Python Classes and Objects

Python is an object oriented programming language.

Almost everything in Python is an object, with its properties and methods.

Create a Class

To create a class, use the keyword class:

In [2]:
```
class MyClass:
  x = 5
```

Create Object

Now we can use the class named MyClass to create objects:

In [3]:
```
p1 = MyClass()
print(p1.x)
```

```
5
```

The **init** () Function

The examples above are classes and objects in their simplest form, and are not really useful in real life applications.

To understand the meaning of classes we have to understand the built-in **init** () function.

All classes have a function called **init** (), which is always executed when the class is being initiated.

Use the **init** () function to assign values to object properties, or other operations that are necessary to do when the object is being created:

In [4]:
```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

```
John
36
```

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:

In [5]:
```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1)
```

```
<__main__.Person object at 0x000001CB7C277610>
```

In [6]:
```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def __str__(self):
    return f"{self.name}({self.age})"

p1 = Person("John", 36)

print(p1)
```

```
John(36)
```

Object Methods

Objects can also contain methods. Methods in objects are functions that belong to the object.

Let us create a method in the Person class:

In [7]:
```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

```
Hello my name is John
```

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 and it should be the first parameter.

Access Modifiers in Python : Public, Private and Protected

The members of a class that are declared public are easily accessible from any part of the program. All data members and member functions of a class are public by default.

Protected Access Modifier:

The members of a class that are declared protected are only accessible to a class derived from it. Data members of a class are declared protected by adding a single underscore '_' symbol before the data member of that class.

In [8]:
```
class Student:

    # protected data members
    _name = None
    _roll = None
    _branch = None

    # constructor
    def __init__(self, name, roll, branch):
        self._name = name
        self._roll = roll
        self._branch = branch

    # protected member function
    def _displayRollAndBranch(self):

        # accessing protected data members
        print("Roll: ", self._roll)
        print("Branch: ", self._branch)
```

Private Access Modifier:

The members of a class that are declared private are accessible within the class only, private access modifier is the most secure access modifier. Data members of a class are declared private by adding a double underscore '__' symbol before the data member of that class.

In [9]:
```
class Student:

    # private members
    __name = None
    __roll = None
    __branch = None

    # constructor
    def __init__(self, name, roll, branch):
```

```
            self.__name = name
            self.__roll = roll
            self.__branch = branch

    # private member function
    def __displayDetails(self):

            # accessing private data members
            print("Name: ", self.__name)
            print("Roll: ", self.__roll)
            print("Branch: ", self.__branch)

    # public member function
    def accessPrivateFunction(self):

            # accessing private member function
            self.__displayDetails()

# creating object
obj = Student("R2J", 1706256, "Information Technology")

# calling public member function of the class
obj.accessPrivateFunction()
```

```
Name:  R2J
Roll:  1706256
Branch:  Information Technology
```

Python Inheritance

Inheritance allows us to define a class that inherits all the methods and properties from another class.

Parent class is the class being inherited from, also called base class.

Child class is the class that inherits from another class, also called derived class.

Create a Parent Class

Any class can be a parent class, so the syntax is the same as creating any other class:

In [1]:
```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

#Use the Person class to create an object, and then execute the printname method:

x = Person("John", "Doe")
x.printname()
```

```
John Doe
```

Create a Child Class

To create a class that inherits the functionality from another class, send the parent class as a parameter when creating the child class:

In [2]:
```
class Student(Person):
  pass
```

Now the Student class has the same properties and methods as the Person class.

In [3]:
```
x = Student("Mike", "Olsen")
x.printname()
```

```
Mike Olsen
```

Add the **init**() Function

So far we have created a child class that inherits the properties and methods from its parent.

We want to add the **init**() function to the child class (instead of the pass keyword).

In [5]:
```
class Student(Person):
  def __init__(self, fname, lname):
    #add properties etc.
    pass
```

When you add the **init**() function, the child class will no longer inherit the parent's **init**() function.

To keep the inheritance of the parent's **init**() function, add a call to the parent's **init**() function:

In [6]:
```
class Student(Person):
  def __init__(self, fname, lname):
    Person.__init__(self, fname, lname)
```

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:

In [7]:
```
class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)
```

Add Properties

In [8]:
```
class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year
```

```
x = Student("Mike", "Olsen", 2019)
```

Add Methods

In [9]:
```
class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year

  def welcome(self):
    print("Welcome", self.firstname, self.lastname, "to the class of", self.graduatior
```

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.

In [ ]:
```
```