* Industry coding python and R language. *

* PYTHON LANGUAGE *

⇒ uses of python :-

1) Web development (Django, Flask, web2py)
2) Machine learning : It is a subset of artifical Intelligence that can learn & performing the task with abstracting the data / models.
3) API's
4) Data Science
5) Data visualization [scipy, scikit-learn] → models
   Pytorch [weka, java, deeplearning4j, mallet].

API :- Application programming Interface. It is a set of predefined libraries that can be used to create software applications.
- [operators, conditionalset, loops, functions, array].

* Python :- It is a programming language and a Platform independent.

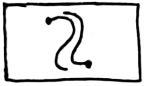
① Any hardware [processor] or software [operating system] in which we can remember the source code.

Processor :- controls & manages all the instructions in our code → order given to the processor by the user
Ex:- put a

* Founder of python :-

Developed by guido van rossum
first version Feb 20-1991
latest version oct 02 - 2023

Van rossum picked the name from the T.v show called monty python's flying circus the logo shows. two intertwined snakes.



* Features of python :-

There are varies reasons why python is growing popularly in programming community.

1. platform independent
2. portability
3. Open source software.
4. Interpreted language
5. simple
6. dynamically typed
7. object oreiented
8. Exception handling
9. large number of libraries

* Platform Independent :-

Any software (or) Hardware we can able to run the source code.

* Portability :-

The most contribution of the python over other languges in python easily we can move from one system to the another system anywhere and any time.

Note :- Write once run anywhere.

* Open source software :-

Python is a open source software anybody can freely download from the (website : WWW.python.org) to develop the programs

* Interpreted languages :- PVM also called as interpreted. It converts.

→ By the help of the python virtual machine it will convert the each bicode instructions into the machine understabindable instructions.

* Simple :- Python is a simple programming language that means more clearity and less stress on understanding the syntax of the programming language.

* dynamically typed :-

Here we need not to declare anything

C	Java	Python
int a ;	int a ;	a = 10 ;
int a = 10 ;	int a = 10 ;	

Note : Based upon the value interpreted can assign the type to the variable name at run time (dynamic memory allocation).

→ Both c language and Java statically typed (compile time)

Python and Java dynamically typed (run time)

* Object oriented :- Here we can build the program using classes and objects.

⇒ Principles :-

class, object, encapsulation, data abstraction, polymorphism, inheritance.

* Exception handling :-

It is an abnormal event that rises an Error during the execution of the program.

C++ : try, catch, throw

Java : try, catch, throw, throws, finally.

Python : try, except, else, finally.

* Large number of libraries :-

for devopling the web development.

C+ :- Django, web2py, flask

Java :- ~~scipy~~, ~~scikitlearn~~ Pandas, numpy.

Python :- scipy, scikitlearn

2D Visualisation :- Matplotlib, seabrn.

⇒ To check the version of the python software open comand prompt type.

python . -- version.

* There are three ways for executing python code :-

1) python's command line window.
2) IDLE graphical window (Integrated development environment)
3) command line window (notepad)

* First way name :-

Python's command line window

print ("ICFAI")

⇒ a = 10, b = 10
print (a + b).
30

⇒ a = 10, b = 20, c = a+b
print (c)
c = 30

print ("ictai")
ictai

int a = 10

Syntax Error : invalid syntax

a = 10
b = 20
print (a+b)
30.

file → new file

a = 4
b = 10
print (a+b)
save as
create a folder. In desktop give a name as one.py (or) two.py, etc.
save

O/p :- 14

C: \ user \ This pc > cd \

cd section : > python one.py
file name

output : 3

google colab
welcome to colabaratory
new notebook

Parameter	Java	Python
compilation — ①	compiled & compiled and interpreted lang	Interpret language prog. can run using a python Interpreter
Static / dynamic ②	Static :- int a	② - python - file name .py dynamically type a=1;
③ Machine learning	weka, mallet, deeplearning.	scipy, scikit learn protocol.
④ learning curve	complex learning	simple learning & easy to use
⑤ multiple Inheritance	doesn't supports directly	supports directly
⑥ curly braces vs Indentation	curly braces { }	Indentation four white spaces tnr : _ _ _ _ 1.
Architecture	JVM	PVM
String handling functions.	Java offers limited no of string handling functions.	lots of related String functions.

⇒ In Java :- The block of the scope starts with the opening braces

⇒ In python :- The block the scope starts with Induction (_ _ _ _ 1).

* Variables and Rules :-

Variable is a name given to the element to store the information

Rules for declearing the variables :

1) There should not be given keyword as a variable name.
2) There should not be given space in b/w the variables names.
3) cannot start with the digit.
4) can start with the underscore (_)
5) cannot start with the special symbols
6) can start with the either upper case or lower case letters.

Ex :- 1) a=10
print (a)
10

2) elif = 34
syntax error

3) ab = 20
print (ab)
20

4) a b = 20
Syntax error

5) 2a = 40
Syntax error

6) _2a = 50
print (_2a)
50

7) @ = 34
Syntax error

8) Aa = 45
print (Aa)
45.

⇒ Write a python program employee name, employee salary, employee gender?

Sol:-

```
name = "Lucky"
Salary = 45000.0
age = 20
gen = 'Female'
Print (name, salary, age, gen)
```

output :- Lucky 45000.0 20 F

Type function : { }

① Type function is a buil in function that is used to written the type of data stored in variable.

② python does not have a character data type, a single char is a simply a string.

③ Strings can be enclosed by either " " (or) ' '.

*⇒ Write a python program to print data type of the name, salary, age, gender?

Sol:-

```
name = "Harini"
sal = 45000.0
age = 20
gen = 'F'
print (type (name)) → string
print (type (sal)) → float
print (type (age)) → int
print (type (gen)) → string
```

*⇒ Write a python program to delete name variable?

Sol:-

```
del name
print (name)
```

→ 'name' is not defined.

⇒ Write a python program to declare a multiple Variables?

Sol:-

```
a = b = c = 100
print (a, b, c)
```

output:

100 100 100

Ex:-

→ a, b, c = 1, 2, 3
print (a, b, c)

O/P: 1 2 3

→ a, b, c = 100
print (a, b, c)

O/P: declaration variables does not match the assigned values.

Type error:- cannot unpack non-iterable int object.

⇒ Write a python program to import math module and perform square root operation?

Sol:
```
import math
print (math.sqrt(16))
```
↳ accessing.

