1. Basic Python Programming Constructs:-

Monty Python family circus (1969)

↓

1991

→ Guido Van Rossum

→ version - 3.7.1 [python - IDLE]

→ Time complexicity is very less for python code when compared to java.

Way all are predefined methods.

→ Remembering all these methods is very difficult

Control flow statements:- (10)

conditional statements:
1. if
2. if-else
3. if-elif-else

Looping statements:
1. for
2. while
3. for-else
4. while-else

unconditional statements (or) branching:
1. break
2. continue
3. pass

→ There is no "switch" statement in python.

Operations in Python:-

→ An operator is a some graphical symbol used to perform some mathematical and logical operations.

→ An operator is also statement.

1) Arithmetic
2) Relational
3) Logical
4) Compound assignment operators
5) Memship Operations
6) short hand if else statement:

1. Arithmetic Operators:

+ → concatenate symbol used with strings.
+, - → unary (or) Binary

* → Repeatation operator used in strings.

% → Java & python allows Integers and floating point values (C - only integer values.)

/ → By default it is float division in python. 10/3 = 3.3

// → floor division 10//3 = 3

** → exponent. - Right → left associativity.

→ Left → Right Associativity.

Precedence:-

① ** - highest priority. (R→L)

② → /, %, //, * (L→R)

③ → +, - (L→R)

Ex-1: 2 ** 2 ** 3 (8)

= 2 ** 8

= 256

→ 3 ** 0.5 → (square root)

→ 3 ** 0.25 → (4th root)

Ex-2: 5 + 8 / 6 ** 2 ** 2 (3, 4)

→ 5 + 8 / 3 ** 4

→ 5 + 8 / 81

→ 5 + 0.09

→ 5.09

Ex-3: 15 / 3 // 4 * 2

→ 5 // 4 * 2

→ 1 * 2

→ 2

→ In python strings are represented in single quotes (or) double quotes.

→ There is no character in python

Ex. 3 * 'IT'
o/p: ITITIT

→ 12 + 18 % 3 // 4 - 16 ** 2 ** 1

= 12 + 18 % 3 // 4 - 16 ** 2

= 12 + 18 % 3 // 4 - 256

= 12 + 0 // 4 - 256

= 12 + 0 - 256

= 12 - 256

= -244

→ 19 * 5 / 3 ** 2 + 4 - 8 // 6 % 2.

= 19 * 5 / 9 + 4 - 8 // 6 % 2

= 95 / 9 + 4 - 8 // 6 % 2

= 10.56 + 4 - 8 // 6 % 2

= 10.56 + 4 - 1 % 2

= 10.56 + 4 - 1

= 14.56 - 1

= 13.56

24

How to compile and run python code?

file Edit shell Debug Options window Help

Options - configure IDLE - size - code context

Help - turtle - already pre-defined code is available.

file - new file - Then type python code.

Press f5 key and save the program.

→ There is no predefined datatypes in python.
→ All are objects.
→ Python is pure OOP language.
→ No seperate memory bytes.
→ c:\users\admin>d:
→ D:\> python u.py
Srgec.

→ Variable:-

→ Variable is a storage location to store some value.
→ These storage locations are identified by identifiers.

Rules of Identifiers:

1) an identifier begins with an alphabet
2) It does not contains spaces and special symbols. it may contains underscores.
3) doesn't start with digit.
4) keyword are not used as identifiers.

Initialization:

Literal: Literal is a constant value used to store in a variable.

Ex: 1) a=12
op: 12

2) a$=5
syntax error

3) a=b=c=5
>>>a
5
>>>b
5
>>>c
5

4) >>> a,b,c=3,4,6.
>>>a
3
>>>b
4
>>>c
6

5) a=3; b=4; c=6
>>>a
3
>>>b
4
>>>c
6

"alias" is the powerful future in python.

```
>>> a=5
>>> b=5
>>> c=5
```

→ a='it'
→ a="it"
→ a="'it'"
→ a="" it" ""

} all are valid.

```
>>>a*10
'itititititititititit'
>>> 10*"_gec-"
'gec gec gec gec gec gec gec gec gec gec'
```

```
>>> a,b=5,6
>>>a
 5
>>>b
 6
>>> a,b=b,a
 >>>a
 6
>>>b
5
```

→ How to swap given two numbers in python.

```
>>> a,b=3,'it'
 >>>a
 3
>>>b
'it'
```

Literals in Python:

Literal is a constant value assigned to a variable.

Types of Literals:-

1) Integer Literals
2) Floating Literals
3) String Literals
4) Boolean Literals
5) Special Literals.

1) Integer Literals:

Integer means it doesnot contain precision part.

Ex:. > 3496
> 1234
> 987

Binary literals:

1) > a=0b111
7
> a=0b1010
10
> a=0b112
Syntax Error: invalid Syntax.

