Operator's in python

1- ARITHMETIC OPERATOR (+ , -, , /, %, %%, *, ^

2- ASSIGNMENT OPERATOR (=)

3- RELATIONAL OPERATOR

4- LOGICAL OPERATOR

5- UNARY OPERATOR

Arithmetic operator

In [6]:
```
x1, y1 = 10, 5
```

In [8]:
```
x1  + y1
```

Out[8]: 15

In [10]:
```
x1 - y1
```

Out[10]: 5

In [12]:
```
x1 * y1
```

Out[12]: 50

In [14]:
```
x1 / y1
```

Out[14]: 2.0

In [16]:
```
x1 // y1
```

Out[16]: 2

In [18]:
```
x1 % y1
```

Out[18]: 0

In [20]:
```
x1 ** y1
```

Out[20]: 100000

In [22]:
```
2**3
```

Out[22]: 8

Assignment operator

In [25]:
```
x = 2
```

In [27]:
```
x=x+2
```

In [29]:
```
x
```

Out[29]: 4

In [31]:
```
x += 2
```

In [33]:
```
x
```

Out[33]: 6

In [37]:
```
x*=2
```

In [39]:
```
x
```

Out[39]: 12

In [41]:
```
x-=2
```

In [43]:
```
x
```

Out[43]: 10

In [45]:
```
x/=2
```

In [47]:
```
x
```

Out[47]: 5.0

In [49]:
```
a,b=5,6
```

In [51]:
```
a
```

Out[51]: 5

In [53]:
```
b
```

Out[53]: 6

unary operator

Here we are applying unary minus operator(-) on the operand n; the value of m becomes -7, which indicates it as a negative value.

In [68]:
```
n = 7
```

In [60]:
```
m = -(n)
```

In [62]:
```
m
```

Out[62]: -7

In [64]:
```
n
```

Out[64]: 7

In [66]:
```
-n
```

Out[66]: -7

Relational operator

we are using this operator for comparing

In [71]:
```
a = 5
b = 7
```

In [73]:
```
a==b
```

Out[73]: False

In [75]:
```
a<b
```

Out[75]: True

In [77]:
```
a>b
```

Out[77]: False

In [80]:
```
# a = b # we cannot use = operatro that means it is assigning
```

In [82]:
```
a == b
```

Out[82]: False

In [84]:
```
a = 10
```

In [86]:
```
a!=b
```

Out[86]: True

In [88]:
```
# hear if i change b = 6
b = 10
```

In [90]:
```
a == b
```

Out[90]: True

In [92]:
```
a >= b
```

Out[92]: True

In [94]:
```
a <= b
```

Out[94]: True

In [96]:
```
a < b
```

Out[96]: False

In [98]:
```
a>b
```

Out[98]: False

In [100…
```
b = 7
```

In [102…
```
a != b
```

Out[102… True

LOGICAL OPERATOR

AND, OR, NOT

In [105…
```
a = 5
b = 4
```

In [107…
```
a < 8 and b < 5 #refer to the truth table
```

Out[107… True

In [109…
```
a < 8 and b < 2
```

Out[109… False

In [111…
```
a < 8 or b < 2
```

Out[111… True

In [113…
```
a>8 or b<2
```

Out[113… False

In [115…
```
x = False
x
```

Out[115… False

In [117…
```
not x
```

Out[117… True

In [119…
```
x = not x
x
```

Out[119… True

In [121…
```
-x
```

Out[121… -1

In [123…
```
x
```

Out[123… True

In [125…
```
not x
```

Out[125… False

Number system coverstion (bit-binary digit)

binary : base (0-1) --> please divide 15/2 & count in reverse order

octal : base (0-7)

hexadecimal : base (0-9 & then a-f)

when you check ipaddress you will these format --> cmd - ipconfig

In [130…
```
25
```

Out[130… 25

In [132…
```
bin(25)
```

Out[132… '0b11001'

In [134…
```
bin(35)
```

Out[134… '0b100011'

