The Learning Hub

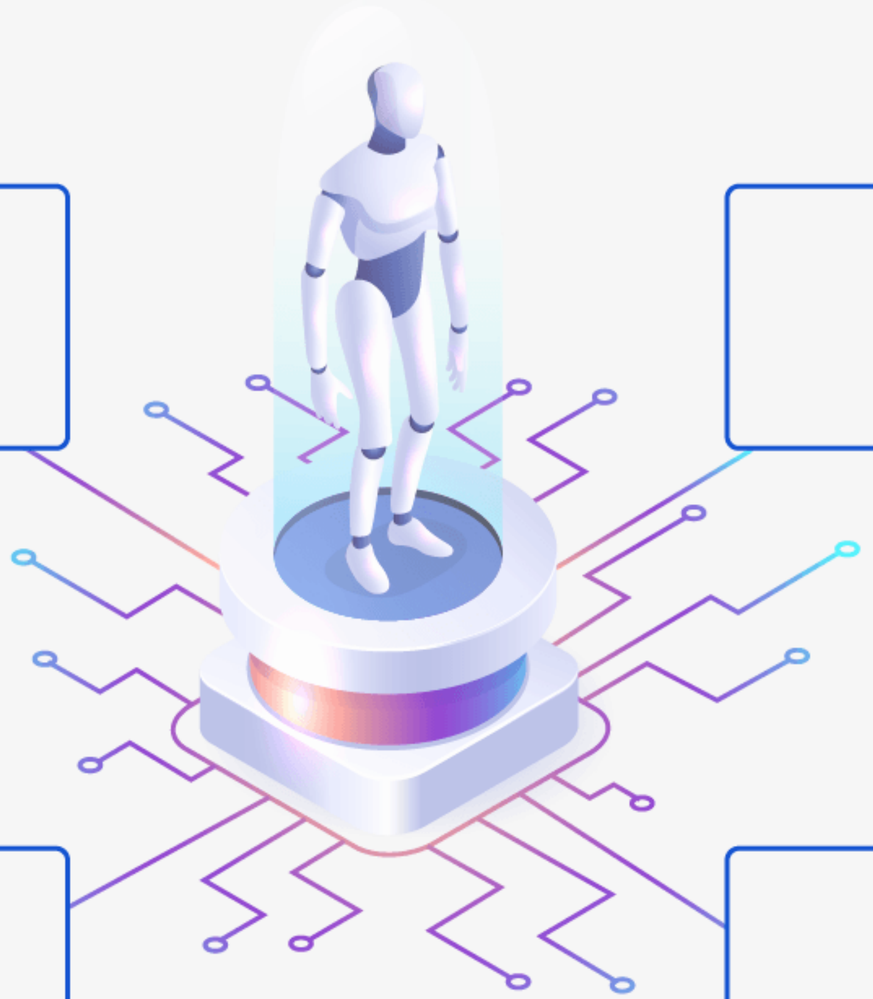
Object Oriented Programming (Classes)

Education and Training Solutions 2023

The Learning Hub

Classes

Scopes



Inheritance
& Polymorphism

Iterators
& Generators

The Learning Hub

Classes

Class Definition Syntax

- Python is an object-oriented programming language. Everything in python is an object.
- Class is an object constructor, which means of creating a structured code in the same block.

- Python Class Syntax:

```
class ClassName:
    <statement-1>
    .
    .
    .
    <statement-N>
```

Create a Class

- Use the keyword class when you want to initialize a class.

```
1  class MyClass:
2      x = 5
3
4  print(MyClass)
5
```

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

Create an Object

- You can use the class name (MyClass) to create objects.

```
1  class MyClass:
2      x = 5
3
4  obj = MyClass()
5  print(obj.x)
```

```
5
```

The __init__() Function

- The previous examples are the simplest form of using the classes and objects, in real world applications it is different.
- All classes have function called __init__(), which will be executed when the class is executed.

```
1  class Person:
2      def __init__(self, name, age):
3          self.name = name
4          self.age = age
5
6  obj2 = Person("John", 36)
7
8  print(obj2.name)
9  print(obj2.age)
```

```
John
36
```