Octal literals:

1) > a=0o23 (→ 2 zeros)
>a
19
2) >>>a= 023
Syntax Error
3) >>> a = 0o29
syntax error:

Hexadecimal literals:

```
>>> a = 0xface
>>>a
64206
>>> a=0x10
>>> a
16
>>>0xabcd
43981
```

Tuple is immutable datatype in python

1) >a = 12,3,4,5
   >a
   (12,3,4,5)

2) >a = 3,'it',33.4,7
   >a
   (3,'it',33.4,7)

Floating Point literals:

Ex:1) >>> a = 3.123
   >>> a
   3.123

2) >a = 0.8
   >a
   0.8

3) >a = .8
   >a
   0.8

4) >a = 4.
   >a
   4.0

5) >a = 3e-3
   >a
   0.003

6) >a = 3E-3
   >a
   0.003

7) >a = 3e3
   >a
   3000.0

3) String Literals:-

These are enclosed inbetween single quote or double quote.

```
Ex: > a='welcome'
    >str = "WELCOME"
    >a
    'welcome'
   > str
   'WELCOME'
```

```
> '''this
is a python
program'''
'this \n is a python\n program'
```

a) Unicode Literals: Preceeds with U

These are definitely 4 digit hexadecimal number.

```
Ex: >a= u'\u0c05'
   >a
    'అ'
   >a*10
   'అఅఅఅఅఅఅఅఅఅ'
   > a= u'\u0c06'
   >a   ↳ upper case / Lower case
   'ఆ'
```

b) Raw Literals: Preceeds with R

```
s= R'welcome\nto'
print(s)
```

o/p: welcome\nto.

c) format Literals: preceeds with f

```
Ex:  a=5
     b= 6
     s= f'(a)+(b) = (a+b)'
     print(s)
```

'(a) + (b) = (a+b)'

o/p: 5+6 = 11.

4) Boolean Literals:

These are always True or false.

a = True
b = false

(only capital no small letter)

a = 1
b = 0.

Ex: 1 == True 0 == false
    o/p: True o/p: True.

→ Special Literals:

5) a) 'None' Literals:

a = None

→ without initial value variables are invalid in python.

b) Complex Literals:

Ex: > a = 3+4j
(3 → real value, 4j → imaginary value)

> a
(3+4j)

Addition of two complex numbers:

> a = 3+4j
> b = 5+6j
> a+b
~~8+10j~~ ~~-9+38j~~ 8+10j

> a * b
(-9+38j)

> a.real
3.0

> a.imag
4.0

> a = complex(5,6)
> a
(5+6j)

>>> a = 3.4+5.6j
>>> b = 8.7 + 4.5j
>>> a+b
(12.1 + 10.1j)

→ How to represent comment lines in python...

1) single line comment line.

i) a = 5 # it is a integeer value.

ii) a = 10 ''' 2) Multi-line comment lines:

a is integeer type

'''

Keywords: These are reserved words.

Keywords are basic building blocks of programming language

import keyword!

>import keyword

> keyword.kwlist

['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', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

> len(keyword.kwlist)

35

> keyword.iskeyword('if')

True

Mathematical functions in python!

import Math!

>import math

> dir(math)

syntax:

dir([object]) → list of strings

>help (keyword)

#Shows the commands with explanation.

> math.factorial(4)

24

2) Relational Operators: There is no precedence and associativity.

<
<=
>
>=
==
!=

>> 3<4

True.

>> 3>4

false

>>> 3<10>5

True

>>> 3<10 and 10>5

True.

>>> 12>4<20

True.

>>> 4==4.0

True.

>>> 4 == True

false

>>> 1 == True 'except 1 everything is false'

True.

>>> 0 == false

True

>>> 1 != false

True

>>> 1 != True

false.

>>> 4 == false

false

8/2/24

```
>>> 4<'a'
 error
>>>"a"<"b"
True.
>>> 'abc' > 'bcd'
false
>>> 'aaa'=='aaa'
True.
```

3) Logical operators: or short circuit operators!

i) and } → short circuit operators.
ii) or } → Binary operators.
iii) not → unary operator only.

"In python logical operators never returns a boolean value"

```
>>> a,b,c=5,6,7
>>> a and b and c
7
```

'not' operators returns boolean value.

```
→ a=0; b=4
>>> a and b
 0
→ a=5; b=6
 >>> a and b
  6
→ a=5; b=0
>>> a and b
 0.
>>> a,b,c=3,0,4
 >>> a and b and c
 0
```

'and' operator always searches for '0' and short circuits remaining one.

```
>>> a=0; b=10; c=5; d=9
>>> a and b and c and d
 0
```

short circuit.

I/O statements in Python:

output statements:

print() is an output statement in python.

Syntax:

print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

sep: seperator; ' ': space; file: module; sys: console related; stdout: standard output

Ex-1:

```
a = 10
b = 20
print(a,b)
```

o/p: 10 20 (space)

Ex-2:

```
a=10
b=20
c=30
d=40
print(a,b,c,d,sep='*')
```

o/p: 10*20*30*40

Ex-3:

```
a=10
b=20
c=30
d=40
print(a,b,c,d,sep='+',end='IT')
```

o/p: 10+20+30+40IT

Ex-4:

```
a=10
b=20
print('addition of',a,'and',b,'is',a+b);
```

o/p: addition of 10 and 20 is 30.