O/P:- 4.0

⇒ Write a python program to perform factorial, ceiling value, floor value, pi value, power value using math module?

Sol: from math import factorial, ceil, floor, pi, pow

```
print (factorial(3))
print (ceil(3.4))
print (floor(3.4))
print (pi)
print (pow)(2, 3))
```

O/P:-
6
4
3
3.14
8.0

* Operators in Python :-

→ An operator is a symbol that performs an operation. An operator acts on some variables called operands.

① Arithematic operators :-

- +
- -
- *
- /
- %
- // - floor division
- ** - power

→ a = 30
b = 20

→ a+b
50

→ a-b
10

→ a*b
600

→ a/b
1.5

→ a%b
10

→ a//b
1

⇒ a = 2
b = 3

→ a**b
8

⇒ a = 15
b = 4

→ a/b
3.75

→ a//b
3.

② Comparision operators :-

It returns either True (or) False based on the condition.

<
>
<=
>=
==
!=

Ex:-

```
a = 2
b = 3
a > b
False
a < b
True
a != b
True
a == b
False
a <= b
True
a >= b
False.
```

③ logical operators :-

and
or
not

Ex:-

```
a = True
b = False
print (a and b)
False
print (a or b)
True
print (not a)
False
print (not b)
True.
```

* Assignment operators :-

Compact	Expansion
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 = 10
b = a
print (a)
10
print (b)
10
a = a + b
print (a)
20
a = a - b
print (a)
10
a = a / b
print (a)
1.0
a = a % b
print (a)
1.0
a = a // b
print (a)
0.0
a = a ** b
print (a)
0.0
```

* Membership operators :-

1) In and not In are the membership operators in python.

2) They are used to test whether a value or variable is found in a sequence data types (string, list, tuple, set and dictionary). " " [ ] ( ) { } { }

* In operator :-

→ True if value or variable is found in the sequence.

* NOT IN operator:

→ It returns true iff value or variable is not found in the sequence.

Ex :-

```
str = "Raju"
print ('R' in str)
True.
str = "Raju"
print ('R' not in str)
False.
```

```
a = 10
b = 10
c = 20
print (id(a)) → o/p : 140711122975448
print (id(b)) → o/p : 140711122975448
print (id(c)) → o/p : 140711122975488.
```

* Identity Operators :-

→ Python language offers some special types of operators like identity operator or membership operators

→ When it comes to the identity operator, is and is not are the identity operators in python. They are used to check if 2 values are located on the same part of the memory.

→ Is operator :-

→ It returns true if the operands are identical. (refer to the same object)

→ Is not :-

→ It returns true if the operands are not identical (do not refer to the same object).

Ex :-

```
a = 10
b = 10
c = 20
print ( c is 20)
print ( 10 is a) → o/p :- True.
print( 20 is b)
o/p: False
```

* data types in python:-

→ Data type represents sizes and different values that can be stored into a variable.

→ Data type hierarchy in python:

Sequence datatypes
1) range
2) string
3) list
4) tuple

maps : dictionary

sets : set, frozenset

Boolean : True, False

Number : int, float, complex.

* Array disadvantages:

⇒ 1) Array cannot grow dynamically. once we can declaring the array size it size is fixed.

2) Arrays can store homogenous elements (It can store single type of data)

3) Adding the elements at the end of the array is very easy. inserting or removing the elements from the middle of the array is very difficult.

⇒ 4) retriving the elements from the array is very easy after retriving the elements if you want to process them then there is no such methods are available.

⇒ Range datatype :-

→ The Range datatype represents a sequence of numbers. The numbers in the range are not modified. generally, range used for repeating a for loop for a specific no of times.

⇒ Write a python program to print 0 to 9 numbers using range function

sol:-
```
r = range(10)
for i in r:
    print(i)
```

o/p:-
```
0
1
2
3
4
5
6
7
8
9
```

⇒ write a python program to print 20 to 38 with step size is increased by 3. apply range function?

```
x = range(20, 40, 2)
for i in x:
    print(i)
```

O/P:-
```
20
22
24
26
28
30
32
34
36
38
```

* String data type:

→ String is a collection of characters or sequence of characters that can be enclosed with either ' ' (or) " " (or) Triple quotes (''' ''')(""" """)

Ex:
```
str = 'core python language'
print(str)
```
O/P: core python language.

```
str = """core python language"""
print(str)
```
O/P: core python language

```
str = """core\t python\n language"""
print(str)
```
O/P:
```
core        python
language
```

=) Write a python program to print a length of the string.

Ex:
```
str = "core java and python programming"
print(len(str))
```
O/P: 32

* Slicing the strings:-

→ A slice represents a part or pieace of a string the format of slicing is

Syntax ⇒ stringname[:startindex:stopindex:stepsize]

Note: If start and stop are not specified then slicing is done from 0 to n-1 elements. If stepsize is not written, then it is taken to be 1.

Quary:- 1) Access string from 0 to 8th element in steps of 1. (stringname = "core python programming")

Ex (-1)
```
str = "core python programming"
print(str[0:9:1])
```
O/P: core pyth

2)
```
str = "core python programming"
print([::-1])
```
O/P: gnimmargorp nohtyp eroc

Access the string 2 to ending.

```
Str = "core python programming"
print(str[::2])
```

O/p:- cr yhnpormig

Index number -4 to -2 stepsize 1.

```
print(str[-4:-2:1])
```

O/p: mi

Access from -6 to ending

```
print(str[-6::])
```

O/p: amming

* Write a py. prog to access the characters of a string using for loop.

Sol:

```
str = "core python programming language"
for i in str:
    print(i)
```

O/p:

```
c
o
r
e

p
y
t
h
o
n

p
r
o
g
r
a
m
m
i
n
g

l
a
n
g
u
a
g
e
```

reverse:- O/p:

```
e
g
a
u
g
n
a
l

g
n
i
m
m
a
r
g
o
r
p

n
o
h
t
y
p

e
r
o
c
```

* Repeating strings :-

The repeation operator is denoted by '*' symbol and is useful to repeat the string several times.

```
str = "core python programming language"
print (str * 3)
```

O/p:

core python programming language core python programming language core python programming language.

```
rcs = str[2:4:1] * 3
print(rcs)
```

O/p:

rc rc rc

* Concatenation of strings :-

→ we can use '+' operator on strings to attach a string at the end of the other string. This operator '+' is called addition operator when used on numbers. But, when used on strings is called concatenation operators.

* Write a py prog to print core py prog lang and core Java prog lang. using '+' concatenation operators.

So!:- str = "core python programming language"
str1 = "core Java programming language"
print (str + str1).

o/p: core python programming language core Java programming language.

* Removing spaces from a string:-

→ A space is also considered as a character inside a string sometimes, the unnecessary spaces in a string will lead to wrong results.

→
* strip method:- removes spaces from both sides of the strings. This method do not remove spaces which are in the middle of the string.

Ex:- str = " core python programming language ";
print (str) ⇒ # strip() method
res = str.strip()
print (res)
#length function

res1 = len(str)
print (res1)

o/p:
core python prog lang
core python prog lang

* counting sub strings in a string:-

→ The method count is avaible to count the numbers of a occureses of a sub~~count~~ strings in a main string.

Ex:-
→ str = "new delhi"
res = str.count('e')
print (res)

o/p: 2.

→ res = str.count('delhi')
print (res)

o/p: 1

→ res = str.count('e', 0, len(str))
print (res)

o/p: 2.

* Replacing a string with another string:-

→ The Replace method is used to replace a sub string in a string with another sub string.

Syntax:
string.replace (old, new)

Ex:- str = " core Java programming language "
res = str.replace ("Java", "python")
print (res)

o/p: core python programming language.

* Splitting the strings:

→ split method is used to break a strings into a pieces this pieces are returned as a list.

→ whenever the delimiter (,.etc) is formed the entire string is divided into small pieces called tokens. that tokens are stored into the list.

Syntax:-

```
#str.split("delimiter")
```

Ex:-

```
str = "core,python,programming,language"
res = str.split(",")
print(res)
print(str[0])
print(res[0])
print(str[1])
print(res[1])
```

O/P:

```
['core','python','programming','language']
c
core
o
python
```

* write a py. prog to convert upper case letters to the lower case letters and vice versa.

→
```
str = "core python programming"
res = str.upper()
print(res)
```

O/P: CORE PYTHON PROGRAMMING

→
```
str = "Core Python Programming"
res = str.swapcase()
print(res)
```

O/P:- CORE · PYTHON PROGRAMMING.

→
```
res = str.title()
print(res)
```

O/P: Core Python Programming.

→
```
res = str.lower()
print(res)
```

O/P: core python prog

* Join method:-

→ When a group of strings are given, it is possible to join them and make a single string. For this purpose we can use join method.

Syntax:

```
separator.join(str)
```

Note: Where the separator represents the character to be used b/w the strings in the o/p. Here 'str' represents a list. [ ].

Ex:-

```
str = ["one", "two", "three"]
sep = "_"                      sep = ":"
res = sep.join(str)            o/p: one:two:three
print(res)
print(str[0])
print(res[0])
print(str[1])
print(res[1])
```

o/p:-

```
one-two-three
one
o
one
n
```

* Checking starting and ending of a string:-

→ The startswith method is useful to know wheather a string is starting with a substring or not.

Syntax:

```
str.startswith(substring)
```

Note: When the substring is found in the main string 'str' this methods returns true if sub string is not found than it is false.

Ex:-

```
str = "core python book"
res = str.startswith("core")
print(res)
```

o/p: True

→

```
res = str.endswith("core")
print(res)
```

o/p: True

* Formatting the strings:-

→ Formatting a string means presenting a string in a clearly understandable manner. The format method is used to format the strings.

Syntax:

```
'format string with replacement fields'.
    format(values)
```

Note: We should 1st understand the meaning of the 1st attribute that can be i.e format string with replacement fields. The replacement fields are represented with '{}' that contains indexs. This indexs represent the order of the values.

Ex:-

→
```
id = 100
name = "raju"
sal = 10000
res = '{}, {}, {}'.format(id, name, sal)
print(res)
```

O/P:- 100, raju, 10000

→
```
res = '{} {} {}'.format(id, name, sal)
print(res)
```

O/P:- 100 raju 10000

→
```
res = "{}\n{}\n{}".format(id, name, sal)
print(res)
```

O/P:-
```
100
raju
10000
```

→
```
res = "name = {1}\nid = {0}\nsal = {2}".format
(id, name, sal)
print(res)
```

O/P:-
```
id = 100
name = raju
sal = 10000.
```

* List :-

→ list items are changable and allow duplicate elements.

→ list items are indexed elements, 1st element has index [0], 2nd element has index [0], etc.

* write a py. prog to create a list ?

→
```
l = ["pavan", "raju", "Kiran"]
print(l)
```

O/P:- ["pavan", "raju", "Kiran"]

→
```
l = ["pavan", "raju", "Kiran", True, 2, 2.5]
print(l)
```

O/P:- ['pavan', 'raju', 'Kiran', True, 2, 2.5]

* Write a py. prog to print the length of the list.

→
```
l = ["pavan", "Raju", "Kiran", True, 2, 2.5]
print(len(l))
```

O/P:- 6

* Write a py. prog to print the type of the particular list.

```
l = ["Pavan", "raju", "Kiran", True, 2, 2.5]
print(type(l))
```

O/P:- class 'list'

* write a py. prog to create interger value and Boolean value & string value using list.

Sol:-
```
l = [True, 2, "raju"]
print(l)
```

O/p:- [True, 2, 'raju']

* Write a py. prog to impliment list constructor.

Sol:-
```
l = list(("ravi", "Komal", 1))
print(l)
print(type(l))
```

O/p:- ['ravi', 'komal', 1]
<class 'list'>

* Write a py. prog to print the second item of the list using indexing.

```
l = list(("ravi", "Komal", 1))
print(l[1])
print(l[0])
print(l[2])
```

O/p:-
Komal
ravi
1.

* Write a py. prog to print all the elements in the forward and reverse direction. using Solicing method.

Solp forward:-
```
l = list(("ravi", "Komal", 1))
print(l[::])
```

O/p:- ['ravi', 'Komal', 1].

reverse:-
```
print(l[::-1])
```

O/p: [1, 'komal', 'ravi']

* write a py. prog to change the value of a specific item refer to the index number.

Sol:-

```
1 = list [( " ravi ", "komal", 1, "rama", 5])
print 1[2] = " siva"
print (1)
```

o/p:-

['ravi', 'komal', 'siva', 'rama', 5]

* Change a range of item values :-

→ To change the value of items with in a specific range, define a list with the new values and refer to the range of index numbers where you want to insert the new values.

ex:-

```
1 = list [("ravi ", "komal", 1, "rama", 5])
1[1:3] = "siva", "raju"
print (1)
```

o/p:-

['Ravi', 'siva', 'Raju', 'Rama', 5]

* Insert items :

→ To insert a new list item, without replacing any of the existing values we can use the insert method.

→ The insert method inserts an item at the specified index.

ex:-

```
1 = list [("ravi", " komal", 1 , "rama", 5])
1. insert ( 3, " raju")
print (1)
```

o/p:-

['ravi', 'komal', 1, 'raju', 'rama', 5]

* Append items :-

To add an item to the end of the list use the append method.

ex:-

```
1 = list ([("ravi", "komal", 1 , "rama", 5])
1. append ( "ictai")
print (1)
```

o/p:-

['ravi', 'komal', 1, 'raju', 'rama', 5, 'ictai']

* Extend list :-

→ To append elements another list to the current list use the extend method.

Ex:-
```
1 = ["one", "two", "three"]
11 = ["four", "five"]
1.extend(11)
print(1)
```

o/p:- ['one', 'two', 'three', 'four', 'five']

* Remove method :-

→ Remove method removes the specified item

Note: If more than one item with the specified value remove method removes the 1st occurances.

Ex:-
```
1 = ["one", "two", "three"]
1.remove("three")
print(1)
```

o/p:- ['one', 'two']

* Remove specified index :-

→ pop method removes the specified index

Note: If you do not specify the index value pop method removes the last item.

Ex:-
```
1.pop()
print(1)
```

* Clear method :-

→ It clears the entire method., it empties the entire bracket.

Ex:-
```
1.clear()
print(1)
```

* looping using list comprehension :-

→ write a py. prog to print all the elements using for loop ?

```
1 = ["one", "two", "three", "four", "five"]
for i in 1:
    print(i)
```

op:- one
two
three

→ for i in l[::-1]:
    print(i)

o/p:- five
four
three
two
one.

* list comphension:-

list comphension offers the "shortest syntax" for looping through list.

ex:-
[print(i) for i in l]

o/p:- one
two
three
four
five.

* write a py. prog to print the sequence numbers from 0 to 9, using range function and apply list comphension technique.

ex:- #l = ["one", "two", "three", "four", "five"]
[print(i) for i in range(10)]

o/p:- 0 2 4 6 8
1 3 5 7 9

→ l = ["one", "two", "three", "four", "five"]
new = ['hello' for i in l]
print(new)

o/p:- ['hello', 'hello', 'hello', 'hello', 'hello']

~~upper method~~:-

~~-> str.upper~~

→ Apply list comphensim to convert 'l.c' to upper course letters using upper() method.

l = ["one", "two", "three", "four", "five"]
new = [i.upper() for i in l]
print(new)

o/p:- ['ONE', 'TWO', 'THREE', 'FOUR', 'FIVE']

* Write a py. prog to sorting elements using sort method.

l = ["99", "144", "0", "69"]
l.sort()
print(l)

o/p:- [0, 2, 69, 99, 144]

* join the two lists using concatenation method.

Ex:-
```
l = ["one", "two", "three", "four", "five"]
l1 = ["five", "six", "seven"]
print (l + l1)
```

o/p:-
['one', 'two', 'three', 'four', 'five', 'six', 'seven']

* copy a list :-

→ make a copy of a list with copy method.

Ex:-
```
new = l.copy()
print(new)
```

o/p:- ['one', 'two', 'three', 'four', 'five']

* write a py. prog to print the type of tuple.

```
t = ("pavan", 1, 2, "raju", 23)
print (type(t))
```

o/p:
< class 'tuple' >

* Create tuple with one item:-

→ To create a tuple with only one item, we have to add a comma, after the item. otherwise it will not recognise it as a tuple.

Ex:
```
t = ("pavan",)
print(type(t))
```

o/p:
< class 'tuple' >

* Tuple constructor:

It is also possible to use the tuple constructor to make a tuple.

```
t = tuple(("pavan", 1, 23.5, 5, 6))
print(t)
```

o/p: ('pavan', 1, 23.5, 5, 6)

* Access tuple items :-

→ You can access tuple items by refering to the index number, in side of the square brackets.

```
t = tuple (("pavan", 1, 23.5, 5, 6))
print (t[1])
```

o/p: 1.

* Write a py. prog to print the elements in a tuple forward direction and reverse direction using +ve index and -ve index apply slicing operator.

```
t = tuple (("pavan", 1, 23.5, 5, 6))
print (t[::1])
print (t[::-1])
```

o/p: ('pavan', 1, 23.5, 5, 6)

(6, 5, 23.5, 1, 'pavan')

* Range of Indexing:

→ you can specify a range of indexing by specifying where to start and where to end the range

```
t = tuple (("pavan", 1, 23.5, 5, 6))
print (t[1:4])
```

o/p: (1, 23.5, 5)

* update values:

→ Tuples are unchangable, meaning that you cannot change, add or remove items once the tuple is created

Note: you can convert the tuple into a list, change the list, and convert the list back into a tuple.

Ex:-

```
t = ("pavan", 1, 23.5, 5, 6)
t1 = list (t)
t1[1] = "raju"
res = tuple (t1)
print (res)
```

o/p: ('pavan', 'raju', 23.5, 5, 6)

* write a py. prog to append the element at the end of the list.

```
t = ("pavan", 1, 23.5, 5, 6)
t1 = list (t)
t1.append ("sai")
res = tuple (t1)
print (res)
```

o/p: ('pavan', 1, 23.5, 5, 6, 'sai')

* Write a py. prog to implement the remove method

```
t1.remove("pavan")
```

o/p:
(1, 23.5, 5, 6)

* write a py. prog to print the all element in forward and reverse direction using for loop.

forward:

```
t = ("pavan", 1, 23.5, 5, 6)
for i in t:
    print(i)
```

o/p:
pavan
1
23.5
5
6

reverse:

```
for i in t[::-1]:
```

6
5
23.5
1
pavan.

* write a py. prog to join to tuples using concatenation operator.

```
t = ("pavan", 1, 23.5, 5, 6)
t1 = ("kumar",)
print(t + t1)
```

o/p:
('pavan', 1, 23.5, 5, 6, 'kumar')

* Repeatation of the tuple :-

→ Write a python program to print the tuple elements in two time with using p Repeatation operation.

```
t = ("one", "two", "three")
print(t*2)
```

o/p:-
('one', 'two', 'three', 'one', 'two', 'three')

→ print(t[1]*2)

o/p:
('one', 'two', 'two', 'three')

* count method :-

→ It returns no. of times a sepecified value occured in a tuple...

```
t = ("one", "two", "three", True, 1, False, 0)
res = t.count("three")   → res = t.count(1)
print(res).
```

o/p:
2
2

* Index method :-

→ It returns the tuple for a specified value and returns the position of where it was found.

→
```
res = t.index("three")
print(res)
```

O/P: 2

* Dictonaries :- { }

→ Dictonaries are used to store the data values in key and value pairs.

→ Dictonary is a collection, which is changable and do not allow duplicate elements.

→ The advantage of the dictonary to search the particular item very fast when compare to the list.

* write a py. prog to implement dictonaries concept with three key and value pairs.

Sol:-
```
d = {"name": "ramu", "age": 56, "sal": 67000}
print(d)
```

O/P: {'name': 'ramu', 'age': 56, 'sal': 67000}

* write a py. prog to print salary value on the display.

→
```
print(d["sal"])
```

O/P:
67000

* Duplicates not allowed :-

→ Dictonaries can not have two items with the same key

→ duplicate values will over ride existing values.

→
```
d = {"name": "ramu", "age": 56, "sal": 67000,
     "age": 45}
print(d)
```

O/P: {'name': 'ramu', 'age': 45, 'sal': 67000}

* write a py. prog to apply length function.

→
```
print(len(d))
```

O/P: 3

* write a py. prog to store string value, integer value, boolean value and list in a dictionary.

```
d = {"name": "ramu", "age": 56, "bool": True, "li":
                                ["one", "two", "three"]}
print(d)
```

o/p:

```
{'name': 'ramu', 'age': 56, 'bool': True,
                'li': ['one', 'two', 'three']}
```

→ print(type(d))

o/p: <class 'dict'>

* Accessing or referring the ~~elements~~ items :-

→ you can access the items of a dictonary by refering to its key name inside of the square brackets '[]'.

→ print(d["name"])

o/p: Ramu.

Note: There is also a method called Get that will give you the same result.

→ print(d.get("name"))

o/p: ramu.

* Get Keys :-

→ The keys method will return a list of all the keys in the dictonary

→ print(d.keys())

o/p: dict keys {('name', 'age', 'bool', 'li')}

* Get values :-

→ The values method will return a list of all values in the dictonary.

→ print(d.values())

o/p: dict value([('ramu', 56, True, ['one', 'two', 'three']])

* Get items:-

→ The items method will return each item in a dictonary, as tuples in a list.

→ print

o/p:- dict_items([('name','ramu'),('age',56),('bool',True),('li','one','two','three')])

* loop using dictonary :-

→ Write a py prog to implement for loop

```
for i in d:
→ print(i)
```

o/p:
name
age
bool
li

```
→ for i in d.items():
      print(i)
```

o/p:
('name', 'ramu')
('age', 56)
('bool', True)
('li', ('one', 'two', 'three'))

* change the values :-

→ you can change the values of a specific item by refering to its key name.

ex:
```
d={"name":"pavan","age":56,"sal":78000}
d["name"]="raju"
print(d)
```

o/p:
{'name':'raju','age':56,'sal':78000}

* update a dictonary :

→ update method will update the dictonary with the items from the given argument.

Note: The argument must be dictonary.

ex:
```
d.update({"age":67})
print(d)
```

o/p: {'name':'raju','age':67,'sal':78000}

* Adding items :-

→ Adding an item to the dictonary is done by using a new index key and assigning a value to it.

ex:
```
d["state"]="ts"
print(d)
```

o/p: {'name':'raju','age':67,'sal':78000,'state':'ts'}

* Removing the items :-

→ The pop method removes the item with the specified key name

Ex:
```
d.pop("age")
print(d)
```

O/P: {'name': 'raju', 'sal': 78000}

* write a py. prog to copy the elements from one dictonary to the another dictonary using copy method.

Ex:
```
d = {"name": "raju", "age": 56, "sal": 78000}
d1 = d.copy()
print(d1)
print(d)
```

O/P: {'name': 'raju', 'age': 56, 'sal': 78000}

→
```
d.clear()
print(d)
```

O/P:- { }

* Sets *

⇒ A set can be defined as a collection which is unordered and unindexed. Sets are written with "{ }".

* unordered :-

Items in a set do not have a defined order.

* unindexed :-

→ Items in a set do not have a defined index. it prints the value in a random order.

→ once a set is created, you can not change its items, but you can add new items and removing existing items.

→ By refering to an index, you cannot access items in a set. since sets are unordered the items has no index

→ By using for loop we can access the set items.

* Write a py. prog to create 5 elements in a set

```
S = {"one", "two", "three", "four", "five"}
print(s)
```

o/p:- {'three', 'four', 'one', 'five', 'two'}

* Duplicates not allowed :-

→ Sets cannot have 2 items with the same value

Note :- The values True and 1 are considered as same and False and 0 are considered as same.

```
→ s = {"one", "two", "three", "four", "five", True, 1}
    print(s)
```

o/p:- {'one', 'two', True, 'five', 'four'}

```
→ print(len(s))
```

o/p:- 6

```
→ print(type(s))
```

o/p: <class 'set'>

* set constructor :-

→ It is possible to use the set constructor to make a set

```
→ s = (("one", "two", "three", "four", "five", True, 1)
    print(s)
```

o/p:- " "

* Access or retreving the items :

→ you can not access items in a set by referring to an idex or a key.

→ But you can use the set items using a for loop.

ex:-

```
s = set (("one", "two", "three", "four", "five", True, 1))
for i in s:
    print(i).
```

o/p:- True

* Change items :-

→ Once a set is created, you cannot change its items but you can add new items.

→ To add one item to a set use the add method.

Syntax :-

⇒
```
s.add(3)
print(s).
```

o/p:- ('one', 'thrice', '3', 'Four', 'two', the')

* Add sets :-

⇒ To add items from another set into a current set use the update method.

⇒
```
s = set(("one", "two", "three", "four", "five", True, 1))
s1 = (("six", "seven"))
s.update(s1)
print(s)
```

o/p:
```
{True, 'two', 'five', 'one', 'three', 'seven', 'six', 'four'}
```

* Remove the items :-

→ To Remove an item in a set use the remove method.

ex:-
```
s.remove("one")
print(s)
```

o/p:
```
{'two', True, 'five', 'three', 'seven', 'six', 'four'}.
```

→
```
s.clear()
print(s)
```

o/p:- set().

Write a py. prog to print the all the set elements using for loop.