In [136…
```
bin(20)
```

Out[136… '0b10100'

In [138…
```
int(0b10100)
```

Out[138… 20

In [140…
```
int(0b1111)
```

Out[140… 15

In [142…
```
oct(15)
```

Out[142… '0o17'

In [144…
```
hex(9)
```

Out[144… '0x9'

In [146…
```
int(0xf)
```

Out[146… 15

In [148…
```
hex(10)
```

Out[148… '0xa'

In [150…
```
hex(25)
```

Out[150… '0x19'

In [152…
```
int(0x15)
```

Out[152… 21

swap variable in python

(a,b = 5,6) After swap we should get ==> (a, b = 6,5)

In [155…
```
a = 5
b = 6
```

In [157…
```
a=b
b=a
```

In [159…
```
a,b
```

Out[159… (6, 6)

In [163…
```
a = 5
b = 6
```

In [165…
```
a,b=b,a
```

In [167…
```
a,b
```

Out[167… (6, 5)

In [169…
```
a1=7
b1=8
```

In [171…
```
temp=a1
a1=b1
b1=temp
```

In [173…
```
print(a1)
print(b1)
```

```
8
7
```

In [175…
```
a2 = 5
b2 = 6
```

In [177…
```
#swap variable formulas
a2 = a2 + b2
b2 = a2 - b2
a2 = a2 - b2
```

In [179…
```
print(a2)
print(b2)
```

```
6
5
```

In [181…
```
print(0b101) # 101 is 3 bit
print(0b110) # 110 also 3bit
```

```
5
6
```

In [183…
```
#but when we use a2 + b2 then we get 11 that means we will get 4 bit which is 1
print(bin(11))
print(0b1011)
```

```
0b1011
11
```

In [185…
```
#there is other way to work using swap variable also which is XOR because it wil
a2 = a2 ^ b2
b2 = a2 ^ b2
a2 = a2 ^ b2
```

In [187…
```
print(a2)
print(b2)
```

```
5
6
```

In [189…
```
bin(a2)
```

Out[189… `'0b101'`

In [191…
```
bin(b2)
```

Out[191… `'0b110'`

In [193…
```
a2=a2^b2
bin(a2)
```

Out[193… `'0b11'`

In [195…
```
b2=a2^b2
bin(b2)
```

Out[195… `'0b101'`

In [197…
```
a2=a2^b2
```

In [199…
```
a2
```

Out[199… `6`

In [201…
```
b2
```

Out[201… 5

In [203…
```
a2 , b2 = b2, a2
```

In [205…
```
print(a2)
print(b2)
```

```
5
6
```

BITWISE OPERATOR

- WE HAVE 6 OPERATORS

COMPLEMENT (~) || AND (&) || OR (|) || XOR (^) || LEFT SHIFT (<<) || RIGHT SHIFT (>>)

In [208…
```
print(bin(12))
print(bin(13))
```

```
0b1100
0b1101
```

complement --> you will get this key below esc character

12 ==> 1100 || first thing we need to understand what is mean by complement. complement means it will do reverse of the binary format i.e. - ~0 it will give you 1 ~1 it will give 0 12 binary format is 00001100 (complement of ~00001100 reverse the number - 11110011 which is (-13)

but the question is why we got -13 to understand this concept (we have concept of 2's complement 2's complement mean (1's complement + 1) in the system we can store +Ve number but how to store -ve number

lets understand binary form of 13 - 00001101 + 1

In [213…
```
# COMPLEMENT (~) (TILDE  OR TILD)
~12 # why we get -13 . first we understand what is complment means (reversr of b
```

Out[213… -13

In [215…
```
bin(12)
```

Out[215… '0b1100'

In [219…
```
bin(~12)
```

Out[219… '-0b1101'

In [221…
```
~45
```
Out[221… -46

In [223…
```
~6
```
Out[223… -7

In [225…
```
~-1
```
Out[225… 0

In [227…
```
~0
```
Out[227… -1

bit wise and operator

AND - LOGICAL OPERATOR ||| & - BITWISE AND OPERATOR
(we know that 1 & 1 is 1) 12 - 00001100 13 - 00001101 when we are add both then then outut we will get as 12

In [230…
```
12 & 13
```
Out[230… 12

In [232…
```
1 & 1
```
Out[232… 1

In [234…
```
18 & 13
```
Out[234… 0

In [236…
```
1 | 0
```
Out[236… 1

In [238…
```
1 & 0
```
Out[238… 0

In [240…
```
12 | 13
```
Out[240… 13

In [242…
```
# 35 & 40 -please do the homework conververt 35,40 to binary format
```

In [244…
```
bin(35)
```
Out[244… '0b100011'

In [246…
```
bin(40)
```
Out[246…
```
'0b101000'
```
In [256…
```
int(0b100000)
```
Out[256…
```
32
```
In [258…
```
35 & 40
```
Out[258…
```
32
```
In [260…
```
35 | 40
```
Out[260…
```
43
```
In [262…
```
# in XOR if the both number are different then we will get 1 or else we will get

12 ^ 13
```
Out[262…
```
1
```
In [264…
```
25 ^ 30
```
Out[264…
```
7
```
In [266…
```
bin(25)
```
Out[266…
```
'0b11001'
```
In [268…
```
bin(30)
```
Out[268…
```
'0b11110'
```
In [270…
```
int(0b000111)
```
Out[270…
```
7
```
In [272…
```
# BIT WISE LEFT OPERATOR
#bit wise left operator bydefault you will take 2 zeros ( )
#10 binary operator is 1010 | also i can say 1010
10<<2
```
Out[272…
```
40
```
In [274…
```
bin(10)
```
Out[274…
```
'0b1010'
```
In [276…
```
bin(40)
```
Out[276…
```
'0b101000'
```
In [278…
```
20<<4 #can we do this
```

Out[278… 320

In [280…
```
bin(20)
```

Out[280… '0b10100'

In [282…
```
bin(320)
```

Out[282… '0b101000000'

BITWISE RIGHTSHIFT OPERATOR

In [285…
```
10>>2
```

Out[285… 2

In [287…
```
bin(20)
```

Out[287… '0b10100'

In [289…
```
20>>4
```

Out[289… 1

import math module

In [292…
```
x = sqrt(25) #sqrt is inbuild function
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[292], line 1
----> 1 x = sqrt(25)

NameError: name 'sqrt' is not defined
```

In [294…
```
import math # math is module
```

In [296…
```
x = math.sqrt(25)
x
```

Out[296… 5.0

In [298…
```
x1 = math.sqrt(15)
x1
```

Out[298… 3.872983346207417

In [300…
```
print(math.floor(2.9)) #floor - minimum or least value
```

```
2
```

In [302…
```
print(math.ceil(2.9)) #ceil - maximum or highest value
```

```
3
```

In [304…
```
print(math.pow(3,2))
```
```
9.0
```

In [306…
```
print(math.pi) #these are constant
```
```
3.141592653589793
```

In [308…
```
print(math.e) #these are constant
```
```
2.718281828459045
```

In [310…
```
import math as m
m.sqrt(10)
```

Out[310… 3.1622776601683795

In [312…
```
from math import sqrt,pow # math has many function if you want to call specific
pow(2,3)
```

Out[312… 8.0

In [314…
```
from math import * # math has many function if you want to call specific functio

print(pow(2,3))
print(floor(2.3))
```
```
8.0
2
```

