



Class is a template/blue-print for real-world entities



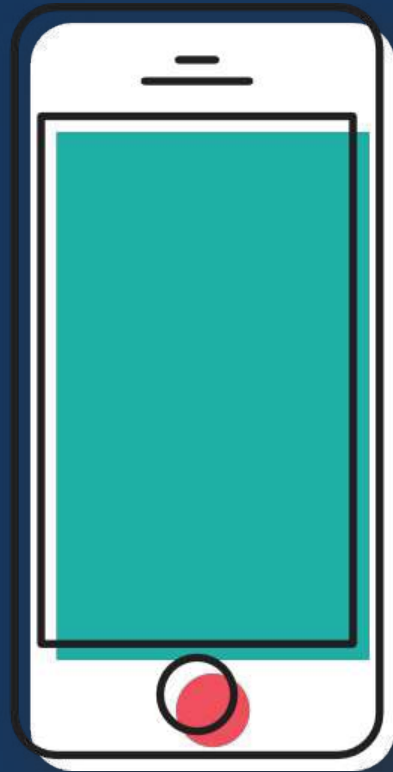
Properties

- Color
- Cost
- Battery Life

Behavior

- Make Calls
- Watch Videos
- Play Games

Class is a user-defined data-type



I am a user-defined data type

int

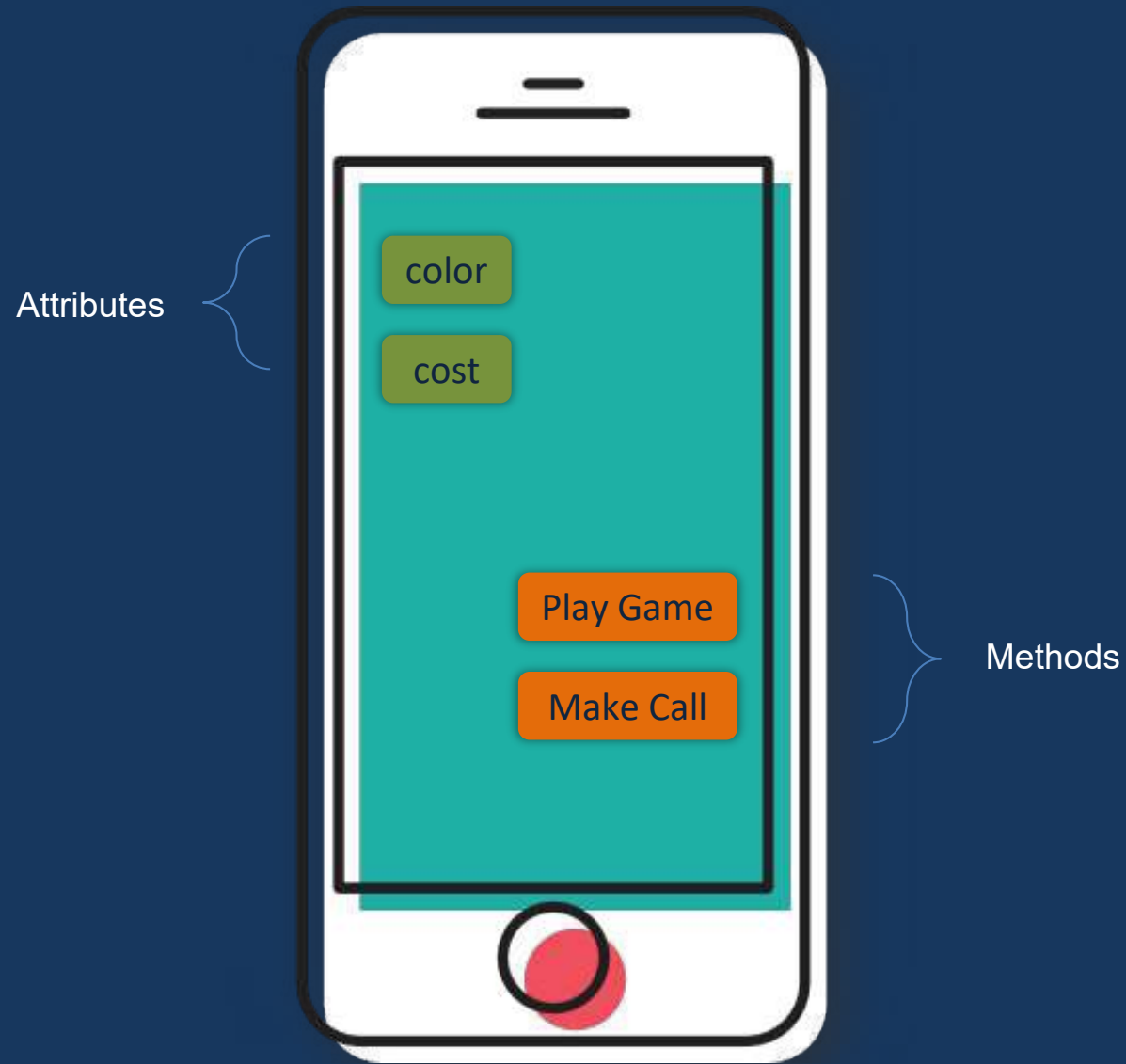
float

bool

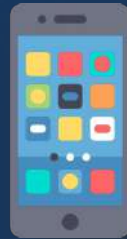
str

Mobile

Attributes and Methods



Objects are specific instances of a class



Apple



Motorola



Samsung

Specific instances of Mobile data type



Apple



Motorola



Samsung

`a = 10`

`b = 20`

`c = 30`

Specific instances of integer data type

Creating the first Class

```
In [1]: class Phone:

        def make_call(self):
            print("Making phone call")

        def play_game(self):
            print("Playing Game")
```

Creating the 'Phone' class

```
In [38]: p1=Phone()
```

Instantiating the 'p1' object

```
In [39]: p1.make_call()

        Making phone call
```

```
In [40]: p1.play_game()

        Playing Game
```

Invoking methods through object

Adding parameters to the class

```
