**Python - Notes**

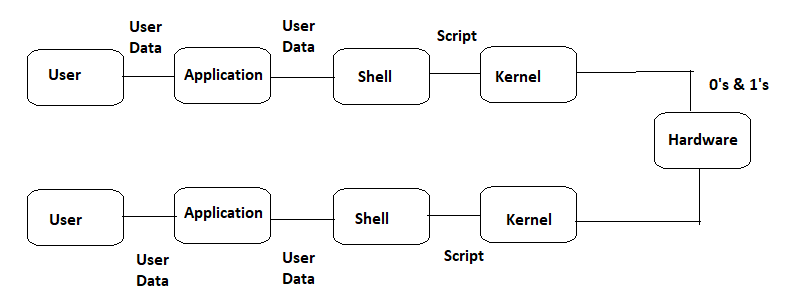
***Python installation***

http://www.python.org

Downloads > Windows > Python 3.6.7 > Download windows x86 executable installer(depends on the bit that your laptop is) > Double click on the installer and click on run > Check add python path (check box)

***Architecture***

|  |  |
| --- | --- |
| 1. Hardware 2. Kernel 3. Shell 4. Application 5. User data | **Python resides in kernel** |



***Kernel*** - It has some predefined compilers OR it is a collection of compilers, Which is responsible for the interaction between hardware and the user

To make python to work we need to add interpreter at kernel, IDLE is the interpreter of python

***Types of compilers***

1. Compiler - which converts whole program to machine understandable language at a time or at once

2. Interpreter - Which converts the code line by line

\*\*\* Python makes use of interpreter

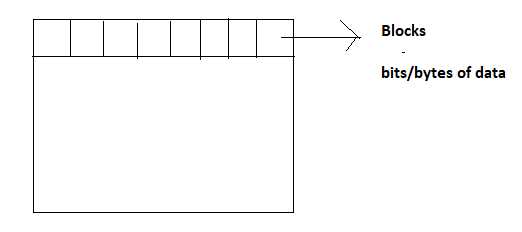
***Python editors***

**IDE** - Integrated Development Environment

**IDLE -** Integrated Development and Learning Environment

***Memory***

Memory is a place where the data is stored, It is divided into blocks where the blocks can contain bits of data or bytes of data



***Variable***

Variables are the named memory location

Each and every variable can store value or the address

Syntax --- **VariableName = Value ----** example:- a=10 , b=1

*NOTE*: If more than one value is stored then it is stored as a "**TUPLE**" example a=1,2,3

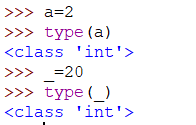
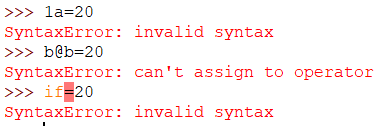


***Identifiers*** -- The variable name is known as identifier which identifies the memory

***Rules to define Identifiers***

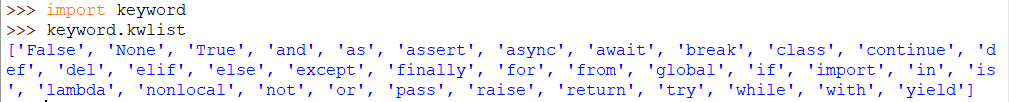
* + Identifiers must always start with the character or underscore (\_)
  + It can be a combination of alphabet and numbers (but should never start with number)
  + Identifier cannot be a numeric value because one integer value cannot define another integer value
  + Identifiers should not start with/contain special characters except \_
  + Keywords cannot be used as identifiers

Example Bad Syntax

***Keywords*** –

They are the reserved words which has some predefined meaning



***Data are of two variants***,

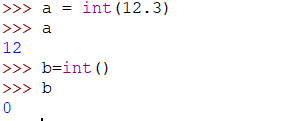
1. Individual data
2. Grouped data

*Individual data types*

1. **Int** – It specifies that the value stored in the memory as an integer value, It can be positive or a negative value , Positive values are not prefixed with any operators, Negative values will be prefixed with – operator, which is also called as signed integer value

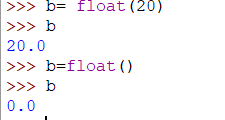
To convert the decimal value to an integer value we have to use following syntax

Variable name = int(decimal value)



1. Float:- It specifies that the value stored in the memory is of type decimal, In programing language we call the decimal value as floating point value. Floating point value can be both positive or negative. To convert integer to floating point value we have to use syntax as below

Variable name = float(integer value)

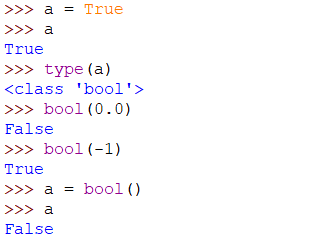


1. Bool:- Bool data type which is used to convert any value into either true or false, Boolean data type majorly are of two values True and False

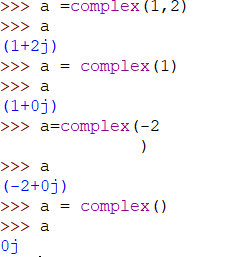
In all the cases we will convert the values from the respective type to Boolean type, While converting numeric value 0 is considered as false and non zero is considered as true

To convert anything to Boolean type we have to use syntax as

Variable name = bool(integer / float)



1. Complex Number:- A number which has both real and imaginary value



*Grouped Data items*

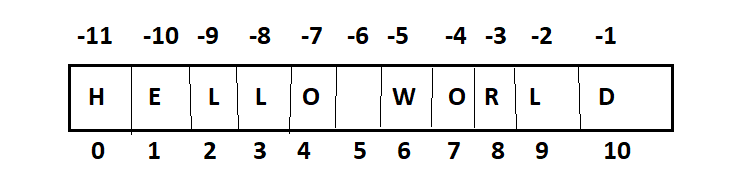
To store the collection of values or group of values we use grouped data items

1. String :- It is a collection of characters which are enclosed with pairs of “ ”or ‘ ’, Both “ ” and ‘ ’ will give the same meaning

In string the index of the characters are considered as sub address,

In python index value from left to right starts with 0,1… , from right to left it starts with -1,-2….

Phenomenon of assigning index is called indexing, String will also store the special location for white space



1. List—[]:- It is a collection of homogenous or heterogeneous data items, which are separated with , operator and enclosed with pairs of square brackets

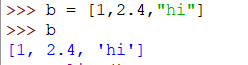
List also supports indexing, while moving from left to right index starts from 0 and ends with (length -1)

While moving from right to left, index starts from -1 and ends with –(length)

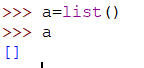
Indexing in list works similar to that in string

VariableName = [value1,value2,….]

Homogeneous list Hetrogeneous list

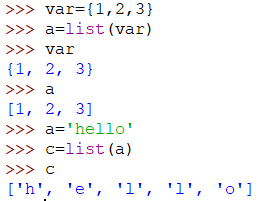
 

To create an empty list



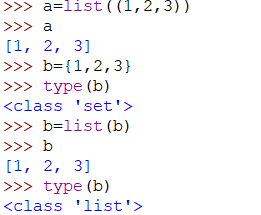
To convert from set , tuple or string into list, the syntax is

VariableName=list(variable)

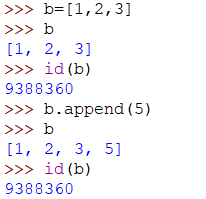


\*\*\*\*\*\*Inside a list - as an element either we can have string, List , tuple, set or dictionary





List is modifiable in python



1. **Tuple—():** it is a collection of homogeneous or heterogeneous items separated by , operator and enclosed within pairs of parenthesis

Indexing in tuple which is similar to as that of string and list , ***In python the tuple is immutable that is modification data items present in tuple is not possible***

To create an empty tuple the syntax is a=tuple()

**VariableName=(value1,value2,…)**



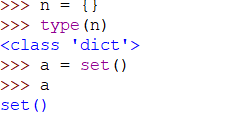
1. **Set ---{}:** It is a collection of non repeated homogenous and heterogeneous data items, when it is a collection of heterogeneous***, we can store only tupple and string inside set***

Each and every element is separated by comma and all the elements are enclosed with pairs of {}

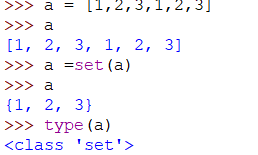
***Set doesnot support indexing that is the identification of individual data items is not possible in set, set is modifiable***

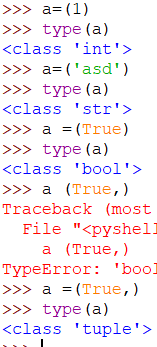
Set is modifiable that is we can modify the set by adding an element or removing an element

In set the elements are stored in random order



Variablename = {data1,data2…}





1. Dict – It is a collection of key value pairs, where each key value pair is separated by comma. Where each key and value are operated by colon operator (:)

All the key value pairs will be enclosed with pairs of {}

In dict each key should have a single value, If the dict key has more than one value then it has to be stored in the form of list set or tuple

Dict keys must always be uniques it cannot be repeated

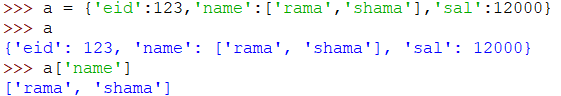
With the dictionary values are in the form of key value pairs, then the value must be stored as

dictVar ={‘key1’:’value1’,’key2’:’value2’….}

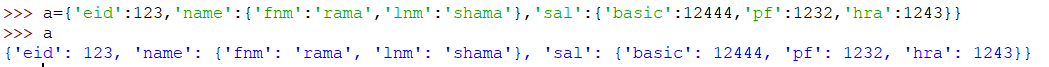
example: {‘eid’:123,’name’:’Rama Chandra tejas’,’sal’:12000}

Copy the diagram

Example:



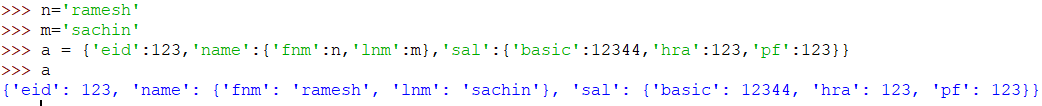
Copy the diagram



Write the diagram

|  |
| --- |
| Note:- [] --- List  () --- Tuple  {} --- Set |

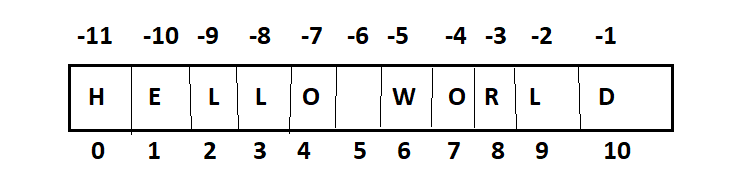
Use of variables inside a dictionary



|  |  |
| --- | --- |
| **Grouped Data** | |
| String | Supports  indexing |
| List |
| Tupple |
| Set | Does not support indexing |
| Dictionary |

1. ***Slicing:*** Identification of individual data items in grouped data items like string, list, tuple and dictionary

a=Hello World



StringVar[Index]

a[0]=a[-11]=H

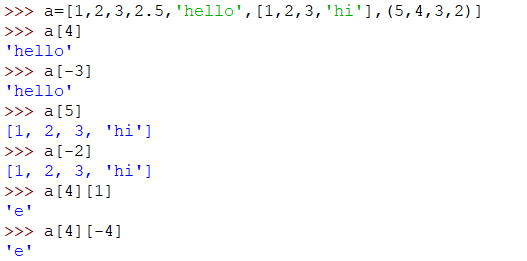
a[10]=a[-1]=d

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 2.5 | |  |  |  |  |  | | --- | --- | --- | --- | --- | | H | E | L | L | o | | |  |  |  |  | | --- | --- | --- | --- | | 5 | 4 | 3 | 2 | |

**List**

a=

**ListVar[Index]**



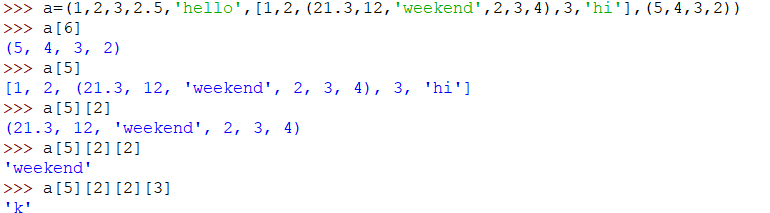
**Tupple**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 2.5 | Hello | |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 1 | 2 | |  |  |  |  |  | | --- | --- | --- | --- | --- | | 21.3 | 12 | Weekend | 2 | 3 | | 3 | Hi | | |  |  |  |  | | --- | --- | --- | --- | | 5 | 4 | 3 | 2 | |

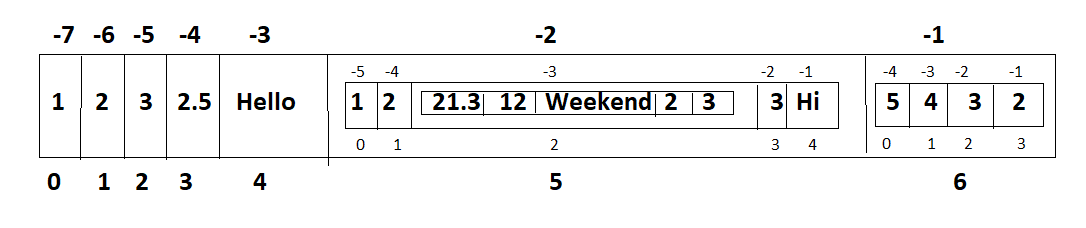
a=

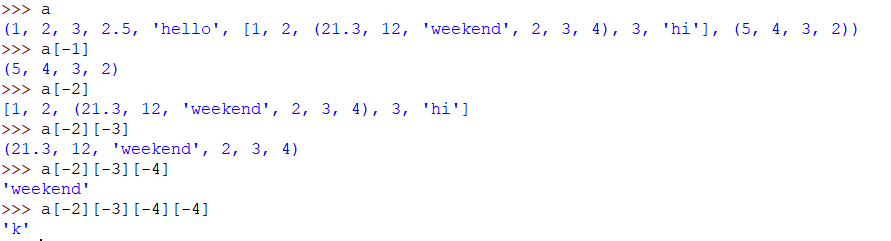
* Display k(weekend) with both positive and negative index

*Positive index*



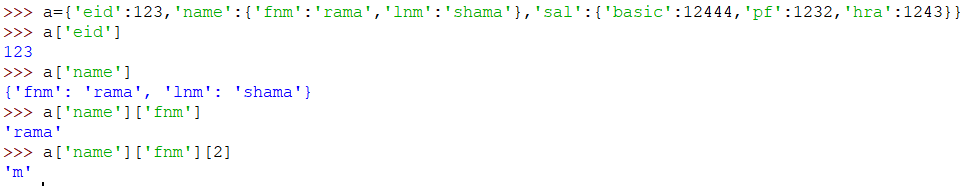
*Negative Index*





**Dictionary**

**dict[‘key’]**

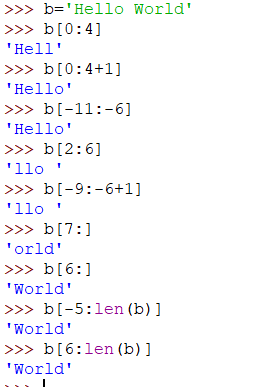


**Identification of Sequence of data items in string list and tupple**

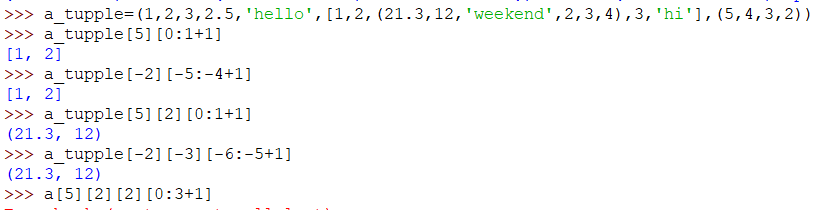
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| H | E | L | L | O |  | W | O | R | L | d |