Using formats:

1)
```
a=10
b=20
#addition of 10 and 20 is 30
print ('addition of %d and %d is %d:' %(a,b,a+b))
```

o/p: addition of 10 and 20 is 30:

2) print ('addition of {} and {} is {}:'.format(a,b,a+b))

3) print ('addition of {0} and {1} is {2}:'.format(a,b,a+b))

4) print ('addition of {2} and {1} is {0}:'.format(a+b,b,a))

5) print (f'addition of {a} and {b} is {a+b}')

6) print ('{0:.d}'.format(a))

o/p: 10

7) print ('{0:10d}'.format(a))

o/p: ________10
(8 spaces)

8) print ('{0:010d}'.format(a))

o/p: 0000000010

9) print ('{0:b}'.format(a))

o/p: 1010.

10) print ('{0:0}'.format(a))

o/p: 12

11) print ('{0:0x}'.format(a))

o/p: a

12) print ('{0:-10d}'.format(a))

o/p: 10________
(8 spaces)

→ a = 1035353
b = 20 thousand seperated

print('{:,}'.format(a))

o/p: 1,035,353

→ a = 2.33336

print('{:.2f}'.format(a))

o/p: 2.33

→ a = 2.33336 21

b = 20

print('{:*<5}'.format(a))

21***

→ a = 21

b = 20

print('{:*>5}'.format(a))

o/p: ***21

print('{:*^11}'.format(a))

o/p: *****21******

print('{:*^5}'.format(a))

o/p: **21**

print('{:$^10}'.format(a))

o/p: $$$$21$$$$

>>> a = 2

>>> print('{:*^5}'.format(a))

2.

9/2/24

Write a python program to perform addition of given two numbers.

1)
```
a=int(input('enter first number:'))
b=int(input("enter second number:"))
print("Addition is",a+b)
```

o/p: enter first number:4
enter second number:5
Addition is 9

2)
```
a,b=int(input('enter first number')),int(input('enter second number'))
print("Addition is",a+b)
```

o/p:

3)
```
print("enter two values")
a,b=int(input()),int(input())
print("Addition is",a+b)
```

o/p:

4)
```
print
a,b=input().split(',')
print("Addition is",int(a)+int(b))
```

o/p:
3,4
Addition is 34

o/p: addition is 7

5)
```
print("enter two values")
a,b=map(int,input().split(','))
print("Addition is",a+b)
```

o/p: 4,5
Addition is 9

4

5**8**2**1||4||2-6.

5*8**2||4||2-6

5*64||4||2-6

320||4||2-6

80||2-6

= 40-6

= 34

or!

>>> a=0; b=10; c=5; d=9

>>> a or b or c or d

10

>>> a=5; b=6

>>> a or b

5

>>> a=1; b=2; c=3

>>> a or b and c (short circuits)

1

>>> a and b or c

2

not: It is a unary operator.

It is used with single variable and symbol is 'not'.

>>> a=0
>>> not a
True

>>> a=1
>>> not a
false

>>> a=5
>>> not a

false

4) Membership Operators: These are also statements not symbol.

1) in
2) not in

in:

It returns 'True' whenever a value is present in a sequence, otherwise it returns false.

ex:

```
>>> 'e' in 'welcome'
True
>>> 'z' in 'welcome'
False.
```

strings, List, Tuple, Dictionary } sequence (indicates set of elements)

>> 'in' operator with 'range()' function.

Syntax: range (start, stop, step)

1) range (stop) (by default start value is '0')
2) range (start, stop)
3) range (start, stop, step)

"start always less than stop"

for 3 parameters only positive numbers for 3 parameters

```
>>> 10 in range(20)
```
→ 0 to 19 values

True

```
>>> 20 in range (20)
False
>>> 0 in range (20)
True
>>> 10 in range (3,20)
```
→ 3 19

True

```
>>> 10 in range (15,20)
```
→ 15 to 19

False

```
>>> 10 in range(1,20,2)
                  ↳1,3,5,7,9 ... 19
false
>>>
```

not in: [opposite of in]

```
>>> 10 not in range (1,20,2)
True
```

Control flow statements in Python: (10)

conditional (or) selection statements:

a) if - one way selection statement

b) if-else: - two way selection statement

c) if-elif-else Multi-way selection statement

→ default identation is 4 spaces, line starting spaces are called "indentation"

if syntax: if condition:
    statements

ex:

```
if 10 in range(20):
    print('valid')
print('gec')
```

o/p: valid
gec

How to find even|odd integer using python.

```
a,b = int(input()), int(input())
if a%2==0:
    print('even')
if a%2==1:
    print('odd')
if b%2==0:
    print('even')
if b%2==1:
    print('odd')
```

o/p: 3 4
odd
even.

if-else: syntax: if condition:

EX:

statements

else:

statements.

```
a=int(input())
if a%2==0:
    print(even)
else:
    print(odd)
```

o/p: 5

odd

LAB Program:

2) There is a JAR full of candies for sale

Program:

```
N=10
k=int(input("Enter number of candies"))
if k in range(1,6):
    print('NUMBER OF CANDIES SOLD:',k)
    print('NUMBER OF CANDIES AVAILABLE :',N-k)
else:
    print('INVALID INPUT')
    print('Given candies:',k)
```

o/p:

c) if-elif-else:

Syntax:

```
if  condition1:
      | statements
elif  condition2:
      | statements
elif  condition3:
      | statements
.....
.....
.....
else:
      | statements
```

LAB PROGRAM:

3)

```
n=int(input("enter weight"))
if n==0:
    print('Time estimated: 0 minutes')
elif n in range(1,2001):
    print('Time estimated: 25 minutes')
elif n in range(2001,4001):
    print('Time estimated: 35 minutes')
elif n in range(4001,7001):
    print('Time estimated: 45 minutes')
elif n>7000:
    print('OVER LOADED')
else:
    print('Invalid Input')
```

Output:

enter weight : 2005

Time estimated: 35 minutes.

Looping (or) Iterative staments:

1) for loop: definitely required membership operators.

Syntax:

1) for with sequences:

```
for var in seq:
    statements
```

Ex:

```
for i in 'welcome':
    print(i)
```

o/p:
w
e
l
c
o
m
e

```
print(i,end=' ')
```

o/p: w e l c o m e

```
print(i,end='*')
```

o/p:
w*e*l*c*o*m*e*

* → write a python program to count no.of vowels in a given string.

Program:

```
v='aeiou'
c=0
s=input('Enter a string')
for i in s:  #welcome
    if i in v:
        c=c+1
print("vowels count is:",c)
```

o/p: vowels count is:3

→ write a python to reverse a given string.

```
s=input("enter string")
r=''
for i in s:
    r=i+r
print(r)
```

o/p: enter string: welcome
emoclew

```
print('Reverse of a string is:',r)
```

o/p: Reverse of a string is: emoclew

2) for with range function:

Syntax:

1) for var in range(stop):
       statements
2) for var in range(start,stop):
       statements
3) for var in range(start,stop,step):
       statements

→ There is no infinite for loop in python due to range() function.

ex:

```
for i in range(5):
    print(i)
```

o/p: 0
1
2
3
4

2) for i in range(0):
    print(i)

o/p: empty output

3) for i in range(-4):
    print(i)

o/p: empty output.

>>> for i in range(1,5):
    print(i)

o/p: 1
2
3
4

>>> for i in range(4,5):
    print(i)

o/p: 4

>>> for i in range(4,4):
    print(i)

o/p: whenever start = stop then blank output:

>>> for i in range(10,2):
    print(i)

o/p: Blank output.

>>> for i in range(1,5,1):
    print(i)

o/p: 1
2
3
4

>>> for i in range(1,5,3):
    print(i, end=' ')

o/p: 1 4

>>> for i in range(1,1,1):
    print(i)

o/p: Blank o/p.

>>> for i in range(10,2,2):
    print(i)

o/p: Blank o/p.

>>> for i in range(10,2,-2):
    print(i)

o/p: 10
8
6
4

>>> for i in range(1,-10,-2):

o/p: 1
-1
-3
-5
-7
-9

12/2/24

→ 1. write a python program to find factorial of a given number without using pre-defined function.

```
n=int(input('enter integer'))
f=1
for i in range(1,n+1):
    f=f*i
print('factorial is:',f)
```

(or)

```
n=int(input('enter integer'))
f=1
for i in range(n,0,-1)
    f=f*i
print('factorial is:',f)
```

To handle with more no. of inputs

2. write a pp to display even series upto given integer n.

```
n=int(input('enter integer'))
for i in range(1,n+1):
    if(i%2==0):
        print(i)
```

```
for i in range(2,n+1,2):
    print(i)
```

3. Write a PP to display given number is prime or not.

```
n=int(input('enter value'))
f=1
for i in range(2,n+1):
    if(n%i==0):
        f=0
        break
if(f==0):
    print('not a prime')
else:
    print('prime')
```

```
c=0
for i in range(1,n+1):
    if n%i==0:
        c=c+1
if c==2:
    print('prime')
else:
    print('not a prime')
```

→ primepy is a predefined function in python.
but we have to install this using pip command.

4. write a python program to display prime series upto given integer 'n'.

```
n=int(input('enter a value'))
c=0
for i in range(1,n+1):
        if n%i==0
            c=c+1
if c==2:
        print(prime)
else:
        print(Not prime)
```

5. write a python program to display prime numbers upto the given integer n.

```
n=int(input("enter an integer:"))
c=0
for k in range(1,n+1):
    c=0
    for i in range(1,k+1):
        if k%i==0:
            c=c+1
    if c==2:
        print(i,end=' ')
```

```
n=int(input())
c=1, i=1
while
```

LAB PROGRAM!

```
n=int(input("enter an integer"))
c=0;s=0
for k in range(1,n+1):
    c=0
    for i in range(1,k+1):
        if k%i==0:
            c=c+1
    if (c!=2):
        s=s+k
print("sum of Non-primes is:{0}".format(s))
```

14/2/

While Loop: There is a chance of infinite loops.

