

# python note plus related wox

## ① Installation — python.org

compiler — Engine that understands and runs your code.

IDE — Integrated Development Environment

→ software/tools where you write, edit, test, run and debug code.

→ Eg:- VS code,

Jupyter notebook,

PyCharm,

spyder

## ② dynamic and static type

Dynamic } - variable can change its type anytime.

{ - language decide the type automatically.

i.e.  $x = 10$  int

$x = "hello"$  string

- IS decide at runtime

- Dynamic language are :-

python

Ruby

JS.

Static } Must tell the language the type and it cannot change later.

Eg: int  $x = 10$  ✓

$x = "hello"$  ✗

- C, C++, Java, Go are static type

### Comments.

- Used to ignored by the py interpreter.
- single line ( # )
- Multi line comment are not available in py  
but we can used

DOC string  
[ " " " , " " ]

### Variables in py :-

(1) containers used to store data.

eg:- → value stored

$x = 10$

↳ variable

i.e.

Variable is a place where you keep data  
in a program -

You can write variable in py using 3 ways:-

I) camel case

→ myNameHome = "She - - - "

II) pascal case ( OOP )

→ MyNameHome = " - - - "

III) Snake case

→ my\_name\_home = " - - - "

Types of variables

① Local Variable

→ created inside a function and can be used only inside the function.

eg:

def Hello():

x = 10 # local

print(x)

② Global Variable

→ created outside a function and can be used anywhere in the program.

eg:

x = 50 # global

def myfunc(): print(x)      mutvar()

### ③ Instance variable

→ Variable belongs to each object of a class.

Eg:-

class Student:

```
def __init__(self, name):  
    self.name = name  
    # instance
```

### ④ Class variable

→ Variable shared by all objects of a class.

Eg:-

class Student:

```
college = "ABC - - "
```

## DATA TYPES

→ Data type tells what kind of value is stored in a variable.

define ↗ what the value looks like

↗ what operations you can do on it

Eg:-  $x = 10$  ↗ data store

variable

→ integer (Data types)

NOTE:- Kind of data stored in a variables.

Real life example:

Variable = a jar

Data type = what is inside the jar  
(rice, water, sugar).

→ py has built-in data types for different kinds of data.

### \* Number

→ Integer (whole number)

→ Float (Decimal number)

→ complex ( $z = 3 + 4j$ )

[combination of real and imaginary numbers]

### \* String (str)

→ collection of one or more characters.

Anything available on the keyboard can be stored

By default = str | a = "Hello123"

### \* Boolean

→ Result in True or False.

e.g., is-pass = True.

⇒ list [ ] , tuple () , range () → [ sequence  
data type ]

⇒ set data type →  
Set , Frozenset

⇒ map data type →  
Dictionary

⇒ Binary data type → Bytes and bytearray

⇒ None datatype  
↳ no value,  
nothing stored  
empty result.

eg:-  
`x = None  
print(x) → [None]`

→ `def hello():  
 print('Hi')`

`result = hello()  
print(result)`

Output :- Hi ←  
None ←

# String

## NOTE:-

String takes more space than other data types (int, float)

This happens because String stores every character with their own Unicode.

Unicode:- A universal character encoding standard.

Like A → 65

You can check the Unicode of any one by using

ord() function and

convert them back using chr() function.

e.g:-

print(ord('A')) → 65

print(chr(65)) → A

ord → check Unicode number

chr → In a number which Unicode is.

## Indexing

[ Start 0 → ]

a [ 0 : 0 : 0 - stop ].

a = 'Hello'

'start end'

print(a[0]) → H

print(a[:]) → Hello

print(a[0:3:2])

start from 0  
Date \_\_\_\_\_  
Step 1

silicing  $a[0:5:1]$  end in 4

## Conversion :-

Used to changing one datatype to another.

int  $\rightarrow$  float

float  $\rightarrow$  int

String  $\rightarrow$  int

int  $\rightarrow$  string

## Two types :-

### ① Implicit type conversion

$\rightarrow$  Automatically convert

eg:-

$a = 10, b = 5.5$

$c = a + b$

print(c)  $\rightarrow$  15.5 float.

bool  $\rightsquigarrow$  int  $\rightsquigarrow$  float  $\rightsquigarrow$  complex

### ② Explicit type conversion

$\rightarrow$  Manual.

eg:-  $x = "100" \rightarrow$  String.

$y = int(x)$

print(y)  $\rightarrow$  integers.

Rules:

(i) int  $\rightarrow$  float [ $a = 5, a = 5.0$ ]

(ii) float  $\rightarrow$  int [ $a = 5.0, a = 5$ ].

eg:  $a = 5.0$

$b = \text{int}(a)$

$\text{print}(\text{type}(b))$  [int]

(iii) only number can convert to complex

eg:

$\text{complex}(5) \rightarrow (5 + 0j)$

$\text{complex}('abc') \rightarrow X$ .