**b=**

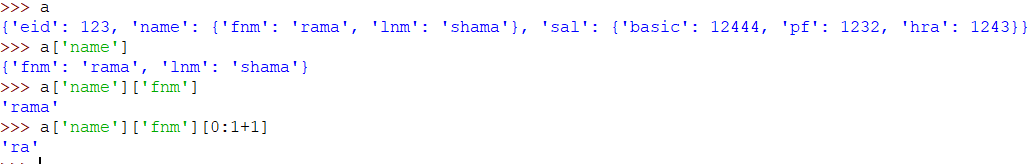
**strVar[startIndex : endIndex+1] \*\*\*\*\* since end index is not included to display we use +1**



**Note: Slicing a slice is not possible in python in order to achieve that the slice should be first stored in a variable and then it should be sliced further**



**Example to slice a dict value**

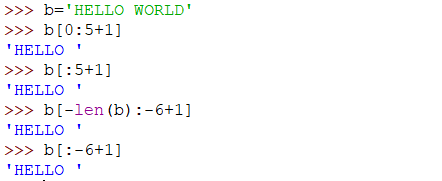


**Identification of sequence of data items in string, list and tupple. If the start index equals to 0 or –(length)**

**Syntax--- var[:endIndex+1]**

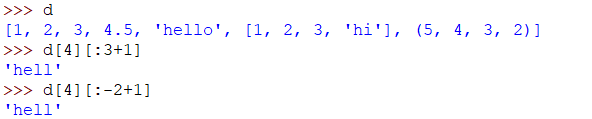
**String:** Start index can be skipped when it is 0 or –length

* **Print ‘HELLO ‘**



**List:**

* Print ‘hell’



**Tupple** :

Write the examples

**Identification of sequence of data items in string list and tupple. If the end index +1 = length or 0**

**Syntax--- var[startIndex:]**

**String**

**Write the examples**

**List**

**Write the examples**

**Tupple**

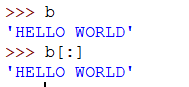
**Write the examples**

**Identification of sequence of data items in string list and tuple based on user defined updation or update value**

**Syntax--- var[startIndex : endIndex+/-1 : updateValue]**

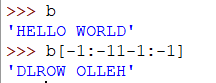
* **If the update value is positive ---- endIndex+1**
* **If the update value is negative ---- endIndex-1**

*Note:* if both the start index and end index is skipped it returns the entire data in the grouped data items(String, List and Tupple)



|  |  |
| --- | --- |
| To display the even position element or the characters | To display the odd position element or the characters |

\*\*\*\*\*\*\*\*\*To display the reverse of a string / List /Tupple

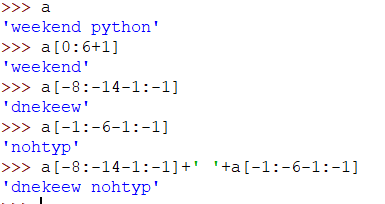


*Note*: To print the data in the reverse order in grouped data items then the update value must be negative

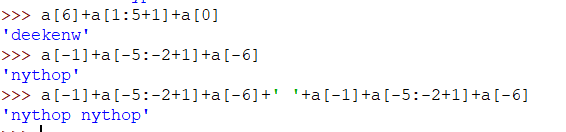


***Assignement***: I/P ‘Weekend Python’----

1. O/P ‘dnekeew nohtyp’



1. O/P ‘deekenw nythop’



***To know all the methods supported,execute the following commands***

***String = dir(‘’)***

***List = dir([])***

***Tupple = dir(())***

***Set = dir({})***

**Operation on grouped data items**

***String:***

1. count: count is a function which is used to count the number of occurrence of a substring in a main string

Strvar/value.count(substring)

Strvar/value.count(substring,startIndex)

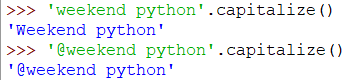
Strvar/value.count(substring,startIndex,endIndex)

If the substring is not present in the main string then the count function returns 0,

* The boundries are not mandatory
* If the count function is given with single argument then by default start index and endIndex will be 0 and len(String) respectively
* If the count function is given with two arguments then by default, then endIndex will be len(String)

2.Capitalize: Capitalize function is used to capitalize the first character present in the string to upper case

strVar/value.capitalize()



3.startswith: starts with function is used to check weather the string is starting with the given substring or not and returns Boolean value true or false

Strvar/value.startswith(substring)

Strvar/value.startswith(substring,startIndex)

Strvar/value.startswith(substring,startIndex,endIndex)

4.endswith: endswith function is used to check weather the string is ending with the given substring or not and returns the Boolean value true or false

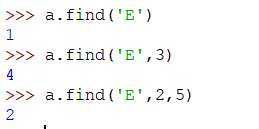
Strvar/value.endswith(substring)

Strvar/value.endswith(substring,startIndex)

Strvar/value.endswith(substring,startIndex,endIndex)

5.find: find function is used to findout the position of the first occurance of the substring between the boundries, it always gives the first occarance position

If the substring is not present in the main string then find function returns the result as -1

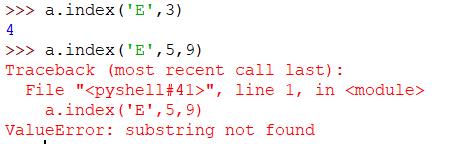


Strvar/value.find(substring)

Strvar/value.find(substring,startIndex)

Strvar/value.find(substring,startIndex,endIndex)

6.Index: it is similar to find function but the only difference is if the substring is not present in the main string then the index function throws an error



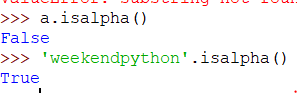
Strvar/value.index (substring)

Strvar/value.index(substring,startIndex)

Strvar/value.index(substring,startIndex,endIndex)

7.isalpha: it is used to check whether the entered string is having only alphabets or not returns the Boolean result true or false

strVar/value.isalpha()



8.isdigit: it is used to check whether the entered string is having only the numbers or digits and returns the Boolean result true or false

strVar/value.isdigit()

9. lower(): lower function is used to convert the data into lower case ie if any character in upper case into lower case

Strvar/value.lower()

10.Upper(): upper function is used to convert the data into upper case ie if any character in lower case then it is converted into upper case

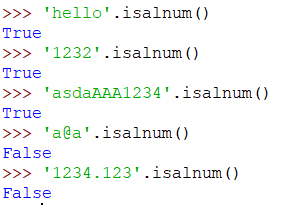
strVar/value.upper()

11. isalnum: It is a function which is used to check whether the string is collection of either characters or numbers or combination of both and returns the Boolean result true or false

If it is collection of only character or collection of only number or combination of both it returns the Boolean result --- true

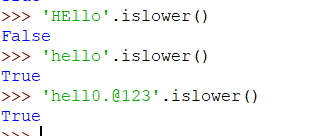
If the string contains any special character it returns the Boolean result false

Strvar/value.isalnum()



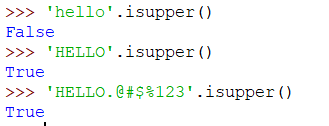
12. islower(): it is used to check all the character in the entered string is in lower case or not and returns Boolean result, true or false

Strvar/value.islower()



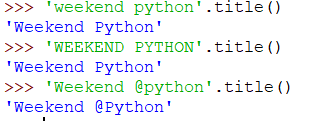
13.isupper() : it is used to check all the charaters in the entered string is in upper case or not and returns Boolean result, true or false

Strvar/value.isupper()



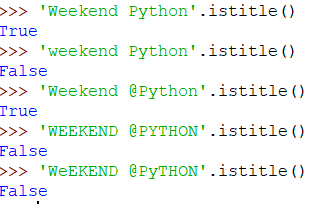
14. title(): It is a function which is used to convert the starting character of all the words in a string to upper case

Strvar/value.title()



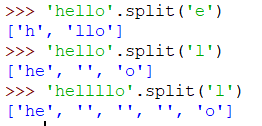
15. istitle(): it is a function which is used to check whether the starting character of each word in the string is in upper case or not and returns the Boolean result true or false

If all the words starting character is in upper case and rest of the character in lower case then it returns the Boolean result true



16. split(splitString): it is used to break the string into smaller pieces whenever the given substring or the split string occurred or present

Strvar/value.split(‘splitString’)



\*\*\*\*\* understand the split with l in the above example

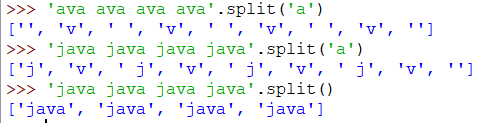
Assignement:-



\*\*\*Store each element of a string as element in the list

list(‘hello world’)

***Note:*** When ever there is not argument passed to the split function then it is considered as a space



A String can also be split based on count

Strvar/value.split(‘splitstring’,count)



The total number of split words are always equals to count plus 1

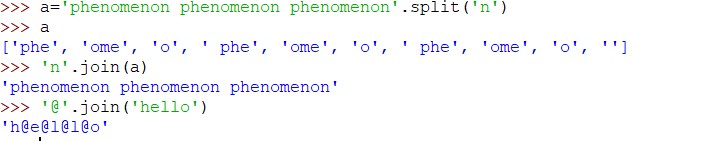
\*\*\*If the count is not mentioned then the total number of elements are equal to number of occurrences of split string plus 1

17. join: Join function is used to join the collection of string as a single string with the help of gluestring

While joining the data the first priority is given to glue string

Gluestring can be single character or more than one character of any type

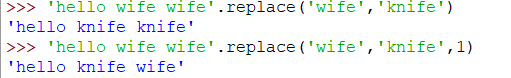
‘gluestring’.join(argument)---------- argument must be of iterable type



18. replace: replace function is used to replace the older substring in a string with the new substring

Strvar/value.replace(‘old substring’,’new substring’)

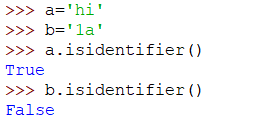
Strvar/value.replace(‘old substring’,’new substring’, count)



Based on the count value given replace function desides how many times the older substrings to be replaced with new substring

19. isidentifier: this function is used to check whether the content present inside the string can be a name given to any memory location or not

Strvar/value.identifier()

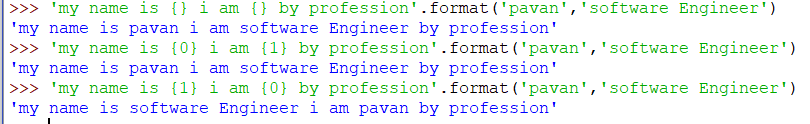


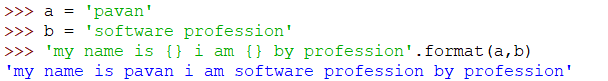
20. format: format function which is used to substitute the given value into a specified place holders

“{} {}”.format(arguments)

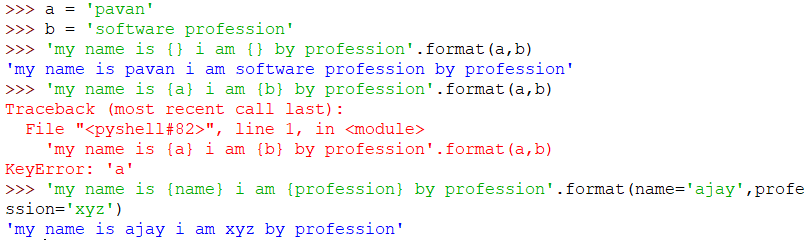
It can have n number of place holders within a string

By default the index value of the place holder starts with 0





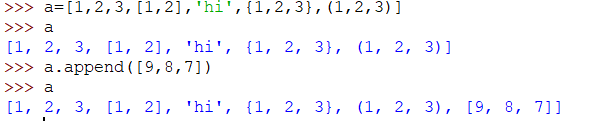
Analyse the example below



**Operations on list**

1. **Append:** This is used to insert an element at the end of the list

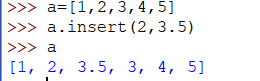
**listvar/value.append(element)**



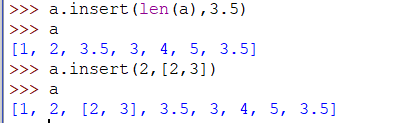
Here the original values will be modified after appending the data

1. **Insert:** it is used to insert the element at the position specified by the user

**Listvar/value.insert(position,element)**

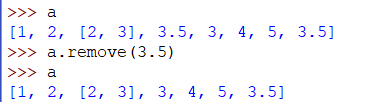


**When the insert position specified then the element present at that position will be shifted towards right by one bit**



1. **Remove: it is used to remove the first occurance of user specified element in the list**

**Listvar/value.remove(value/element)**

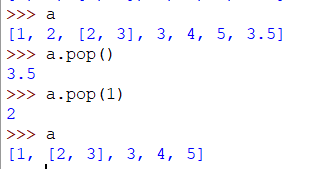


1. **Pop : if the function is called without the argument then it always removes last element**

**If the function is called with argument then it removes the element based on the index given by the user**

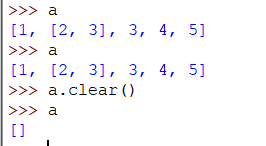
**Listvar/value.pop()**

**Listvar/value.pop(index)**

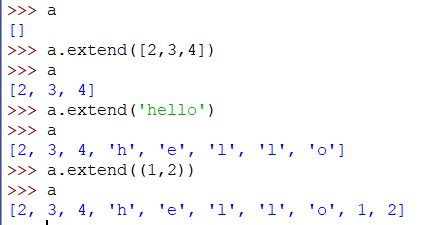


1. **Clear: clear function is used to remove all the elements present in the list**

**Listvar/value.clear()**

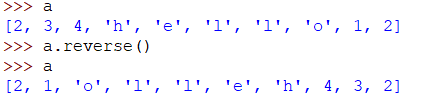


1. **Extend: This is used to add the elements of list as individual elements at the end of list**



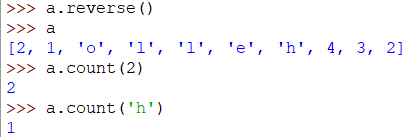
1. **Reverse: it is used to reverse the values of list**

**Listvar/value.reverse()**



1. **Count: It is used to count the occurance of value in a list**

**Listvar/value.count(value)**



1. **Index: It is used to findout the index of an element between the boundries**