Syntax: While condition:
    statements

Ex: 1. write a pp to find reverse of a given number.

```
n=int(input("enter an integer"))
s=0
while(n>0):
    s=s*10+n%10
    n=n//10
print("Reverse is:%d "%(s));
```

n=n/10
OVERFLOW ERROR:
cannot convert float infinity to integer.

2. To find out given number is strong or not.

```
import math as m
from math import factorial
from math import *
n=int(input('Enter an integer'))
s=0; k=n
while n>0:
    s=s+m.factorial(n%10)
    n=n//10
if k==s:
    print("Given number is strong")
else:
    print("Given number %d is not strong"%(k));
```

3-To find out given number is armstrong or not. 153, 370.

```
import math
n=int(input('enter an integer'))
k=n
s=0
c=len(str(n))  # c=int(math.log10(n)+1)
while n>0:
    s=s+int(math.pow(n%10,c))
    n=n//10
if k==s:
    print("Given number %d is Armstrong"%(k))
else:
    print("Given number %d is not Armstrong"%(k))
```

Data types in Python:-

1) Immutable data types
2) Mutable data types

1) Immutable Data types:

These Immutable datatype values not possible to update.

Ex: integers, float and strings, boolean, char, tuple.

Scalar datatypes, Atomic data types.

single value — accessed many times.

Tuple:- 1. It is default data type for sequence of elements in the form of list.

2. It is represented with ( )

3. It contains set of different data type elements, seperated with comma,

4. for accessing tuple elements, use positive index or negative index, slicing.

5. It may contains duplicates

6. It maintains preserves the order assigned to the tuple.

Ex: t = (1, 2, 3, "33.3", "IT", 3424) — index 0 1 2 3 4 / -5 -4 -3 -2 -1

t[0] ⇒ 1

t[-1] ⇒ IT

t[5] ⇒ 1

t[9] ⇒ error   If index is not present it throws an exception.

IndexError: tuple index out of range.

```
>>> t[1:3]        single slicing - single colon
(2,3)
                  double slicing - double colon - start:stop:step
>>> t[::-1]
('IT', 33.3, 3, 2, 1)
>>> t
(1, 2, 3, 33.3, 'IT')
```

```
>>> t[0] = 3
TypeError: 'tuple' object does not support item assignment
>>> t = 3,4,4,56,"IT"
(3,4,4,56,"IT")
```

By default all are call by reference

2) Mutable data types:-

1. List
2. set
3. Dictionary. It is most powerful feature in Python

i) List: It also contains sequence of elements with different data types, enclosed between [ ].

Properties:

i) It may contains duplicates

ii) Deletions, Insertions and updates are possible in List.

iii) List items are accessed with both positive and negative index or slicing.

iv) It preserves the same order assigned to the List.

ex: m = [10,20,30,"IT", 44.3, True]

```
>>> m[0] = 3
>>> m
[3, 20, 30, 'IT', 44.3, True]
```

2) set: It also contains sequence of elements with different data types, enclosed between { }.

Properties:

i) It doesn't allow duplicates

ii) set elements are not possible to access with indexes and slicing also not applicable.

iii) It doesn't preserve the order.

Ex: S = {4, 1, 7, 9}

```
>>> S = {4, 1, 7, 9, 1, 4}
>>> S
{1, 4, 9, 7}
```

```
>>> S[3]
TypeError: set object doesnot support indexing.
>>> S[0:2]
Type Error:
```

3. Dictionary:

A Dictionary is a combination of key and value and it is also represented with curly braces { }. The key and value are seperated with colon : symbol.

keys are unique	It doesnot support indexing and slicing.
Values may repeat	It doesnot preserves the order.

ex: d = {1:"IT", 2:"CSE", 3:10}

```
>>> d
{1:'CSE', 2:'IT', 3:10}
```

Conversion functions:

1) int() :- It is used to convert given data into integer format.

ex: `>>> int('33.3')`

ValueError: invalid literal for int() with base 10: '33.3'

```
>>> int('33')
33
>>> int('111',2)
7
>>> int('112',2)
```

It is possible to convert other base to decimal but decimal to other base not possible in python.

value Error: invalid literal for int() with base 2: '112'

```
>>> int('112',3)
14.
```

2) float():- It is used to convert given data into float format.

```
>>> float(3)
3.0
>>> float(3.333)
3.333
>>> float('it')
ValueError: could not convert string to float: 'it'
```

3) str(): It is used to convert given data into string format.

```
>>> str(3)+str(5)
'35'
>>> str(1,2)
'1,2'
>>>
```

It is not possible to concatenate different datatypes.
we can concatenate only string type

4. bin():

```
>>> bin(10)
'0b1010'
>>>
```

5) oct():

```
>>> oct(10)
'0o12'
```

6) hex():

```
>>> hex(10)
'0xa'
```

```
>>> bin(33.3)
TypeError: 'float' object cannot be interpreted as integer
```

