Methods

Methods are functions defined inside the body of a class. They are used to perform operations with the attributes of our objects. Methods are a key concept of the OOP paradigm. They are essential to dividing responsibilities in programming, especially in large applications.

You can basically think of methods as functions acting on an Object that take the Object itself into account through its *self* argument.

Let's go through an example of creating a Circle class:

In [1]:

```
class Circle:
    #class object attribute pi
    pi = 3.14
    # Circle gets instantiated with a radius (default is 1)
    def __init__(self, radius=1):
        self.radius = radius
        self.area = Circle.pi * radius * radius

    # Method for resetting Radius
    def setRadius(self, new_radius):
        self.radius = new_radius
        self.area = self.pi * new_radius * new_radius

    # Method for getting Circumference
    def getCircumference(self):
        return 2 * self.pi * self.radius

c = Circle()

print('Radius is: ',c.radius)
print('Area is: ',c.area)
print('Circumference is: ',c.getCircumference())
```

```
Radius is:  1
Area is:  3.14
Circumference is:  6.28
```

In the __init__ method above, in order to calculate the area attribute, we had to call Circle.pi. This is because the object does not yet have its own .pi attribute, so we call the Class Object Attribute pi instead.

In the setRadius method, however, we'll be working with an existing Circle object that does have its own pi attribute. Here we can use either Circle.pi or self.pi.

Now let's change the radius and see how that affects our Circle object:

In [2]:

```
c.setRadius(2)

print('Radius is: ',c.radius)
print('Area is: ',c.area)
print('Circumference is: ',c.getCircumference())
```

```
Radius is:  2
Area is:  12.56
Circumference is:  12.56
```

Great! Notice how we used self. notation to reference attributes of the class within the method calls. Review how the above code works and try creating your own method.

In [ ]: