**Operators**

1. Arthematic operator
2. Relational operator
3. Logical operator
4. Bit wise operator
5. Assignment operator
6. Special operator

**Arthamatic operator: -**

1. + =>add

2. - => sub

3. \* =>multiplication

4. / =>division

5.% =>modular division

6. // => flour division

7. \*\*=>exponent

**flour division(//)** :-

=>argument ‘int’ display integer values,

=>argument float display float values.

Ex:-

a=10  
b=20  
print('a+b=',a+b)  
print('a-b=',a-b)  
print('a\*b=',a\*b)  
print('a/b=',a/b)  
print('a%b=',a%b)  
print('a//b=',a//b)  
print('a\*\*b=',a\*\*b)

outout:-

a+b= 30

a-b= -10

a\*b= 200

a/b= 0.5

a%b= 10

a//b= 0

a\*\*b= 100000000000000000000

Ex:2

a=10.5  
b=2.5  
print('a+b=',a+b)  
print('a-b=',a-b)  
print('a\*b=',a\*b)  
print('a/b=',a/b)  
print('a%b=',a%b)  
print('a//b=',a//b)  
print('a\*\*b=',a\*\*b)

output:-

a+b= 13.0

a-b= 8.0

a\*b= 26.25

a/b= 4.2

a%b= 0.5

a//b= 4.0

a\*\*b= 357.2508309997333

**’+’ operators**: -

* It is used for concatenation of two strings.

Ex:-

>> "sek"+"3"

'sek3'

>>> "sek"+str(4)

'sek4'

=>if arguments are strings it display the it results if arguments is number and string does not available.

Ex:- "sekhar"+3

Traceback (most recent call last):

File "<pyshell#0>", line 1, in <module>

"sekhar"+3

TypeError: can only concatenate str (not "int") to str

>>>type error: can only concatenal str (not ‘int’ to ‘str’ )

\*(star operator) :-Is also string .

=> repetition operator (multiplication)

=> if we you want apply star operator string compulsory one arguments is ‘int’ type otherwise it displays error.

Ex:-

"sekhar"+"3"

'sekhar3'

>>> 2\*"sekhar"

'sekharsekhar'

>>> 5\*"sekhar"

'sekharsekharsekharsekharsekhar'

>>> "sekhar"+str(4)

'sekhar4'

>>> "sekhar"\*5

'sekharsekharsekharsekharsekhar'

>>> "sekhar"\*3.0

Traceback (most recent call last):

File "<pyshell#12>", line 1, in <module>

"sekhar"\*3.0

TypeError: can't multiply sequence by non-int of type 'float'

>>> "sekhar"\*'3'

Traceback (most recent call last):

File "<pyshell#13>", line 1, in <module>

"sekhar"\*'3'

TypeError: can't multiply sequence by non-int of type 'str'

>>> "sekhar"\*"3"

Traceback (most recent call last):

File "<pyshell#14>", line 1, in <module>

"sekhar"\*"3"

TypeError: can't multiply sequence by non-int of type 'str'

>>> 10\*\*-2

0.01

>>> 10+3j\*\*10+3j

(-59039+3j)

Note :-x/10 x%10 any number ‘0’ zero will get division error

Ex:

>>> 10/10

1.0

>>> 10/0

Traceback (most recent call last):

File "<pyshell#18>", line 1, in <module>

10/0

ZeroDivisionError: division by zero

>>> 10%0

Traceback (most recent call last):

File "<pyshell#19>", line 1, in <module>

10%0

ZeroDivisionError: integer division or modulo by zero

>>> 0/0

Traceback (most recent call last):

File "<pyshell#20>", line 1, in <module>

0/0

ZeroDivisionError: division by zero

>>> 0/0.0

Traceback (most recent call last):

File "<pyshell#21>", line 1, in <module>

0/0.0

ZeroDivisionError: float division by zero

* (\*)Any thing power ‘0’ zero result is one ‘1’

Ex:- 10\*\*0

1

>>100\*\*0

1

**Relational operator** :-

* Grater than( >)
* Grater than equal to(>=)
* Less than (<)
* Less than equal to (<=)

Ex:1

a=10  
b=20  
print("a>b",a>b)  
print("a>=b",a>=b)  
print("a<=b",a<=b)  
print("a<=b",a<=b)

output:-

a>b False

a>=b False

a<=b True

a<=b True

with string :-

AscII values => A=65 a=97

=> based on alphabetical order it check the condition .

Ex:- cherry , ram

* Incase first letter is same it compares second letter and check the capital and small letter based on ASCII values.

Ex:

a="cherry"  
b="ram"  
print("a>b",a>b)  
print("a.=b",a>b)  
print("a<b",a<=b)  
print("a<=b",a<=b)

output:-

a>b False

a.=b False

a<b True

a<=b True

Bool type grater than :-

Ex:

a=True  
b=False  
print("a>b",a>b)  
print("a>=b",a>=b)  
print("a<b",a<b)  
print("a<=b",a<=b)

output:-

a>b True

a>=b True

a<b False

a<=b False

chaining of relational operator: -

* All comparison performed by python.
* All comparison True result is true, if all the comparisons are false result is false.
* If one comparison is false all condition is false.

Ex:-

>>> 10<20

True

>>> 10<20<30

True

>>> 10<20>30

False

>>> 0>1<0

False

>>> 10<20<30<40<50

True

Equality Operator :-

== and !=

Ex:

>>> 10==20

False

>>> 10==10

True

>>> True==True

True

>>> False==True

False

>>> 'chitti'=='darling'

False

>>> 'anu'=='anu'

True

>>> 10!=20

True

>>> 10!=10

False

Chaining Operator Equality:-

>>> 10==20==30

False

>>> 10+5+5==4+2

False

Note :

* = Assignment operator.
* == comparison operator .

Logical Operator :-

* And
* OR
* Not

Boolean type :-

* And-----both operands are True then only result is True

Ex:-(AND)

>>> True and True

True

>>> True and False

False

>>> False and True

False

OR: -

* It at least one argument is True the results is True other wise False .

Ex:- >>> True or True

True

>>> True or False

True

>>> False or False

False

Not: - (oposit action )

>>> not True

False

>>> not False

True

Non Boolean Type –Behavior :

1. ‘0’ (zero) means False
2. Non ‘0’ zero means True
3. Empty string always treated as False.

Non Boolean and

* If x value is False then result is x otherwise it print the returns y.
* Ex:

>>> 0 and 20

0

>>> 0 and 2000

0

>>> 10 and 20

20

>>> 1 and "sekhar"

'sekhar'

>>> 0 and "sekhar"

0

Non Boolean OR:-

* if x values is true then result is –x otherwise it print the value of Y.

Ex:-

>>> 1 or 20

1

>>> 1 or "sekhar"

1

>>> 0 or ''”

''

>>> 1 or 20

1

>>> 30 or 45

30

>>> 0 or 40

40

>>> 0 or 'sekhar'

'sekhar'

Non Boolean Not:-

* If it is True then output is false ,if it is false it evaluate true .

Ex:-

>>not 1

False

>>> not 0

True

>>> not''

True

>>> not int

False

>>> not str

False

>>> not True

False

>>> not''

True

>>> 1 or 20

Bitwise Operator: -

* & (and)
* ! (OR)
* ~ ( not)
* ^ (exclusive)
* << (left shift )
* >>(right shift)
* int type and Boolean type applicable only other type will show error.

1. **bitwise and:**-

=> if both bits are one (‘1’) then only result is one (‘1’) otherwise result is zero (‘0’)

4=>100(binary code)

5=>100(binary code)

4&5 =100 (binary code)

Ex:-

>>> 4&5

4

Ex1:

>>> bin(10)

'0b1010'

>>> bin(12)

'0b1100'

>>> 10&12

8

bitwise (‘OR’):-

=> if at least one bit 1 then result is 1 , otherwise result is zero (0)

Ex:-

>>10|12

14

^ Bitwise Exclusive :-

* if bits are different then only result is one(1) other wise result is zero (0).

Ex:-

>>bin (10)

0b100

>>bin(5)

0b101

>>4^5

1

~Bitwise Not: -

=>it perform the compliment operation.

Ex:-

>>~4

-5

>>~5

-6

Bitwise left shift (<<) :

5<<2 (shifting number of position)

To move shifting left shift

Ex:-

>> 5<<2

20

Bitwise Right Shift (>>) :

* right we cent cells by default file with zero’0’

Ex:-

10>>2

>>> 2

Assignment Operator:-

Syntax:

**variable =value**

Ex:-

X= 100

>>> print(x)

>>>100

Multiple variable multiple values.

