        + " , " + str(self.y) + ")"

origin = Point(0,0)
print(hasattr(origin,x))
```

1. 0
2. True
3. False
4. There will be an error message

4.4 Select all choices that result in the value of True after the following codes is run.

```
class Point:
    def __init__(self,xcoord,ycoord):
        self.x = xcoord
        self.y = ycoord
    def __str__(self):
        return "(" + str(self.x)
        + " , " + str(self.y) + ")"

origin = Point(0,0)
origin.x = 12
```

```
other = Point(12,0)
```

1. print(isinstance(other,point))
2. print(origin is other)
3. print(origin.x == other.x)
4. print(hasattr(other, "ycoord"))

4.5 Write the result obtained if we'd run the following program

```
class Point:
    def __init__(self,xcoord,ycoord):
        self.x = xcoord
        self.y = ycoord
    def __str__(self):
        return "(" + str(self.x)
        + " , " + str(self.y) + ")"

origin = Point(0,0)
new = origin
origin.y = 12
new.x = origin.y + 5
print(origin.x)
```

Your answer here:

4.6 Select all choices for the body of method `move_up` such that the answer when running the code is `(4,30)`.

```
class Point:
    def __init__(self,xcoord,ycoord):
        self.x = xcoord
        self.y = ycoord
    def __str__(self):
        return "(" + str(self.x)
        + " , " + str(self.y) + ")"
    def move_up(self, offset):

one = Point(0,0)
one.move_up(25)
print(one)
```

1. y += offset
2. self.y = offset
3. one.y = one.y + offset
4. self.y += offset

4.7 Select the result obtained when you run the program below

```
class Point:
    def __init__(self,xcoord,ycoord):
        self.x = xcoord
        self.y = ycoord
    def __str__(self):
        return "(" + str(self.x)
        + " , " + str(self.y) + ")"
    def move_up(self, offset):
        self.y = self.y + offset

one = Point(0,0)
two = one.move_up(20)
print(two.x)
```

1. 4
2. 10
3. 30
4. There will be an error message

4.8 Write the result obtained if we'd run this program

```
class Point:
    def __init__(self,xcoord,ycoord):
        self.x = xcoord
        self.y = ycoord
    def __str__(self):
        return "(" + str(self.x)
        + " , " + str(self.y) + ")"

    def between(first, second):
        return Point( (first.x + second.x)/2,
                        (first.y + second.y)/2)
one = Point(4,10)
two = Point(16,16)
print( between(one,two).x )
```

Answer:

4.9 Select all choices that are true concerning making the function `between` into a method in the program below.

```
class Point:
    def __init__(self,xcoord,ycoord):
        self.x = xcoord
        self.y = ycoord
    def __str__(self):
        return "(" + str(self.x)
        + " , " + str(self.y) + ")"
```

```
def between(first, second):
        return Point( (first.x + second.x)/2,
                        (first.y + second.y)/2)
```

1. The definition of `between` should be indented
2. Each use of `first` should be replaced by `self`
3. Each use of `x` should be replaced by `xcoord`
4. Each use of `y` should be replaced by `ycoord`

4.10 Suposing that the method `between` has been created as in the previous question, select the correct syntax for determining the value of the `x` attribute of the point between two points `one` and `two`

1. between(one,two).x
2. between(one.x,two.x)
3. one.between(two).x
4. one.between(two.x)

4.11 The Egg class can be used to create categories for eggs. For example, a "Large" egg should have mass in the range from 56 to 62 grams, so that 56 is the lowest mass and 62 is the highest mass possible for this category. Using the provided class definition and docstring, create methods `__init__` and `__str__`.

```
class Egg:
    """Egg low and high range of mass
       and category name.

       Attributes:
       low: int > 0, lowest mass in grams
       high: int > 0, highest mass in grams
       name: nonempty string, category
    """
```

4.12 Add to your methods for the class Egg by creating the method is_size, where egg_type.is_size(num) is True if an egg of mass num is of the category egg_type and False otherwise.

```
class Egg:
    """Egg low and high range of mass
       and category name.

       Attributes:
       low: int > 0, lowest mass in grams
       high: int > 0, highest mass in grams
       name: nonempty string, category
    """
```

4.13 Using the provided class definition and methods, create a new method aligned such that for objects `circ_1` and `circ_2`, `circ_1.aligned(circ_2)` is `True` if `circ_1` and `circ_2` share the same x and y coordinates and `False` otherwise.

```
class Circle:
    """Circle radius, x and y coordinates, colour

       Public methods:
       __init__: initializes a new object

       Attributes:
       radius: int or float >= 0; circle radius
       x: int >= 0; x-coordinate of centre
       y: int >= 0; y-coordinate of centre
       colour: string; colour of the circle
    """

    def __init__(self, radius, x, y, colour):
        """Creates new circle
        """

        self.radius = radius
        self.x = x
        self.y = y
        self.colour = colour

    def __str__(self):
        """Prints object
        """

        return self.colour + " circle of radius " + \
            str(self.radius) + " centred at (" + \
            str(self.x) + "," + str(self.y) + ")"

    def area(self):
        """Determines area.
        """
        return math.pi * self.radius ** 2
```

4.14 Now add a method `bigger` such that `circ_1.bigger(circ_2)` is `True` if `circ_1` is bigger than `circ_2` and `False` otherwise.

