Date
Page

python note plus related work

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.
  ie 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, C#0 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 :-

(I.) container used to store data.
eg:- → value stored
x = 10
└ variable

i.e.
Variable is a place where you keep data in a program.

Date
Page

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 variable:-

(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 thi program.
eg:
```
x = 50      # global
def myfunc(): print(x)
myfunc()
```

Date
Page

(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 object 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 valve looks like
⊢ what operations you can do on it

eg: X = 10 — variable; data store → integer (Data type)

NOTE :- Kind of data stored in a variables.

Date
Page

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 number]

* String (str)
  → collection of one or more characters. Anything available on the keyboard can be stored
  By default = str) a = "hello$123")

* Boolean
  → Result in True or False.
  eg:, is-pass = True.

Date
Page

* 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.

Date
Page

String

NOTE:-
String taxes more space than other data types (int, Float)
This happen because String store 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() functi

eg:-

```
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- step]
start end

```
a = 'hello'
print(a[0]) → H
print(a[:]) → Hello
print(a[0:3:2]
```

Slicing a[0:5:1] — start from 0 → end in 4 — Date — step 1.

Conversion :-

Used to changing one datatype to another.

int → float
float → int
string → int
int → string

Two types :-

(1.) Implicit type conversion
→ Automatically convert
eg :-
a = 10, b = 5.5
c = a + b
print (c) → 15.5 float.

bool → int → float → complex

(2.) Explicit type conversion
→ Manual.
eg :- x = "100" → string.
y = int (x)
print (y) → integer.

Rules:

(i) int ⟶ float [a = 5, a = 5.0]

(ii) float ⟶ int [a = 5.0, a = 5]

eg:
```
a = 5.8
b = int(a)
print(type(b)) [int]
```
5

(iii) only number can convert to complex

eg:
```
comple(5) → (5+0j)
complex('abc') → X
```

(iv) String allowed any type

→
```
Str(10)
Str(3.14)
Str(True)
Str([1,2,3])
```

(v) Bool() follow truth values.

→
```
bool(0) → False
bool("") → False
bool(None) → False
bool(10) → True
bool('A') → 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]) → {1,2,3}
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 0 and 1]
→ import random
print(random.random())
→ 0.675423.

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

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

Here, start, step and stop are include that differs 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))
→ (11.288282)

(v) random.choice(sequence)
→
Return one random item from a list, tuple, string.

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

Fruits = 'apple'
print(random.choice(fruits))

→ seq list, tuple, ...

(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']

(vi) random.sample (sequence, k = ?)

→ NO repetition
Unique Value.

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

(vii) random.shuffle (list)

→ change original list

(viii) random.seed()

→

Operators.

Operators are symbol 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 ** y → 1000 power.

Comparison operators.
[==, !=, >, <, >=, <=]

Logical operators
[and, or, not]
return [True / False].

Assignment operators:
[=, +=, -=, *=, /=, %=, //=, **=]

i.e $x = 10$
$x += 5$ i.e $x = x + 5 = 15$

Bitwise operators

→ works on binary (bit) of numbers.

i.e.

[& , | , ^ , ~ , << , >>]

AND, OR, XOR, NOT, left shift, right shift.

eg: $5 \& 3 \rightarrow 1$ [x]

$5 | 3 \rightarrow 7$ [+]

$5 \wedge 3 \rightarrow 6$ [0110] XOR

5 → 0101
3 → 0011
0001
0001 → 1

a = 5
print(~a) → -6

5 → 0101
1010

rule (-(n+1))

a = 6 → -7
a = 7 → -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)
→ we are "Hello" Hi

(b) Single Quote (\')
→
print('It\'s a sunny')
→ [It's a sunny]

(c) Backslash (\\)
→ print("This is a backslash: \\")
[This is a backslash: \]

(d) New line (\n)
→
print("Hello\nworld")
→ [Hello
World]

(e) Form Feed (\f)
→
print("Hello\fworld")
[Hello
world]

(f) Backspace (\b)
→ print("Hello\b")
[Hell]

String

Date
Page

1. x = 'hello'
   print (len(x)) → 5

2. print (x[0]) → h

3. print (x.upper()) → HELLO
   print (x.lower()) → hello

4. strip() → remove white space/anything from first and last not in the middle.
   eg:-
   a = " Hello "
   print (a.strip()) ("Hello")

   rstrip() → remove the end space.

