1) Python (Introduction of Fundamental of Python)

i) What is python?
It is world's fastest growing programing language. It is a general purpose language with a simple, and beginner friendly syntax.

Benefits: It solve complex prob in less time with a fewe lines of code.

Feautres of: Beginner-friendly, Cross platform, vers

2) Fields where we can use Python

Data-Analysis → ML/Ai
Automation → Testing
web Apps → Hacking
Desktop Apps → Mobile App

2 History

Date: / /
Page No.

Implementation was started by Guido Van Rossum in 1989

It developed at National Research Center called CWI in netherland

centerum wiskunde information

* Parent language of python
  - ABC ⇒ Exception Handling
  - Amoeba OS ⇒ mainly used ABC

He noticed throughout his journey that so many bugs are there and it is also not a multi purpose language. So he created a python.

ABC is influenced by SETL

- Modula-3 ⇒ oops mechnaism

* why named python (snake)

Guido called the name python after 1970's BBC comedy series

Monty Python's flying Circu

So not bzo of he love that series, but he wants a short unique and slightly mysterious name.

* version history → major version.

Python 0.90 (1991), 1.0 (Jan 1994), 2.0 (Oct 16 2000), 3.0 (Dec 3 2008)

3.8.3 (may 13 2020)

most latest or stable version of python ⇒ python 3.13.3 released on April 8 2025

* In Industry 2 major versions are available. 1) python 2 vs python 3

python 2 → 2000 - 2020 (legacy version)

Extra feath

1. No updates,
No maintenance
No security updates.

Date: / /
Page No.

Python 3 ⟹ Future version
⇓ OR
current Stable
LTS version
(long term support).

- constantly updates
- Routine maintenance
- with security updates

} as it is latest

⟹ let's begin our Journey

Installation on windows

1) Python executable file
⇓
official website.

2) After installation open cmd
Python --version

⟹ lets Start Our first Py program

When we download py 1st time it also install a text editior (Built in) which is idle ⇒ integrated development enviornment.

Date: / /
Page No.

To run a program in Py we have total 2 mode.

1) interactive Mode (Shell mode)
2) Script mode. → .py

By default it's open in interact...

>>> ⇒ it indicates ↑ or interpreter mode.

>>> statement + Enter
Execute at a time.

printing 'hello world

Printf ("Hello world") ⇒ C lan..
print ("hello world") + Enter //.

OR.

Terminal ⇒ python3 + enter

SO Basically in interactive mode the py interpreter runs comands line by line. Each comand executes as its entered, and result is entered immediately

Script mode ⇒ it written in a file .py (extension) and then execute as a whole

exit()

Print() fn

Built in fn : used to print anything in screen

Syntax :

Print (object(s), [sep = seperator, end = end])

↑ string variables class objects

↳ optional

sep

1) print ("Ak", "Paul")

2) print ("Ak", "paul", sep = "-")

3) Reuse of print ⇒ end + Enter

End

print ⇒ helo
print ⇒ world

print ("helo"); print ("world")

↑ Semicolon ⇒ multiple statements,

print$("hello", end="\n");
print("world")

↳ By default

Sept ⟹ a print f^n ke multiple string ko join karti he

End ⟹ 2 diff print f^n ko join karti he.

Input from use

input() ⇒ built in f^n, use to take "input from use

Syntax:

input([prompt_message])
↳ optional

To hold the value we use variable (container)

var_name = data ⇒ string number collection input dt

x = input()
print x

declaration is not possible in variable //,

int x; ✗ ⇒ x = None ✓

⇒ x = 5 (string value //.) y = 5 ⇒ z = x + y
55

input by default use ~ string
convert too

Data type & type casting / conversion

Data ⇒ indicates data type.

Py is a dynamically typed langu
means Run time objects have a type
↳ it allocates data type at run time
no need to allocate via another step.

How many Datatypes ⇒ Primitive
collection

Built in primitive

integer ⇒ 98
float ⇒ 4.2
string
complex ⇒ c = 5 + 3j
↳R ↳I

integer ⇒ x = 98
type(x) ⇒ checks data type.

x = 98
print (x, type(x))

String

'helo', "hello", """hello"""
simple string
multiline string

Date: / /
Page No.

Complex ⇒ c = 5 + 3j
5 → Real part
3j → imigenry part

If no real numbers in complex then.

d = 0 + 7j or + 7j

* Type casting constructors.

used for converting datatype into another.

int()
float()
str()
ord() ⇒ ordinal
chr() ⇒ character

bin() ⇒ binary
hex() ⇒ hexadecim
oct() ⇒ octal
complex()

1) int ⇒ s = '1 2 3 4 9'
print(s, type(s)) ⇒ str//

int() ⇒ int(s)
↳ var or obj

2) a = int(input("Enter value A: "))
b = int(input("Enter value B: "))
c = a + b ⇒ 10//

3) ord('A') ⇒ 65 ⇒ Returns ASCII value

4) chr(65) ⇒ 'A' ⇒ Returns character

5) a = 12
bin(a) ⇒ '0b 1100'
↳ binary value.
↳ whole number is in binary represents that.

6) oct(12)
'0o14
↳ octal

7) hex(15)
'0xf
↳ hex.

8) print(bin(ord('A'))) ⇒ provides binary code.