(iv) string allowed any type

$\rightarrow \text{str}(10)$

$\text{str}(3.14)$

$\text{str}(\text{True})$

$\text{str}([1, 2, 3])$ .

(v) bool() follow truth values.

$\rightarrow \text{bool}(0) \rightarrow \text{False}$

$\text{bool}(" ") \rightarrow \text{False}$

$\text{bool}(\text{None}) \rightarrow \text{False}$

$\text{bool}(10) \rightarrow \text{True}$

$\text{bool}('A') \rightarrow \text{True}$

iterable means object that can be looped over

Date \_\_\_\_\_

Page \_\_\_\_\_

(6) list, tuple, set → convert only iterables.

eg:-  
list ("ABC") → ['A', 'B', 'C']  
tuple (1, 2, 3) → {1, 2, 3}  
set ([1, 2, 3]) → {1, 2}  
set ('code') → {'c', 'o', 'd', 'e'}

Invalid :-

list (10) → not iterables.

NOTE:

→ character can't convert into int()  
→ 7 only values contain false in bool  
i.e. 0, 0.0, False, "",  
[], {}, ()

## Input and output.

OUTPUT → print()

INPUT →  
    input()

## Random

Random is a built-in python modules  
used to generate

- random number
- random choices
- random sequences
- random float, booleans
- shuffling list.

→ import random

(i) random.random() → [return float between  
→ import random    0 and 1]  
print(random.random())  
→ 0.675423.

ii) random.randint(a, b)

→ return random no. between a and b  
print(random.randint(1, 10))

Date \_\_\_\_\_  
Page \_\_\_\_\_

(iii) random.randrange ( start, stop, step )

→

Here, start, step and stop are  
include that differ than randint  
— more control

eg:

```
print(random.randrange(1, 10, 2))  
pick (1, 3, 5, 7, 9)  
→ pick one output  
→ (3)).
```

---

(iv) random.uniform (a, b)

→ return random float between a and b

```
print(random.uniform(10, 13))  
→ (11.28282).
```

---

(v) random.choice ( sequence )

→

Return one random item from a list,  
tuple, string.

eg:- Fruits = ['apple', 'banana', 'car']  
print (random.choice(Fruits))  
→ banana.

Fruits = ('apple')

```
print (random.choice(Fruits))
```

## (vi) random.choices (sequence, k = ?)

→

Return multiple random item with repetition allowed.

→ print (random.choices ([1, 2, 3, 4], k=3))

⇒ [1, 3, 3].

→ print (random.choices ('shekhar', k=5))

⇒ ['s', 'h', 'h', 'e', 's']

## (vii) random.sample (sequence, k = ?)

→ NO repetition

Unique Value.

e.g:

print (random.sample ([1, 2, 3, 4], k=2))

[2, 4.]

## (viii) random.shuffle (list)

→ change original list

## (ix) random.seed ()

→

## Operators.

Operators are symbols that perform operations on values or variables.

### Arithmetic operators.

[ +, -, \*, /, //, %, \*\* ]

// → floor division       $x = 10, y = 3$

eg:  $(x // y)$

→ 3 not include decimal.

% →  $x \% y$  → return remainder

\* \*\* →  $x * x^y$  → 1000 power.

### Comparison operators.

[ ==, !=, >, <, >=, <= ]

### Logical operators

[ and, or, not ]

return [ True / False ].

### Assignment operators:

[ =, +=, -=, \*=, /=, %=, //=, \*\*= ]

i.e  $x = 10$  ~~$x + 5 \quad \text{ie } x = x + 5 = 15$~~ 

## Bitwise operators.

→ works on binary (bit) of numbers.

i.e.

$\lceil \wedge \rceil \wedge \lceil \wedge \rceil \lceil \ll \rceil \lceil \gg \rceil$   
 AND, OR, XOR NOT Leftshift Rightshift

eg.:  $5 \& 3 \rightarrow 1 [x]$  $5 \rightarrow 0101$  $3 \rightarrow 0011$  $0001$  $0001 \rightarrow 1$  $5 \vee 3 \rightarrow 7 [+]$ 

XOR

$a = 5 \cancel{5 \times 3} \rightarrow -6$   
 print ( $\|a\|$ )

$5 \rightarrow 0101$   
 $1010$

rule ( $-n \leftrightarrow n+1$ ) $a = 6 \rightarrow -7$  $a = 7 \rightarrow -8$

membership operator

[ Exist or not in a sequence ]  
i.e in, not in  
gives True False.

Identity Operator

[ is, is not ].

eg:-

lst = [1, 2, 3]

print (2 in lst)      True.

b = lst

print (b is lst)      True.

