

## \* Introduction to python :-

1. python is popular language for programming.
2. It was developed by Guido van Rossum in 1991.
3. It is used for web development , software development Mathematics and system scripting.

## \* Advantages for python :-

1. Python works on different platforms (windows, Mac, Linux, etc)
2. Simple syntax
3. Concise code compared to other languages.
4. Immediate execution with interpreter system.
5. It supports procedural, object-oriented and functional programming paradigms.

## \* what Python can do :

1. web application development
2. workflow automation.
3. Database connectivity and file manipulation
4. Handling big data and complex mathematics.

## \* python Indentation :-

1. the indentation of python is very important.
2. python use indentation to indicate block of code.
3. indentation refers to the space at the beginning of code line.

## \* python comments

1. comments can be used for explain the code
2. comments start with a #, python will skip that line.

Examples:-

### 1. single line comment:-

```
# this is comment
```

### 2. Multiline comment:-

\* There is no multiline comment

Eg:- # this is comment

```
# this is comment
```

```
# this is comment
```

### 3. Multistring comment:-

\* It is start and ends with Triple quotes.

Eg:- ""

```
this is comment
```

More than just one line

"".

## \* Variables

1. Variables are containers for storing data values.
2. Variable can be short names (like x, like y) or more descriptive name (Name, age).

Rules for python variables:-

- A variable name must start with a letter or underscore.
- A variable name cannot start with a number.
- A variable can contains alpha-numeric and underscore (A-z, 0-9, \_)
- Variable names are case-sensitive (age, Age, AGE are three different variables)
- Python variable cannot be any of python keywords.

Examples:-

x = 5

x, y, z = 'orange', 'Banana', 'apple'

x=y=z = "Apple"

## \* unpack collection

Fruits = ['Apple', 'orange', 'Banana']

x, y, z = Fruits

Global variable:-

- \* A variable can be created outside of the function known as Global variable.
- \* Global variable can be used both inside and everywhere outside of function. [Note:- use global keyword while create inside the function]

## \* python datatypes

| Datatype   | Example                    |                  |
|------------|----------------------------|------------------|
| str        | $x = "Anil"$               | → Text type      |
| int        | $x = 20$                   |                  |
| float      | $x = 20.09$                |                  |
| complex    | $x = 1j$                   |                  |
| list       | $x = ['Apple', 'Banana']$  | → Numeric types  |
| tuple      | $x = ('Apple', 'Banana')$  |                  |
| range      | $x = range(5)$             |                  |
| dict       | $x = \{'Name': 'Anil'\}$   | → Sequence types |
| set        | $x = \{'Anil'\}$           |                  |
| frozenset  | $x = frozenset(\{'Ram'\})$ |                  |
| bool       | $x = True$                 | → Mapping type   |
| bytes      | $x = b>Hello$              |                  |
| bytearray  | $x = bytearray(5)$         |                  |
| memoryview | $x = memoryview(bytes(5))$ | → set types      |
| NoneType   | $x = None$                 |                  |
|            |                            | Binary types     |
|            |                            | → Boolean type   |
|            |                            | → None type      |

## \* Python Casting :

| Integers                         | Floors                               | strings                            |
|----------------------------------|--------------------------------------|------------------------------------|
| $x = int(1) \rightarrow x=1$     |                                      |                                    |
| $x = int(2.3) \rightarrow x=2$   | $x = float(2) \rightarrow x=2.0$     | $x = str(2) \rightarrow x='2'$     |
| $x = int("20") \rightarrow x=20$ | $x = float(2.5) \rightarrow x=2.5$   | $x = str(2.5) \rightarrow x='2.5'$ |
|                                  | $x = float('15') \rightarrow x=15.0$ | $x = str('K') \rightarrow x='K'$   |

## \* Python Strings

### \* Assign string to variable

A = "Hello"

### \* Assign multiline string to variable

x = """

sample Text of string

""""

### \* string length

\* To get the length of string, use the "len()" function

Eg:- Name = "Anil"

