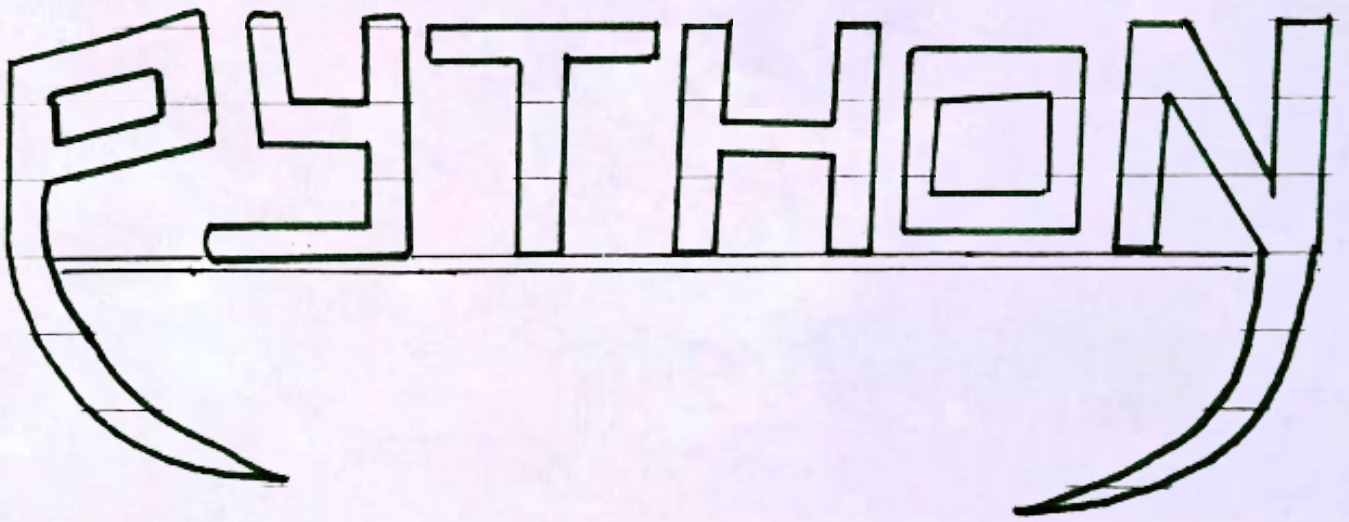
Complete notes on



Written By:-
Dipti Gaydhane
(Software Engineer)

INDEX

PYTHON TUTORIAL

11) python Lists

12) python Tuples

13) python sets

14) python Dictionaries

15) python If...else

16) python While Loops

17) python for Loops

18) python Functions

19) python Arrays

20) python Inheritance

21) python Iterators

22) python polymorphism

23) python Math

24) python JSON

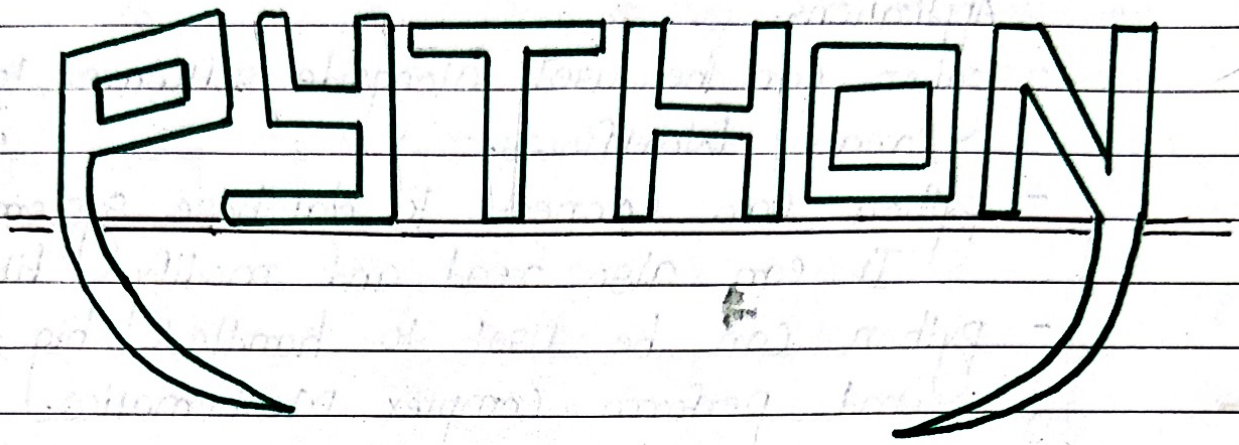
25) python File Operation

26) python Opening files

27) python Reading files

28) python Writing to files

29) python Closing files.