## Escape Sequence in python

(a) `txt = " We are \"Hello\" Hi "` [  
`print (txt)`  
 $\rightarrow$  we are "Hello" Hi]

(b) single quote (' )

$\rightarrow$  `print (' It's a sunny )`  
 $\rightarrow$  [ It's a sunny ]

(c) Backslash (\ )

$\rightarrow$  `print ("This is a backslash: \\ ")`  
[ This is a backslash: \ ]

(d) New line (\n)

$\rightarrow$  `print ("Hello\nworld")`  
 $\rightarrow$  [ Hello  
World ]

(e) Form Feed (\f)

$\rightarrow$  `print ("Hello\fworld")`  
[ Hello  
World ]

(f) Backspace (\b)

$\rightarrow$  `print ("Hello\b")`  
[ Hello ]

# String

①  $x = 'hello'$   
`print(len(x))`  $\rightarrow$  5

② `print(x[0])`  $\rightarrow$  h

③ `print(x.upper())`  $\rightarrow$  HELLO

`print(x.lower())`  $\rightarrow$  hello

④ `strip()`  $\rightarrow$  remove white space from first and last not in the middle. anything.

eg:-

$a = "Hello"$   
`print(a.strip())` (+Hello)

`rstrip()`  $\rightarrow$  remove the end space.

eg:-

$a = "Hello //"$   
`print(a.rstrip('//'))` [Hello]

⑤ Replace  
 $\rightarrow a = 'hello'$  replace place is  
`print(a.replace('h', 'm'))`  
[metto].

### ⑥ split()

→ IS in list form return.

eg:-

a = "she man He"  
print(a.split(' ')) / [ 'she', 'man', 'He' ]

### ⑧ eg:-

a = 'hello'

b = 'hi'

print(a+b) → hellohi

print(a+' '+b) → hello hi

### ⑨ capitalize() → first letter capital.

eg:-

print(a.capitalize())  
→ Hello.

### ⑩ swapcase

→ change lowercase to uppercase  
and vice versa

a = 'Hello'

print(a.swapcase())  
→ hEllO

(11) centre

print (a. centre (50))

- 25 next - 25

print (a. centre (50, "x"))

(12) isalpha() [A-Z, a-z]

return True False

(13) islower()

isspace()

isalnum() [A-Z, a-z, 0-9].

(14) count()

→ print (a. count ('l'))

→ 2

(15) startswith()

→

print (a. startswith ('h')) .)

## Terminology

(I) Literals :- Fixed values written directly in your code.

Eg:-

`a = 5` (numeric literal).

(II) Iteration

→ process of repeating a block of code multiple times, usually over a collection (like list, string or range).

Eg:-

`fruits = ['apple', 'car']`

`for fruit in fruits:`

`print(fruit)`

`apple`  
`car`

(III) Case Sensitive

→ treat different to uppercase and lowercase letters

Eg: `a = "Hello"` ]  
`a = "Hello"` ]

(IV) Keyword: A reserved word in py that has a special meaning and cannot be used as a variable name, function name or identifiers.

For ex:- if, else, def, while, break,  
True, False, None --- so on.

Used by py to define syntax and structure  
of the language

(V) parameter :-

variable written inside the function  
definition.

Receive value when the function is called.

def add(a, b): → a, b parameters  
return (a+b)

(Vi) Argument :-

Actual value you pass to the function  
when calling it.

e.g.

add(5, 3) → 5, 3 are argument

## conditional statement

- allows decision making by executing different blocks of code based on conditions.
- control flow of our program based on some conditions.

3 types of its:-

(1) if statement :-

Execute a block of code only if the condition is True.

(2) if-else statement

(3) if-elif-else statements

→ check multiple condition in order.

→ check one block if the condition is True, another block if it's False.

eg:-

```
x ≠ 7  
if x > 10:  
    print("x is greater")  
elif x > 5:  
    print("x is less")  
else:  
    print('x is here')
```

## Loops in py.

Loop is a programming tools that repeats a block of code multiple times until a condition is met.



used to avoid writing the same code again and again.

→ To process multiple items in a collection.

### Types of Loops :-

① For loop

② while loop

### Intuition of loops

→ When you know the no. of iterations you will use a for loop.

→ When you don't know how many iteration you have to use but you know a condition that determine when to stop you will use while loop.

## For loop.

Range (start, end, step)

For i in range(1, 6): (1 to 5)  
    print(i)

## Loop for string :-

a = 'nature'

(I) for i in a: (directly)  
    print(a, end='')

(II) for i in range(len(a)):  
    print(a[i])  
    - using index values.

Break Statement:- Used to exit the loop prematurely when a certain condition is met.

For i in range(1, 5):  
    if i == 3:  
        break  
    print(i)

## continue Statement:-

Used inside loops to skip the current iteration and move to the next iteration.

eg:-

```
for i in range(1, 10):  
    if i == 5:  
        continue  
    print(i)
```

1
2
3
4
6
7
8
9

### while loops:-

used when the number of iteration is unknown before execution.

