Python for Data Science

Page No.
Date

Python Basics

→ Very Simple Syntax. Easy to learn for beginners.

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

General Purpose language

- Console Applications and scripts
- Desktop Applications
- Web Applications
- Game Development
- Machine learning, Deep learning, AI, Big Data, IOT, etc

Multiparadigm Support

- Procedural Style Programming like C
- Object Oriented Programming like Java
- Functional Programming like Lisp

Page No.
Date

Portable or Platform Independent

Programs are typically first compiled into an intermediate code, then the code is run by the interpreter.

Dynamically Typed:

```
n = 10
y = "geeks"
n = "python"
```

dynamically typed lang. are slower (Python) than statically typed languages (Java & C++)

More Chances of run time Error (assigned at run time)
In Python.

In C++ & Java (value assigned at compile time)

Automatic Garabage Collection:

In Java & Python, you don't have to release the dynamically allocated memory, but you have to in C++.

Page No.
Date

Popular app. built in Python

Youtube, Dropbox, Reddit, Quora, Instagram, Mozilla Firefox, Netflix

→ print() in Python

function :— A function is a set of instruct that takes some parameters from you, some input from you, do some work on those parameters and produce some output.

```
print ("Hello")
print ("Welcome", "to", "bfbr")
print ()
print ("Hope you are enjoying Python")
```

Output :
Hello
Welcome to bfbr

Hope you are enjoying Python.

end and sep in print()

```
print ("Welcome", end = " ")
print ("to GFG")
print ("25", "08", "2020", sep = "-")
print (10, 20, 30, sep = "+", end = " ")
print (40, 50)
```

Output:
```
Welcome to GFG
25-08-2020
10+20+30 40 50
```

→ Variables in Python

```
age = 38
name = "Ankit"
weight = 58.5
```

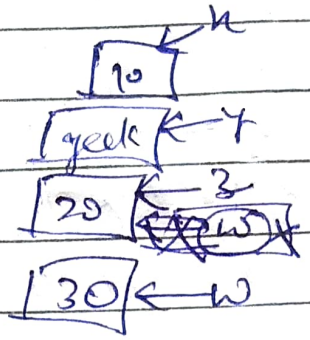
Output:
```
38
Ankit
58.5
```

```
print (age)
print (name)
print (weight)
```

Page No.
Date

How Variable Work ?

```
x = 10
y = "geek"
z = 2
w = z
w = 30
print (x, y, z, w)
```



Output :

10 geek 2 30

Python is Dynamically Typed

```
x = 10
print (x)
x = "geek"
print (x)
```

output :

10
geek

Example Program

```
is_valid = True
marks = 90
pi = 3.14
city_name = "Noida"
print (is_valid)
print (marks)
print (pi)
print (city_name)
```

output :

True
90
3.14
Noida

using a variable without assigning, it causes Error.

```
print (n)
```

output:

NameError : name 'n' is not defined

```
n = None
print (n)
```

output:

None

→ input() in Python

- ```
  name = input ("Enter your name : ")
  print ("welcome" + name)
  ```

output:

>> Enter your name : Sandeep
>> Welcome Sandeep

- ```
  name = input ("Enter your name : ")
  age = input ("Enter your age : ")
  print ("Welcome" + name)
  print ("your age is" + age)
  ```

Output:

Page No.
Date

Python Program for addition of two input numbers -

```
x = int (input ())
y = int (input ())
res = x + y.
print (sum)
```

→ Type () in Python

- Examples of numeric type.

```
a = 10
print (type (a))
b = 10.5
print (type (b))
c = 2 + 4j
print (type (c))
```

output :

```
<class 'int'>
<class 'float'>
<class 'complex'>
```

- Example of none & bool type.

```
a = True
print (type (a))
b = None
print (type (b))
```

output :

```
<class 'bool'>
<class 'NoneType'>
```

• Example Program for Sequence Types

```
str = "gfg"  # String
print(type(str))

l = [10, 20, 30]  # List
print(type(l))

t = (10, 20, 30)  # Tuple
print(type(t))

s = {10, 20, 30}  # Set
print(type(s))

d = {10: "gfg", 20: "ide"}  # Dict
print(type(d))
```

Output :

```
<class, 'str'>
<class, 'list'>
<class, 'Tuple'>
<class, 'set'>
<class, 'Dict'>
```

Page No.
Date

→ Type Conversion in Python

- Implicit Type Conversion
- Explicit Type Conversion

Implicit Type Conversion

```
a = 10
b = 1.5
c = a + b
print (c)
d = True
~~print~~ e = a + d
print (e)
```

Output :

11.5
11

Explicit Type Conversion

```
S = "geeks"
print (list (S))
print (tuple (S))
print (set (S))
```

Output :

2

['g', 'e', 'e', 'k', 's']
('g', 'e', 'e', 'k', 's')
{'e', 'g', 'k', 's'}

•3

```
l = ['a', 'b', 'c']
print (str (l))
a = 10
b = 11
print (str (a) + str (b))
c = 12.5
print (str (c))
```

output:

['a', 'b', 'c']

10 11

12.5

•4

```
t = (10, 20, 30)
print (list (t))
s = {10, 20, 30}
print (list (s))
```

output

[10, 20, 30]

[10, 20, 30]

•5

```
a = 20
print (bin (a))
print (hex (a))
print (oct (a))
```

	Quotient	Remainder
20/2	10	0
10/2	5	0
5/2	2	1
2/2	1	0
1/2	0	1

10100

output : 0b10100

0x14

0o24

Octal group in 3: 010 100 → 2 4

Hex group in 4: 0001 0100 → 1 4 = 14

$\frac{2^4+2^1+2^0}{4}$

$\frac{2^2+2^1+2^0}{2}$

* Comments in Python :-

- # Comments starts with hashtag
  # And they are simply ignored by interpreter.
- """
  This is a docstring
  not a multiline comment
  """ And they are proccessed by
  interpreter.
  """

Date
Page

* if, else and elif in Python

- Nested if-else

Ex.

```
n = int(input())
if n>0:
    print("Positive", end=" ")
    if n%2==0:
        print("Even")
    else:
        print("Odd")
elif n<0:
    print("Negative", end=" ")
    if n%2==0:
        print("Even")
    else:
        print("Odd")
else:
    print("Zero")
```

↳ Decide if an input no is :-
a) Positive Even b) Positive Odd c) -ve Even
d) -ve Odd e) Zero.

Operators

* Arithmetic Operators in python

```
x=9                    O/P:-
y=4
print(x+y)             13
print(x-y)             5
print(x*y)             36
print(x/y)             2.25
print(x//y)            2
print(x%y)             1
print(x**y)            6561
```

(//) → floor division: It simply ignore the no after decimal if it positive int or neg in case of negative int it gives you a negative smaller no ignore decimal value

ex
```
floor(3.2) = 3
floor(5.9) = 5
floor(-3.2) = -4   } small no.
floor(-5.9) = -6
```

Date
Page

```
x=-5
y=2
print(x//y)
x=5.0
y=2
print(x//y)
x=2
y=-2
print(x**y)
```

O/P:-
-3
2.0
0.25

- Arithmetic Precedence :- Operators

Help in for evaluating which operator is evaluate first.

+ -
* / //
**

↓ Precedence (Increasing)

Ex
```
print(5+2*3)
print(5+3*4**2)
```

O/P :-
11
53

Date
Page

- Arithmetic operators Associativity

Helpin to decide the Evaluation oder, when you have two or more operators of same precedence in an expression.

- `+ -` Left to Right
- `* / //` Left to Right
- `**` Right to Left

- ```
  print (5+4-2)
  print (2**2 ** -1)
  print ((2**2) ** -1)
  ```

Output:

```
7
1.4142135623730951
0.25
```
```

Date
Page

* Logical Operators :-

O/P :-

- a = 10
  b = 20
  c = 30
  print(a < b and b < c) — True
  print(a < b or b > c) — True
  print(not a > b) — True

– Logical Operators Can be use to give non boolean values :-

- Expresian that are treated as false: None, Empty string, list, tuple, dictionary, etc.

s1 = " "
s2 = s1 or "DefaultStr"
print(s2)

O/P :-
DefaultStr.

- x = 10
  print(x or 20) — 10
  y = 0
  print(y or 30) — 30
  z = 40
  print(z or 50) — 50

Date
Page

* Identity Comparision Operators in Python :-

- There are two main identity comparision operators :-

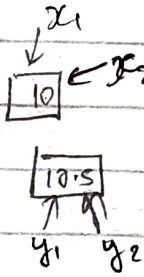
is

is not.

is : True, if the operands are identical means reffering to same memory location

is not : True, if the operands are not identical means reffring diff - diff memory location.

Ex




```
x1 = 10                    O/P
x2 = 10                    True
y1 = 10.5                  True
y2 = 10.5                  True
z1 = "geeks for geeks"
z2 = "geeks for geeks"
print(x1 is x2)
print(y1 is y2)
print(z1 is z2)
```

Date
Page

ex
```
l1 = [10, 20, 30]
l2 = [10, 20, 30]
print (l1 is l2)
```

O/P :- false

is operators true, for in case of literals but for collection & Containers it give always false even if they have same value.

* Membership test operators :-

in : True if value / variable is found in the sequence.

not in : True if value/variable is not found in the sequence.

string : check for substring.

Dictionary : check for key.

List / Set / Tuple etc : checks for Membership.

Date
Page

ex

```
s = "geeksforgeeks"
print("g" in s)
print("for" in s)
print("gk" in s)
```

O/P
True
True
False

ex

```
d = {10: "abc", 20: "efg"}
print(10 in d)
print(15 in d)
print("abc" in d)
```

O/P
True
false
false

ex

```
l = [10, 20, 30, 15]
print(30 in l)
print([20,30] in l)
```

O/P
True
false

ex

```
l = [10, 20, 30, 15]
print(30 not in l)
print(40 not in l)
print([20,30] not in l)
```

O/P
false
True
True

Date
Page

* Bitwise Operators in Python :-

eg
```
print(bin(18))
print(bin(12))
print(int("0b10010",2))
print(int("0b1100",2))
```

Output
```
0b10010
0b1100
18
12
```

- **Bitwise AND : &**

ex
```
x=3
y=6
print(x&y)
```

o/p
2

6 : 110
3 : 011
3&6 : 010

Decimal of 010 = 2

- **Bitwise OR : |**

o/p
7

```
x=3
y=6
print(x|y)
```

6 : 110
3 : 011
3|6 : 111

Decimal of 111 = 7

- **Bitwise XOR : ^**

```
x=3                    O/P
y=6                    5
print(x^y)
```

^ operator give True if both bits are diff. otherwise gives false.

- **Left Shift Operators :-**

ex

```
x=5                    O/P
print (x<<1)           10
print (x<<2)           20
print(x<<3)            40
```

x=5

Binary Representation of 5 : 101

(x<<1) = 10 (1010)

(x<<2) = 20 (10100)

(x<<3) = 40 (101000)

formula = $x * 2^n$

Date
Page

- **Right Shift Operators :-**

```
x=5                    O/P
print(x >> 1)          2
print(x >>2)           1
print(x >>3)           0
```

formula :-

$$\left\lfloor \frac{x}{2^n} \right\rfloor$$

floor.

5 :- 101
x >> 1 : 2 (010)
x >>2 : 1 (001)
x >>3 : 0 (000)

- **Bitwise Not : ~**

```
x=5                    O/P
print(~x)              -6
```

5 :- 00.....0101
Bitwise Not of 00.....0101
:- 11.....1010
(This is 2's complement of 6)

6 :- 0000...0110
1's comp :- 1111 ... 1001
+1
2's Comp :- 1111...1010 ← -6

Date
Page

* AP:-

Ap is a sequence of no in order, in which the difference b/w any two consecutive no is a constant value.

$a_n = a + (n-1)d$ ← nth Term

a → First Term, d → Common diff

$S_n = \frac{n}{2}(2a + (n-1)*d)$

$a + a + (n-1)\,d$

$a + l = \text{last term}$

AP:- $a, a+d, a+2d \ldots\ldots a+(n-1)d$

* GP:-

$a, ar, ar^2, ar^3 \ldots\ldots ar^{(n-1)}$

$a_n = ar^{n-1}$ ← nth term of GP.

r:- is common ration.

a: first term

n: Total no of terms.

Date
Page

finding last digit of -ve no through, dividing no by '%10' gives you diff result, So change the no to +ve for ease.

Day Before 'n Days' :-

I/P :- d = 1
n = 1

O/P :- 0 // Sunday

I/P :- d = 0
n = 9

O/P :- 5 // Friday

d	Day.
0	Sun
1	Mon
2	Tue
3	Wed
4	Thru
5	Frid
6	Sat

ex
```
d = int(input("Enter d : "))
n = int(input("Enter n : "))
print((d-n)%.7)
```

Output : Enter d : 0
Enter n : 9
5

(explaination
→ (0-9)%.7
(5 - 2*7)%.7
↑ Next multiple of 7 Greater than 5

cuz → Treating -ve diff bython did this past

Date
Page

Loops

* Application of Loops :-

- Doing some work repeatedly.
- Traversing through collections like lists, tuples, sets, etc.
- Running Services in Systems.

* While loops in Python :-

```
while condition_test:
    Statement 1
    Statement 2
    Statement 3
    i = i+1
```

* range() in Python :-

range(x) : 0, 1, ---- x-1
range(x, y) : x, x+1, ---- y-1
range(x, y, z) : x, x+z, x+2z ----

Ex

```
r = range(5)
print(r)
l = list(r)
print(l)
print(type(r))
l = range(10,20)
k = list(l)
print(k)
r = range(10,20,3)
l = list(r)
print(l)
r = range(20,10,-3)
l = list(r)
print(l)
```

O/P

```
range(0,5)
[0, 1, 2, 3, 4]
<class 'range'>
[10, 11 ..... 19]
[10, 13, 16, 19]
[20, 17, 14, 11].
```

* For Loop in Python:-

```
for x in seq:
    State ... 1
    State ... 2
    State ... 3
    State ... 4
```

Here, seq can be list, string, tuple, range, ...

Ex
```
l = [10, 20, 80, 40]
for x in l:
    print(x)
```

O/P
10
20
30
40

Ex
```
s = "gfg"
for x in s:
    print(x)
```

O/P
g
f
g

Ex
```
s = "gfg"
for x in s:
    print(x)
    print(x)
```

O/P
g
g
f
f
g
g

Ex
```
for x in range(5):
    print(x)
```

O/P
0
1
2
3
4

Ex
```
for x in range(20):
    if x % 6 == 0:
        print(x)
```

O/P:-
0
6
12
18

Ex
```
l = [10, 20, 30, 40]
for i in range(len(l)):
    print(i, l[i])
```

O/P
0 10
1 20
2 30
3 40

Date
Page

* Break in Python:-

n = int(input())
Find the smallest divisor of a no such that the divisor is greater than 1.

ex
```
n = int(input())
for x in range(2, n+1):
    if(n%x == 0):
        print(x)
        break
```

O/P
15
3

* Continue in Python:-

```
l = [10, 16, 17, 18, 9, 15, 21, 13]
for x in l:
    if x%5 == 0:
        continue
    if x%7 == 0:
        break
    print(x)
print("Bye")
```

O/P :-
16
17
18
9
Bye

Date
Page

* Nested Loop in Python

O/P

eg for i in range(1,3):
    j=1
    while(j<3):
        print(i,j)
        j=j+1
    print("GfG")

1 1
1 2
GfG
2 1
2 2
GfG

eg ll=[[10,20,30],[40,50,60],[70,80]]
for l in ll:
    for x in l:
        print(x, end=" ")
    print()

O/P :-

10 20 30
40 50 60
70 80

11/09/2023

Page No.
Date

STRINGS IN PYTHON

→ Function

→ Default arguments & keyword arguments

```
def myfun(n, y=80):
    print("n", y=50):
    print(n)
    print(y)
```

o/p
10
50

```
myfun(10)
myfun(y=60, n=10)
```

of
60
10

→ Variable length arguments :-

We can pass a variable no. of arg. to a fun.

Two types of variable-length arguments

- *args (Non-keyword arguments)
- **kwargs (keyword Arguments)

Page No.
Date

Ex

```
def fun (*args):
    for arg in args:
        print (arg)
fun (20, 30, 40, 50)
```

Op
20
30
40
50

→ Docstring :-

Usually docString is ~~called to describe fun~~ initialized under function to describe it.

By

- To print docstring Syntax :-

~~print (function name.doc)~~

print (function name.__doc__)

→ Pass by value or pass by reference

In Python every variable is a reference. When we pass variable to ~~object~~ a function a new reference to the object is created.

Strings in Python

- Python has Unicode sys:-

Characters 'A' to 'Z'

⇓

65 to 90

'a' to 'z'

97 to 122

```
print(ord("a"))
print(ord("A"))
print(chr(97))
print(chr(65))
```

O/P

```
97
65
a
A
```

- Strings indexing:-

ex

```
s="geeks"
print(s)
print(s[-2])
print(s[1])
```

O/P

```
geek
k
e
```

g	e	e	k
0	1	2	3
-4	-3	-2	-1

Date
Page

O/P

```
s = "geek"
s[0] = "e"
print(s)
```

Type Error :- as Strings are Immutable.

- Multiline strings :-

O/P

```
s = """Hi,
        This is a Python Course. Hope you are Enjoying it.
    """
print(s)
```

Hi,
This is a Python Course. Hope you are Enjoying it.

→ Escape Sequences and Raw Strings

	\ to print	
Works only with string	\n	new line
	\t	Tab space
	\\	\
	\\n	\n
	\\t	\t

By the help of 'r' & 'R' before the Symbols, we can use string as raw string. That means those strings are not processed by their escape sequences through interpreter.

eg: s1 = "C:\project\name.py"
s2 = r"C:\project\name.py"
print (s1)
print (s2).

op:

C:\project
ame.py
C:\project\name.py

Page No.
Date

→ String operation (7)

S_1 = "geeks for geeks"
S_2 = "geeks"

O/P
True
False

print (S_2 in S_1)
print (S_2 not in S_1)

Substrings are Contiguous

- Concatenation :-

S_1 = "geeks"
S_2 = " for geeks"
print ($S_1 + S_2$)

O/P
geeks for geeks.

- index() & rindex() :-

S_1 = "geeks for geeks"
S_2 = "geeks"
print (S_1.index(S_2))
print (S_1.rindex(S_2))
print (S_1.index(S_2, 0, 3))

output
0
8

They are used to find the first/last pos of Sub-String
In case substring doesn't found, they give Value Error → Imp

12/09/2023

String Operations (Part 2)

```
S1 = "geeks"
print (len(S1))

S2 = S1.upper()
print (S2)
S3 = S2.lower()
print (S3)
print (S1.islower())
print (S2.isupper())
```

Output

```
5
GEEKS
geeks
True
True
```

→ starts with & ends with.

```
S = "Geeks for Geeks Python Course"
print (S.startswith("Geeks"))
print (S.endswith("Course"))
print (S.startswith("Geeks", 1))
print (S.startswith("Geeks", 8, len(S)))
```

O/P

```
True
True
False
True
```

Page No
Date

Split & join

```
S2 = "geeks, for, geeks"
print (S2.split(","))
```

```
l = ["geeks for geeks", "Python", "Course"]
print ("".join(l))
print (",".join(l))
```

['geeks', 'for', 'geeks']

geeks for geeks python Course
geeks for geeks, python, Course

→ strip, rstrip & lstrip

```
S = "--geeks for geeks--"
print (S.strip("-"))
print (S.lstrip("--"))
print (S.rstrip("--"))
```

o/p
geeks for geeks
geeks for geeks--
--geeks for geeks

→ find() Method

```
S1 = "geeks for geeks"
S2 = "geeks"
print (S1.find(S2))
print (S1.find("gfg"))

n = len(S1)
print (S1.find(S2, 1, n))
```

O/P

```
0
-1
10
```

→ String Comparison in Python

```
S1 = "geeks forgeeks"
S2 = "ide"
```

	O/P
print (S1 < S2)	True
print (S1 <= S2)	True
print (S1 > S2)	False
print (S1 >= S2)	False
print (S1 == S2)	False
print (S1 != S2)	True

"abcd" > "abc"
"zAB" > "ABC"
"abc" > "ABC"
"x" > "abcd"

Page No.
Date

→ Pattern Searching in Python

② Input & O/P

I/p : text = "geeks for geeks"
pat = "geeks"

O/p : 0 10

I/p : text = "AAAAA"
pat = "AAA"

O/p : 0 1 2

Source Code

```
text = input ("Enter Text : ")
pat = input ("Enter Pattern : ")
pos = text.find (pat)
while pos >= 0 :
    print (pos)
    pos = text.find (pat, pos+1)
```

O/P

Enter Text : geeks for geeks
Enter pattern : geeks
pos = 0

1st Iteration :
print (0)
pos = 10

2nd Iteration
print (10)
pos = -1

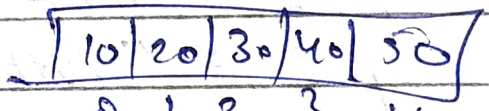
DATA STRUCTURE

→ List Introduction

```
→ l = [10, 20, 30, 40, 50]
print (l)
print (l[3])
print (l[-1])
print (l[-2])
```

O/P

```
[10, 20, 30, 40, 50]
40
50
40
```



→ Insert & Search functions

```
l = [10, 20, 30, 40, 50]
l.append (30)
print (l)
l.insert (1, 15)
print (l)
print ('15' in l)
print (l.count (30))
print (l.index (30))
print (l.index (30, 4, 7)
```

O/P

```
[10, 20, 30, 40, 50, 30]
[10, 15, 20, 30, 40, 50, 30]
True
2
3
6
```

→ Removal of items.

l = [10, 20, 30, 40, 50, 60, 70, 80]

	O/P
l.remove(20)	
print(l)	[10, 30, 40, 50, 60, 70, 80]
print(l.pop())	80
print(l)	[10, 30, 40, 50, 60, 70]
print(l.pop(2))	40
print(l)	[10, 30, 50, 60, 70]
del l[1]	[10, 50, 60, 70]
print(l)	[60, 70]
del l[0:2]	
print(l)	

→ Inbuilt Purpose functions

l = [10, 40, 20, 50]

	O/P
print(max(l))	50
print(min(l))	10
print(sum(l))	120
l.reverse()	[50, 20, 40, 10]
print(l)	[10, 20, 40, 50]
l.sort()	
print(l)	

If numbers are replace with string then result is evaluated using lexographically.

→ Tuples in Python

- Tuples are similar to list in many ways
- ~~differences~~ Tuples are.
  - → Immutable / faster than List
  - → They are ordered & fixed.

Ex
```
t = (10, 20, "geek")
print (t)
t = ()
print (type (t))
print (t)
t = (10)
print (type (t))
t = (10,)
print (type (t))
```

O/P
```
(10, 20, "geek")
<class, 'tuple'>
()
<class, 'int'>
<class, 'tuple'>
```

Ex
```
t = 10, 20, 30, 40, 10
print (t [2])
print (t [-1])
print (t [1:3])
print (len (t))
print (t.count(10))
print (t.index (20))
```

O/P
```
30
10
(20, 30)
5
2
1
```

Some of the functions which modify List are not ~~Implemented~~ Implemented in tuple.

Page No.
Date

Set in Python

Set in Python

- Distinct Elements
- Unordered, Mutable
- No Indexing
- Union, Intersection, Set Difference, etc are fast (Hashing)
- Uses Hashing Internally.
- ~~Mutable~~

Set In Python

ex

```
S1 = {10, 20, 30}
print (S1)
S2 = set([20, 30, 40])
print (S2)
S3 = {}
print (type (S3))
S4 = set ()
print (type (S4))
print (S4)
```

O/P

```
{10, 20, 30}
{40, 20, 30}
<Class, 'dict'>
<Class, 'set'>
set ()
```

→ Set Insertion

O/P

```
S = {10, 20}                    {10, 20, 30}
S.add (30)                      {10, 20, 30}
print (S)                       {40, 10, 30, 20, 30}
S.add (30)                      {20, 40, 10, 80, 80, 20, 90, 60
print (S)                                    30}
S.update ([40, 50])
print (S)
S.update ({60, 70}, [80, 90])
print (S)
```

→ Removal Operation

O/P

```
S = {10, 30, 20, 40}            {40, 10, 20}
S.discard (30)                  {40, 10}
print (S)
S.remove (20)                   set()
print (S)
S.clear ()
print (S)
S.add (50)
del S
```

'discard' works silently while 'remove' will ~~through~~ throw 'Error' if the element is not present in the set.

Page No.
Date

'clear()' will give you the empty set (set()) out of a given set while 'del' will delete completely the set.

ex:

```
S1 = {2,4,6,8}                    o/p: {2,3,4,6,8,9}
S2 = {3,6,9}      | Alternatives  | {6}
print(S1, S2)     | S1.union(S2)  | {2,4,8,3,9}
print(S1 & S2)    | S1.intersect(S2) | {2,4,8}
print(S1 - S2)    | S1.differ(S2) | {2,3,4,8,9}
print(S1 ^ S2)    | S1.symmetric_diff(S2)
```

~~Alternatives~~

ex: S1 = {2,4,6,8}
S2 = {4,8}

print(S1.isdisjoint(S2)) → False → Common
                         → True → Nothing common
print(S1 <= S2) → S1 is subset of S2
print(S1 < S2) → S1 is proper subset of S2
print(S1 >= S2) → S1 is superset of S2
print(S1 > S2) → S1 is proper superset of S2

o/p:
False
False
False
True
True.

13/04/2023

Page No. | Date

→ Dictionaries in Python

- Similar to sets
  - → Collection of key-value pairs
  - → unordered
  - → All keys must be distinct
  - → values may be repeated
  - → uses Hashing internally

ex: d = {110: "xyz", 101: "abc", 105: "bcd", 104: "abc"}

known as associative arrays as well, bcoz we can access the item by key values.

We can access and add a key by using Square brackets

print(d[125]) → It will give Error if key is not present

print(d.get(125)) → It will give none and not Error.

ex: d = {110: "abc", 101: "xyz", 105: "pqr"}

```
print(d.get(101))
print(d.get(125))
print(d.get(125, 'NA'))
```

OR

```
if 125 in d:
    print(d[125])
else:
    print("NA")
```

O/P

xyz
None
NA
NA

Page No.
Date

CX13

d = {110: "abc", 101: "xyz", 105: "pqr", 106: "bcd"}

O/P

```
d[101] = "wny"                    {110: "abc", 101: "wny", 105: "pqr", 106: "bcd"}
print(len(d))                     4
print(d)                          {110: "abc", 101: "wny", 105: "pqr", 106: "bcd"}
print(d.pop(105))                 pqr
print(d)                          {110: "abc", 101: "wny", 106: "bcd"}
del d[106]                        {110: 'abc', 101: 'wny'}
print(d)                          {110: 'abc', 101: 'wny', 108: 'cde'}
d[108] = "cde"                    ('108', 'cde')
print(d.popitem())
```

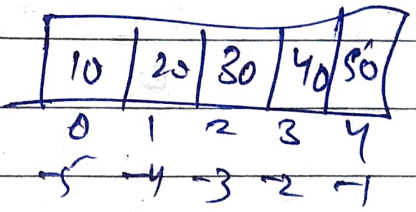
'popitem()' will remove the last inserted key-value and print its tuple

→ Slicing in Python (List, tuple, string)

l = [10, 20, 30, 40, 50]

print(l[0:5:2])

output : [10, 30, 50]



l[start : stop : step] : l[start], l[start+step], l[start + 2*step] ...

stop not included (stop - 1 before stop)

Page No.
Date

Ex:

```
l = [10, 20, 30, 40, 50]
print(l[-1 : -6 : -1])
print(l[::-1])
```

10	20	30	40	50
0	1	2	3	4
-5	-4	-3	-2	-1

output: [50, 40, 30, 20, 10]
[50, 40, 30, 20, 10]

print(l[1 : -1]) → gives you reverse list,

print(l[:]) → gives you whole list

Ex:

```
l1 = [10, 20, 30]
l2 = l1[:]
```

?/1

False

```
t1 = (10, 20, 30)
t2 = t1[:]
```

True

```
S1 = "geeks"
S2 = S1[:]
print(l1 is l2)
print(t1 is t2)
print(S1 is S2)
```

True.

Page No.
Date

→ Comprehensions in Python

List & Set Comprehension

```
l1 = [x for x in range(11) if x%2 == 0]
print(l1)
l2 = [x for x in range(11) if x%2 != 0]
print(l2)
```

O/P : [0, 2, 4, 6, 8, 10]
[1, 3, 5, 7, 9]

Q → Elements of list that are smaller than x.

```
def getSmaller(l, n)
    return [e for e in l if e<n]
l = [9, 15, 12, 3, 7, 11]
n = 10
print(getSmaller(l, n))
```

O/P [9, 3, 7]

ex:-

```
def getEvenOdd(l):
    even = [n for n in l if n%2==0]
    odd = [n for n in l if n%2!=0]
    return even, odd

l = [10, 3, 20, 5, 12]
even, odd = getEvenOdd(l)
print(even)
print(odd)
```

o/p [10, 20, 12]
[3, 5]

ex:-

```
s = "geeksforgeeks"
l1 = [n for n in s if n in "aeiou"]
print(l1)

l2 = ["geeks", "ide", "courses", "gfg"]
l3 = [n for n in l2 if n.startswith("g")]
print(l3)

l4 = [n*2 for n in range(6)]
print(l4)
```

O/P
['e', 'e', 'o', 'e', 'e'].
['geeks', 'gfg']
[0, 2, 4, 6, 8, 10]

Set Comprehension

l = [10, 20, 3, 4, 10, 20, 7, 3]
S_1 = {n for n in l if n*2 == 0}
S_2 = {n for n in l if n*2 != 0}

print (S_1)
print (S_2)

O/P {10, 20, 4, 10, 20,}
{3, 7}

Dictionary Comprehension

l = [1, 3, 4, 2, 5]
d_1 = {n: n**3 for n in l}
print (d_1)

d_2 = {n: f"ID{n}" for n in range (5)}
l_2 = [101, 103, 102]
l_3 = ["gfg", "ide", "courses"]

$d_3 = \{ l_2[i] : l_3[i] \text{ for } i \text{ in range } (len(l_2)) \}$

→ A better way ...

$d_3 = dict(zip(d_2, l_3))$

zip function takes many iterable as our attributes and done mapping internally & through that tuple created of (key & value). After that we have made used dict() to make dictionary.

→ Inverting a Dictionary (key becomes value & vice versa)

```
d1 = {101: "gfg", 103: "practice", 102: "ide"}
d2 = {v:k for (k, v) in d1.items()}
print(d2)
```

O/P

{'gfg': 101, 'practice': 103, 'ide': 102}

Page No.
Date

Object Oriented Programming (OOPs)

Classes and Objects : Intro to OOPs

Procedural Programming :-

we break the code into a set of functions and these functions call each other.

Object Oriented Programming :-

We break the code into a set of entities and these entities talk to each other. These entities have data and method.

Classes and Objects

Class : Blueprint
Object : Instance

```
Class Complex:
    def __init__(self, real, imag):
        self.real = real
        self.imag = imag
    def print(self):
        print(str(self.real) + "+i" + str(self.imag))
```

```
def add (self, c):
    self.real += c.real
    self.imag += c.imag

C1 = Complex(10,20)
C1.print()
C2 = Complex(20,30)
C1.add(C2)
C1.print()
```

O/P

10 + i20

30 + i50

Self represents the instance (object) of the class.

Every method should have self parameter

__init__ is used to initialize the objects attributes using constructor.

Page No.
Date

→ Encapsulation

Encapsulation refers to data hiding

The Idea of encapsulation is to hide the Data Members so that you hide the internal representation so that if you later change the representation it becomes very easy for you.

Apart from maintainability, Encapsulation also helps in Consistency.
- Marks (0 ≤ marks ≤ 100)
- Email ID (should contain @ and .)
- URL

Encapsulation is also defined as bundling of data members and methods.

→ Class Instance Attributes

Class Attributes: Shared with all objects

Instance Attributes: Unique to every object.

```
class Employee:
    companyName = "gfg"          # class attribute
    def __init__(self, id):
        self.id = id             # instance attribute

e = Employee(1001)                    O/P
print(e.companyName)                  gfg
print(e.id)                           1001
print(Employee.companyName)           gfg
```

In python, you can add attributes later.

Ex:

```
class Employee:
    companyName = "gfg"
    def __init__(self, id):
        self.id = id
    def fun(self, n):
        self.name = n

e = Employee(1001)                    O/P
e.fun("Sandeep")
print(e.name)                         Sandeep
e.designation = "CEO"                 CEO
print(e.designation)                  Noida
Employee.officeAdd = "Noida"
print(e.officeAdd)
```

Page No.
Date

Instance Attribute is accessed for an Instance, if both class and Instance attributes have same name.

Ex: Class Employee:
    Company = "gfg"
    def __init__(self, id):
        self.id = id
e = Employee(1001)
Employee.officeAdd = "Noida"
e.officeAdd = "NCR"
print(Employee.officeAdd) → Noida
print(e.officeAdd) → NCR

→ Class Members Access

Three Rules

1) In Python, every member is accessible everywhere

```
class Test:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def fun(self):
        print("Hi")
t = Test(10, 20)
print(t.x)
print(t.y)
t.fun()
```

Output:
10
20
Hi

2) When we use underscore before a variable name, we suggest not to use it outside class.