Operators

Arithmetic ⇒ mathematical Operation
Assignment ⇒ Assigning values.
Comparison ⇒ compare 2 values
Logical ⇒ combine condition
Identity ⇒ compare 2 variables
Membership ⇒ check object has sequence or not.

Arithmetic operators

+ , − , × , / , %, ,
** , // ⇒ floor Division

↳ Exponentiation (power) $2^3 = 8$

(2×2×2)

2 ** 3 ⇒ 8.

Floor Division ⇒ 12 // 3 ⇒ 4

12/3 = 4.00

Assignment operator.

= , += , −= , ×= , /=
%= , **= , //=

a = 5
a = a + 2 or a += 2.

Date: / /
Page No.

* Comparision Operator.

==, !=, >, <, >=, <=

* Logical Operator → Truth table
= True or false

and, or, not.

a > b and a < b → (i)
a > b or a < b → (ii)
a > b → true not a > b → false

* Identity Operator.

is ⇒ Returns True if both var are the same object.

is not → Returns True if both var are not the same object.

a = 3
b = 3 } a is b / a == b } true.

a = 3
a = 3.0 } a == b } true.
a is b → false

↓
it's check the value and the data type also

Date: / /
Page No.

Membership operator.

Used to test if a sequence is presented in an object.

Sequence ⇒ collection of a data type.

eg.
String or list

in and not in

in ⇒ Returns True if a sequence with the specified value is present in the object.

not in ⇒ Returns true if a Sequence with the specified value is not present in the object.

eg.

S = 'Akshay Hello'

'e' in S ⇒ True.

for reverse order use not in

Equality vs identity

a = 3
b = 3
} a == b } True
a (is) b } True.

↳ checks value and their data type //.

how to know that is it really same or not, so python provide us a id function.

id ()

a = 3
b = 3
] id(a) ⇒ 45...908.
id(b) ⇒ 45...908 //.

means they are same... //.

Date: / /
Page No.

Datatypes in Detail

* String formatting

⇒ inject variables or obj in string

⇒ Types of string

1) % operator
2) format() method
3) f-string

Py uses. C-style formatting to create new formatted strings.

The % operator is used to format a set of variables together with a format string, which contains normal text with argument specifiers, like %s and %d.

Syntax:

" string argument Specifier "
%(object(s))

Ex n = 3

s = 'value is n'

s = 'value is %s' %n

↳ %d ✗

for multiple string value.

```
n1 = 'Akshay'
n2 = 30
n3 = 'My name is %s and age is %s' % (n1, n2)
print(n3)
```

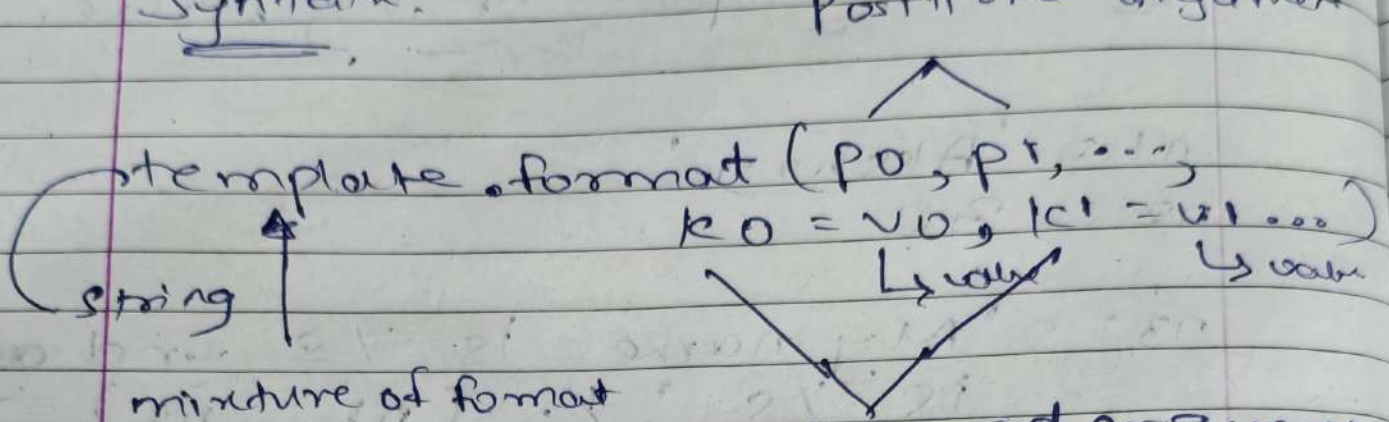
(2) format method ⇒ It is a Built in method that formats the given string into a nicer output in python.

It takes any number of parameters But, is divided into 2 types of parameters:

1) positional parameter ⇒ which we have to wrap in a { index } ⇒ array

2) keyword parameters ⇒ key = value that can be accessed with key of parameter inside { key }

Syntax.



Ex. n1 = 3
n2 = 5
s = "Hello, Addition n1 + n2" ✗
→ place index also.
s = "Hello {1} + {2}".format(n1, n2)
Hello 3 + 5 ⇒ output.

Example of positional argument

Ex. keyword argument.

s = "Hello {key1} + {key2}"
.format(key1 = n1,
key2 = n2)

Output ⇒ Hello 3 + 5.