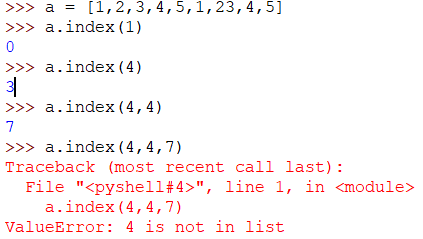
**Boundries are not mandatory**

**If the index function is given with only single argument, then by default start index is 0 and end index is len of the list**

**Listvar/value.index(value)**

**Listvar/value.index(value,startIndex)**

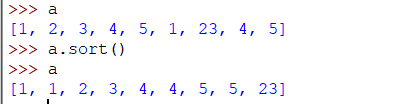
**Listvar/value.index(value,startIndex,endIndex)**



1. **Sort:** sort function is used to sort or arrange only the ***homogeneous*** elements in ascending order

Here the original values will be modified

Listvar/value.sort()



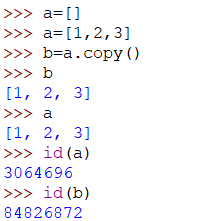
1. Copy: copy is of two types,
   1. Shallow copy
   2. Deep copy

*Shallow copy* It is a process of copying the content of one variable into another variable, is known as shallow copy

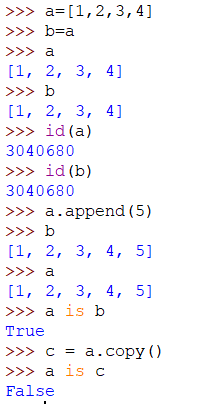
Shallow copy operation which can be achieved using copy()

a=[]

a=listvar/value.copy()



*Deep copy* It is a process of copying one variable address into another variable is known as deep copy, to perform deep copy operation we use =(assignment operator)



**Operations on tupple**

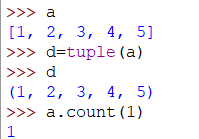
**Insertion and deletion is not possible in tupple, thus the only operation which can be performed on the tupple variables are the opertions which doesn’t affect the tupple variable**

**The only two function of tupple are count and index**

**If a single element whithout , is enclosed within () then it is never considered as tuple**

1. **Count():** It is used to count the number of elements present in the tupple

**Tupplevar/value.count(data)**



1. **Index (): It is used to get the index of the value entered**

**Tupplevar/value.index(value)**

**Tupplevar/value.index(value,startIndex)**

**Tupplevar/value.index(value,startIndex,endIndex)**

**Operations on set**

1. **add():** add function is used to add an element in the set

**setvar/value.add(value)**

1. **copy():** it is used to perform shallow copy operation

**variablename=setvar/values.copy()**

1. **clear():** it is used remove all the elements present in the set and returns empty set

**setvar/value.clear()**

1. **del:** it is used to delete the memory containing the data item as well as the memory where it is stored

**del var**

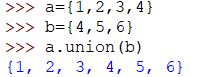
1. **difference:** difference function is used to identify, the elements of set 1 which is not present in set 2

this works similar to – operation is set

set1.difference(set2)

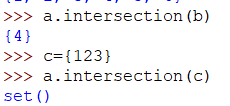
1. **union**: it is used to identify only the unique elements from both the sets

**here a.union(b) is equal to b.union(a)**



1. **Intersection**: it is used to identify only the common elements between sets

**Set1**.intersection(set2)



1. **Isdisjoint**: it is used to check both the sets are having any common elements or not and returns the Boolean result true or false

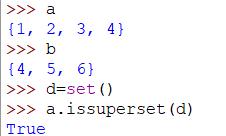
If there are no common elements then it will return false

Set1.isdisjoint(set2)

1. **Issuperset**: in the below example in case of d it contains all the elements in a then a is called as super set of d, In set b since having the additional elements which is not present in then a is not called as superset

**In general issuperset is used to check whether the set1 is super set of set2 or not**

**If the set1 is superset of set2 it returns true/false otherwise**



1. **Issubset**: if the set contains all the elements which is same as ***some*** of the elements of another set then it is called as subset

**If the set2 is derived only from set1 then set2 is called as subset of set1**

**Set2.issubset(set1)**

1. **Pop():** it is used to remove the **first** element present in the set

**Operations on dictionary**

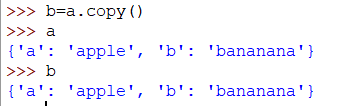
1. **Len():** it is used to return the length of the dictionary, ie it returns the number keys

**Len(dictvar/value)**



1. **Copy():** it is used to copy data items from one dictionary variable to another dictionary variable

It is used to perform shallow copy

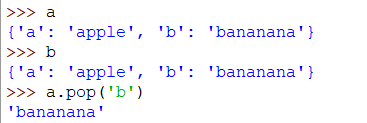


1. Pop(key): it is used to remove the user specified data item

It can take only the key or both key and value to remove the user specified item

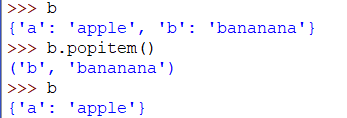
Dictvar/value.pop(key)

Dictvar/value.pop(key,value)



1. **Popitem():**  it is used to remove key and value pair which is present at the end of dictionary

**Dictvar/value.**popitem()

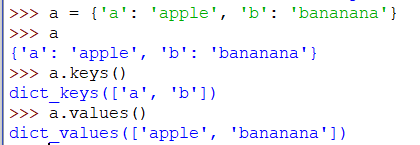


1. **Keys:** it is used to get all the dictionary keys

**Dictvar/value.**keys()

1. Values: it is used to return all the dictionary values

Dictvar/value.values()



1. **Items():** items function is used to display all the functions which are present in the dictionary

**Dictvar/value.**items()



1. **Setdefault:** set default function which is used to assign the default generic value to a specified key

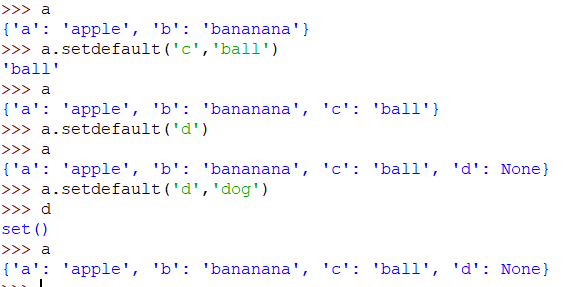
**When ever pass single argument to the set default function ie key it creates a new key and assigns the value as the default generic value**

If the set default function is given with 2 arguments ie key and values ,if the key is not present then creates key value pair in the dictionary

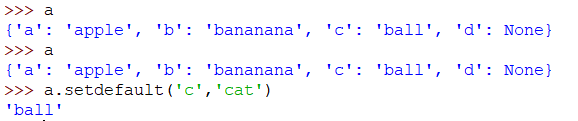
**Set default function will never override the dictionary keys**

**Dictvar/value.setdefault(key)**

**Dictvar/value.setdefault(key,value)**



**When ever we try to override the dictionary key it will display the value present for that dict key instead of overriding**

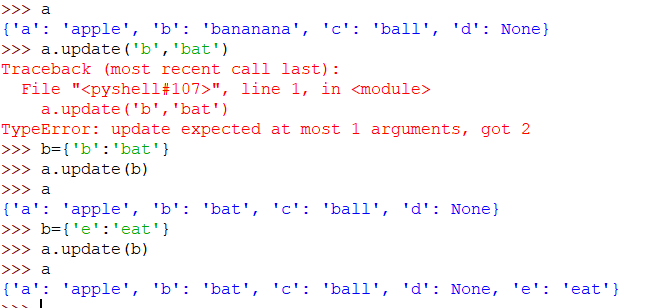


1. **Update:** update function which is used to update given dictionary with the specified variable

**If the variable contains the key is already present in the dictionary then it overrides the value of key in the dictionary**

If the variable doesnot contain the key which is used to update in the dictionary then it appends a new key value pair at the end of dictionary

Dictvar/value.update(var) **where var is of dict type**



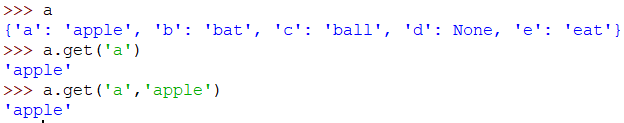
1. **Get(): get function which is used to the value of the give dictionary key**

**We can get the value of a key as argument to the get function or by passing both key and value**

**The second argument ie value is optional for the get function**

**Dictvar/value.get(key)**

**Dictvar/value.get(key,value)**



**Operators**

Operators are the special symbols which are used to perform the operations on the operands

**Operands** are nothing but the data or the variable on which the operations are performed

Operators are classified into the following categories

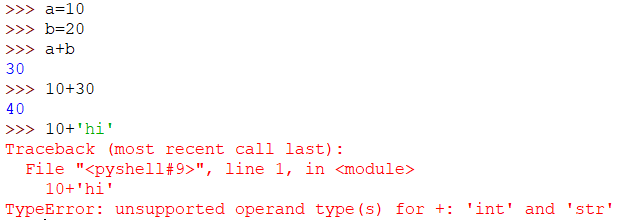
1. Arithmetic operators
2. Logical Operators
3. Relational Operators
4. Bitwise operators
5. Assignment operators
6. Membership operators
7. Identity operators

**Arithmetic Operators**

|  |  |
| --- | --- |
| **Symbol** | **Description** |
| + | Addition/Concatenation |
| - | Subtraction |
| \* | Multiplication/Replication |
| % | Modulus |
| / | True Division |
| // | Floor Division |
| \*\* | Power |

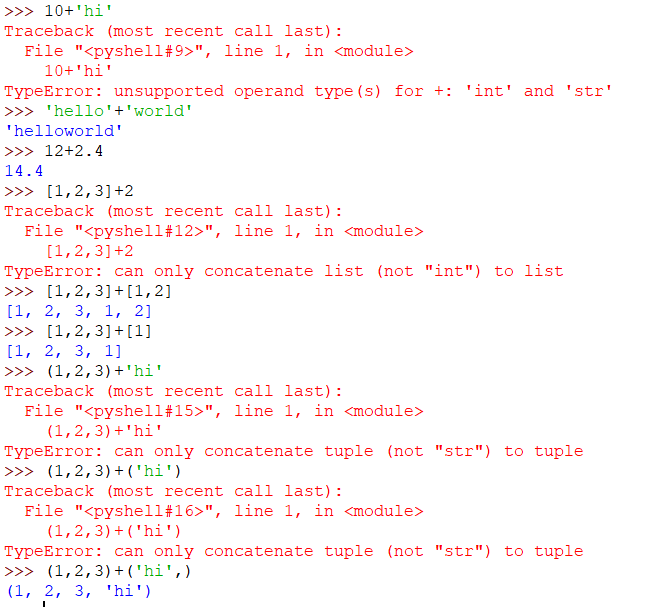
**Addition/concatenation:**

To perform concatenation operation both the operands should be of **same type**



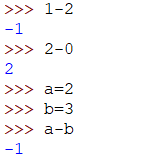
**If a single element whithout , is enclosed within () then it is never considered as tuple**

**Analyse**



**Subtraction:**

**Subtraction** is used to perform operation only on numeric data and returns signed or unsigned data

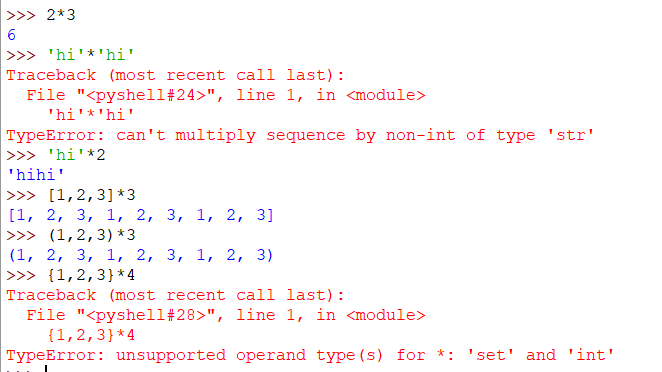


**Multiplication/Replication**

If the operand is of same type then it performs multiplication operation

Replication: same set of values repeating again and again is knows as replication

***NOTE:*** replication operation cannot be performed on set grouped data type

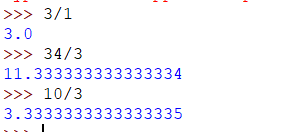


**Modulus:** it returns the reminder, and this operator can be used only on numerics

10%3 = 1

**True Division**

In other programming language, int/int will return int but in case of python int/int will always give float

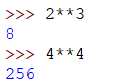


**Floor division**

Decimal value which lies between 2 set of integer known as floor and ceil value always floor value must be less than ceil value

|  |  |
| --- | --- |
| 4 -------- ceil  10//3 = 3.3333  3 --------- floor  -3 ------- ceil  -10//3 = -3333  -4 ------ floor |  |

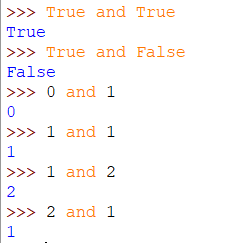
**Power**:



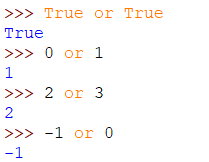
**Logical Operator**

It used to check the given conditions and return Boolean results

1. **AND :** it is used to check whether the given both the operands are True

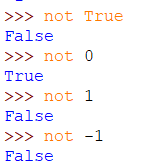
 Analyse the last 2 in the example (if the value is more 1 then it returns the second operand)

1. **OR:** it is used to check whether the given either of the operand is true

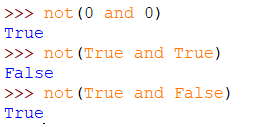
 Analyse the last 2 in the example (if the value is more 1 then it returns the first operand)

1. **Not**: It returns the inverse result that is if the condition is true it returns false and vise versa

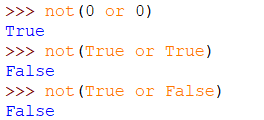
Always the result of not logical operator will be the Boolean result true or false



1. Nand: Nand is negation of and (**not (op1 and op2)**)



1. Nor: Nor is negation of OR (**not(op1 or op2)**)

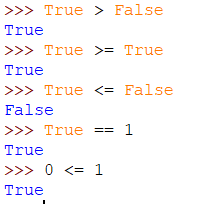


**Relational Operators**

**Relational operators are used to check the relation between the operands**

**Always the result of relational operators will be Boolean result true or false**

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



**Bitwise Operators**

Bitwise operators are used to perform the operations on binary representation of the values

|  |  |
| --- | --- |
| **Operators** | **Operation** |
| | | Bitwise OR |
| & | Bitwise And |
| ^ | Bitwise xor |
| << | Bitwise left shift |
| >> | Bitwise right shift |
| ~ | Bitwise not |

***Bitwise Or (op1 | op2)***

|  |  |  |
| --- | --- | --- |
| **8 🡪 1000**  **12 🡪 1100**  **1100 🡪 12** | 12 🡪 1100  16 🡪 10000  11100 🡪 28 | 23 🡪 010111  32 🡪 100000  110111 🡪 55 |
| **8 🡪 1000**  **27🡪 11011**  **11011 🡪 27** |  |  |

Note: In bitwise or operation if both the bits are 1 or either of the bits are 1, then it returns 1 else it returns 0