Syntax:- while condition:

# code of execution.

- Also have break and continue and else

eg:-

i = 0

while i < 5 :

print("Hye")  
 i += 1

# function

A block of reusable code that performs a specific task

- Reusability
- Reduce complexity
- makes program modular and readable.

Types of it:-

(i) Built-in function,

→ already available in python

i.e `len()`, `input()`

`print()`

`type()`

`max()`

`sum()` etc.

(ii) User defined function :-

created by us using `def` keyword -

Syntax:-

```
def function-name(parameters):  
    # code block  
    return value
```

(i) def greet():  
    print("Hello guys")

greet() # calling the function.

Return

vs

print

- (i) send a value back to the caller
- (ii) used for calculation
- (iii) cannot be used in further operation  
Returned value can be recycled

- (i) display output
- (ii) used for showing result
- (iii) cannot be used in further operation

### Types of Arguments.

(i) positional Argument :-

```
def intro(name, age):  
    print(name, age)  
intro('she', 24)
```

(ii) Keyword Argument :-

```
intro(name='she', age=24)
```

### (3) Default Arguments

→ A parameter that already has a default value

eg:-

```
def greet(name = "user").  
    print("Hello", name)
```

greet() → no argument than uses default user.

greet('she') → Hello, she

### (4) Variable length Arguments:

Allow you to pass any number of values to a function.

TWO types :-

i) \*A. args (Non-keyword variable arguments)

→ When you pass multiple values without knowing how many

eg:-

```
def add(*number).  
    print(number)
```

add(1, 2, 3, 4)  
add(1, 2)

| (1, 2, 3, 4 )  
| ( 1, 2 )

Inside the function,  
\* args become tuple.

(ii) \* Kwargs [Keywords Variable Arguments]  
→ When you want to pass multiple key-value pairs.  
eg:

def info(\*details):  
 print(details)

info(name='she', age=20)

Output becomes dictionary

NOTE:- \*args → handle many values  
\*\*Kwargs → handle many key-value pairs

## Lambda functions.

Date \_\_\_\_\_  
Page \_\_\_\_\_

- only one expression  
used to make small, anonymous function.

Syntax:-

Lambda argument : expression

(i)  $a = \lambda x : x^{\star\star} x$   
 $\text{print}(a(5)) \rightarrow 25$

(ii)  $\text{add} = \lambda x, y : x + y$   
 $\text{print}(\text{add}(5, 2)) \rightarrow 7$

(iii)  $\text{msg} = \lambda : 'Hello'$   
 $\text{print}(\text{msg}())$

(iv) lambda with map()

→ map applied function to each element.

numbers = [1, 2, 3, 4, 5]

squares = list(map(lambda x : x<sup>2</sup>, numbers))

$\text{print}(\text{squares})$

Output :- [1, 4, 9, 16, 25].

(V) lambda using filter()

→ filter keeps only values that satisfy condition

numbers = [1, 2, 3, 4]

even = list(filter(lambda x: x % 2 == 0,  
numbers))

print (even).

[2, 4].

(vi) lambda using reduce()

→ reduce combine all values into a single result

eg:

from functools import reduce

numbers = [1, 2, 3, 4]

sum\_all = reduce(lambda a, b: a + b,  
numbers)

print (sum\_all)

→ 10.

# Match statement.

- used to perform different actions based on different condition.
- used as a switch-case in other language.

## Syntax:

```
match expression:  
    case X:  
        code block  
    case Y:  
        code block  
    case Z:  
        code block  
    case _:  
        print("Uffs")
```

default:

```
day = 4  
match day:  
    case 1:  
        print('Sunday')  
    case 2:  
        print('---')  
    case 3:  
        print('')  
    case 7:  
        print('Saturday')
```

# Data Structure.

Date \_\_\_\_\_  
Page \_\_\_\_\_

Data structures are used to store, organize and manipulate data efficiently.

- List
- Tuple
- Dictionary
- set.

## NOTE:-

Now, there are some custom data structures as well like stack,  
queue,  
linked list,  
graph etc.

And around these data structure there are some algorithms like

• searching algorithms

• sorting algorithms

And this is why the study is called DSA.

## Terminology

### List

- (a) Mutable - changeable value after creation.
- (b) Ordered - Follow a fixed sequence and keep their position (index).  
 $a[0] = 2$  - Fixed one.
- (c) Heterogeneous - store different types of data together.  
eg:-  $a = [1, 'hello', 3.5, \text{True}]$ .
- (d) Duplicate - repeatable value

### list

List use square brackets ( $[]$ ).

eg:-

```
fruits = ['apple', 'banana']
```

- Ordered
- Mutable
- Heterogeneous
- Allow duplicate

Indexing: start from 0.

$$a = [10, 20, 30]$$

$$a[0] = 10$$

$$a[2] = 30$$

$$a[-1] = 30$$

$$a[-2] = 20$$

-3.	-2	-1
10	20	30

0 1 2

Updating list  $\rightarrow [a[1] = 50, a[2] = 30]$ .

$$a = [10, 20, 30]$$

### Methods of list:

1. `append()`  $\rightarrow$  add element at end.

e.g.

$$a.append(50) \quad [10, 20, 30, 50]$$

2. `insert()`  $\rightarrow$  add elements at specific index.

e.g.

$$a.append.insert(2, 50) \quad [10, 20, 50, 30]$$

$$[10, 20, 50, 30]$$

3. `Extend()` → join two list.

e.g.: `a.extend(b)`.