Python Introduction :

What is python ?

python is a popular programming language. It was created by Guido Van Rossum, And released in 1991.

It is used for :

- Web developement (Server-side),
- Software developement,
- Mathematics
- System Scripting

Why can python do ?

- Python can be used on a Server to create Web

Applications.

- Python can be used alongside software to create Workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex Mathematics.
- Python can be used for rapid prototyping, or for production-ready software developement.

Example:

```
print ("Hello, World")
```

Python Syntax :

Execute Python Syntax

Python Syntax can be executed by writing directly in the Command Line.

```
>>> print ("Hello, World!")
Hello World!
```

Or by creating a python file on the server, using the .py file extension, and running it in the command line.

```
C:\Users\your name>python myfile.py
```

Python Variables

In python, variables are created when you assign a value to it.

Example↓

```
x = 5
y = "Hello, World!"
```

Comments

Python has commenting capability for the purpose of in-code documentation.

Comments start with a #, and python will render the rest on the line as a comment.

Example 1

```
# This is a comment.
print ("Hello, World!")
```

Python Comments :

Comments can be used to explain python code.

Comments can be used to make the code more readable.

Comments can be used to prevent execution when testing code.

Creating a Comment

Comments starts with #, and python will ignore them.

Example 1

```
# This
is a comment
```

```
print ("Hello, World)
```

Multiple Comments :

Example ↓

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

Python Variables :

Creating Variables :

Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

Example ↓

```
x = 5
y = "John"
```

```
print (x)
print (y)
```

Casting :

If you want to specify the data type of a variable, this can done with casting.

Example ↓

```
x = str (3)       # x will be '3'
y = int (3)       # y will be 3
z = float (3)     # z will be 3.0
```

Get the Type

You can get the data type of a variable with the type() function.

Example ↓

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

print (type (y))

Single or Double Quotes ?

Example ↓

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

Case - Sensitive :

Example ↓

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

Python Data Types :

Built - in Data Types

In programming, data types is an important concept.

Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories.

Text Type.	-	str
Numeric Types	-	int, float, complex
Sequence Types	-	list, tuple, range
Mapping Type	-	dict
Set Types	-	set, frozenset
Boolean Type	-	bool
Binary Types	-	bytes, bytearray, memoryview
None Type	-	NoneType

Getting the Data Type :

Get the data type of any object by using the type() function.

Example ↓

```
print the
data type of
the variable x:

x = 5
print (type(x))
```

Setting the Data Type :

In python, the data types is set when you assign a value to a variable.

Example ↓

```
x = "Hello World"

x = 20
```

```
x = 20.5

x = 1j

x = ["apple", "banana", "cherry"]

x = ("apple", "banana", "cherry")

x = range(6)

x = {"name": John", "age": 36}

x = {"apple"; "banana", "cherry"}

x = frozenset({"apple", "banana", "cherry"})

x = True

x = b"Hello"

x = bytearray(5)

x = memoryview(bytes(5))

x = None
```

Python Numbers :

Python Numbers

There are three numeric types in python.

- int
- float
- complex

Example:

```
x = 1       # int
y = 2.8     # float
z = 1j      # complex
```

Int

Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

Example:

Integers.

```
X = 1
```

```
y = 35656222554887711
z = -3255522

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

Float

Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

Example:

Floats

```
x = 1.10
y = 1.0
z = -35.59

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

Complex

Complex numbers are written with a "j" as the imaginary part.

Example :

```
Complex.

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

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

Type Conversion

You can convert from one type to another with the int(), float(), and complex() methods.

Example :

Convert from one type to another.

```
x = 1       # int
y = 2.8     # float
z = 1j      # complex

# Convert From int to float:
    a = float(x)

# Convert from float to int:
    b = int(y)

# Convert from int to complex:
    c = complex(x)

print(a)
print(b)
print(c)

print(type(a))
print(type(b))
print(type(c))
```

Python Strings :

Strings

Strings in python are surrounded by either

Single quotation marks, or double quotation Marks.

'hello' is the same as "hello".

You can display a String literal with the print() function.

Example:

```
print ("Hello")
print ('Hello')
```

Python Modify Strings :

Python has set of built-in methods that you can use on strings.

Upper case

Example:

The
upper()
Method returns the string in upper case.