***Bitwise And (op1 & op2)***

|  |  |  |
| --- | --- | --- |
| **8 🡪 1000**  **12 🡪 1100**  **1000 🡪 8** | 12 🡪 01100  16 🡪 10000  00000 🡪 0 | 23 🡪 010111  32 🡪 100000  000000 🡪 0 |
| **8 🡪 1000**  **27🡪 11011**  **01000 🡪 8** |  |  |

Note: In bitwise and logical operator only if both the bits are 1 then it returns 1 else it returns 0

***Bitwise Xor (op1 ^ op2)***

|  |  |  |
| --- | --- | --- |
| **8 🡪 1000**  **12 🡪 1100**  **0100 🡪 3** | 12 🡪 01100  16 🡪 10000  11100 🡪 28 | 23 🡪 010111  32 🡪 100000  110111🡪 55 |
| **8 🡪 1000**  **27🡪 11011**  **10011 🡪 19** |  |  |

*Note:* if both the bits are 0 or if both the bits are 1 it returns 0 else it returns 1

***Bitwise right shift (op1 >> op2)***

Op1 🡪 number to be shifted

Op2 🡪 number of times the number to be shifted

In bitwise right shift operation for every shift given number is floor divided by 2 (x//2)

|  |  |  |
| --- | --- | --- |
| **8 >> 1**  **8** 🡪 1000 🡺 0100 = **4** | **16 >> 3**  **16** 🡪 10000 🡺 01000 🡺 00100 🡺 00010 = **2** | **15 >> 3**  15 🡪 1111 🡺 0111 🡺 0011 🡺 0001= **1** |

***Bitwise left shift (op1<<op2)***

Op1 🡪 number to be shifted

Op2 🡪 number of times the number to be shifted

***Result :*** Given number \* 2n (n is op2)

|  |  |  |
| --- | --- | --- |
| 4 << 1  4 🡪 0100 🡺 1000 = 8 | 4 << 2  4🡪 0100 🡺 1000 🡺 10000 = 16 |  |

***Bitwise not (~n)***

~ n = -(n+1)

~4 = -(4+1) = -5

~-5 = -(-5+1) = 4

**Assignment operators**

Assignment operators are used to assign a value to a variable after performing some particular operation

|  |  |
| --- | --- |
| = |  |
| += |
| -= |
| \*= |
| /= |
| //= |
| \*\*= |
| %= |
| >>= |
| <<= |

**Membership operators**

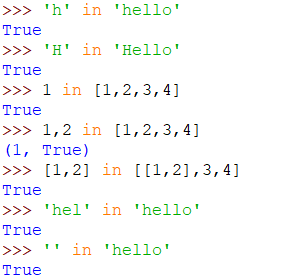
Membership operators are used to chech whether the values or the variables is present in the group or not and it returns Boolean result true or false

* In
* Not in

**In:** it is used to check whether the value or variable is present in the group

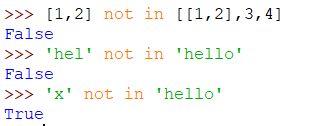
If the value is present in the group it returns true else it returns false

Var/value in group



**Not In**: It is used to check whether the value or variable is not present in the group

If it is not present then it returns the Boolean result true else it returns false



**Identity operators**:

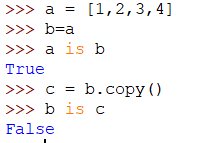
identity operators are used to check whether both the variables are pointing to the same memory or not and it returns the boolean result true or false

* Is
* Is not

**Is:** is identity operator is used to check whether both the variables are pointing to same memory, if both the variables are pointing to same memory then it returns the boolean result True else it returns false

Var is var1

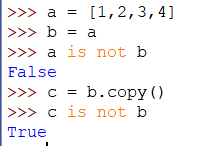
*Analyse the below example*

read the concept of shallow copy and deep copy\*\*\*\*\*

**Is Not:** it is used to check whether both the variables are not pointing to same memory

If it is not pointing to same memory then it returns the Boolean result true else it returns false

Var is not var1



**Control statements:**

These are the statements which are used to change the flow of execution of the program based on the conditions

Control statements are of two variants

* Decisional statements
* Looping statements

|  |  |
| --- | --- |
| **Decisional Statements** | **Looping statements** |
| If | For |
| If else | While |
| Elif |  |
| Nested if |  |

***Decisional statements***

These are the set of instructions or the statements which works on certain decisions,

**If** : simple if statement is used to check single condition

If Condition : **:** denotes that the body contains more than 1 line

-------

-------

True block statements

--------

--------

uname = 'admin'

pwd = 'pass'

print('this statement is before if block')

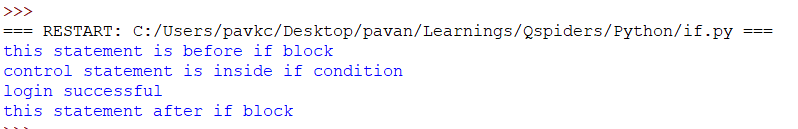
if uname=='admin' and pwd=='pass':

print('control statement is inside if condition')

print('login successful')

print('this statement after if block')

Output:



**IF else:** it is used when ever there are two possible option for a condition and either of one should be executed in such case we use if else statements

**If condition:**

**-----------------**

**-----------------**

**else:**

**----------------------**

**--------------------**

**uname='admin'**

**pwd='pass'**

**print('before if else')**

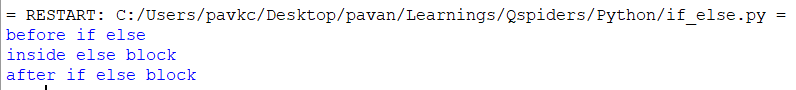
**if uname=='admin123' and pwd=='pass':**

**print('if condition is satisfied')**

**else :**

**print('inside else block')**

**print('after if else block')**



**Write a program to find the greatest of two numbers**

**a=10**

**b=20**

**if a>b:**

**print('variable a with value:'+str(a)+' is greater')**

**else:**

**print('variable b with value:'+str(b)+' is greater')**

**elif: (if else ladder)**

*syntax*

if condition1:

true block statement

elif condition2:

true block statement

| |

| |

else

else block statement

>write a program to print the greatest of three numbers

a,b,c=10,20,3

if a>b and a>c:

print("a is greater"+str(a))

elif b>c:

print("b is greater"+str(b))

else:

print("c is greater"+str(c))

output:

c is greater 30

* elif ladder is used whenever there are multiple conditions available and one of the condition is satisfied, the statements which are written in which the condition is satisfied will be executed
* if none of the multiple conditions is not satisfied then the default statement block will be executed

print("done with +2 and cet and waiting for counselling")

print("going to counselling")

branch=input("enter the branch").upper()

if branch=="ECE":

print("selected ECE")

elif branch=="CSE":

print("selected CSE")

elif branch=="EEE":

print("selected EEE")

else:

print("selected Mechanical")

**Output:**

done with +2 and cet and waiting for counselling

going to counselling

enter the branchece

selected ECE

* find the greatest of four numbers using elif ladder

a,b,c,d=input("enter a"),input("enter b"),input("enter c"),input("enter d")

if a>b and a>c and a>d:

print("a "+str(a)+ " is greatest")

elif b>c and b>d:

print("b "+str(b)+" is greater")

elif c>d:

print("c "+str(c)+" is greater")

else:

print("d "+str(d)+" is greater")

Output:

enter a20

enter b30

enter c60

enter d10

c 60 is greater

* ATM

withdrawal=int(input("enter the amount to be withdrawn "))

atm\_balance=200

user\_account\_balance = 180

if withdrawal<100:

print("invalid amount entered please enter amount in multiple of 100")

elif withdrawal<atm\_balance and withdrawal<user\_account\_balance:

atm\_balance-=withdrawal

current\_balance= user\_account\_balance-withdrawal

print("you have successfully withdrawn "+str(withdrawal)+" and user current balance is "+str(current\_balance))

elif withdrawal>atm\_balance or withdrawal>user\_account\_balance:

if withdrawal>atm\_balance:

print("no sufficent funds available")

else:

print("withdrawal amount is greater than amount in your account")

else:

print("atm is out of service")

enter the amount to be withdrawn 100

you have successfully withdrawn 100 and user current balance is 80

* Write a program to print greatest of 5 numbers

Write with sort

**Nested if:** whenever there are multiple conditions in a single line separated using and logical operator then we call it as a condition eligble for a nested if statement

Nested if statement is the if condition within if condition

If condition:

If condition:

True block statement

else condition

False block statement

Elif condition:

----

Write a program to display greatest of 3 numbers using nested if

a,b,c=1,2,3

if a>b:

if a>c:

print("a is greater")

else

print("c is greater")

elif b>c:

print("b is greater")

else:

print("c is greater")

**Looping statements/iterative statements:**

Looping statements are the statements whose starting point and the ending point are the same

* Loop statements are also called as self repetitive statements until the condition is been satisfied
* In loops starting point and the ending points are the same
* There are two types of looping statements
  + While
  + For

***While***

Syntax: while condition:

----------

-----------

----------

Write the flow chart

While loop is a statement, which is used to execute set of statements again and again until the given condition is satisfied

While loop is used whenever the number of iterations are unknown

i=1

while i<=10:

print(i,end=' ')

i+=1

1 2 3 4 5 6 7 8 9 10

# Write a program to print only the even numbers between 1 to 10

i=1

while i<=10:

if i%2==0:

print(i,end=' ')

i+=1

**For Loop:**

It is a statement which will be executed for each and every item in the rule

**Range:** Range is a function which is used to generate the range of values

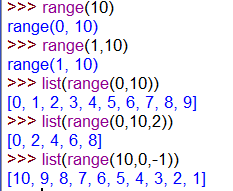
We cannot see the generated values until it is typecasted\*\*

Range function take minimum of 1 argument and max of 3 arguments

Range(endpoint) By default start point is 0 and updatevalue is 1

Range(start,endpoint) By default updatevalue is 1

Range(start,endpoint,updatevalue)



For loop can loopover the grouped data items or the groups

* The groups which can be looped over is String List Tuple Set or Dictionary

for i in range(1,10):

print(i) 🡪 1,2,3,4,5,6,7,8,9

for i in range(-10,-1+1):

print(i) 🡪 -10,-9,-8,-7,-6,-5,-4,-3,-2,-1

for i in range(10,1,-1):

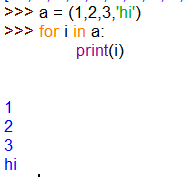
print(i) 🡪 10,9,8,7,6,5,4,3,2

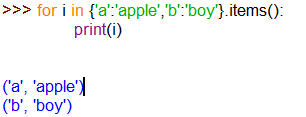
for i in ‘hello’:

print(i,end=’ ’) 🡪 h e l l o

for i in [1,2,3,4]

print(i) 🡪 1 2 3 4

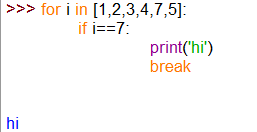




Note: It is not possible to get both key and value without any built in functions, in order to achive that the group function has to be defined in a variable….

***Statements which is used to halt or break or intrupt the flow of execution***

**Break:** break statement which is used to break the loop and make the control come out of the loop



a=[1,2,3,4,7,5]

i=1

while i<=len(a):

if a[i]==7:

print('element found')

break

i+=1

**Continue:** Continue statement which is used to come out of current execution and work normally for other executions

***Using For loop***

i=1

for i in range(1,10+1):

if i==5:

continue

print(i)

***Using while loop***

*i=1*

*while i<=10:*

*if i==5:*

*i+=1*

*continue*

*print(i)*

*i+=1*

# Print 1 2 3 4 8 9 10 using for and while

**Using For loop**

for i in range(1,10+1):

if i>=5 and i<=7:

continue

print(i,end=' ')

**Using while loop**

i=1

while i<=10:

if i in (5,6,7):

i+=1

continue

print(i,end=' ')

i+=1

**Print:** print is a statement which is used to print the variable value in output console

Syntax

* Print value in new line 🡪 print(var/value/String)
* Print value in same line 🡪 print(var/value/string , end=’ ‘)

**Pattern Programs**

# Print 5 rows with 5 stars in each row

n =5

for i in range(1,n+1):

for j in range(1,n+1):

print('\*',end=' ')

print()

\* \* \* \* \*

\* \* \* \* \*

\* \* \* \* \*

\* \* \* \* \*

\* \* \* \* \*

To Print any pattern follow the below diagram with 2 for loops and conditions in if condition

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  |  |  |  | | --- | --- | --- | --- | --- | | 1,1 | 1,2 | 1,3 | 1,4 | 1,5 | | 2,1 | 2,2 | 2,3 | 2,4 | 2,5 | | 3,1 | 3,2 | 3,3 | 3,4 | 3,5 | | 4,1 | 4,2 | 4,3 | 4,4 | 4,5 | | 5,1 | 5,2 | 5,3 | 5,4 | 5,5 | |

|  |  |
| --- | --- |
|  | i==j |
|  | i+j==n+1 |
|  | i>=j |
|  | I+j<=n+1 |
|  | I<=j |
|  | i+j>=n+1 |
|  | i>=j and i+j<=n+1 |
|  | i<=j and i+j>=n+1 |
|  | i>=j and i+j>=n+1 |
|  | I<=j and i+j<=n+1 |
|  | I<=j and i+j<=n+1 or i>=j and i+j>=n+1 |
|  | i>=j and i+j<=n+1 or i<=j and i+j>=n+1 |

# write a program and print row value 5 times

n =5

for i in range(1,n+1):

for j in range(1,n+1):

print(i,end=' ')

print()

*Output*

1 1 1 1 1

2 2 2 2 2

3 3 3 3 3

4 4 4 4 4

5 5 5 5 5

# Write a program to print column numbers in each row

n =5

for i in range(1,n+1):

for j in range(1,n+1):

print(j,end=' ')

print()

*output*

1 2 3 4 5

1 2 3 4 5

1 2 3 4 5

1 2 3 4 5

1 2 3 4 5

# write a program to print below pattern

\* # # # #

$ \* # # #

$ $ \* # #

$ $ $ \* #

$ $ $ $ \*

n =5

for i in range(1,n+1):

for j in range(1,n+1):

if i==j:

print('\*',end=' ')

elif i>j:

print('$',end=' ')

else:

print('#',end=' ')

print()

# Write a program to print below pattern

\*

\*

\* \* \* \* \*

\*

\*

n =5

for i in range(1,n+1):

for j in range(1,n+1):

if i==(n+1)/2 or j==(n+1)/2:

print('\*',end=' ')

else:

print(' ',end=' ')

print()

# write a program to print below pattern

\* \*

\* \*

\*

\* \*

\* \*

n =5

for i in range(1,n+1):

for j in range(1,n+1):

if i==j or i+j==(n+1):

print('\*',end=' ')

else:

print(' ',end=' ')

print()

# write a program to print the below pattern

\* \* \* \* \*

\* \* \* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

n =5

for i in range(1,n+1):

for j in range(1,n+1):

if i==j or i+j==(n+1) or i==1 or j==1 or i==n or j==n:

print('\*',end=' ')

else:

print(' ',end=' ')

print()

# write a program to print below pattern

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

n =5

for i in range(1,n+1):

for j in range(1,n+1):

if i>=j :

print('\*',end=' ')

print()

#write a program to print

1

1 0

1 0 1

1 0 1 0

1 0 1 0 1

n =5

for i in range(1,n+1):

for j in range(1,n+1):

if i>=j :

if j%2==0:

print(0,end=' ')

else:

