Table of Contents

1-Classes, Inheritance, Function Decoration, Overriding

Example:

Here we have 2 classes, `Robot` and `PhysicianRobot`. Each `Robot` has a `name` and a `health_level`, and it can `say_hi`. `PhysicianRobot` can also `heal` any `Robot`. The Parent class is `Robot` and the Subclass is `PhysicianRobot`.

```
import random
greeting_list = ["Hello", "SUP!", "Hey BRO!","Yo!","Hey there"]

class Robot:
    "Creating a Robot which has a name and a health level"
    "The Robot can say_hi and can determine if it needs_a_doctor"

    #pick a random greeting
    Z=greeting_list[random.randint(0,4)]

    def __init__(self, name):
        self.name = name
        #health_level is between 0 and 1
        self.health_level = random.random()

    def needs_a_doctor(self):
        if self.health_level < 0.8:
            return True
        else:
            return False

    #We want to add a parameter to the decorator to be capable of customizing the greeting
    #We have to wrap "greeting" function around the greeting_decorator
    def greeting(expr):
        def greeting_decorator(func):
            def function_wrapper(x):
                print(expr ,end="")
                func(x)
                print(" -- Current health is: %.2f" %x.health_level)
            return function_wrapper
        return greeting_decorator

    #passing a parameter "Z" to the decorator
    @greeting(Z)
    def say_hi(self):
        print(", I am " + self.name,end="")
```

```
class PhysicianRobot(Robot):

    "Creats a PhysicianRobot which can say_hi and heal a Robot"

    #overriding
    def say_hi(self):
        print("Everything will be okay! " + self.name + " takes care of you!")


    def heal(self, robo):
        #heal the Robot by generating a random number between the previous health level and 1
        robo.health_level = random.uniform(robo.health_level, 1)
        print(robo.name + " has been healed by " + self.name + "!")
```

If you look at the code of our `PhysicianRobot` class, you can see that we haven't defined any attributes or methods in this class. As the class `PhysicianRobot` is a subclass of `Robot`, it inherits, in this case, both the method `__init__` and `say_hi`. Inheriting these methods means that we can use them as if they were defined in the `PhysicianRobot` class. When we create an instance of `PhysicianRobot`, the `__init__` function will also create a name attribute.

```
#create an object of the class PhysicianRobot
doc = PhysicianRobot("Dr. Frankenstein")

rob_list = []

#creating 5 Robots and determining if they need healing or not
for i in range(5):
    x = Robot("Marvin" + str(i))
    if x.needs_a_doctor():
        print("Marvin" + str(i) + " NEEDS HEALING!")
        doc.say_hi()
        print("health_level of " + x.name + " before healing: %.2f" %x.health_level)
        doc.heal(x)
        print("health_level of " + x.name + " after healing:  %.2f" %x.health_level)
    else:
        print("Marvin" + str(i) + " doesn't need healing, health = %.2f" %x.health_level)
    rob_list.append(x)
    print()

#making the robots say hi
for j in rob_list:
    j.say_hi()
```

The output of the previous code is:

```
Marvin0 NEEDS HEALING!
Everything will be okay! Dr. Frankenstein takes care of you!
health_level of Marvin0 before healing: 0.29
Marvin0 has been healed by Dr. Frankenstein!
health_level of Marvin0 after healing:  0.49

Marvin1 NEEDS HEALING!
Everything will be okay! Dr. Frankenstein takes care of you!
health_level of Marvin1 before healing: 0.65
Marvin1 has been healed by Dr. Frankenstein!
health_level of Marvin1 after healing:  0.66

Marvin2 doesn't need healing, health = 0.99

Marvin3 NEEDS HEALING!
Everything will be okay! Dr. Frankenstein takes care of you!
health_level of Marvin3 before healing: 0.63
Marvin3 has been healed by Dr. Frankenstein!
health_level of Marvin3 after healing:  0.84

Marvin4 doesn't need healing, health = 0.89

Hey BRO!, I am Marvin0 -- Current health is: 0.49
Hey BRO!, I am Marvin1 -- Current health is: 0.66
Hey BRO!, I am Marvin2 -- Current health is: 0.99
Hey BRO!, I am Marvin3 -- Current health is: 0.84
Hey BRO!, I am Marvin4 -- Current health is: 0.89
```

Running it again would give a different output:

```
Marvin0 NEEDS HEALING!
Everything will be okay! Dr. Frankenstein takes care of you!
health_level of Marvin0 before healing: 0.34
Marvin0 has been healed by Dr. Frankenstein!
health_level of Marvin0 after healing:  0.89

Marvin1 NEEDS HEALING!
Everything will be okay! Dr. Frankenstein takes care of you!
health_level of Marvin1 before healing: 0.50
Marvin1 has been healed by Dr. Frankenstein!
health_level of Marvin1 after healing:  0.81

Marvin2 doesn't need healing, health = 0.90

Marvin3 doesn't need healing, health = 1.00

Marvin4 NEEDS HEALING!
Everything will be okay! Dr. Frankenstein takes care of you!
health_level of Marvin4 before healing: 0.03
Marvin4 has been healed by Dr. Frankenstein!
health_level of Marvin4 after healing:  0.16

Yo!, I am Marvin0 -- Current health is: 0.89
Yo!, I am Marvin1 -- Current health is: 0.81
Yo!, I am Marvin2 -- Current health is: 0.90
Yo!, I am Marvin3 -- Current health is: 1.00
Yo!, I am Marvin4 -- Current health is: 0.16
```

From the previous outputs we notice 2 things:

- Each time we run the code we'll get different results since it's mainly dependent on the random module.
- Robots can say different greeting as they pick them up randomly from the `greeting_list` and pass them as parameters to the decorator (`greeting`) of the `say_hi` method.

END OF SESSION ☺