

Classes and Objects

26 June 2025 10:21 PM

A class in Python is a user-defined template for creating objects. It bundles data and functions together, making it easier to manage and use them. When we create a new class, we define a new type of object. We can then create multiple instances of this object type.

Classes are created using **class keyword**. Attributes are variables defined inside the class and represent the properties of the class. Attributes can be accessed using the dot **operator** (e.g., MyClass.my_attribute).

Create a Class

```
Create a Class

1 class Dog:
2     Sound="Bark"
3 Dog().Sound

✓ 0.0s

'Bark'
```

```
1 class Dog:
2     Sound="Bark"
3 Dog.Sound

✓ 0.0s

'Bark'
```

```
Python
# define a class
class Dog:
    sound = "bark" # class attribute
```

Create Object

An Object is an instance of a Class. It represents a specific implementation of the class and holds its own data.

Now, let's create an object from **Dog class**.

```
Create Object

1 class Dog2:
2     Sound2='Bark'
3 # create an object
4 Dog3=Dog2()
5 print(Dog3.Sound2)

✓ 0.0s

Bark
```

```
Python
class Dog:
    sound = "bark"
# Create an object from the class
dog1 = Dog()

# Access the class attribute
print(dog1.sound)
```

sound attribute is a class attribute. It is shared across all instances of Dog class, so can be directly accessed through instance **dog1**.

Using `__init__()` Function

In Python, class has `__init__()` function. It automatically initializes object attributes when an object is created.

Python

```
class Dog:
    species = "Canine" # Class attribute
    def __init__(self, name, age):
        self.name = name # Instance attribute
        self.age = age # Instance attribute
```

Explanation:

- **class Dog:** Defines a class named Dog.
- **species:** A class attribute shared by all instances of the class.
- **__init__ method:** Initializes the name and age attributes when a new object is created.

Initiate Object with __init__



Python

```
class Dog:
    species = "Canine" # Class attribute
    def __init__(self, name, age):
        self.name = name # Instance attribute
        self.age = age # Instance attribute
```

Creating an object of the Dog class

```
dog1 = Dog("Buddy", 3)
```

```
print(dog1.name) # Output: Buddy
```

```
print(dog1.species) # Output: Canine
```

Output

Buddy

Canine

Explanation:

- **dog1 = Dog("Buddy", 3):** Creates an object of the Dog class with name as "Buddy" and age as 3.
- **dog1.name:** Accesses the instance attribute name of the dog1 object.
- **dog1.species:** Accesses the class attribute species of the dog1 object.

Self Parameter

[self](#) parameter is a reference to the current instance of the class. It allows us to access the attributes and methods of the object.

Self Parameter

```
1 class Dog7:
2     def __init__(self,name,age):
3         self.name=name
4         self.age=age
5     def Bark(self):
6         print(f"{self.name} is barking !")
7     # creating an instance of Dog
8     Dog8=Dog7('Buddy',3)
9     Dog8.Bark()
```

✓ 0.0s

Buddy is barking !

Python

```
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def bark(self):
        print(f"{self.name} is barking!")
```

Creating an instance of Dog

```
dog1 = Dog("Buddy", 3)
```

```
dog1.bark()
```

Output

Buddy is barking!

Explanation:

- Inside bark(), self.name accesses the specific dog's name and prints it.
- When we call dog1.bark(), Python automatically passes dog1 as self, allowing access to its attributes.

str Method

`__str__` method in Python allows us to define a custom string representation of an object. By default, when we print an object or convert it to a string using `str()`, Python uses the default implementation, which returns a string like `<__main__.ClassName object at 0x00000123>`.

str Method

```
1 class Dog9:
2     def __init__(self,name,age):
3         self.name=name
4         self.age=age
5     def __str__(self):
6         return f"{self.name} is {self.age} Years Old"
7     Dog10=Dog9("Buddy",3)
8     Dog11=Dog9('Charlie',5)
9     print(Dog10)
10    print(Dog11)
```

✓ 0.0s

Buddy is 3 Years Old

Charlie is 5 Years Old

Python

```
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def __str__(self):
        return f"{self.name} is {self.age} years old." # Correct: Returning a string
```

```
dog1 = Dog("Buddy", 3)
```

```
dog2 = Dog("Charlie", 5)
```

```
print(dog1)
```

```
print(dog2)
```

Output

Buddy is 3 years old.
Charlie is 5 years old.