[42]: class Phone:

    def set_color(self,color):
        self.color=color

    def set_cost(self,cost):
        self.cost=cost

    def show_color(self):
        return self.color

    def show_cost(self):
        return self.cost

    def make_call(self):
        print("Making phone call")

    def play_game(self):
        print("Playing Game")
```

Setting and Returning the
attribute values

Creating a class with Constructor

```
In [4]: class Employee:
        def __init__(self, name, age, salary, gender):

            self.name = name
            self.age = age
            self.salary = salary
            self.gender = gender

        def employee_details(self):
            print("Name of employee is ", self.name)
            print("Age of employee is ", self.age)
            print("Salary of employee is ", self.salary)
            print("Gender of employee is ", self.gender)
```

init method acts as the constructor

Instantiating Object

Instantiating the 'e1' object

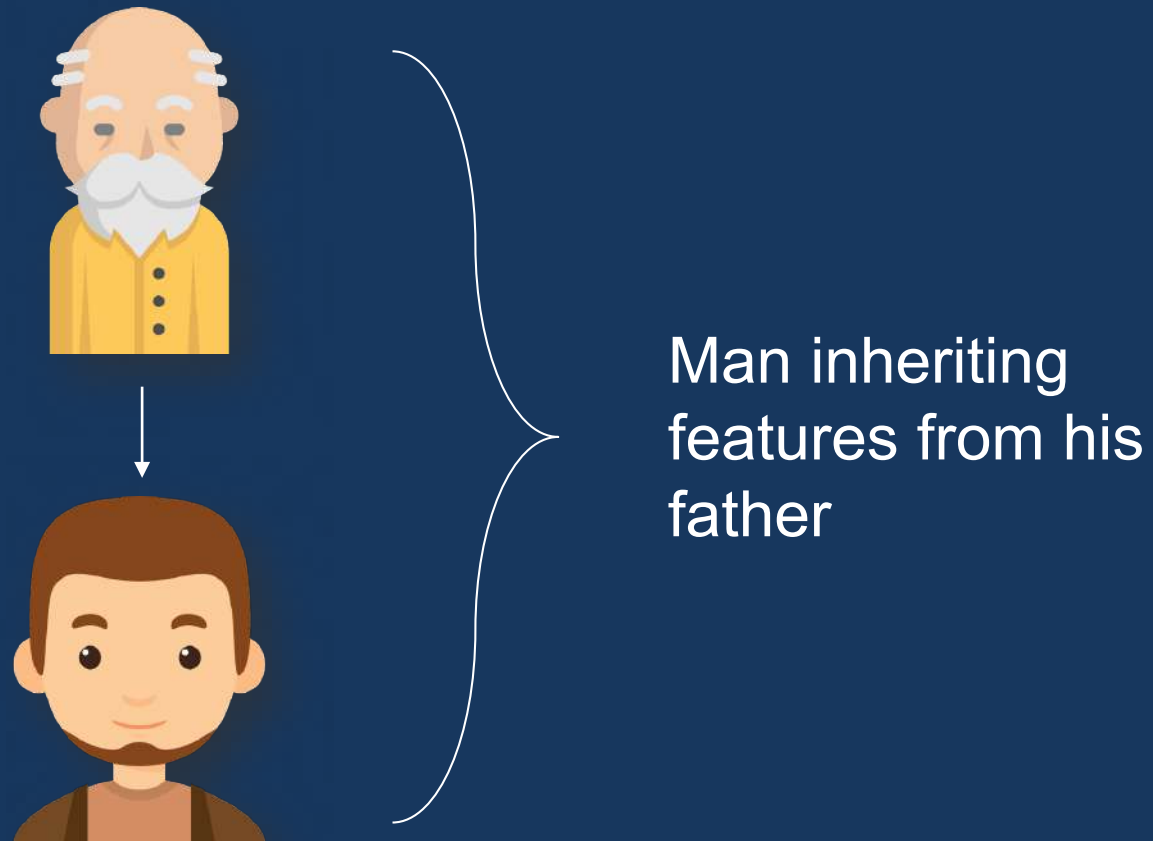
```
In [5]: e1 = Employee('Sam', 32, 85000, 'Male')
```

