

Python tokens

Wednesday, August 17, 2022 2:29 PM

A

In Python, every logical line of code is broken down into components known as **Tokens**

Normal Token Types

Keywords

Identifiers

Literals

Operators

1. Python keywords

Python Keywords

True	False	None	and	as
Asset	def	class	continue	break
else	finally	elif	del	except
global	for	if	from	import
raise	try	or	return	pass
nonlocal	in	not	is	lambda

2. Identifier

An **identifier** is the name used to identify a variable, function, class, or object

Rules for naming an identifier:

1. No special character, except underscore (_), can be used as an identifier
2. Keywords should not be used as an identifier
3. Python is case sensitive, i.e., 'Var' and 'var' are two different identifiers
4. The first character of an identifier can be a alphabet or underscore (_) but not a digit

3.literals

A **literal** is the raw data given to a variable

Various Types of Literals

String literals

Numeric literals

Boolean literals

Special literals

String literals

What are string literals?

Formed by enclosing a text within quotes. Both single and double quotes can be used

Input

```
name1="John"
name2='James'
print(name1)
print(name2)
text1='hello\
World'
print(text1)
multiline='''st1
st2
st3'''
print(multiline)
```

Output

```
John
James
helloWorld
st1
st2
st3
```

Note

We can also type whole paragraph in the python

```
>>> line = '''
... nnuifwbviwv jsbfuwia cakjssanvcnavuiwadhc iuwac
... awfuhwacdwaiofjcwaiddoah dhfiualhfdasio
... cduwhciuwdhiua guafid
... fuhawiduhuiladbiayd ahfuia
... '''
>>> line
'\nnnuifwbviwv jsbfuwia cakjssanvcnavuiwadhc iuwac\nawfuhwacdwaiofjcwaiddoah
nfuhawiduhuiladbiayd ahfuia\n'
>>>
```

Numeric literals

What are numeric literals?

Formed by a character string of digits from 0 to 9, decimal point, and a plus/minus sign

Numeric Literal Formats

}

Int	Long	Float	Complex
+ve and -ve numbers (integers) with no fractional part E.g.: 100,-234	An unlimited string of integers followed by upper or lowercase L E.g.: 233424243L	Real numbers with both integer and fractional parts E.g.: -213.3	Strings in the form of $a+bj$, where 'a' is the real part & 'b' is the imaginary part E.g.: 3.14j

Boolean literals

What are Boolean literals?

It can either be **True** or **False**

```
var1 = True
var2 = False
var1 == var2
```

False

```
var1 = True
var2 = True
var1 == var2
```

True

Special literals

What are special literals?

There is a special literal in Python called **None** which means that the variable is yet to be initialized

```
val1 = 10  
val2 = None  
print(val1)  
type(val1)
```

10

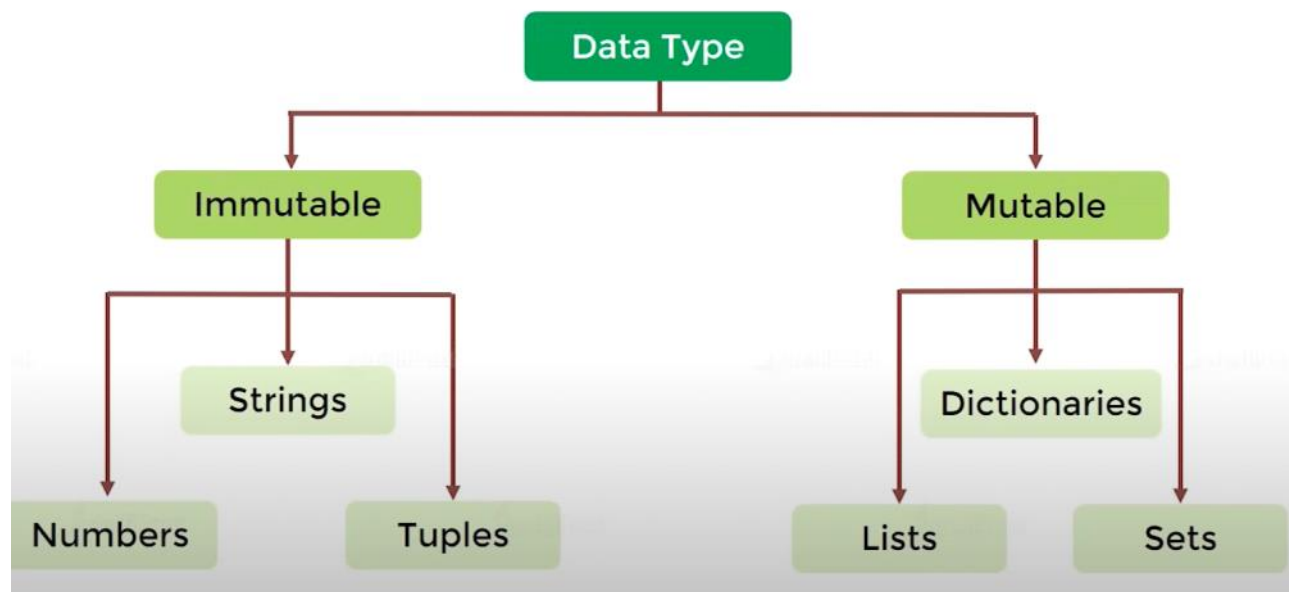
int

```
val1 = 10  
val2 = None  
print(val2)  
type(val2)
```