Result = len(Name) → Result = 4

### \* check string

\* To check if a certain character or phrase is present in a string, we can use keyword "in"

Eg:- Name = "Anil Kumar"

Result = "Anil" in Name → Result = True

\* To check if a certain character or phrase is not present in a string, we can use keyword 'not in'

Eg:- Name = "Anil Kumar"

Result = "Anil" not in Name → Result = False

## \* Slicing strings ( syntax:- variablename[start:stop:step] )

- \* you can return range of characters using slice index
- \* specify start index and end index, separated by colon

Eg:- Name = "Anil Kumar"

output = Name[1:4] → output = "nil"

### 1. Slice from start

→ By leaving out the start index, give end index alone

Eg:- Name = "Anil Kumar"

output = Name[:4] → output = "Anil"

### 2. Slice From the End

→ By leaving out the end index, the range will go till end  
give start index alone.

Eg:- Name = "Anil Kumar".

output = Name[5:] → output = "Kumar"

### 3. Slice with step

→ Allows you to skip elements in the step sequence

Eg:- List = [1, 2, 3, 4, 5, 6]

output = List[::2] → output = [1, 3, 5]

### 4. Reverse the string using slice

Eg:- Name = "Anil"

output = Name[::-1] → output = "nilA"

## \* Modify Strings

### 1. Upper Case :-

→ The upper() method returns the string in uppercase

Eg:- Name = "Anil"

output = Name.upper() → output = ANIL

### 2. Lower Case :-

→ The lower() method returns the string in lowercase

Eg:- Name = "Anil"

output = Name.lower() → output = anil

### 3. Capitalize :-

→ The capitalize() method returns the first character of string in uppercase.

Eg:- Name = "anil Kumar"

output = Name.capitalize() → output = "Anil Kumar"

### 4. Remove whitespace :-

→ whitespace is the before and/or after the actual text.

→ The strip() method removes any whitespace from the begining and end.

Eg:- Text = " <sup>←</sup>Hellow world"

output = Text.strip() → output = "Hellow world"

## 5. Replace string

→ The replace() method replaces a string with another string

Name = "Anil Kumar"

output = Name.replace("Kumar", "Reddy")

output = "Anil Reddy"

## 6. Split string:-

→ The split() method returns a list. where the text between the specified separator becomes list items.

→ The split method splits the string into substrings. if it find instances of the separator.

Eg:- Text = "Anil, kumar, Reddy"

output = Text.split(",")

output = ['Anil', 'kumar', 'Reddy']

## 7. Concatenate strings:

→ To concatenate or combine two strings you can use + operator

Eg:- Merge variable x with variable y into variable z

x = Anil

y = Kumar

z = x+y → z = "Anil Kumar"

## 8. Format string

→ use the `format()` method to insert numbers into string.

Eg:- Name = "Anil"

Age = 23

output = "My name is {} and I am {} years old."

• `format(Name, Age)`

output = "My name is Anil and I am 23 years old."

### \* F-strings:-

→ F-string provide more readable to format strings

Eg:- Name= "Anil"

Age = 23

output = f"I am {age} years old and My {Name} is {Name}"

output = I am 23 years old and My Name is Anil.

# \* String Methods

| Method         | Description   |
|----------------|---|
| Capitalize()   | → converts the first character into uppercase.                      |
| casefold()     | → converts string to lower case.                                    |
| center()       | → Returns a centered string.  |
| count()        | → Returns the number of times occurs a specified value in a string. |
| encode()       | → Return encode version of string                                   |
| endswith()     | → Return True if string ends with specified value                   |
| expandtabs()   | → set Tab size of the string.                                       |
| find()         | → search specified string and returns position                      |
| format()       | → formats specified values in a string.                             |
| format-map()   | → "   |
| index()        | → search specified value and returns position                       |
| isalnum()      | → Returns True if All characters of string is Alphanumeric.         |
| isalpha()      | → Returns True if all characters are alphabet.                      |
| isascii()      | → Returns True, if all characters are ascii characters.             |
| isdecimal()    | → Returns True, if all characters are decimal                       |
| is digit()     | → Returns True, if all characters are digits.                       |
| isidentifier() | → Returns True, if the string is an identifier.                     |
| islower()      | → Returns True, if all the characters are lowercase.                |
| isnumeric()    | → Returns True, if all the characters are numbers.                  |
| is printable() | → Returns True, if all characters are printable.                    |

