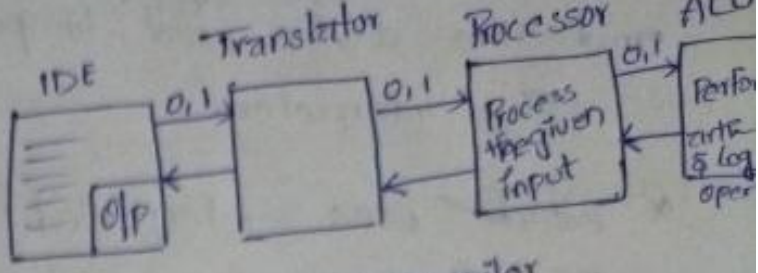
Importance of Python:

1. Easy to learn and analyze
2. Dynamically Typed
3. Intercepted Language
4. thigh level Language
5. less members of instructions
6. job is secured
7. platform independent
8. Open source Language
9. Scripting Language

Process OF execution:



Translator → compiler
Translator → Interpreter

IDE:

IDE (Integrated Development Environment) is used to write a source code any type OF Programming Langu

Translator:

Translator is a device that is used to convert the program code to binary language (0,1)

Translators are two types.

1. compiler:

The process of converting Entire code in the form of binary instruction at once is called as compalation and the use to do compile is called compiler.

2. Interpreter:

The process of converting line-by-line in the form of binary language is called as interpretation and the device used to perform interpretation is called as interpreter.

* Which uses interpreted for the execution then the language is called scripted language.

Python Installation:

1. Download:

www.python.org
click on downloads
select the dedicated sflow
search for the version 3.7.4 x86-64 Executable

2. Installation:

click on the dowlozal file
☑ add python 3.7 to path
Install now

Note:

help() is a function which is used to get the content related to the in-built function

Introduction to library function:

→ Library function are the pre-defined function

→ These are the functions whose task is pre-define by the developer, when ever we want to access library function in our program it is Easy to access, but we can't modify the original task of library function

→ Library function are classified into three types

1. Keywords

   Eg: def, if

2. Operators

   Eg: +, -, %.

3. In-built function

   Eg: print(), len()

Keywords:

→ Keywords are the universal standard words whose task is pre-defined by the developer

→ When ever we want to use keywords in our program it is Easy to use but it is not possible to modify the original meaning of keywords

→ There are 33 Keywords in 3.5+ version,
and 35 keywords in 3.7+ version,
(and 36 keywords in 3.9+ version)

→ It is not possible to remember all the 35 keywords so that we are going to use a syntax to get all the keywords on the screen

Syntax:

```
import keyword
keyword.kwlist
```

Keywords:

False None True and as
assert async await break class
continue def del Elif else
Except finally for from global
if import in is lambda
nonlocal not or yield pass
raise return try while with

→ There are three diff (or) special keywords are the diff from other keywords, those are

False, None, true

→ False, None, True are the keywords which can be assign as a value while creating var

Eg:

```
a = True
True
a= None

a
a = False
False
```

→ Keywords will get displayed in Orange color

Note:

len() is a function which is used to get the count of values present inside the collection.

Program:

Program is an Organized set of instructions which are sent to perform some task

Eg:
```
a = 10
b = 20
Print (a+b)
```

Error
```
a = 10
Print (a+b)
b = 20
```

Variable:

Variable is a name give to memory location where the value is stored

(or)

Variable is a name memory block where the value is store.

→ whenever we want to access the store value within a short period of time variable is used

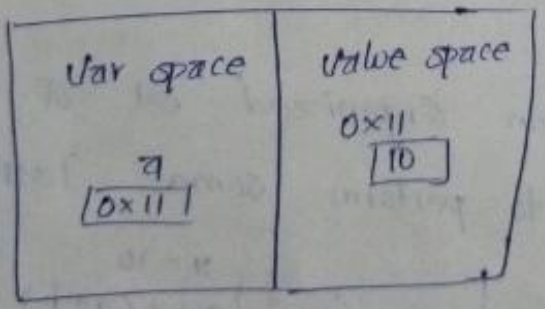
→ Syntax:

The syntax to create a variable

Var_name = value

Eg: a = 10

→ As soon as ~~seen~~ control sees is var creation it will store the value inside value space, address will be given and that address will get store inside variable space with the variable name.



Variable space:

It is a memory location where the address of value is store

(or)

It is a memory where the variable name is store.

Value space:

It is a memory location where the value is store

Multipule variable:

whenever we want to create multipule variable in a single line we are going to make use of multiple variable creation process

The syntax use for the multiple variable is,

Syntax:

Var 1, var 2 ---- varn = val 1, val 2, ---- val n

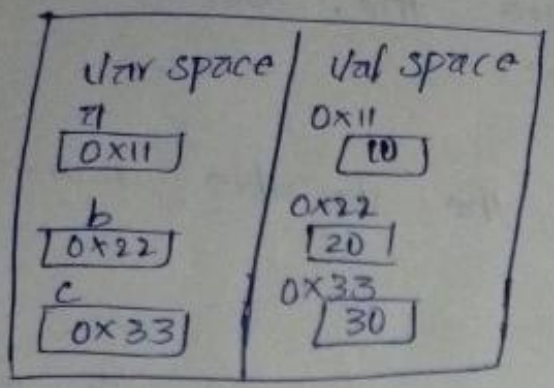
Eg: a, b, c = 10, 20, 30

where, val 1 is stored inside var 1,
val 2 is stored inside var 2, and so on upto valn,
valn is stored inside var n.

Note:

The no: of var is a creating with the help of multiple variable creation should be equal to the no: of values going to assign

a, b, c = 10, 20, 30



Eg: a, b, c = 10, 20,

ValueError : not Enough value to unpack (Expected 3, got 2)

Eg: a, b = 10, 20, 30

ValueError : not Enough value to unpack (Expected 3)

identifier (id()) :

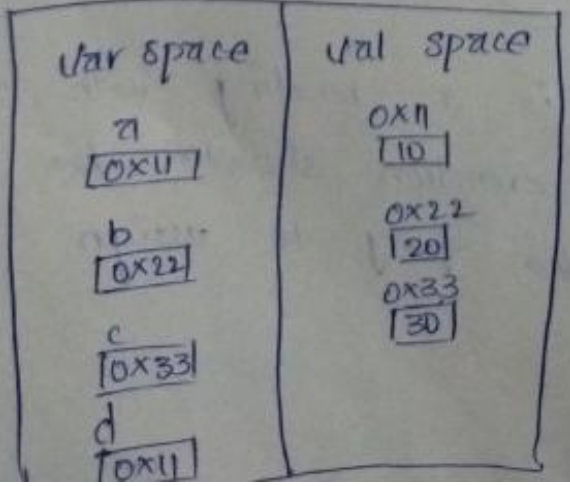
id() fun, is a function which is use to get the address of value stored inside a memory

Syntax :

The syntax used is,

id(var_name)

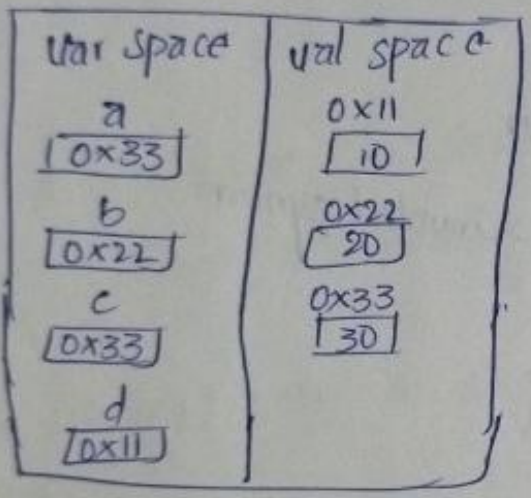
Eg: a, b, c, d = 10, 20, 30, 10



* If two var of the value is same and then the address also same in the both variable.

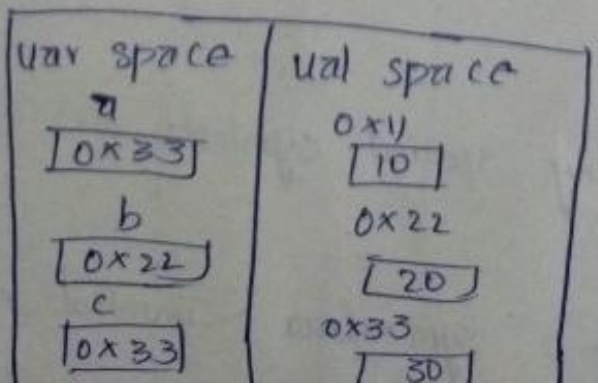
* assume that, sum of a=(a+b) = 30 if the value 30 present in the memory then the address of the value is stored or) change in the var space (a)

Eg: a = a+b
= 10+20
a = 30



val for

* If we assign another d var with 5 then if the value not in the memory that may creat a another location for the value 5 then automatically the value space address is updated in the varspace d



identifiers :

identifiers are the name given to the memory location to identify the value store in the memory

Rules OF identifiers :

1. Indentifier shouldn't start with keyword

Eg:
```
if = 10
True = 90
def = 20
```
} SyntaxError : Invalid syntax

2. shouldn't start with number.

Eg: 1. 2a = 10 SyntaxError : Invalid syntax

2. a2 = 10
   a2
   10

3. shouldn't contain space in between

Eg:
1. a b = 78
   b = 78
} SyntaxError : Invalid syntax

2. b = 67
   b
   67

1. shouldn't have any special symbols (or) characters Except (-)

Eg: 1. a+b = 0 SyntaxError : Invalid syntax

2. a_b = 56 _ = 56
   a_b -
   56

5. can be alphabet, group alphabet, alphabomeric (or) (-)

Eg: a = 10
M = 90
var = 67
abc12 = 30

6. len() of var shouldn't have more that 32 character

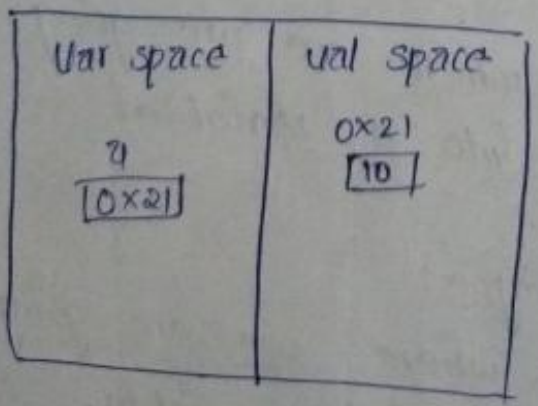
Note:

→ It is not possible to store multiple data in a single block of memory

→ We can store only a single data in a block of memory.

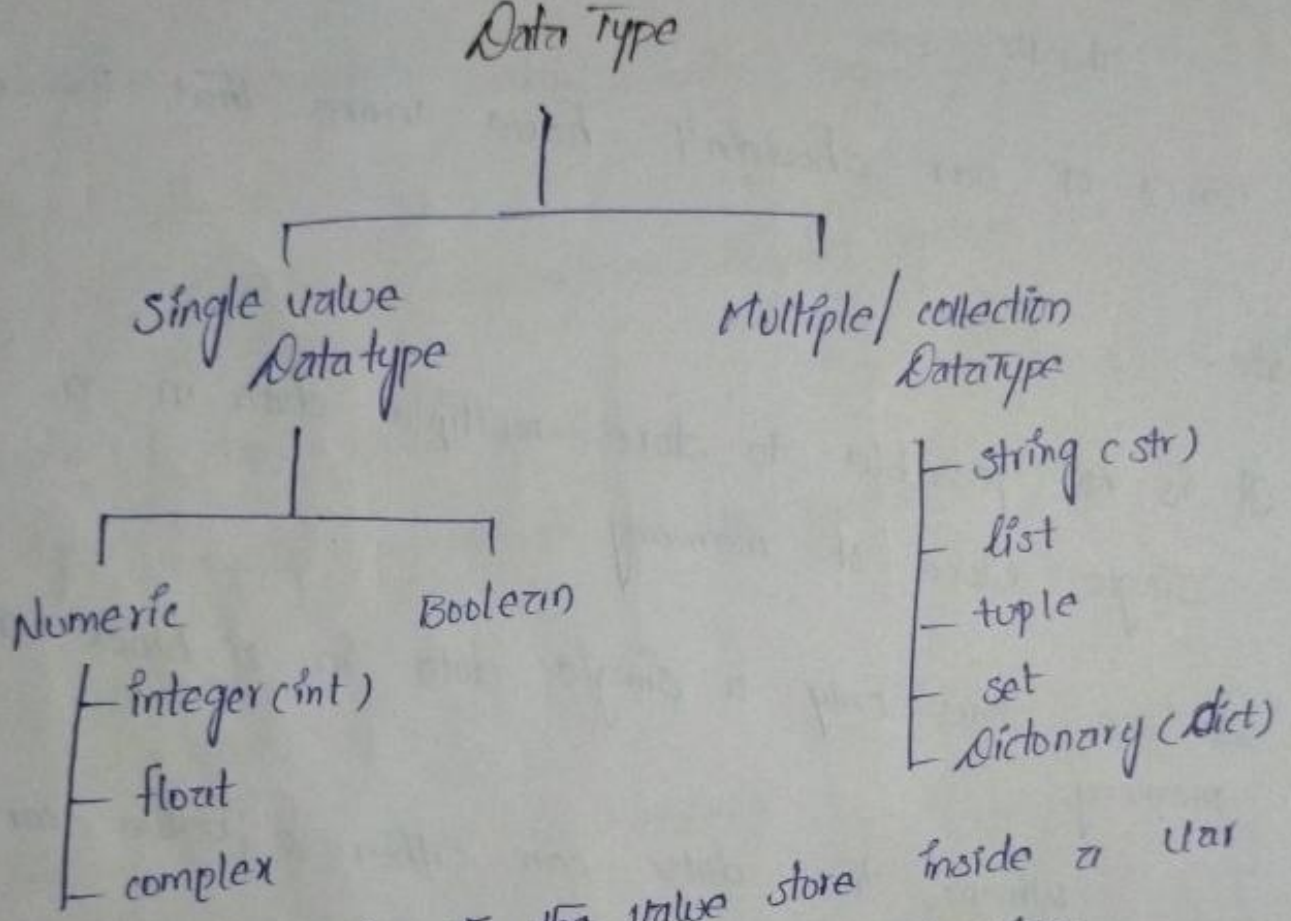
Where, the data can either a value (or) an address