None

NoneType

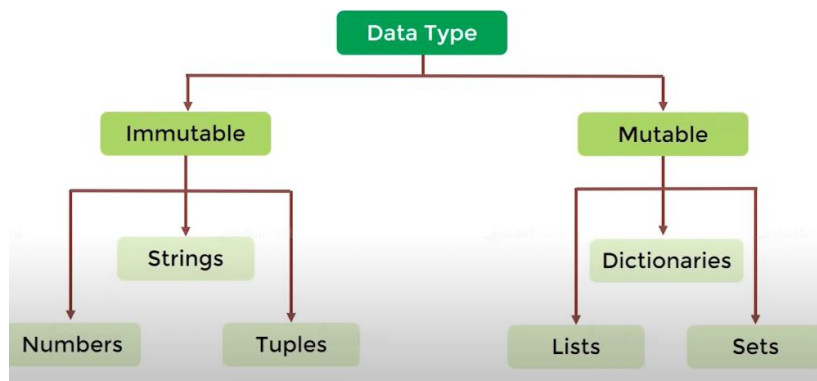
DATA TYPES



DATA TYPES and also for loop

Thursday, August 18, 2022 10:51 AM

A



Numbers

There are three numeric data types:

Data Type	Description
Int	Integer type (whole number, no decimals)
float	Floating point type (one or more decimals)
complex	Written with a imaginary number j

```
num = 1234
rate = 13.6
com = 6j
print(type(num))
print(type(rate))
print(type(com))

<class 'int'>
<class 'float'>
<class 'complex'>
```

String methods

```
Select Python 3.7 (64-bit)
>>> string = "abcdefgh"
>>> len(string)
8
>>> _
```

We can find the value by the index number

```
>>> string[0]
'a'
>>> string[4]
'e'
>>> string[1]
'b'
>>> string[2]
'c'
>>> string[3]
'd'
>>> string[0:4]
'abcd'
>>>
```

In the above array we can also perform the replace

```
>>> string.replace('f', 'g')
'abcdeggh'
>>> _
```

Tuples

Tuples

A sequence of immutable Python objects

```
myGroup = ('a', 'b', 'c', 'd')
```

concatenation

```
#Concatenation - Adds two string/character
myGroup += ('f',)
print(myGroup)
```

```
('a', 'b', 'c', 'd', 'f')
```

```
>>> weeks = ("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday")
>>> weeks
('Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday')
```

LIST

Lists

A sequence of mutable objects

```
myGroup = ('a', 10, 7.12, 'data')
```

concatenation

```
#Concatenation - Add elements to the list
myList = ['a', 1, 3.14, 'python']
myList += ['d',]
print(myList)
```

```
['a', 1, 3.14, 'python', 'd']
```

```
>>> salaries = [10, 15, 20]
>>> salaries
[10, 15, 20]
>>> type(salaries)
<class 'list'>
```

```

>>> salaries
[18, 15, 20]
>>> salaries[0] = 10
>>> salaries[0] = salaries[0] + 8
>>> salaries
[18, 15, 20]
>>> salaries[0] += 8
>>> salaries
[26, 15, 20]
>>> salaries.append(28)
>>> salaries
[26, 15, 20, 28]