5. eg:-
   a = "Hello //"
   print (a.strip('/')) [Hello].

6. Replace
   → a = 'hello'
   replace place i.e.
   print (a.replace ('h', 'm'))
   [mello].

(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

Date
Page

(11.) centre

—— print (a.centre(50))

—— print (a.centre(50, "*"))

(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'))

Date
Page

Terminology.

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

eg:-

a = 5 . (numeric literals).

(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 upper 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 identifier.

CamScanner

Date
Page

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.
eg:
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 statement**
   → check multiple condition in order.

→ check one block if the condition is true, another block its false.

eg:-

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

Date
Page

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.

Intution 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.

CamScanner

Date
Page

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 loop 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 loop :-

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
ie 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
```

@ def greet():
    print("Hello guys")
greet() # calling the function.

Return vs	print.
(i) Send a value back to the caller	(i) Display output
(ii) used for calculation	(ii) used for showing result
(iii) ~~cannot be used in further operation~~ Returned value can be reused	(iii) cannot be used in further operation.

Types of Arguments.

(i) positional Argument :-

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

(ii) Keyword Argument :-

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

Date
Page

(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)	(1,2,3,4)
Add(1,2)	(2,2)

Date
Page

Inside the function,
*args become tuple.

(ii) **B. 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.

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

Syntax:-
lambda argument : expression

(i) a = lambda x : x * x
print (a(5)) → 25.

(ii) add = lambda x, y : x + y
print (add (5, 2)) → 7.

(iii) msg = lambda : 'Hello'
print (msg())

(iv) lambda with map()
→ map applied function to each element.
number = [1, 2, 3, 4, 5]
Squares = list(map(lamda x : x * x, number))

print (Squares)

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

(v) lambda using filter()

→ filter keeps only values that satisfy condition

```
number = [1,2,3,4]
even = list(filter(lambda x: x%2==0,
                   number))
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 2:
        code block
    case _:
        print("Hi")
```

default:

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

Date
Page

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, True].

(d) Duplicate - repeatedable value

list

List use square brackets ([ ]).
eg:-
fruits = ['apple', 'banana']

- Ordered
- Mutable
- Heterogeneous
- Allow duplicate

Date
Page

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

Udating List → [a[1] = 50, a[2] = 30].

a = [10, 20, 30].

Methods of list:

1. append() → add element at end.

eg:

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

2. Insert() → add elements at specific index.

eg:

a.~~append~~ insert(2, 50)

[10, 20, ~~30~~, 50].

[10, 20, 50, 30].

3. Extend() → join two list.

eg:. a.extend(b)

4. pop() → delect last index value

eg:. a.pop() [10, 20, 50].

5. Remove() → remove specific item.

eg:. → 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.

a.reverse()

i.e. a=[10, 20, 30]

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

a = [2, 3, 1]

Date
Page

9. sort → aranging the data in ascending and descending
→ It mediky in the orginal list.

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

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

10. 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 → X 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

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

Syntax:

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

T → O(n)
S → O(n).

Date
Page

example 2:-

```
square = [x*x for x in range(1,6)]
Evens = [i for i in range(10) if i%2==0]
chars = [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 SeA, Dict, List are also.

Date
Page

Tuple.

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

a = (2, 3, 4, 5)
= (2, 4, 5.5, 7, True)

Tuple methods:

t = (1, 2, 2, 3)

1. count()
   → t.count(2) → 2

2. index()
   → t.index(20) → 1

Date
Page

Sets.

- unordered
- mutable
- Unique-element collection.
- NO duplicate a, automatically remove.

① unordered
→ S = {10, 20, 30}
You can't use S[0].

Syntax:
S = {1, 2, 3}
or,
S = set([1, 2, 3])

(I) Empty set creation
S = set() → ✓
S = {} → ×
↳ This is empty dict.

Methods of set.

(I) add() → add a single value
→
a.add(54).

