Python:

- created by Guido van Rossum
- released in 1991.
- Used for web development,software development,mathematics and scripting

Used for

- used on a server to create web applications.
- used alongside software to create workflows.
- connect to database systems. It can also read and modify files.
- used to handle big data and perform complex mathematics.
- used for rapid prototyping, or for production-ready software development.

Why to use it:

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.

Python Syntax:

```
if 5 > 2:
  print("Five is greater than two!")
```

```
#Five is greater than two!
```

Python Comments:

#print(“sarthak”)
print((“gupta”)
	"""
	This is a comment
	written in
	more than just one line
	"""
	print("Hello, World!")

>>gupta	>>Hello, World!

Python variables:

- Creating variables

```
x = 4       # x is of type int
x = "Sally" # x is now of type str
print(x)
```

>> Sally

- Casting variables

```
x = str(3)    # x will be '3'
y = int(3)    # y will be 3
z = float(3)
```

>>

- Get the type

```
x = 5
y = "John"
print(type(x))
print(type(y))
```

```
>>
```

- Single or double quotes

```
x = "John"
# is the same as
x = 'John'
```

```
>>
```

- Case Sensitive

```
a = 4
A = "Sally"
#A will not overwrite a
```

```
>>
```

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Camel Case

Each word, except the first, starts with a capital letter:

```
myVariableName = "John"
```

Pascal Case

Each word starts with a capital letter:

```
MyVariableName = "John"
```

Snake Case

Each word is separated by an underscore character:

```
my_variable_name = "John"
```

How we can assign variables:

- ```
  fruits = ["apple", "banana", "cherry"]
  x, y, z = fruits
  ```

Global variable and local variable

```
x = "awesome" #Global variable

def myfunc():
  x = "fantastic" #Local variable
  print("Python is " + x)

myfunc()

print("Python is " + x)
```

```
>>Python is Fantastic
  Python is awesome
```

Built in Data-Types:

Text_type-String
Numeric_type- int,float,complex
Boolean_type-bool
Sequence_type-list,tuple,range
Mapping_type-dict
Set_type-set
Binary_type- bytes,bytearray,memoryview
```

Binary Types:

bytes:

```
x = b"Hello"

#display x:
print(x)

#display the data type of x:
print(type(x))
```

```
>>b’hello’
>>bytes
```

Bytearray:

```
x = bytearray(5)

#display x:
print(x)

#display the data type of x:
print(type(x))
```

```
>>bytearray(b'\x00\x00\x00\x00\x00')
>> <class 'bytearray'>
```

Memoryview:

```
x = memoryview(bytes(5))

#display x:
print(x)

#display the data type of x:
print(type(x))
```

```
<memory at 0x01368FA0>
<class 'memoryview'>
```

Integer Types:

- Int: Whole,positive,negative,without decimals
- Unlimited length

```
x = 1
y = 35656222554887711
z = -3255522

print(type(x))
print(type(y))
print(type(z))
```

```
>> <class 'int'>
>> <class 'int'>
>> <class 'int'>
```

- Float: positive,negative
- Containing one or more decimals

```
x = 35e3
y = 12E4
z = -35.59

print(type(x))
print(type(y))
print(type(z))
```

```
<class 'float'>
<class float'>
<class 'float'>
```

- Complex:written as j as the imaginary part

```
x = 3+5j
y = 5j
z = -5j

print(type(x))
print(type(y))
print(type(z))
```

Random Number

- Function makes random no
- Built in module called random

```
import random

print(random.randrange(1, 10))
```

```
>> 7
```

Python Casting

- Specifying the type of variable
- int(),float(),str()

```
a = int(1)    # a will be 1
b = int(2.8) # b will be 2
c = int("3") # c will be 3


e = float(1)      # e will be 1.0
f= float(2.8)    # f will be 2.8
g = float("3")    # g will be 3.0
h = float("4.2") # h will be 4.2


x = str("s1") # x will be 's1'
y = str(2)    # y will be '2'
z = str(3.0)  # z will be '3.0'
```

Python string

- Surrounded by single quotation marks or double.
- And both are considered as same
- Use with three single or double quotes

```
 = '''Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.'''
print(a)
```

```
```

- Strings are Arrays of bytes representing unicode character
- element with single character inside quotes is considered as string
- To access elements use square brackets

```
a = "Hello, World!"
print(a[1])
```

```
>>e
```

- Looping through string

```
for x in "banana":
  print(x)
```

```
>>b
>>a
>>n
>>a
>>n
>>a
```

- Using len() to calculate length
- Using `in` or `not in` to check if any particular element is present or not