```

For tuple we use () and for list we use []

```

>>> mixed = [1, 'a', 1.5, [155, 'b']]
>>> mixed = [1, 'a', 1.5, [155, 'b'], (88, 777)]
>>> mixed
[1, 'a', 1.5, [155, 'b'], (88, 777)]
>>> mixed[4]
(88, 777)
>>> type(mixed[4])
... )
<class 'tuple'>
>>> type(mixed[3])
<class 'list'>

```

The above image justify that in a list we can save the tuple as you can see we saved a int string and float with the list in it and a tuple in it

Dictionaries

Dictionaries

An unordered collection of items

```

myDict = { 1: 'John' , 2: 'Bob', 3: 'Alice' }
myDict

```

```

{1: 'John', 2: 'Bob', 3: 'Alice'}

```

Empty dictionary

```

#empty dictionary
myDict = {}

```



```
<class 'dict'>
>>> salaries = { "John": 15, "Jane": 14, "Johnny": 5 }
>>> salaries
{'John': 15, 'Jane': 14, 'Johnny': 5}
>>> type(salaries)
<class 'dict'>
>>> salaries['Jane']
14
>>> salaries['John']
15
```

```
>>> salaries.get("John", 15)
15
```

Here we are using get method where I want value of john if john is there in dictionary it will return john value or if there is no john it will give 15

```
>>> salaries.get("Johnasd", 18)
18
```

As you can see we do not have Johnasd so we get 18

Sets

Sets

An unordered collection of immutable data which has no duplicate elements

```
mySet = {1, 2, 3}

#Creating set
mySet = {1, 2, 3, 3}
print (mySet)

{1, 2, 3}

#Union
myS1 = {1, 2, 'c'}
myS2 = {1, 'b', 'c'}
myS1 | myS2

{1, 2, 'b', 'c'}
```

```
>>> s = set()
>>> type(s)
<class 'set'>
>>> s.add(1)
>>> s
{1}
>>> s.add(2)
>>> s
{1, 2}
>>> s.add(2)
>>> s
{1, 2}
>>> 1 in s
True
```

For loop in python

```

Python Course | Intellipaat
nums = [1, 2, 3, 4]
print(nums)

nums.append(5)
print(nums)

x = nums.pop(0)
print(x)
print(nums)

for x in nums:
    print(x)

```

Then it will read the values in the array

To add some thing new in the dictionary

```

sal = { "Jane": 15, "John": 20 }
sal["Andy"] = 25

print(sal)

```

Now we are using for to the dictionary

```

sal = { "Jane": 15, "John": 20 }
sal["Andy"] = 25
print(sal)
del sal["Andy"]
print(sal)

for item in sal.keys():
    print(item)

```

So now for will read only the value not the key pair of it like it will read john and jane

```

C:\Users\ANIRUDH\Desktop\Hands On>python app.py
{'Jane': 15, 'John': 20, 'Andy': 25}
{'Jane': 15, 'John': 20}
Jane
John

```

But in some we also need the key with it

```

sal = { "Jane": 15, "John": 20 }
sal["Andy"] = 25
print(sal)
del sal["Andy"]
print(sal)

for key, val in sal.items():
    print(key, val)

```

```
C:\Users\ANIRUDH\Desktop\Hands On>python app.py
{'Jane': 15, 'John': 20, 'Andy': 25}
{'Jane': 15, 'John': 20}
Jane 15
John 20
```

Here I am writing a program which will save automatically numbers in the empty list

```
l = []

for x in range(0, 10):
    l.append(x)

print(x)
```

We need to print l not x

```
C:\Users\ANIRUDH\Desktop\Hands On>python app.py
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Compressed version of above code

```
l = [x for x in range(0, 9)]

print(l)
```

Conditional statement if else

Thursday, August 18, 2022 12:03 PM

A

If else

```
x = 20

if x < 20:
    print("X is less than 20")
else:
    print("X is greater than 20")
```

Elif