```
Class Test:
    def __init__(self, x, y):
        self._x = x
        self.y = y
    def _fun(self):
        print("Hi")
t = Test(10, 20)
print(t._x)
```

O/P:
10
20
Hi

3) When we use two underscores before a member name, it becomes inaccessible (only when there are no two underscores at the end.)

```
Class Test:
    def __init__(self, x, y):
        self.__x = x
        self.y = y
    def __fun(self):
        print("Hi")
t = Test(10, 20)
print(t.__x)
print(t.y)
t.__fun()
```

O/P:
Error
Error

print(t._Test__x) would work

t._Test__fun() would work

Page No.
Date

In case of '__x' what python internally do is it changes the name to '_Test__x' when you access through this new name you will not get an Error.

→ Private members can be accessed within the class

```
Class Test:
    def __init(self, x):
        self.x = x
        self.__y = 10
    def printTest(self):
        print (self.x)
        print (self.__y)
```

This works fine

```
t = Test(5)          O/P  5
t.printTest()             10
```

→ Decorators

② Properties of functions:

functions are first-class Objects.

A function can have inner functions (functions defined inside it)

Page No.
Date

A Decorator is a function that takes another function as argument and enhances the behaviour of the passed function.

→ Functions assigned to a variable and passed as parameters

```
def fun1():
    print("Inside fun1")
def fun2(f):
    print("Inside fun2")
    f()
```

```
f = fun1
f()
print()
fun2(f)
```

O/P:
Inside fun1
Inside fun2
Inside fun1

→ Function Inside a function

```
def fun2():
    print("Inside fun2")
    def fun1():
        print("Inside fun1")
    fun1()
fun2()
```

O/P
Inside fun2
Inside fun1

Ex of Decorators

```
def decfun(f):
    def innerfun():
        print("Welcome")
        f()
    return innerfun
```

Takes a function as argument and returns an enhanced version of it

```
def fun():
    print("Users")
fun = decfun(fun)
fun()
```

O/P
Welcome
users

Ex of Decorators short Syntax

```
def decFun(f):
    def inner fun():
        print("Welcome")
        f()
    return innerfun
```

```
@decFun
def fun():
    print("User")
fun()
```

O/P
Welcome
user

→ Class Methods and Static Members

→ class method Example to change class Attribute

```
class Employee:
    compName = "gfg"
    def __init__(self, name, age):
        self.name = name
        self.age = age
    @classmethod
    def setCompName(cls, cName):
        cls.compName = cName
```

```
Employee.setCompName("geeksforgeeks")
print(Employee.compName)
e = Employee("Sandeep", 41)
print(e.compName)
```

O/P
geeksforgeeks
geeksforgeeks

→ class method Example to create an Instance

```
from datetime import date
class Employee:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

```
@classmethod
def getfromBirthYear(cls, name, year):
    return cls(name, date.today().year - year)
e = Employee.getfromBirthYear("Sandeep", 1982)
print(e.name)                                   o/p  Sandeep
print(e.age)                                         40
```

→ Static Method Example

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    @staticmethod
    def isAdult(age):
        return (age > 18)
    def printDetails(self):
        print(self.name)
        print(self.age)
        print(Person.isAdult(1))
p = Person("Nikhil", 22)
p.printDetails()
print(Person.isAdult(29))
```

o/p:
Nikhil
22
True
True

print(self.__class__.is_adult()) would also work for print(Person.is_adult())

→ Static method in Python are made by (@staticmethod) (Static method Decorater), Basically, static method are utility functions that do not have access to any ~~functions~~ properties of a class And static method can be called without object for that class, means that they are bound to the class.

→ There are Three Types of methods in Python

- Class method :- You have to pass 'cls'
- Static method :- Nothing to pass.
- Instance method :- You have to pass 'self'

→ Inheritance

Allows us to define a class that inherits all the methods and properties from another class

Page No.
Date

→ Inheritance Uses :-

- Implements is a relationship.
- Code Reusability.
- Method Overriding
- Abstract Classes

Ex:

```
Class Person:
    def __init__(self, id, name):
        self.id = id
        self.name = name