```
txt = "The best things in life are free!"
print("free" in txt)
```

```
>>True
```

- Slicing String:returning range of character

```
= "Hello, World!"

rint(b[2:5])
```

```
>>llo
```

- Slicing can be done with:

```
b = "Hello, World!"    #slicing to the end
print(b[2:])

c = "Hello, World!"    #slicing from the start
print(c[:5])

d= "Hello, World!"    #negative indexing
print(c[:5])
```

- Some important methods to remember

```
a= “Hello World!”

print(a.upper())

print(a.lower())

print(a.strip())

print(a.replace(“H”,”J”))

print(a.split(“,”))
```

Boolean Values:

- Many values evaluate to true which are not empty which gives values or meaning to something 1,[1,”sar”].
- Many values evaluate to false which are empty like (),[],{},0 and False.
-

```
def myFunction() :
  return True

if myFunction():
  print("YES!")
else:
  print("NO!")
```

YES!

- isinstance() used to determine if an object is of a certain data type:

```
x = 200
print(isinstance(x, int))
```

True

Python Operator:

- Arithmetic operator- + , - , * , / , % , **, //
- Assignment Operator- =, (-,*,/,%,//,**) +=,(&,|,^,>>)<<=
- Comparison Operator- ==,!=,>,<>=,<=
- Logical Operator- and, or, not
- Identity Operator- is, is not
- Membership operator- in, not in
- Bitwise operator- &, |, ^, ~, <<, >>

Iterable Objects:

1. List:
   - list(("apple", "banana", "cherry"))
   - Ordered
   - Changeable
   - Duplicate members allowed
2. Tuple:
   - tuple(("apple", "banana", "cherry"))
   - Ordered
   - Unchangeable
   - Duplicate values alllowed
3. Dictionary:
   - dict(("apple", "banana", "cherry"))
   - Unordered
   - Changeable
   - No duplicates
4. Set:
   - set(("apple", "banana", "cherry"))
   - Unordered
   - Unchangeable

- No duplicates

list()	dict()	tuple()	set()

Functions:

- Use def to declare the function block
- def funcname():
- To call outside the function we call funcname(andNumberOfArgumentToBeGiven)
-

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil", "Refsnes")
```


- You can pass list of parameters for strings only.
- *argumentsName in this way function will receive tuple of arguments
-

```
def my_function(*kids):
  print("The youngest child is " + kids[2])

my_function("Emil", "Tobias", "Linus")
```

The youngest child is Linus

- Using pass statement inside def will avoid getting error when function doesn't have any content
- Keyword arguments

Lambda Functions:

- Uses variable to save lambda function
- And that variable work as a function for example
-

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


- Another way to use lambda is to return lambda inside function and call

-

Arrays:

- Python does not have built-in support for Arrays, but Python Lists can be used instead.
- When more than one values is on the work the best way to work is with array
- As it can hold and perform many operation under single name as you want
- You can calculate length of array using len()
- Loop through an array use for...in…:
- Adding element inside an array is as simple as calculating length by using arrayName.append("anyThing")
- Use pop to delete any element otherwise remove can work for the same but difference is that it remove the first occurance arrayName.pop(ArrayIndex) or arrayName.remove("ElementName")
- Remember all these methods for array:
    - ❖ append()
    - ❖ clear()
    - ❖ copy()
    - ❖ count()
    - ❖ extend()
    - ❖ index()
    - ❖ insert()
    - ❖ pop()
    - ❖ remove()
    - ❖ reverse()
    - ❖ sort()

OOPS:

- Class
    - ❖ Blueprint of object
    - ❖ class parrot:
      Pass

- example

```
class Sarthak:
    def __init__(self,color,food,subject,webseries,game):
        self.color=color
        self.food=food
        self.subject=subject
        self.webseries=webseries
        self.game=game


sarkLike=Sarthak("blue","Chicken
Noodles","Maths","BigBangTheory","shadowFightArena")
sarkNotLike=Sarthak("red","Lauki ki
sabzi","HistoryCivics","Mirzapur2","FlappyBird")

print(sarkLike.color)
print(sarkNotLike.color)
```

```
blue
red
```

- Use keyword class to define empty class.
- Pass is used to define the class without contributing any type of functions.
- Using function format
-

```
print('{} {}'.format(sarkLike.color,sarkLike.food)
```

```
blue chicken noodles
```

- Attributes are the variable of the class shared between all the instances
- While printing attributes we dont use parentheses while calling for the class like above example