```
py > ...
x = 20

if x < 20:
    print("X is less than 20")
elif x == 20:
    print("X is equal to 20")
else:
    print("X is greater than 20")
```

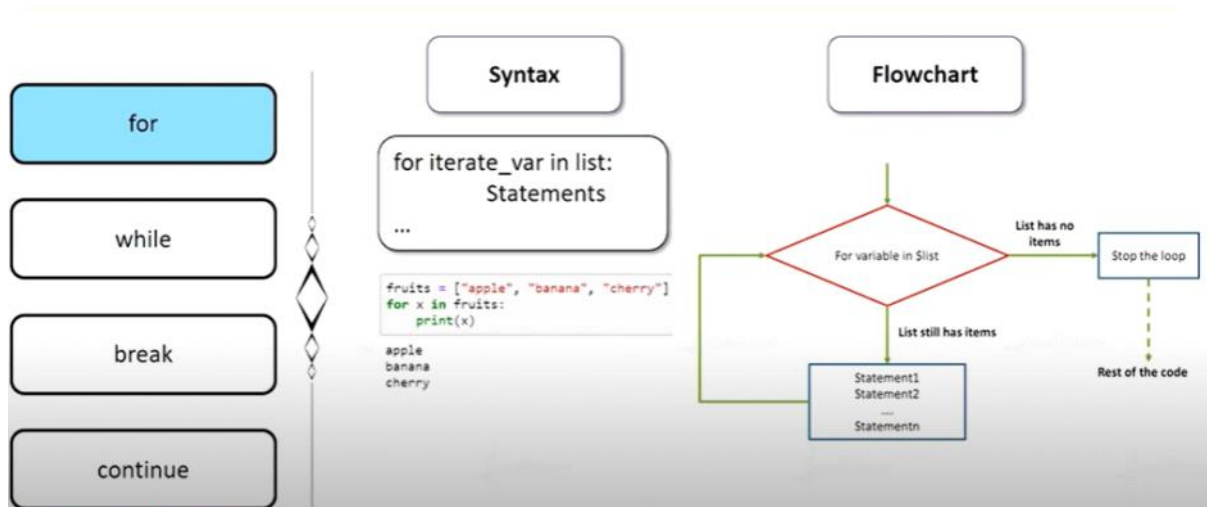
Nested if else

```
x = 10

if x < 20:
    print("X is less than 20")
    if x < 15:
        print("X is less than 15")
    else:
        print("X is greater than 15")
else:
    print("X is greater than 20")
```

Loops

Thursday, August 18, 2022 12:12 PM



Using for and break

```
nums = [x for x in range(0, 10)]

for x in nums:
    if (x == 4): break
    print(x)

print("Done")
print()
```

Output will be 0 1 2 3 4

For with continue

```

nums = [x for x in range(0, 10)]

for x in nums:
    if (x == 4): continue
    print(x)

print("Done")

```

here it will skip the 4
Output 0 1 2 3 5 6 7 8 9 10

While

```

a = 10

while a != 0:
    print(a)
    a -= 1

```

Here value is 10
Then we need to read it up to 1
And then a-=1 so slowing value will decrease by 1

So output will be 10 9 8 7 6 5 4 3 2 1

Functions in Python

Friday, August 19, 2022 10:13 AM

A

There are totally 2 function

1. User define function
2. Built in function

User define function

Example

```
def add (a,b):  
    sum = a + b  
    return sum
```

these are commonly used in the code

Build in functions

A function already available in a language that we can directly use in our code

abs(): Returns the absolute value of a number

all(): Returns True if all items in an iterable object are true

any(): Returns True if any item in an iterable object is true

ascii(): Returns a readable version of an object and replaces non-ASCII characters with an 'escape' character

bin(): Returns the binary version of a number

bool(): Returns the Boolean value of a specified object


```

C:\Users\akshay.kanemoni>python
Python 3.9.5 (tags/v3.9.5:0a7dcbd, May 3 2021, 17:27:52) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> abs(10.234)
10.234
>>> any("string")
True
>>> any(10)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not iterable
>>> ascii("A")
"'A'"
>>> ascii(1)
'1'
>>> bin(1)
'0b1'
>>> bin("a")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object cannot be interpreted as an integer
>>> bool("string")
True
>>>

```

The main difference between the ANY and the All is in all the series which we give everything should be true
In any there is no need the series things should all be true we can also give some false one

Lambda functions

Lambda Function

An anonymous function, i.e., a function having no name. A Lambda function cannot contain more than one expression