Invoking the
'employee_details'
method

```
In [6]: e1.employee_details()
```

```
Name of employee is Sam  
Age of employee is 32  
Salary of employee is 85000  
Gender of employee is Male
```

With inheritance one class can derive the properties of another class



Inheritance Example

```
In [23]: class Vehicle:

    def __init__(self, mileage, cost):
        self.mileage = mileage
        self.cost = cost

    def show_details(self):
        print("I am a Vehicle")
        print("Mileage of Vehicle is ", self.mileage)
        print("Cost of Vehicle is ", self.cost)
```

Creating the base class

```
In [24]: v1 = Vehicle(500, 500)
v1.show_details()
```

```
I am a Vehicle
Mileage of Vehicle is  500
Cost of Vehicle is  500
```

Instantiating the object for base class

Inheritance Example

```
In [25]: class Car(Vehicle):  
         def show_car(self):  
             print("I am a car")
```

→ Creating the child class

```
In [26]: c1 = Car(200,1200)
```

```
In [27]: c1.show_details()
```

```
I am a Vehicle  
Mileage of Vehicle is 200  
Cost of Vehicle is 1200
```

→ Instantiating the object for child class

```
In [28]: c1.show_car()
```

```
I am a car
```

→ Invoking the child class method

Over-riding init method

```
In [9]: class Car(Vehicle):  
  
    def __init__(self,mileage,cost,tyres,hp):  
        super().__init__(mileage,cost)  
        self.tyres = tyres  
        self.hp =hp  
  
    def show_car_details(self):  
        print("I am a car")  
        print("Number of tyres are ",self.tyres)  
        print("Value of horse power is ",self.hp)
```

→ Over-riding init method

Invoking show_details()
method from parent class

```
In [10]: c1 = Car(20,12000,4,300)
```

```
In [11]: c1.show_details()
```

```
I am a Vehicle  
Mileage of Vehicle is 20  
Cost of Vehicle is 12000
```

Invoking show_car_details()
method from child class

```
In [12]: c1.show_car_details()
```

```
I am a car  
Number of tyres are 4  
Value of horse power is 300
```

Types of Inheritance



These are the types
of inheritance in
Python....