7) chr(): It gives the ascii symbol for given ascii value.

```
>>> chr(97)
'a'
```

8) ord(): It is reverse of chr() → used to display ascii code for corresponding ascii symbol.

```
>>> ord('a')
97
```

9) list(): to convert any sequence into list format.

```
>>>list('welcome')
['w', 'e', 'l', 'c', 'o', 'm', 'e']
>>>list(range(1,11))
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>>list(range(2,10,2))
[2, 4, 6, 8]
```

10) tuple(): to convert any sequence into tuple format.

```
>>>tuple('welcome')
('w', 'e', 'l', 'c', 'o', 'm', 'e')
```

11) set():

```
>>>set('welcome')
{'o', 'l', 'c', 'w', 'm', 'e'}
>>>set('1122')
{'1', '2'}
```

```
>>> m=[1,2,1,2,1,3]
>>> set(m)
{1,2,3}
>>> list(set(m))
[1,2,3]
>>> tuple(list(set(m)))
(1,2,3)
```

12) dict(): It is used to convert

(or) zip(): combines two lists, two tuples or one list and one tuple.

```
>>> k=[1,2,3]
>>> v=['it','cse','eee']
>>> dict(zip(k,v))
{1:'it', 2:'cse',3:'eee'}
>>> k=[10]
>>> v=(1,2,3,4)
>>> dict(zip(k,v))
{10:1}
>>> dict(zip(v,k))
{1:10}
```

13) map():

```
>>> import math
>>> list(map(math.factorial,[3,4,5]))
[6,24,120]
>>> m=[3,6,7]
>>> list(map(chr,[97,98,99]))
['a','b','c']
```

```
>>> v = list(map(bin, [3,4,6]))
>>> v
['0b11', '0b100', '0b110']
>>> v = list(map(ord, list('welcome')))
>>> v
[119, 101, 108, 99, 111, 109, 101]
```

5/2/24

numpy - numeric python

[(3,4,2), (1,2), ('it')] → list of tuples.

[[1,2], [3,4]] → nested list or 2-D list.
m[0][0] = 1

[{1:'a', 2:'n'}] → list of dictionary.

([1,2], (1,3))
↓ mutable ↓ immutable.

```
m = []                    m = [1,2,3]
m[0] = 10 x error.        m[0] = 10 ✓

t = ()
d = {}
s = set()
```
} initialisation of tuple, set, dictionary.

```
t = ()
t[0] = 10 error.
d = {}
d[0] = 10
d : {0:10}
```

```
s = set()
s[0] = 10
error.
```

LAB - ex-2:

4) write a python program to convert decimal number into other bases (Ex: base2, base3, base4 ... etc)

```
k = list(range(10,21))
v = list(map(chr, range(65,76)))
d = dict(zip(k,v))
#d = {10:'A', 11:'B', 12:'C', 13:'D', 14:'E', 15:'F'}
n = int(input('enter decimal number'))
b = int(input('enter base'))
s = ''                    s = ''
```

```
while n>0:
    if n%b<10:
        s=str(n%b)+s
    else:
        s=s+d[n%b]+s
    n=n//b
print("base",b,"value is",s)
```

5) **Compound Assignment Operators:-** The compound assignment operators cannot be nested in Python.

```
==
+=
-=
*=
/=
&=
|=
<<=
>>=
^=
**=
//=
%=
```

e.g. a+=b-=5 # error, due to alias concept

addition of two numbers in manual way

Geeks for Geeks.

Prev write a pp to count no. of vowels and consonants in a given string (without using pre-defined function).

```
s=input('enter a string')
v='aeiou'
c1=0;c2=0;
for i in s:
    if i in v:
        c1=c1+1
    else:
        c2=c2+1
print("vowels count is:",c1)
print("consonants count is:",c2)
```

```
s=input('enter a string:')  # welcome@123
v='aeiou'
a=list(map(chr, range(97,122))
c1=0; c2=0;
for i in s:
    if i in a:
        if i in v:
            c1=c1+1
        else:
            c2=c2+1
print("vowels count is:", c1)  # 3
print("consonants count is:", c2)  # 4
```

```
s=input('enter a string:') # welcome@123
v='aeiou'
a=list(map(chr,range(97,122))
d=list(range(10))  '0123456789'  str(list(range(10))
c1=0; c2=0; c3=0; c4=0;

for i in s:
    if i in a:
        if i in v:
            c1=c1+1
        else:
            c2=c2+1
    else:
        if i in d:
            c3=c3+1
        else:
            c4=c4+1
```

spaces - count

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

```
print("vowel count is:", c1)  3
print("consonants count is:", c2)  4
print("Digits count is:", c3)  3
print("special symbols count is:", c4)  1
```

```
n=int(input("enter an integer:"))  #1234
s=0
p=str(n)
c=1
for i in p:
    print(str(i)*c)
    s=s+int(str(i)*c)
    c=c+1
print(s)
```

```
1
2 2
3 3 3
4 4 4 4
4800
```

o/p: enter an integer:1234

4800

```
m=[i for i in map(int,input().split())]
print(m)
```

o/p: 1 2 3 4 5

[1, 2, 3, 4, 5]

```
>>> m=[[0 for i in range(9)] for j in range(9)]
print(m)
```