```

#Syntax
...
lambda arguments : expression
...

x = lambda a : a + 10
print(x(5))

```

There is also a facility where we can use a function in a lambda function

```
lambda.py > ...
1  def myfunc(n):
2      return lambda a: a+n
3  mySum = myfunc(3)
4  print(mySum(10))
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** JUPYTER

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell <https://aka.ms/pscore6>

PS D:\akshay.kanemoni\OneDrive - Infosys Limited\Documents> python 9/python.exe "d:/akshay.kanemoni/OneDrive - Infosys Limited/lambda.py"
13
PS D:\akshay.kanemoni\OneDrive - Infosys Limited\Documents>

An another way to describe the lambda functions

```
5
6  r = lambda x,y:x*y
7  print(r(12,3))
8
```

PROBLEMS OUTPUT **DEBUG CONSOLE**

Windows PowerShell
Copyright (C) Microsoft Corporation

Try the new cross-platform PowerShell

PS D:\akshay.kanemoni\OneDrive - Infosys Limited\Documents> python 9/python.exe "d:/akshay.kanemoni/OneDrive - Infosys Limited/lambda.py"
36

```
def createMultiplier(x):
    return lambda y: x * y

multiply = createMultiplier(10)

def execute(f, arg):
    print("Called F with " + str(arg))
    return f(arg)

print(execute(multiply, 15))
print(execute(multiply, 25))
```

Here f = multiply then f = args = multiply= y

Arrays in python

Friday, August 19, 2022 11:06 AM

Arrays in python

Used to store multiple values in a single variable

Python does not have built-in support for arrays, but Python lists can be used instead

```
#Storing in multiple variable
car1 = "Ford";
car2 = "Volvo";
car3 = "BMW";
#Using Array
cars = ["Ford", "Volvo", "BMW"]
print(car1)
cars
```

Ford

01

Accessing an element

```
x = cars[0]
car1 = cars[0]
print(len(cars))
4
'Honda'
```

02

Removing an element from a position

```
cars
cars[0] = "Honda"
print(cars[0])
Honda
cars.pop(1)
['Ford', 'BMW']
```

03

Getting the length of an array

```
x = len(cars)
x
```

04

Removing a specific element

```
for x in cars:
    print(x)
```

05

Adding an element

```
cars.append("Opel")  
print(len(cars))
```

4

06

Removing an element from a position

```
cars  
['Ford', 'Volvo', 'BMW']  
  
cars.pop(1)  
cars  
['Ford', 'BMW']
```

07

Removing a specific element

```
cars  
['Ford', 'Volvo', 'BMW']
```

Files in Python

Friday, August 19, 2022 11:12 AM

A

What is a class and what is an object in Python?

1. Python is an object-oriented programming language
2. Almost everything in Python is an object, with its properties and methods
3. A class is like a 'blueprint' for creating objects

Class

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

Object

```
obj1 = MyClass()  
print(obj1.x)
```

5

Why do we need file handling?

File handling is important in any application that handles permanent data. We will need file handling if we have to read from or write to files



There are 4 aspects in the file management

- 1.read
- 2.Write/Create
- 3.Open
- 4.Delete

Create

To create a new file

- **'x' – Create:** Creates a file; returns an error if the file already exists

```
# Create a new file  
f = open("myfile.txt", "x")
```

Open

Open

Read

Write/Create

Delete

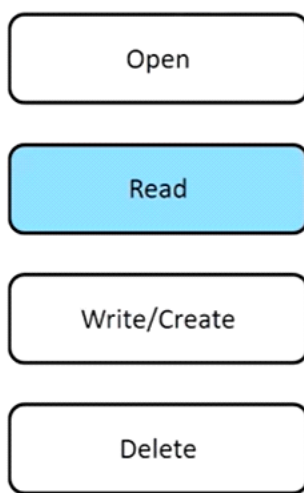
The open() function takes two parameters:
filename and mode

```
#Syntax  
f = open("path of file")
```

