# PYTHON

## what is Python ?

Python is high level object oriented programming language
it was created by gui gauido vs Ressum
first version was released in 1991- 0.90

## why to learn Python ?

1) Python has simple syntax
2) python is open source which means it is avaliable free of cost
3) Python is easy to learn
4) No compilation is needed in python
5) Python has powerful libraries like numpy pandas
6) Python is also used to create the web application

### * Application *

1) web application
2) app
3) Mathematics
4) Software

### * features *

1) easy to learn
2) easy to read
3) easy to maintain
4) easy to connect with database
5) Standard libraries like numpy, pands and matlib

! python -- version → To cheak version

### *Python indentation

(i) It refers to the space at the beggining of code.
(ii) uses indentation to indicate a block of code.

## * Comments *

1) Comments can be used to make your code more redable
2) comment can be used to explain the python code
3) it starts with "#" symbol.
4) multi line comment is given by """ """

* Python is Case Sensetive *

## * Keywords *

1) Keywords are reserved words
2) you can not use them as a constant or as any variable name
3) All the keywords are in lower case letters.

```
import keyword
print(keyword.kwlist)
```

## * Python Variables *

1) Python variables are the reserved memory location used to store the values with in python program.
2) This means when you create a variable at that time you reserved a space in memory.
3) Variable are containers for storing the data values.

## * Rules to create the Variable *

1) Variable name only contains alpha numeric characters & _ (underscore)

2) Variable name can not start with number or special character

3) we can not use the python reserved key words.

4) Variable names are case sensetive (name & Name both are diff)

5) Variable name can start with the alphabets or underscore (_)

type() → used to cheak datatype

x, y → 10, 20 (multiple variable)

x = y = z = 10 → one value for multiple variable

6) we can not use special character in variable name.

## * Data Types *

1) Numeric → int, float, complex (atbj form)

2) string

3) dictonary

4) list

5) Tuple

6) boolean

7) set

Type Conversion = Convert any type of data to other data type

Complex can not convert into int or float

But Vice Versa can happen

# * Operators *

1) Arithmatic
2) Assignment
3) comparision
4) Logical
5) Identity
6) Membership

① Arithmatic → = / → float
+ // → floar
− ** → pow
* % → reminder (mod)

② Assignment → =
+=
-=

③ comparision → == >=
!= <=
>
<

④ Logical → and
or
not

⑤ Identity → 1) is = cheak if they are identical
2) is not

⑥ Membership →
in = (If value found then true)
not in

## Mutable

If value has been assigned to a datatype and it can be changed then it is called mutable

① list
② set
③ dictonary

## Immutable

If value has been assigned to a datatype and it can not be changed then it is called Immutable

① tuple
② numbers (int, float, complex)
③ string

---

# * String *

String is a collection of character Immutable

- String are used to represent the textual data
- String is a sequence of character enclosed in either " " or ' '

## Indexing

1) forward (left to right) ⟶ 0 to (n-1)
2) Backward (right to left) ⟶ (-1) to (-n)

Space also take space in memory so it will also have index

Concatenation ⟶ string1 + string2

Repeatation ⟶ string1 * 1

## Slicing

name_of_string [start : end+1 : stepsize]

By default step size is 1

+1 step size you are traveling from left to Right

-1 step size you are traveling from right to left

## String method

1) len()

2) upper()

3) lower()

4) strip() → A) Lstrip, B) Rstrip } unwanted space will be removed

5) capitalize()

6) title

7) swapcase() → capital to small
small to capital

8) replace(previous, new) → replace any word

9) isalpha()
isnumeric()
isalpum()

10) .end with(), .start with()

11) split(way of spliting) → space quama (IMP)

10) Join → #, _, / ↑ method_of_Joining.Join(string name) } IMP

# * List *

1) commenly used data type
2) it is ordered mutable (It means modified and iterable sequence of elements)
3) list can be used to store the collection of items & these items can be of any data type.
4) list is defined in [ ] and the items are seprated by ,

list ⟶ ordered, Mutable, iterable

If a value inserting in a list have Multiple character then it must be given in list

fruit [1:] = "banana" ⟶ single character will be inserted
['b', 'a', 'n', '-', '-', 'a'] → like this

fruit [1:] = ["banana"] ⟶ ["banana"]

## * list method

1) .append() ⟶ add element to list at end
2) .extend() ⟶ add list of elements to list at end
3) .insert() ⟶ add element at a specific condition
4) .index() ⟶ index of first occuring
5) .count() ⟶ number of occurance of a value
6) .sorted() ⟶ sorted(name, reverse=true(dec))
7) .reverse() ⟶ reverse order.