```
Set = {"one", "two", "three", "four"}
for i in set:
    print(i)
```

o/p:
```
four
three
one
two.
```

* Join two sets :-

→ There are several ways to join two or more sets in python.

→ you can use the union method that returns a set containing all items from the both ~~sets~~ sets.

Note: union method will exclued all duplicate elements.

⇒ set1 = {"one", "two", "three", "four"}
set2 = {"four", "five", "six", "seven"}

$s_3 = s_1.union(s_2)$
print(s_3)

O/p: {'five', 'six', 'seven', 'one', 'four', 'three', 'two'}

* Keep only the duplicates :-

⇒ Intersection method - will return a new set, that contains the items that are present in both the sets.

⇒ $s_3 = s_1.intersection(s_2)$
print(s_3)

O/p:- {'four'}

* Keep all, but not the duplicates :-

→ Symmetric ~~method~~ difference method will return a new set, that contains only the elements that are not present in both the sets.

Ex:-

$s_3 = s_1.symmetric\_difference(s_2)$
print(s_3)

O/p: {'five', 'six', 'one', 'two', 'three', 'seven'}

* difference method :-

→ Return a set that contains the items that only exist in set 'x', and not in set 'y'.

Ex:

$s_3 = s_2.difference(s_1)$
print(s_3)

O/p:- {'six', 'seven', 'five'}

* Input statement :-

→ By the help of the input function we can read the data from the keyboard.

* write a py prog to read a string value from the ~~keyword~~ board. using input function.

```
name = input("enter name")
print("entered name", name)
```

o/p: Enter name ictai

* write a py. prog to read IFHE on the display.

```
college = input ("Enter college name")
print ("college name =", college)
```

o/p: Enter college name ifhe
college name = ifhe

* write a py. prog to read two integer numbers from the keyboard and perform sum of 2 numbers

```
one = int(input ("enter first value"))
two = int(input ("enter second value"))
res = one + two
print (res)
```

o/p: Enter fist value 3
enter ~~first~~ second value 4.
7

without int:
Enter first value 3
" second " 4
34

⇒
```
one = float (input ("enter first value"))
two = float (input ("enter ~~the~~ second value"))
res = one + two
print (res)
```

o/p: Enter first value 2.3
" second 3.2
5.5

→ Write a py. prog to calculate sum of two numbers using eval().

```
one = eval (input ("enter first number"))
two = eval (input ("enter second number"))
res = one + two
print (res)
print (type (one))
print (type (two))
```

o/p:- first no :- 2
second no :- 3
2+3 = 5
<class 'int'>

Without eval()

```
one = (input ("enter first no"))
two = (input ("enter second no"))
res = one + two
print (res)
print (type (one))
print (type (two))
```

o/p :- first no : 2
second no : 4
→ 24.

* frozenset( ) :-

→ Freeze the list, and make it unchangeable the frozenset() function returns an unchangable frozenset object [which is like a set obj. only unchangable].

→ Write a py. prog to implement frozenset() function.

```
l = ["one", "two", "three"]
res = frozenset (l).
res[1] = "siva"
Print (res)
```

∴ set is unindexed list is indexed.

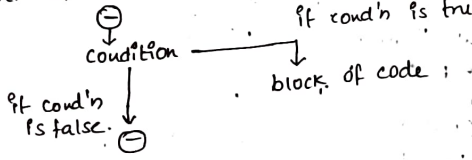
o/p:- res[1] = "siva"
TypeError: 'frozenset' object does not support item assignment.

* Conditional statement in python :-

→ Based on the conditions we can control the flow of our program and also execute different types of codes snippets or code blocks based on the decision (user defined) (1) if (2) if-else (3) if elif else

* If-statement :-

The if statement is used to test a particular condition & if the cond'n is true, it executes a block of code known as if-block.



→ write a py. prog to find given no is even or odd?

```
num = 10
if num % 2 == 0:
    print ("even")
```

with runtime

```
read the element dynamically !
num = int (input ("enter number"))
if num % 2 == 0:
    print ("even")
```

O/P :- enter num : 44
even

→ write a py. prog to print the largest of the three numbers ? (use if).

```
a = 10          a = int(input ("enter first number"))
b = 20          b = int (input ("enter second number"))
c = 30          c = int (input ("  "  third  "  "))
```

① if a>b and a>c :

```
    print (a)
② if b>a and b>c :
    print (b)
③ if c>a and c>b :
    print (c)
```

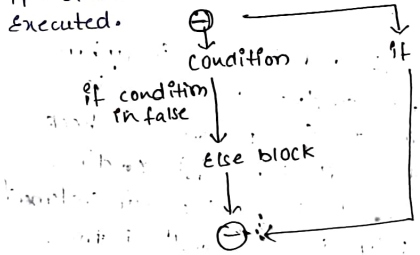
```
cd \
cd secl
python three.py
30
```

dynamical
python three.py
enter first number 20
" second " 100
" third " 30

O/P :- 100.

* if-else :-

The if-else statement provides an else block combined with the if statement which is executed the false case of the cond'n. If the cond'n is true, then the if-block is executed otherwise, the else-block is executed.



→ Write a py. prog to check whether a person is eligible to vote (or) not ?

```
age = int (input ("enter age"))
if age > 18
print ("eligible to vote").
```

```
else
    print ("not eligible to vote")
```

O/p: ① enter age 11
not eligible to vote

② python three.py
enter age 21
eligible to vote

→ prog to check wheath a number is even or odd.

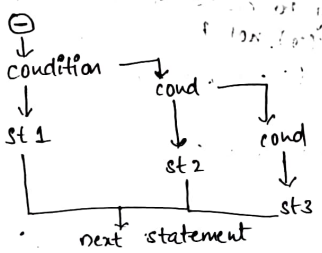
```
num = int(input ("enter a number"))
if num % == 0
    print ("even")
else:
    print ("odd")
```

O/p: python three.py
enter a number
odd

```
python
enter a number 2
even
```

* elif statement :-

elif stands for else if and is used in python prog to test multiple conditions. It is written following an if statement in python to check an alternate cond'n if the first cond'n is false the code block under the elif statement will be executed only if its cond'n is true.



→ write a py. prog if num==10: print ("given no is 10")

```
num = int (input ("enter number"))
if num == 10:
    print ("given number is equal to 10")
elif num == 50:
    print ("given number is equal to 50")
elif num == 100:
    print ("given number is equal to 100")
else:
    print ("given number is not equal to 10,50,100")
```

Output :-
1, enter no. 101
given no is not equal to 10, 50, 100
2, enter no. 10
given no is equal to 10
3, enter no. 50
given no is equal to 50
4, enter no. 100
given no is equal to 100

⇒ 1; if num > 0 : print ("positive number")
2, elif num < 0 : print ("negative number")
3) else : print ("zero").

