

# Understanding Classes and Objects

---

In the realm of data science, where information is abundant and diverse, mastering the basics of classes and objects in Python becomes crucial. These concepts serve as the building blocks for organizing and handling data efficiently. Classes and Objects empower data scientists to structure and manipulate data effectively.

## Python Classes

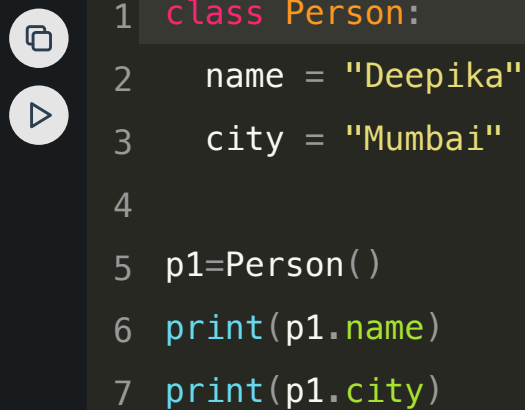
In Python, a class is like a blueprint or template for creating objects. It defines a structure that helps organize data and functions related to that data. Think of a class as a recipe that describes how to create something.

Creating a new class creates a new type of object, allowing new instances of that type to be made. Each class instance can have attributes attached to it for maintaining its state. Class instances can also have methods (defined by their class) for modifying their state.

Feature of Python class:

- Classes are created by keyword `class`.
- Attributes are the variables that belong to a class.
- Attributes are always public and can be accessed using the dot (`.`) operator. Eg.: `My class.Myattribute`

Python3



```
1 class Person:
2     name = "Deepika"
3     city = "Mumbai"
4
5 p1=Person()
6 print(p1.name)
7 print(p1.city)
```

## Object of Python Class

An Object is an instance of a Class. A class is like a blueprint while an instance is a copy of the class with actual values. It's not an idea anymore, it's an actual dog, like a dog of breed pug who's seven years old. You can have many dogs to create many different instances, but without the class as a guide, you would be lost, not knowing what information is required.

An object consists of

- State: It is represented by the attributes of an object. It also reflects the properties of an object.
- Behavior: It is represented by the methods of an object. It also reflects the response of an object to other objects.
- Identity: It gives a unique name to an object and enables one object to interact with other objects.

### `__init__()` method

The `__init__` method is similar to constructors in C++ and Java. Constructors are used to initializing the object's state. Like methods, a constructor also contains a collection of statements(i.e. instructions) that are executed at the time of Object creation. It runs as soon as an object of a class is instantiated. The method is useful to do any initialization you want to do with your object.

Python3



```
1 class Student:
2     def __init__(self, name, age):
3         self.name = name
4         self.age = age
```

## Modify Object Properties

You can modify properties on objects like this:

Python3



```
1 class Person:
2     name = "Deepika"
3     city = "Mumbai"
4
5 p1=Person()
6 p1.name = "Rose"
7 print(p1.name)
```

## Delete Object Properties

You can delete properties on objects by using the del keyword.

Delete the age property from the p1 object:

Python3



```
1 class Person:
2     name = "Deepika"
3     city = "Mumbai"
4     age = 20
5
6 p1=Person()
7 del p1.age
```

## Delete Objects

You can delete objects by using the del keyword.

Python3

```
1 class Person:
2     name = "Deepika"
3     city = "Mumbai"
4
5 p1=Person()
6 del p1
7 print(p1.name)
```

NameError: name 'p1' is not defined

## Encapsulation and Modularity

Classes promote encapsulation, which means bundling data and the methods that operate on that data together. This enhances modularity, making it easier to manage and maintain code, a crucial aspect in data science projects.

Python3

```
1 class Student:
2     def __init__(self, name, age):
3         self.name = name
4         self.age = age
5         self.grades = []
6
7     def add_grade(self, grade):
```

```
9
10     def average_grade(self):
11         return sum(self.grades) / len(self.g
```

SALE

Courses ▾

Tutorials ▾

Jobs ▾

Practice ▾

Contests ▾



Dash



Polymorphism: Adapting to Data Variability

All



Articles



Videos



Problems



Quiz

« Prev

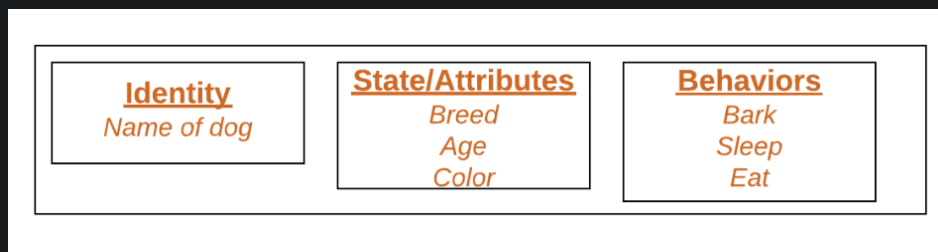
Next »

Polymorphism allows objects to be treated as instances of their parent class, enabling flexibility in handling various data types. This concept is particularly useful in scenarios where the same operation can be applied to different types of objects.

Python3

```
1 # Polymorphism in Action
2 def describe(animal):
3     animal.make_sound()
4
5 # Usage
6 dog = Dog("Woof")
7 cat = Cat("Whiskers")
8
9 describe(dog) # Output: Woof!
10 describe(cat) # Output: Meow!
```

Here, the describe function works with different types of animals, showcasing polymorphism.



## Relevance to Data Science: Structuring Data with Classes

In the context of data science, classes provide a powerful way to structure and organize data. For example, you might create a class to represent a dataset, with attributes for columns and methods for data analysis.

Python3



```
1 class DataSet:
2     def __init__(self, data):
3         self.data = data
4
5     def analyze(self):
6         # Perform data analysis here
7     pass
```

Marked as Read



Report An Issue

If you are facing any issue on this page. Please let us know.