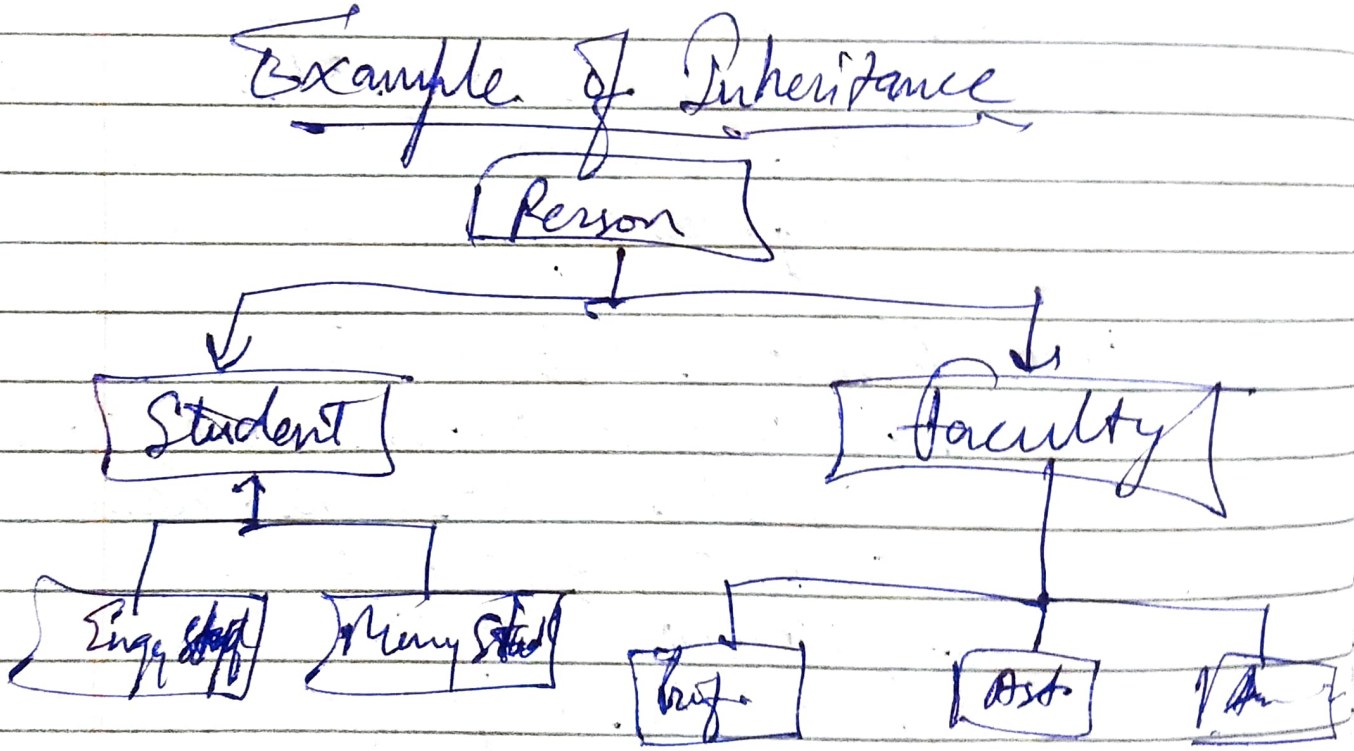
Class Employee(Person):
    def __init__(self, id, name, salary):
        super().__init__(id, name)
        self.salary = salary
    def printdetails(self)
        print(self.id)
        print(self.name)
        print(self.salary)
c = employee(101, "Rahul", 40000)
e.printDetails()
```

Super() method is used to call a method from a parent class. (Where you have doubt use Super)

O/P

101
Rahul
40,000

Example of Inheritance



Object class is the Super-class of all class or root of all classes.

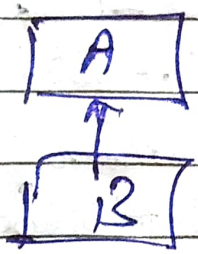
```
class ABC(object):
    ...
class XYZ:
    ...
```

Both have same meaning for you reference. It's only for backward compatibility

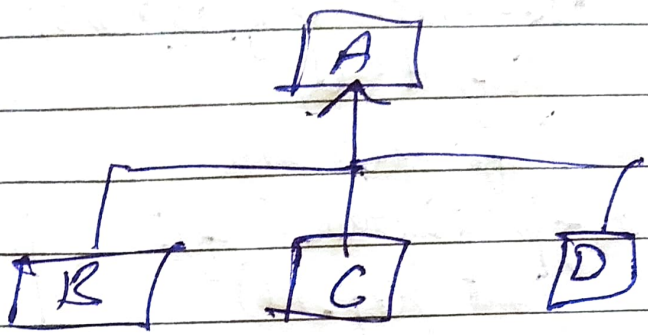
Page No.
Date

Types of Inheritance

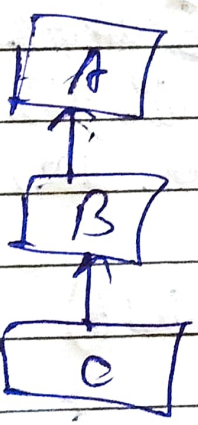
Single



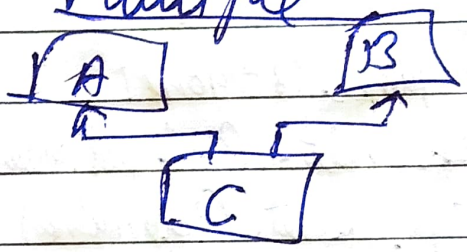
Hierarichal



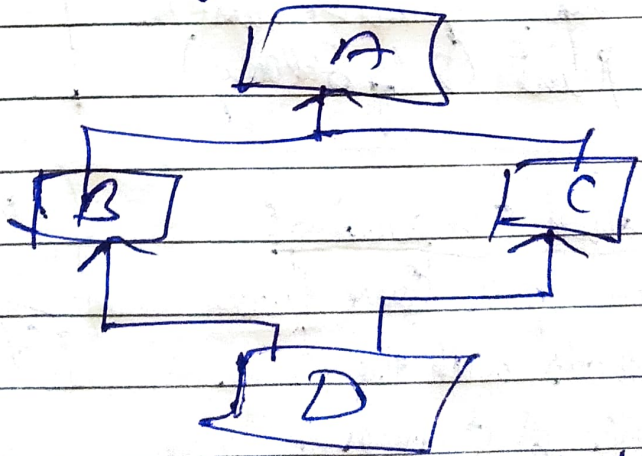
Multilevel



Multiple



Hybrid



Multiple Inheritances create a problem called diamond problem in Java but not in python.

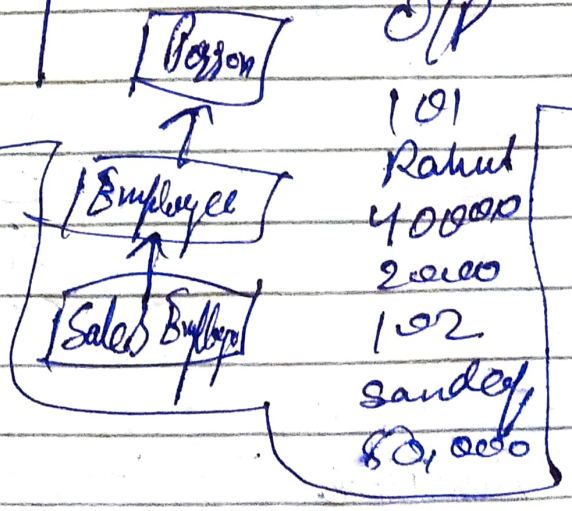
Multilevel Inheritance Example

```
class Person:
    def __init__(self, id, name):
        self.id = id
        self.name = name
    def printDetails(self):
        print(self.id)
        print(self.name)
```

```
class Employee(Person):
    def __init__(self, id, name, salary):
        super().__init__(id, name)
        self.salary = sal
    def printDetails(self):
        super().printDetails()
        print(self.salary)
```

```
class SalesEmployee(Employee):
    def __init__(self, id, name, sal, si):
        super().__init__(id, name, sal)
        self.SalesInc = si
    def printDetails(self):
        super.printDetails()
        print(self.SalesInc)
```

```
se = SalesEmployee(101, "Rahul", 40000, 2000)
se.printDetails()
e = Employee(102, "Sandeep", 50000)
print()
e.printDetails()
```



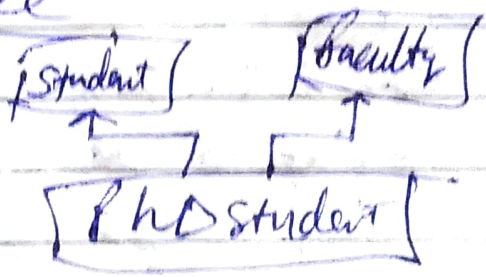
O/P