```
num = int (input ("enter number"))
    if num > 0:
    print ("positive number");
    elif num < 0:
        print ("negative number")
    else:
        print ("zero")
```

O/P:- enter number 3
positive number
enter number -2
Negative number
enter number
zero

* loops in python:-

→ loops are used to execute the block of statement untill the particular cond'n is satisfied.

1. while loop:

Syntax:- while (test expression):
body of the statements are executed

Note: first test exp. is evaluated if is true body of the statements are executed.

→ Write a py. prog to print 1 to 10 numbers using while loop.

```
i = 1
while (i<=10):
    i = i+1
```

O/P:-
```
Python three.py
1
2
3
4
5
6
7
8
9
10
```

* while with else :-

→ code to be executed when the cond'n is true is intended under the while block. the 'else' keyword, followed by another colon(:), starts the else block, " this block will execute once the while cond'n become false".

```
i = 1
while (i<=10)
    i = i+1
else: print ("icfai")
```

O/P:
```
1
2
3
4
5
6
7
8
9
10
icfai
```

```
i = 11
while (i<=10)
    print (i)
    i = i+1
else:
    print ("icfai")
```

O/P: "icfai"

* for loop :-

```
for variablename in collection:
    print (variablename)
```

collection: group of elements.

Note: if one or more elements are available in the collection it returns true otherwise it returns false

Ex:-
```
l = ["one", "two", "three"]
for i in l:
    print (i)
```

→ Write a py. prog to print list, string, set, tuple and dictinaries using for loop.

```
Str = "core python"
for i in str:
    print(i)
```

O/P:
```
c
o
r
e

p
y
t
h
o
n
```

#list
```
l = ["one", "two", "three"]
for i in l
    print (i)
```

O/P:
```
one
Two
Three
```

tuple

```
t = ("four", "five")
for i in t:
    print(i)
```

o/p:
four
five

set

```
se = {"seven", "eight", "nine"}
for i in se
    print(i)
```

o/p:
eight
seven
nine

dic

```
d = {"name": "ram", "age": 45}
for i in d :
    print(i)
```

Keys oriented

```
for i in d.keys:
    print(i)        → name
                      age

for i in d.values:
    print(i)        → ram
                      45

for i in d.items
    print(i)        → name : ram
                      age : 45
```

⇒ Write a py. prog to print all the list elements using for loop (apply else block).

→ With Elements

```
l = ["one", "two"]
for i in l:
    print(i)
else:
    print("ictai")
```

o/p:
one
two
ictai

No Elements

```
l = []
for i in l:
    print(i)
else:
    print("ictai")
```

o/p:- ictai

* Loop control statements :-

1) Continue statement
2) break statement
3) pass statement

1) continue :-

It is used inside of a loop to repeat the next iteration of the loop. If continue is executed subsequent statement in the loop are not executed and control of execution goes back to the next repetition of the loop.

→ write a py. prog to print a sequence of numbers 0 to 9 using range function, next apply (continue statement)

```
→ for i in range(10):        (0,1,2,3,4)
      if i == 5;
          continue  → again ctrl jump to the next repetition after 5
```

o/p:-
0 6
1 7
2 8
3 9
4

6, 7, 8, 9

6 == 5 ✓, 7, 8, 9

2) Break :-

→ Break statement in python terminates the loop immediately and the control of program moves to the next statement following the loop.

```
→ for i in range(10)
      if i == 5;
          break
      print(i)
print("ictai")
```

o/p:-
0
1
2
3
4
"ictai"

→ for i in range(10)
    if i == 5;
        break
    print(i)
---- print("ictai")

O/P: 0 ictai 1 ictai 2 ictai 3 ictai 4 ictai

Indentation :- four white spaces

→ Pass statement :-

→ In python programming the pass statement is a null (nothing) statement which can be used as a place holder (sample code) for future code.

Note :- Suppose we have a loop or a function that is not implemented yet, but we want to implement it in the future In such case, we can use the pass statement.

ex :- i = 1
    if i < 10:
    print(i)

→ if print not there indented error.

O/P: 1

→ Write a py. prog to implement pass statement

→ i = 1
    if i < 10:
        pass

O/P :- nothing

→ i = 1
    if i < 10:
    # write in future (or) write code later
    Indentation error.

→ Write a py. prog to implement pass st with user defined function. (with pass).

→ def myfunction():
        pass

O/P :- nothing
No error

To implement pass st with classes

class Example :

O/P : Indentation Error

class Example :
    pass

O/P : Nothing
No Errors.

* Switch Statement : (Selection control structure)

To implement match / case statements :-

Case 0 : print("sunday")
Case 1 : print("monday")
Case 2 : print("Tuesday")
Case 3 : print("wednesday")

→ num = int(input("enter a number"))
    match num :
        case 0 :
            print("sunday")
        case 1 :
            print("Monday")
        case 2 :
            print("Tuesday")
        case 3 :
            print("wednesday")

O/P :-
Python - seven.py
Enter a number : 3
"Wednesday"

⇒ using eval() function:-

```
num = eval(input("enter a num"))
    match num:
        case 0:
        print("sunday")
```

O/p: enter a number : 0
" sunday".

* User defined function:-

Advantages:-

1, we can divide the large tasks into small tasks.
2, No need to declare same type of variables or function as multiple times.
3, reusability
4) def :- "function represents a group of statements that perform a task".

task represents calculation or processing the data that generate the reports or results.

```
def myfunction():
```
def stands for "define"

Ex:-
```
a=10
b=20
print(a+b)
# 30.
```

* Creating a function:-

→ In a python a function is defined using the def keyword:

→ write a py. prog to print ongole on the display /using function name: city)

```
def city():
    print("ongole")
def city():
    print("ongole")
```

O/p: nothing to be printed.

* Calling a function:-

→ To call a function use the function name followed by paranthesis.