- Object
  - Object is used to describe/give what kind of features class have
  - For example class name parrot which have some methods which describe its features like color of feathers, shape,size ,weight,etc object call those function to give its features
  - In another words object is instantiation of class.
  - When class is defined only description of object is defined that means no memory is allocated.
  - Obj = parrot()
  -

```
class Sarthak:
    def __init__(self,color,food,subject,webseries,game):
        self.color=color
        self.food=food
        self.subject=subject
        self.webseries=webseries
        self.game=game

    def foodColor(self):
        return print('Like {} {}'.format(self.color,self.food))

sarkLike=Sarthak("blue","Chicken
Noodles","Maths","BigBangTheory","shadowFightArena")
sarkNotLike=Sarthak("red","Lauki ki
sabzi","HistoryCivics","Mirzapur2","FlappyBird")

sarkLike.foodColor()
Sarthak.foodColor(sarkLike)
```

```
Like blue Chicken Noodles
Like blue Chicken Noodles
```

- In "sarkLike.foodColor()" calling the method of the class using an instance
- While "Sarthak.foodColor(sarkLike)" calling the method of the class using a class.Well both will give same answers

- Methods:
  - Its a features of the class which is known as functions
  - Which defined the behaviour of the object
  - When we want to print the result inside the method we just remember that we have to use parentheses.
  - def sing():
        pass
    def dance():
        pass
- Inheritance:
  - it can easily be defined when there is an existence of more than one class
  - As one class contains or inherits the property of another class
  - Class which inherits the property known as child class
  - Class which have property and to be inherited by child class known as parent class or base class
  - class person:
        pass
    class student(person):

pass

- ❖ super().__init__(fname,lname)

- Encapsulation: we use _ or __ for privatizing variable
- Polymorphism: there is a class which have multiple existence

```
class Parrot:

    def fly(self):
        print("Parrot can fly")

    def swim(self):
        print("Parrot can't swim")
```

Exception Handling:
Try- this block let you test code for errors
Except-this block let you handle the error
Finally -this block lets you execute code.

```
try:
  print(x)
except:
  print("Something went wrong")
finally:
  print("The 'try except' is finished")
```

- Raise an exception can throw exception if condition occurs

```
x = -1

if x < 0:
  raise Exception("Sorry, no numbers below zero")
```

File Handling:

- In order to persist data forever we use file
- Python program can talk to the file by reading writing content

- Textfiles and binary files are two types of files
-

- Some methods to operate on file-
- open(filename,mode)
- read()
- readline()
- close()
- write()
- Import os
  os.remove("xyz.txt")
- os.rmdir("folder")
- r,a,w,x
- t,b

PEP8:

- PEP- Python Enhancement Proposal
- PEP8- set of style guidelines
- Rules are:-
  1. Regular_vairables
  2. REGULAR_VARIABLES
  3. Def random_regualar_variables
  4. class RegularVariable
  5. Factory functions
  6. self._instrument()
  7. If variable already exist use underscore at last a_=21

Method:

- Filter
- Map
- Format
- Any
- All
- xrange

Generator:

- Way for creating iterators
-

```
# A simple generator function
def my_gen():
    n = 1
    print('This is printed first')
    # Generator function contains yield statements
    yield n
```

```
    n += 1
    print('This is printed second')
    yield n

    n += 1
    print('This is printed at last')
    yield n


# Initialize the list
my_list = [1, 3, 6, 10]

# square each term using list comprehension
list_ = [x**2 for x in my_list]//here x**2 is what kind of
number is going to print. As i just write x instead of x**2 then
it will giv me value inside x otherwise square of x

# same thing can be done using a generator expression
# generator expressions are surrounded by parenthesis ()
generator = (x**2 for x in my_list)

print(list_)
print(generator)
```

-

Decorator:

- Part of program tries to modify another part of program
-

```
def first(msg):
   print(msg)
first("hello")
second=first
second("any")
```

-

Class method

- Class method which is bound to class not object
-

```
from datetime import date

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

    @classmethod
```

```
    def fromBirthYear(cls, name, birthYear):
        return cls(name, date.today().year - birthYear)

    def display(self):
        print(self.name + "'s age is: " + str(self.age))

person = Person('Adam', 19)
person.display()

person1 = Person.fromBirthYear('John',  1985)
person1.display()
```

```
Adam's age is: 19
John's age is: 31
```