Date: / /
Page No.

f-string ⟹ It used to make string interpolation simpler.

It provide a concise and convinent way to embed py expressions inside string literals for formatting.

Syntax:

f"string {object/expression}"

```
name = "Ak"
age = 14
s = f"My name is and age is {age} {name}"
```

String manipulation

Perform some operations to retrieve our intended output from a string.

Concatenation, length, Replace, Multiplication, finding, Slicing, Appending, changing case, Escape sequence

1) Concatenation ⇒ Join strings together

Syntax :- str1 + str2

2) Multiplication ⇒ It multiple copies of a string.

Syntax ⇒ str * (n) ⇒ Any number

S = "AK" * 3 ⇒ output - AKAKAK

3) Appending ⇒ To add something at end of the string.

⇒ (+=) operator.

Syntax$ String t = content ↦ Can be any string

S = "AK"

S+ = "paul"

4) length ⇒ It provide the length or character present at String using len() fun

Syntax ⇒ len(string)

S = "Akshay"
len(S) ⇒ output : 6//.

5) finding ⇒ Search any thing and return the starting index if not ⇒ -1
Syntax ⇒ string.find(substring)

6) changing Case ⇒ lower() & upper() doesn't change in orignal state.
Syntax ⇒ string.lower()

7) Replace ⇒ replace() method.
string.replace(old, new)

8) slicing ⇒ str[start:end[step value]]
optional.

S = 'Akshay'
S=[0] ⇒ A//.

S[4:6]
inclusive ↳ exclusive 6 7

S[0:] ⇒ give the whole string
↓
S[:] or S//.

Date: / /
Page No.

Step value.

A (k) S (h) a (y) ⇒ A S a.

S[0 : : (1)]
↳ returns the whole Name

S[0 : : (2)] ⇒ Skip one value.

9) Escape Sequence :- "\n" ⇒ ~~(\n)~~ a ~~string~~.

"String \Ak"
↳ \n, \t, \\

"My country name is \"India\""

Collection Datatype

Date: / /
Page No.

ordered or unordered group of similar or diff datatypes.

List ⇒ same as arrays.

Set ⇒ unordered collection, mutable

Tuple ⇒ immutable (same as list)

Dictionary.

Arrays ⇒ collection of items.

1) list ⇒ mutable, contain Data types like integers, strings as well as obj
It is represented by list class.
It is declared by [ ]

i) ⇒ l = [ ]
↳ 3.4, 4.2, True (3+2j)
print (l) → get all those values

2) ⇒ l[0:2] or l[::2]

2) Set ⇒ unordered collection of data types, mutable, no duplicate Elements.

The order of elements in a set is undefined though it may consist of various elements.

represent by set class.
declared by { },

Syntax ⇒ obj name = { value1, value2

S = { 2, 5, 2 }
print (s.) ⇒ 5. ⇒ do not print duplicate Elements

3) Tuple ⇒ ordered collection of py objects much like list

Sequence of values stored in a tuple can be of any type, and they are indexed by integer

tuples are immutable.
represent by tuple class.
declared by ( ) ⇒ parentheses

t = (3 , 4, 3.2 , false)

Date: / /
Page No.

4) Dictionary is unordered collection of data values, used to store data values like a map with key: value pair

key - value pair in a Dictionary separated by a colon (:), whereas each key is separated by a (,)

represent by dict class, declared by { }

ex// d = {
"name": "rahul",
"age": 22
}

d['name'] ⇒ output ⇒ rahul.

CDT - Methods

Date: / /
Page No.

append, count, sort, pop, remove, reverse, clear, extend, copy, insert, index

List

i) l = ['Ak', 'Paul', 'Py']

l.pop() ⇒ remove last element

print(l) ⇒ [Ak, paul]

2) l.append('Py')

print(l) ⇒ Ak paul py

3) l.clear ⇒ Empty element or del everything.

Tuple ⇒ count(), index()

t = (4, 9, 6, 4, 2, 6)

t.count(4)

↳ (2) ⇒ repeating 2 times.

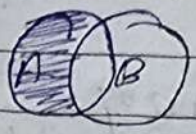
t.index(6) ⇒ 2//

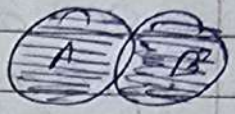
Set ⇒ we can perform mathematical operations in set.

⇒ (1) union (A)(B) ⇒ ∪

(2) intersection (A)(B) ⇒ ∩

Date: / /
Page No.

3) Difference 

4) Symmetric Diff 

add, update, isdisjoint, Clear, difference-update, issubset, copy, discard, difference, intersection symmetric difference, pop, remove, union, intersection-update, symmetric-difference-update.

Example. A = {2, 6, 3, 8, 2, 4}
print A = 2, 3, 6, 4, 8

B = {6, 2, 1, 8, 7, 0, 3}
print B = (0, 1, 2, 3, 6, 7, 8)

A.union(B) ⇒ 0, 1, 2, 3, 4, 6, 7, 8.

A.difference(B) ⇒ 4.

1) difference-update → remores the items that exists in both sets.

2) issubset() ⇒ returns true if all items in the set exists in the specified set. Otherwise returns false.

x = {"a", "b", "c"}
y = {"d", "e", "a", "b", "c"}
z = x.issubset(y)
Print(z)

Date: / /
Page No.

Dictionary :- clear, items, setdefault, copy, keys, update, fromkeys, pop, values, get, popitem.

d = {
name : "Rahul"
age : 23
}

d.values() → get all values.
d.keys() → get all keys.
d.items → get everything
d.clear → clear everything.

★★★★★★★★★★★★★★★★★★★★★★★

1) thisset = {"apple", "banana", "cherry"}
thisset.discard ("banana")
print (thisset)

2) x, y → z = x.difference(y) = print(z)

Shallow or deep copy

Date: / /
Page No.

* L1 = [2, 8, 4, 5]
  L2 = L1
  L1.pop ⇒ 2, 8, 4
  L2 = (?)
  → Deep copy

* L1 = [2, 8, 4, 5]
  L2 = L1.copy()
  L1.append(9)
  L1 ⇒ 2, 8, 4, 5, 9
  L2 ⇒ (?)
  ⇒ Shallow copy

* mutability of tuple

t = (2, 3, 6, 8)
type(t) ⇒ tuple

t = (5) ⇒ type ⇒ int.
t = (5,) ⇒ type ⇒ tuple.

Types of comments.
Collection type casting
constructors & del keyword

Date: / /
Page No.

⇒ Single line comment → #
multi line comment → """ """ statement

⇒ type casting constructors,

list(), tuple(), set(), dict()
frozenset()

L = [4, 6, 2, 4] ⇒ list
type(L) ⇒ list.

tuple(L) ⇒ convert into tuple
t = tuple(L)
type(t) ⇒ tuple.

S = set(L)
print(S) ⇒ 2, 4, 6.

frozenset(L) ⇒ unique and immutable.

* del // ⇒ remove obj from memory
a = 5
del a ⇒ print(a) ⇒ name a is not defined
del L[1] ⇒ Deleted 1 index value

3. # Control Structure & looping.

Date: / /
Page No.

It is just a way to specify flow of control in programs.

It analyzes and choose in which direction a program flows based on certain parameters or condition

Types ⇒ if, if else, ladder if else, nested if else

```
if condition :
----// statement
↑
indentation
(white space)
```

Eg. 1) a = 4 b = 3

```
if a > b :
    print("a is max")
else :
    print("b is max")
```

2) a = 40 b = 9 c = 5

```
if a > b and a > c :
    print("a is max")
else :
    print("b is max")
```

Date: / /
Page No.

* ladder ⇒ 2 or more conditions.

```
if condition1:
elif condition2:
    # statement
elif conditionN:
    # statement
[else:
    # statement ] ⇒ optional.
```

Ex

```
ag = 19

if age > 18:
    print("you can vote.")
elif age == 18:
    print("y c v")
elif age < 18:
    print("you c't vote")
```

* Nested.

```
if con1:
    # statement
    if cond2:
        # statement
else:
    # statement
    if cond1:
        a statement
```

short hand (ternary operator)

Date: / /
Page No.

a>b ? true : false ✗

✓ if a>b else

(True) if a>b else (false)

→ a = 22 b = 30 c = 23

```
if a>b :
    if a>c :
        print("a is max")
    else:
        print("c is max")
else:
    if b>c:
        print("b is max")
    else:
        print("c is max")
```

Loop

Date: / /
Page No.

1) while loop 2) for loop

```
i = 1

while ic = 10:
    Print (i)
    x (i++) ⇒ i = i + 1 ⇒ i += 1
```

2) for loop

```
for var in sequence:
    # statements.
```

Eg:

1) for i in "gurukul":
   print (i)

2) S = (3, 24, 8, 2, 6, 7, 9)

```
for i in S:
    if i % 2 == 0:
        print (i)
```

3) S = {. . .}

```
for i in s:
    print (i) ⇒ get key.
for i in s.values():
    Print (i)   ↳ s.items()
```

```
for i,j in S.items():
    print(f"{i}:{j} => {i} {j}")
else:
    print("always executes")
```

* Range() ⇒ Returns a sequence of numbers.

numbers are starting from 0 by default, and increments by 1 (by default), and stops before a specified number

range(start, end, [step])

end ↑ exclusive
step ↑ optional

[0, 1, 2, 3, 4, 5, 6]

range(7) ⇒ output all

range(0, 7) output ⇒ all

range(0, 7, (1)) ⇒ output ⇒ all
↳ (2) ⇒ output : 0, 2, 4, 6

```
for i in range (9):
    print (i)
```

* A control statements is a statement that determines whether other statements will be executed or not.

break → terminate the loop

continue → jump to beginning, skipping the execution

pass → write to empty loops

1)
```
i = 1
while i <= 10
    if i == 5
        break (continue)
    print (i)
    i += 1
```
1, 2, 3, 4 → for ↵
1, 2, 3, 4, 6..

2)
```
for i in range (1, 11):
    if i == 5:
        continue
    print (i)
```

3) Pass is used to write empty loops, is also used for empty control statements, function and classes.

d) for i in range (10, 10):
    if i == 9:
        pass
    print(i)

nothing happens

e) for i in [4, 6, 2, 5, 2]:

we can let it be empty.

pass #.

Comprehension

Date: / /
Page No.

It provide us ~~will~~ a short and concise way to construct new sequences such as list, set, dict etc using already defined sequences.

list, Tuple/Generator, Set, Dictionary.