- isspace() → Returns True, if all characters are space.
- istitle() → Returns True, if string follow the rules of title.
- isupper() → Returns True, if all characters are uppercase.
- join() → Join the elements of an iterable to end of the string.
- ljust() → Returns a left justified version of string
- lower() → converts string into lower case
- lstrip() → Returns left trim version of string
- maketrans() → Returns a translation table to be used in translator.
- partition() → Returns a tuple where string parted into 3 parts.
- replace() → Returns string, where specified value replaced to string.
- rfind() → search string for specified value, Returns last position.
- rindex() → "
- rjust() → Returns Right justified version of string
- split() → splits the string at the specified separator.
- splitlines() → splits the string at line breaks and returns a list
- startswith() → Returns true, if the string starts with specified value.
- strip() → Returns a trimmed version of string
- swapcase() → swaps cases, lower case becomes uppercase and vice versa
- title() → converts the first character of each word into uppercase
- translate() → Returns a translated string
- upper() → converts a string into uppercase
- zfill() → Fill the string with specified value of 0 values at begin

## \* Python Booleans

- Booleans represent one of two values : True or False
- Most values are True, if it has some sort of content
- Any string is True, except empty string
- Any number is True, except 0.
- Any list, tuple, set and dict are True, except empty ones

Eg:- \* the following will return False

bool(False)      bool(0)      bool("")      bool({})  
bool(None)      bool("")      bool([])

## \* Python keywords:

- |            |              |              |
|------------|--------------|--------------|
| 1. as      | 12. except   | 24. is       |
| 2. assert  | 13. finally  | 25. in       |
| 3. await   | 14. if       | 26. global   |
| 4. and     | 15. elif     | 27. del      |
| 5. async   | 16. else     | 28. raise    |
| 6. True    | 17. for      | 29. with     |
| 7. False   | 18. while    | 30. lambda   |
| 8. def     | 19. break    | 31. None     |
| 9. class   | 20. continue | 32. import   |
| 10. return | 21. pass     | 33. from     |
| 11. try    | 22. not      | 34. nonlocal |
|            | 23. or       | 35. yield    |

## \* Python Operators

→ operators are used to perform operations on variables & values

→ python devides the operators in the following groups:-

1. Arithmetic operators
2. Assignment operators
3. Comparison operators
4. logical operators
5. Identity operators
6. Membership operators
7. Bitwise operators.

### 1. Arithmetic operators:-

→ Arithmetic operators are used with numeric values to perform common mathematical operation

| operator | Example  | Name             |
|----------|----------|------------------|
| +        | $x+y$    | Addition         |
| -        | $x-y$    | Subtraction      |
| *        | $x*y$    | Multiplication   |
| /        | $x/y$    | Division         |
| %        | $x \% y$ | Modulus          |
| **       | $x**y$   | Exponentiation   |
| //       | $x//y$   | Floor devision , |

## 2. Assignment operators

→ Assignment operators are used to assign values to variables.

operator

Example

=

$x = 20$

+=

$x += 20$

→  $x = x + 20$

-=

$x -= 20$

\*=

$x *= 20$

/=

$x /= 20$

%=

$x \% = 20$

//=

$x // = 20$

\*\*=

$x ** = 20$

&=

$x \& = 20$

|=

$x | = 20$

^=

$x ^ = 20$

>>=

$x >> = 20$

<<=

$x << = 20$

## 3. Comparison operators

→ Comparison operators are used to compare two values

operator

Example

Name

==

$x == y$

equal

!=

$x != y$

not equal

>

$x > y$

less than

<

$x < y$

greater than

>=

$x >= y$

less than & equal to

<=

$x <= y$

Greater than & equal to

## 4 Logical operators

→ logical operators are used to combine conditional statements.