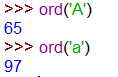
print(1,end=' ' )

else:

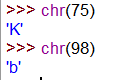
print(' ',end=' ')

print()

**ord :** ord is a function which is used to get the ascii value of the respective characters



**chr:** chr is a function which is used to get the character of the respective value



Write a program to print below pattern with ascii value

A

A A

A A A

A A A A

A A A A A

n =5

asciiVal=ord('A')

for i in range(1,n+1):

for j in range(1,n+1):

if i>=j:

print(chr(asciiVal),end=' ' )

else:

print(' ',end=' ')

print()

Note: since range function do not work for alphabets to increment we have to use ascii

#write a program to print the below pattern

*A*

*B B*

*C C C*

*D D D D*

*E E E E E*

n =5

asciiVal=ord('A')

for i in range(1,n+1):

for j in range(1,n+1):

if i>=j:

print(chr(asciiVal+i-1),end=' ' )

else:

print(' ',end=' ')

print()

#write a program to print the below pattern

A

A B

A B C

A B C D

A B C D E

n =5

asciiVal=ord('A')

for i in range(1,n+1):

for j in range(1,n+1):

if i>=j:

print(chr(asciiVal+j-1),end=' ' )

else:

print(' ',end=' ')

print()

#write a program to print the below pattern

1

A B

1 2 3

A B C D

1 2 3 4 5

n =5

asciiVal=ord('A')

for i in range(1,n+1):

for j in range(1,n+1):

if i>=j:

if i%2==0:

print(chr(asciiVal+j-1),end=' ' )

else:

print(j,end=' ')

else:

print(' ',end=' ')

print()

#Write a program to print below pattern

1

1 B

1 B 3

1 B 3 D

1 B 3 D 5

n =5

asciiVal=ord('A')

for i in range(1,n+1):

for j in range(1,n+1):

if i>=j:

if j%2==0:

print(chr(asciiVal+j-1),end=' ' )

else:

print(j,end=' ')

else:

print(' ',end=' ')

print()

#write a program to print below pattern

1

1 A

1 A 2

1 A 2 B

1 A 2 B 4

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

n =5

for i in range(1,n+1):

for j in range(1,n+1):

if i+j>=n+1:

print('\*',end=' ')

else:

print(' ',end=' ')

print()

1

0 0

1 1 1

0 0 0 0

1 1 1 1 1

n =5

for i in range(1,n+1):

for j in range(1,n+1):

if i+j>=n+1:

if i%2==0:

print(0,end=' ')

else:

print(1,end=' ')

else:

print(' ',end=' ')

print()

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

n = 5

for i in range(1,n+1):

k=1

for j in range(1,n+1):

if i+j>=n+1:

print(k,end=' ')

k+=1

else:

print(' ',end=' ')

print()

# assignement

A

B A

C B A

D C B A

E D C B A

n = 5

for i in range(1, n+1):

asciiVal = ord('E')

for j in range(1, n+1):

if i+j >= n+1:

print(chr(asciiVal), end=' ')

else:

print(' ', end=' ')

asciiVal -= 1

print()

# assignement

A

B C

D E F

G H I J

K L M N O

n = 5

asciiVal = ord('A')

for i in range(1, n+1):

for j in range(1, n+1):

if i+j >= n+1:

print(chr(asciiVal), end=' ')

asciiVal += 1

else:

print(' ', end=' ')

print()

# assignement

A

4 A

B 4 A

2 B 4 A

C 2 B 4 A

n = 5

for i in range(1, n+1):

asciiVal = ord('D')

for j in range(1, n+1):

if j % 2 == 1:

asciiVal -= 1

if i+j >= n+1:

if j % 2 == 1:

print(chr(asciiVal), end=' ')

else:

print(j, end=' ')

else:

print(' ', end=' ')

print()

\* \* \* \* \*

\* \* \* \*

\* \* \*

\* \*

\*

n = 5

for i in range(1,n+1):

for j in range(1, n+1):

if i<=j:

print('\*',end=' ')

else:

print(' ',end=' ')

print()

1 2 3 4 5

2 3 4 5

3 4 5

4 5

5

n =5

for i in range(1,n+1):

for j in range(1,n+1):

if i<=j:

print(j,end=' ')

else:

print(' ',end=' ')

print()

1 2 3 4 5

1 2 3 4

1 2 3

1 2

1

n =5

for i in range(1,n+1):

l=1

for j in range(1,n+1):

if i<=j:

print(l,end=' ')

l+=1

else:

print(' ',end=' ')

print()

5 4 3 2 1

5 4 3 2

5 4 3

5 4

5

n =5

for i in range(1,n+1):

k=n

for j in range(1,n+1):

if i<=j:

print(l,end=' ')

k-=1

else:

print(' ',end=' ')

print()

A B C D E

B C D E

C D E

D E

E

n =5

asciiVal=ord('A')

for i in range(1,n+1):

for j in range(1,n+1):

if i<=j:

print(chr(asciiVal+j-1),end=' ')

else:

print(' ',end=' ')

print()

3 4 5 6 7

3 4 5 6

3 4 5

3 4

3

n = 5

for i in range(1,n+1):

k=3

for j in range(1,n+1):

if i<=j:

print(k,end=' ')

k+=1

else:

print(' ',end=' ')

print()

E D C B A

E D C B

E D C

E D

E

n =5

for i in range(1,n+1):

k=ord('E')

for j in range(1,n+1):

if i<=j:

print(chr(k),end=' ')

k-=1

else:

print(' ',end=' ')

print()

A B C D E

4 3 2 1

A B C

4 3

A

n =5

for i in range(1,n+1):

k,l=ord('A'),n

for j in range(1,n+1):

if i<=j:

if i%2==1:

print(chr(k),end=' ')

k+=1

else:

print(l-1,end=' ')

l-=1

else:

print(' ',end=' ')

print()

# Assignment

I H G F E

6 5 4 3

D C B

2 1

A

n = 5

asciiVal = ord('I')

l = 6

for i in range(1, n+1):

for j in range(1, n+1):

if i <= j:

if i % 2 == 1:

print(chr(asciiVal), end=' ')

asciiVal -= 1

else:

print(l, end=' ')

l -= 1

else:

print(' ', end=' ')

print()

\* \* \* \* \*

\* \* \* \*

\* \* \*

\* \*

\*

n = 5

for i in range(1,n+1):

for j in range(1,n+1):

if i+j<=n+1:

print('\*',end=' ')

else:

print(' ',end=' ')

print()

1 1 1 1 1

2 2 2 2

3 3 3

4 4

5

n = 5

for i in range(1,n+1):

for j in range(1,n+1):

if i+j<=n+1:

print(i,end=' ')

else:

print(' ',end=' ')

print()

1 0 1 0 1

1 0 1 0

1 0 1

1 0

1

n = 5

for i in range(1,n+1):

for j in range(1,n+1):

if i+j<=n+1:

if j%2==1:

print(1,end=' ')

else:

print(0,end=' ')

else:

print(' ',end=' ')

print()

1 0 3 0 5

1 0 3 0

1 0 3

1 0

1

n = 5

for i in range(1,n+1):

for j in range(1,n+1):

if i+j<=n+1:

if j%2==1:

print(j,end=' ')

else:

print(0,end=' ')

else:

print(' ',end=' ')

print()

1 1 3 1 5

1 2 3 2

1 3 3

1 4

1

n = 5

for i in range(1,n+1):

for j in range(1,n+1):

if i+j<=n+1:

if j%2==1:

print(j,end=' ')

else:

print(i,end=' ')

else:

print(' ',end=' ')

print()

E D C B A

1 2 3 4

E D C

5 6

E

n,l=5+1

for i in range(1,n+1):

k=ord('E')

for j in range(1,n+1):

if i<=j:

if i%2==1:

print(chr(k),end=' ')

k-=1

else:

print(l,end=' ')

l+=1

else:

print(' ',end=' ')

print()

E D C B A

1 2 3 4

E D C

5 6

E

n,l=5,1

for i in range(1,n+1):

k=ord('E')

for j in range(1,n+1):

if i+j<=n+1:

if i%2==1:

print(chr(k),end=' ')

k-=1

else:

print(l,end=' ')

l+=1

else:

print(' ',end=' ')

print()

# assignment

E 2 D 4 C

E 2 D 4

E 2 D

E 2

E

n = 5

for i in range(1, n+1):

asciiVal = ord('E')

for j in range(1, n+1):

if i+j <= n+1:

if j % 2 == 1:

print(chr(asciiVal), end=' ')

asciiVal -= 1

else:

print(j, end=' ')

else:

print(' ', end=' ')

print()

\*

\* \*

\* \* \*

\* \*

\*

n = 5

for i in range(1,n+1):

for j in range(1,n+1):

if i>=j and i+j<=n+1:

print('\*',end=' ')

else:

print(' ',end=' ')

print()

\* \*

\* \* \* \*

\* \* \* \* \*

\* \* \* \*

\* \*

n = 5

for i in range(1,n+1):

for j in range(1,n+1):

if i>=j and i+j<=n+1 or i<=j and i+j>=n+1:

print('\*',end=' ')

else:

print(' ',end=' ')

print()

\* \* \* \* \*

\* \* \*

\*

\* \* \*

\* \* \* \* \*

n = 5

for i in range(1,n+1):

for j in range(1,n+1):

if i<=j and i+j<=n+1 or i>=j and i+j>=n+1:

print('\*',end=' ')

else:

print(' ',end=' ')

print()

#Print a diamond pattern

\*

\* \* \*

\* \* \* \* \*

\* \* \* \* \* \* \*

row = 4

space= 3

star=1

for i in range(1,row+1):

for j in range(space):

print(' ', end=' ')

for k in range(star):

print('\*',end=' ')

space-=1

star+=2

print()

\*

\* \* \*

\* \* \* \* \*

\* \* \* \* \* \* \*

\* \* \* \* \*

\* \* \*

\*

row = 7

space = row//2

star=1

for i in range(1,row+1):

for j in range(space):

print(' ',end=' ')

for k in range(star):

print('\*',end=' ')

if i<=row//2:

space-=1

star+=2

else:

space+=1

star-=2

print()

\*

\* \*

\* \*

\* \*

\* \*

\* \*

\*

row = 7

space = row//2

star=1

for i in range(1,row+1):

for j in range(space):

print(' ',end=' ')

for k in range(star):

if k==0 or k==star-1:

print('\*',end=' ')

else:

print(' ',end=' ')

if i<=row//2:

space-=1

star+=2

else:

space+=1

star-=2

print()

# assignment

\*

\* A \*

\* A B A \*

\* A B C B A \*

\* A B A \*

\* A \*

\*

row = 7

space = row//2

star = 1

for i in range(1, row+1):

asciiVal = ord('A')

for j in range(space):

print(' ', end=' ')

for k in range(star):

if k == 0 or k == star-1:

print('\*', end=' ')

else:

print(chr(asciiVal), end=' ')

if k >= star//2:

asciiVal -= 1

else:

asciiVal += 1

if i <= row//2:

space -= 1

star += 2

else:

space += 1

star -= 2

print()

**Functions:**

* Functions is a set of instructions designed to perform a specific task
* It is also called as reusable programs
* Function helps in code reusability
* According to functions there are two types
* Built in functions
* User defined functions

***Built in functions***

These are predefined functions which are predefined in the software to perform a specific task

Ex: ‘hello’.count(‘l’) 🡪 2

***Userdefined functions***

These are the functions that are created by the users performed the particular tasks

def <functionName>(arguments):

-----------------

---Body-------

----------------

return data

**function name**: it is the name of the function

**arguments**: it is the input parameters to perform operation

**body:**  set of instructions to be performed by the function when ever it is called

**return:** it is used to return the data from the function back to the caller(function call)

***Note: In python always the user defined function must start with ‘def’ keyword***

**Return:** return statement which is used to return the value or the data from the function back to the caller

return data

or

return data1,data2

return statement can return single or multiple values

* Always the return statement of a function must written inside the function
* Return statement is not mandatory for each function
* \*\* when ever the return statement returns more than one value or data then it will be stored in the form of tuple

**How to call a function**

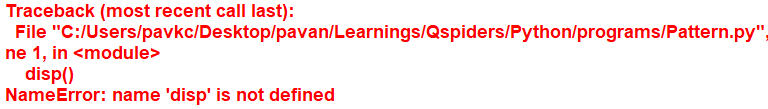
**functionName(arguments)**

**varName=functionName(arguments)**

**\*\*\* In python function definition should always be before the function call\*\*\* if in case the function is called before the function definition then compiler throws an error**

**XXXXXX wrong way to call a function**

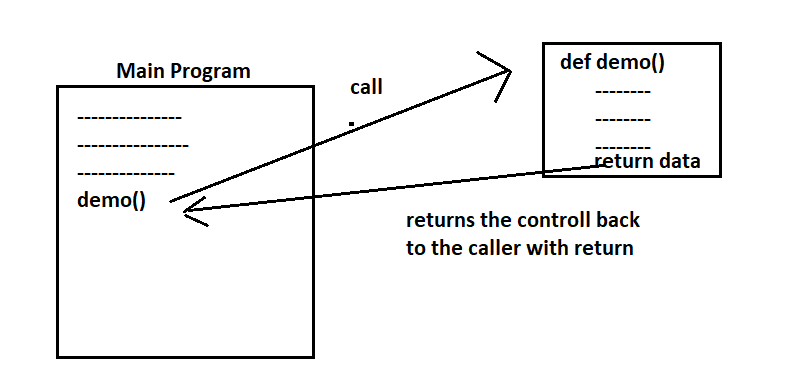






Output : inside disp function

***Program calling functions***



As long as the function is not called the function will not be executed, always the function definition must be return before the function call or the caller

The return statement to a function is not mandatory

Arguments to a function is also not mandatory

Function without return type: when it returns the control back to caller the caller will always hold the default generic value called as **‘None’**

***Types of user defined functions***

User defined functions are broadly classified into four types

* Function without argument and without return type
* Function without argument and with return type
* Function with argument and without return type
* Function with argument and with return type

**Function without argument and without return type**

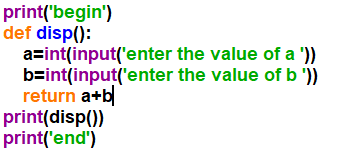
These kinds of functions are used whenever we need to perform some action without the user input and not returning back anything to the user

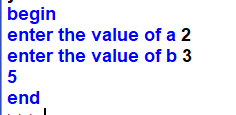


Output : inside disp function

**Function without argument and with return type**

These kinds of functions are used





Function with argument and without return type

These types of functions are used whenever we need input from the user to perform some operation in such case we use these functions

def add(a,b):

print(a+b)

add(3,2)

output: 5

**Functions with argument and with return type**

These functions are used whenever we need to take the input from the user to perform the operation inside the function and also to return the data back to the caller

def add(a,b):

print(‘add is invoked’)

return a+b

print(add(3,2))

output: add is invoked

5

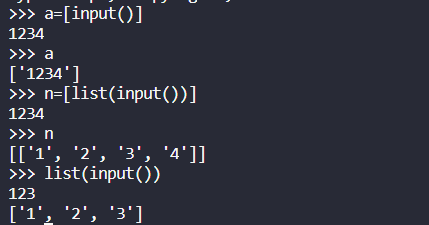
**Input:** it is a function which is used to take the input from the user,