8) **Pop** → (i) remove any item from list.

(ii) String_name.pop(index num)

(iii) If index number is not given then by default It always take the last index

9) **remove** → remove any item from list

| pop | remove |
|---|---|
| only take integer values as input | it takes integer as well as character values as input |

10) **max**(name of string) → (i) only works on same data type

(ii) it will not work on the list that has different data type

(iii) false = 0
true = 1
} In the max

# * Tuple *

Tuple is data type that represents ordered imutable sequence of elements.

(i) similar to the list

(ii) () this is used to create the tuple

(iii) tuples are an immutable data type that is there element can not be changed after they are generated.

(iv) each element in the tuple has a specific order that will never change because the tuple is order sequence.

| list | Tuple |
|---|---|
| once they are created they can be modified | once they are created they can not be modified. |

single element tuple ⟶ b = ("om", )

## * Concatination *

```
t1 = (1, 2, 3, 4, 5)
t2 = ('a', 'b', 'c')

print (t1 + t2)
```

## * Repetation *

```
t1 * t2
```

* There could be no changes can not made on the tuples *

## * count *

it will return the number of occurance of specific value

. count('thing you want to yount')

## * index *

return the index number of given element.

## * To make change in the tuple forcefully *

Then change the data type of the tuple in other data type & make changes & after that again change the data type into the tuple.

# * Dictionaries *

mutable

(i) dictionary is a data structure that allows you to store and retrive data in key value pair.

(ii) dictionary is also called as associate arroy in other programming language.

(iii) dictonary is defined using the { } curly braces.

(iv) dictonary consist of the key value pair

Syntax ⟶ `D1 = { key1 : value, key2 : value, - - - }`

keys are used to ascess the elements from the Dictonary

* To add new key value pair *

```
D1 ["key nam"] = "value"
```

## * Dictonary Methods *

### 1) Get

returns the value for specified key.

it allows you to provide a default value if the key is not present.

```
g = stud.get ("gender")
```

g
⟶ Male

```
h = stud.get("location", "not provided")
```

↳ not present in the dictonary.

↳ default set.

h
⟶ not provided

2) keys()

gives the keys name present in the dictionary.

```
stud.keys()
```

3) values()

gives the values present in the dictionary

```
stud.values()
```

4) Items()

gives the Both keys and value pair

```
stud.items()
```

5) Pop Items()

used to remove the last last element from the dictonary

```
stud.popitems()
```

6) Pop() *

removes specific key value pair from the dictonary

only take key as a input

```
stud.pop("Name")
```

7) del *

delete specific element from dictonary

```
del stud["marks"]
```

8) Update *

Updates the dictonary with elements from other dictonary

```
new_data = {"age": 30, "grade": 23}

stud.update(new_data)
```

# * SET *

mutable

Set is a collection of unique and well defined elements.

(i) set is created using the {}
(ii) set is unordered
(iii) set is mutable
(iv) set do not allows duplicate values.

a = set() ⟶ empty set

## * set methods *

1) add()

```
set_name.add("value")
```

2) update()

```
set_name.update("value")
```

3) remove()

```
set_name.remove(name of item to remove)
```

4) len()

```
len(set name)
```

5) Union()

all distinct elements from the given list

```
set1.union(set2)
```

6) intersection

Common elements from sets

set1. intersection (set2)

7) difference

elements which are not common are shown

set1. difference (set2)

## * INPUT *

(i) To take a value from user the Input is used

(ii) Result of Input is always in string

(iii) It can be type converted (type conversion can be done)

# * CONTROL FLOW STATEMENTS *

## 1) If

```
If <condition> :
        <do this>
```

```
If marks > 70 :
        print ("In loop")
print ("not in loop")
```

It always print if loop work or not Because it is outside of loop

## 2) If Else

```
if (condition) :
        print (if block)

else :
        print (else block)
```

## 3) If - Elif - Else

```
if (condition) :
        print (if block)

elif (condition 2) :
        print (elif block)

else :
        print (else block)
```

## 4) Nested if

```
if (condition) :
    if (condition) :
        print (nested if)
    else :
        print (nested else)

else :
    print (else)
```