Mode Options

- **'r' – Read:** The default value; opens a file for reading; returns an error if the file does not exist
- **'a' – Append:** Opens a file for appending; creates the file if it does not exist
- **'w' – Write:** Opens a file for writing; creates the file if it does not exist
- **'x' – Create:** Creates the specified file; returns an error if a file with the same name already exists

Read



The read() function is used to read n bytes from the mentioned file

```
#Example
f = open("demofile.txt", "r")
print(f.read())
```

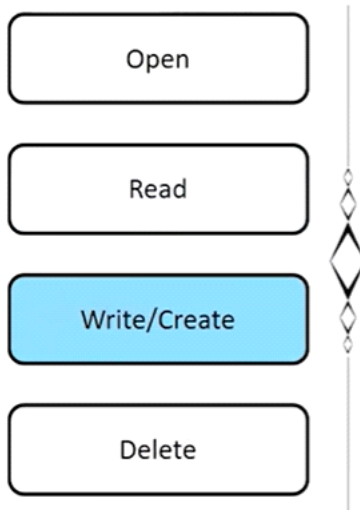
Reading the first 5 lines

```
#Reading parts of file
f = open("demofile.txt", "r")
print(f.read(5))
```

Reading line by line

```
#Loop through the file
#Read the file line by line
f = open("demofile.txt", "r")
for x in f:
    print(x)
```

Write



To write to an existing file, we must add a parameter to the open() function

- **'a' – Append:** Appends at the end of the file
- **'w' – Write:** Overwrites the existing content of the file

```
#Example: Append
f = open("demofile.txt", "a")
f.write("Now the file has one more line!")
```

```
#Example: Overwrite
f = open("demofile.txt", "w")
f.write("Woops! I have deleted the content!")
```

Delete

Open

Read

Write/Create

Delete



To import the OS module

Use the remove() function to delete the mentioned file

```
# Deleting the file  
import os  
os.remove("demofile.txt")
```


OOPS In python

Friday, August 19, 2022 11:36 AM

A

elliPaat

Agenda

- | | |
|--------------------------------------|----------------------------------|
| 01 Introduction to OOPs | 02 Real-world OOP example |
| 03 OOPs – Classes and Objects | 04 Inheritance in Python |
| 05 Encapsulation in Python | 06 Polymorphism in Python |
| 07 Python Modules | 08 Standard library |
| 09 Installing Packages | 10 Exception Handling |

Basic Principle of OOPS



What is object && class

What are Objects and Classes?

- 01** Object is the basic unit of object-oriented programming
- 02** An object represents a particular instance of a class
- 03** There can be more than one instance of an object
- 04** Each instance of an object can hold its own relevant data
- 05** Objects with similar properties and methods are grouped together to form a Class

In it method

__init__() method in Python

Example

```
class Student(object):  
    def __init__(self, name, branch, year):  
        self.name = name  
        self.branch = branch  
        self.year = year  
        print("A student object is created.")  
  
    def print_details(self):  
        print("Name:", self.name)  
        print("Branch:", self.branch)  
        print("Year:", self.year)  
  
ob1= Student( "Paul", "CSE", 2019)  
ob1.print_details()
```

```
A student object is created.  
Name: Paul  
Branch: CSE  
Year: 2019
```

- __init__ is a special method in Python classes is a constructor method for a class
- __init__ is called when ever an object of the class is constructed

```
>_
class Dog:
    def __init__(self, name):
        self.name = name

    def talk(self):
        print("Woof!")

    def printName(self):
        print("My name is: {}".format(self.name))

dog = Dog("Charlie")

dog.talk()
dog.printName()
```