```
a = "Hello, World!"
print (a.upper())
```

Lower Case

Example:

The lower() method returns the string in lower case.

```
a = "Hello, World!"
print (a.lower())
```

Remove Whitespace

Whitespace is the space before and/or after the actual text, and very often you want to remove this space.

Example:

The strip() method removes any whitespace from the beginning or the end.

```
a = " Hello, World! "
Print (a.strip()) # returns "Hello, World!"
```

Replace String

Example:

The replace() Method replaces a String with another String.

```
a = "Hello, World!"
print(a.replace("H", "J"))
```

Split String

The split() Method returns a list where the text between the specified separator becomes the list items.

Example:

The split() Method splits the string into substrings if it finds instances of the seperator.

```
a = "Hello, World!"
print(a.split(","))  # returns
['Hello', 'World!']
```

Python - Format - Strings

String Format

Example ↓

```
age = 36
txt = "My name is John, I am " + age
print(txt)
```

String Methods

Method	Description
capitalize()	converts the first character to upper case
casefold()	converts string into lower case.
center()	Returns a centered string
count()	Returns the number of times a specified

value occurs in a string.

endsWith()	Returns true if the string ends with the specified value.
Find()	Searches the string value and returns the position of where it was found.
format()	Formats specified values in a string.
index()	Searches the string for a specified value and returns the position of where it was found.
isanum()	Returns True if all characters in the string are alphanumeric
isascii()	Returns True if all

	Characters in the string are ascii character
isdigit()	Returns True if all characters in the string are digits.
isspace()	Returns True if all characters in the string are whitespaces.
isupper()	Returns True if all characters in the string are upper case.
join()	Joins the elements of an iterable to the end of the string
ljust()	Returns a left justified version of the string
lower()	Converts A string into lower case
partition()	Returns a tuple where

the string is parted into three parts.

split()	Splits the strings at the specified separator, and returns a list.
strip()	Returns a trimmed version of the string.
upper()	Converts a string into upper case

Python Booleans :

Booleans represent one of two values: True or False.

Boolean Values

In programming Often need to know if an expression is True or False.

You can evaluate any expression in python, and get one of two answers, True or False.

When you compare two values, the expression is evaluated and python returns the Boolean answer.

Example ↓

```
print ( 10 > 9 )
print ( 10 = = 9 )
print ( 10 < 9 )
```

Python Operators :

Python Operators

Operators are used to perform operations on variables and values.

In the example below, we use the + operator to add together two values.

Example ↓

```
print (10 + 5)
```

Python - Arithmetic Operators :

Arithmetic operators are used with numeric values to perform common Mathematical operations.

Operator	Name	Example
+	Addition	x+y
-	Subtraction	x-y
*	Multiplication	x*y
/	Division	x/y
%	Modulus	x%y
**	Exponentiation	x**y
//	floor division	x//y

Python Assignment Operators :

Assignment Operators are used to assign values to variables.

Operator	Example	Same as
=	X = 5	X = 5
+=	X += 3	X = X + 3
-=	X -= 3	X = X - 3
*=	X *= 3	X = X * 3
/=	X /= 3	X = X / 3
%=	X %= 3	X = X % 3
//=	X //= 3	X = X // 3
**=	X **= 3	X = X ** 3
&=	X &= 3	X = X & 3
|=	X |= 3	X = X | 3
^=	X ^= 3	X = X ^ 3
>>=	X >>= 3	X = X >> 3

<<=	x<<=3	x = x<<3

Python - Comparison Operators :

Comparison Operators are used to compare two values.

Operator	NAME	Example
==	Equal	x==y
!=	Not equal	x!=y
>	Greater than	x>y
<	Less than	x<y
>=	Greater than or equal to	x>=y
<=	Less than or equal to	x<=y

Python - logical Operators :

Logical Operators are used to combine Conditional Statements.

Operator	Description	Example
and	Returns True if both statements are true	x<5 and x < 10
Or	Returns True if one of the statements is true	x<5 or x<4
not	Reverse the result, returns False if the result is true	not(x< 5 and x < 10)

Python - Identity Operators :

Identity Operators are used to compare the Objects, not if they are equal, but if

they are actually the same object, with the same memory location.

Operator	Description	Example
is	Returns True if both variables are the same Object	x is y
Is not	Returns True if both variables are not the same object	x is not y

Python - Membership Operators :

Membership operators are used to test if a sequence is presented in an object.

Operator	Description	Example
in	Returns True if a sequence	x in y

With the Specified Value is present in the project

not in	Returns True If a sequence With the Specified Value is not present in the Object.	x not in y

Python - Bitwise Operators :

Bitwise Operators are used to compare (binary) numbers

Operator	Name	Description	Example
&	AND	sets each bit to 1 if both bits are 1	x & y

|	OR	Sets each bit to 1 if one of two bits is 1	x|y
^	XOR	Sets each bit to 1 if only one of two bits is 1	x^y
~	NOT	Inverts all the bits	~x
<<	Zero Fill left Shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off	x<<2
>>	Signed right Shift	Shift right by pushing copies of the leftmost bit	x>>2

in from the
left, and let
the rightmost
bits fall off

Python Lists :

List :

Lists are used to store multiple items in a single variable.

Lists are one of 4 built-in data types in python used to store collections of data, the other 3 are Tuple, Set and Dictionary, all with different qualities and usage.

Example ↓

