Object Oriented Programming - Assignment

1. Basic Class

1.1 Class `Point`

Create a Python class `Point` which contains 2 attributes, `x` and `y`, representing x and y coordinate of the point.

- Implement initializer method which initialize `x` and `y`.
- Implement a instance method `dist_to_origin()` which return distance from origin using formular `math.sqrt(x**2 + y**2)`.
- Implement `__str__()` which returns "(x,y)", e.g. "(3.0,4.0)"

Sample Output

```
(3.0, 4.0)
Point(3.0,4.0)
5.0
```

In [3]:
```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def dist_to_origin(self):
        import math
        return math.sqrt(self.x**2 + self.y**2)

    def __str__(self):
        return "({s.x:.1f},{s.y})".format(s = self)
#         return "({:.1f}, {:.1f})".format(self.x, self.y)

    def __repr__(self):
        return "Point({:.1f},{:.1f})".format(self.x, self.y)

### Test
p = Point(3, 4)
print(str(p))
print(repr(p))
print(p.dist_to_origin())
```

```
(3.0,4)
Point(3.0,4.0)
5.0
```

1.2 Class `Rectangle`

Create a Python class `Rectangle` which contains 3 attributes, `width`, `height` and `corner`. The `corner` is of `Point` type, which gives coordinate of bottom left corner of the rectangle.

- Implement initializer method which initialize `width`, `height` and `corner`.
- Implement a instance method `get_centre()` which returns a `Point` boject representing centre point of the rectance.
- Implement a instance method `scale(val)` which scale width and height by `val` times.

Sample Output

```
(7.0, 12.0)
Rectangle(20, 40) at point (2.0, 2.0)
```

In [5]:
```
class Rectangle(object):
    def __init__(self, width, height, corner):
        self.width = width
        self.height = height
        self.corner = corner

    def get_centre(self):
        x = self.corner.x + 1/2 * self.width
        y = self.corner.y + 1/2 * self.height
        return Point(x, y)

    def scale(self, val):
        self.width = self.width * val
        self.height = self.height * val

    def __str__(self):
        return 'Rectangle({}, {}) at point {}'.format(self.width, self.height, se


### Test
r = Rectangle(10, 20, Point(2, 2))
print(r.get_centre())
r.scale(2)
print(r)
```

```
(7.0,12.0)
Rectangle(20, 40) at point (2.0,2)
```

2. Class Attribute, Static Method and Class Method

2.1 Class Attribute

Implement a class attribute `counter` which keep track of number of instance created for class `Machine`.

- The `counter` has inital value of 0
- In `__init__(self)` method, increment counter by 1

- In `__del__(self)` method, decrement counter by 1

Sample Output

```
2
1
```

In [7]:
```
class Machine:
    counter = 0

    def __init__(self):
        Machine.counter += 1
        print("init instance {} from class {}. new counter = {}"
              .format(id(self), id(type(self)), self.counter))

    def __del__(self):
        Machine.counter -= 1
        print("del instance {} from class {}. new counter = {}"
              .format(id(self), id(type(self)), self.counter))

# Test
m1 = Machine()
m2 = Machine()
print(Machine.counter)
del m1
print(Machine.counter)
```

```
init instance 2885866736776 from class 2885860698424. new counter = 1
init instance 2885866736720 from class 2885860698424. new counter = 2
del instance 2885866735656 from class 2885860659720. new counter = 1
1
del instance 2885866736776 from class 2885860698424. new counter = 0
0
```

In [83]:
```
import math
print(id(math))

import math
print(id(math))

class A:
    pass
print(id(A))

class A:
    pass
print(id(A))

def B():
    pass
print(id(B))

def B():
    pass
print(id(B))
```

```
2344182350696
2344182350696
2344209307256
2344209330856
2344218088312
2344218088720
```

2.2 Class Method

Implement a class method `get_serial()` in the `Customer` class which returns __next_serial value and increase the __next_serial by 1.

```
class Customer:

    __next_serial = 1

    def __init__(self):
        self.serial = Customer.__next_serial
        Customer.__next_serial += 1
```

Sample Output

```
1
2
```

In [1]:
```
class Customer:

    __next_serial = 1

    @classmethod
    def get_serial(cls):
        x = cls.__next_serial
        cls.__next_serial += 1
        return x

    def __init__(self):
        self.serial = Customer.get_serial()

## Test
c1 = Customer()
c2 = Customer()
print(c1.serial)
print(c2.serial)
```

```
1
2
```

2.2 Static Method

Implement following 2 static method in the `Temperature` class which convert value between celsius and fahrenheit.