Explanation:

- **__str__ Implementation:** Defined as a method in the Dog class. Uses the self parameter to access the instance's attributes (name and age).
- **Readable Output:** When print(dog1) is called, Python automatically uses the __str__ method to get a string representation of the object. Without __str__, calling print(dog1) would produce something like <__main__.Dog object at 0x00000123>.

Class and Instance Variables in Python

In Python, variables defined in a class can be either [class variables](#) or instance variables, and understanding the distinction between them is crucial for object-oriented programming.

Class Variables

These are the variables that are **shared across all instances of a class**. It is **defined at the class level**, outside any methods. **All objects of the class share the same value for a class variable unless explicitly overridden in an object.**

Instance Variables

Variables that are **unique to each instance (object) of a class**. These are defined within __init__ method or other instance methods. Each object maintains its own copy of instance variables, independent of other objects.

Example:

```
Class and Instance Variables in Python

1 class Dog12:
2     Species3="Canine"
3     def __init__(self,name,age):
4         self.name=name
5         self.age=age
6 Dog13=Dog12("Buddy",3)
7 Dog14=Dog12("Charlie",5)
8 print(Dog12.Species3)
9 print(Dog13.name)
10 print(Dog14.name)

✓ 0.0s

Canine
Buddy
Charlie
```

Instance Variables

```
1 class Dog:
2     # Class variable
3     species = "Canine"
4     def __init__(self, name, age):
5         # Instance variables
6         self.name = name
7         self.age = age
8 # Create objects
9 dog1 = Dog("Buddy", 3)
10 dog2 = Dog("Charlie", 5)
11 # Access class and instance variables
12 print(dog1.species) # (Class variable)
13 print(dog1.name)    # (Instance variable)
14 print(dog2.name)    # (Instance variable)
15 # Modify instance variables
16 dog1.name = "Max"
17 print(dog1.name)    # (Updated instance variable)
18 # Modify class variable
19 Dog.species = "Feline"
20 print(dog1.species) # (Updated class variable)
21 print(dog2.species)
22
```

✓ 0.0s

Canine
Buddy
Charlie
Max
Feline
Feline

Python

```
class Dog:
    # Class variable
    species = "Canine"
def __init__(self, name, age):
    # Instance variables
    self.name = name
    self.age = age

# Create objects
dog1 = Dog("Buddy", 3)
dog2 = Dog("Charlie", 5)

# Access class and instance variables
print(dog1.species) # (Class variable)
print(dog1.name)    # (Instance variable)
print(dog2.name)    # (Instance variable)

# Modify instance variables
dog1.name = "Max"
print(dog1.name)    # (Updated instance variable)

# Modify class variable
Dog.species = "Feline"
print(dog1.species) # (Updated class variable)
print(dog2.species)
```

Output

Canine
Buddy
Charlie
Max
Feline
Feline

Explanation:

- **Class Variable (species):** Shared by all instances of the class. Changing `Dog.species` affects all objects, as it's a property of the class itself.
- **Instance Variables (name, age):** Defined in the `__init__` method. Unique to each instance (e.g., `dog1.name` and `dog2.name` are different).
- **Accessing Variables:** Class variables can be accessed via the class name (`Dog.species`) or an object (`dog1.species`). Instance variables are accessed via the object (`dog1.name`).
- **Updating Variables:** Changing `Dog.species` affects all instances. Changing `dog1.name` only affects `dog1` and does not impact `dog2`.

Completed

From <<https://www.geeksforgeeks.org/python/python-classes-and-objects/>>