```
thislist = [ "apple" , "banana", "cherry" ]

print (thislist)
```

Python - Access List Items :

Access Items

List items are indexed and you can access them by referring to the index number.

Example ↓

```
print ( thislist [1] )
```

Change List Items

example ↓

```
thislist = [ "apple", "banana", "cherry", ]
thislist [1] = "blackcurrant"
  print ( thislist )
```

Add List Items

Append Items

Example ↓

```
thislist . append ( "orange" )
```

```
print (thislist)
```

Remove List Items

The remove() Method removes the specified item.

Example↓

```
thislist.remove("banana")
  print (thislist)
```

Loop Lists

Example↓

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

List Comprehension

Example↓

```
fruits
=["apple","banana","cherry","kiwi","mango"]
newlist = []
```

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

  print(newlist)
```

Python Lists

Example ↓

```
print ( thislist )
```

Sort Lists

Example ↓

```
thislist.sort()
   print (thislist)
```

Copy lists

Example ↓

```
mylist = thislist . copy ()
  print (mylist)
```

Join Lists

Example:

```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]

list3 = list1 + list2
print(list3)
```

List Methods

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value.

extend()	Add the elements of a list to the end of the current list
index()	Returns the index of the first element with the specified value.
insert()	Adds an element at the specified position.
pop()	Removes the element at the specified position.
remove()	Removes the item with the specified value.
reverse()	Reverses the order of the list
sort()	Sort the list

Python Tuples :

Tuple : Tuples are used to store multiple items in a single variable.

Tuple is one of 4 built-in data types in python used to store collections of data, the other 3 are List, Set and Dictionary. all with different qualities and usage.

A tuple is collection which is ordered and unchangeable.

Tuples are written with round brackets.

Example:

```
thistuple = ("apple", "banana", "cherry")
  print(thistuple)
```

Access Tuple Items:

Example:

```
thistuple = ("apple", "banana", "cherry")
  print(thistuple[1])
```

Update Tuples

Example → `x = ("apple", "banana", "cherry")`

```
y = list (x)
y[1] = "kiwi"
x = tuple (y)

print (x)
```

Unpack Tuples

Example↓

```
Fruits = ("apple", "banana", "cherry")
```

Loop Tuples

Loop through a Tuple

Example↓

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

Join Tuples

Example →
```
tuple1 = ("a", "b", "c")
tuple2 = (1, 2, 3)
```

```
tuple3 = tuple1 + tuple2
print (tuple3)
```

Tuple Methods

Method	Description
count()	Returns the number of times a specified value occurs in a tuple.
index()	Searches the tuple for a specified value and returns the position of where it was found.

Python Sets :

Set :

Sets are used to store multiple items in a single variables.

A set is a collection which is unordered, unchangeable,* and unindexed.