Eg: a = 10



Data Types:

Data types will specifies the type and size of the value store inside a variable.



Based on the size of the value store inside a var data type is classified into two catagories.

i) Single value Datatype:

It is a datatype where we are going to store only a single data into a variable.

ii) Multiple/ collection Datatype:

It is a catagorie where we are going to store multiple value into a single variable.

Note :

For all the data types we will be have two kinds of values.

i) Default value :

→ The meaning of default value is nothing.

→ All the default values are internally Equal to False.

ii) Non-default value :

→ Other than default value, all the values are called as non-default values.

→ Non-default values are internally Equal to ~~false~~. True

type() :

type() is a function which is used to get the type of the value store inside a variable.

Syntax :

type(var/val)

Datatypes :

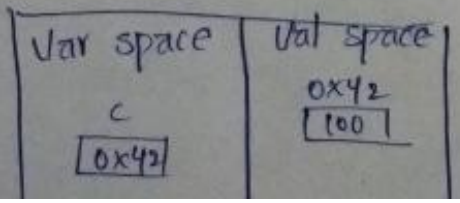
1. Integer (int) :

→ Integer is a real number without decimal point

→ Integer can be Either +ve (or) -ve

→ we can identify all the integer number with 'int'

c = 100



Eg:

1. a = 10
   a = 0
   a = -13
   } type(a) → <class 'int'>

2. '0' is the default val for the Every var it always gives 'False'

```
bool(0)
False.
```

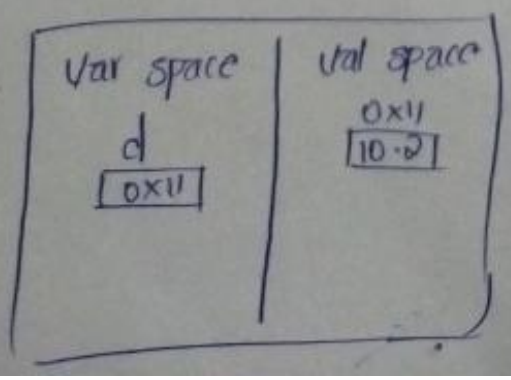
3. Except '0' other numbers are non-default they gives always True.

```
bool(-12)
bool(100)
} True.
```

. float :

→ float is a real number with decimal point.
→ It is not possible to have more than one decimal point in a float number
→ float number can be Either +ve (or) -ve
→ '0.0' is the default value for the float number
→ their is no limit to having floating values.

d = 10.2



Eg:

1. a = 12.3
   a = -78.90
   } type(a) → <class 'float'>

2. bool(0.0)
   False

3. bool(8.9)
   bool(0.56)
   } True.

3. Complex:

complex number is a number which is a combina
of real and ~~compler~~ imaginary number

(or)

The number which is in the form of (a+bj) is called as complex number.

int|float int|float
↑ ↑
a + bj
↑ ↑
Real term img term

j = img num where the value of j is $\sqrt{-1}$

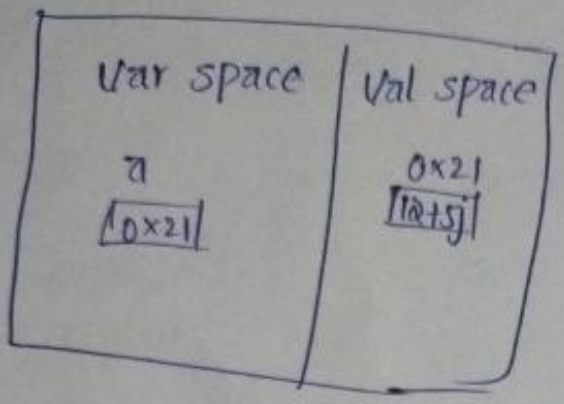
→ It is not possible to any other characters as img number Except j (or) J

→ If we use J as an img number, internally it will get converted into 'j'

→ 0j is the default value for the complex number.

→ It is not possible to write independent 'j' as an img term

a = 12 + 5j



Eg:

1. 3+j
NameError : name 'j' is not defined

2. 3+0j = (3+0j)
3+1j = (3+1j)

3. a = 12+5j
a
(12+5j)
a = 12+8j
a
(12+8j)
} <class 'complex'>.

4. 12+6l
12+6R
1+6i
} Invalid syntax

5. bool (0j) = False

6. 2+8.9j = (2+8.9j)
2.3+4j = (2.3+4j)
-1-1j = (-1-j)
-2.3+7.8j = (-2.3+7.8j)

7. 3j
type(3j)
<class 'complex'>

4. Boolean :

→ In python boolean is going to act as binary language

→ Boolean datatype is having two values

True and False

→ True is internally considered as integer val '1'

→ False is internally considered as integer val '0'

→ True and False are Keywords

→ the default value of boolean data type is False (0)

→ Boolean value can be used in two senarioes

i) we can assign boolean values as a value while creating variable

ii) Boolean value will act as resultant while checking the conditions.

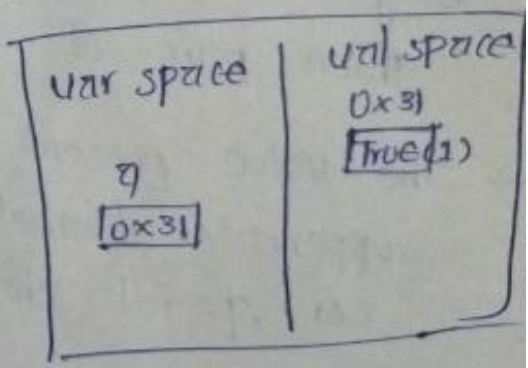
Eg:

1. int (True)
   1
2. int (False)
   0
3. m = True
   n = False
   m
   True
   n
   False
4. a = 10
   b = 20
   a == b
   False

a < b
TRUE

5. True + True
   2
6. True + False
   1

n = True



Multivalue Datatype / collection datatype :

1. String :

String is a collection of characters enclose between pair of single Quote, double Quotes (or) three pair of single Quote.

→ If the string is created with three pairs of single Quote then that string is called as 'doc' string and which is used for comment purpose.

→ The syntax to creat string is,

Syntax :

```
var = 'var1var2var3 ---- varn'
    = "var1var2var3 ---- varn"
    = '''var1var2var3 ---- varn'''
```

→ If we are starting with single Quote we have to Ended with single Quote and same thing for double Quote and three pair of single Quote.

→ The value present inside the string can be Either uppercase alphabet (or) lower case alphabet (or) Number (or) special characters.

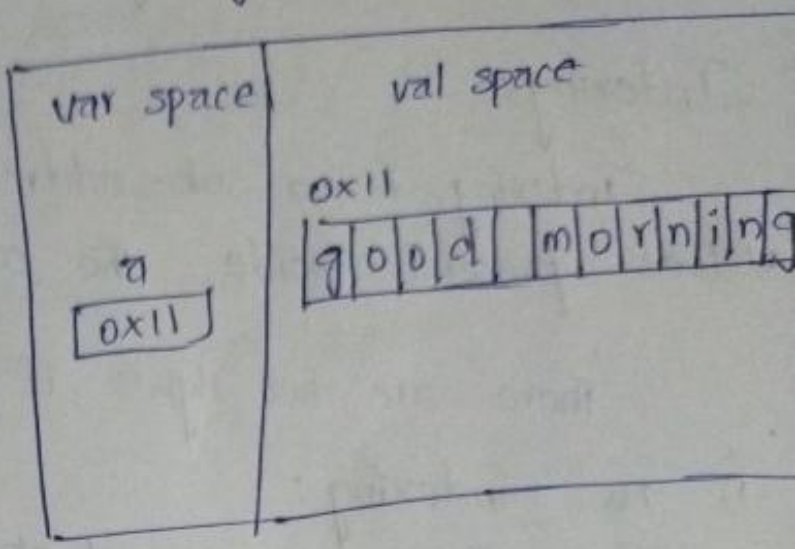
→ If the string is created with respect double Quote internally it will get converted in the form of single Quote.

→ If the

Eg:

1. st = 'hai'
   st
   hai
2. type(st)
   <class 'string'>
3. "hai good afternoon"
   'hai good afternoon'
4. 'happy father's day'
   SyntaxError
5. "happy father's day"
   "happy father's day"
6. 'hai"
   SyntaxError
7. "hai'
   SyntaxError

a = 'good morning'



Memory allocation part collection datatype:

→ As soon as control see the value is of collection datatype it will create layer of memory

→ After creating a layer of memory it will divide that layer into no: of blocks which is exactly Equal to the length of collection

→ The values will get store one-by-one into the created blocks, address will be given to Entire layer and that address will get store with respect to variable name

→ If we want to access individual values from the collection it is not possible to over this problem we have the concept called indexing.

Indexing:

Indexing is a sub-address given to the individual values present inside the collection to identify its location

There are two types of indexing.

i) +ve indexing:

Whenever we want to travel from left to Right of the collection will may use of +ve indexing and +ve will start from '0' till Length of the collection '-1'

ii) -ve indexing:

Whenever we want to travel from Right to left of the collection will may use of -ve indexing and -ve will start from '-1' till '-length' of the collection.

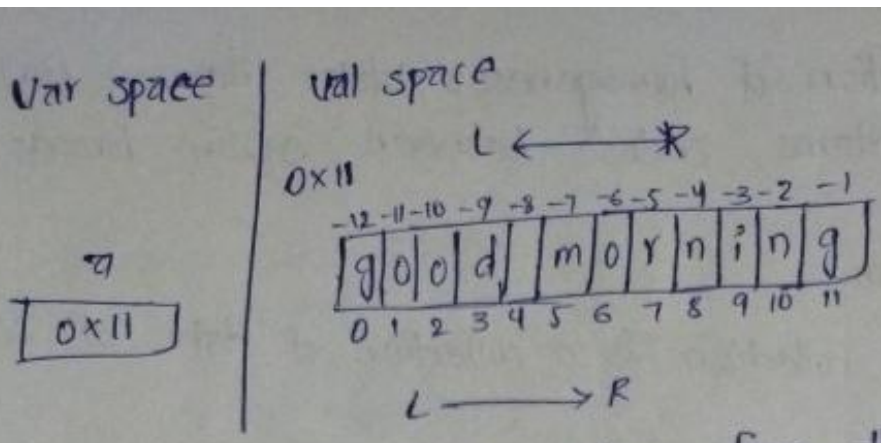
→ System knows only +ve indexing if the user enters -ve index internally it will get converted in the form of +ve index with the help of formula.

+ve indexing = len(collection) + (-ve index)

if we want to get -7 = 12 + (-7)

= 12 - 7

= 5



→ To access the individual values from the collection with respective indexing will make use of syntax.