The input taken from the user will be always in the form of string\*\*\*\*

In order to change the input value received then type casting must be done explicitly

Var=input(‘Instruction to user’)

Var=type(input(‘instruction to user’))



**Dynamic Insertion of data or elements in grouped datatypes like list, set or dictionary**

* **Dynamic insertion of elements into list**

a=[]

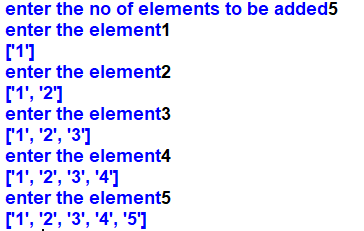
n = int(input('enter the no of elements to be added'))

for i in range(n):

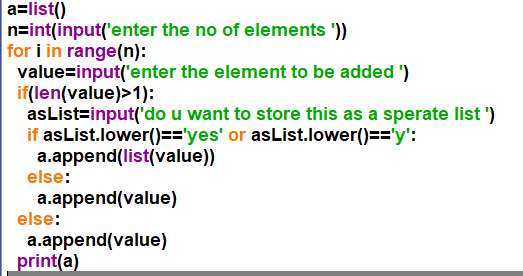
value=input('enter the element')

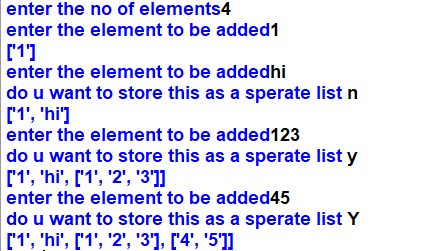
a.append(value)

print(a)

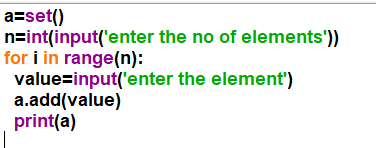


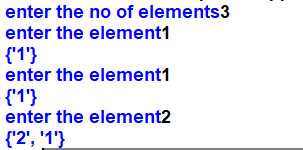
* Dynamic insertion of elements in to list with different grouped data types



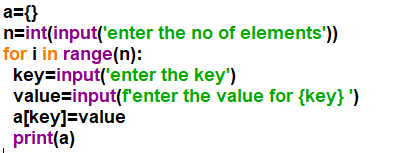


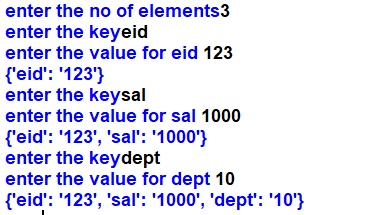
* **Dynamic insertion of elements into set**





* Dynamic insertion of elements into dict





# Assignment:

* a={‘eid’:1234,’Name’:{‘fname’:’ABC’,’lname’:’xyz’},’sal’:[100,200,300]}

a={}

n=int(input('enter the no of key value pairs to be added'))

for i in range(n):

m=int(input('enter the number of values to be added to a key'))

n1=input('enter the type of values to be added')

key=input('enter the outer dict key')

if n1=='list':

a1=[]

for j in range(m):

v=input('enter the value')

a1.append(v)

a[key]=a1

print(a)

elif n1=='dict':

a2={}

for k in range(m):

k1=input('enter the inner dict key')

v1=input('enter the inner dict value')

a2[k1]=v1

a[key]=a2

print(a)

else:

v2=input('enter the single value to be added')

a[key]=v2

print(a)

= RESTART: C:/Users/pavkc/AppData/Local/Programs/Python/Python37-32/pract.py =

enter the no of key value pairs to be added3

enter the number of values to be added to a key1

enter the type of values to be addedx

enter the outer dict keyid

enter the single value to be added1234

{'id': '1234'}

enter the number of values to be added to a key2

enter the type of values to be addeddict

enter the outer dict keyname

enter the inner dict keyfnm

enter the inner dict valueabc

enter the inner dict keylnm

enter the inner dict valuexyz

{'id': '1234', 'name': {'fnm': 'abc', 'lnm': 'xyz'}}

enter the number of values to be added to a key3

enter the type of values to be addedlist

enter the outer dict keysal

enter the value12000

enter the value1288

enter the value1500

{'id': '1234', 'name': {'fnm': 'abc', 'lnm': 'xyz'}, 'sal': ['12000', '1288', '1500']}

* {1,[1,2],’hi’,{1,2,3},{‘a’:’abc’,’d’:’def’}}

Write the program

* Start app():

Admin --- add student (name,phone,book id)

Delete student

Display

Exit

User—display

Exit

**Recursive function**

A function which calls itself until the given condition is satisfied is called recursive function

* Recursive function can be used to avoid loops in some cases
* Recursive function makes use of stack memory for its operation

def rec(n):

print(n,end=' ')

if n>1:

rec(n-1)

>>> rec(5)

5 4 3 2 1

\*\*\*\*\*\*\*\*\* analyse

def rec(n):

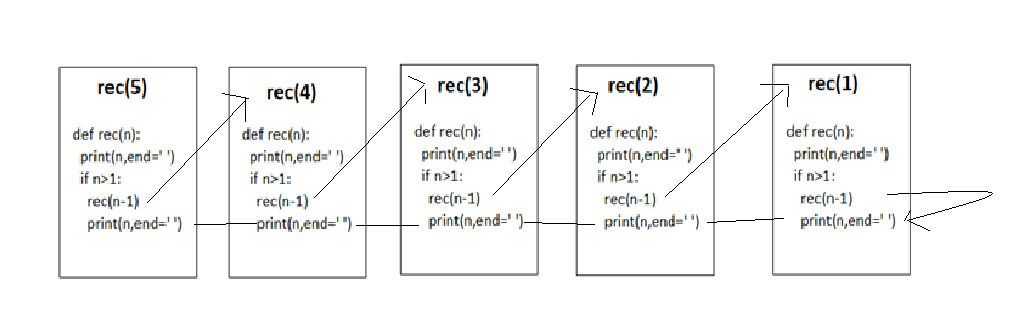
print(n,end=' ')

if n>1:

rec(n-1)

print(n,end=' ')

5 4 3 2 1 2 3 4 5



Recursive functions are basically used in searching and sorting algorithms and also in the implementation logic of data structures

Assignement:

* Factorial of a number using for loop and recursive function

def factorial(num):

factorial =1

if num==0:

print("factorial of 0 is 1")

else:

for i in range(1,num+1):

factorial = factorial\*i

print("factorial of ",num," is ",factorial)

* Fibonicci series using for loop and recursive function

Write a program to print below pattern

goodoog

def rec(i,n):

print(n[i],end=' ')

if i<len(n)-1:

rec(i+1,n)

print(n[i],end=' ')

>>> rec(0,'good')

g o o d o o g

**Import:** Import statement is used to import another file into current file

* It is basically used to get the content of another file or directory
* It is also used to add a module to another module

***Syntax***: Import Modulename/Package

Modules are nothing but the source file which exposes class functions and variables

Package: It is a collection of modules and it is also called as directories

Modules are classified into two types:

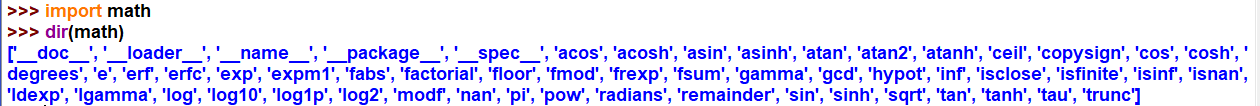
1. Built in modules
2. User defined modules

Built in modules are the modules which are pre defined within the software

User defined modules: these are the modules which are basically created by the users

Modules helps in code reusability

Built in Modules:



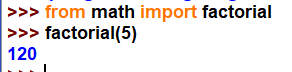
Suppose if the entire module is imported then the function is called as ***Modulename.functionName()***

Suppose if the specific function or the functionality of a module is imported then the function is called as ***functionName()***

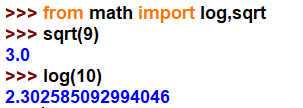
To import a specific function or the functionality we must use ***from*** keyword

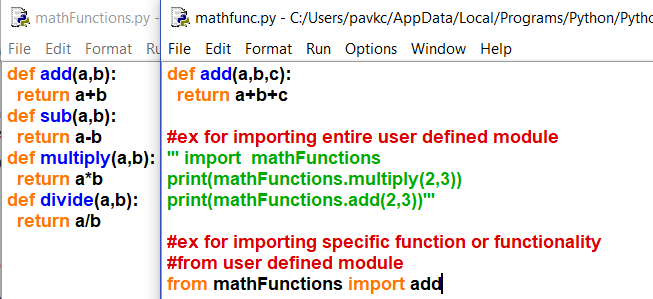
From keyword is used to import the specific function or the functionality from the module

Syntax to import single specific functionality or function: from <ModuleName> import <functionName>

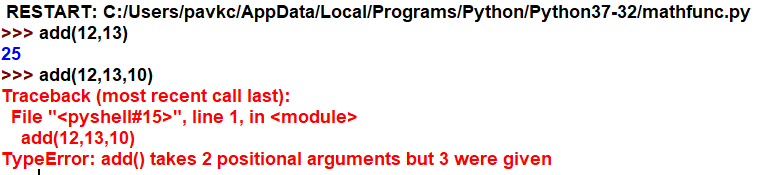


Example to import multiple specific functionality or functions





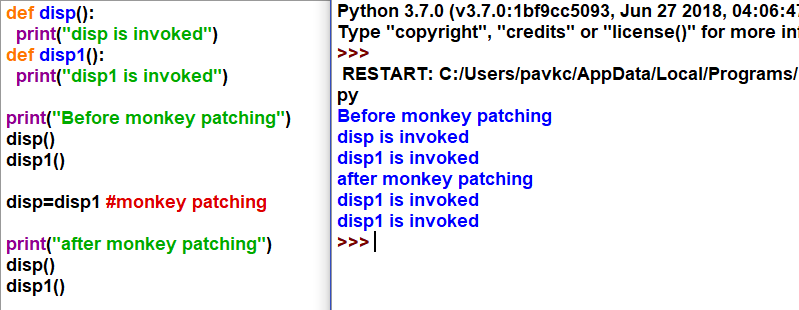
***Note***: Whenever multiple functions are written with the same name in the module or a file always when the function is called it in invokes the implementation of lastly returned function

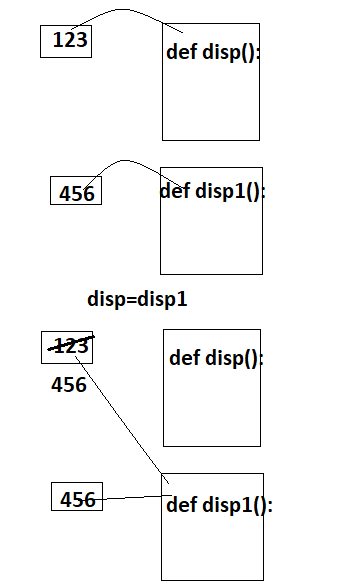


The above error is thrown only when specific function is imported

**Monkey Patching:**

It is a phenomenon of overriding the address of one function with another function is know as monkey pathching



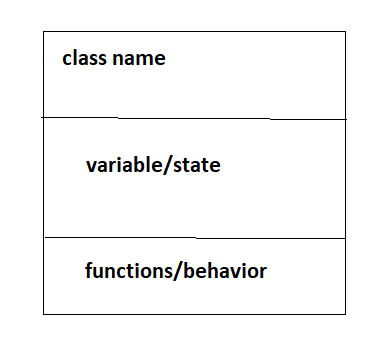
 Memory diagram

**Object Oriented Programing**

**Class:** class is an area which consists of state and behavior of a real world entity

* It is also called as user defined data types
* It is also called as blue print

The following representation is known as class diagram



Syntax to create a class:

Class <classname>:

---------------

---state-----

---------------

---Behavior—

----------------

Below is the code To create an empty class

Class A:

Pass

* Pass is a keyword which is used to represent the class as nothing

Class A:

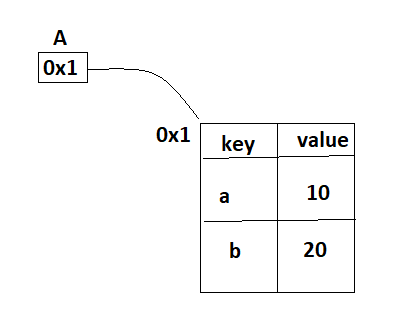
a=10

b=20

print(A.a,A.b)

>>> 10 20

***Note:*** A class internally creates a dictionary and everything will be stored in the form of key value pairs

****

**Object:** Object is a variable which is created using user defined data type called as class

* It is also called as instance of a class

Syntax to create an object: varName=className(arguments) -🡪 arguments—only if we speak about specific data

|  |  |  |
| --- | --- | --- |
| class A:  a=10  b=20  oa=A()  oa1=A()  print(A.a,A.b)  print(oa.a,oa.b)  print(oa1.a,oa1.b) | Output  10 20  10 20  10 20 | Copy diagram |

***Note:***For every object created automatically a dictionary will be created

1. Modification w.r.t to class

|  |  |  |
| --- | --- | --- |
| class A:  a=10  b=20  oa=A()  oa1=A()  print(A.a,A.b)  print(oa.a,oa.b)  print(oa1.a,oa1.b)  print('Modification w.r.t to class')  A.a=12  print(A.a,A.b)  print(oa.a,oa.b)  print(oa1.a,oa1.b) | Output  10 20  10 20  10 20  Modification w.r.t to class  12 20  12 20  12 20 | Copy diagram |

1. Modification w.r.t to oa

|  |  |  |
| --- | --- | --- |
| class A:  a=10  b=20  oa=A()  oa1=A()  print(A.a,A.b)  print(oa.a,oa.b)  print(oa1.a,oa1.b)  print('Modification w.r.t to class')  A.a=12  print(A.a,A.b)  print(oa.a,oa.b)  print(oa1.a,oa1.b)  print('Modification w.r.t to object oa')  oa.a=14  print(A.a,A.b)  print(oa.a,oa.b)  print(oa1.a,oa1.b) | Output  10 20  10 20  10 20  Modification w.r.t to class  12 20  12 20  12 20  Modification w.r.t to object oa  12 20  14 20  12 20 | Copy diagram |

1. Modification w.r.t to object oa1

|  |  |  |
| --- | --- | --- |
| class A:  a=10  b=20  oa=A()  oa1=A()  print(A.a,A.b)  print(oa.a,oa.b)  print(oa1.a,oa1.b)  print('Modification w.r.t to class')  A.a=12  print(A.a,A.b)  print(oa.a,oa.b)  print(oa1.a,oa1.b)  print('Modification w.r.t to object oa')  oa.a=14  print(A.a,A.b)  print(oa.a,oa.b)  print(oa1.a,oa1.b)  print('Modification w.r.t to object oa1')  oa1.a=16  print(A.a,A.b)  print(oa.a,oa.b)  print(oa1.a,oa1.b) | Output  10 20  10 20  10 20  Modification w.r.t to class  12 20  12 20  12 20  Modification w.r.t to object oa  12 20  14 20  12 20  Modification w.r.t to object oa1  12 20  14 20  16 20 | Copy diagram |

1. Further modification w.r.t to class

|  |  |  |
| --- | --- | --- |
| class A:  a=10  b=20  oa=A()  oa1=A()  print(A.a,A.b)  print(oa.a,oa.b)  print(oa1.a,oa1.b)  print('Modification w.r.t to class')  A.a=12  print(A.a,A.b)  print(oa.a,oa.b)  print(oa1.a,oa1.b)  print('Modification w.r.t to object oa')  oa.a=14  print(A.a,A.b)  print(oa.a,oa.b)  print(oa1.a,oa1.b)  print('Modification w.r.t to object oa1')  oa1.a=16  print(A.a,A.b)  print(oa.a,oa.b)  print(oa1.a,oa1.b)  print('Further Modification w.r.t to class')  A.a=22  print(A.a,A.b)  print(oa.a,oa.b)  print(oa1.a,oa1.b) | Output  10 20  10 20  10 20  Modification w.r.t to class  12 20  12 20  12 20  Modification w.r.t to object oa  12 20  14 20  12 20  Modification w.r.t to object oa1  12 20  14 20  16 20  Further Modification w.r.t to class  22 20  14 20  16 20 | Copy diagram |

***Note:*** any modification with respect to class or class name changes get affected in itself and all its decedents(object)

Modification with respect to object changes get affected only in the object in which modification is done

All the data which is related to class or object will be stored in the form of dictionary, where the dictionary keys are nothing but name of the state or the name of the function and value pair is nothing but the data or the address of the behavior for function which is stored in the method area, basically the data are of two types.