| operator | Example              | Description                              |
|----------|----------------------|--|
| and      | $x < 5$ and $x < 10$ | both conditions<br>Need to be TRUE       |
| or       | $x < 5$ or $x > 1$   | Atleast one condition<br>Need to be TRUE |
| not      |                      | Reverse the result                       |

## 5 Identity operators

→ Identity operators are used to compare the objects not if they are equal, but if they are actually the same object and with same memory location

| operator | Example               |   |
|----------|-----------------------|---|
| is       | $x \text{ is } y$     | Returns TRUE, if both variables are same object     |
| is not   | $x \text{ is not } y$ | Returns True, if both variables are not same object |

## 6 Membership operators

→ Membership operators are used to test if sequence present in object.

| operator | Example               | Description         |
|----------|-----------------------|---------------------|
| in       | $x \text{ in } y$     | if present True     |
| not in   | $x \text{ not in } y$ | if not present True |

## 7. Bitwise operators

→ Bitwise operators are used to compare (binary) numbers.

| operator | Name | Example      | Description                |
|----------|------|--------------|----------------------------|
| $\sim$   | Not  | $\sim x$     | invert all the bits        |
| $\&$     | AND  | $x \& y$     | set 1 if both bits 1       |
| $ $      | OR   | $x   y$      | set 1 if any one bit 1     |
| $\wedge$ | XOR  | $x \wedge y$ | set 1 if only one bit is 1 |

## \* Python Collections

→ There are four collection data types in the python.

1. List → It is ordered and changeable, allows duplicates
2. Tuple → It is ordered and unchangeable, allows duplicates
3. Set → It is unordered, unchangeable and indexed, don't allow duplicates.
4. Dictionary → It is ordered and changeable, NO duplicates

1. List :-

Note:- The first item has index 0.

### \* List Methods :-

| Method    | Description                                     |
|-----------|---|
| append()  | Add element at the end of list                  |
| clear()   | Remove all elements from list                   |
| copy()    | Returns a copy of list                          |
| count()   | Returns count of specified value                |
| extend()  | Add elements of list to the end of current list |
| index()   | Return index of first element                   |
| insert()  | Add element at the specified position           |
| pop()     | Removes element at specified position           |
| remove()  | Remove item with specified value.               |
| reverse() | Reverse the items with of list                  |
| sort()    | Sort the list                                   |

## \* Access items

list = ["Apple", "orange", "banana"]

1. Find first element

First element = list[0]

2. Find last element

Last element = list[-1]

3. Get range of elements

Range of elements = list[1:3]

## \* change item value

→ To change the value of specific item, refer to index number

list = ["Apple", "banana", "orange"]

list[1] = "grapes"

list = ["Apple", "grapes", "orange"] → Result.

## \* Add List items

1. Append items

→ To add an item to the end of the list, used append() method.

list = ["Anil", "Hari"]

list.append("Vamsi")

list = ["Anil", "Hari", "Vamsi"] → Result

## 2. Insert items

→ To insert a list item at a specified index. Use `insert()` method

`List = ["Ram", "Kiran", "Hari"]`

`list.insert(1, "Anil")`

`list = ["Ram", "Anil", "Kiran", "Hari"]` → Result

## 3. Extend list

→ To append elements from another list to the current list. Use `extend()` method.