4. `pop()` → delete last index value

e.g.: `a.pop()` [10, 20, 50].

5. `remove()` → remove specific item.

e.g.: `a.remove(20) → value.`

`a.remove(20)` → [10, 50].

6. `clear()` → remove all those content  
but not list.

`a.clear()` → [ ].

7. `del()` → remove whole list

or, `del(a)`

or, `del a`.

8. `Reverse()` → reverse list.

i.e. `a.reverse()`

`a=[10, 20, 30]`

`a.reverse()` [30, 20, 10]

$a = [2, 3, 1]$

Date \_\_\_\_\_  
Page \_\_\_\_\_

- g. sort → arranging the data in ascending and descending  
→ It modify in the original list.

$a.sort()$   $[a = [1, 2, 3]]$ .

$a.sort(reverse = True)$  (Descending).

- ii. sorted() function.

→

Built-in function that returns a new sorted list, without modifying the original.

→ It works on any iterable (list, tuple, set, dict).

eg:  $a = [3, 1, 4, 2]$

$b = sorted(a)$

$print(b)$   $[b = [1, 2, 3, 4]]$ .

$b = sorted(a, reverse = True)$ .

if you do:

~~a = [2, 3, 4, 1]  
b = a.sort()  
print(b)  
print(a)~~

b → None

a → [1, 2, 3, 4]

11. Replicate → × 3.

~~a = b \* 3~~

~~print(a)~~

→ [1, 2, 3, 1, 2, 3, 1, 2, 3]

List comprehension in python.

A short and powerful way to create list in python using a single line

e.g.: nums = [i for i in range(5)].

Syntax:

new-list = [expression for item in iterable  
if condition (optional)]

$$T \rightarrow O(n)$$
$$S \rightarrow O(n)$$

Date \_\_\_\_\_

Page \_\_\_\_\_

example 2 :-

square = [x \* x for x in range(1, b)]

Evens = [i for i in range(10), if i % 2 == 0]

char = [c.upper() for c in 'hello']

result = ['even' if i % 2 == 0 else 'odd'  
for i in range(6)]

pairs = [(x, y) for x in range(3) for  
y in range(2)]

In comprehension set, dict, list are also

## Tuple

Date \_\_\_\_\_  
Page \_\_\_\_\_

- ordered
- immutable (can't change, update).
- heterogeneous
- Allow duplicate or delete individual element.

$a = (2, 3, 4, 5)$  ✓  
 $= (2, 4, 5, 5, 7, \text{True})$  ↴

### Tuple methods:

$t = (1, 2, 2, 3)$

1. `count()`  
 $\rightarrow t.\text{count}(2)$  → 2
2. `index()`  
 $\rightarrow t.\text{index}(2)$  → 1

## Set.

Date \_\_\_\_\_  
Page \_\_\_\_\_

- unordered
- mutable
- unique - element collection.
- No duplicate, automatically remove.

- ① unordered

→  $s = \{10, 20, 30\}$

# You can't use set.

Syntax:

$s = \{1, 2, 3\}$

or,

$s = \text{set}([1, 2, 3])$

- ① Empty set creation

$s = \text{set}()$  → ✓

$s = \{\}$  → ✗

↳ This is empty dict.

Methods of Set.

- ① add() → add a single value



a.add(54).

- ② removed()

→ a.remove(value).

### (III) discard()

→ a. discard(value)

If the value in the set is not present and want to remove / delete than,

remove() occur error but discard can't occur error.

e.g:

```
a = { 1, 2, 3, 4 }  
a. remove(5) — error.  
a. discard(5) — no error.
```

### (IV) pop()

→ remove random item -

a.pop()

### (V) clear() and del()

→

- a. clear() → clean all content
- a. del() → delete all tuple and content.

(VI). update ( )

→ used for adding multiple values.

e.g.: a. update ({7,8,24}).

### Set operations. (VVI)

I. Union (combine)

→ a.union(b) or,  
a|b

II. Intersection (common item)

→ a.intersection(b)  
or,  
a & b.

III. Difference (items in a but not b)  
→ (a - b).

IV. Symmetric Difference (not common)

(a ^ b).