- `c2f()` which takes in a value in celsius and return a value in fahrenheit
- `f2c()` which takes in a value in fahrenheit and return a value in celsius

Sample Output

```
86.0
30.0
```

In [ ]:
```
class Temperature:

    @staticmethod
    def c2f(c):
        return (c * 9/5) + 32

    @staticmethod
    def f2c(f):
        return (f - 32) * 5/9

## Test
print(Temperature.c2f(30))
print(Temperature.f2c(86))
```

2.3 Multiple Initializer using Class Method (Optional)

Modify class `Book` to add a class method `from_json()` to create an instance from JSON string

```
class Book:

    def __init__(self, title, author):
        self.title = title
        self.author = author

    def __str__(self):
        return '({},{})'.format(self.title, self.author)

    ## define class method here


## Test
c1 = Book('Rich Dad Poor Dad', 'Robert Kiyosaki')
c2 = Book.from_json('{"title":"Rich Dad Poor Dad", "author":"Robert Kiyo
saki"}')
print(c1)
print(c2)
```

Sample Output

```
(Rich Dad Poor Dad,Robert Kiyosaki)
(Rich Dad Poor Dad,Robert Kiyosaki)
```

In [4]:
```
class Book:

    def __init__(self, title, author):
        self.title = title
        self.author = author

    def __str__(self):
        return '({},{})'.format(self.title, self.author)

    ## define class method here
    @classmethod
    def from_json(cls, input_json):
        import json
        rgb = json.loads(input_json)
        return cls(rgb['title'], rgb['author'])


## Test
c1 = Book('Rich Dad Poor Dad', 'Robert Kiyosaki')
c2 = Book.from_json('{"title":"Rich Dad Poor Dad", "author":"Robert Kiyosaki"}')
print(c1)
print(c2)
```

```
<__main__.Book object at 0x000002B7EF1654A8>
<__main__.Book object at 0x000002B7EF165518>
```

3. Properties

Implement a class `Person` which has 2 attributes, `firstName`, `lastName`.

- Use python convention to make both attributes "private"
- Implement an initializer which initialize both attributes
- Implement both attributes as properties
- Implement another property `fullname` which returns "firstName lastName"

Sample Output

```
Alan Goh
Alan Goh
```

In [ ]:
```python
class Person:
    def __init__(self, firstName, lastName):
        self._firstName = firstName
        self._lastName = lastName

    @property
    def firstName(self):
        return self._firstName

    @firstName.setter
    def firstName(self, value):
        self._firstName = value

    @property
    def lastName(self):
        return self._lastName

    @lastName.setter
    def lastName(self, value):
        self._lastName = value

    @property
    def fullname(self):
        return "{} {}".format(self._firstName, self._lastName)

## Test
p1 = Person("Alan", "Goh")
print(p1.firstName, p1.lastName)
print(p1.fullname)
```

4. Inheritance

- Construct a class `Shape` with 2 abstract property `area` and `perimeter`
- Construct a subclass `Rectangle` from class `Shape`. Its initializer takes in 2 attributes `width` and `height`. It implements both abstract properties.
- Construct a subclass `Circle` from class `Shape`. Its initializer takes in 1 attribute `radius`. It implements both abstract properties.

Sample Output

```
200 60
314.1592653589793 62.83185307179586
```

In [14]:
```
# from abc import ABCMeta, abstractmethod

# class Shape(metaclass=ABCMeta):

#     @property
#     @abstractmethod
#     def area(self):
#         pass

#     @property
#     @abstractmethod
#     def perimeter(self):
#         pass

class Shape:

    @property
    def area(self):
        raise NotImplementedError("Must implement get_area()")

    @property
    def perimeter(self):
        raise NotImplementedError("Must implement get_perimeter()")

    def __str__(self):
        return '{}: area={:.1f}, perimeter={:.1f}'.format(type(self).__name__,
                                                          self.area, self.perimeter)

class Rectangle(Shape):

    def __init__(self, width, height):
        self._width = width
        self._height = height

    @property
    def area(self):
        return self._width * self._height

    @property
    def perimeter(self):
        return 2 * (self._width + self._height)


class Circle(Shape):

    def __init__(self, radius):
        self._radius = radius

    @property
    def area(self):
        import math
        return self._radius * self._radius * math.pi

    @property
    def perimeter(self):
        import math
```

```
        return 2 * self._radius * math.pi

## Test
r = Rectangle(10, 20)
# print(r.area, r.perimeter)
c = Circle(10)
# print(c.area, c.perimeter)
print(r)
print(c)
```

```
Rectangle: area=200.0, perimeter=60.0
Circle: area=314.2, perimeter=62.8
```