It provides an elegant way to create a new lists.

Syntax:

optional

1) newlist = [expression for member in iterable (if)]

```
l = [3, 2, 5, 1, 2, 9,]
ans = [i for i in l
       if i%2==0]
```

ans ⇒ print → [2,] → even.

2) Tuple/ Generator ⇒ It don't allocate memory for the whole list. instead, they generate each value one by one which is why are memory efficient.

t = (1, 2, 3, 4, 5, 6)

squares = (i*i for i in t)

squares

<generator object>

print (tuple (squares))

print (list (squares)) → []

3) Set → It provides an elegant way to create new set with unique elements.

s = {3, 21, 6, 3, 1, 5, 21, 6, 4}

ans = {i**3 for i in s}

4) Dictionary : Create a new dictionary using existing sequence with our modification.

Syntax :

new = {key : value expression for member in iterable (if condition)}

Date: / /
Page No.

l = [2, 4, 6, 1, 3]

d = { i : i x i for i in l }

print(d)

Functions

Built in (6r)
UDF

⇒ Built in.

1) abs() ⇒ return the absolute value.
-ve ⇒ +ve ⇒ abs(-50) ⇒ 50

2) all() ⇒ scan every element if true ⇒ true, false ⇒ false
all([1, 2, 3, 4]) ⇒ 0, null, false

3) any() ⇒ Reverse of all.
any((3, 2, 45, false) ⇒ True.

4) bin() ⇒ return binary
bin(15) ⇒ "0b 1111"
↳ represents it is binary format.

5) bool(), int(), type(), list(), set(), dict(), float(), str(), type(), print(), chr() ⇒ Ascii value
chr(97) ⇒ a
ord('a') ⇒ 97
↳ ordinal

Date: / /
Page No.

6) dir([12, 34]) } return all method of list
7) dir(list())
8) eval("print('hello')")
↳ it only works when we have python code inside the eval
9) help(str)
10) len("key Ak")
11) len([3, 5, 23, 2, 1]) ⇒ 5

2) UDF (user Define functions)

1) TNRN
2) TSRN
3) TNRS
4) TSRS

Syntax:

def raj(): ⇒ similar like { }
↓
define

1) TNRN

```
def raj():
    print("my name is Raj")
    print("my age is 22")
```

raj() ⇒ calling the function

2) raj(150)

```
def raj(n):
    print("hi receive any...")
```

3)

```
def display():
    a = 5
    b = 3
    return a+b
```

display() ⇒ output nothing

print(display())

Date: / /
Page No.

4) cube (8) ⇒ don't forget to store or print

```
def cube(n):
    return n*n*n
           or
           n**3
a = cube(8)
print(a)
```

HW

1) make a calc using if else and UDF.

i) 2 number from user.

ii) 1 for +, 2 for -
3 for x, 4 for /

iii) show the output

(loop) ⇒ 6 for exit.

Date: / /
Page No.

Types of function arguments.

(* args, **kwargs)
* keyword

1) def display(a, b, c):
       print(a, b, c)

display(5, 2.3, True)

a b c

display(5, 2.3, True, 15)
→ show error.

now what if we store the value all in one variable.

def display(*n):
       print(n) → (5, 2.3..)
                  ↓
                  Store in a tuple format

*n ⇒ x not valid
* args ⇒ all arguments in one parameter.

2) display (sid = I , name = "Raj"
age = 19)

def display (**keyargs)
print (keyargs) →
key and value output in dict //.

print (kwargs.values()) →
dict values ([1, 'Raj', 19])

print (kwargs.keys()) →
dict keys (['sid', 'name', 'age'])

we can get both also single and xx args.

⇒ display (1, 2, 3, 4, sid = I,
name = "Raju")

def display (*args, **kwargs):
print (kwargs)
print (args)

Date: / /
Page No.

docstring ⇒ document string

To check the comment in a function.

```
def display (*args, **kwargs):
    """
    This is a simple fn.
    created for representing
    arguments
    """
    print (args)
    print (kwargs)

display (1,2,3, sid=1, age=22)
print (display.__doc__)
```

global keyword

```
a = 5  # Global area

def display():
    # local area
    print(a)

display()
```

To change the global variable

```
# global area
a = 15

def disploy():
    # local area
    global a
    a = 7
    print("local a =", a)
print("global a =", a)
disploy()
print("global a =", a)
```

New

Date: / /
Page No.

→ Return multiple values → (Tuple unpacking)

```
def hello():
    a = 3
    b = 5
    return a, b

ans = hello()
print(ans)
```

Now what if we want to ~~write~~ return the values seprately.

return a ✗
return b

return a, b ✓ (Tuple unpacking)

output (3, 5) → In a tuple format.

But i want both the value in a seprate.

```
ans1, ans2 = hello()
print(ans1, ans2)
```

Date: / /
Page No.