-

Static method:

-

```
class Dates:
    def __init__(self, date):
        self.date = date

    def getDate(self):
        return self.date

    @staticmethod
    def toDashDate(date):
        return date.replace("/", "-")

date = Dates("15-12-2016")
dateFromDB = "15/12/2016"
dateWithDash = Dates.toDashDate(dateFromDB)

if(date.getDate() == dateWithDash):
    print("Equal")
else:
    print("Unequal")
```

-

Overloading:

- Same method name arguments are different

- Python not support method overloading

Overriding:

-

```
Class A:
  def show(self):
     print("in A show")
class B:
   Def show(self):
      print("in B show")

a1=B()
a1.show()//it overrides the property of A and take its own property and give output "in B show"
```

-

Types of error:

- Two types of error:
  Syntax error and logical error
- Syntax error or parsing error indicates where the parser run into the syntax error
-

Exception	Cause of Error
`AssertionError`	Raised when an `assert` statement fails.
`AttributeError`	Raised when attribute assignment or reference fails.
`EOFError`	Raised when the `input()` function hits end-of-file condition.
`FloatingPointError`	Raised when a floating point operation fails.
`GeneratorExit`	Raise when a generator's `close()` method is called.
`ImportError`	Raised when the imported module is not found.
`IndexError`	Raised when the index of a sequence is out of range.

`KeyError`	Raised when a key is not found in a dictionary.
`KeyboardInterrupt`	Raised when the user hits the interrupt key (`Ctrl+C` or `Delete`).
`MemoryError`	Raised when an operation runs out of memory.
`NameError`	Raised when a variable is not found in local or global scope.
`NotImplementedError`	Raised by abstract methods.
`OSError`	Raised when system operation causes system related error.
`OverflowError`	Raised when the result of an arithmetic operation is too large to be represented.
`ReferenceError`	Raised when a weak reference proxy is used to access a garbage collected referent.
`RuntimeError`	Raised when an error does not fall under any other category.
`StopIteration`	Raised by `next()` function to indicate that there is no further item to be returned by iterator.
`SyntaxError`	Raised by parser when syntax error is encountered.
`IndentationError`	Raised when there is incorrect indentation.
`TabError`	Raised when indentation consists of inconsistent tabs and spaces.
`SystemError`	Raised when interpreter detects

	internal error.
`SystemExit`	Raised by `sys.exit()` function.
`TypeError`	Raised when a function or operation is applied to an object of incorrect type.
`UnboundLocalError`	Raised when a reference is made to a local variable in a function or method, but no value has been bound to that variable.
`UnicodeError`	Raised when a Unicode-related encoding or decoding error occurs.
`UnicodeEncodeError`	Raised when a Unicode-related error occurs during encoding.
`UnicodeDecodeError`	Raised when a Unicode-related error occurs during decoding.
`UnicodeTranslateError`	Raised when a Unicode-related error occurs during translating.
`ValueError`	Raised when a function gets an argument of correct type but improper value.
`ZeroDivisionError`	Raised when the second operand of division or modulo operation is zero.

-

CRUD operation:

- CREATE , READ , UPLOAD , DELETE

List comprehension:

-

`newlist = []`
 	
`for x in fruits:`
`  if "a" in x:`
`  newlist.append(x)`
	`newlist = [x for x in fruits if "a" in x]`

For x in range(10):
  If x<5:
  newlist.append(x)
	newlist = [x for x in range(10) if x < 5]

-

Requests:

- Send https requests
-

```
import requests

x =
requests.get('https://w3schools.com/python/demopage.htm')

print(x.text)
```

-

Method	Description
delete(*url, args*)	Sends a DELETE request to the specified url
get(*url, params, args*)	Sends a GET request to the specified url
head(*url, args*)	Sends a HEAD request to the specified url
patch(*url*, *data, args*)	Sends a PATCH request to the specified url
post(*url, data, json, args*)	Sends a POST request to the specified url
put(*url*, *data, args*)	Sends a PUT request to the specified url

request(*method*, *url*, *args*)	Sends a request of the specified method to the specified url

-

pickling :

- *“Pickling”* is the process whereby a Python object hierarchy is converted into a byte stream.
- To store or preserve.
-

```
import pickle

cars=[“Audi”,”BMW”,”Maruti Suzuki”]
file=”mycar.pkl”
fileobj=open(file,’wb’)
pickle.dump(cars,fileobj)
fileobj.close()
```

-

unpickling

- *“unpickling”* is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy.
-

```
file=”mycar.pkl”
fileobj=open(file,’rb’)
mycar=pickle.load(fileobj)
print(type(mycar))
```

I can do your work in any price you want.