Centre for Development of Advanced Computing (C-DAC)
Patna

Artificial Intelligence and Data Science using Python

Module 2: Basics of Python

Neeraj Kumar
Scientist ‘C’, C-DAC Patna
kneeraj@cdac.in

Prabhakar Mishra
Scientist ‘B’, C-DAC Patna
prabhakarm@cdac.in

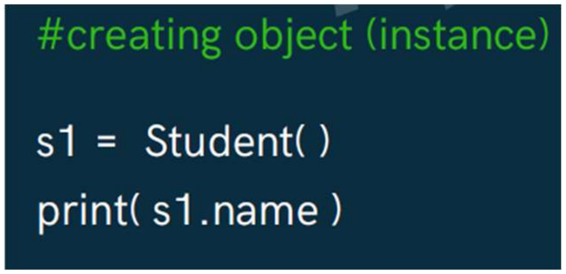
OOP in Python

- To map with real world scenarios, we started using objects in code.
- This is called object oriented programming.

Class & Object in Python

- Class is a blueprint for creating objects.
- #creating class

```
class Student:
    name = “karan kumar”
```



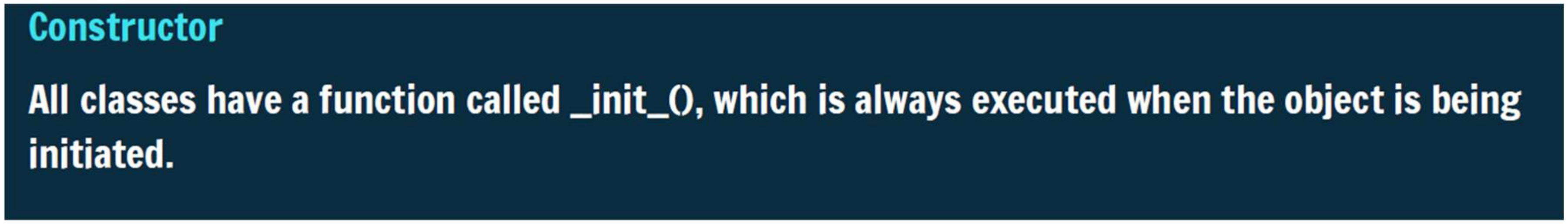
```
#creating object (instance)

s1 =  Student( )
print( s1.name )
```

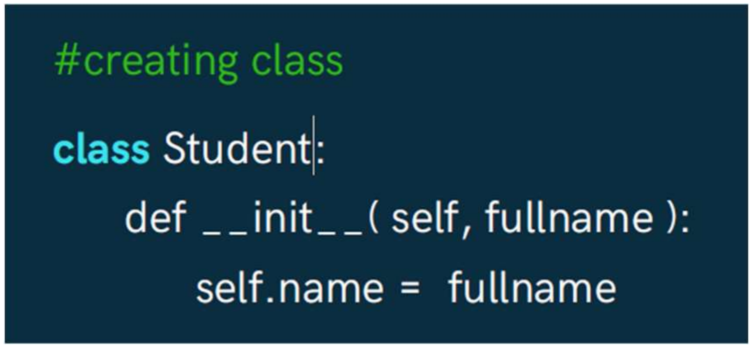
Class & Instance Attributes

- Class.attr – use dir() function
- obj.attr

_ _init_ _ Function


Constructor

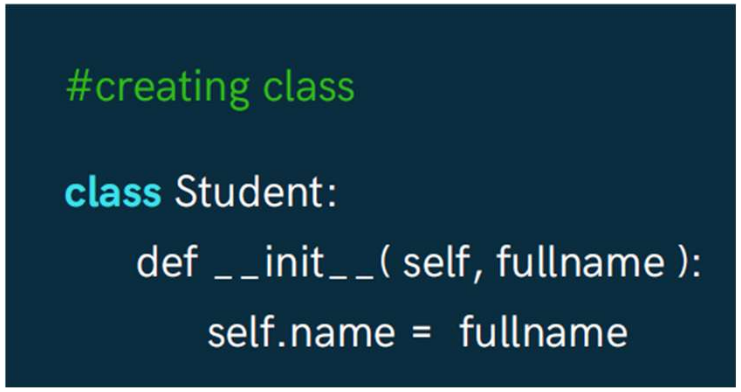
All classes have a function called _init_(), which is always executed when the object is being initiated.


```
#creating class

class Student:
    def __init__( self, fullname ):
        self.name =  fullname
```