⇒ Function Recursion ⇒ f^n call itself

factorial

n = int(input("Enter any num: "))

fact(n)

def fact(a):

Condition

5 ⇒ 5 x 4 x 3 x 2 x 1 ⇒ 120

1 ⇒ 1 & 0 ⇒ 1 & -5 ⇒ 1

if a <= 1:
    return 1
else:
    return a x fact(a-1)

⇒ lambda / Anonymous function

f^n without name.

def cube(n):
    return n**3
print(cube(5))

lambda argum: expression

many ↓ only 1

Date: / /
Page No.

```
ans = lambda n : n x x 3
print (ans(3))
```

Anonymous fn

```
add = lambda a, b : a+b
print (add(5,3))
```

OR. Tuff

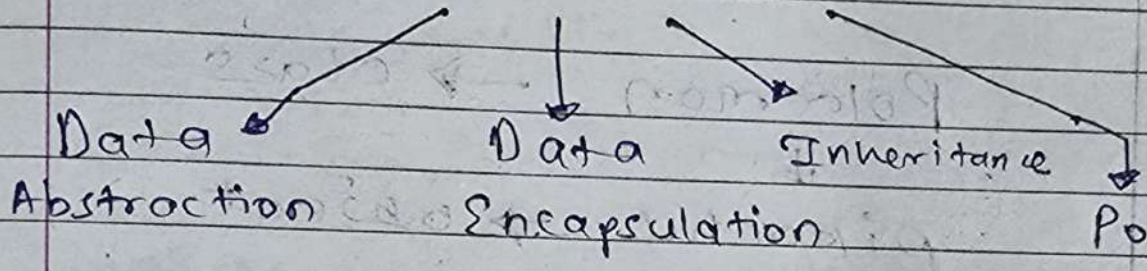
```
def Eqetme (n) : #n=4
    return lambda a : a x n

ans = getme (4) #lambda a : a x n
print (ans(2))
```

⇒ Nested function.

OOP

New way to program (Advance)

OOP (class & object)



1) Data Abstraction
   i) Access Modifiers: Public, Private, Protected
   ii) Abstract class
   iii) interface.

2) Data Encapsulation: Setter & getter, constructor, Destructor

3) inheritance.
   i) single / simple inheritance
   ii) multi level inheritance
   iii) multiple inheritance
   iv) heirarchical inheritance
   v) 'Hybrid' inheritance

4) Polymorphism: method overloading, method overriding, operator overloading

→ structure

Date: / /
Page No.

1) class | object (Follow the structure)
car | Audi Bmw Volvo

2) Polcemon → class
↓
pikachu → object
↓
Name: Pikachu
Type: Electric
Health: 20
→ Attributes

attack()
dodge()
evolve()
→ methods

→
class Car {
Basic
audi = car() ⇒ object
nisson = car() ⇒ object

Aab class k andar kya kya hai
Ao sakta he.

1) Attributes / properties / variables
2) methods / functions
3) Constructor
4) destructor.

Class Example.

Date: / /
Page No.

①
```
class Pokemon:
    name = None
    ptype = None
    health = None

pikachu = Pokemon()  ⟹ object

pikachu.name = "pikachu"
pikachu.ptype = "Electric"
pikachu.health = 70.

print("Name ⇒" pikachu.name)
```

②
```
class student:
    sid = None
    name = None
    age = None

s1 = student()
s2 = student()

s1.sid = int(input("Enter id"))
s1.name = input("Enter stud name")
s1.age = int(input("Enter age"))

same for s2 //.

print(f"ID = {s1.sid}
      name = {s1.name}")
```

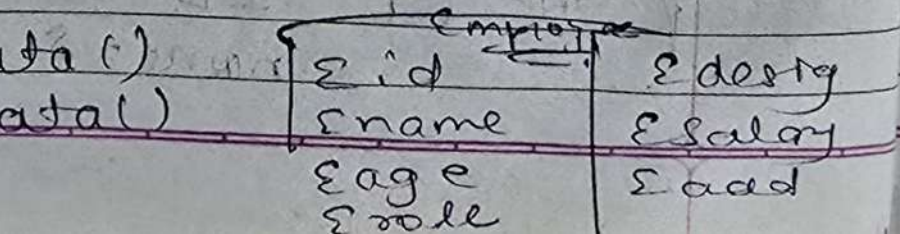
Encapsulation

Date: / /
Page No.

(3) function inside class

compulsory pass parameter in method.

```
Class Student:
    sid = None
    name = None
    age = None

## setter
    def setData(self, i, n, a)
        self.sid = i
        self.name = n
        self.age = a

## getter
    def getData(self):
        print(self.sid,
              self.name,
              self.age)

S1 = Student()
S2 = Student()

S1.setData(1, "Nehul", 20)
S2.setData(2, "Raj", 23)

S1.getData()
S1.getData()
```



Date: / /
Page No.

⇒ Array of objects.

```
class Student:
    name = None
    age = None

    def set(self):
        self.name = input("Enter name")
        self.age = input("Enter age")

    def get(self):
        print(f"Name = {self.name}
        & Age = {self.age}")

obj = []

n = int(input("how many student"))
for i in range(n):
    obj.append(student())

for i in range(len(obj)):
    obj[i].set()

for i in range(len(obj)):
    obj[i].get()
```

Date: / /
Page No.