# * loop *

① for loop

② while loop

① for loop → for loop is used to iterate items from a sequence such as list, tuple, string.

```
for (variable) in (List/Tuple/string/Range):
    print(string)
```

Use for loop when you know exactly how many times you want to execute the loop

```
f = ["banana", "mango", "apple"]
```

① 
```
for i in f:
    print(f)
```
} when we want to get elements form a given list

② 
```
for i in range(2,6):
    print(i)
```
} numbers will be print from given range which is 2 to 6

[start, end(n-1)]

output will be → 2 3 4 5

③ 
```
for i in range(0, 11, 2):
    print(i)
```
} [start, end(n-1), steps]

even, odd could be found by this

④ `for i in range(0,4):` / `print(f[i])` } specified index needs to be found from given list at that time used

`f[0], f[1], f[2], f[3]` ⟶ These index will be iterated.

② **while loop** ⟶ While loop is used to repeatedly execute a block of code as long as a certain condition is true.

```
i = 1                          ⟶ from where to start
while (i<=10):                 ⟶ upto what the body must execute
    print(i)
    i+=1                       } Body
```

Use while loop when you don't know how many times you need to execute the loop.

```
name = "OM"

i = 0
while(i < len(name)):
    print(name[i])             } extract characters from a string at that time it is used.
    i+=1
```

# * Break, Continue, Pass *

## ① Break

(i) Used at a time were need to stop when some specific condition is met.

(ii) alway return after the print statement.

## ② Continue

(i) All loop will work excluding the condition value

(ii) writen before print statement

(iii) stop only for particular condition

```
for i in range(1,11):
    if i==7:
        continue
    print(i)
```

```
i=1
while (i<=10):
    if i==7:
        i=i+1
        continue
    print(i)
    i=i+1
```

→ for all next number it will cheak if they are equal to 7

## ③ Pass

(i) work as a place holder for your feture result

(ii) when we don't know the result then we will pass to that

(iii) when pass statement is executed then nothing will happens

empty code is not allowed in python and to avoid this we will use the pass statement.

# * List Comprehension *

It is shortcut for creating list
write for loop in reverse order

**normal**

```
for i in l1 :→(step 2)
    print(i**i)
          ↳result
          (step 1)
```

**list comprehension**

```
x = [i**i for i in l1]
```

} square

```
for i in L1 :— step2
    if i%2 == 0 : step3
        print(i)
              ↳result
              (step 1)
```

```
y = [i for i in L1 if i%2 == 0]
```

} even no

* Write for loop in Reverse Order *

⇓

LIST COMPREHENSION

# * function *

(i) when one action need to be performed many times at these situation we will use function

(ii) This will eliminate the process to write the code multiple time.

(iii) function is a block of code that only runs when it is called.

Two Types in function → ① buit in functions
② user define function

Syntax:

```
def function_name (parameter) :
        fun(body)
        return result
```

* return = sends the result back to the caller
only inside of the function works.

* Argument → Information we can pass inside the function (Parameter)

1) default Argument
2) keyword Argument

# * lambda function *

(i) It is a small anonymous function

(ii) lambda key word is used to create lambda function

(iii) lambda function can have any number of argument

(iv) But there will be only one expression in lambda function

(v) this is a function with out any name.

Syntax :-

```
lambda arguments : expression
       ↓                ↳
   any number          only
                       one
```

```
Cube = lambda x : x **3
```

↳ This will store the lambda fun bcz it don't Have any name.

## * Condition In lambda function (If Else) *

```
lambda argument : (If result) If condition else condition (else result)
```

```
result = lambda x : f"{x} is even" if x%2==0 else "odd"
                    ↓               ↳ If        ↳
                  If result         condition   else result.
```

## * filter * गाळणी

Used to add conditions in lambda function / normal fun

Return in true & false

goes one by one to the value.

```
marks = [77, 97, 64, 35, 55]

def fail(score) :
    return score<60

result = filter (fai, marks)

print ( list (result))
```

filter in normal function

```
L = [1, 2, 3, 5, 12, 16]

even = filter (lambda n : n%.2 ==0, L)

odd = filter(lambda n : n%.2 !=0, L)

for i in even :
    print(i)
```

filter in lambda function

Syntax →

```
filter ( function_name (or lambda), iterable list)
```

# * Map *

Syntax →

```
map(function, iterable)
```

apply function to All the element of a sequence

## Map in Normal function

```
t = (2,1,6,8,12,19,25)

def mult(n):
    return n*5

y = map(mult, t)

list(y)
```

## Map in lambda function

```
t = (2,1,6,4,8,10,15)

x = lambda n : n*5

y = map(x, t)

list(y)
```

## Map VS filter

| Map | filter |
|---|---|
| Apply function on every item of an iterable | create a new iterable which will satisfy the condition |
| for loop | IF Else |

## * ZIP *

Take elements index wise and merge them together

name = ["raj", "raj25", "gaurav"]

roll_no = [111, 222, 333]

raj 111
raj25 222
gaurav 333

- like this, It will be merge Index wised.