Syntax:

```
var[index]
```

Eg:

```
a[3]
'd'
a[8]
'n'
a[-5]
'r'
```

Their are two types of collections.

i) Mutable collection:
It is a collection which will allow the user to modify its original value.

ii) Immutable collection:
It is a collection which will allow the user to modify its original value.

→ whenever we want to modify the value present inside the collection we will make use of syntax

Syntax: var[index] = new_value.

→ since, the string will not allow the user to modify its original value it will come under immutable catagorie.

2. List:

List is a collection of homogeneous data items (or) hetrogeneous data items Enclose between square braces []

Homogeneous collection:

Homogeneous collection is a collection of data of same type

Hetrogeneous collection:

It is a collection of data of different types.

The syntax to create list is,

Syntax:

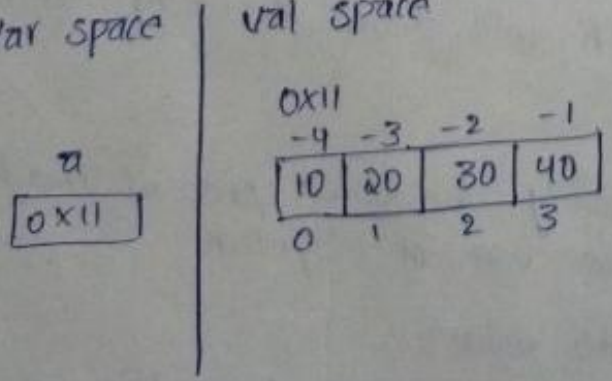
var = [val1, val2, val3 - - - - valn]

→ where, values are seperated by ',' operator and the values can be either homogeneous (or) hetrogeneous

→ The default values for list is [] open and close square braceses

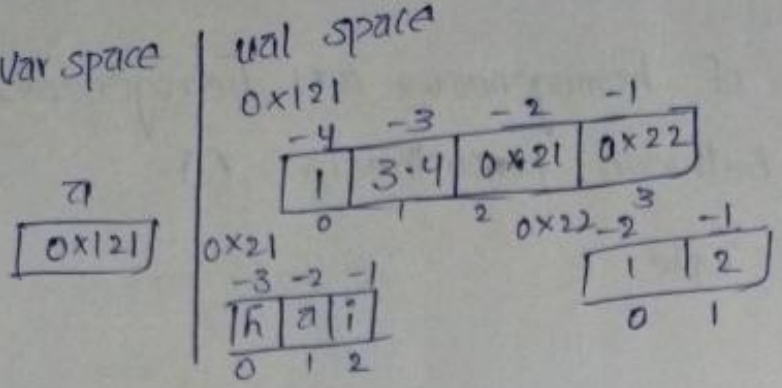
Homogeneous:

Eg: a = [10, 20, 30, 40]



Hetrogeneous

Eg: a = [1, 3.4, 'hai', [1,2]]



a[2][1]

a

a[3][0]

1

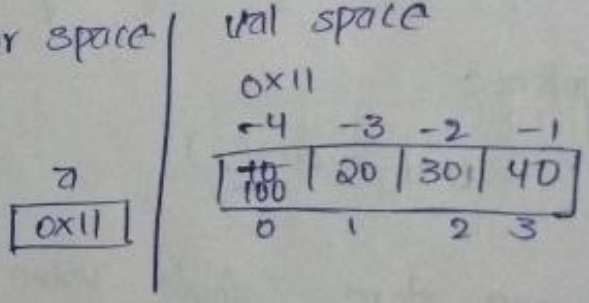
modifing the values present inside a list collection

var[index] = new_value

if we want to modify 10 with 100

a = [10, 20, 30, 40]

a[0] = 100

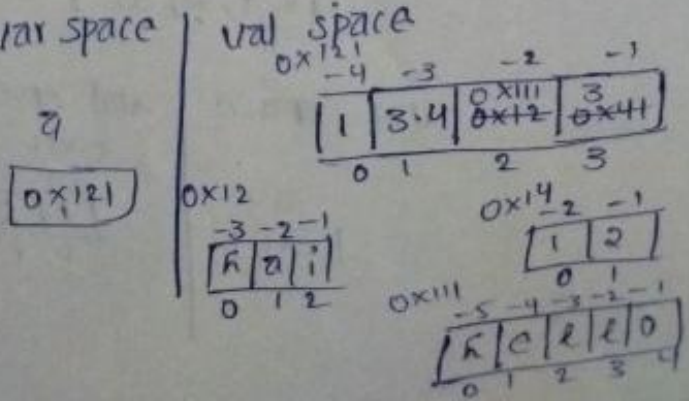


a = [1, 3.4, 'hai', [1,2]]

if we are going modify hai with hello, [1,2] with 3

a[3] = 3

a[2] = hello



→ list will allow the user to modify the original value present in it, so that list is called as mutable collection

3. Tuple:

Tuple is a collection of homogeneous (or) hetrogeneous data items Enclose between paranthesis ()

→ The syntax to create tuple is

Syntax:

var = (val 1, val 2, val 3 ---- valn)

var = val 1, val 2, val 3 ---- valn

→ where, Each and Every values are seperated ',' operator

→ The default value for the tuple is () its a Empty tuple

→ Tuple is used in secured data translation

→ if we want to store only single data item in the tuple we have to make use of syntax.

Syntax:

var = (val,)

Note:

if we store single value in tuple without, then it will take the respective data type of the value which we have store.

a = 1, 2, 3, 4

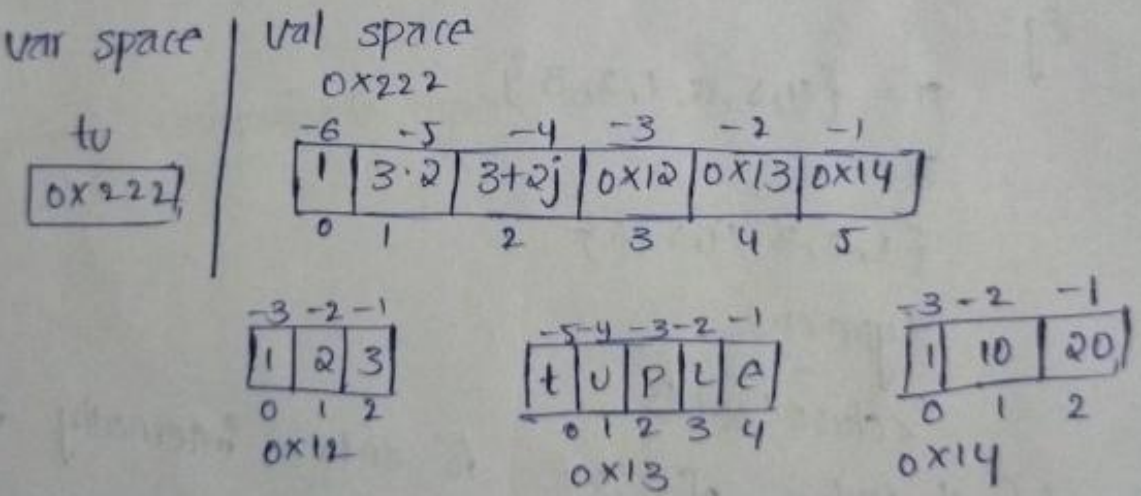
var space	val space	
a	0x11	a[2] = 3
0x11	-4 -3 -2 -1	a[0] = 1
	1 2 3 4	
	0 1 2 3	

It is not possible to modify the values store in the tuple so that we can call tuple as immutable collection

Eg: b = 10, 20, 30, 40

```
type (b)
<class 'tuple'>
b[2]
30
b[2] = 300
TypeError: 'tuple' object does not support item assignment
```

Eg: tu = (1, 3.2, 3+2j, (1,2,3), 'tuple', [1,10,20])



9. Set :

Set is an unordered and non-duplicate collection of homogeneous data items (or) non-homogeneous data items Endosed between flower braceses {}

The syntax to creat set is

Syntax :

var = {val1, val2, ---- valn}

→ Indexing is not ~~present~~ possible because the data which we have store not be in order

→ The duplicate values are not present in set

→ The order of storing the values will not be same

Eg:

```
a = {4,5,6,1,2,3}
a
{1,2,3,4,5,6}
type(a)
<class 'set'>
```

→ The default value of set is set() internally it consider as False

Eg: bool(set())

→ We can't modify the value in the set, but can add the new values by using the in-built function

Eg:

```
a = {1,2,3,4,5,6}
a.add(67)
a
{1,2,3,4,5,6,67}
```

```
b = {1, 2, 3, 4, 5, 2, 3, 4, 6, 7}
b
{1, 2, 3, 4, 5, 6, 7}
```

→ Only immutable data type are allowed to store in set like (int, str, tuple, float, complex, bool)

→ list and set are mutable data types so these are not allowed to store in the set

```
a = {1, 2.3, 2+7j, False, 'hello', (1,2,3)}
a
{1, 2.3, 2+7j, False, 'hello', (1,2,3)}

a = {1, 2.3, 2+7j, False, 'hello', (1,2,3), {1,2,3}, [3,4,5]}
a
TypeError: unhashable type: 'set'
```

5. Dictionary:

Dictionary is a collection of key value place Enclose between pair of flower braceses

The syntax to create dictionary is,

Syntax: `var = {K1:V1 , K2:V2 , .... Kn:Kn}`

→ where, key value pair seperated by ',' operator and key and, key and values are seperated by ':' colon operator.

→ In dictionary we can store only immutable data items as a key and values can be anything

→ It is not possible to store duplicate keys in dictonary, if we give duplicate keys the value of first key will get overided by the value of next key

→ Indexing is not possible in dictionary, but we can access the values with the help of keys and the syntax used is

Syntax: `var [key]`

→ If we want to modify the value present inside the dictionary, we will make use of syntax is

Syntax: `var [key] = new_value`

→ The default value for dictionary is {}

Memory allocation for dictionary:

→ As soon as control got to know the value is of dictionary it will create two layers of memory.

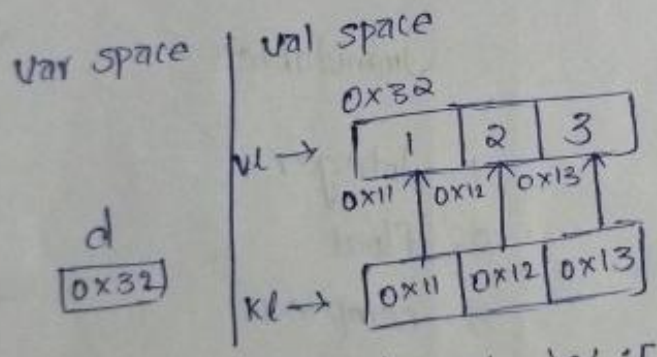
i) key layer:
It is a layer where the keys are store

ii) value layer:
It is a layer where the keys values are store.

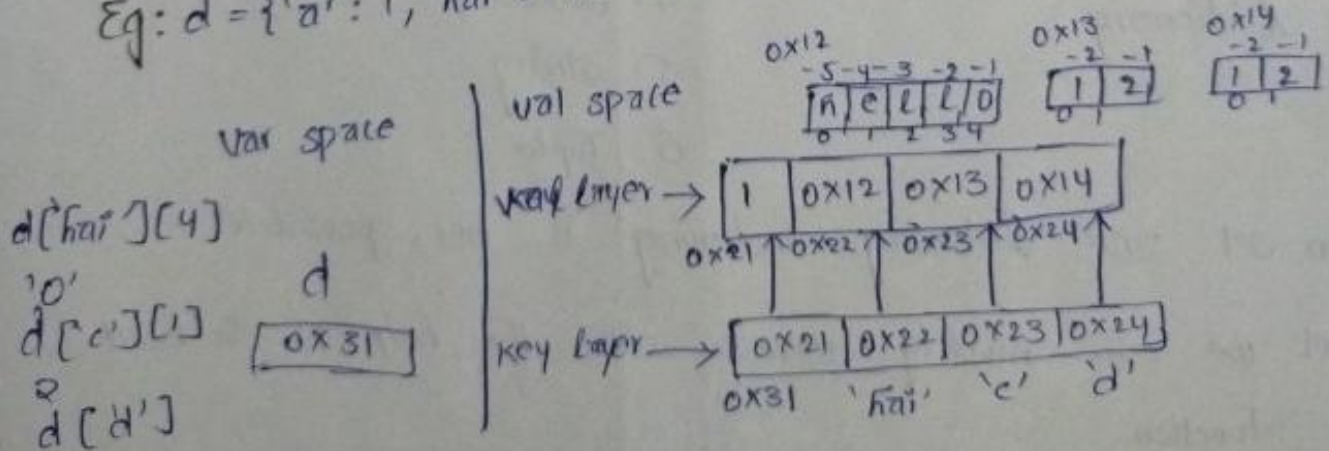
→ For the control the visible layer is key layer

→ Since, dictionary will the user to modify its original value it will comes under mutable categorie.