`list1 = ["Sumanth", "Bharath"]`

`list2 = ["Anil", "Hari"]`

`list1.extend(list2)`

`list1 = ["Sumanth", "Bharath", "Anil", "Hari"]` → Result

## \* Remove list items

### 1. Remove specified item

`List = ["Anil", "Hari"]`

`list.remove("Anil")`

`list = ["Hari"]` → Result

### 2. Remove specified index

`List = ["Anil", "Hari", "Vamsi"]`

`list.pop(1)`

`list = ["Anil", "Vamsi"]` → Result

### 3. Using del keyword

- \* Remove the specified index

```
list = ["Anil", "Hari", "praveen"]
```

```
del list[2]
```

```
list = ["Anil", "Hari"] → Result
```

- \* delete entire list

```
del list
```

### 4. Clear the list

→ The list still remains, but it has no content.

```
list = ["Apple", "orange"]
```

```
list.clear()
```

```
list = [] → Result
```

### \* Sort list Items

#### 1. Sort the list (Ascending order)

```
list = [5, 10, 3, 6]
```

```
list.sort()
```

```
list = [3, 5, 6, 10] → Result
```

#### 2. Sort the list (Descending order)

→ To sort descending order using key argument reverse=True

```
list = ["Apple", "orange"]
```

```
list.sort(reverse=True)
```

```
list = ["orange", "Apple"]
```

### 3. Case sensitive sort

→ By default the `sort()` method is case sensitive, resulting in all capital letters being sorted before lower case letters.

`List = ["hari", "Vamsi", "anil"]`

`list.sort()`

`list = ["Vamsi", "anil", "hari"]` → Result

### 4. Case insensitive sort

→ if you want a case-insensitive sort function, use `str.lower` as a key function

`List = ["hari", "Vamsi", "anil"]`

`list.sort(str.lower)`

`list = ["anil", "hari", "Vamsi"]` → Result

### \* Reverse order

→ Reverse the current sorting order of the elements

`List = ["Anil", "Ram", "Somu"]`

`list.reverse()`

`list = ["Somu", "Ram", "Anil"]` → Result

### \* Copy a list

→ you cannot copy simply by tying `list1 = list2`, whatever you change both lists it will change.

1. `list2 = list1.copy()`

2. `list2 = list(list1)`

## 2. Tuple

### \* Customize sort Function

→ we can also customize your own function by using the keyword argument `key = function`

Example:-

\* sort the list based on how close the number to 50

```
def myfun(n)
```

```
    return abs(n-50)
```

```
list = [100, 50, 65, 82, 23]
```

```
list.sort(key=myfun)
```

```
list = [50, 65, 23, 82, 100] → Result.
```

### \* list Comprehension

→ with list comprehension, you can do all that with one line

```
Fruits = ["apple", "banana", "cherry"]
```

```
Newlist = [x for x in fruits if "a" in x]
```

```
Newlist = ["apple", "banana"] → Result.
```

## 2. Tuple

- Tuples are used to store multiple items in single variable
- Tuples are written in round brackets.

### \* Access the Tuple items

`Tuple = ("Anil", "Hari", "Ram")`

`first item = Tuple [0]`

`First item = "Anil"` → Result

### \* change tuple values

→ once a tuple is created, you cannot change its value

Tuples are unchangeable or immutable.

→ we can convert the Tuple into list, change the list, and convert list back to tuple.

### Example:-

`Tuple = ("Anil", "Hari")`

`list = list (tuple)`

`list [1] = "Vamsi"`

`New-Tuple = tuple(list)`

`New-Tuple = ("Anil", "Vamsi")` → Result

Note:- Same procedure for both add and remove also

### 3. Sets

- sets are written with curly brackets
- set is a collection which is unordered, unchangeable and unindexed. duplicates not allowed.

#### \* Access Items

- you cannot access items in a set, by referring to an index or a key
- But you can loop through the set items using for loop or ask if specified value is present in a set By using 'in' keyword.

Example:-

set = {"Anil", "Hari"}

1. For x in set  
print(x)
2. if "Anil" in set  
print("Anil")

#### \* change set:-

Note:- once set created we cannot change items, but we can add items.

#### \* Add Items

- To add one item to set used Add() method.

set = {"Apple", "orange"}

set.add("banana")

set = {"Apple", "orange", "banana"} → Result

## \* Add sets or Any Iterable

→ To add items from another set or another iterable to current set, use update() method.