```
class Circle:
    """Circle radius, x and y coordinates, colour

       Public methods:
       __init__: initializes a new object
       Attributes:
       radius: int or float >= 0; circle radius
       x: int >= 0; x-coordinate of centre
       y: int >= 0; y-coordinate of centre
       colour: string; colour of the circle
    """

    def __init__(self, radius, x, y, colour):
        """Creates new circle
        """

        self.radius = radius
        self.x = x
        self.y = y
        self.colour = colour

    def __str__(self):
        """Prints object
        """

        return self.colour + " circle of radius " + \
            str(self.radius) + " centred at (" + \
            str(self.x) + "," + str(self.y) + ")"

    def area(self):
        """Determines area.
        """
        return math.pi * self.radius ** 2
```

4.15 Finally, add a method `is_colour` such that `circ_1.is_colour(col)` is `True` if the colour of `circ_1` is `col` and `False` otherwise.

```
class Circle:
    """Circle radius, x and y coordinates, colour

       Public methods:
       __init__: initializes a new object

       Attributes:
       radius: int or float >= 0; circle radius
       x: int >= 0; x-coordinate of centre
       y: int >= 0; y-coordinate of centre
       colour: string; colour of the circle
    """

    def __init__(self, radius, x, y, colour):
        """Creates new circle
        """
        self.radius = radius
        self.x = x
        self.y = y
        self.colour = colour

    def __str__(self):
        """Prints object
        """

        return self.colour + " circle of radius " + \
            str(self.radius) + " centred at (" + \
            str(self.x) + "," + str(self.y) + ")"

    def area(self):
        """Determines area.
        """
        return math.pi * self.radius ** 2
```

5 Extra problems (15%)

5.1 Using the Time class defined earlier, create a new method safe which determines whether the values of the attributes are integers in the correct ranges. If a type is wrong the method should produce “Incorrect type”, if the types are correct but the hour value is an integer out of range the method should produce “Hour out of range”, and if the types are correct and the hour value is in range but the minute value is an integer out of range the method should produce “Minute out of range”. If the values are correct, the method should produce “Safe”.

```
class Time:
    """Time stored as hour and minutes

       Methods:
       __init__: initializes a new object
       __str__: prints an object
       earlier_time: returns earlier time

       Attributes:
       hour: int, 0 <= value < 24
       minute: int, 0 <= value < 60
    """

    def __init__(self, hour, minute):
        """Initializes a new object.

           Preconditions:
           hour: int, 0 <= value < 24
           minute: int, 0 <= value < 60

           Parameters:
           hour: hour in time
           minute: minutes in time

           Side effect: attributes set with values
        """
        self.hour = hour
        self.minute = minute

    def __str__(self):
        """Prints time.

           Side effect: prints
        """

        if self.minute < 10:
            min_word = "0" + str(self.minute)
        else:
            min_word = str(self.minute)
        return str(self.hour) + ":" + \
            min_word

    def earlier_time(self, other):
        """Determines earlier of two Times.

           Preconditions:
           other: Time object

           Parameters:
           other: Time compared to self

           Returns: earlier of two times,
           or other if equal
        """

        if self.hour < other.hour:
            return self
        elif other.hour < self.hour:
            return other
        elif self.minute < other.minute:
            return self
        else:
            return other
```

5.2 Now add a method elapsed such that `one.elapsed(two)` returns the number of minutes that have passed between `one` and `two`, where `two` is a time later in the day.

5.3 Again using the same class, create a method overlap where `one.overlap(two, three)` returns `True` if (1) `one` is equal to either `two` or `three` or (2) `two` is earlier than `one` and `one` is earlier than `three`. You might wish to use the method `earlier_time` which has been provided for you. You might consider writing a helper function that determines if two times are equal and another helper function function that determines if one time is earlier than another.

```
class Time:
    """Time stored as hour and minutes

       Methods:
       __init__: initializes a new object
       __str__: prints an object
       earlier_time: returns earlier time

       Attributes:
       hour: int, 0 <= value < 24
       minute: int, 0 <= value < 60
    """

    def __init__(self, hour, minute):
        """Initializes a new object.

           Preconditions:
           hour: int, 0 <= value < 24
           minute: int, 0 <= value < 60

           Parameters:
           hour: hour in time
           minute: minutes in time

           Side effect: attributes set with values
        """
        self.hour = hour
        self.minute = minute

    def __str__(self):
        """Prints time.

           Side effect: prints
        """

        if self.minute < 10:
            min_word = "0" + str(self.minute)
        else:
            min_word = str(self.minute)
        return str(self.hour) + ":" + \
            min_word

    def earlier_time(self, other):
        """Determines earlier of two Times.

           Preconditions:
           other: Time object

           Parameters:
           other: Time compared to self

           Returns: earlier of two times,
           or other if equal
        """

        if self.hour < other.hour:
            return self
        elif other.hour < self.hour:
            return other
        elif self.minute < other.minute:
            return self
        else:
            return other
```