V. Subset and Superset.

$$S_1 = \{22, 23, 24, 25\}$$

$$S_2 = \{24, 25\}$$

$$S_3 = S_1 \cdot \text{issuperset}(S_2) \rightarrow \text{True}$$

$$S_4 = S_2 \cdot \text{issubset}(S_1) \rightarrow \text{True}$$

### Frozenset.

→ An immutable set.

Date \_\_\_\_\_  
Page \_\_\_\_\_

## Dictionary

Date \_\_\_\_\_  
Page \_\_\_\_\_

- used to store data values in key:value pairs
- Unordered
- mutable
- key-value pair collection.

Keys → must be unique and immutable  
Values → can be any data type, duplicate allowed.

Syntax → { } or, dict() constructor.

We can perform CRUD (Create, Read, Update, Delete) operations on values but not all on keys cause the key cannot be changed after creation.

In dict: n = {1: 2, 2: 5, 5: 3}  
for i in n:  
 print(i) → 1

print(n[i]) → 2  
3  
5  
3

Methods:

O pop(key)  
→ d.pop('b')

(i) popitem() → remove last item  
d.popitem()

(III) del  
→ del d ['a']

(IV) clear()  
→ d.clear()

(V) a.update(b)

Exception handling.

Error :- It occurs due to mistake in the code that prevent it from running.  
These can be syntax error or logical errors.

Syntax error:- print ('hello')

Indentation Error:- Spacing error  
def fun():  
print('Hello') → ↴

Exception:- Unexpected error that occur during the execution of a program, which disrupts the normal flow of a program

Eg:-

print(10/0) —> error.

When py sees an error → your program  
stops suddenly.  
To stop this crash, we use Exception handling.

{ print(10/0) —> error.  
print('Hi') —> not run }

Now, this is zeroDivision error and can be  
counted as exception and because of this  
exception the next line cannot be executed

and the good part is we can handle it.

### Exception handling

Keyword

purpose.

(i) try  
wrap the block of code  
that might cause an  
exception.

(ii) Except  
Handles the exception if it  
occurs.

(iii) else

Run code only if no exception occur.

(iv) finally

Run code no matter what, whether there's an exception or not.

v) raise

Manually throw an exception.

① Try [Exception is written inside try].

try:

x = 10/0

print(x)

② Except — handle the exception

try:

x = 10/0

print(x)

except:

print('hi')

If in try code than goes to except.  
it not than it the first execute a + in

and ignore except.

eg:

```
try:  
    x = 10/10  
    print(x) → 1.0  
except:  
    print('hi') - x
```

③ ELSE → Execute only when no exception.

```
eg: try:  
    x = 10/10  
    print(x)  
except:  
    print('hi')  
else:  
    print('Hye')
```

best eg:

```
try:  
    num = int(input("Enter a number"))  
except:  
    print("Invalid input")  
else:  
    print('You entered:', num)
```

4. finally — Always execute

e.g.: code in finally runs no matter what

5. Raise — Manually throw an exception  
→

age = 15

if age < 18: value

Raise Exception('Age must be 18')

compulsory.

## File handling

- It means creation, reading, updating, deleting (RUD) operations that we can perform in files.

Used open() function to open a file.

Mode	Description
'r'	Read (default) - file must exist.
'w'	Write - Create file or overwrite
'a'	Append - adds to end of file.
'x'	Create - Create a new file, fails if it exists.
'rt'	Read + Write.
'wt'	Write + Read.
{Syntax}	file = open ('filename', 'mode')
'rt'	Append + Read.

eg:-

### (I) Reading a file :-

- `read()` - read entire file.

eg:-

```
file = open('data.txt', 'r')
content = file.read()
print(content)
file.close()
```

- `readline()` → read one line

file → ✓

```
line = file.readline()
print(line)
file.close()
```

- `readlines()` → all lines read as list

file → ✓

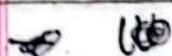
```
line = file.readlines()
```

## II.) Writing to a file

- write()

eg:-

```
file = open('data.txt', 'w')  
file.write('yes, I am ____')  
file.close()
```

→ 

for Best practice:

use With, It automatically open and close the file.

eg:

```
with open('data.txt', 'r') as f:  
    content = f.read()  
    print(content)
```

check it file exist or not:

import os

```
if os.path.exists('hello.txt'):  
    print('file found')
```

else:

```
    print('not')
```

## Deleting a file

```
import os  
os.remove('data.txt')
```

## OOP

Back to the paper topic is :-

(I) Intro of OOP

(II) class and objects

(III) constructor

(IV) str

(V) method - super  
 & class

(VI) Inheritance

# OOP

class →

2 types of things inside class.

- (a) Attributes
- (b) Methods.

variable defined inside the class are Attr.

function defined inside the class are method.

eg:

class Animal:

species = 'Dog' # Attribute.

def make\_sound(self): # method  
 print('Bark')

print(Animal().species)  
Animal().sound()

# Access attribute  
# call method.

t = Animal()  
print(t.species)

object create.