[[0,0,0,0,0,0,0,0,0], [0,0,0,0,0,0,0,0,0], [0,0,0,0,0,0,0,0,0]

[0,0,0,0,0,0,0,0,0], [0,0,0,0,0,0,0,0,0], [0,0,0,0,0,0,0,0,0]

[0,0,0,0,0,0,0,0,0], [0,0,0,0,0,0,0,0,0], [0,0,0,0,0,0,0,0,0]]

```
>>> m=[[0 for i in range(3)] for j in range(3)]
for i in m:
    print(i)
```

o/p:

[0, 0, 0]

[0, 0, 0]

[0, 0, 0]

```
>>> m= [i: i*i for i in range(1,5)]
print(m)
```

1: 1, 2: 4, 3:

21/8/24

1. Write a PP to display fibonacci series upto given integer n.

```
n=int(input("enter an integer"))
a=0; b=1
while n>0:
    print(a)
    a,b = b, a+b
    n=n-1
```

2. # Harshad number: 156 = 1+5+6 = 12 156%12 = 0.

```
n=int(input("enter a number!"))
s=0
while (n>0):
    r=n%10
    s=s+r
    n=n//10
if n%s==0:
    print("harshad number")
else:
    print("not a harshad number")
```

Ex: 12, 36, 24.

A Harshad number is a +ve integer which is exactly divisible by the sum of its digits.

3. # Pronic number: Product of two consecutive numbers

Ex: 2, 6, 12, 20, 30, 42 .....

```
n=int(input("enter an integer"))
f=0
for i in range(1,n+1):
    if i*(i+1)==n:
        f=1
        break
if f==0:
    print(n,'is not pronic number')
else:
    print(n,'is a pronic number')
```

Unconditional statements:- / unconditional branching

break
continue } These are only used in loops. There are some restrictions
pass. → It is used in anywhere in the program, There is no restriction.

break: It is used to teminate the loops based on certain conditions.

Ex:
```
for i in range(1,10):
    if (i==5):
        break
    print(i)  # 1234.
```

Continue: continue statement continues the execution of loops.

Ex:
```
for i in range(1,10):
    if (i==5):
        continue
    print(i)  # 123467 89.
```

Pass:

Pass statement is a place holder for future code.

Ex: 1)
```
def display():
    pass
print(display())  # Null
```

2)
```
for i in range(1,20):
    if i<10:
        pass
    else:
        print(i)
```

o/p: 10, 11, 12, 13, 14, 15, 16 17 18 19

22/2/22

Python is Scripting language
PHP is also "

3) class A: mostly used in class and functions.
    pass

for else:-

Syntax:
```
for var in seq:
    #statements-1
else:
    #statements-2
```

1) wherever, for loop executes all iterations successfully without any interruption. then else execute.

ex:
```
for i in range(1,5):
    print("Hello")
else:
    print("GEC")
```

o/p:
Hello
Hello
Hello
Hello
GEC

```
for i in range(1,5):
    if i==3:
        break
    print("Hello")
else:
    print("GEC")
```

o/p:
Hello
Hello

Pronic number:

```
n=int(input("enter an integer"))
for i in range(1,n+1):
    if i*(i+1)==n:
        print(n,"is pronic")
        break
else:
    print(n,'is not a pronic number')
```

2) when there is break, else will not execute.

```
for i in range(10,1):
    print(i):
else:
    print("hello")
```

3) If condition is failed, 0 iterations, then else will execute

o/p:- hello.

numpy
pillow
pickle
pandas.
pip.

While-else statement:-

It is also same as with for-else statement.

Syntax:

```
while condition:
    #statement1
else:
    #statement2
```

Ex:

1) successful iterations

```
i=1
while i<=5:
    print(i)
    i=i+1
else:
    print(i,"hello")
```

o/p:
1
2
3
4
5
6 hello.

2) 0 iterations

```
i=10
while i<=5:
    print(i)
    i=i+1
else:
    print(i,"hello")
```

o/p: 10 hello.

3) break

```
i=1
while i<=5:
    i==3:
        break
    print(i)
    i=i+1
else:
    print(i,"hello")   # 1 2
```

→ In python there is no ternary operator.

6) Short hand if else statement:

Using short hand if else statement we achieve the property of ternary operator.

Syntax: truestatement if condition else falsestatement

ex: 1) a=5; b=8
a if a>b else b

2) a if a>c else c if a>b

7) Identity operators:-

The purpose of Identity operators is to display memory address.

i) is
ii) is not

To display address use id( ) function.

ex: >> a=10
>> id(a)
140716364313712
>>b=10
>> id(b)
140716364313712

Memory Efficient:
The imp feature in python is if value is same it prints same address.

Ex: >>> a is b
True.
Here, it compares address.

>>> c=3
>>> a is c
>>> false.
>>> a is not c
True.
>>> id(c)
140716364315488.