(II) remove()
→ a.remove(value).

CamScanner

(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.

eg:
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.

Date
Page

(VI) update ( )
→ used for adding multiple values.

eg: a.update({78,24}).

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) Issubset and Issuperset.

Date
Page

s1 = {22, 23, 24, 25}
s2 = {24, 25}

s3 = s1.issuperset(s2) → True
s4 = s2.issubset(s1) → True

Frozenset.

→ An immutable set.

Dictionary.

- 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
                    2
                    3
    print(n[i])   → 2
                    5
                    3
```

Methods:

① pop(key)
→ d.pop('b')

② popitem() → remove last item
d.popitem()

(III.) del
→ del d['a']

(IV.) clear()
→ d.clear()

(V.) a.update(b)

Exception handling

Error :- It occur 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 nxt 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')
```

→ Hi

If in try error than goes to except
if not than it the first execute a try

and ignore except.

eg:

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

→ 1.0

(3) Else → Execute only when no exception.

eg:

```
try:
    x = 10/10
    print(x)
Except:
        print('hi')
Else:
        print('Hyee')
```

1.0
he

best eg:

```
try:
    num = int(input("Enter a number"))
Except:
    print("Invalid input")
Else:
    print("you entered:", num)
```

Date
Page

4. finally — Always execute.
eg:
code in finally runs no matter what

5. Raise — Manually throw an exception
→

```
age = 15
if age < 18:
    raise ValueError('Age must be 18')
```

compulsory.

Date
Page

File handling.

- It means creating, reading, updating, Deleting (CRUD) 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.
r+	read + write.
w+	write + read.
a+	Append + Read.

Syntax:

```
file = open('filename', 'mode')
```

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()
```

Date
Page

(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')
```

Date
Page

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 of super class
(VI) Inheritance.

OOp.

Date
Page

class →

2 types of things inside class.

ⓐ Attributes
ⓑ Methods.

Variable defined inside the class are Att.

function defined inside the class are method.

eg.

```
class Animal:
    specie = 'Dog'      # Attribute.

    def make_sound(self):      #method
        print('Bark')
```

```
print(Animal().species)      # Access attribute
Animal().make_sound()        # call method.
```

```
t = Animal()      → object create.
print(t.species)
```

Date
Page

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 ---

eg:

```
class Car:
    wheel = 4    # class Attri.
    def __init__(self,color):
        self.color = color
        # Instance Atf.
```

modules and packages | Dunder method | decorator | Recursion | comprehension of list, tuple, set

Types of method :-

1. Instance method :-
2. class method :-
3. static method :-
4. super method :-

Date
Page

python

OOP

procedural
↓
functional
↓
OOP

reusability

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

Advantages:- (I) clear structure
(II) Help keep your code DRY (Don't repeat Yourself)
(III) 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 = "Kaxan"

# object call/creating
s1 = Student()
print(s1.name)
```

constructor.

py automatic executed the init function while object is created.

All classes have a function called -init-() which always executed when the object is being initiated.

--init-- function.

```
class Student:
    def --init--():
```

e.g.:

first called is --init--

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

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

Output:

Hello
shekhar.

Date
Page

(1.) Class Student :

```
def __init__(self, fullname, age):
    self.fname = fullname
    self.ageofstd = age

s1 = Student("shekhar", 22)
print(s1.fname)
print(s1.ageofstd)
```

→ parameters

(2.) Default constructor.

```
class Student :

    # default constructor
    def __init__(self):
        pass

    # parameterized constructor.
    def __init__(self, name ...)
```

__str__() method.

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

Without __str__()

eg:

```
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]

Date
Page

create methods:- methods are function 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 reference to the current instance of the class, and is used to access variable that belong to the class.