## Attributes and Methods

### Types of Attributes:-

#### • Class Attributes →

Normal variables created inside a class is class attributes.

#### • Instance Attributes →

Attributes created using an instance like self.name,  
self.age etc. is - - -

e.g.:

class Car:

wheel = 4 # class Attri.

def \_\_init\_\_(self, color):

self.color = color

# Instance Attr.

~~modules  
and  
package~~ | ~~under method / decorator, recursion,  
comprehension or list comprehension~~

Types of method:-

- (1) Instance method :-
- (2) class method :-
- (3) static method :-
- (4) super method :-
- (5)

# python

OOP

procedural

↓  
functional

versus  
biting

→ python is an object-oriented language, allowing you to structure your code using classes and objects for better organization and reusability.

Advantages:-

- ① clear structure
- ② Help keep your code DRY  
(Don't repeat yourself)
- ③ Allows you to build reusable application with less code-

## class and object in python

- class is a blueprint for creating objects.
  - Instance of class called object
- # creating class

class Student:

    name = "Karan"

# object creation  
s1 = Student()  
print(s1.name)

constructor

py automatically  
executed the init function  
while object is created.

All classes have a function called -init-(<sup>object</sup>)  
which always executed when the ~~class~~ is being  
initiated.

### --init-- function

```
class Student:  
    def __init__(self):
```

e.g. -

first called is -init-

```
class Student:  
    name = "shekhar"  
    def __init__(self):  
        print("Hello")
```

```
s1 = Student()  
Student.print(s1.name)
```

Output:-

Hello
shekhar

(1) class student :

def \_\_init\_\_(self, fullname, age):

self.fullname = fullname

self.ageofstd = age

s1 = Student("shekhawat", 22)

print(s1.fullname)

print(s1.ageofstd)

(2) Default constructor

class student :

# default constructor ]

def \_\_init\_\_(self): ]

pass

# parameterized constructor.

def \_\_init\_\_(self, name --)

:

,

,

—

## - `str()` method.

It controls what should be returned when the class object is represented as a string.

Without `--str__()`

e.g.: class person:  
def \_\_init\_\_(self, name, age):  
 self.name = name  
 self.age = age  
p1 = person("she", 22)  
print(p1)

[Output: address of p1],  
i.e. <main-person object  
at 0x150...>

with `str()`

```
def __init__(self, name, age):  
    self.name = name  
    self.age = age  
def __str__(self):  
    return f'{self.name} {self.age}'
```

p1 = person("she", 22)  
print(p1)

she 22].

Create Methods :- Methods are functions  
that belong to objects.

class person;

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def hello(self):

print(f'{self.name}, {self.age}')

print(self.name, self.age)

s1 = person("she", 22)

s1.hello()

[she, 22].

self parameter in this is used to  
referencie to the current instance of the class,  
and is used to access variable that belong  
to the class.

Methods return the value:

def get\_marks(self):

return self.marks

modify the value.

p1 = person ("John", 36)

p1.person = 4

p1.age = 40

print(p1.age)  $\Rightarrow$  40.

Delete object properties

del p1.age

Delete object

del p1

pass statement

If you create a class and have no content  
then used a pass

class Hello:  
 pass

Static Methods. self  
Method that does not used parameters

e.g.:

class Student:

@staticmethod → Decorators.

def hello():

print("Hello")

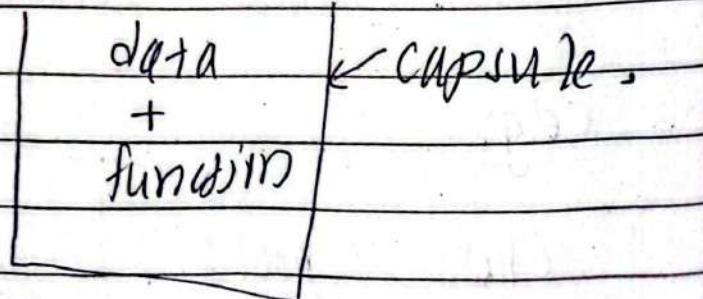
[Hello].

Abstraction.

Hiding the implementation details of a class  
and only showing the essential features to the user

Encapsulation.

Wrapping data and functions into a single  
unit (object).



## Absentiation

private entity.

double underscore.

self - name = "shekhar"  
used - score in attribute and  
method also.