Example:-

① set1 = {"Ram", "Logesh"}.

set2 = {"Arun", "Suman"}

set1.update(set2)

set1 = {"Ram", "Logesh", "Arun", "Suman"} → Result

② set1 = {"Anil", "Hari"}

list1 = ["Vamsi", "Bharat"]

set1.update(list1)

set1 = {"Anil", "Hari", "Vamsi", "Bharat"}

## \* Remove Items from set

→ To remove an item in a set, use the remove() or discard() method.

Note:- 1. if the item to remove does not exist, remove() will raise error.

2. if the item to remove does not exist, discard() will not raise error

# OOPS Concepts in Python

\* → python is an object oriented programming language

1. class
2. object
3. Abstraction
4. Inheritance
5. Encapsulation
6. polymorphism

1. class :-

- It is blueprint or template for creating object
- It defines the attributes and methods of that objects of the class.
- Create class by using 'class' keyword

Eg:-

```
class Bank:  
    print("Anil")
```

2. object :-

- It is instance of a class.

- we can access the attribute and methods of class by using object.

Eg:- obj = Bank()

## \* `--init--()` Method

→ when, class initiated, the '`--init--()`' method is automatically called.

Note:- The 'self' parameter is reference to the current instance of the class. and it is used to access variables that belongs to the class.

Example:-

class Bank:

`def --init--(self, ac-NO, Bank-name, bal):`

`self.ac-NO = ac-NO`

`self.Bank-Name = Bank-Name`

`self.Bal = bal`

`def Balance(self):`

`print(self.Bal)`

`def withdraw(self, amount)`

`self.Bal = self.Bal - amount`

`def deposite(self, amount)`

`self.Bal = self.Bal + amount`

`obj1 = Bank(123, "SBI", 5000)`

→ object creation

`obj1.withdraw(1000)`

`print(obj1.Balance) → 4000` → Result

### 3. Abstraction

→ Hiding complex implementation details and showing only essential features of the object.

Example:-

```
from abc import ABC, abstractmethod
```

```
class shape(ABC):
```

```
    @abstractmethod
```

```
    def area(self):
```

```
        pass
```

```
class Rectangle(shape)
```

```
    def __init__(self, width, height):
```

```
        self.width = width
```

```
        self.height = height
```

```
    def Area(self):
```

```
        return self.width * self.height
```

```
rectangle = Rectangle(5, 4)
```

```
print(rectangle.Area())
```

20 → Result

Note:-

→ If class contains one or more than one abstract method then it is called abstract class.

→ The method without implementation it is called abstract method.

## 4. Inheritance

- Create a new class from already created class.
- In new class we can access attributes and methods from all created class.
- Types of inheritances

1. Single inheritance
2. Multiple inheritance
3. Multilevel inheritance
4. Hierarchical inheritance
5. Hybrid inheritance

### 1. Single inheritance:

→ Create a single child class from single parent class.

Example:-

```
class parent:  
    def method1(self):  
        print("parent")
```



```
class child(parent):
```

```
    def method2(self):  
        print("child")
```

```
child_obj = child()
```

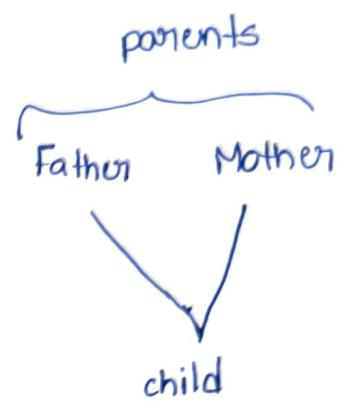
child\_obj.method1() → parent  
child\_obj.method2() → child } → Results

## 2. Multiple inheritance

→ Create single child class from more than one parent class

Example:-

```
class Father:  
    def method1(self):  
        print("Father")
```



```
class Mother:  
    def method2(self):  
        print("Mother")
```

```
class child(Father, Mother):  
    def method3(self):  
        print("child")
```

obj-child = child()

obj-child.method3() → child

obj-child.method1() → Father

obj-child.method2() → Mother

print(child.method1())

} output

B.