Ex:-

>> a,b,c=10,20,30

>>a=10

10

>>b=20

20

>>c=30

30

compound assignment operator:

* assignment operator - mixed with same other operator.

Ex:-

>>> x=10

>>> x+=2

>>> x

12

>>> x-=2

>>> x

10

>>> x\*3

30

>>> x\*=2

>>> x

20

>>> x/=2

>>> x

10.0

>>> x%=2

>>> x

0.0

>>> x//=2

>>> x

0.0

>>> x\*\*=5

>>> x

0.0

>>> x\*\*=2

>>> x

0.0

>>> a=4

>>> a&=5

>>> print(a)

4

>>> b=4

>>> b|=5

>>> print(b)

5

>>> a=5

>>> a^=5

>>> print(a)

0

>>> a=<<5

SyntaxError: invalid syntax

>>> a=5

>>> a<<=2

>>> print(a)

20

>>> b=10

>>> b>>=2

>>> print(b)

2

Notes :-

* ++ ==> not possible.
* -- ==> not possible.

Possible – operators:

1. +=

2. -=

3. \*=

4. /=

5. %=

6. //=

7. &=

8. ^=

9. <<=

10. >>=

11. \*\*=

**Ternary operator** :-

Syntax:

**X=first value if condition else second value**

Ex:-

>>> a=10

>>> b=20

>>> x=30 if a>b else 40

>>> x

40

x=30 if 10=20 else =40

Print(x)

Ex 1:-

>>> a=10

>>b=20

>>> x=30if a<b else 40

>>> print(x)

30

Ex 2:-

a=int (input(“enter first no”))

b=int(input (“enter second no”))

Min=a if a<b else b

Print (“minimum value:”, min)

Output :- enter first no 5

Enter second no 4

Minimum no 4

Ex3:-

a=int(input(“enter first no “))

b=int(input(“enter second no “))

max=a if a>b else b

print(maximum value :”,max)

output:- enter first no 5

enter second no10

maximum value:20

Nesting Ternary Operator :-

Syntax:

**X=first value if condition1 else second value ifcondition2 else third value**

* If condtion1 is True result is first value other wise second value. Again check ifcondtion2 it is true result is second value otherwise third value.

Ex:-

a=int(input(“enter no1”))

b=int(input(“enter no 2”))

c=int(input(“enter no 3”))

max=a if a>b>c else b if b>c else c

print(maximum value:”,max)

output :-enter no 1 10

enter no 2 20

enter no 30

max value :30

Ex1:-

a=int(input(“enter no1 “))

b=int(input(“enter no2”))

c=int(input(“enter no 3”))

min=a if a<b<c else b if b<c else c

print(“minimum value :”,mim)

output :-enter no 1 40

enter no 2 50

enter no3 60

min value is 40

Special Operator: -

* Identity operator
* Membership operator

**1). Identity operator** :-

* is
* is not
* these are used for address comparison purpose .

Ex:-

>>> a=10

>>> type(a)

<class 'int'>

>>> id(a)

1580879604304

>>> b=10

>>> id(b)

1580879604304

>>>

>>> print(a is b)

True

>>> print(a is not b)

False

>>> a="deepu"

>>> b="deepu"

>>> print(a is b)

True

>>> print(a is not b)

False

**2. membership operator**: -

* To check whether the given object member are not.
* in
* not in

>>> l=[10,20,30,"0"]

>>> print(10 in l)

True

>>> print(50 in l)

False

>>> print("s" in l)

False

>>> print("s" not in l)

True

* In operator applicable only one argument .

Ex:-

S=learn python

Print(“learn “ in s)

>>>True

>>> print(“sekhar” in s)

>>> False

>>>print(“python “ not in s)

>>> False

Generalized Rules Programming: -

* Unary operator --- highest priority
* Binary operator ---second highest priority.
* Ternary operator --- third highest priority
* Assignment operator ----less priority.

**Priority in python** :-

1. () ------parenthesis.
2. \*\* -----exponent operator.
3. {~,\*,/,%,//,+,- }\_ unary operator.
4. +,-
5. << , >> ------- bitwise shift .
6. & , ^ , | ---------bitwise .
7. >, >= ,< ,<= , == ,!=
8. += , -= , \*= , \=…..etc.
9. Is , is not

10.In , not in

11.And, not ,or----- logical operator .