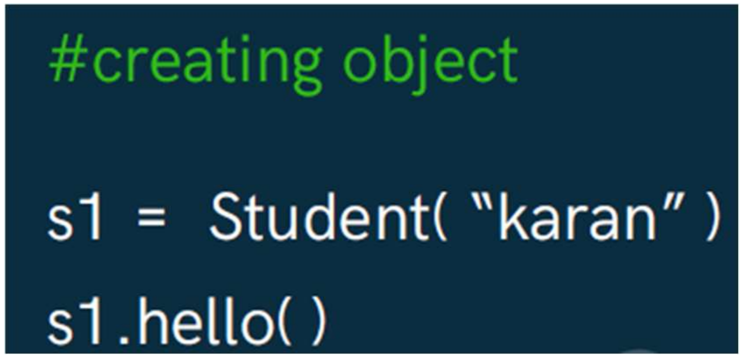
The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

Methods



```
#creating class

class Student:
    def __init__( self, fullname ):
        self.name =  fullname
```



```
#creating object

s1 =  Student( “karan” )
s1.hello( )
```

Let‘s Practice

- Create student class that takes name & marks of 3 subjects as arguments in constructor. Then create a method to print the average.

Important

- Abstraction

Hiding the implementation details of a class and only showing the essential features to the user.

- Encapsulation

Wrapping data and functions into a single unit (object).

Let‘s Practice

- Create Account class with 2 attributes - balance & account no.
- Create methods for debit, credit & printing the balance.

Inheritance

```python
class Animal:
    def __init__(self, name):
        self.name = name

    def info(self):
        print("Animal name:", self.name)

class Dog(Animal):
    def sound(self):
        print(self.name, "barks")

d = Dog("Buddy")
d.info()      # Inherited method
d.sound()
```

Initialise Parent Class

```
class Animal:
    def __init__(self, name):
        self.name = name

    def info(self):
        print("Animal name:", self.name)

# Child Class: Dog
class Dog(Animal):
    def __init__(self, name, breed):
        super().__init__(name)   # Call parent constructor
        self.breed = breed

    def details(self):
        print(self.name, "is a", self.breed)

d = Dog("Buddy", "Golden Retriever")
d.info()      # Parent method
d.details()   # Child method
```

Single Inheritance

```
class Person:
    def __init__(self, name):
        self.name = name

class Employee(Person):  # Employee inherits from Person
    def show_role(self):
        print(self.name, "is an employee")

emp = Employee("Sarah")
print("Name:", emp.name)
emp.show_role()
```

Multiple Inheritance

```python
class Person:
    def __init__(self, name):
        self.name = name

class Job:
    def __init__(self, salary):
        self.salary = salary

class Employee(Person, Job):  # Inherits from both Person and Job
    def __init__(self, name, salary):
        Person.__init__(self, name)
        Job.__init__(self, salary)

    def details(self):
        print(self.name, "earns", self.salary)

emp = Employee("Jennifer", 50000)
emp.details()
```

Multi-Level Inheritance

```python
class Person:
    def __init__(self, name):
        self.name = name

class Employee(Person):
    def show_role(self):
        print(self.name, "is an employee")

class Manager(Employee):  # Manager inherits from Employee
    def department(self, dept):
        print(self.name, "manages", dept, "department")

mgr = Manager("Joy")
mgr.show_role()
mgr.department("HR")
```

Hierarchical Inheritance

```python
class Person:
    def __init__(self, name):
        self.name = name

class Employee(Person):
    def role(self):
        print(self.name, "works as an employee")

class Intern(Person):
    def role(self):
        print(self.name, "is an intern")

emp = Employee("David")
emp.role()

intern = Intern("Eva")
intern.role()
```

Hybrid Inheritance

```python
class Person:
    def __init__(self, name):
        self.name = name

class Employee(Person):
    def role(self):
        print(self.name, "is an employee")

class Project:
    def __init__(self, project_name):
        self.project_name = project_name

class TeamLead(Employee, Project):  # Hybrid Inheritance
    def __init__(self, name, project_name):
        Employee.__init__(self, name)
        Project.__init__(self, project_name)

    def details(self):
        print(self.name, "leads project:", self.project_name)

lead = TeamLead("Sophia", "AI Development")
lead.role()
lead.details()
```

सीडैक
CDAC
पटना | PATNA

Thank You !