Methods returns the value:

```
def get_marks(self):
    return self.marks
```

modify the value.

```
p1 = person1("John", 36)
p1.person = 4
p1.age = 40
print(p1.age)
```

⇒ 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.
```

Date
Page

Static Methods.

Method that does not used self parameters

eg:

```
class Student:
    @Staticmethod → Decorators.
    def hello():
        print("Hello")
```

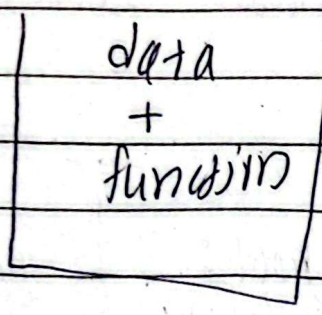
[Hello].

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 (objects).



Date
Page

~~Abstraction~~

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 form another class.

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

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

eg:

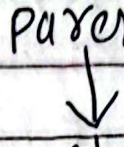
```
class School:
    def __init__
```

```
parent class A
    |
child class B(A):
    |
if
class C(A,B):
```

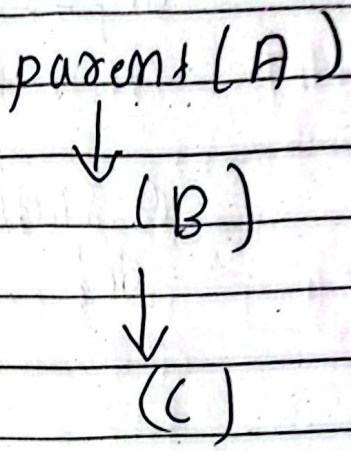
Date
Page

Inheritance — Types.

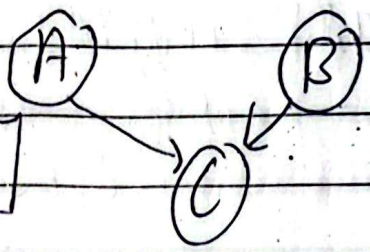
(i) Single Inheritance

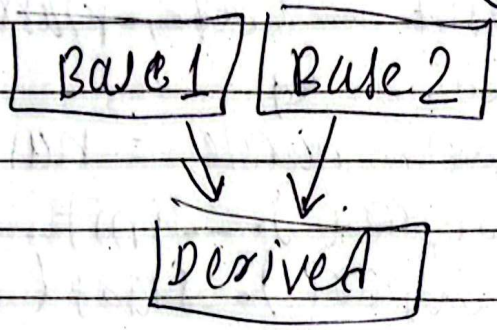


(ii) Multi-level inheritance.



(iii) Multiple Inheritance.





C(A,B).

Super method.

used to access methods of the parent class.

eg:-

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

    @ static method
    def start():
        print ("Hello")

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

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

output :- electric
Hello!

(i) Static method
(ii) class
(iii) instance me...

class method.

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

```
class person:
    name = "Hello"
    @classmethod
    def changeName(cls, name):
        cls.name = name

p1 = person()
p1.changeName("Ram")
print(p1.name)          [ Ram ]

print(person.name)      [ Ram ]
```

↓

you can also right like this

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

(class name)

Date
Page

property decorators.

To make behave like an attribute.
Instead of calling obj.method(),
you can just use obj.method (without
parenthesis).

Helps in hiding data,

eg:
```
@property
def percentage(self):
    return (self.phy + self.chem)/2

s1 = student(90, 86)
print(s1.percentage)
```

Recursion.

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

Here, a, b is a parameterized recursion

code: ↙ parameterized recursion

sum of 1 to 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)

n=abs(n)

Date
Page

Recursion.

① factoral of a number.

5! = 5x4x3x2x1

```
def factorial(n):
    if num==0 or num==1:
        return 1
    else:
        return n * factorial(n-1)
```

NOTE: 0! = 1

```
def factorial(n):
    n = abs(n)  # negative to positive
    if n==0 or n==1
        return 1
    else:
        return n * factorial(n-1)
```

Date
Page

Reverse an array using recursion.

nums = [2, 7, 3, 5, 4]

index → 0 1 2 3 4

reverse just left 2 and right 3.

?

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

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

```
nums.reverse()
nums[::-1]
```

Logic

0	1	2	3	4	5
8	7	3	2	6	1

nums = [8, 7, 3, 2, 6, 1]

left 0

right = len(nums) - 1 = 5

nums[L], nums[R] = nums[R], nums[L]

left + 1

right - 1.

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

Code:

```
def reverse(nums, left, Right):
    if 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)
```