1. Generic data
2. Specific data

**Generic data:** It is a data which is general or common for all the objects.

We call these kind of data as class data and the variable which holds class data are called as class variables

These class variables are defined inside a class or store inside the class

**Specific data:** It is a data which is specific for each and every individual object,

As the data is specific we don’t store it inside a class, we store it only inside object

As this data is stored only inside object we call it as object members

These object members have been intialized inside the special method with the name \_\_init\_\_

This method must take atleast 1 argument which holds the address of the object in which we wanted to insert the data

These kinds of methods is called as constructor or initialization method in other programming languages

Constructors will be invoked at the time of object creation by the application itself

We don’t have to call this functions externally, that one argument inside the special method are constructor which holds the address of an object is named as ‘self’ as per the industry standards

But as programmer we can give any name for the argument, but when ever we are working with team of people or in real time environment we must follow industrial standards

Syntax: def \_\_init\_\_(self,arguments):

Self.arguments=value

**Constructor without any argument:**

class A:

a=10

b=20

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

print('inside INIT constructor')

print('address in self',self)

oa=A()

print(oa)

inside INIT constructor

address in self <\_\_main\_\_.A object at 0x056872B0>

<\_\_main\_\_.A object at 0x056872B0>

**Calling a constructor:**

1. Calling constructor using object name

objectName.\_\_init\_\_(args)

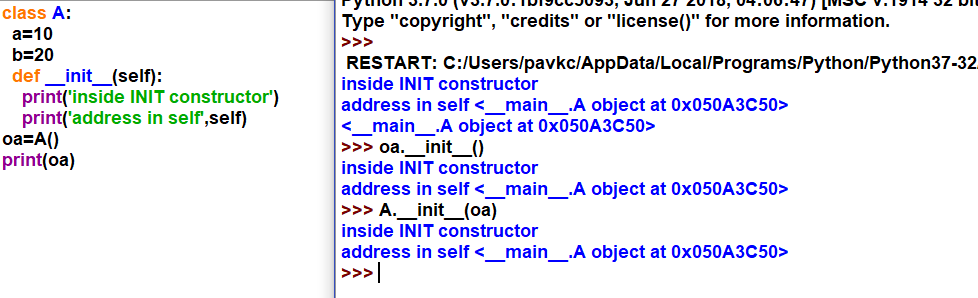
1. Calling constructor using class name

className.\_\_init\_\_(self,args)

Note: If the constructor is not defined in a class then by default the application itself adds the following two lines of statement

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

pass



Create a customer class with the class members as bankname IFSC code and manager name

Create an object and object members as customer name and phone numbers

class Customer:

BankName='abc'

IFSCCode='cde'

ManagerName='xyz'

def \_\_init\_\_(self,cusName,phNum):

self.cusName=cusName

self.phNum=phNum

oa=Customer('pavan',1234)

print(Customer.BankName, Customer.IFSCCode,Customer.ManagerName)

print(oa.BankName,oa.IFSCCode,oa.ManagerName,oa.cusName,oa.phNum)

abc cde xyz

abc cde xyz pavan 1234

* Write the dictionary diagram

class Customer:

BankName='abc'

IFSCCode='cde'

ManagerName='xyz'

def \_\_init\_\_(self,cusName,age,phNum,email):

self.cusName=cusName

self.phNum=phNum

self.age=age

self.email=email

oa=Customer('james',18,123456,'a@a.com')

oa1=Customer('bond',20,123456,'b@b.com')

oa2=Customer('alex',20,123456,'c@c.com')

print(Customer.BankName, Customer.IFSCCode,Customer.ManagerName)

print(oa.BankName,oa.IFSCCode,oa.ManagerName,oa.cusName,oa.age,oa.phNum,oa.email)

print(oa1.BankName,oa1.IFSCCode,oa1.ManagerName,oa1.cusName,oa1.age,oa1.phNum,oa1.email)

print(oa2.BankName,oa2.IFSCCode,oa2.ManagerName,oa2.cusName,oa2.age,oa2.phNum,oa2.email)

abc cde xyz

abc cde xyz james 18 123456 a@a.com

abc cde xyz bond 20 123456 b@b.com

abc cde xyz alex 20 123456 [c@c.com](mailto:c@c.com)

* Write the dict diagram

There are three types of methods which can be defined inside a class

1. Object method
2. Class method
3. Static method

Object methods are used to access and modify object members

These types of methods will only deal with OR works with object data for all its operation

def objectMethodName(self,arguments):

-------------------------

-------------------------

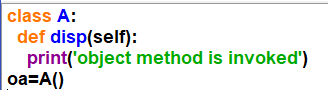
-------------------------

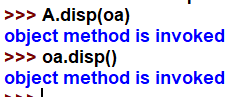
Object methods can be accessed either by using the class name or by using object name

Syntax to call the object method using the class name: className.objectMethodName(self,arg)

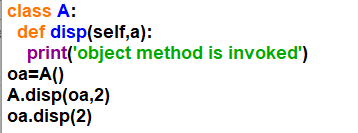
Syntax to call the object method using the object name: objectName.ObjectMethodName(arg)

# Example for the object method without argument





# Example for object method with argument





class Customer:

BankName='abc'

IFSCCode='cde'

ManagerName='xyz'

def \_\_init\_\_(self,cusName,age,phNum,email):

self.cusName=cusName

self.phNum=phNum

self.age=age

self.email=email

def disp(self):

print('Bankname is ',self.BankName)

print('IFSCCode is ',self.IFSCCode)

print('Bankname is ',self.ManagerName)

print('ManagerName is ',self.cusName)

print('AGE is ',self.age)

print('PhNum is ',self.phNum)

print('Email is ',self.email)

oa=Customer('james',18,123456,'a@a.com')

oa1=Customer('bond',20,123456,'b@b.com')

oa2=Customer('alex',20,123456,'c@c.com')

oa.disp()

oa1.disp()

oa2.disp()

Bankname is abc

IFSCCode is cde

Bankname is xyz

ManagerName is james

AGE is 18

PhNum is 123456

Email is a@a.com

Bankname is abc

IFSCCode is cde

Bankname is xyz

ManagerName is bond

AGE is 20

PhNum is 123456

Email is b@b.com

Bankname is abc

IFSCCode is cde

Bankname is xyz

ManagerName is alex

AGE is 20

PhNum is 123456

Email is c@c.com

# in the above program write a modify method which modifies the object members based on the user input

class Customer:

BankName='abc'

IFSCCode='cde'

ManagerName='xyz'

def \_\_init\_\_(self,cusName,age,phNum,email):

self.cusName=cusName

self.phNum=phNum

self.age=age

self.email=email

def disp(self):

print('Bankname is ',self.BankName)

print('IFSCCode is ',self.IFSCCode)

print('Bankname is ',self.ManagerName)

print('ManagerName is ',self.cusName)

print('AGE is ',self.age)

print('PhNum is ',self.phNum)

print('Email is ',self.email)

def modify(self):

dataToModify=input("enter the field name to be modified")

if dataToModify=="cusName":

self.cusName=input("Please enter customer name")

elif dataToModify=="age":

self.age=int(input("enter your age"))

elif dataToModify=="phNum":

self.phNum=int(input("enter your phone number"))

elif dataToModify=="email":

self.email=input("enter email")

else:

self.cusName=input("Please enter customer name")

self.age=int(input("enter your age"))

self.phNum=int(input("enter your phone number"))

self.email=input("enter email")

oa=Customer('james',18,123456,'a@a.com')

oa.disp()

oa.modify()

print("----Data after modification-----")

oa.disp()

class Customer:

BankName='abc'

IFSCCode='cde'

ManagerName='xyz'

def \_\_init\_\_(self,cusName,age,phNum,email,atm=False):

self.cusName=cusName

self.phNum=phNum

self.age=age

self.email=email

self.atm=atm

def disp(self):

print('Bankname is ',self.BankName)

print('IFSCCode is ',self.IFSCCode)

print('Bankname is ',self.ManagerName)

print('ManagerName is ',self.cusName)

print('AGE is ',self.age)

print('PhNum is ',self.phNum)

print('Email is ',self.email)

def atmVerifcation(self):

if self.atm==False:

self.atm=True

print("need to issue a fresh atm card")

else:

print("atm card is already issued")

oa=Customer('james',18,123456,'a@a.com')

print("------------------------")

oa1=Customer('alex',12,124345,'b@b.com',True)

oa.disp()

oa.atmVerifcation()

oa1.disp()

oa1.atmVerifcation()

**class Method()**

class method which is used to access and modify the class members

class methods must always start with, “***@classmethod***” keyword

syntax:

@classmethod

def classmethodname(cls,<arguments>)

------------------------------

-------Body----------------

------------------------------

Class methods can be accessed either by using the class name or by using the object name

Syntax to access the class method using the class name

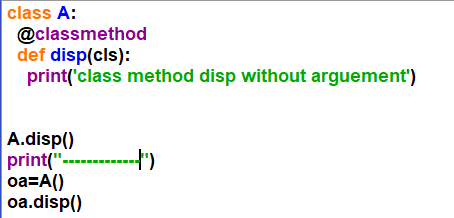
*Class method without argument:* **className.methodName()**

*Class method with argument:* **className.methodName(args)**

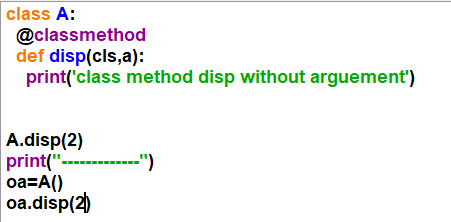
Syntax to access class method using the object name

*Class method without the argument:* **objectName.classMethodName()**

*Class method without the argument:* **objectName.classMethodName(args)**



*Note:* cls🡪 in the class method cls is a minimum mandatory argument, every class method must be defined in order to hold the address of the class in which the modification has to be done irrespective of the input parameters that is arguments



class Customer:

BankName='abc'

IFSCCode='cde'

ManagerName='xyz'

def \_\_init\_\_(self,cusName,age,phNum,email):

self.cusName=cusName

self.phNum=phNum

self.age=age

self.email=email

def disp(self):

print('Bankname is ',self.BankName)

print('IFSCCode is ',self.IFSCCode)

print('Bankname is ',self.ManagerName)

print('ManagerName is ',self.cusName)

print('AGE is ',self.age)

print('PhNum is ',self.phNum)

print('Email is ',self.email)

@classmethod

def modifyClassMethod(cls):

dataToModify=input("enter the field name to be modified")

if dataToModify=="BankName":

cls.BankName=input("Please enter new bank name")

elif dataToModify=="IFSCCode":

cls.IFSCCode=int(input("enter new IFSC Code"))

elif dataToModify=="ManagerName":

cls.ManagerName=input("enter new Manager number")

else:

cls.BankName=input("Please enter new bank name")

cls.IFSCCode=int(input("enter new IFSC Code"))

cls.ManagerName=input("enter new Manager number")

oa=Customer('james',18,123456,'a@a.com')

oa.disp()

print("Data belore modification")

Customer.modifyClassMethod()

print("----Data after modification in class method-----")

oa.disp()

**Static method():**

Static method neither deals with class member nor with object members, where the static methods can be used to perform some common operation

* As it performs common operation it is also called as generalized method
* Static method always starts with “@staticmethod” keyword

Syntax:

@staticmethod

def staticMethodName(arg):

-----------------------------

----------Body------------

------------------------------

Static method can be accessed either by using the class name or by using the object name

*Syntax to call the static method using class name:*

className.methodName()

className.methodName(args)

*Syntax to call the static method using object name:*

ObjectName.methodName()

objectName.methodName(args)

**static without argument**

@staticmethod

Def disp():

Print(“static method disp invoked”)

A.disp()

Oa.disp()

Static with argument

@staticmethod

Def disp(a):

Print(“static method disp invoked”)

A.disp(2)

Oa.disp(2)

# write an example which can access object member and class member static method

class Customer:

BankName='abc'

IFSCCode='cde'

ManagerName='xyz'

def \_\_init\_\_(self,cusName,age,phNum,email,atm=False,sal):

self.cusName=cusName

self.phNum=phNum

self.age=age

self.email=email

self.atm=atm

self.sal=sal

@staticmethod

def smethod(self,cls):

basic=self.sal-2000

print("the basic salary is ",basic)

print("the bank name is ",cls.Bname)

def disp(self):

print('Bankname is ',self.BankName)

print('IFSCCode is ',self.IFSCCode)

print('Bankname is ',self.ManagerName)

print('ManagerName is ',self.cusName)

print('AGE is ',self.age)

print('PhNum is ',self.phNum)

print('Email is ',self.email)

def atmVerifcation(self):

if self.atm==False:

self.atm=True

print("need to issue a fresh atm card")

else:

print("atm card is already issued")

oa=Customer('james',18,123456,'a@a.com',True,12000)

print("------------------------")

oa.disp()

Customer.smethod(oa,Customer)

# example for class method static method object method in a single class

**Inheritance :**

It is an object oriented programing concept in which the properties of one class is used in another class

Inheritance is broadly classified into 5 types

* Single Inheritance
* Multiple Inheritance
* Multilevel Inheritance
* Hierarchical inheritance
* Hybrid inheritance

Base class/ Parent class /super class is the class from which properties have been adopted

Derived class/child class/sub class is the class which has the properties of another class

* Single Inheritance

In single inheritance the derived class has the properties of single parent class

OR

When a class is derived from single parent class then it is called as single inheritance

Syntax:

Class Parent:

---------------------

------Body--------

---------------------

Class Child(Parent):

---------------------

------Body--------

---------------------

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Step1: Single inheritance is created  A   |  |  | | --- | --- | | K | V | | a | 10 | | b | 20 | |  |
| B(A)   |  |  | | --- | --- | | K | V | | a | a1 | | b | a2 | | c | 30 | |  |

