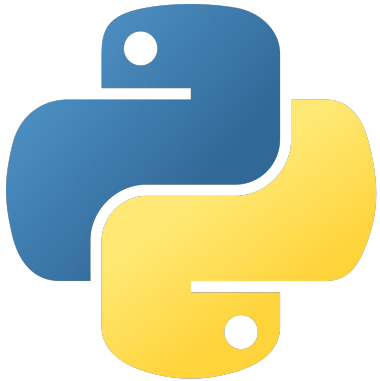
Python Programming Language



Input Data

```
name = input("name")
print(name)

Result

input -> 'Testing'

result -> Testing
```

Object-oriented programming (OOP)

Object-Oriented Programming (OOP) is a programming paradigm that emphasizes the use of objects as the key elements of structure and functionality in a program. It utilizes classes as templates or blueprints for creating objects.

1. Class: A class is a blueprint or template for creating objects, defining their properties and behaviors.
2. Object: An object is an instance of a class, possessing state (data) and behavior (methods) as defined by the class.
3. Inheritance: Classes can inherit properties and behaviors from other classes, allowing code reuse and enhancing flexibility in development.
4. Encapsulation: Classes and objects control access to and modification of their internal data, using techniques to prevent direct access to data and utilizing methods to manipulate data.
5. Polymorphism: The ability to provide different implementations of methods depending on the context, or to have methods with the same name exhibit different behaviors based on the class.

OOP helps to organize code in a more structured manner, making it easier to understand and reusable, as it allows for clear separation of concerns and encapsulation of object characteristics and behaviors. It provides flexibility in making changes and extending functionality efficiently.

Class / Object

```
class MyClass:
  x = 5

print(MyClass)
```

Result

```
<class '__main__.MyClass'>
```

```
p1 = MyClass()
print(p1.x)
```

Result

```
5
```

The __init__() Function

```
class Person:
  def   init  (self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

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

Result

```
John
36
```

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:

The __str__() Function

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1)
```

Result

```
<__main__.Person object at 0x15039e602100>
```

```
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)
```

Result

```
John(36)
```

Object Methods

```
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()
```

Result

```
Hello my name is John
```

The self Parameter

```
class Person:
  def   init  (mysillyobject, name, age):
    mysillyobject.name = name
    mysillyobject.age = age

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

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

Result

```
Hello my name is John
```

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:

Modify And Delete Object Properties

```
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.age = 40

print(p1.age)
```

Result

```
40
```

```
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)

del p1.age

print(p1.age)
```

Result

```
attributeError: 'Person' object has no
attribute 'age'
```

The pass Statement

```
class Person:
  pass
```