INHERITANCE

Friday, August 19, 2022 12:00 PM

Types of inheritance

Different Types of Inheritance in Python

Single Inheritance

Multiple Inheritance

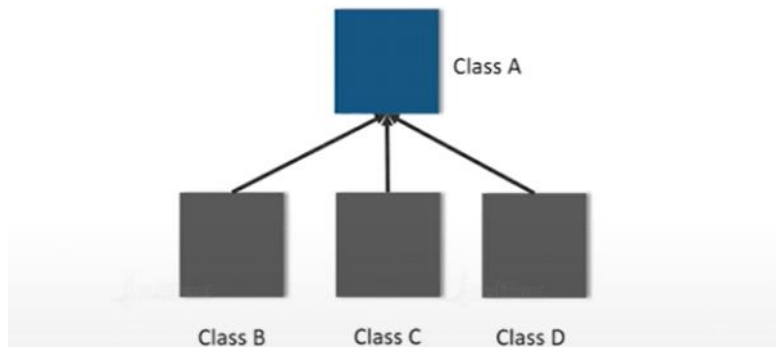
Multilevel Inheritance

Hierarchical Inheritance

Hybrid Inheritance

Hierarchical inheritance

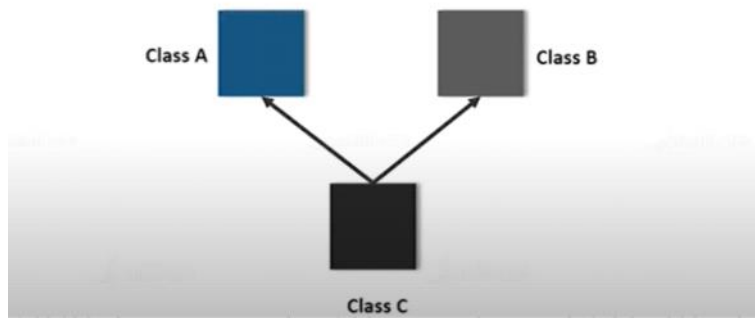
More than one class inherits from a class



Here a class is acquiring things from multi classes

Multiple inheritance

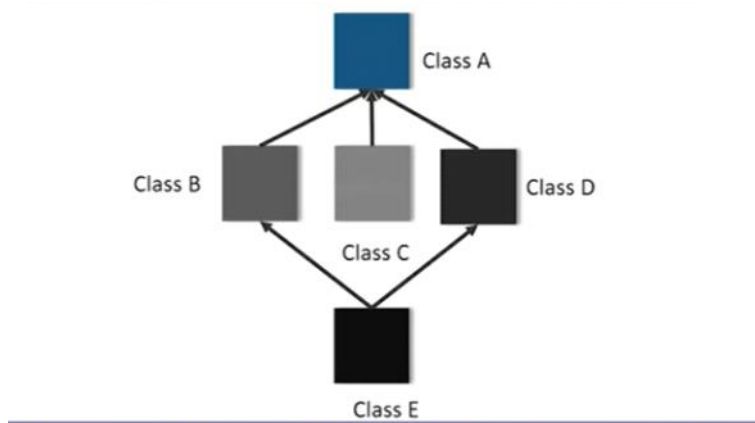
A class inherits from multiple classes



A single class is giving its things to many classes

Hybrid inheritance

Combination of any two kinds of inheritance



Here a class is giving to multiple classes and another class is taking things from multiple classes
A combination of
Multiple inheritance
Hierarchical inheritance

Encapsulation in Python

Saturday, August 20, 2022 11:04 AM

A

How to access a Private Method?

```
class Car:
    def __init__(self):
        self.__updateSoftware()

    def drive(self):
        print('driving')

    def __updateSoftware(self):
        print('updating software')

redcar = Car()
redcar.drive()
redcar._Car__updateSoftware()
```

Private method can be called using
redcar._Car__updateSoftware()

In python the private is describe by "__" two underscores but in it is A constructor not an private methods

How to access a Private Method?

To change the value of a private variable, a setter method is used

```
def setMaxSpeed(self,speed):
    self.__maxspeed = speed

redcar = Car()
redcar.drive()
redcar.__maxspeed = 10 # will not
change variable because its private
redcar.setMaxSpeed(320)
redcar.drive()
```

So as we all know the private methods cannot be accessed by the user

So we will call it by class as a setter in a particular syntax "_class name__private method name ()"

```
1 class SomeClass:
2     def public(self):
3         print("Public Function")
4
5     def __private(self):
6         print("Private Function")
7
8 obj = SomeClass()
9
10 obj.public()
11 obj._SomeClass__private()
12
```