~~zip~~ zip(name, roll_no)

```
zip(iterator1, iterator2, iterator3)
```

## * Recursive function *

Call the function inside the function

limit of Recursion is upto 1000 times →

```
import sys
print(sys.getrecursionlimit())
```

We can increase it by →

```
import sys
sys.setrecursionlimit(No_you_want)
```

```
def factorial(n):
    if (n==1 or n==1):
        return 1
    else:
        return n * factorial(n-1)
```

} factorial of any Number

↳ called within function.

factorial of 7 → 7! = 7X6X5X4X3X2X1

logic →

7 X 6! ⟹ n * factorial(n-1)

factorial (4)

4 * factorial(3) — 1

4 X 3 X factorial(2) — 2

4 X3 X 2 X factorial(1) — 3

4X3X2X 1 — 4

This will goes in if block and return 1

Output → 24

## * local Variable *

we can only use these within the single cell or single function

```
def add(b):
        a=10 ⟶ local variable.
        return  a+b
```

## * global Variable *

we can use this in all over the notebook

```
a=100 ⟶ global Variable
def add(b):
        return a+b
```

# Object Oriented Programming

2 main entity → class
→ object.

## * Class *

(i) class is a real word entity.
(classification of real world object)

(ii) Every data type in python is in class

(iii) everything writen in python is in the format of class object.

(iv) class is the blueprint of the code.

## * Object *

(i) instance(variable) of a class is object.

(ii) we can create any number of object for a class.

## *Rules To create objects (function) in A class *

1) While creating function inside the class we have to pass first parameter as self.

2) If we don't pass self as first parameter then it will give us error.

3) Self is not a reserved keyword.

4) we can write anything on the position of self.

5) Self is the first parameter of the function.

```
class skills() :          → Name of class
    def welcome(self) :   → function of class, first parameter for object
        print("Hello to the skills")
```

```
b = skills()
```

object of class

`b.welcome()` → object is calling the class function.

O/P → Hello to the skills.

`dir(object Name)` → This will provide you all the function name present in the class.

Self → instance of the class

Tells the difference between class variable & method (fun) variable

If it is not given in method (fun) then error will occurs.

`__init__`

it act as a constructor

constructor is basically used in orderd to initilized how many number of property getting used inside a particular unit.

(Tells the no of para to class)

provide data to class

Class only understand self associated words

```
Class car():
    def __init__(self, name, window, door)
        self.name = name
        self.window = window
        self.door = door
    def display(self):
        print(f"name of the car {self.name}")
        print(f"_____ {self.door}")
        print(f"_____ {self.window}")
```

diff parameter of class

Name store in this

} initilization of parameter.

} method 1 display

```
b = car("BMW", 4, 4)
b.display()
```

→ object of class & with parameter given by class

# * Abstraction *

It is used to hide the internal functionality of the function from the user.

Need → ① reduces the complexity of code
② no object can be create for abstraction class.

work like
Encapsulation → only want part is shown
ascess of some part of code is restricted

python does not provide abstract class

for that we are using

from abc import ABC

ABC → Abstract Base Classes.
↑ full form

```
from abc import ABC

class shape (ABC) :
    @abstractmethod
    def area (self) : pass

#   @abstractmethod
    def ~~area~~ (self) : Pass
        perimiter

class square (shape) :
    def __init__ (self, side) :
        self.side = side

    def area (self) :
        area = self.side ** 2
        return area
```

It is a decorator → must implement once given, then no need to give every position

abstraction class

If abstractmethod given then all the method of the abstraction class need to add in it's subclass other wise error

class created on Half of shape.

① abstract method → If it is given then that method of abstract class need to be add into the subclass other wise it will generate error

It is a type of decorator.

advantage → 1) simplyfy complex system

2) Hide Unnessary data

3) shows only relevant data

② → तुमच्या code मध्ये abstraction class call hote तेव्हा ती class देर नाही व्हायला पाहिजे

## Destructor:

1) Destructor are called when an object get destroyed.