Eg: d = {'a': 1, 'b': 2, 'c': 3}



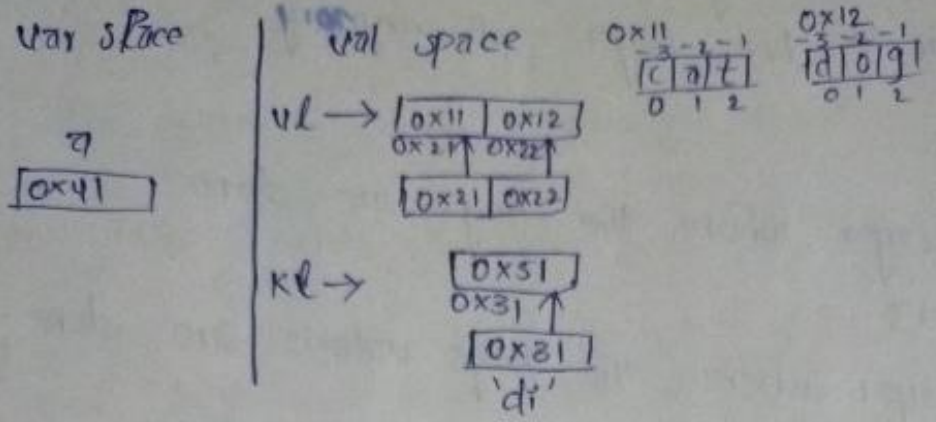
Eg: d = {'a': 1, 'hai': 'hello', 'c': [1,2], 'd': (1,2)}



d['hai'][4]
'o'
d['c'][1]
2
d['d']
(1,2)
d['c'][1] = 25
d
d['e'] = 5
d

d = {'a': 1, 'hai': 'hello', 'c': [1,25], 'd': (1,2), 'e': 5}

Eg: a = { 'di' : {'c' : 'cat', 'd' : 'dog'}}



classification of mutable & immutable collection data types

Mutable	Immutable
1. list	1. Integer
2. set	2. Float
3. Dictionary	3. complex
	4. Boolean
	5. string
	6. Tuple

* In set and dictonary indexing is not possible
* set we can modify values with the help of In-built function.

Default values:

Default values are the values which will be acting as intial values for all the datatypes.

DataType	Default values
1. Integer	0
2. float	0.0
3. complex	0j
4. Boolean	False
5. string	''
6. list	[]
7. Tuple	()
8. set	set()
9. dict	{}
10. generic	None.

Type Casting:

Type casting is a Finominal of converting the data of onetype into another type based on users requirment.

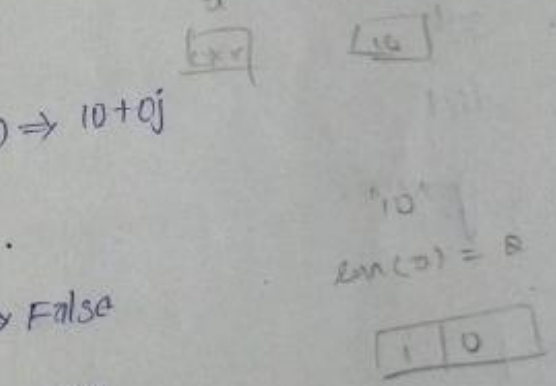
The syntax to perform type casting is

Syntax: dest_var = dest_type (source_var)

1. Integer:

Performing type casting integer into different datatypes.

Source_type	Dest_type
int	float a = 10 float (a) ⇒ 10.0
int	complex a = 10 complex (a) ⇒ 10+0j
int	Boolean a = 0 bool (a) ⇒ False a = 1 bool (a) ⇒ True
int	string (memory will get wasted) a = 10 str (a) ⇒ '10' len (a) → 2
int	list, tuple, set dict a = 10 → we get 'typeError'



It is not possible to convert any single value datatype into collection datatype Except string.

2. Float :

Performing typecasting float into different datatypes

Source_type	dest_type
float	integer a = 12.3 int(a) ⇒ 12
float	complex a = 12.3 complex(a) ⇒ 12.3+0j
float	Boolean a = 0.0 bool(a) = False a = 0.1 bool(a) = True
float	string (memory will get wasted) a = 12.34 str(a) ⇒ '12.34' len(a) ⇒ 5

Float is not possible to convert single value datatype into collection datatype Except string.

3. Complex :

Performing Typecasting complex into different datatype.

Source_type	dest_type
Complex	Boolean a = 3+5j bool(a) ⇒ True a = 0j bool(a) ⇒ False
complex	string (memory will get wasted) a = 3+5j str(a) ⇒ '(3+5j)'

Here, complex it not possible to convert into int, float, list, tuple, set, dict but, complex is allow to convert into boolean & string.

4. Boolean :

Performing typecasting Boolean into different datatypes

Source_type	dest_type.
Boolean	integer a = False int(a) ⇒ 0 a = True int(a) ⇒ 1
Boolean	float a = false float(a) → 0.0 a = True float(a) ⇒ 1.0

Boolean → string

a = False
str(a) ⇒ 'False'

a = True
str(a) ⇒ 'True'

Boolean → complex

a = True
complex(a) ⇒ (1+0j)

a = False
complex(a) ⇒ (0+0j)

Here, Boolean is not possible to convert into the list, tuple, set and dictionary remaining these are allowed to convert the typecasting.

5. String:

Performing typecasting string into different datatypes.

Source_type → dest_type

string → int (only if str is having numeric data without decimal point)

a = 'hello'
int(a) ⇒ Error
a = '123'
int(a) ⇒ 123
a = '123.5'
int(a) ⇒ Error
int(float(a)) ⇒ 123

string → float (numeric data with (or) without decimal point)

a = '123'
float(a) ⇒ 123.0
a = '2.36'
float(a) ⇒ 2.36

a = 'hai'
float (a) ⇒ Error

a = '3+2j'
float (a) ⇒ Error

String

complex (only if string is having numeric data)

a = 'hai'
complex (a) ⇒ Error

a = '10'
complex(a) → (10+0j)

a = '10.3'
complex(a) ⇒ (10.3+0j)

a = '1+2j'
complex(a) ⇒ (1+2j)

string

Boolean

a = 'h'
bool(a) ⇒ True

a = ''
bool(a) ⇒ False

string

list
a = 'good afternoon'
list(a)
→ ['g','o','o','d',' ','a','f','t','e','r','n','o','o','n']

String

tuple
a = 'Hello world'
tuple(a)
→ ('h','e','l','l','o',' ','w','o','r','l','d')

String → set

a = 'good afternoon'

set(a)

→ {'g', 't', 'd', 'e', ' ', 'o', 'a', 'n', 'r', 'f'}

string → dictionary

a = 'good morning'

dict(a) ⇒ Error

string to dictionary is not possible.

6. list:

Note:

We can't convert any collection data item into single value datatype Except boolean.

Source_type	Dest_type
list a = [1,2,3,4] b = []	Boolean bool(a) ⇒ True bool(b) ⇒ False
	string str(a) → '[1,2,3,4]'
	Tuple tuple(a) → (1,2,3,4)
	set set(a) ⇒ {1,2,3,4}
	Dictionary X dict(a) → {only if list is having (k,v)} a = [(1,2),(3,4),(5,6)] dict(a) ⇒ {1:2, 3:4, 5:6}

a = [(1,2), ('a', 4), ([1,2], 5)]

dict(a)

⟶ TypeError : unhashable type : 'list'

7. Tuple :

source_type	dest_type
Tuple a = (1,2,3,4) b = ()	Boolean bool(a) ⇒ True bool(b) ⇒ False
	string str(a) ⇒ '(1,2,3,4)'
	set set(a) ⇒ {1,2,3,4}
	list list(a) ⇒ [1,2,3,4]
	Dictionary : dict(a) ⇒ Error Because dictionary contains key values (5,6)

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

dict(a)

⇒ {1:2, 3:4, 5:6}

8. set :

source_type

a = {10, 20, 30}

b = { }

dest_type

Boolean

bool(a) → True

bool(b) → False

String

str(a) = '{10, 20, 30}'

List

list(a) = [10, 20, 30]

Tuple

tuple(a) = (10, 20, 30)

Dictionary

dict(a) ⇒ Error

Only set is having (K, V)

a = {(1, 2), (3, 4)}

dict(a) ⇒ {1:2, 3:4}

9. Dictionary :

source_type

a = {'a':1, 'b':2, 'c':3}

dest_type

Boolean

bool(a) ⇒ True

bool({}) ⇒ False

Note :

i) values() :

- is a function which is use to access only values from the dictionary

and, the syntax is : var.values()

ii) items() :

It is a function which is use to get entire key value pair from the dictionary.

Syntax : var.items()

dictionary

String

str(a) ⇒ '{'a':1, 'b':2, 'c':3}'

List

list(a) ⇒ ['a', 'b', 'c']

list(a.values()) ⇒ [1,2,3]

list(a.items()) ⇒ [('a',1), ('b',2), ('c',3)]

Tuple

tuple(a) ⇒ ('a','b','c')

tuple(a.values()) ⇒ (1,2,3)

tuple(a.items()) ⇒ (('a',1), ('b',2), ('c',3))

set

set(a) ⇒ {'a', 'b', 'c'}

set(a.values()) ⇒ {1,2,3}

set(a.items()) ⇒ {('b',2), ('E',3), ('a',1)}

Slicing :

Slicing is a fnominan of Extracting the group of data from the original collection.

→ Slicing can be done on list, string and Tuple

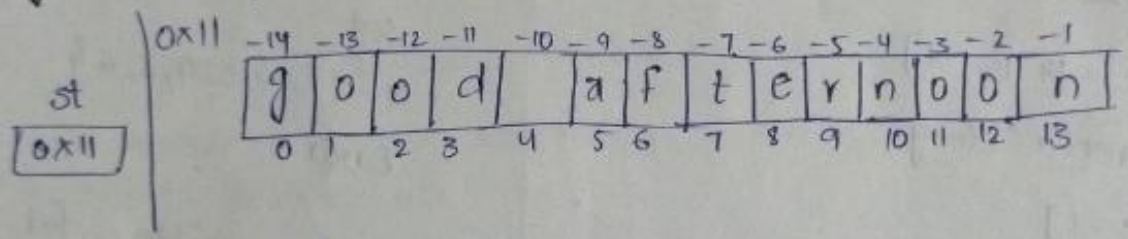
→ The syntax to perform slicing is

Syntax : Var = [SI : EI ± 1 : updation]

→ When we are travelsing from left to Right of the collection will make use of 'End Index + 1'

→ And, while travelsing from Right to left of the collection will make use of 'End Index - 1'

eg : st = 'good afternoon'



st = 'good afternoon'

st[3]
>>> d

st[0:3+1]
>>> good

st[5:13+1]
>>> afternoon

st[-14:-11+1]
>>> good

st[-9:-1+1]
>>> afternoon

st[-1:-9-1:-1]
>>> noonretfa

st[13:5-1:-1]
>>> noonretfa

st[0:13+1:2]
noon >>> go fend

st[0:13+1:2]
>>> go fend

st[13:0-1:-2]
>>> 'n'

st[-2:-14-1:-2]
>>> onef og

st[1:13+1:2]
odation

st[-1:-14-1:-2]
odation

Var [SI : EI ± 1 : up]

* If SI == 0
  ⇒ var[: EI±1 : up]

* If EI ± 1 == len(var)
  → var[SI : : up]

* If up == 1
  ⇒ var[SI : EI ± 1 :] / [SI : EI ± 1]

Eg: st = 'good afternoon'

st[: :]

>>> 'good afternoon'

st[5 : :]

>>> 'afternoon'

st[-1 : -14-1 : -1]

>>> 'noonretfa doog'

st[: : -1]

>>> 'noonretfa doog'

List:

a = [1, 2, 3, 4, 5, 6]

a

>>> 1, 2, 3, 4, 5, 6

a[: :]

1, 2, 3, 4, 5, 6

a[: : -1]

6, 5, 4, 3, 2, 1

Tuple

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

a

1, 2, 3, 4, 5, 6

a[: :]

1, 2, 3, 4, 5, 6

a[: : -1]

6, 5, 4, 3, 2, 1

a[: : 2]

1, 3, 5

a[1 : : 2]

2, 4, 6

set

a = {1, 2, 3, 4, 5, 6}

Error

slicing is possible for list, tuple and string

Copy Operation :

It is a finamiman of copying the content of one var to another variable, where content can be either var space (or) value space.

→ copy operation is classified into three types

i) General / Normal copy

ii) Shallow copy

iii) Deep copy

i) General / Normal copy :

It is a type of copy operation where the content of var. space of one var will get copy to another var

The syntax used is

Syntax : dest_var = Source_var

→ In normal copy the modification with respective one var will Effect another var.

Eg: a = [10, 20, 30, 40]

b = a

Mod wrto a → a[2] = 50

a

>>> 10, 20, 50, 40

b

>>> 10, 20, 50, 40

Mod wrto b → b[0] = 1000

b

>>> 1000, 20, 50, 40

a

>>> 1000, 20, 50, 40

a = [1, 2, [12, 45], 78]

b = a

Mod wrto a

a[2][0] = 120

a

[1, 2, [120, 45], 78]

b

[1, 2, [120, 45], 78]

Mod w.r.to b

b[2][1] = 450

b

[1, 2, [120, 450], 78]

a

[1, 2, [120, 450], 78]

ii) Shallow copy:

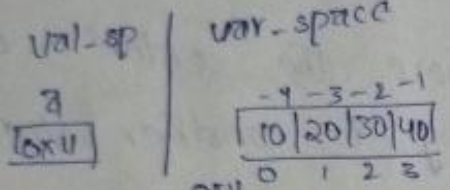
Shallow copy is a finamiran of copying the content of val space of one var to another var.

The syntax to perform shallow copy is

Syntax: dest_var = source_var.copy()

→ In shallow copy the modification with respective one var to another var it will not effect if it is linear list

Eg: a = [10, 20, 30, 40]



Modify w.r.to a

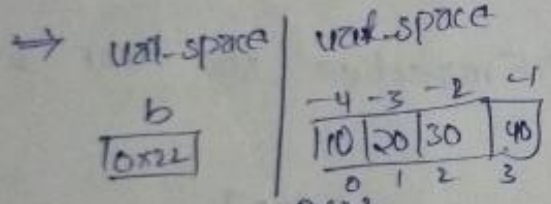
a[0] = 5

>>> a = [5, 20, 30, 40]

b

>>> [10, 20, 30, 40]

b = a.copy()



Modify w.r.to b

b[3] = 400

>>> [10, 20, 30, 400]

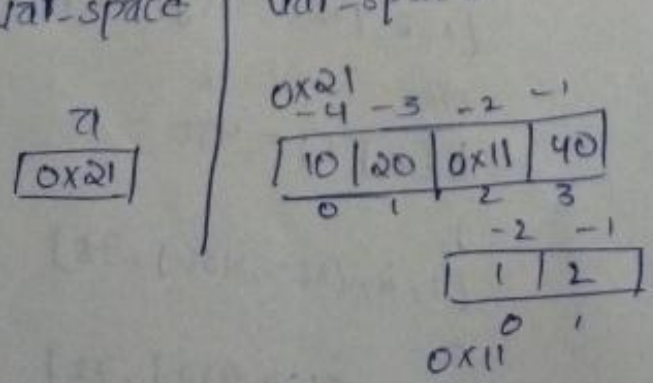
a

>>> [5, 20, 30, 40]

When we are performing copy operation a&b values are same in both var but address is different.

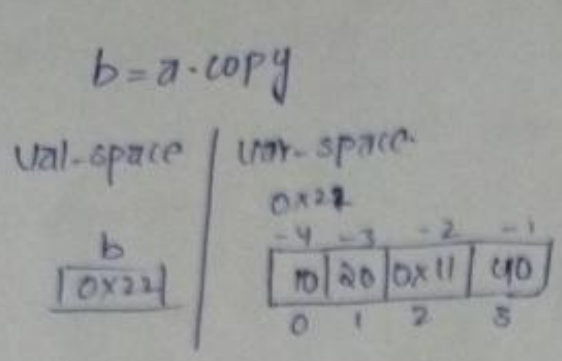
Eg: a = [10, 20, [1, 2], 40]

b = a.copy()



In case of nested collection the modification with respective nested collection of one var will effect another var.

b = a.copy



modifing w.r.b a

a[2][0] = 1000

>>> a
[10, 20, [1000, 2], 40]

>>> b
[10, 20, [1000, 2], 40]

mod w.r. to b

b[2][1] = 2000

>>> b
[10, 20, [1000, 2
40]

>>> a
[10, 20, [1000, 2000

(iii) Deep copy:

Deep copy is a finominan of copying the entire content from one var to another var.

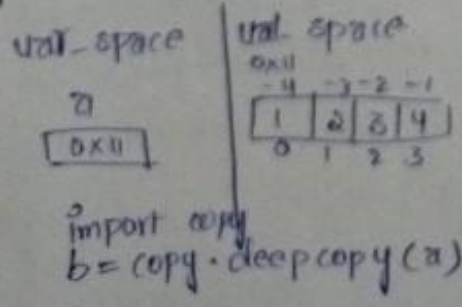
→ we can't do deep copy operation directly for that we will import the functionalites of copy module then from that we will access deep copy function.

The syntax to perform deep copy is,

Syntax: import copy
dest_var = copy.deepcopy(source_var)

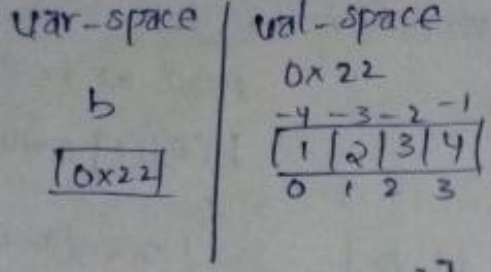
→ In deep copy operation the modification w.r. to one var will not effect another var, it may be linear list (or) nested list

Eg: a = [1, 2, 3, 4]

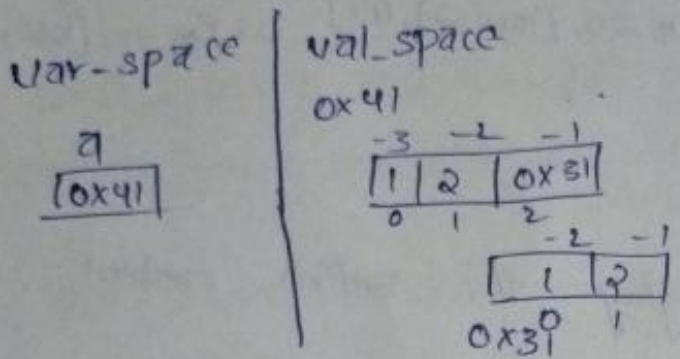


import copy
b = copy.deepcopy(a)

b
>>> [1, 2, 3, 4]



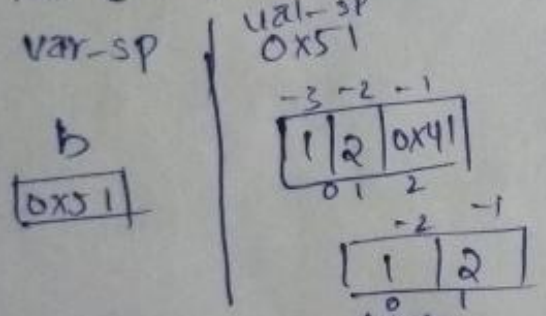
Eg: a = [1, 2, [1, 2]]



b = copy.deepcopy(a)

b

>>> [1, 2, [1, 2]]



Operators:

Operator is a type of library function which is use to perform some specific task

→ To perform any task two things are required operands and operators.

→ Operators are classified into seven types.

i) Arthematic operator
ii) Logical operator
iii) Bitwise operator
iv) Relational operator
v) Assignment operator
vi) Membership operator
vii) Identity operator

i) Arthematic Operator:

1. Addition Operator (+):

It is an operator which is use to get the sum of minimum two operands (or) maximum n-operands

→ addition operator will support for all the single value data types.

int + int ⇒ int
10 + 20 ⇒ 30
int + float ⇒ float
int + complex ⇒ complex
int + Boolean → int

float + float = float
float + int = float
float + complex → complex

complex + complex ⇒ complex
complex + int → complex
complex + float ⇒ complex

Eg: Integer:

10+20 → 30
10+2.3 ⇒ 12.3
10+5+2j ⇒ 15+2j
10 + True → 11

float:

2.3+2.4 ⇒ 4.7
2.3+10 ⇒ 12.3
2.3+1.2j ⇒ 3.3+2j

complex:

1+2j + 3+2j ⇒ 4+4j
1+2j + 10 ⇒ 11+2j
2+3j + 1.0 ⇒ 3.0 + 3j

→ In case of collection '+' Operator is going to act as concatination Operator.

→ We can perform concatination on collection data types only if both the operands are of same type.

→ concatination will not support for set and dictionary

Eg:

```
12+'hai'
>>> Error
4.5+[1,2,3]
>>> Error
True + [1,2]
>>> Error
True+'hai'
>>> Error
'hai' + [1,2,3]
>>> Error
```

```
'hai'+'hello'
>>> haihello
[1,2,3]+[1,2,3]
>>> [1,2,3,1,2,3]
>>> (1,2,3)+(1,2,3)
>>> (1,2,3,1,2,3)
{1,23}+{5,6}
>>> Error
{'a':1}+{'b':2}
>>> Error
```

2) Subtraction Operator (-)

Subtraction operator will support for all the single value data types and between two sets.

→ In case of set, the result will be all the values from the first set which are not present in second set

Eg:
```
20-10 → 30
23-10.2 → 12.8
10-(5+6j) → (5-6j)
23.4-10.2 → 11.2
10.8-(1+2j) → 11.3+2j
'hai'-'X' → Error
{1,2,3}-{5,6,2} → {1,3}
{'a':1}-{'b':2} → Error
{'hai',1}-{1,2} → {'hai'}
```

Pg 1

3) Multiplication Operator (*)

Multiplication operator is use to find the product between two (or) operands

→ star (*) operator will support all the single value data types.

→ In collection, we can multiply string, list and tuple with only integer number

Eg: 2*3 → 6

2.3*2 → 4.6

2.3*4.2 → 9.66

2.1*(1+1j) → (2.1+2.1j)

(1+4j)*(2+3j) → (-10+11j)

'hai'*'hello' → Error

'hai'*2 → haihai

[1,2,3]*2 → [1,2,3,1,2,3]

(1,2)*3 → (1,2,1,2,1,2)

'hai'*2.3 → Error

{1,2}*3 → Error

'hai'*(1,2) → Error

4. Divison Operator

divison operator is further classified in to three types.

1. True division (/)
2. floor division (//)
3. Modulus (%)

1. True division :

It is a operator which is used to get Exact division Output

2. Floor Division:

It is a division operator which is used to remove floating values from the exact division output

3. Modulus Division:

It is a division operator which is used to get the reminder of division process.

Note :

Floor division and modulus operator will not support complex and collection data type.

Eg: 5/2
>>> 2.5

(2+4j)/2
>>> (1+2j)

(2+3j)//(4+2j)
>>> Error

[1,2]//45
>>> Error

(12+3j)%2
>>> Error

5//2
>>> 2

(2+4j)//2
>>> Error

'hai'/2
>>> Error

10%.5
>>> 0

7%.2
>>> 1

4/2.1
>>> 1.9047
>>> 1.0

4%.2.1
>>> 1.9024

(2+8j)/(4+2j)
>>>(0.7+0.4j)

'hai'/'hello'
>>> Error

3.4%.3
>>> 0.3999

5. Power Operator (* *):

Power operator is used to multiply the given operator based on the specified number of items.

The syntax used is:

Operand ** n

```
>>> 2**3
>>> 8
>>> 2.3**2
>>> 5.28999
```

```
>>> (1+2j)**2
(-3+4j)
>>> 2.3**1.2
2.7168984
```

```
>>> 'hai'**2
Error
```

ii) Logical Operator:

1. Logical And Operator:

Logical And operator will give the result of True if both the Operands are true, else if will return False

The syntax used is:

Syntax: Op1 and Op2

control will check whether the 1st Operand is true or false, if 1st Operand is True it will print Op2 as result

→ If Op2 is false it will print Op1 as result

```
2 and 3
>>> 3
True and False
>>> False
```

```
{} and 'hai'
>>> {}
() and 'hello'
>>> 'hello'
```

```
set() and ''
>>> set()
```

2. Logical OR Operator:

Logical OR Operator will return true if only one of the Operand is true else it will return false.