Program for single inheritance:

class A:

a=10

b=20

class B(A):

c=30

print(A.a,A.b)

print(B.a,B.b,B.c)

print("modification w.r.t parent")

A.a=12

print(A.a,A.b)

print(B.a,B.b,B.c)

print("modification w.r.t to child")

B.a=14

print(A.a,A.b)

print(B.a,B.b,B.c)

print("further modification of same member in parent class")

A.a=16

print(A.a,A.b)

print(B.a,B.b,B.c)

### Output

10 20

10 20 30

modification w.r.t parent

12 20

12 20 30

modification w.r.t to child

12 20

14 20 30

further modification of same member in parent class

16 20

14 20 30

* In the child class first we store the derived members followed by the non derived members
* Child class members(Non derived or not accessible for the parent class)
* When the modification is done with respect to parent class the changes get affected in itself and also in the child class and its object
* When the modification is done with respect to child class the parent class remains unaffected
* Here the changes get affected only in the child class and its objects

Step1: Single inheritance is created

Step2: when the object of child class is created

ob=B()

Step3:When the modification is done with respect to parent class

* Here the changes also get affected in child class and its objects

Step4: When the modification is done with respect to child class

* Here the changes get affected only the child class and its objects where the parent class remains unaffected

Step5: When the modification is done with respect to child class object

The changes get affected only in the object, here the parent class and child class remains unaffected

# Write the program for the above dict diagram

class A:

a=10

b=20

class B(A):

c=30

print("single inheritance is created")

print(A.a,A.b)

print(B.a,B.b,B.c)

print("When the objects is created")

ob=B()

print(A.a,A.b)

print(B.a,B.b,B.c)

print(ob.a,ob.b,ob.c)

print("modification w.r.t parent")

A.a=12

print(A.a,A.b)

print(B.a,B.b,B.c)

print(ob.a,ob.b,ob.c)

print("modification w.r.t to child")

B.a=14

print(A.a,A.b)

print(B.a,B.b,B.c)

print(ob.a,ob.b,ob.c)

print("modification is done with respect to child class object")

ob.a=16

print(A.a,A.b)

print(B.a,B.b,B.c)

print(ob.a,ob.b,ob.c)

print("further modification is done with respect to parent class and child class")

A.a=40

B.a=50

print(A.a,A.b)

print(B.a,B.b,B.c)

print(ob.a,ob.b,ob.c)

#### output

RESTART: C:/Users/pavkc/AppData/Local/Programs/Python/Python37-32/practice.py

single inheritance is created

10 20

10 20 30

When the objects is created

10 20

10 20 30

10 20 30

modification w.r.t parent

12 20

12 20 30

12 20 30

modification w.r.t to child

12 20

14 20 30

14 20 30

modification is done with respect to child class object

12 20

14 20 30

16 20 30

further modification is done with respect to parent class and child class

40 20

50 20 30

16 20 30

When the constructor is defined only in the parent class

\*\*\*\*\*\*copy the dict diagram

class A:

a=10

b=20

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

print("init of parent class is invoked")

class B(A):

c=30

ob=B()

#output

RESTART: C:/Users/pavkc/AppData/Local/Programs/Python/Python37-32/practice.py

init of parent class is invoked

When the constructor is defined only in the parent class it is inherited in the child class as well, then the object of child class refers to parent class constructor

When the \_\_init\_\_ is defined in both parent class and child class, then the child class constructor is invoked (dict cannot have duplicate key value pairs)

\*\*\*\*\* copy the dict diagram

class A:

a=10

b=20

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

print("init of parent class is invoked")

class B(A):

c=30

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

print("init of child class is invoked")

ob=B()

Output:

RESTART: C:/Users/pavkc/AppData/Local/Programs/Python/Python37-32/practice.py

init of child class is invoked

When the \_\_init\_\_ is defined in both parent class and child class, then the object of child class will refer child class version of the constructor

It is possible to call parent class constructor from the child class(inside the child class constructor)

* The object of child class can call both parent class and child class constructor provided the child class must contain, the class to its parent class constructor within it

**Constructor chaining:**

It is the process of calling the parent class constructor inside child class

Example: without argument

\*\*\*\*\*\*\* copy the same dict diagram from above

class A:

a=10

b=20

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

print("init of parent class is invoked")

class B(A):

c=30

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

print("init of child class is invoked")

#A.\_\_init\_\_(self)

super().\_\_init\_\_()

ob=B()

RESTART: C:/Users/pavkc/AppData/Local/Programs/Python/Python37-32/practice.py

init of child class is invoked

init of parent class is invoked

We can call the parent class constructor either by using the class name or by using super keyword or method(check the above program)

When the super keyword is used to invoke parent class constructor, we need not to pass ‘self’ as an argument

Super() key word by default call its immediate parent

Syntax: **super().\_\_init\_\_(args)**

Syntax to call the parent class constructor inside the child class : **classname.\_\_init\_\_(self,arg)**

**Construct chaining with argument:**

class A:

a=10

b=20

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

print("init of parent class is invoked",b)

class B(A):

c=30

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

print("init of child class is invoked",a)

#A.\_\_init\_\_(self,a)

super().\_\_init\_\_(a+2)

ob=B(2)

RESTART: C:/Users/pavkc/AppData/Local/Programs/Python/Python37-32/practice.py

init of child class is invoked 2

init of parent class is invoked 4

**\*\*\*\*When the object method is defined only in the parent class**

class A:

def objectMethod(self):

print('parent class object method')

class B(A):

pass

ob=B()

B.objectMethod(ob)

ob.objectMethod()

RESTART: C:/Users/pavkc/Desktop/pavan/Learnings/Qspiders/Python/programs/classtests.py

Parent class object method

Parent class object method

When the object method is defined only in the parent class, then the child class and its objects will refer to the object method which is defined within the parent class

**\*\*\*\* When the object method is defined in both parent class and child class**

class A:

def objectMethod(self):

print('parent class object method')

class B(A):

def objectMethod(self):

print('child class object method')

ob=B()

B.objectMethod(ob)

ob.objectMethod()

RESTART: C:/Users/pavkc/Desktop/pavan/Learnings/Qspiders/Python/programs/classtests.py

child class object method

child class object method

*Note:* When the object method is defined in both parent class and child class, then object of child class and the child class will refer to the object method which is defined within the child class

\*\* static method defined only in the parent class

class A:

@staticmethod

def staticMethod():

print("static method of parent class invoked")

class B(A):

pass

ob=B()

B.staticMethod()

ob.staticMethod()

RESTART: C:/Users/pavkc/Desktop/pavan/Learnings/Qspiders/Python/programs/classtests.py

static method of parent class invoked

static method of parent class invoked

\*\*static method defined both in parent class and child class

class A:

@staticmethod

def staticMethod():

print("static method of parent class invoked")

class B(A):

@staticmethod

def staticMethod():

print("static method of child class invoked")

ob=B()

B.staticMethod()

ob.staticMethod()

RESTART: C:/Users/pavkc/Desktop/pavan/Learnings/Qspiders/Python/programs/classtests.py

static method of child class invoked

static method of child class invoked

\*\*\* class method defined only in the parent class

class A:

@classmethod

def classMethod(cls):

print("class method of parent class invoked")

class B(A):

pass

ob=B()

B.classMethod()

ob.classMethod()

class method of parent class invoked

class method of parent class invoked

\*\*\*class method defined in both parent class and child class

class A:

@classmethod

def classMethod(cls):

print("class method of parent class invoked")

class B(A):

@classmethod

def classMethod(cls):

print("class method of child class is invoked")

ob=B()

B.classMethod()

ob.classMethod()

RESTART: C:/Users/pavkc/Desktop/pavan/Learnings/Qspiders/Python/programs/classtests.py

class method of child class is invoked

class method of child class is invoked

**Multilevel Inheritance**

It is a type of inheritance in which a new class is created using another derived class. I.e the inheritance that takes place at different level

**A()** 🡪 Parent class

**B(A)** 🡪 child of A

**C(B)** 🡪 subchild/Child of B

In muiltilevel inheritance the derived member are inherited first , When the derived members are inherited, the child (sub child) will refer to its immediate parent

When the modification is done at the parent class all the child classes will get affected in the hierarchy

When the modification is done at immediate parent the changes get affected in itself and its child class i.e. subchild class here the parent class remains unaffected

When the modification is done at sub child class the changes get affected only in itself and its objects. Here parent and immediate parents remains unaffected

**Step1:** Multilevel Inheritance is created

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **K** | **V** | | **a** | **10** | | |  |  | | --- | --- | | **K** | **V** | | **a** | **a1** | | **b** | **20** | | |  |  | | --- | --- | | **K** | **V** | | **a** | **b1** | | **b** | **b2** | | **c** | **30** | |

**Step 2:** When the modification at parent class

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **K** | **V** | | **a** | **10 12** | | |  |  | | --- | --- | | **K** | **V** | | **a** | **a1** | | **b** | **20** | | |  |  | | --- | --- | | **K** | **V** | | **a** | **b1** | | **b** | **b2** | | **c** | **30** | |

**Step3 :** When the modification is done at immediate parent

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **K** | **V** | | **a** | **10 12** | | |  |  | | --- | --- | | **K** | **V** | | **a** | **a1 14** | | **b** | **20** | | |  |  | | --- | --- | | **K** | **V** | | **a** | **b1** | | **b** | **b2** | | **c** | **30** | |

**Step4:** When the modification is done at subchild

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **K** | **V** | | **a** | **10 12** | | |  |  | | --- | --- | | **K** | **V** | | **a** | **a1 14** | | **b** | **20** | | |  |  | | --- | --- | | **K** | **V** | | **a** | **b1 16** | | **b** | **b2** | | **c** | **30** | |

class A:

a=10

class B(A):

b=20

class C(B):

c=30

print(A.a)

print(B.a,B.b)

print(C.a,C.b,C.c)

print('modification wrt to parent')

A.a=12

print(A.a)

print(B.a,B.b)

print(C.a,C.b,C.c)

print('modification wrt to child')

B.a=14

print(A.a)

print(B.a,B.b)

print(C.a,C.b,C.c)

print('modification wrt to subchild')

B.a=16

print(A.a)

print(B.a,B.b)

print(C.a,C.b,C.c)

RESTART: C:/Users/pavkc/Desktop/pavan/Learnings/Qspiders/Python/programs/classtests.py

10

10 20

10 20 30

modification wrt to parent

12

12 20

12 20 30

modification wrt to child

12

14 20

14 20 30

modification wrt to subchild

12

16 20

16 20 30

***Constructor in multilevel inheritance***

When the constructor is defined only in the parent class, the immediate parent(child) and subchild class also refers to parent class constructor

The object created for any of the three class in the hierarchy will refer to same parent class constructor

class A:

a=10

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

print("parent class init is invoked")

class B(A):

b=20

class C(B):

c=30

oa=A()

ob=B()

oc=C()

*RESTART: C:/Users/pavkc/Desktop/pavan/Learnings/Qspiders/Python/programs/classtests.py*

*parent class init is invoked*

*parent class init is invoked*

*parent class init is invoked*

Write the dict diagram

|  |  |
| --- | --- |
|  |  |

When the constructor is defined in both parent and immediate parent class🡪 here the object of parent class will refer the parent class constructor

The object of immediate parent and the subchild class refers to the constructor which is defined in immediate parent class

class A:

a=10

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

print("parent class constructor is invoked")

class B(A):

b=20

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

print("immidiate parent class constructor is invoked")

class C(B):

c=30

oa=A()

ob=B()

oc=C()

*RESTART: C:/Users/pavkc/Desktop/pavan/Learnings/Qspiders/Python/programs/classtests.py*

*parent class constructor is invoked*

*immediate parent class constructor is invoked*

*immediate parent class constructor is invoked*

class A:

a=10

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

print("parent class constructor is invoked")

class B(A):

b=20

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

print("immidiate parent class constructor is invoked")

class C(B):

c=30

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

print("subchild class constructor is invoked")

oa=A()

ob=B()

oc=C()

#write a program for constructor chaining in multilevel inheritance

class A:

a=10

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

print("init of parent class is invoked")

class B(A):

b=20

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

print("init of child class is invoked")

super().\_\_init\_\_()

class C(B):

c=30

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

print("init of subchildclass")

super().\_\_init\_\_()

ob=C()

RESTART: C:/Users/pavkc/Desktop/pavan/Learnings/Qspiders/Python/programs/classtests.py

init of subchildclass

init of child class is invoked

init of parent class is invoked

#constructor chaining with parameters

class A:

a=10

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

print("init of parent class is invoked")

class B(A):

b=20

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

print("init of child class is invoked")

super().\_\_init\_\_()

class C(B):

c=30

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

print("init of subchildclass")

super().\_\_init\_\_("a")

ob=C()

*RESTART: C:/Users/pavkc/Desktop/pavan/Learnings/Qspiders/Python/programs/classtests.py*

*init of subchildclass*

*init of child class is invoked*

*init of parent class is invoked*

class A:

a=10

def \_\_init\_\_(self,a,b):

print("init of parent class is invoked")

print("value of a ",a,"value of b",b)

class B(A):

b=20

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

print("init of child class is invoked")

print("value of a ",a)

super().\_\_init\_\_(2,2)

class C(B):

c=30

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

print("init of subchildclass")

super().\_\_init\_\_("hello")

ob=C()

*RESTART: C:/Users/pavkc/Desktop/pavan/Learnings/Qspiders/Python/programs/classtests.py*

*init of subchildclass*

*init of child class is invoked*

*value of a hello*

*init of parent class is invoked*

*value of a 2 value of b 2*

class A:

a=10

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

self.x=x

print("init of parent class is invoked")

class B(A):

b=20

def \_\_init\_\_(self,x,y):

self.y=y

print("init of child class is invoked")

super().\_\_init\_\_(x)

class C(B):

c=30

def \_\_init\_\_(self,x,y,z):

self.z=z

print("init of subchildclass")

super().\_\_init\_\_(x,y)

ob=C(1,2,3)

**Invoking the base class methods using the child class object reference even after overriding the method in the child class can be achieved**

1. By creating has-a relationship
2. By using super keyword or method

By Creating has-a relationship

class A:

a=10

def disp(self):

print('disp in A')

class B:

b=20

oa=A()

def disp(self):

print('disp in B')

ob=B()

#ob.disp()

**ob.oa.disp()**

RESTART: C:/Users/pavkc/Desktop/pavan/Learnings/Qspiders/Python/programs/classtests.py

disp in A

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Key | Value | | a | 10 | | disp | Address of disp in A | | |  |  | | --- | --- | | Key | Value | | a | a1 | | disp | a2  Address of disp in B | |
|  |  |

**Using super keyword or method:**

Syntax: super(childClassName,ChildClassObjectName).Methodname()

Where childClassName 🡪 class name whose parent member to be accessed

childClassObjectName🡪 Object reference of child class

class A:

a=10

def disp(self):

print('disp in A')

class B(A):

b=20

def disp(self):

print('disp in B')

ob=B()

super(B,ob).disp()

RESTART: C:/Users/pavkc/Desktop/pavan/Learnings/Qspiders/Python/programs/classtests.py

disp in A

###Invoking the base class methods using the child class objects even after overriding in the child class (multilevel inheritance)

Copy class notes