Class & Objects

Wednesday, March 6, 2024 9:47 AM

- Class definition -

```
class MyClass:
  x=10
print(MyClass.x)
p1=MyClass()
print(p1.x)
```

- Constructors: __init__()
- __init__() function:
  - All classes have a function called __init__(), which is always executed when the class is being initiated.
  - The __init__() function is called automatically every time the class is being used to create a new object.
- The __str__() Function:
  - The __str__() function controls what should be returned when the class object is represented as a string.
  - If the __str__() function is not set, the string representation of the object is returned
- The self Parameter:
  - The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.
  - It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class:
- The pass Statement:
  - class definitions cannot be empty, but if you for some reason have a class definition with no content, put in the pass statement to avoid getting an error.
  - class Person:
    pass
- Python Inheritance: Parent class also called base class, Child class also called derived class.
  - Syntax for inheritance:
    - Class Child(Parent):
      <body of the child class>
- When you add the __init__() function, the child class will no longer inherit the parent's __init__() function.
  - The child's __init__() function overrides the inheritance of the parent's __init__() function
  - To keep the inheritance of the parent's __init__() function, add a call to the parent's __init__() function
- Use the super() Function:
  - Python also has a super() function that will make the child class inherit all the methods and properties from its parent
- If you add a method in the child class with the same name as a function in the parent class, the inheritance of the parent method will be overridden.
- **self** : Self is a convention and not a Python keyword. Self is a parameter in Instance Method and the user can use another parameter name in place of it. But it is advisable to use self because it increases the readability of code, and it is also a good programming practice.
- Destructor is a special method which will be invoked automatically when the object gets removed from the memory.
  - class Product:

```
  def __init__(self,price,brand):
    self.price=price
    self.brand=brand
  def __del__(self):
    print('Deleting the object')
p1=Product(10000,'Apple')
p2=Product(7000,'Samsung')
del p1
```

Function	Method
Is a block of code with a name	Is part of an object and represents the behavior of the object
It can be invoked using the name of the function and passing parameters	Can be invoked only on an object, using dot operator. Without an object we cannot invoke a method
Example: len([1,2,3])	Example: [1,2,3].reverse()
Parameters are optional in a function	A method must have at least one parameter : self

- Method invocation: Inbound & Outbound

```
class Mobile:
  def __init__(self):
    print("Inside constructor")
  def purchase (self):
    print("Purchasing a mobile")
mob1=Mobile()
mob1.purchase() #Inbound method invocation, We need not pass the value for self.
Mobile.purchase(mob1) #Outbound method invocation, We have to pass the object as the value of self.
```

Variable Define and initialize in	Accessible outside class	Accessible inside class	Accessible inside method	Accessible inside another class	Syntax	Example
Outside Class	Yes	Yes	Yes	Yes	Variable name	first_name
Inside Class	No	Yes	Yes	No	For Class - Variable namae For Method - Class Instance.variable nama	For Class - first_name For Method - self.first_name
Inside Method	No	No	Yes	No	Variable name	first_name

- Class variables: Such variables which are created at a class level are called class variables.
  - Class variables belong to the class and hence it is incorrect to update them using the object reference variable or self. Doing so may cause unexpected consequences in the code and should be refrained from.
  - Class variables can be made as private by adding a double underscore in front of it and create getter and setter methods to access or modify it.
  - Read more here

```
class Mobile:
  discount = 50 #Class variable
  def __init__(self, price, brand):
    self.price = price
    self.brand = brand

#Class variables can be private and accessed by using getters and setters.
class Mobile:
  __discount = 50
  def get_discount(self):
    return Mobile.__discount
  def set_discount(self,discount):
    Mobile.__discount = discount
m1=Mobile()
print(m1.get_discount())
```

- Creating classmethod
  - a class variable is independent of the object, a way is needed to access the getter/setter methods without an object.
  - Class methods can be accessed without an object
  - the first argument to a class method is the reference of the class itself called cls.
  - There are two rules in creating such class methods:
    - Definitions of these methods should be prefixed with @classmethod
    - The methods should not have self. Instead, they should have the class reference cls as the first argument to the class method

```
@classmethod
def get_discount(cls):
  return cls.__discount
@classmethod
def set_discount(cls, discount):
  cls.__discount=discount
```

- Static methods
  - In a class, a method may be defined that neither accesses the class attributes nor the instance attributes.
  - Usually generic utility functions defined within the scope of a class.
  - static methods do not need an object to invoke them. They are accessed using the class name.
  - There are two rules in creating such static methods:
    - Definitions of these methods should be prefixed with @staticmethod
    - The methods should neither have self nor cls as the first argument. Instead, they can have zero or more arguments just like any other Python function

```
@staticmethod
def calculate_tax(cust_type):
  if(cust_type=='member'):
    return 0.10
  else:
    return 0.20
```

- Following is the complete solution using static methods, class methods and instance methods.

```
class Mobile:
  __discount = 50
  def __init__(self, price, brand):
    self.price = price
    self.brand = brand
  def purchase(self):
```

```
    total = self.price - self.price * Mobile.__discount / 100
    print (self.brand, "mobile with price", self.price, "is available after discount at", total)
  @classmethod
  def enable_discount(cls):
    cls.set_discount(50)
  @classmethod
  def disable_discount(cls):
    cls.set_discount(0)
  @classmethod
  def get_discount(cls):
    return cls.__discount
  @classmethod
  def set_discount(cls, discount):
    cls.__discount = discount
  @staticmethod
  def calculate_tax(cust_type):
    if(cust_type=='member'):
      return 10
    else:
      return 20

print('Tax percent to be paid by members:',Mobile.calculate_tax('member'))
print('Tax percent to be paid by non-members:',Mobile.calculate_tax('non-member'))
mob1=Mobile(20000, "Apple")
mob2=Mobile(30000, "Apple")
mob3=Mobile(5000, "Samsung")
Mobile.disable_discount()
mob1.purchase()
Mobile.enable_discount()
mob2.purchase()
```

Class Method	Static Method
1. Method is prefixed with @classmethod	Method is prefixed with @staticmethod
2. First argument is mandatory which is *cls* by default, followed by zero or more arguments	Arguments are optional. That is, zero or more arguments can be passed
3. Accesses class attributes within the method	Does not access class or instance attributes
4. Can modify state of the class	Cannot modify state of the class
5. Used to manipulate class attributes	Used as a generic utility function