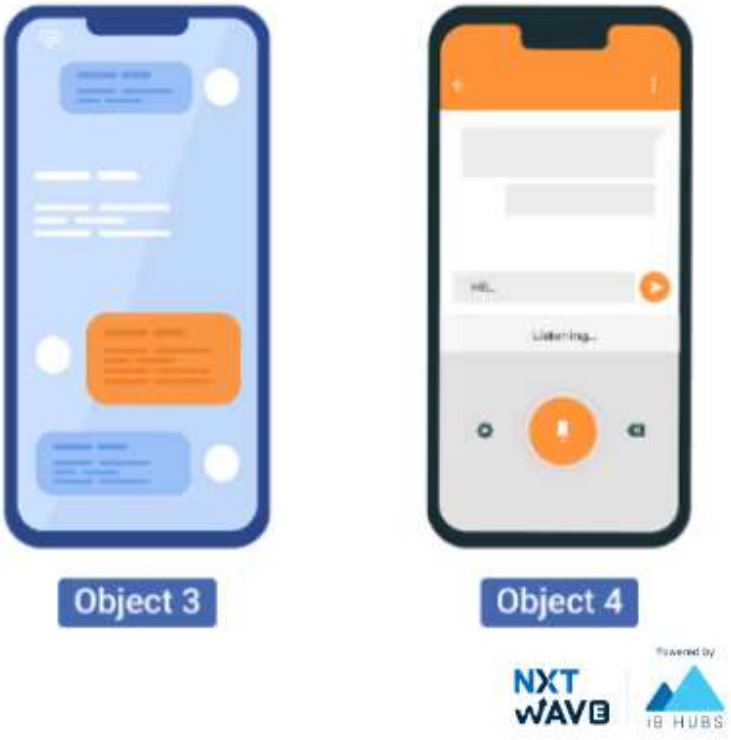
Cheat Sheet

Describing Similar Objects

Sometimes, the objects that we are describing are so similar that only values of the properties differ.




```
Object 3 is a Mobile
Properties
    camera : 13 MP
    storage : 16 GB
    battery life : 21 Hrs
    ram : 3 GB
    and so on ...

Object 4 is a Mobile
Properties
```

Expand

In this case, the objects that we describe have completely the same set of properties like camera, storage, etc.

Template

For objects that are very similar to each other (objects that have the same set of actions and properties), we can create a standard **Form** or **Template** that can be used to describe different objects in that category.

Mobile Template

```
Model :
Camera:
Storage:
Does it have a Face Unlock? Yes | No
```

Filled Template

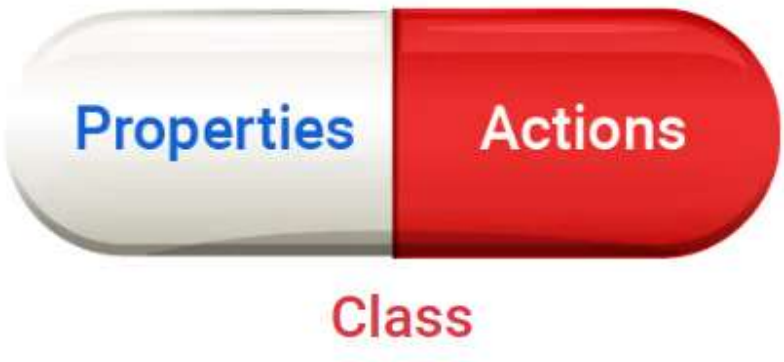
```
Model : iPhone 12 Pro
Camera: 64MP
Storage: 128GB
Does it have a Face Unlock? Yes
```

Bundling Data

While modeling real-life objects with object oriented programming, we ensure to bundle related information together to clearly separate information of different objects.

Bundling of related properties and actions together is called Encapsulation.