→ Constructor ⇒ block of code that is automatically invoked when a class is instantiated

```
class City:
    name = None ⇒ no need to write

def __init__ (self):
    self.name = input ("Enter name")

def get (self):
    print ("Name ⇒ ", self.name)
```

```
c1 = city()
c2 = city()
```

```
c1.get()
c2.get()
```

⇒ Reflection enabling functions
1) type () → print (type (a))
2) dir () → directory
3) callable ()
4) isinstance ()
5) getattr ()

* 1) dir ⇒ return, properties or methods, which we can perform.

```
a = [3, 5, 2]
print (dir (a)) ⇒ retrives built in methods
```

Destructor def __del__(self)
print("obj deleted")

del c1
del c2

we can retrive data from a class also.

```
class Student:
    name = Name
    age = age

    def getData(self):
        print(f"{self.name} &
                {self.age}")

s1 = student()
print(dir(s1))
```

⇒ callable ⇒ True or false

```
def hi():
    print("hello")

hi()
print(callable(hi))
```

True //.

```
x = 8
print(callable(x))
```

false //.

isinstance("hello", (float, int, list, dict, tup))

⇒ isinstance
↳ object
also known as instance

```
class Student:
    name = None

    def getData(self):
        print(f"{self.name}")

# object / instance

s1 = student()
print(isinstance(s1, student))
```

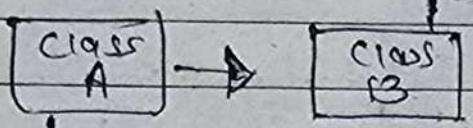
⇒ getattr() ⇒ suppose we want to access a attr inside a class.

```
class Student:
    name = None

    def getData(self)
        print(f"{this.name}")

# objects / instance

s1 = student()

s1.name = "Akshay"
        OR.
Print(getAttr(s1, "name"))
```

↳ in a string format

Inheritance and its types

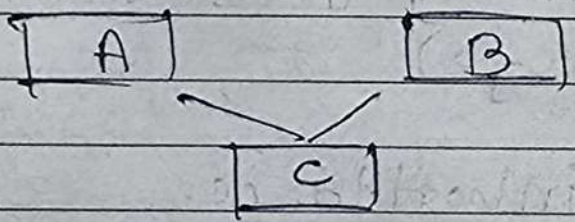
1) single / simple Inheritance



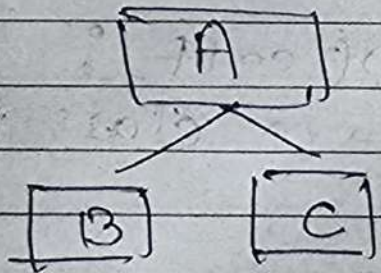
2) multilevel Inheritance.

A → B → C

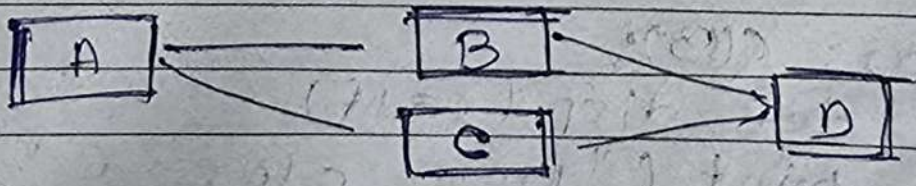
3) multiple Inheritance.



4) Hierarchical inheritance



5) Hybrid Inheritance



Always make the obj of child class

Date: / /
Page No.

1) Single Inheritance

```
class A:
    name = "Rahul"
    school = "AK school"

class B(A):
    age = 18
    pass

o1 = B()
print(o1.name, o1.school,
      o1.age)
```

2) multilevel inheritance

```
class A:
    def hello(self):
        print("from class A")

class B(A):
    def get(self):
        print("from class B")

class C(B):
    def disp(self)
        print("from class C")

o1 = C()
o1.disp()
o1.get()
o1.hello()
```

Date: / /
Page No.

3) multipe Inheritance. (countop)

```
class A:
    a = 5

class B:
    b = 3

class C(A, B):
    pass

o1 = C()
print(o1.a, o1.b)
```

Ab agar dono class me same method ho to konsa print karega.

```
class A:
    def iron(self)
        print("Iron from A")

class B:
    def iron(self)
        print("Iron from B")

class C(A, B)
    pass

o1 = C()
o1.iron() → print iron A
```

4) Hearchical Inheritance → Tree inheritance.

```
class Movie:
    update = "oscar"

class Bollywood (Movie):
    b = 150

class hollywood (Movie):
    h = 250
```

```
b1 = Bollywood()
print (b1.b)
print (b1.update)
```

```
h1 = hollywood()
print (h1.h,
       h1.update).
```

5) Hybrid Inheritance

```
class A:
    top = 10

class B (A):
    b = 1

class C (A):
    c = 2

class D (B, C):
    pass

o1 = D()
print (o1.top, o1.b, o1.c)
```

Date: / /
Page No.