```
101
Rahul
40000
2000
102
Sandeep
50,000
```

Method Overriding: When you have two methods with the same name that each perform different tasks.

Page No.
Date

Multiple Inheritance

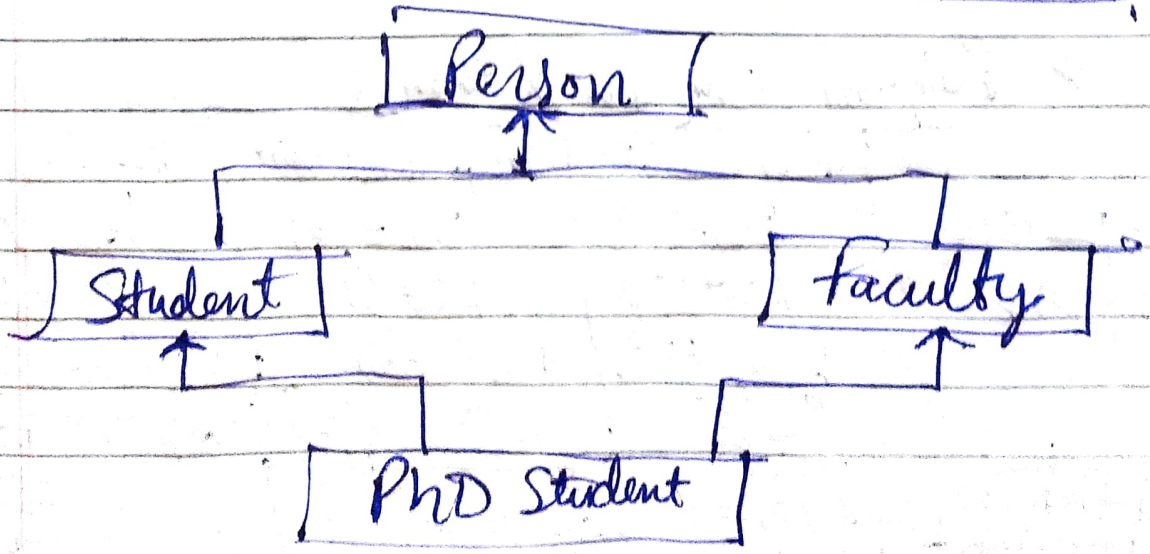



```
class Student :
    def __init__ (self, sid, dept):
        self.sid = id
        self.dept = dept
class Faculty :
    def __init__ (self, eid, dept):
        self.eid = id
        self.dept = dept
class PhDstudent (Student, Faculty):
    def __init__ (self, id, dept):
        super().__init__ (id, dept)

ps = PhDStudent (101, "CSE")
print (ps.sid)
print (ps.dept)
```

O/P
101
CSE

→ The Diamond Problem



~~#~~ four Main features of OOPs.

1) Encapsulation 3) ~~Abstraction~~ Abstraction
2) Inheritance 4) Polymorphism

```
class Person:
    def __init__(self, id, n):
        self.id = id
        self.name = n

class Student(Person):
    def __init__(self, id, n):
        super().__init__(id, n)
    def printDetails(self):
        print("Student:")
        print(self.id)
        print(self.name)

class Faculty(Person):
    def __init__(self, id, n):
        super().__init__(id, n)
    def printDetails(self):
        print("Faculty:")
        print(self.id)
        print(self.name)
```

```
class PhDStud(Student, Faculty):
    def __init__(self, id, n):
        super().__init__(id, n)

ps = PhDStud(101, "Sandeep")
ps.printDetails()
```

Output: Student:
101
Sandeep

~~#~~ In C++ ~~this~~ problem came & in Java it doesn't support it.

~~#~~ The Diamond problem does not arise in python because python calls only the first class's constructor when you have multiple classes. So only one instance of the person's parameter would be created.

→ Polymorphism in Python:

Polymorphism:- One name having diff forms.

In C++ → Multiple fun diff type parameter
In Python → It may give Error

```
def fun(a,b):
    print(a)
    print(b)
def fun(a,b,c):
    print(a)
    print(b)
    print(c)
fun(10,20)
```

O/P: ERROR!

```
def fun(a,b):
    print(a)
    print(b)
fun(10,20)
fun("Sandeep","ggg")
```

O/P:
10
Sandeep
ggg

```
def fun(l):
    for x in l:
        print(x)
fun([10,20,"ggg"])
```

O/P:
10
20
ggg

Page No.
Date

→ Abstraction in OOP

Abstraction is not providing all the details of implementation only providing the methods, which are to be implemented.

Abstract classes become the base class and then the classes which inherit from it and provide all the limitations of the abstract methods which are provided, they become the concrete class. If they don't provide the implementations of the all the methods, they also become Abstract Class

→ Above Eg of Polymorphism

Method overriding : Derived class has same name and parameter method as base class.

```
Class Employee :
    def __init__(self, id, name):
        self.id = id
        self.name = name
    def printDetails(self):
        print(self.id)
        print(self.name)
Class SalesEmployee(Employee):
    def __init__(self, id, name, s):
        Super().__init__(id, name)
        self.salesInc = s
    def printDetails(self):
        Super().printDetails()
        print(self.salesInc)
el = [Employee(101, "Sandeep"),
      SalesEmployee(102, "Rahul", 5000)]
for n in el:
    n.printDetails()
```

O/P

```
101
Sandeep
102
Rahul
5000
```

→ Polymorphism in Unrelated Classes

eg:-

```
Class Employee :
    def fun(self):
        print("fun() of Employee")
Class Customer :
    def fun(self):
        print("fun() of Customer")
```

l = [Employee(), customer()]
for x in l:
    x.fun()

O/p fun() in Employee
fun() in customer

→ Inbuilt Polymorphic Functions

print(len("ggfg"))
print(len([10, 20, 30, 40]))
O/p: 3
4

type("ggfg") <class, str>
id("ggfg")

→ Polymorphic Operators

print(3+2)
print("geeks" + "for" + "geeks")
print(4 < 100)
print("geeks" < "for")
print(3 * 2)
print("geeks" * 2)

5
geeks for geeks
True
False
6
geeksgeeks

→ Operator Overloading

class Product:
    def __init__(self, name, price):
        self.name = name
        self.price = price
    def __add__(self, other):
        return self.price + other.price

p1 = Product("keyboard", 600)
p2 = Product("mouse", 400)
print(p1 + p2)
O/p: 1000

Operator overloading is not supported in Java but is in C++

Dunder Methods:
__lt__(self, other)
__le__(self, other)
__eq__(self, other)
__ne__(self, other)
__gt__(self, other)
__ge__(self, other)
__sub__(self, other)
__mul__(self, other)

→ Abstract Class

```
from abc import ABC, abstractmethod
class Polygon(ABC):
    def __init__(self, colour):
        self.color = color
    @abstractmethod
    def printSides(self):
        pass
```

Abstract class

abstract method

```
class Triangle(Polygon):
    def __init__(self, colour):
        super().__init__(colour)
    def printSides(self):
        print("There are 3 sides")
p = Triangle("Red")
p.printSides()
```

o/p: There are 3 sides

Ex: Abstract class can contain concrete methods.

```
from abc import ABC, abstractmethod
class Polygon(ABC):
    def __init__(self, color):
        self.color = color
    @abstractmethod
    def printSides(self):
        pass
    def printColor(self):
        print(self.color)
class Triangle(Polygon):
    def __init__(self, color):
        super().__init__(color)
    def printSides(self):
        print("There are 3 sides")
p = Triangle("Red")
p.printColor()
```

Output:

Red