→ The syntax used is op1 or op2 control will check op1, if op2 is equal true it will return op1 else return op2

```
if op1 == True
    o/p = op1
if op1 == False
    o/p = op2
```

Eg: set() or 3

```
>>> 3
```

'' or {}

```
>>> {}
```

True or False

```
>>> True
```

36 or 89

```
>>> 36
```

3. Logical NOT Operator:

NOT is a logical operator which will return the result as true if input is false else it will return false

Syntax is not (op)

```
if op == True
    o/p = False
if op == False
    o/p = True
```

not (0)

```
>>> True
```

not (False)

```
>>> True
```

not ({})

```
>>> True
```

not (34)

```
>>> False
```

iii) Bitwise Operator:

1. Bitwise And (&) Operator:

It is an operator which is use to perform the '&' operation between two operands by considering bit-by-bit

Eg: 5 & 7
>>> 2

2|5
2|2 - 1
1 - 0
⇒ 101

2|7
2|3 - 1
1 - 1
⇒ 111

⇒ 101
⇒ 8

101
111
101 → 5

11 & 22
>>> 2

2|11
2|5 - 1
2|2 - 1
1 - 0
⇒ 1011

2|22
2|11 - 0
2|5 - 1
2|2 - 1
1 - 0
⇒ 10110

⇒ 01011
01110
01010

1 0 - 0
1 1 - 1
0 1 - 0
0 0 - 0

2. Bitwise OR(|) Operator:

It is an operator which is use to perform bit-by-bit OR (|) operation between two operands

1 0 - 1
0 0 - 0
0 1 - 1
1 1 - 1

Eg: 5|8
>>> 13

5 - 0101
8 - 1000
1101
8+4+1
⇒ 13

3. Bitwise NOT (~) Operator:

It is an operator which is use to perform bitwise NOT (~) operation to the given operand.

→ To perform bitwise NOT operation, internally controller will consider a formula.

-(n+1)

where, n is an integer value.

→ If the value of n is positive result will be in the form of -ve value, else result will be positive value

Eg: n = 7 | n = -5

⇒ -(n+1) | → -(-5+1)

⇒ -(7+1) | ⇒ -(-4)

⇒ -8 | → 4

4. Bitwise XOR (^) Operation:

It is a bitwise operator which is use to perform the XOR (^) operation between two operands.

→ The o/p of XOR operation is true if both the operands are different from eachother; else it will return False.

Eg: 3 ^ 7 | 3 - 0 1 1

>> 4 | 1 1 1

1 0 0

2^2 2^1 2^0

5. Bitwise Rightshift (>>) Operator:

It is an operator which will eliminate the binary value from right size base on the value of 'n'.

→ Syntax used is: Operand >> n

Eg: 5 >> 2 → 0010 >> 1

→ 1 ⇒ 001

12 >> 1 → 1100 >> 1

→ 6 → 6

4 >> 4 ⇒ 0 0100

⇒ 0 ⇒ 0

6. Bitwise left shift (<<) Operator:

It is an operator which will add 0's in the Right size side of given operand based the number of shifts.

Syntax: The syntax used is

Op << n | 7 << 2

Eg: 3 << 1 | 1100

→ 0110 ⇒ 6 | ⇒ 16+8+4

iv) Relational Operator:

These are the operator which are used to check how the first Operand is related is second operand.

→ the o/p of Relational operator will be in the form of boolean values (True and False).

1. Relational Equal to (==):

It is an operator which will give the result as True if both the operands are Equal, Else it will return False.

Eg: 12==12
>>> True

12.0==12
>>> True

'hai'=[1,2,3]
>>> False

{1,2,3}=={1,2,3}
>>> True

{1,2,3}=={4,5,6}
>>> False

{1,2,3}=={1,2,3,4}
>>> False.

2. Relational not Equal to (!=):

It is an operator which will return they result as True if both the operands are not Equal, Else it will return False.

Eg: [1,2]!=10
>>> True

12!==12
>>> False

3. Greaterthan (>) Operator:

It is an Operator which is use to find (or) check wheather the first Operand is greater than the second operand (or) not

→ It will return True if first operand is greater than second operand Else it will return false

→ greater-than operator will not work for complex and dictionary data

Collection 1 collection 2

$v_1, v_2, v_3 \dots v_n > val_1, val_2, val_3 \dots val_n$

if $v_1 > val_1$
→ True

if $v_1 < val_1$
→ False

if $v_1 == val_1$
go and check the next val

Note:
Greater than operator will support to set collection but we will not get the stable output.

Eg: $10 > 2$
→ True

$5 > 5$
→ False

$2-3 > 1$
→ True

$(10+3j) > 9$
→ Error

True > False
→ True

'hai' > 'hello'
→ False

'hello' > 'haiaa'
→ True

'hai' > [1,2,3]
→ Error

[1,2,3] > [3,4,5]
→ False

[1,2] > [1,2,3]
→ False

[1,2,3] > [1,2]
→ True

(1,2,3) > (1,2)
→ True

(2,3,4) > (2,5,1)
→ False

{1,2,3} > {1,2,3}
→ False

{1,2,3} > {3,4}
→ False

{3,4,5} > {2,3,4}
→ False

{3,4,5} > {3,4}
→ True

{'a':1} > {'b':1}
→ Error

4. Less Than (<) Operator :

It is an operator which will return the result as True if first operand is less than second operand hence it will return false

collection 1 collection 2

$v_1, v_2, v_3 \cdots v_n < val_1, val_2, val_3 \cdots val_n$

if $v_1 < val_1$
→ True
if $v_1 > val_1$
→ False
if $v_1 = val_1$
go and check the next val.

Eg: $12 < 60$ → True
$3.4 < 1.2$ → False
False < 1 → True

$(1+2j) < (2+3j)$ → Error
'hai' < 'bye' → False
'hai' < 'haii' → True

[12, 34] < [12, 1] → False
[12, 24] < [12, 24] → False
[12, 1] < [12, 34] → True

5. Lessthan (or) Equal to (<=) Operator :

It is an operator which is a combination of lessthan and Equal to operator

→ It will return True if Operand 1 is less than operand 2 (or) $Opd_1 = Opd_2$

collection 1 collection 2

$v_1, v_2, v_3 \cdots v_n <= val_1, val_2, val_3 \cdots val_n$

if $v_1 < val_1$
→ True
if $v_1 > val_1$
→ False
if $v_1 == val_1$
it will compare for other val,
it will compare to 2nd collection the result is

Eg:

12 <= 56
→ True

12 <= 12
→ True

1.2 <= 1.2
→ True

(1+2j) <= (1+2j)
→ Error

'hai' <= 'hai'
→ True

'hai' <= 'hello'
→ False

'hello' <= 'hai'
→ False

[12, 3] <= [12, 3]
→ True.

6. Greater Than (or) Equal to (>=) Operator:

It is an operator which is a combination of greater than and Equal to operator.

→ It will return result as True if operand 1 is greater than operand 2 (or) operand 1 is Equal to operand 2.

else it will return false.

Eg:

collection 1 collection 2

$u_1, u_2, u_3 \ldots u_n >= val_1, val_2, val_3 \ldots val_n$

if $u_1 > val_1$

→ True

if $u_1 < val_1$

→ False

if $u_1 == val_1$

it will compare for the val

if all the values of 1st collection is Equal to 2nd collection the result is True

Eg: 3 >= 4

→ False

7.8 >= 3.1

→ True

[4, 6, 5] >= [4, 6, 1]

→ True

(23, 45) >= (56, 67)

→ False

v) Assignment Operator:

It is an operator which will help the user to reduce the redueancy of a code by merging two variable as a single variable with the help of '=' operator.

a = a+b ⇒ a+ =b
a = a-b ⇒ a- =b
a = a* b ⇒ a* =b
a = a/b ⇒ a/ =b
a = a||b ⇒ a|| =b
a = a%.b ⇒ a% =b
a = a**b ⇒ a** =b
a = a|b ⇒ a| =b
a = a^b ⇒ a^ =b
a = a&b ⇒ a& =b
a = a<<n ⇒ a<< =n
a = a>>n ⇒ a>> =n

vi) Membership Operator:

It is a special operator which is use to check wheather the value is present inside the collection (or) not.

→ Membership operator got classified into twotypes.

1. IN Operator
2. NOT IN Operator

1. in Operator:

'in' operator will return the result as True the value is Present in side the collection.

Else it will return false.

Syntax used is,

var/val in collection

Eg: 1 in [1, 2, 3, 4]

>>> True

10 in 'hai'

>>> Error

'h' in 'hai'

>>> True

'ha' in 'hai'

>>> True

'hai' in [12, 45, 'hai', 89]

>>> True.

'a' in {'a' : 1}

>>> True

2. not in Operator:

'not in' is an operator which will return the result as True if the value is not present inside the collection.

Else it will return False.

Syntax used is:

var/val not in collection

Eg: 34 not in [12, 34, 67]

>>> False

3 not in {1, 2, 3}

>>> False

9 not in {3, 4}

>>> True

'a' not in {'a' : 1}

>>> False.

'ni' in 'india'

>>> False

vii) Identity Operator:

It is an operator which is use to check whether both the variables are pointing to same memory location (or) not

→ Identity Operator got classified into two types.

i) is operator
ii) is not operator

i) is operator:

It will return True if both the variables are pointing to the same memory location.

Syntax:

var1 is var2

Eg: a = 10
b = 20
a is b
>>> False

a is c
c = 10
a is c
>>> True

a = [1,2,3]
b = [1,2,3]
a is b
>>> False [because Deep copy is performi

ii) is not Operator:

It is an operator which will return the result as True if both the variables are not pointing the same memory location.

Syntax: var1 is not var2

Eg: 34 is not 34
>>> False

a = [1,2,3]
b = a.copy()
>>> a is not b
>>> True

a = [1,2,3]
b = [1,2,3]
a is not b
>>> True

Input statement :

Input is a function which is use to get the input from the user and the function useal is,

```
input()
```

→ The input taken by input fun will be in the form of string by default
→ To the get the required the data type we need to typecast taken by input() fun

Eg:
```
a = input('Enter the value :')
Print (a, type(a))
```

Print statement :

Print is use to display the output on the output screen the syntax of Print is

syntax : `Print ('v1, v2---vn , sep = ' ', End = '\n')`

→ If we don't specify the value of seperator and End by defaultly it is going to take space as seperator, new line as End

Eg:
```
Print (10,20)
Print ('hai')
```

O/P : 10 20
'hai'

→ whenever we want to chringe the default value of seperator and End we can make use of an syntax

Syntax:
```
End = 'new_value'
seperator = 'new_value'
```

Eg:
```
Print (10, 20, sep = '$', End = '@')
Print ('hai')
```

O/P : 10 $20@ hai

Write a Pgm to find the sum of two numbers.

```
a = 10
b = 20
Print (a+b)
```

o/p : 30

(or)

```
a = int (input ('Enter the value :'))
b = int (input ('Enter the value :'))
Print (a+b)
```

o/p : Enter the value : 100
Enter the value : 20
120

Write a Pgm to find the square of a given number.

```
a = int (input ('Enter the value :'))
Print (a**a)
```

(or)

```
sqr = a*a # a**2
Print (sqr)
```

(or)

```
n = 2
sqr = a**n
```

o/p : Enter the value = 2
4

Write a Pgm to find the sum, difference, product of two numbers.

```
n = int (input ('Enter the value :'))
q = int (input ('Enter the value :'))
p = n+q
p = n-q
p = n*q
print (n+q, n-q, n*q)
```

(or)

```
Print ('sum is:', a+b)
Print ('diff is :', a-b)
Print ('prod is :', a*b)
```

(or) Print(n+q, n-q, n*

o/p : Enther the value : 2
Enter the value : 3
5 -1 6

sum is : 5
sum is : -1
sum is : 6

Control statements:

These are the statements based on which the flow of Execution takes place.

(or)

control statements are the statement which will control the flow of Execution.

→ There are two types of control statemes.

1. Decisional statements
2. Looping statements.

1. Decisional statements:

These are the control statements which will control the flow of Execution based on some decision

→ Decisional statements are again classified into four type.