## Inheritance.

Inheritance allows us to define a class that inherits all the methods and properties from another class.

parent class :- class being inherited from, also called base class

child class :- class that inherits from another class, also called derived class.

e.g.,

```
class School: parent class A
def __init__():
    child class B(A):
        if class C(A,B):
```

## Inheritance — Types.

Date \_\_\_\_\_  
Page \_\_\_\_\_

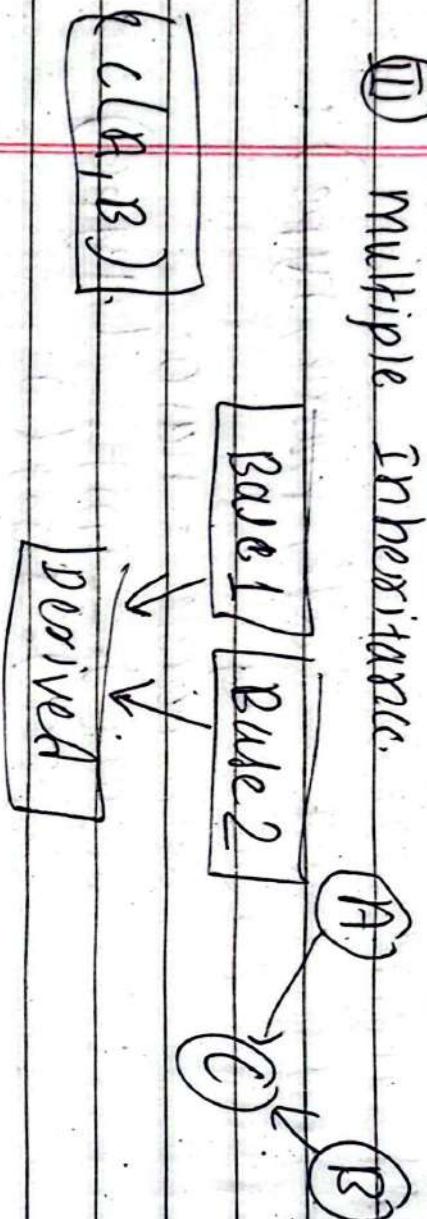
① Single Inheritance

Parent  
↓  
child.

② Multi-level inheritance. Parent (A)



③ Multiple Inheritance.



## Super method.

Used to access methods of the parent class.

e.g:-

```
class car:  
    def __init__(self, type):  
        self.type = type
```

Q static method

```
def start():  
    print("Hello")
```

```
class Toyota(car):  
    def __init__(self, name, type):  
        super().__init__(type)  
        self.name = name  
    def start(self):  
        print("Hello")
```

```
car1 = Toyota("pixus", "electric")  
print(car1.type)
```

Output ~ electric

Hello

- (1) static method
- (2) class
- (3) instance method

## class method.

A class method is bound to the class and receive the class as an implicit first argument.

class person:

    name = "Hello"

    @classmethod

```
def changeName( cls, name ):
    cls.name = name
```

pt = person()

pt.changeName("Ram")

print(pt.name)

[ Ram ]

print(person.name) [ Ram ].



You can also write like this

```
def changeName( cls, name ):
```

    person.name = name

class name

## property decorators

To make behave like an attribute.

Instead of calling obj.method(),  
you can just use obj.method (without  
parentheses).

Helps in hiding data,

eg: @property

```
def percentage(self):
```

```
    return (self.physics + self.chemistry) / 2
```

```
s1 = Student(90, 81)
```

```
print(s1.percentage)
```

# Recursion.

```
def sum(a, b):
```

~~return a+b~~

Here, a, b is a parameterized recursion

Code:

parameterized recursion

sum of 1 + 0 N

+ .. 1, 2, 3, N ⑤

```
def fun(sum, i, n)
```

if i >= n:

print(sum)

~~return~~

```
fun(sum+i, i+1, N)
```

fun(0, 1, 5)

## Recursion.

① factorial of a number.

$$5! = 5 \times 4 \times 3 \times 2 \times 1$$

```
def factorial(n):
```

~~if n <= 0:~~ if num == 0 or num == 1:

return 1

else:

~~return n \* factorial(n-1)~~

return n \* factorial(n-1)

NOTE: 0! = 1

```
def factorial(n):
```

n = abs(n) # negative traversal

if n == 0 or n == 1

return 1

else:

return n \* factorial(n-1)

# Reverse an array using recursion.

num = [2, 7, 3, 5, 4]  
index 0 1 2 3 4

reverse just left 2

and right 3

ex: num = [8, 7, 3, 2, 6, 1, 5, 9]

reverse = left 2 and right = 5 index  
using recursion ?

↓ [num.s.reverse()  
[num.s[: :-1]]]

logic

0 1 2 3 4 5

num.s = [5, 7, 3, 2, 6, 1].

left 0

right = len(num.s)-1

num.s[L], num.s[R] = num.s[R], num.s[L]  
left + 1  
right - 1.

Date \_\_\_\_\_  
Page \_\_\_\_\_

~~nums = [1, 2, 3, 4, 5]~~

Code:

```
def reverse(nums, left, right):
    if nums left >= Right:
        return
    nums[left], nums[right] =
        nums[Right], nums[Left]
    reverse(nums, left+1, right-1)
return nums.
```

```
print(reverse([2, 5, 7, 8, 9], 0, 1))
```

✓