Here we get the output

Public Function

Private Function

Modules in python

Saturday, August 20, 2022 11:32 AM

A

To put it simply, Module is a file containing python code



A module can define functions, classes and variables and can also include runnable code. There are pre-defined modules in python standard library

There two keywords which are important in the modules of python

from

import

Allows you to specify the things that you want to import from a module

Import command allows you to import the complete module

```
from utils import add, subtract
```

```
import os
```

Here they are majorly we will write a function in one file and by using from and import we can use that public method in another

File-1

```
def add(x, y):  
    return x + y  
  
def multiply(x, y):  
    return x * y
```

File-2

```
1 from utils import add, multiply  
2  
3 print(add(7, 9))
```

We can also import the 2 functions


```
from utils import add, multiply
print(add(7, 9))
print(multiply(7, 9))
```

Standard libraries in the Python

Python's standard library is very extensive and offers a wide range of facilities



The standard library contains built-in modules, that provides access functionalities such as I/O. All the data types, functions and modules that we learned so far are available because of the standard library.

A

Here by using libraries of OS I can print any file data in my console in python

```
1 import os
2
3 print(os.listdir('./data'))
```

INSTALLATION OF PACKAGES

Saturday, August 20, 2022 11:45 AM

A

To install colorama

Open cmd write "pip install colorama"

By this we can the change the color of output in the command prompt

```
from colorama import init
from termcolor import colored

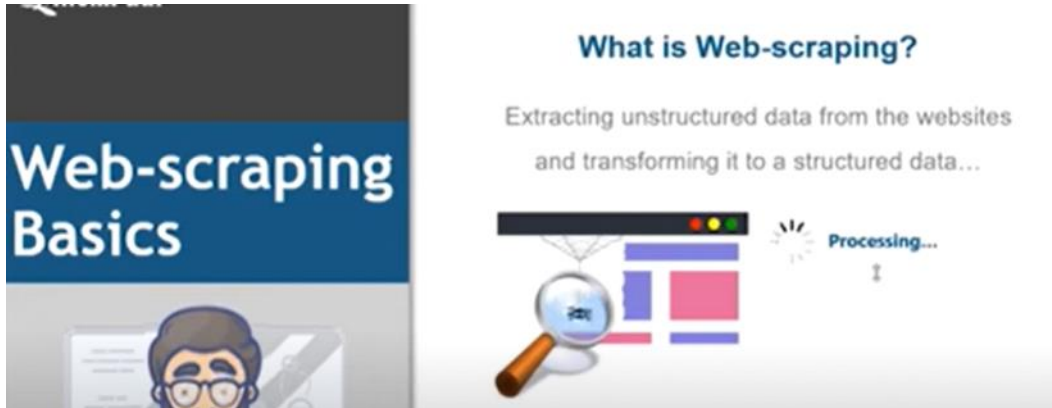
init()

print(colored("Hello, World!", "white", "on_red"))
```

Web scrapping

Saturday, August 20, 2022 11:58 AM

A



What is a web crawling and how does it work?

A web crawler works by **discovering URLs and reviewing and categorizing web pages**. Along the way, they find hyperlinks to other webpages and add them to the list of pages to crawl next. Web crawlers are smart and can determine the importance of each web page.

18-Feb-2022

What is Web Scraping?

Web scraping is an automatic method to obtain large amounts of data from websites. Most of this data is unstructured data in an HTML format which is then converted into structured data in a spreadsheet or a database so that it can be used in various applications. There are many different ways to perform web scraping to obtain data from websites. These include using online services, particular API's or even creating your code for web scraping from scratch. Many large websites, like Google, Twitter, Facebook, StackOverflow, etc. have API's that allow you to access their data in a structured format. This is the best option, but there are other sites that don't allow users to access large amounts of data in a structured form or they are simply not that technologically advanced. In that situation, it's best to use Web Scraping to scrape the website for data.

Web scraping requires two parts, namely the **crawler** and the **scraper**. The crawler is an artificial intelligence algorithm that browses the web to search for the particular data required by following the links across the internet. The scraper, on the other hand, is a specific tool created to extract data from the website. The design of the scraper can vary greatly according to the complexity and scope of the project so that it can quickly and accurately extract the data.