>>> p=a
>>> id(p)
140716364313712.
>>> b=6
>>> id(b)
140716364313584

whenever value is changed, memory also changes.

26/12/24

Command line arguments in Python:

Command line arguments are specified at the time of running python program, in the form of argument vector (argv).

→ A vector is a sequence of elements

By importing/using 'sys' module access the commandline arguments in the form of list. 'import sys'.

In Python the first argument is reserved for file name like as c language. (index '0' is reserved for file name)

Ex:

```
import sys
for i in sys.argv:
        print(i)
```

o/p: c:\users\admin>d:

D:\>python c.py

c.py.

D:\>python c.py 5 8 99

[0] [1] [2] [3]

```
c.py
5
8
99
```

1. Write a PP to find out given number is even or odd using command line arguments.

```
import sys
n=int(sys.argv[1])
if n%2==0:
        print(n,'is even')
else:
        print(n,'is odd!')
```

o/p: D:\> python c.py 5 8 99

5 is odd

2. write a pp to find out sum of given list of elements using command line arguments.

```
import sys                    from sys import *
s=0
for i in range(1,len(sys.argv)):
        s=s+int(sys.argv[i])
print("sum is:"+s).
```

o/p: D:\> python c.py 5 8 99

sum is: 112.

3. write a pp to find out simple interest using command line arguments.

```
import sys
p=float(sys.argv[1])
t=float(sys.argv[2])
r=float(sys.argv[3])
s=(p*t*r)/100
print("simple interest is:",s)
```

→

```
import argparse
p=argparse.ArgumentParser()
p.add_argument('a', type=int)
p.add_argument('b', type=int)
k=p.parse_args()
print("Addition is:", k.a+k.b)
```

we should not give parameters more than required.

* Bitwise operators in Python:

~ tilde bitwise 1's complement (1)
& bitwise AND (3)
| bitwise OR (5)
^ bitwise XOR (4)
<< bitwise leftshift } (2)
>> bitwise Right shift }

```
>> a=5          >>> a=-10        >>> a=-3
>> ~a           >>> ~a           >>> ~a
-6              9                2.
```

```
>>> a,b=3,4
>>> a=b-~a-1
>>> b=a+~b+1
>>> a=a+~b+1
>>> (a,b)
(4,3)
```

Bitwise AND (&):

a = 6 → 1 1 0
b = 5 → 1 0 1
a&b → 1 0 0 → 4.

a	b	a&b
0	0	0
0	1	0
1	0	0
1	1	1

Bitwise OR (|):

a = 6 → 1 1 0
b = 5 → 1 0 1
a|b → 1 1 1 → 7

a	b	a|b
0	0	0
0	1	1
1	0	1
1	1	1

Bitwise XOR (^):

a = 6 → 1 1 0
b = 5 → 1 0 1
a^b → 0 1 1 → 3

a	b	a^b
0	0	0
0	1	1
1	0	1
1	1	0

Swapping of given two using exclusive-OR.

```
a=a^b          >>> a^b
b=a^b          (5,6)
a=a^b .
```

Bitwise leftshift <<: $a \times 2^b$

a=20, b=2

```
a<<b
a<<2
a=80.
```

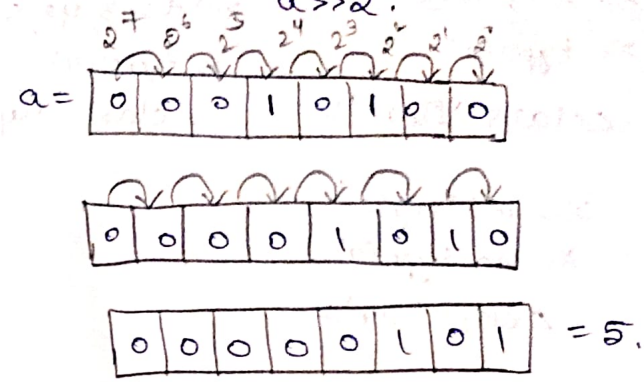
20 × 4 = 80

	128	64	32	16	8	4	2	1	
a =	0	0	0	1	0	1	0	0	
	0	0	1	0	1	0	0	0	= 40
	0	1	0	1	0	0	0	0	= 80

Bitwise Rightshift >>: $a/2^b$

a = 20, b = 2, a>>b $= 20/2^2 = 20/4 = 5$.

a>>2.



Note:

1. There is no increment or decrement operators in python.
2. There is no Teornary operator in python.

Unary operators in python: +, −, ~, not.

Special operators: ~~#, , , not~~.

1) semicolon; a=3; b=6; c=10
2) colon: → starting of block, slicing.
3) comma operator, a,b = 3,4 a,b,c = 3,6,10.

Type casting:

conversion from one data type to another data type.

1) Implicit type casting.
2) Explicit type casting or type conversion.

Implicit type casting:

Casting is automatically done by the compiler is called as Implicit type casting.

```
a = 5          #autoboxing
a = 2.333      #autoboxing
a = 'srgec'    #autoboxing.
```