PAGE:
DATE: / /

Example ↓

```
print(thisset)
```

Access Set Items :

Access Items

Example ↓

```
thisset = {"apple", "banana", "cherry"}

for x in thisset
    print(x)
```

Add Set Items :

Add Items

Example ↓

```
thisset.add("orange")

print(thisset)
```

Remove Set Items

Example ↓

```
thisset.remove("banana")

  print(thisset)
```

Loop sets

Example ↓

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

Join Sets

Example ↓

```
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}

set3 = set1.union(set2)
  print(set3)
```

Set Methods

Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets.
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item.
intersection()	Returns a set, that is the intersection of two

Other Sets.

isdisioint()	Returns whether two sets have a intersection or not.
issubset()	Returns whether another set contains this set or not.
pop()	Removes an element from the element
union()	Return a set containing the union of sets.
update()	Update the set with the union of this set and others.

Python Dictionaries :

Dictionary

Dictionaries are used to store data values in key: value pairs.

A dictionary is a collection which is ordered*, changeable and do not allow duplicates.

Example↓

```
thisdict = {
  "brand": "Ford",
  "Model": "Mustang",
  "year": 1964
}
print (thisdict)
```

Access Dictionary Items

Example↓

```
thisdict = {
      "brand": "Ford",
```

```
  "model": "Mustang",
  "year" : 1964.
}
x = thisdict["model"]
```

Change Items

Example↓

```
thisdict = {
"brand" : "ford",
"Model" : "Mustang",
"year" : 1964
}
thisdict["year"] = 2018
```

Adding Items

Adding an item to the dictionary is done by using a new index key and assign a value to it.

Example↓

```
thisdic
t = {
```

```
  "brand": "Ford",
  "Model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
print(thisdict)
```

Remove Items

Example:

The pop() Method

Removes the items with the specified key name.

```
thisdict = {
  "brand": "Ford",
  "Model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
```

Dictionary Methods

Python has a set of built-in methods that

You can use on dictionaries.

Method	Discription
clear()	Removes all the elements from the dictionary.
copy()	Returns a copy of the dictionary.
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key.
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with

the specified key.

popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value.
update()	Updates the dictionary with the specified key-value pairs.
	Returns a list of all the values in the dictionary

Python If...Else

Python Conditions and If Statements

Python supports the usual logical conditions from Mathematics.

- Equals : a == b

- Not Equals : a != b
- Less than : a < b
- Less than or equal to : a <= b
- Greater than : a > b
- Greater than or equal to : a >= b

These conditions can be used in several ways, most commonly in "If Statements" and loops.

An "If Statement" is written by using the if keyword

Example ↓

```
a = 33
b = 200
If b > a:
  print("b is greater than a")
```

While Loops

Python Loops

Python has two primitive loop commands.

- While loops
- For loops

The While Loop

With the While loop we can execute a set of statements as long as a condition is true.

Example ↴

```
i = 1
while i < 6 :
    print (i)
i += 1
```

Python For Loops :

A For loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

Example:

```
Fruits = ["apple", "banana", "cherry"]
  for x in fruits:
    print (x)
```

Python Functions :

A function is a block of code which only runs when it called.

pass data, knowns as parameters, into a function.

A function can return data as a result.

Creating a Function

```
def
 my_function ():
 print ("Hello from a function")
```

Calling a function

Example ↓

```
def my_function():
  print("Hello from a function")

my_function()
```

Arguments

Arguments are specified after the function name, inside the parentheses. you can add as Many arguements as you want, Just Separate them with a comma.

The Following example has a function with one argument (Fname). When the function is called, We pass along a first name, which is used inside the function to print the full name.

Example ↓

```
def my_function(fname):
  print(fname + "Refsnes")
```

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

Returns Values

To Let a function return a value, use the return Statement

Example ↓

```
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

python Arrays :

What is Array ?
An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for Example), storing the cars in

Single variables could look like this.

```
car1 = "Ford"
car2 = "Volvo"
car3 = "BMW"
```

Access the Elements of An Array

Example 1

Get the value of the first array item.

```
x = cars[0]
```

Example 2

Modify the value of first Array item.