Single Inheritance

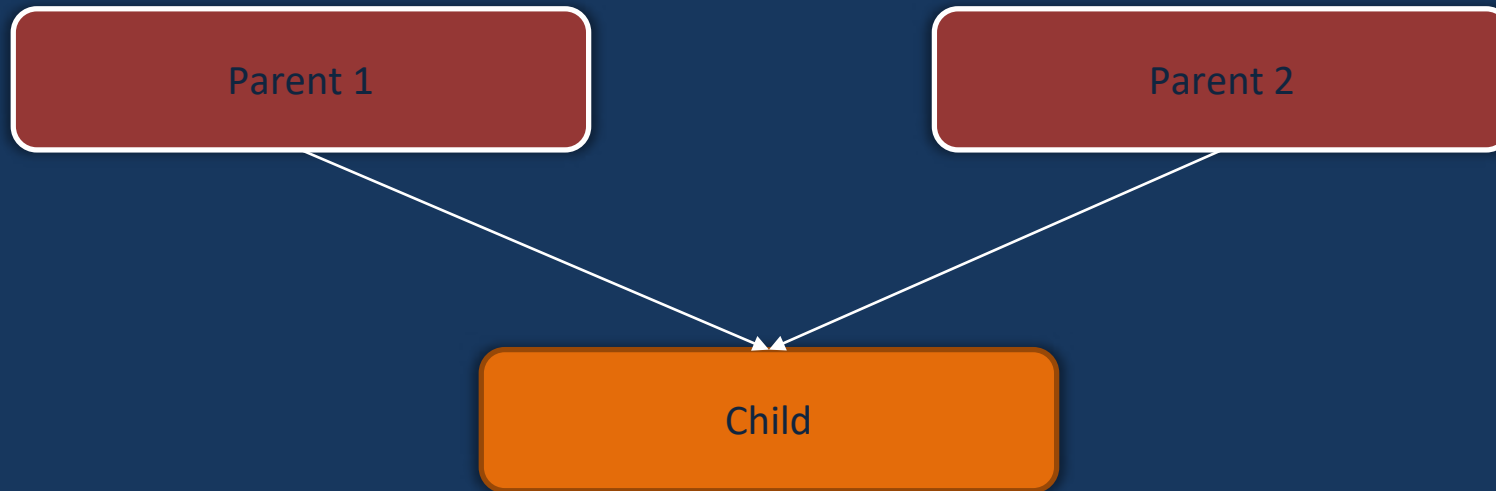
Multiple Inheritance

Multi-level Inheritance

Hybrid Inheritance

Multiple Inheritance

In multiple inheritance, the child inherits from more than 1 parent class



Multiple Inheritance in Python

Parent Class One

```
In [35]: class Parent1():
         def assign_string_one(self, str1):
             self.str1 = str1

         def show_string_one(self):
             return self.str1
```

Child Class

```
In [40]: class Derived(Parent1, Parent2):
         def assign_string_three(self, str3):
             self.str3 = str3

         def show_string_three(self):
             return self.str3
```

Parent Class Two

```
In [36]: class Parent2():
         def assign_string_two(self, str2):
             self.str2 = str2

         def show_string_two(self):
             return self.str2
```

Multiple Inheritance in Python

Instantiating object of child class

```
In [41]: d1 = Derived()
```

```
In [42]: d1.assign_string_one("one")  
d1.assign_string_two("two")  
d1.assign_string_three("three")
```

Invoking methods

```
In [46]: d1.show_string_one()
```

```
Out[46]: 'one'
```

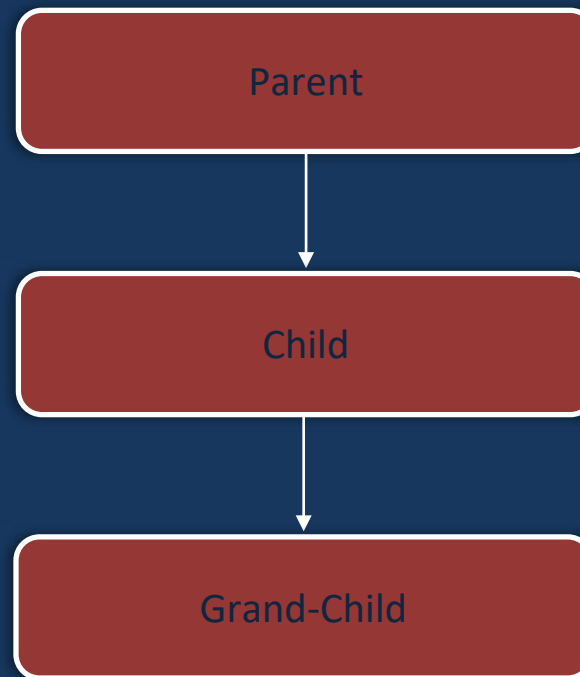
```
In [47]: d1.show_string_two()
```

```
Out[47]: 'two'
```

```
In [48]: d1.show_string_three()
```

```
Out[48]: 'three'
```

In multi-level Inheritance, we have Parent, child, grand-child relationship



Multi-Level Inheritance in Python

Parent Class

```
In [52]: class Parent():  
         def assign_name(self,name):  
             self.name = name  
  
         def show_name(self):  
             return self.name
```

Child Class

```
In [53]: class Child(Parent):  
         def assign_age(self,age):  
             self.age = age  
  
         def show_age(self):  
             return self.age
```

Grand-Child Class

```
In [54]: class GrandChild(Child):  
         def assign_gender(self,gender):  
             self.gender = gender  
  
         def show_gender(self):  
             return self.name
```

Instantiating object of GrandChild class

```
In [55]: g1 = GrandChild()  
  
In [56]: g1.assign_name("Sam")  
         g1.assign_age(25)  
         g1.assign_gender("Male")
```

Invoking class methods

```
In [57]: g1.show_name()  
Out[57]: 'Sam'  
  
In [58]: g1.show_age()  
Out[58]: 25  
  
In [59]: g1.show_gender()  
Out[59]: 'Sam'
```