2) It is not that much Usefull

Because python has garbage collector that Handle memory management.

| |
|---|
| `__del__()` → destructor |
| `__init__()` → constructor |

```
class test():
    def __init__(self):
        print("data provided")

    def __del__(self):
        print("delocation done")
```

t = test()

del t → object will be deleted.

## * Inheritance *

One class takes properties from another class.

1) parent class → the original class
from that we are creating new class.

2) child class → class created with the help of parent class
which inherites the properties.

① single inheritance
② Multiple inheritance
③ Multi level inheritance
④ Hieractical inheritance
⑤ Hybrid inheritance.

1) Single Inheritance → 1) child class is created with the help of only one parent.

2) The child inheritaces properties only from one parent.

```
class employee():
    def __init__(self, emp_no, e_name):
        self.emp_no = emp_no
        self.e_name = e_name
```

} parent class.

```
def display(self):
    print(self.name, self.emp_no)

class manager(employee):
    def __init__(self, emp_no, e_name, role):
        employee.__init__(self, emp_no, e_name)
        self.role = role
    def display(self):
        employee.display(self)
        print(self.role)
```

super can be also written

→ the parameters from parent class

child class

function from parent class

```
emp2 = manager(2, "omkar", "manage")
```

emp2.display() ⟶ child class calling it's fun^n

* emp2.diplay() ⟶ this can not be called

child can not call the method of parent class.

2) Multiple ⟶ child class inherites properties from more than one parent class.

```
class A
class B
class C(A,B)
```

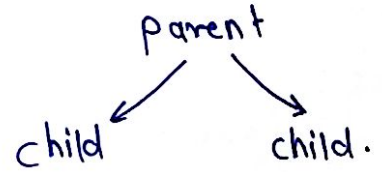
```
class boy (ddd, mom):
    def __init__ (self, d_name, m_name, s_name):
        dad.__init__ (self, d_name)        # parent 1 variable
        mom.__init__ (self, m_name)        # parent2 variable
        self.s_name = s_name

    def child_name (self):
        dad.D_name (self)                  # parent 1 method
        mom.M_name (self)                  # parent 2 method
        print (self.S_name)
```

child which has heritated two parents property

3) **Multi level** → There will be one parent from that one child is created.
and from that child also another p child is created.

parent → child → child

4) **Hieractical** → single parent & that parent had multiple child.




5) **Hybrid** → combination of any type of inheritance.

```
def __init__ (self, Part_class_variable, child_class_variable):
        parent_class_name.__init__ (Names of variable)
    self.child_class_name = name

def fun :
        parent_class_name.method()
        childclass new method.
```

# Polymorphism

# * NUMPY *

Numpy is used to perform all operations for array
we can only store number inside the array

| list | array |
|---|---|
| (i) data is seprated by (,) | (i) data is not seperated by (,) |
| (ii) list can not manage Arithmatic operation | (ii) array can manage arithmetic operation. |
| (iii) Consume large memory | (iii) does not consume large memory |

import Numpy as np.

## 1) ndim

(number of Rows, Number of column)

```
array = np.array([[[1,2],[3,1],[5,6]]])
```

No of brackets will decide the dimension

↳ 3 dimensional.

```
array.ndim
```

O/P ⟶ 3

2) arange → just like range

create array with given range.

That array is always one dimensional.

| np.arange (start, end+1) |
|---|

np.arange (1, 101)

O/P → 1 to 100

3) linespace → 1) random number from given range of number

2) divides data points in same interval as given

3) (1, 101, 20)

1 to 101 numbers will be get divide into 20 equal parts and that ~~num~~ 20 numbers will be given to us.

| np.linespace (start, end, count of random number) |
|---|

np.linespace (1, 101, 20)

O/P → 20 same parts from 1 to 101

4) Reshape → change the shape from 1 dimension to 2 dimension or multiple dimension.