Object Methods

- Objects can contain methods and functions that belong to the same class.

```
1  class Person:
2      def __init__(self, name, age):
3          self.name = name
4          self.age = age
5
6      def myfunc(self):
7          print("Hello my name is " + self.name)
8
9  p1 = Person("John", 36)
10 p1.myfunc()
```

```
Hello my name is John
```

The self Parameter

- The self parameter is a reference to the current instance of the class, used to access all variables that belongs the same class.

```
1  class Person:
2      def __init__(self, name, age):
3          self.name = name
4          self.age = age
5
6      def Name(self):
7          print("Hello my name is " , self.name)
8
9      def Age(self):
10         print("age " , self.age)
11
12 p1 = Person("John", 36)
13 p1.Name()
14 p1.Age()
```

```
Hello my name is  John
age  36
```

Modify Object Properties

- We can modify the object properties by assigning new value.

```
1  class Person:
2      def __init__(self, name, age):
3          self.name = name
4          self.age = age
5
6      def myfunc(self):
7          print("Hello my name is " + self.name)
8
9  p1 = Person("John", 36)
10
11 p1.age = 40
12
13 print(p1.age)
```

```
40
```

Delete Object Properties

- We can delete the object properties by using the del keyword.

```
1  class Person:
2      def __init__(self, name, age):
3          self.name = name
4          self.age = age
5
6      def myfunc(self):
7          print("Hello my name is " + self.name)
8
9  p1 = Person("John", 36)
10
11 del p1.age
12
13 print(p1.age)
14
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [27], in <cell line: 13>()
      9 p1 = Person("John", 36)
     11 del p1.age
---> 13 print(p1.age)

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

The Learning Hub

Scopes

Local Scope

- Variable is available only inside his own region (function, class), this is called scope.
- Variable is created inside a function belongs to the local scope of this function, just can be used inside this function.

```
1  class MyClass:
2
3      def myfunc():
4          x = 300 #local variable
5          print(x)
6
7  MyClass.myfunc()
```

```
300
```

Function inside function

- The variable x is not accessible outside the function, but it is accessible for all functions inside the main function.

```
1  class MyClass:
2
3      def myfunc():
4          x = 150
5          def myinnerfunc(): #function iside function scope
6              print(x)
7
8          myinnerfunc()
9
10 MyClass.myfunc()
```

```
150
```

Global Scope

- Variable can be created in the main of our python code, which will be accessible to all functions in the global scope.

```
1  class MyClass:
2
3      x = 100 #gobal variable
4
5      def myfunc():
6          print(x)
7
8
9  MyClass.myfunc()
10
11 MyClass.x
```

```
100

100
```

Global Keyword

- Global keyword is used when you want to create a global variable inside the local scope.

```
1  x = 300
2
3  def myfunc():
4      global x
5      x = 200
6
7  myfunc()
8
9  print(x)
```

```
200
```

The Learning Hub

Inheritance

Python Inheritance

- Inheritance is used to define a class that inherits methods, prosperities and all variables from another class.

- Parent class is the class being inherited from, called Base class.
- Child class is the class that inherits from another class, also called Derived class.

Create a Parent Class

- Create a class named Student, with name and number properties, and a Result method.

```
1  class Student:
2      def __init__(self, name, number):
3          self.name = name
4          self.number = number
5
6      def Result(self):
7          result = "Name: " + self.name + " || Number: " + str(self.number)
8          return result
9
10 st = Student("Jack", 1234)
11 st.Result()
```

```
'Name: Jack || Number: 1234'
```

Create a Child Class

- Create a class named Child, which will inherit different methods and properties from the Student class.

```
class Student:
    def __init__(self, name, number):
        self.name = name
        self.number = number

    def Result(self):
        result = "Name: " + self.name + " || Number: " + str(self.number)
        return result

class Child(Student):
    pass

st = Child("Rick", 9876)
st.Result()
```

```
'Name: Rick || Number: 9876'
```

Add Methods in Child class

- Add a method named Data to the Child Class.