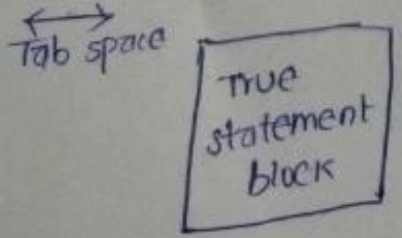
1. Simple if
2. if Else
3. Elif
4. Nested if

1. Simple if statements:

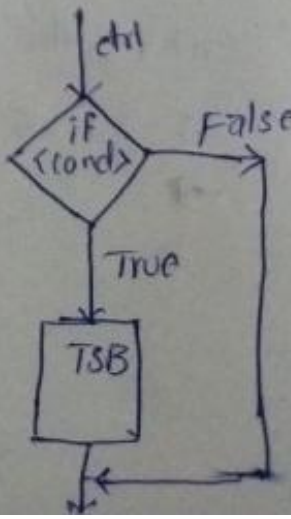
Simple if is used whenEver we have only single condition and set of statements for that particular condition.

→ Control will Execute the statement blocks if the condition is True Else it will ignore.

Syntax: if <condition>:



Flow Diagram:



Write a Prgm to check wheather the given number is Even or odd if it is Even Print the message by saying number is Even then print the number. Else print the number directly.

```
n=2
n = int(input('Enter the value :'))
if n%2==0:
    Print ('num is Even')

Print (n)
```

o/p : Enter the value : 10
num is Even
10

Enter the value : 7
7

Write the Prgm to check wheather the given string is having morethan 5 character or not, if it is True print a message as len is more then print the string Else print the string directly.

```
Str = 'Hello world'
if (len(Str) > 5):
    Print ('len is morethan string')
Print (str)
```

(or)

```
st = input ('Enter the value')
if (len(st) > 5):
    Print ('len is more)
Print (st)
```

o/p : len is morethan string
hello world

Write the Prgm to check whether the given character is lower case alphabet or not

```
ch = 'l'
if 'a' <= ch <= 'z':
    Print('lower case char')
```

'a' 97 → [Ord('a') → to find Ascii the value
65 'A' chr(65) → to find value of the Ascii]

O/P: Enter the value : l
lower case char

Write the program to check whether the character is special symbol is or not

```
ch = '$'
if not ('a' <= ch <= 'z' or 'A' <= ch <= 'Z' or '0' <= ch <= '9'):
    Print('special symbol')
```

O/P : special symbol

Note:

Ord():
It is the function which is use to get the Ascii value of specified character.

chr():
It is the function which is use to convert Ascii value in the from own character

if Else :

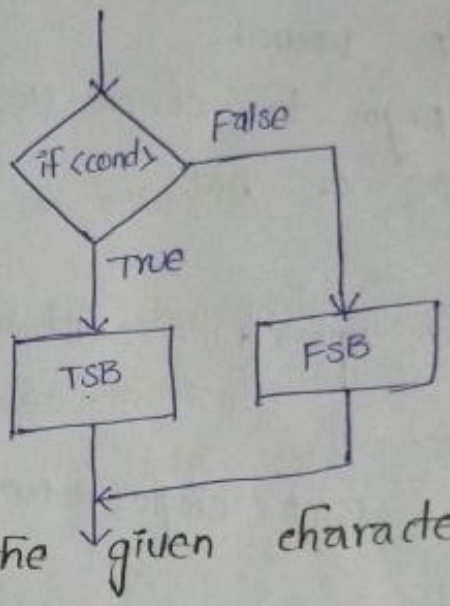
for single condition if we have two set of statement block will go if Else statement.

In this case control will Executes True statement block if the condition is True Else it's going to Execute false statement block

Syntax :

```
if <condition> :
    TSB
Else :
    FSB
```

Flow chart :



Write a Program to check weather the given character is alphabet or not

```
ch = 'a' / @
if 'a' <= ch <= 'z' or 'A' <= ch <= 'Z' :
    Print ('alpha')
Else :
    Print ('not an alpha')
```

Write a Prgm to check weather the given number is Even multiple of 3 or not if it is true then print the msg as number is Even and multiple 3 Else print invalid number.

```
n = int (input ('Enter the val :'))
if n%.2 == 0 and n%.3 == 0 :
    Print ('num is Even')
    Print ('num is Mul 3')
Else
    Print ('Invalid')
```

Write a Prgm to check weather the given character is vowel or not

```
ch = 'A'
if ch in 'AEIOU' 'aeiou':
    Print ('vowel')
Else:
    Print ('not a vowel')
```

```
Str = 'kemo'
for i in str:
    if i not in ('aeiou AEIOU'):
        Print (i)
```

o/p : vowel

Write a prgm to check weather the given value is single value data type or not

```
a = 2
if type(a) == int or type(a) == float or type(a) == complex
                    (or) type(a) == Bool :
    Print ('single value datatype')
Else:
    Print ('not a single value Datatype')

    (or)

a = 10
if type(a) in [int, float, complex, bool]:
    Print ('single value data type')
Else:
    Print ('collection')
```

o/p : int
single value data type

Write a Pgm to check weather the given x value is present inside the collection or not.

```
a = 10
b = [1, 2, 3, 10]                    ①
if a in b :
    Print ('the value is present inside the collection')
else :
    Print ('the value is not present inside the collection')
```

O/p : The value is present inside the collection.

Write a Pgm to check weather the given character uppercase alphabet or not if it is uppercase then print the msg a uppercase alphabet else convert it into uppercase then Print that value.

```
str = 'a'
if 'A' <= str <= 'Z' :
    Print ('uppercase alphabet')
else :
    Print (chr (ord ('a') - 32))
    Print (str)
```

O/p : Uppercase alphabet.

```
str = 'a'
if 'A' <= ch <= 'Z' :
    chr (ord ('a') - 32)
    Print (str)
if 'a'
```

U → L	L → U
A a	a A
65 97	97 65

65 + 32 = 97 — chr (ord ('A') + 32)

97 - 32 = 65 — chr (ord ('a') - 32)

Write a Pgm to check weather given string palindrom or not

```
str = 'hello'
if str[::-1] == str :
    Print ('palindrome')
else :
    Print ('not a palindrome')
```

(or)

```
str = 'gadag'
rev = str[::-1]
if str == rev :
    Print ('palindrome')
else :
    Print ('not a palindrome')
```

3. Elif statement :

Whenever we have multiple conditions and set of statement block for Each and Every condition we will make use of Elif statement.

→ if any of the condition is true control will Execute the set of statement block related to that condition. and it will come out

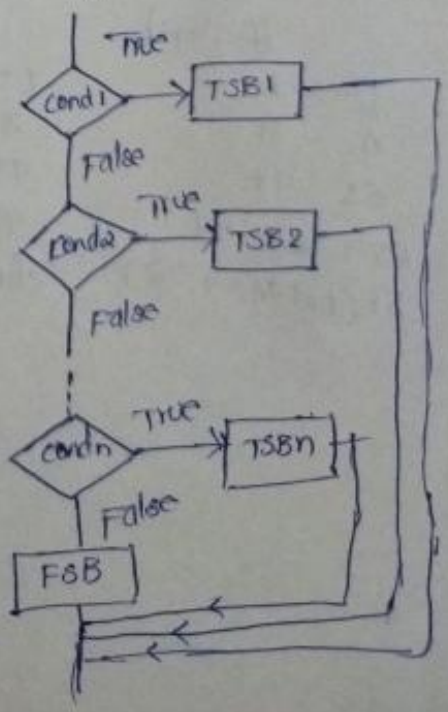
→ if none of the conditions are true it will Execute else statement

→ writing else statement is not mandatory

Syntax :

```
if <cond 1> :
    TSB 1
Elif <cond 2> :
    TSB 2
Elif <cond 3> :
    TSB 3
    :
Elif <condn> :
    TSB n
else :
    FSB
```

Flow diagram :



Write a Prgm to check weather the given character is uppercase (or) lowercase (or) number (or) special symbol

```
str = 'A' / '10' / '$' / 'H'
if 'A'<=str<='Z':
    Print ('upper case str')
elif 'a'<=str<='z':
    Print ('lower case str')
elif not (('A'<=str<='Z') or ('a'<=str<='z') or ('0'<=str<='9')):
    Print ('special symbol str')
elif '0'<=str<'9':
    Print ('number')
else:
    Print ('special symbol')
```

Write a Prgm to find the relationship between two numbers

```
a = 10
b = 30
if a==b:
    Print ('a is Equal to b')
elif a>b:
    Print ('a is greater than b')
else:
    Print ('a is less than b')
```

Write a Pgm to find the greater among three numbers

```
a = 10          (or)    a,b,c = 10, 30, 50
b = 30
c = 50
if a>b and a>c :
    Print ('a is greater than b and c')
elif b>~~c~~ :
    Print ('b is greater than a and c')
else: ~~e>a>b~~
    Print ('c is greater than a and b')
```

Write a Pgm to find the greater among five numbers.

```
a,b,c,d,e = 10,20,50,60,100
if a>b and a>c and a>d and a>e :
    Print('a is greater')
elif b>c and b>d and b>e:
    Print ('b is greater')
elif c>d and c>e:
    Print ('c is greater')
elif d>e:
    Print ('d is greater')
else:
    Print ('e is greater')
```

Write a Rgm to Predict the result of the student based on the percentage that she/he got

```
stu = 75 ,  (or)stu= int ( input ( 'enter stu percentage :'))
if  stu >= 85 :
    Print ('stu is Distiction')

elif
elif  stu >= 75 :
    Print ('stu is first class')

elif  stu >= 65 :
    Print ('stu is second class')

elif  stu >= 50 :
    Print ('stu is pass')

else :
    Print ('stu is fail')
```

(or)

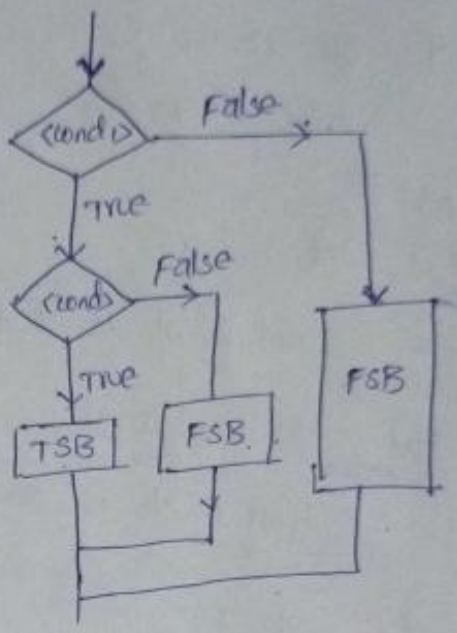
```
m = 99
if m>=80 :
   Print('FCD')
elif  60 <= m<= 79 :
   Print('first class')
elif  45 <= m< 59 :
    Print ('2nd class')
elif  35 <= m <= 44 :
    Print('just pass')
else :
```

4. nested if :

The finamion of writing an the if statement inside one more if statement is called as nested if statement

Syntax:

```
if <cond1> :
    if <cond> :
        TSB
    else:
        FSB
else:
    FSB
```



Write a Pgm to print the middle value present inside the list collection if the len of the collection is odd else print the message as len of the collection is Even. if it is Even

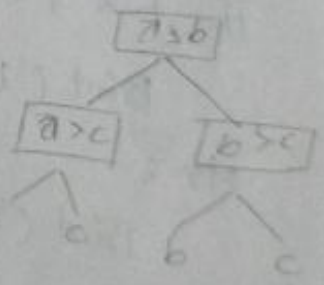
```
a=[10,20,30,40,50]
n=len(a)
if n%2!=0:
    if a[n//2]%2==0:
        Print(a[n//2])
    else:
        Print('middle val is odd')
else:
    Print('len is even')
```

Write a Pgm to check weather the user can login to instagram successfully by Entering the proper username and password

```
Oun = 'Pyspiders.com'
Opw = 'pysp'
un = input('Enter the un:')
pw = input('Enter the pw:')
if Oun == un:
    if Opw == pw:
        Print('Logged in')
    Else:
        Print('inproper pw')
Else:
    Print('invalid un')
```

Oun = Original user name
Opw = Original password

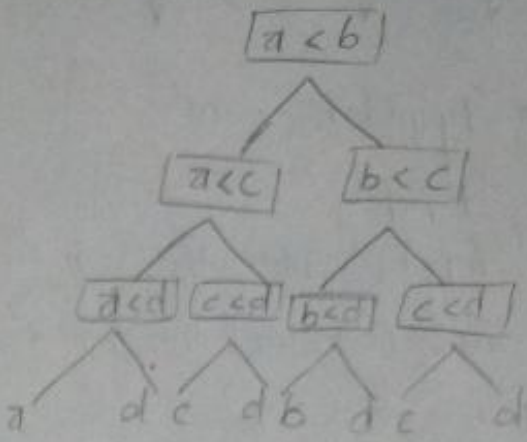
Write a Pgm to find the greater among three numbers with the help of nested if statement.




```
a, b, c = 10, 200, 30
if a>b
    if a>c:
        Print('a is greater')
    else:
        Print('c is greater')
Else:
    if b>c:
        Print('b is greater')
    Else:
        Print('c is greater')
```

```
a,b,c = 10,200,30
if a>b and a>c:
    if b>c:
        Print('b is greater')
    else:
        Print('c is greater')
else:
    Print('a is greater')
```