Output:-

(<class 'main.child'>, <class 'main.Father'>, <class 'main.Mother'>, <class 'object'>)

Note:- myo → method resolution order

### 3. Multilevel Inheritance

→ Create a parent class from grand parent class.  
and also create a child class from parent class

Example:-

```
class grandparent:  
    def method1(self):  
        print("grand parent")
```

```
class parent(grandparent):  
    def method2(self):  
        print("parent")
```

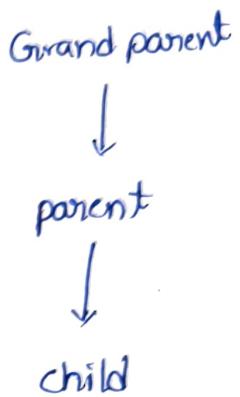
```
class child(parent):  
    def method3(self):  
        print("child")
```

obj-child = child()

obj-child.method1()

obj-child.method2()

obj-child.method3()



Output:-

grand parent

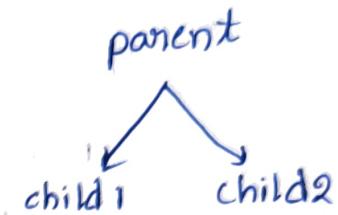
parent

child.

#### 4. Hierarchical Inheritance

→ Create more than one child class from single parent class.

Example:-



```
class parent:
    def method1(self):
        print("parent")
```

```
class child1(parent):
    def method2(self):
        print("child1")
```

```
class child2(parent):
    def method3(self):
        print("child2")
```

```
obj-child1 = child1()
```

```
obj-child1.method1() → parent
```

```
obj-child1.method2() → child1
```

```
obj-child2 = child2()
```

```
obj-child2.method1() → parent
```

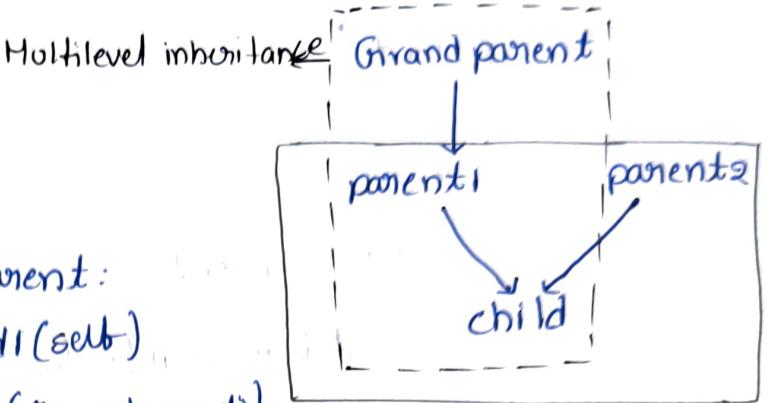
```
obj-child2.method3() → child2
```

} output

## 5. Hybrid Inheritance:-

→ Combination of more than one inheritance.

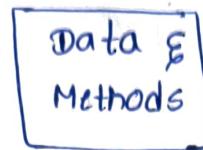
Example:-



```
class grandparent:  
    def method1(self):  
        print("grand parent")  
  
class parent1(grandparent):  
    def method2(self):  
        print("parent 1")  
  
class parent2:  
    def method3(self):  
        print("parent2")  
  
class child(parent1, parent2):  
    def method4(self):  
        print("child")  
  
obj_child = child()  
  
obj_child.method1() → grand parent  
obj_child.method2() → parent 1  
obj_child.method3() → parent2 } output  
obj_child.method4() → child
```

## 5. Encapsulation

→ Combine the data & methods in single unit of class



class

Reasons for using encapsulation:-

1. Code will be organized and clean.
2. prevent the data from accidental removal & deletion
3. Abstraction → Hides implementation logic & allows only use
4. Data hiding

\* Access specifiers & Access modifiers

1. public access specifiers
  2. protected access specifiers
  3. private access specifiers
- } using for security purpose.