```
cars[0] = "Toyota"
```

Length of An Array

Use the len() Method to return the length of an array

Example → `x = len(cars)`

Looping Array Elements :

Use the for in loop to loop through all the elements of an Array.

Example ↓

Print each item in the cars array.

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

Adding Array Elements

Use the append() Method to add an element to an array.

Example ↓

Add one More element to the cars array:

```
cars.append("Honda")
```

Removing Array Elements

Use the pop() Method to Remove an element From the array.

Example↓

Delete the second element of the cars array:

```
cars.pop(1)
```

Example ↓

Delete the element that has the value "Volvo":

```
cars.remove("Volvo")
```

Python classess / Objects

Python is an object oriented programming language.

Almost everything in python is an object, with its properties and Methods.

A class is like an object constructor, or a "blueprint" for creating Objects.

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
```

Create Object

Now we 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)
```

The __init__() Function

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 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.

Create a Parent Class

Example:

```
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
```

Python Polymorphism

The word "polymorphism" means "many forms", and in programming it refers to methods/functions/operators with the same name that can be executed on many objects or classes.

Function polymorphism

An example of a python function that can be used on different objects is the len() function.

String

For Strings len() returns the number of characteristics.:

Example ↓

```
x = " Hello World!"
print (len (x))
```

Tuple

Example ↓

```
mytuple = ("apple", "banana", "cherry")
print (len (mytuple))
```

Dictionary

Dictionaries len() returns the number the value pairs in the dictionary.

Example ↓

```
thisdict = {
```

```
    "brand" : "Ford",
    "Model" : "Mustang",
    "year" : 1964
}

print(len(thisdict))
```

Local Scope

A variable created inside a function belongs to the local scope of that function, and can only be used inside that function.

Example ↓

```
def myfunc():
  x = 300
  print(x)

myfunc()
```

Global Scope

A variable created in the main body of the python

code is a global variable and belongs to the global scope.

Global variables are available from within any scope, global and local.

Example ↓

A variable created outside of a function is global and can be used by anyone.

```
x = 300

def myfunc ():
    print

myfunc ()

print (x)
```

Global keyword

If you need to create a global variable, but are stuck in the local scope, you can use the global keyword

The global keyword makes the variable global.

Example ↓

```
def myfunc():
  global x
  x = 300

myfunc()

print(x)
```

The Math Module

Python has also a built-in module called Math, which extends the lists of Mathematical functions.

To use it, you must import the Math Module.

```
import Math
```

The Math.sqrt() Method for example, returns the square root of a Number.

Example ↓

```
import Math
```

```
x = Math.sqrt(64)

print(x)
```

Examples:

```
x = Math.ceil(1.4)
y = Math.floor(1.4)

print(x) # return 2
print(y) returns 1
```

Python JSON

Python has a built-in package called json. Which can be used to work with JSON data.

Examples:

```
Import the json module.

import json
```

Parse JSON - Convert From JSON to python

Example ↓

```
import json

#Some JSON:
x = '{ "name": "John", "age": 30,
"City": "New York" }'

#parse x:
y = json.loads(x)

# the result is a python dictionary:
print(y["age"])
```

Convert From Python to JSON

Example ↓

```
Convert From python to JSON:

import json

# a python object (dict):
```

```
x = {
"name": "John",
"age": 30,
"city": "New York"
}

# convert into JSON
y = json.dumps(x)

# the result is a JSON string:
print(y)
```

Python Try Except

The try block lets you test a block of code for errors.

The except block lets you handle the error.

The else block lets you execute code when there is no error.

The finally block lets you execute code, regardless of the result of the try- and except blocks.

Exception Handling

When an error occurs, or exception as we call it, python will normally stop and generate an error message.

Example ↓

```
try:
f = open ("demofile.txt")
try:
 f.write ("Lorum Ipsum")
except:
    print ("Something went wrong when
writing to the file")
    finally:
     f.close ()
except:
print ("Something went wrong when
  opening the file")
```

Python File Operation

A file is a named location used for storing data. For example, Main.py is a file that is always used to store python code.

Opening files in python

In python, we need to open a file first to perform any operations on it - we use the open() function to do so. Let's took at an example.

Example ↓

```
file1 = open ("file1.txt")
```

Reading files in python

After we open a file, we use the read() Method to read its content.

Example ↓

```
# open a file in read Mode
file1 = open ("file1.txt")

# read this file content
read_content = file1.read()
print (read_content)
```

Writing to files in python

To write to a python file, we need to open it in write mode using the W parameter.

Example :-

```
# open the file2.txt in write mode
file2 = open ('file2.txt', 'w')

# Write contents to the file2.txt file
file.2write ('programming is fun.\n')
file2. write ( programiz for beginners\n')
```

Closing files in python

When we are done performing operations on the file, we need to close the file properly. We use the close() function to close a file in python.

Example :-

```
# open a file
file1 = open ("file1.txt", "r")
```

```
# read the file
read_content = file1.read()
print(read_content)

# close the file
file1.close()
```