migo_9_classes

July 16, 2024

class

A class in Python is a blueprint for creating objects (a particular data structure). It defines a set of attributes and methods that the created objects can use.

define

[27]:
```
class MyClass:
    pass
```

Creating an Object

[28]:
```
obj = MyClass()
print(obj)
obj.var=7
print(obj.var)
```

```
<__main__.MyClass object at 0x0000025D2F873260>
7
```

[ ]:
```
Attributes and Methods
```

[ ]:
```
Attributes: Variables that belong to an object or class.
Methods: Functions that belong to an object or class.
```

The **init** Method

a special method called a constructor. It is called when an instance of the class is created. It's used to initialize the object's attributes.

[8]:
```
class MyClass:
    def __init__(self, attribute1, attribute2):
        self.attribute1 = attribute1
        self.attribute2 = attribute2

obj = MyClass("value1", "value2")
print(obj.attribute1,obj.attribute2,sep=",")
print(obj.__dict__)
```

```
value1,value2
{'attribute1': 'value1', 'attribute2': 'value2'}
```

self: represents the instance of the class and allows access to its attributes and methods.

Accessing Instance Attributes:

Accessing Other Methods

[ ]:
```
Distinguishing Instance Variables from Local Variables
```

[9]:
```
class MyClass:
    def __init__(self, value):
        self.value = value   # 'self.value' is an instance attribute
                             # 'self.value' refers to the instance variable,␣
 ↪'value' refers to the parameter
    def display_value(self):
        print(self.value)    # Accessing the instance attribute using 'self'

    def method1(self):
        print("Method 1")

    def method2(self):
        self.method1()      # Calling method1 using 'self'

obj = MyClass(10)
obj.display_value()
obj.method2()
```

```
10
Method 1
```

Encapsulation

Encapsulation restricts access to certain components. In Python, this is done by prefixing the attribute name with an underscore (single or double).

In fact, all class variables and methods are public.

[18]:
```
class MyClass:
    def __init__(self):
        self.public_attribute="I am public"
        self._protected_attribute = "I am protected"
        self.__private_attribute = "I am private"

    def get_protected_attribute(self):
        return self._protected_attribute

    def get_private_attribute(self):
        return self.__private_attribute

obj = MyClass()
#Naming Convention
```

```python
print(obj.public_attribute)
print(obj.get_protected_attribute())
print(obj.get_private_attribute())
#in python we can access them
print(obj._protected_attribute)
print(obj._MyClass__private_attribute)
```

```
I am public
I am protected
I am private
I am protected
I am private
```

Polymorphism:

Allows methods with the same name to behave differently based on the object calling the method.

achieved through method overriding (inheritance) or duck typing.

[ ]:
```
Duck Typing: Any object that implements the required method can be used␣
  ↪interchangeably.
```

[20]:
```python
class Dog:
    def speak(self):
        return "Woof!"

class Cat:
    def speak(self):
        return "Meow!"

class Duck:
    def speak(self):
        return "Quack!"

animals = [Dog(), Cat(), Duck()]

for animal in animals:
    print(animal.speak())
```

```
Woof!
Meow!
Quack!
```

Method Overriding: with inheritance, where a base class defines a method and derived classes override it.

[22]:
```python
class Animal:
    def speak(self):
        raise NotImplementedError("Subclass must implement abstract method")
```

```
class Dog(Animal):
    def speak(self):
        return "Woof!"

class Cat(Animal):
    def speak(self):
        return "Meow!"

animals = [Dog(), Cat()]

for animal in animals:
    print(animal.speak())
```

```
Woof!
Meow!
```

class variables and class methods

[ ]:
```
Class Variables:
```

Shared among all instances. Defined within the class but outside any method.

[5]:
```
class MyClass:
    class_variable = "I am a class variable"

    def __init__(self, instance_variable):
        self.instance_variable = instance_variable

print(MyClass.class_variable)

obj1 = MyClass("Instance 1")
obj2 = MyClass("Instance 2")

# accessing using an instance
print(obj1.class_variable)
print(obj2.class_variable)

# Modifying class variable using the class name
MyClass.class_variable = "Class variable modified"
print(obj1.class_variable)
obj1.class_variable="This instance's var with the same name of the class␣
 ↪vriable "
print(obj1.class_variable)
print(MyClass.class_variable)
```

```
I am a class variable
I am a class variable
I am a class variable
Class variable modified
```

```
My var with the same name of the class vriable
Class variable modified
```

Class Methods

Can modify class state that applies across all instances. Defined using @classmethod and take cls as the first parameter.

[8]:
```
class MyClass:
    class_variable = "I am a class variable"

    @classmethod
    def class_method(cls):
        return cls.class_variable

print(MyClass.class_method())
```

```
I am a class variable
```

Static Methods

are methods that do not access or modify class or instance state.

Do not access or modify class or instance state. Defined using @staticmethod and do not take self or cls as the first parameter.

[7]:
```
class MyClass:
    @staticmethod
    def static_method():
        return "I am a static method"

print(MyClass.static_method())
```

```
I am a static method
```

[ ]:
```
```