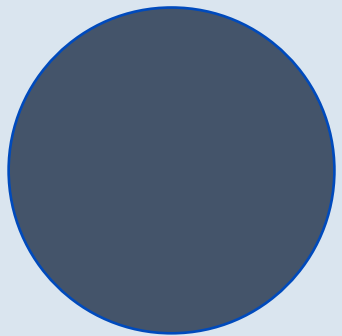




Selected topics 1

Python



Dr: Nashwa Nageh



Python classes

Create a Class

To create a class, use the keyword `class`:

Example

Create a class named MyClass, with a property named x:

```
class MyClass:  
    x = 5
```

Python classes

Create Object

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

Example

Create an object named p1, and print the value of x:

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

Python classes

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:

```
class Member:
    def __init__(self):
        self.name="Adam"
member1 = Member()
print(member1.name)
```

Adam



Python classes

```
class Member:
    def __init__(self):
        self.name="Adam"
member1 = Member()
member2 = Member()
member3 = Member()
print(member1.name)
print(member2.name)
print(member3.name)
```

Adam
Adam
Adam

```
class Member:
    def __init__(self,fname):
        self.name="Adam"
member1 = Member("Adam")
member2 = Member("Anas")
member3 = Member("Elie")
print(member1.name)
print(member2.name)
print(member3.name)
```

Adam
Adam
Adam

```
: class Member:
    def __init__(self,fname):
        self.name=fname
member1 = Member("Adam")
member2 = Member("Anas")
member3 = Member("Elie")
print(member1.name)
print(member2.name)
print(member3.name)
```

Adam
Anas
Elie

when you construct new instance the attribute
name almost equal Adam

Note: The default `__init__` constructor in Python is the constructor that does not accept any parameters.

Python classes

Example

Create a class named Person, use the `__init__()` function to assign values for name and age:

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


p1 = Person("John", 36)

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

Python classes

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

Self: This handy keyword allows you to access variables, attributes, and methods of a defined class in Python.



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

```
print(p1.x)  
print(p1.y)
```

Note: The `__init__()` function is called automatically every time the class is being used to create a new object.

Instance Attributes And Methods

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

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

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

Instance method must take self parameter
which point to instance created from class

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:

Example

Use the words *mysillyobject* and *abc* instead of *self*:

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

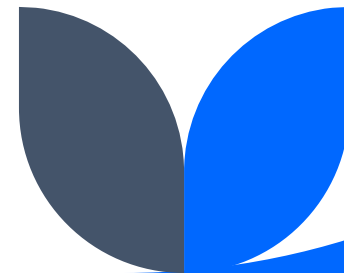
    def myfunc(abc):
        print("Hello my name is " + abc.name)

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

Python classes

```
class Person:  
    def __init__(self2, name, age):  
        self2.x = name  
        self2.y = age  
  
    def myfunc(self3):  
        print("Hello my name is " + self3.x)
```

```
p1 = Person("John", 36)  
p1.myfunc()  
p1.y = 40  
print(p1.y)
```



Delete Object Properties

```
class Person:
    def __init__(self, fname, age):
        self.name = fname
        self.age = age
    def fun(self2):
        print(self2.x)

p1 = Person("John", 36)
del p1.age
print(p1.age)
```

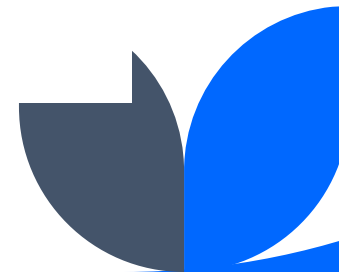
AttributeError

Traceback (most recent call last)

Cell In[19], line 10

```
8 p1 = Person("John", 36)
9 del p1.age
--> 10 print(p1.age)
```

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



Delete Objects

```
class Person:
    def __init__(self, fname, age):
        self.name = fname
        self.age = age
    def fun (self2):
        print(self2.x)

p1 = Person("John", 36)
del p1
print(p1)
```

NameError

Traceback (most recent call last)

Cell In[20], line 10

```
      8 p1 = Person("John", 36)
      9 del p1
--> 10 print(p1)
```

NameError: name 'p1' is not defined

Python classes

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.

Python classes

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

Create a class named `Person`, with `firstname` and `lastname` properties, and a `printname` method:

```
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()
```

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:

Example

Create a class named `Student`, which will inherit the properties and methods from the `Person` class:

```
class Student(Person):  
    pass
```

Note: Use the `pass` keyword when you do not want to add any other properties or methods to the class.

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

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

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

```
x1 = Person("John", "Doe")
x1.printname()
```

```
class Student(Person):
    pass
```

```
x2 = Student("Mike", "Olsen")
x2.printname()
```


Inheritance and subclass

- In a class definition for a subclass:

- To indicate inheritance, the superclass name is placed in parentheses after subclass name
- The initializer method of a subclass calls the initializer method of the superclass and then initializes the unique data attributes
- Add method definitions for unique methods

```
class Person(object):

    # __init__ is known as the constructor
    def __init__(self, name, idnumber):
        self.name = name
        self.idnumber = idnumber

    def display(self):
        print(self.name)
        print(self.idnumber)

# child class
class Employee(Person):
    def __init__(self, name, idnumber, salary, post):
        self.salary = salary
        self.post = post

        # invoking the __init__ of the parent class
        Person.__init__(self, name, idnumber)

# creation of an object variable or an instance
employee1 = Employee('Ahmed', 123, 200000, "Intern")

# calling a function of the class Person using its instance
employee1.display()
```

Python code to demonstrate how
parent constructors are called.

output

Ahmed
123

```
class Person:
    def __init__(self, fname, lname):
        self.firstname = fname
        self.lastname = lname
    def printname(self):
        print(self.firstname, self.lastname)
x1 = Person("John", "Doe")
x1.printname()
```

```
class Student(Person):
    def __init__(self, fname, lname, age):
        Person.__init__(self, fname, lname)
        self.studentage=age
    def printname(self):
        Person.printname(self)
        print(self.studentage)

x2 = Student("Mike", "Olsen", 36)
x2.printname()
```

method overriding