1. public access specifier :-

→ if data declare in public, we can access the data

\* In same class

\* By object

\* we can access subclass & we can access by subclass obj.

Example:-

```
class parent:  
    publicdata = 20  
    def method1(self):  
        print(self.publicdata)
```

```
class child(parent):  
    def method2(self):  
        print(self.publicdata)
```

obj1=parent()

obj1.method1() → 20

print(obj1.publicdata) → 20

obj2=child()

obj2.method2() → 20

print(obj2.publicdata) → 20

} outputs

Note:- we can access by class, class object, subclass and subclass object.

## 2. protected access specifier:-

→ if data is declare in protected, it can access only in same class and subclass.

→ Just it is signal, we can access like public only

→ For protected we need to \_ [single underscore] before data & methods.

Example :-

```
class parent:  
    -protecteddata = 20  
  
    def method1(self):  
        print(self.-protecteddata)
```

```
class child(parent):  
  
    def method2(self):  
        print (self.-protecteddata)
```

```
obj1 = parent()  
obj1.method1() → 20  
  
obj2 = child()  
obj2.method2() → 20  
  
print(obj1.-protecteddata) → 20
```

output

### 3. private access specifier:

- If data and method declare as private, we can access only in the same class.
- It is just way of signal for developers.
- For private we need to \_\_ [double underscore] before data & methods.

Example:-

Class parent:

--privatedata = 20

def method1(self):

print(self.--privatedata)

class child (parent):

def method2(self):

print(self.--privatedata)

obj1 = parent()

obj1.method1()

obj2 = child()

obj2.method2() → it's through error

Note:- we can access in the same class. By using name manling we can access by subclass and objects.

Example:- (Name manling)

print(obj1.parent.--privatedata) → 20

-<class name>

we Need to add

before private data variable

↓  
output

Note:- To avoid name collisions we use {public, private, protected} in python

## 6. polymorphism

- The word polymorphism means "Many forms".
- In programming, it refers to methods/operators with same name that can execute many objects of many tasks.
- An ability to do more than one task.

Example:- 1. Example in operator (+)

print(3+4) → '+' act as sum  
'7'

print("Hi" + "Anil") → '+' act as concatenate  
"Hi Anil"

2. Example in function

```
def sum(*args):  
    if args:  
        start = 0  
        for i in args:  
            start = start + i  
    return start
```

print(sum(1,2,3)) → 6  
output:

print(sum([1,2,3], [4,5,6])) → [1,2,3,4,5,6]  
print(sum("Hello", "world")) → Hello world

Note:- In above sum function work as sum and concatenate

## 1. Method overloading

- If the class contain more than one method has same name &
- The method contains different data types of parameters (i) different number of parameters Both is called method overloading.

Examples:-

1. Same method name with different type of parameter

Class A:

```
def sum(int,int):
```

```
    pass
```

```
def sum(str,str):
```

```
    pass
```

2. Same method with different number of parameters

Class B:

```
def sum(int,int):
```

```
    pass
```

```
def sum(int,int,int):
```

```
    pass
```

## 2. Method overriding:-

→ super class (parent class) & subclass (child class) has same method name. if we access the method in subclass Then only sub class method accessed without accessing super class method.

### Rules for Method overriding:-

1. Super class and sub class must be present (Inheritance)
2. Declare two classes with with same method and some parameter
3. Logic must be different in methods
4. Method override will be done when we access the same method in sub class . if we access in super class the method will not override.

### Example:-

```
class parent:  
    def method(self):  
        print("parent")
```

```
class child(parent):  
    def method(self):  
        print("child")
```

obj = child()  
obj.method() → 'child' → output.