```
1  class Student:
2      def __init__(self, name, number):
3          self.name = name
4          self.number = number
5
6      def Result(self):
7          result = "Name: " + self.name + " || Number: " + str(self.number)
8          return result
9
10 class Child(Student):
11     def __init__(self, name, number, year):
12         super().__init__(name, number)
13         self.year = year
14
15     def Data(self):
16         print("Welcome", self.name, self.number, "to the class of", self.year)
17
18 st = Child("Jordan", 5564, 2019)
19 st.Data()
```

```
Welcome Jordan 5564 to the class of 2019
```

The Learning Hub

Polymorphism

Python Polymorphism

- Polymorphism is taken from the Greek words **Poly** which means **Many** and **Morphism** which means **Forms**. It means that the same name of function will be used in different types. This will make the programming more efficient to use.

- Child Class inherits all functions and methods from the parent class. In some cases, the method inherited from the parent class can not fit into child class. You will have to implement method in the child class.

Polymorphism with Class Methods

- Python can use two different class types in the same way. You can create a for loop which will iterates through a tuple of objects. Each classes have its own properties with the same name.

```
class Jordan():
     def capital(self):
        print("Amman")

     def language(self):
        print("Arabic")

class UAE():
    def capital(self):
        print("Abu Dhabi")

    def language(self):
        print("Arabic")

jor = Jordan()
uae = UAE()
for country in (jor, uae):
    country.capital()
    country.language()
```

```
Amman
Arabic
Abu Dhabi
Arabic
```

Polymorphism with Inheritance

- Polymorphism in python defines methods in the child class that have the same name of the methods in the parent (super) class. By inheritance the **child class** inherits all the methods and properties from the **parent class**. You can modify method in child class that inherits from the parent class.

Polymorphism with Inheritance

- Polymorphism in python defines methods in the child class that have the same name of the methods in the parent (super) class. By inheritance the **child class** inherits all the methods and properties from the **parent class**. You can modify method in child class that inherits from the parent class.

- Overriding is the process of creating or re-implementing a method in the child class where the method inherited from the parent class couldn't fit the child class.

Polymorphism with Inheritance

```
class Bird:
     def intro(self):
        print("There are different types of birds")

     def flight(self):
        print("Most of the birds can fly but some cannot")

class parrot(Bird):
     def flight(self):
        print("Parrots can fly")

class penguin(Bird):
     def flight(self):
        print("Penguins do not fly")

obj_bird = Bird()
obj_parr = parrot()
obj_peng = penguin()

obj_bird.intro()
obj_bird.flight()

obj_parr.intro()
obj_parr.flight()

obj_peng.intro()
obj_peng.flight()
```

```
There are different types of birds
Most of the birds can fly but some cannot
There are different types of birds
Parrots can fly
There are different types of birds
Penguins do not fly
```

Iterators

- Behind the scenes, the for statement calls iter() on the container object. The function returns an iterator object that defines the method __next__() which accesses elements in the container one at a time. When there are no more elements, __next__() raises a StopIteration exception which tells the for loop to terminate. You can call the __next__() method using the next() built-in function.

Example of Iterators

```
1  s = 'abc'
2  it = iter(s)
3  it
4
5  next(it) #a
6
7  next(it) #b
8
9  next(it) #c
10
11 next(it) #StopIteration:
12
```

```
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
Input In [6], in <cell line: 11>()
      7 next(it) #b
      9 next(it) #c
---> 11 next(it)

StopIteration:
```

Generators

- Generators are a simple and powerful tool for creating iterators. They are written like regular functions but use the yield statement whenever they want to return data.

```
1  class rev():
2      def reverse(data):
3          for index in range(len(data)-1, -1, -1): #start stop step
4              yield data[index]
5
6  for char in rev.reverse('tahaluf'):
7      print(char)
```

```
f
u
l
a
h
a
t
```

References

- Python Tutorial (w3schools.com)
- The Python Tutorial — Python 3.10.7 documentation
- Python Tutorial for Beginners: Learn Programming Basics [PDF] (guru99.com)
- https://www.edureka.co/

THANK YOU