```
def city():
    print("ongole")
city() # calling fun
```

O/p: ongole.

Note:- Whenever we can call the calling function control jumps from calling fun to called function & execute the body of the statements.

→ Write a py. prog to print ictai on the display (func name: college)

```
def college():
    print("ictai")
college()
```

O/p:- ictai

To calculate add'n of two numbers using (fun : sum)

```
def sum():
    a=10
    b=20
    print(a+b)
sum()
```

O/P:- 30

dynamic :-

```
def sum():
a=int(input("enter first number"))
b=int(input("enter second number"))
print(a+b)
sum()
```

O/P:- enter first number : 20
" second " : 30
50

INP

* Local variables :-

→ When we declare variables inside a function there variables will have a local scope (within the function) we cannot access them outside the function.

→ Write a py. prog to implement local variable concept ?

→ void sum():

```
    a=10 # local
    b=20 # local
    print(a+b)
sum()
print(a)
```

O/P: a is undefined

* Global variables :-

→ A variable declared outside of the function or global scope is known as global variable that a global variable can be accessed inside or outside of the function.

To implement global variables concept :-

```
a=10
b=20
def sum():
    print(a+b)
sum()
print(a)
print(a+b)
```

O/P:- 30

* Arguments :-

→ Information can be passed into function as arguments

→ Argument are specified after the function name, inside the paranthesis you can add as many argument as you want, just seperate them with a comma.

→ To create a function with one Argument (fun name : person)

```
def person(name): # parameter
    print(name)
person("pavan") # argument
```

O/P:- Pavan

Two create two arguments :-

```
def person(name, age): # parameter
    print(name)
person("pavan", age) # arg
```

O/P:- pavan

→ print(name, age)
person("pavan", 40, 10000)

O/P: error

Imp
* Parameter vs Arguments :-

From func's perspective :-

- A parameter is the variable listed inside the parameter in the function definition

Ex:- def person(name, age, sal):

→ An Argument is the value that is sent to the function when it is called.

Ex:- person("ram", 20, 100000)

Imp
* Python function with Arbitary Arguments :-

Sometimes we do not know in advance the no. of arguments that will be passed into a function. To Handle this kind of situation ; we can use Arbitary arg in Python.

→ Arbitary argument allows us to pass a varying number of values during a function call.

→ we use an asterik (*) before the parameter name to denote the kind of Argument.

→ Write a py. prog to print all the values using subscript (apply arbitary arguments)

```
def myfunction(*names):
    print(names[0])
    print(names[1])
    print(names[2])
myfunction("pavan", "raju", "komal")
myfunction("fbs", "ictai", "ffhe")
myfunction("hyd", "fst", "school")
```

O/P:- All are printed.

⇒ Write a py. prog to print all the values using for loop (apply arbitary arg)

```
def myfunction(*names):
    for i in names:
        print(i)
myfunction("pavan", "raju", "Harini")
" " ("fbs", "ictai", "ffhe")
" " ("hyd", "fst", "school")
" " ('one', 'two', 'three')
```

O/P:- All the values are printed.

→ To find the sum of all numbers!

```
→ def sum(*values)
    def count(*values): # called function
        sum = 0
        for i in values:
            sum = sum+i
        print(sum)
    count(1,2,3) # calling function
```

```
# sum = 0
# i = 1
→ # 0+1 = 1 (sum)
→ # i=2
  # 1+2 = 3 (sum)
→ # i=3
  # 3+3 = 6 (sum)
```

Olp:- 6.

* Multiplication:-

```
sum = 1
for i in values:
    sum = sum+i
    print(sum)
```

* Key word arguments :-

→ you can also sent arguments with the key = value syntax.

* write a py. prog to implement key and value pairs using keyword arguments.

```
def person(name1, name2, name3):
    print(name1, name2, name3)
person(name1 = "raju", name2 = "ictai",
                       name3 = "ibs")
```

o/p:

raju ictai ibs.

* write a py. prog to access all the key and value pairs using arbitary arguments.

```
def person(**names):
    print(names)
person(name1 = "raju", name2 = "ictai",
                       name3 = "ibs")
```

o/p:

{'name1': 'raju', 'name2': 'ictai', 'name3': 'ibs'}

* write a py. prog to access all the keys and value pairs using for loop. (apply arbitary arguments)

```
def person(**names):
    for i in names:
        print(i)
person(name1 = "raju", name2 = "ictai",
                       name3 = "ibs")
```

o/p:-

name1
name2
name3

⇒

```
for i in names.values():
    print(i)
```

o/p:

raju
ictai
ibs

⇒

```
for i in names.items():
    print(i)
```

o/p:

('name1', 'raju')
('name2', 'ictai')
('name3', 'ibs')

=> for i in names.keys():
    print(i)

o/p:
name 1
name 2
name 3

* default parameter value :-

→ If we call the function without arguments, it uses the default value

* write a py. prog to implement default value Arguments

```
def myfunction(country="india"):
    print(country)
myfunction()
myfunction("australia")
myfunction("england")
myfunction()
```

o/p:
india
Australia
England
india

* Passing a list as an argument :

```
def myfunction(l):
    for i in l:
        print(i)
li = ["one", "two", "three"]
myfunction(li)
```

o/p: one
Two
Three

* Return values :-

=> A function return a value use the return statement or keyword

* write a py. prog to calculate the addition of two numbers with return type (function name sum)

```
→ def sum():
    a=10
    b=20
    return a+b
res = sum()
print(res)
```

o/p :- 30

Without res :-
→ print(sum())
o/p :- 30

→ write a py. prog to perform addition of two numbers (without return type and without parameters)

```
def sum():
    a = 10
    b = 20
    print(a+b)
sum()
```

o/p :- 30

→ write a py. prog to perform addition of two numbers without return and with parameters.

```
def sum(a,b):
    print(a+b)
sum(1,2)
```

o/p: 3

→ write a py. prog to perform addition of two numbers with return type and without parameters.

```
def sum():
    a=10
    b=20
    return a+b
res = sum()
print(res)
```

o/p:- 30

→ write a py. prog to perform addition of two numbers with return and with parameters.

```
def sum(a,b):
    return a+b
print(sum(10,20))
```

o/p:- 30

→ write a py. prog to calculate square value using Parameters & return type.

```
def square(x):
    return x*x
print(square(10))
```

o/p:- 100