(dimension count, No of ele in every dim)

x is a array which hold 10 number.

x.reshape(5, 2)
- 5 → rows
- 2 → col

array name . reshape( shape , order)

order = f = col wise
A
c = row wise
↳ c is default set

shape = row x col

10 data pt in array

now shape should be

(rows x col) = 10 → always
↳ inside the formula.

shape (row, col)
- row → This will Hold the No of rows.
- col → This will hold the No of col

5) Resize → 1) Change the shape of array in inany form but
2) if needed then extra elements will be added inside the array.
3) sometime blank space are remained so that spaces will be field with (sequence, mode)

```
b = np.arange(15)

b.resize(shape)
```

reshape to convert in 1 dimensional will not work

```
array-name(1, total_No_of_element)
```

=list given then convert that list in any dimensional array=

ndmin* l = list from 0 to 15 IMP

```
L = range(15)
n_arr = np.array(L, ndmin = 4)
```

→ 4 dimensional array will be created Because of ndmin

---

6 one dimensional to 3 dimensional by reshape IMP

→

```
a.reshape(2,3,(-1))   -1 dim = 6 elements present
```

4 dimensional

```
a.reshape(2,2,2,(2))   +2 dim = 16 element present.
```

---

6) Ravel → Convert any dimension array into one dimension array.

a = 3 dimensional array

a.Ravel() a.ravel()

7) flatten → Convert any dimension array into one dimensional array.

a.flatten()

8) Transpose → row to col, col to row } conversion.

n.transpose()

9) add → add the elements of two array

first element of array A + first element of Array B } like this all the addition Happen.

ap bp

1) $\begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} + \begin{bmatrix} 7 & 10 \\ 8 & 11 \\ 9 & 12 \end{bmatrix} = \begin{bmatrix} 8 & 14 \\ 10 & 15 \\ 12 & 18 \end{bmatrix}$ np.add(ap, bp)

np.add( )

2) [[2,3], [3,2]] + [[2,5], [6,8]] → [4,8] [9,10]

ap bp

np.add(ap,bp)

10) Min → find the minimum number from the array.

default axis is 0 (row)

If same no are present in the dataset then first Number will be taken

```
array_name.min()
```

1 = column ↑ ↓

0 = row → ←

11) MAX ⟶ find the maximum from the array

array_name.max()

12) Argmax ⟶ Return the index of max number from the array.

array_name.argmax()

13) Argmin ⟶ Return the index of min number from the array

array_name.argmin()

index
0 = [1, 5]
1 = [8, 96]
2 = [5, 44]
3 = [9, 54]

argmax(axis=1) ⟶ same for argmin

o/p = 3, 1

from 1st col

from 2nd col

axis=1 ↙ col

14) Slicing

a = [1, 2, 3, 6, 7, 9, 12, 15]
(1 = 2, 5 = 9)

a[start : end : steps]

a[1:5] ⟶ [2 3 6 7 9]

a[3:5] ⟶ [6, 7, 9]

-1 ⟶ Right to left

1 ⟶ left to Right.

15) mean → array_name.mean()

16) median → array_name.median()

17) Quartiles →

```
q1 = np.quantile(array_name, 0.25)   → 25 percentile
```

keyword

```
q2 = np.quantile(array_name, 0.50)   → 50 percentile.
q3 = np.quantile(array_name, 0.75)
```

$IQR = q_3 - q_1$

lower range = $q_1 - 1.5 * IQR$

Higher range = $q_3 + 1.5 * IQR$

18) Boolean Indexing

1) values grater than 15

```
array_name[array_name<15]
```

2) even numbers

```
array_name[array_name % 2 == 0]
```

## 19) Math function

$\pi = 180$

$\pi/2 = 90$

$\pi/4 = 45$

$\pi/6 = 30$

```
np.pi = 180°
np.pi = π
```

# * Pandas *

Most popular software library used for data manipulation

Data in pandas is of two types ⟶ 1) series
2) dataframe.

1) series ⟶ one dimensional. [ ] ⟶ series

Series_data = pd.series ( ____ )

2) Data frame ⟶ Two dimensional [[ ]] ⟶ dataframe

df = pd.dataframe ( __ )

In this the structure will be in the form of rows & col

* Give name to rows and col in ~~data frame~~

pd.dataframe ( data, index = [ ____ ] ) ⟶ for row naming

pd.dataframe ( data, columns = [ ____ ] ) ⟶ for columns

columns ⟶ columns
rows ⟶ index
} Both can be give at a time

* Convert dictonary into dataframe

```
data = {"car" : ["bmw", "rols-royal", "honda"],
        "year" : [3, 4, 2]}

df1 = Pd.Dataframe (data)
```

1) info → cheaks the information of dataframe.

df1.info()

All information is provided.