Super function

```
class A:
    a = 5
    def getA(self):
        print("from class A")

class B(A):
    def getB(self):
        print("from class B")
        super().top()
        print(super().a)

class C(B):
    def getC(self):
        print("from class C")
        super().getB()

o1 = C()
o1.getC()
o1.getB()
o1.getA()
```

* issubclass ⇒ check the subclass is true or not.

child → parent

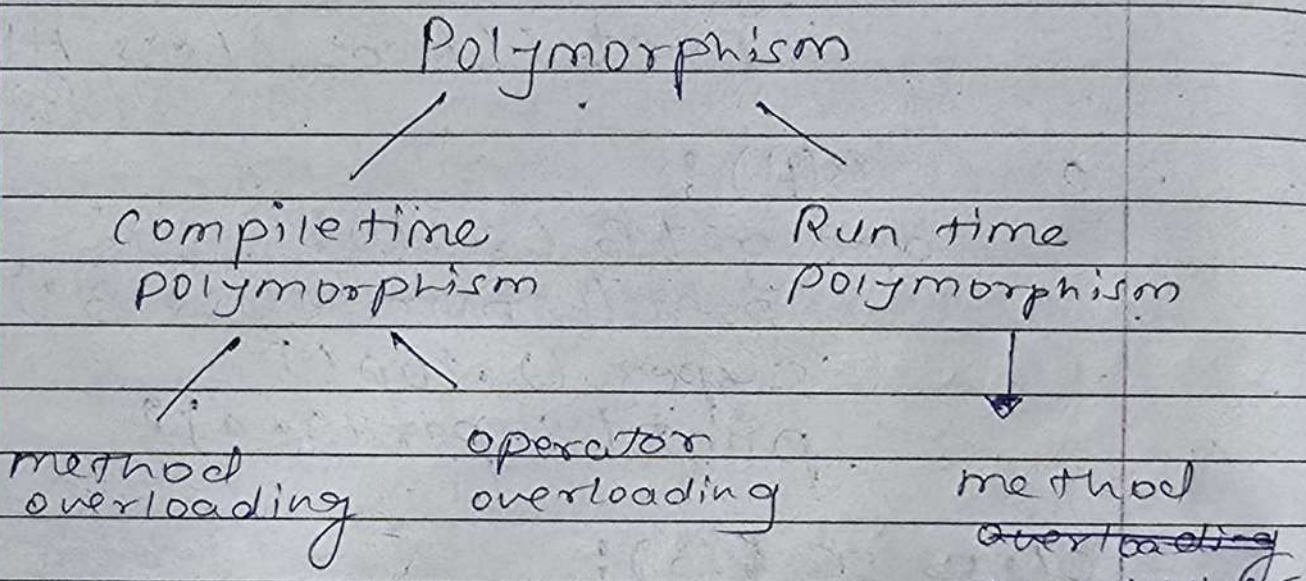
```
print(issubclass(C, B))
```

output → true//

Date: / /
Page No.

Polymorphism

poly → many
morphism → behaviour



1. Method overloading.

1) class ⇒ multiple methods with same names, but arguments different.

Example:

```
class A:
    def truck(self)
    print("empty truck is running").

    def truck(self,n):
    print("truck is running with",n)
```

```
O1 = A()
O1.truck()
O1.truck(30)
```

DO not support method overloading ⇒ override ✓

a=5
a=3 ✓

Date: / /
Page No.

2) Method Overriding ⇒ multiple class multiple methods as same names, some arguments. compulsory Inheritance.

```
class India:
    def Person(self):
        print("Myself Rahul.")
        print("I am in India.")

class Pak(India):
    def person(self):
        print("Myself Rahul.")
        print("I am in Pakistan.")

o1 = Pak()
o1.person()
```

Date: / /
Page No.

Built in dunder Methods (magic methods)

```
class A:
    pass

o1 = A()
print (dir(o1)) ⇒ get all dunder methods
```

Now printing the obj not the directory

```
print (o1) ⇒ provide memory address.
```

⇒ now to get rid of that weird memory address

```
class A:                    → representation
    def __repr__(self):
        return "object printed"

o1 = A()                    → same as
print (o1)                      def __str__(self):
                                    return "obj printed"
```

So we will get the difference in shell.

```
o1 = A()        → on print str
print (o1) ⇒ str.        ↑ on direct repr.
o1 ⇒ enter ⇒ object repr.
```

Date: / /
Page No.

operator overloading

class Complex:

```
def __init__(self, n1, n2):
    self.x = n1
    self.y = n2

def __add__(self, n):
    p = self.x + n.x
    q = self.y + n.y
    return Complex(p, q)

def getData(self):
    print(f"x = {self.x} &
          y = {self.y}")

c1 = complex(5, 3)
c2 = complex(1, 6)

c1.getData()
c2.getData()

print(c1 + c2) # c1.__add__(c2)

c3 = c1 + c2
c3.getData()
```

Exception

Date: / /
Page No.

2 ⇒ Error 1) syntax Error
2) System error.

⇓

To handle the system error we have try, except, else and finally keywords.

1)
```
a = 5
b = 0
c = a / b
print(c) ⇒ zeroDivisionError //.
```

now we will try to solve it using try and except

```
try:
    c = a/b
    print(c)
except ZeroDivisionError:
        print("Not possible").
```

2) x = (y) + (z) ⇒ not define.
nameError: y is not defined.

```
except NameError:
    print("object is not defined.")
```

3) '2' + 2 ⇒ typeError.

```
except TypeError as e:
    Print("not possible", e)
```

If we dont want to handle all the different Exception then simply write

```
try:
    // statement.
except Exception:
    print("Not possible")
```

or

```
except:
    print("Not possible")
```

now else & finally keyword.

```
a = 5
b = 3

try:
    c = a/b
    print(c)
except:
    print("Not possible")
else: → Execute for try
    print("sucess")
finally:
    print("always executes")
```