OOP

PYTHON

RANJA SARKAR

January 2021

For starters

- **Encapsulation, Inheritance and Polymorphism** - *"It is advised to not get lost initially in these 3 big terms (jargons I would say)."*

- **Object and Class**  - *"Rather focus on these 2 and end up understanding the above."*

Object & Class

E.g., *while using an ML model, we execute*

>> model = RandomForestClassifier()

>> model.fit()

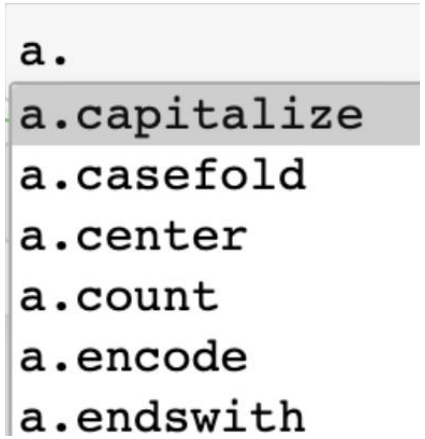
Here model is an object and fit is a method defined in the class RandomForestClassifier.

E.g., a = 2 where a is an object of class integer or 'int', also called 'variable' when NOT class-based

Why Class, when a Function seems easier to create?

E.g., a = "hello"

The 'str' (string) class defined for the object has a lot of functions (see below). That way a class becomes more powerful compared to functions.



This property of a class is called **Encapsulation.** *The class 'str' bundles the data ("hello") with all operate-able methods. Hence, encapsulation makes our code modular and easy to maintain.*

How to create a Class?

E.g., class is created to work with any account in a bank

```
class MyClass:
    def __init__(self, a , b):
        self.a = a
        self.b = b
```

```
class Account:
    def __init__(self, account_name, balance=0):
        self.account_name = account_name
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self,amount):
        if amount <= self.balance:
            self.balance -= amount
        else:
            print("Cannot Withdraw amounts as no funds!!!")
```

In general, __init__ is a method/function which runs whenever we create an object (see below).

The class 'Account' has 2 arguments 'account_name' and 'balance' (equivalent to 'a' and 'b' shown above).

E.g.

```
>> myAC = Account('Ranja', 100)
>> myAC.account_name, myAC.balance
Ranja 100
```

Inheritance: Basing an object/class upon another object/class retaining similar implementation

```
import math

class Shape:
    def __init__(self, name):
        self.name = name

    def area(self):
        pass

    def getName(self):
        return self.name

class Rectangle(Shape):
    def __init__(self, name, length, breadth):
        super().__init__(name)
        self.length = length
        self.breadth = breadth

    def area(self):
        return self.length*self.breadth

class Square(Rectangle):
    def __init__(self, name, side):
        super().__init__(name,side,side)
```

In the example above, multiple level of inheritance has been used in ‘Square’ which is derived from ‘Rectangle’. An object created from ‘Square’ or ‘Rectangle or ‘Shape’ is inherited.

Polymorphism: A property that makes a function do multiple things

```
import math

class Shape:
    def __init__(self, name):
        self.name = name

    def area(self):
        pass

    def getName(self):
        return self.name

class Rectangle(Shape):
    def __init__(self, name, length, breadth):
        super().__init__(name)
        self.length = length
        self.breadth = breadth

    def area(self):
        return self.length*self.breadth
```

In the example above, base Class is ‘Shape’ and derived class is ‘Rectangle’. An object created with these classes is polymorphic.

Summary

- **Encapsulation:** *Object contains all the data for itself*
- **Inheritance:** *Creation of a class hierarchy where methods from parent class passes on to child class*
- **Polymorphism:** *Function takes many forms, or object has multiple types*