2) Head → display the first 5 record from start
default 5 is set can be change.

df.head()

3) Tail → displays the last 5 records from end

df.tail()

4) to_string → To print every thing in string format

df.to_string()

5) describe → stats analysis data shown

df.describe()

6) isnull() → gives output is that cell is empty or not
output is in form of true/false

df.isnull() → true/false

df.isnull().sum() → count of null values.

7) dropna → Remove the null values from the data.

inplace

df.drop()

8) fillna → If large part of data has NA values so we can not drop that so we will fill that

```
df.fillna("__")
```

9) Column → All column names will be shown

```
df.column()
```

16) Duplicate → gives the output is that cell has duplicate value or not
output is in form of true false

```
df.duplicate() → true false table
df.duplicate().sum() → count of duplicate value.
```

17) drop duplicates → Delet the duplicate values from the data.

inplace

```
df.drop_duplicates()
```

18) reset index → Add another index column to the data.

```
df.reset_index()
```

19) Drop → drop the values from data
default axis is 0
axis = 1 — column
axis = 0 — row

```
df.drop("col_name", axis=1, inplace=True)
df = df.drop("col_name", axis=1)
```

} To drop column from data.

```
df.drop(1)
```

→ To drop Row from data.

companies whoes rank is below 10 drop that.

df[df.rank < 10] ⟶ All rank will be found

df.drop(df[df[rank <10] == 10], axis = 0)

20) value count ⟶ show each categories strength
if male, female then it will show the No of male & female

df[col-name].value_counts()

| dataframe ⟶ array<br>Pands ⟶ Numpy | array ⟶ dataframe<br>Numpy ⟶ Pandas |
|---|---|
| array = df.to_numpy() | df = pd.Dataframe(array, col=, index= |

21) Unique ⟶ show all the unique values present in a column

df["col name"].unique()

# File Handling

use to perform wide range of operation on different files.

There are mainly two types of files are used in python in built file Handling.

1) Binary files
2) text files.

## * Modes *

r = only reading mode

r+ = only read & write mode → r+ means → read & something

w = Only write mode

w+ = write & read mode → w+ means write & something

a = append only → append at last

a+ = append and read → appeand & something → read

## * To create new file *

```
file = open("myfile.txt", "x")
```

- file: Here this New file is stored
- "myfile.txt": Name of that file
- "x": extension

f.close() → Always perform this action
If not performed then change won't get stored.

(r) read only → This mode open the file in reading mode only & if file is not present/exist in our folder then error occurs.

(w) write only → 1) This mode open file in the writing mode only and data in existing file are modified & overwritten
2) if file is not present in the folder then new file is created of that name.

(r+) read & write → This mode open the file in read & write mode & it raise error if the file does not exist in the folder.

(w+) write & read → 1) This mode open the file in write & read mode
2) The data is modified and overied if file present
3) if file is not exist then the new file will be created.

(a) append only → 1) This will open the file in the writing mode
2) if file does not exist then new file is created.
3) The new data is added at the end position of the file.

(a+) append & read → 1) This will open the file in the writing & reading mode
2) file not exist then new is created.
3) Data is added at the end of file.

## * read existing file *

```
file = open ("file.txt", "r")

file.read ()
```

## * write in a file *

```
file = open("file.txt", "w")        # w+

file.write (" _____")

file.close ()
```

To read that data use read operation.

## * append operation on file *

```
file = open ("file.txt", "a")
file.write ("_____")
file.close ()
```

→ save the changes.

## * Writeline *

Write multiple lines in the file at a time

```
file.writelines(["my name__", "\n I am doing__ "])

file.close ()
```

↑ To write multipleline at a time

→ To print on new line

writeline hold → list of multiple line.

readlines → read all the lines at a single go & then return them as each line as string element in a list.

(All)

(IMP)

```
with open("om.txt", "r") as f:
    f.readlines()
```

read(n) → read character upto n number of index

```
f.read(5)
```

for binary file all the modes are same just add b at end

for Normal (r) → (rb) for binary } like this

To delet a file

```
import os
os.remove(____)
```

} parmentaly delet the file.

Tell → tells the position of the cursor
by default the position will be 0

```
f.tell()
```

file Name

**seek** → Take the cursor at n position

f.seek(_), from this no of index reading will start

forward & Backward can travel.

position specified in byte.

**truncate** → only given n size will allocate to the file to store data.

f.truncate(_) → This size will decide space of file.

**readline** → write the first line from the file

f.readline()

(IMP)