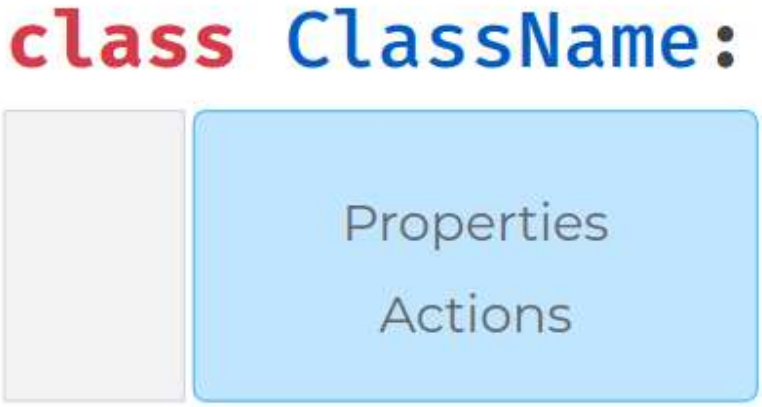
Classes can be used to bundle related properties and actions.



Defining a Class

To create a class, use the keyword

class



Special Method

In Python, a special method

__init__ is used to assign values to properties.

Code

PYTHON

```python
class Mobile:
    def __init__(self, model, camera):
        self.model = model
        self.camera = camera
```

Properties & Values

Code

PYTHON

```python
class Mobile:
    def __init__(self, model, camera):
        self.model = model
        self.camera = camera
    def make_call(self, number):
        print("calling..")
```

In the above example,

`model` and `camera` are the properties and values are which passed to the `__init__` method.

Action

PYTHON

```python
class Mobile:
    def __init__(self, model, camera):
        self.model = model
        self.camera = camera
    def make_call(self, number):
        print("calling..")
```

In the above example, the below function is an action

PYTHON

```python
def make_call(self, number):
    print("calling..")
```

In OOP terminology, properties are referred as **attributes** actions are referred as **methods**

Using a Class

To use the defined class, we have to instantiate it.
A class is like a blueprint, while its instance is based on that class with actual values.

Instance of Class

Syntax for creating an instance of class looks similar to function call.

An instance of class is **Object**.

Code

PYTHON

```python
class Mobile:
    def __init__(self, model, camera):
        self.model = model
        self.camera= camera

mobile_obj = Mobile(
    "iPhone 12 Pro",
    "12 MP")
print(mobile_obj)
```

Class Object

An object is simply a collection of attributes and methods that act on those data.

PYTHON

```python
class Mobile:
    def __init__(self, model, camera):
        self.model = model
        self.camera= camera

mobile_obj = Mobile(
    "iPhone 12 Pro",
    "12 MP")
print(mobile_obj)
```

Method Arguments & Return Values

Similar to functions, Methods also support positional, keyword & default arguments and also return values.

Code

PYTHON

```python
class Mobile:
    def __init__(self, model, camera):
        self.model = model
        self.camera= camera
    def make_call(self,number):
```

```
        return "calling..{}".format(number)
```

Instance Methods of Class

For instance method, we need to first write

`self` in the function definition and then the other arguments.

Code

PYTHON

```
class Mobile:
    def __init__(self, model, camera):
        self.model = model
        self.camera= camera
    def make_call(self,number):
        print("calling..{}".format(number))

mobile_obj = Mobile("iPhone 12 Pro", "12 MP")
mobile_obj.make_call(9876543210)
```

Output

```
calling..9876543210
```

Multiple Instances

Code

PYTHON

```
class Mobile:
    def __init__(self, model, camera):
        self.model = model
        self.camera= camera
    def make_call(self,number):
        print("calling..{}".format(number))

mobile_obj1 = Mobile("iPhone 12 Pro", "12 MP")
print(id(mobile_obj1))
```

```
10  mobile_obj2 = Mobile( Galaxy M51 ,  64 MP )
```

Expand

Output

```
139685004996560
139685004996368
```

Type of Object

The class from which object is created is considered to be the type of object.

Code

PYTHON

```
class Mobile:
    def __init__(self, model, camera):
        self.model = model
        self.camera = camera

obj_1 = Mobile("iPhone 12 Pro", "12 MP")
print(type(obj_1))
```

Output

```
<class '__main__.Mobile'>
```

Submit Feedback