Write the Pgm to find smallest among four numbers.




```
a, b, c, d = 10, 20, 30, 50
if a<b:
    if a<c:
        if a<d:
            Print('a is smaller')
        else:
            Print('d is smaller')
    else:
        if c<d:
            Print('c is smaller')
        else:
            Print('d is smaller')
else:
    if b<c:
        if b<d:
            Print('b is smaller')
        else:
            Print('d is smaller')
    else:
        if c<d:
            Print('c is smaller')
        else:
            Print('d is smaller')
```

Write a Pgm to find the second greatest among four numbers

```
a, b, c, d = 10, 20, 30, 40
if a>b and a>c and a>d:
    if b>c and b>d:
        Print('b is 2nd g')
    Elif c>d:
        Print('c is 2nd g')
    Else:
        Print('d is 2nd g')
Elif: b>c and b>d:
    if a>c and a>d:
        Print('a is 2nd g')
    Elif c>d:
        Print('c is 2nd g')
    Else:
        Print('d is 2nd g')

Elif c>d:
    if a>b and a>d:
        Print('a is 2nd g')
    Elif b>d:
        Print('b is 2nd g')
    Else:
        Print('d is 2nd g')

Else:
    if a>b and a>c:
        Print('a is 2nd g')
    Elif b>c:
        Print('b is 2nd g')
```

Write a Pgm to find the second smallest among five numbers

```
a,b,c,d,e = 10, 20, 30, 40, 50
if a<b and a<c and a<d and a<e: → ①
    if b<c and b<d and b<e:
        Print('b is 2nd smallest')
    Elif c<d and c<e:
        Print('c is 2nd smallest')
    Elif d<e:
        Print('d is 2nd smallest')
    Else:
        Print('e is 2nd smallest')  ②
Elif b<c and b<d and b<e:
    if a<c and a<d and a<e:
        Print('a is 2nd smallest')
    Elif c<d and c<e:
        Print('c is 2nd smallest')
    Elif d<e:
        Print('d is 2nd smallest')
    Else:
        Print('e is 2nd smallest')
Elif c<d and c<e:
    if a<b and a<d and a<e:
        Print('a is 2nd smallest')
    Elif b<d and b<e:
        Print('b is 2nd smallest')
    Elif d<e:
        Print('d is 2nd smallest')
    else:
        Print('e is 2nd smallest')
```

```
Elif d<e :
    if a<b and a<c and a<e :
        Print('a is 2nd smallest')
    Elif b<c and b<e:
        Print('b is 2nd smallest')
    Elif c<e:
        Print('c is 2nd smallest')
    Else:
        Print('e is 2nd smallest')
Else:
    if a<b and a<c and a<d :
        Print('a is 2nd smallest')
    Elif b<c and b<d :
        Print('b is 2nd smallest')
    Elif c<d:
        Print('c is 2nd smallest')
    Else: Print('d is 2nd smallest')
```


Looping statements:

It is a type of control statements which is used to execute some set of instructions repetatily again and again until the given condition become false.

→ with the help of looping statements we can increase the efficience of the code by reducing code repetation problem.

→ In python loopings got classified into two types

1. While loop
2. for loop

1. While loop:

While loop is looping statement which is use to Execute some set of instructions Repetively again and again until the given condition become false.

→ In while loop two thing are mandatory

i) Intialization of looping variable

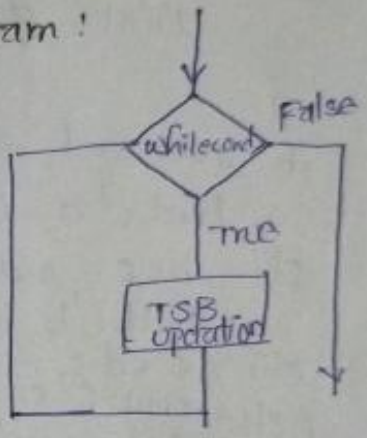
ii) updation of looping variable

if we don't intialize the looping var we will get an Error and if we don't update the var the loop will act as infinet loop by printing the same values

Syntax:

```
initalise
while <cond> :
    TSB
    updation
```

Flow diagram:



Eg: Write a Pgm to print 1 for 5 times.

```
i = 1
while i<=5:
    Print (1)
    i+=1
```

Write a Pgm to Print n - natural numbers

```
n = 10
while n<=10:
    Print (n+1)
    n+=1
```

```
i = 1
n = 5
while i<=n:
    Print (i)
    i+=1
```

(or)

```
n = 1
while n<=5:
    Print (n)
    n+=1
```

Write a Pgm to print all the Even numbers between the range m to n

```
m= int(input('Enter the value:'))
n = int(input('Enter the value:'))
while m<=n%2==0
    Print(n)
    m+=1
```

(or)

```
m=1
n=20
i=m
while i<n:
    if i%2==0:
        Print(i)
    i+=1
```

Write a Pgm to Print all numbers which are multiple of 5 between the range m to n

```
m=1  int(input('Enter the initial value:'))
n=50 int(input('Enter the final value:'))
i=m
while i<m:
    if i%5==0:
        print(i)
    i+=1
```

[1, 5, 2, 6, 200, 7]

Write a Pgm to print n^{th} multiplication table.

```
n = int(input('Enter the value:'))
i=1
while i<=10:
    Print("n*i") (or) Print(n,'*',i,'=',n*i)
    i+=1
```

Write a Pgm to print all special character present inside an input string.

```
st = input('Enter the string:')
i=0
while i<len(st):
    if not('a<=st[i]<='z' or 'A<=st[i]<='Z' or '0<=st[i]
                                            <='9')
        Print(st[i])
    i+=1
```

Write a Prgm to print all the characters present at Even index.

```
st = input('Enter the string:')
i = 0
while i<len(st):
    if i%2==0:
        Print(st[i])
    i+=1
```

```
st = input('Enter the string:')
i = 0
if i<len(st):
    Print st[::2]
```

(or)

if not(i%2==0): (or) it will give odd Even index

Write a Prgm to print all the Even numbers present inside the list collection

```
l = [1,2,3,4,5]
i = 0
while i<len(l):
    if l[i]%2==0:
        Print(l[i])
    i+=1
```

l[i] ⇒ 2-1.2 = 0

l = [1,2,3,4,5]
i = 0

(or)

write a Prgm to ~~Extract~~ all the uppercase characters present inside the an input string to and o/p string variable

```
st = input('Enter the string:')
out = ''  → we have to take empty default value string ('')
i=0
while i<len(st):
    if 'A'<=st[i]<='Z':
        out+=st[i]
    i+=1
Print(out)
```

→ output will be storing to another var by using concatenation

out = out + st[i]

Write a Pgm to Extract all the numbers (or) digits from the input string to and Output string.

```
st = input ('Enter the string :')
res = ''
i = 0
while i < len(st):
    if '0' <= st[i] <= '9':
        res = res + st[i]
    i += 1
print (res)
```

write a Pgm to Extract all the integer values present pre inside the given input in to and output string.

```
l = list (input ('Enter the list elements :'))
['a', ',', '@', '27', '5.0', '2+3j']
a = []
i = 0
while i < len(l):
    if type(a[i]) == int :
        a = a + [l[i]] (or) a += [l(i)]
    i += 1
Print(a)
```

Range function (range())

range is a function which is use to create the sequence of numbers between the specified limits.

→ Since, range function is not having any specific structure we need to typecast it to either list (or) tuple.

→ **syntax:** range (SV, EV±1, UP)

UP == +ve → EV+1
UP == -ve → EV-1

SV == 0 and UP == 1
range (EV±1)
UP == 1
range (SV, EV±1)

Eg: range (1, 10)

```
>>> range (1,10)
    list (range (1,10))
>>> [1, 2, 3, 4, 5, 6, 7, 8, 9]
    tuple (range (1,10))
>>> (1, 2, 3, 4, 5, 6, 7, 8, 9)
    list (range (10, 0, -1))
>>> [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
    list (range (1, 50, 5))
>>> [1, 6, 11, 16, 21, 26, 31, 36, 41, 46]
    list (range (15))
>>> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
    list (range (4, 2))
>>> []
```

Write a Pgm to count the number of spaces present in a given collection.

```
st = input ('Enter the string: ')
i = 0
c = 0
while i < len(st):
    if st[i] == ' ':
        c = c+1
    i += 1
Print ('total num of space present is: ', c)
```

for loop:

To overcome the Problem with while loop we are going to use for loop.

→ In for loop no need to initialize the looping variable as well as no need to update the looping variable.

→ In for loop we can access the values one-by-one directly

Syntax:

```
for var in collection
    TSB
```

(tab)

collection: string, list, tuple, set, dictionary, range()

These are the datatypes are given for the for loop

Eg: 1.
```
for i in 'hello'
    Print (i)
```
O/P: h, e, l, l, o

2.
```
for i in [1,2,3,4,5]:
    Print (i)
```
O/P: 1, 2, 3, 4, 5

3.
```
for i in (10,20,30):
    Print (i)
```
O/P: 10, 20, 30

4.
```
for i in {1,2,3,4}:
    Print (i)
```
O/P: 3, 2, 4, 1

5.
```
for i in {'a':1, 'b':2, 'c':3}
    Print (i)
```
O/P: a, b, c

6.
```
for i in range(1,10,2)
    Print (i)
```
O/P: 1, 3, 5, 7, 9

Write a Pgm to Extract all the string data Present inside an input list to output list.

```
st = [1, 2.3, 'Kavi', 'me', 2+3j, 'you', 'h']
out = []
for i in st:
    if type(i) == str:
        out += [i]
Print(out)
```

Write a Pgm to Print all the float values present at odd index in a given list

```
st = [1, 2.3, 'Kai', '5.0', 30, '0.0']
for i in range(0, len(st)):
    if i%2!=0 and type(st[i]) == float:
        Print(st[i])
```

Note:

In programing question if it consist of a word called range (or) position (or) Index we need to make use of 'range()' in for loop.

Write a Pgm to find the factorial of a given number

```
f = int(input('Enter the num:'))
for i in range(1, f):
    f = f*i
    Print(f*i)
```

o/p: fact = fact * i
= 1 * 1
= 1
i = 2
fact = 1 * 2
= 2
i = 3
fact = 2 * 3
= 6
fact = 6 * 4
= 24
fact = 24 * 5

```
n = 5
fact = 1
if n>=0:
    for i in range(1, n+1):
        fact = fact * i
    Print('fact')
else:
    Print('Enter tve num for n')
```

Write a Pgm to Extract all the Even numbers present at Even position in a given list.

```
l = [1, 21, 5, 24, 30, 6]
out = []
for i in range (0, len(l)):
    if i%2==0 and l[i]%2==0:
        out = out + [l[i]]

Print ('out')
```

Write a Pgm to check whether the given string is palidrome or not without using slicing.

```
str = input ('Enter the string')
rev = ''
for i in str:
    rev = i+rev
    if str == rev:
        Print ('it is a palidrome')

    Else:
        Print ('it is not a palidrome')
```

O/P:

tracing:

st = 'hai'

rev = ''

rev = 'h'+''
= 'h'

i = 'a'

rev = 'a'+'h'
= 'ah'

i = 'i'

rev = 'i'+'ah'
= 'iah'

WAP to Exact Extract all the special characters present at Even index.

```
str= input ('Enter the str:')
Out = ''
for i in range (0, len (str))
    if i%2==0 and not ('a'<= str[i]<='z' or 'A'<=str[i]<='Z'
                       or '0'<=str[i]<='9')
        Out = Out + i
Print (Out)
```

O/P: Enter the str : @ Rupa $ sree

WAP to convert all the lowercase alphabet present inside the an input string to uppercase character.

```
str = input ('Enter the str:')
Out = ''
for i in string
    if 'a'<= i <='z':
    out+chr(ord(str[i]-32)
    else:
        Out += i
Print (out)
```

Out += chr(ord(i)-32)

O/P : Enter the str : Rupa sree
RUPA SREE

Write a Pgm to replace all the spaces by _ (underscore)

```
str= input ('Enter the str:')
Out= ''
for i in str:
    if i == ' ':
        Out = Out + '_'
    else:
        res += i
Print (str)
```

O/P: Enter the str : hai hello how are yoo
hai_hello_how_are_you