In [316…
```
round(pow(2,3))
```

Out[316… 8

In [320…
```
help(math)
```

```
Help on built-in module math:

NAME
    math

DESCRIPTION
    This module provides access to the mathematical functions
    defined by the C standard.

FUNCTIONS
    acos(x, /)
        Return the arc cosine (measured in radians) of x.

        The result is between 0 and pi.

    acosh(x, /)
        Return the inverse hyperbolic cosine of x.

    asin(x, /)
        Return the arc sine (measured in radians) of x.

        The result is between -pi/2 and pi/2.

    asinh(x, /)
        Return the inverse hyperbolic sine of x.

    atan(x, /)
        Return the arc tangent (measured in radians) of x.

        The result is between -pi/2 and pi/2.

    atan2(y, x, /)
        Return the arc tangent (measured in radians) of y/x.

        Unlike atan(y/x), the signs of both x and y are considered.

    atanh(x, /)
        Return the inverse hyperbolic tangent of x.

    cbrt(x, /)
        Return the cube root of x.

    ceil(x, /)
        Return the ceiling of x as an Integral.

        This is the smallest integer >= x.

    comb(n, k, /)
        Number of ways to choose k items from n items without repetition and with
out order.

        Evaluates to n! / (k! * (n - k)!) when k <= n and evaluates
        to zero when k > n.

        Also called the binomial coefficient because it is equivalent
        to the coefficient of k-th term in polynomial expansion of the
        expression (1 + x)**n.

        Raises TypeError if either of the arguments are not integers.
        Raises ValueError if either of the arguments are negative.
```

```
    copysign(x, y, /)
        Return a float with the magnitude (absolute value) of x but the sign of
y.

        On platforms that support signed zeros, copysign(1.0, -0.0)
        returns -1.0.

    cos(x, /)
        Return the cosine of x (measured in radians).

    cosh(x, /)
        Return the hyperbolic cosine of x.

    degrees(x, /)
        Convert angle x from radians to degrees.

    dist(p, q, /)
        Return the Euclidean distance between two points p and q.

        The points should be specified as sequences (or iterables) of
        coordinates.  Both inputs must have the same dimension.

        Roughly equivalent to:
            sqrt(sum((px - qx) ** 2.0 for px, qx in zip(p, q)))

    erf(x, /)
        Error function at x.

    erfc(x, /)
        Complementary error function at x.

    exp(x, /)
        Return e raised to the power of x.

    exp2(x, /)
        Return 2 raised to the power of x.

    expm1(x, /)
        Return exp(x)-1.

        This function avoids the loss of precision involved in the direct evaluat
ion of exp(x)-1 for small x.

    fabs(x, /)
        Return the absolute value of the float x.

    factorial(n, /)
        Find n!.

        Raise a ValueError if x is negative or non-integral.

    floor(x, /)
        Return the floor of x as an Integral.

        This is the largest integer <= x.

    fmod(x, y, /)
        Return fmod(x, y), according to platform C.
```

```
    x % y may differ.

frexp(x, /)
    Return the mantissa and exponent of x, as pair (m, e).

    m is a float and e is an int, such that x = m * 2.**e.
    If x is 0, m and e are both 0.  Else 0.5 <= abs(m) < 1.0.

fsum(seq, /)
    Return an accurate floating-point sum of values in the iterable seq.

    Assumes IEEE-754 floating-point arithmetic.

gamma(x, /)
    Gamma function at x.

gcd(*integers)
    Greatest Common Divisor.

hypot(...)
    hypot(*coordinates) -> value

    Multidimensional Euclidean distance from the origin to a point.

    Roughly equivalent to:
        sqrt(sum(x**2 for x in coordinates))

    For a two dimensional point (x, y), gives the hypotenuse
    using the Pythagorean theorem:  sqrt(x*x + y*y).

    For example, the hypotenuse of a 3/4/5 right triangle is:

        >>> hypot(3.0, 4.0)
        5.0

isclose(a, b, *, rel_tol=1e-09, abs_tol=0.0)
    Determine whether two floating-point numbers are close in value.

      rel_tol
        maximum difference for being considered "close", relative to the
        magnitude of the input values
      abs_tol
        maximum difference for being considered "close", regardless of the
        magnitude of the input values

    Return True if a is close in value to b, and False otherwise.

    For the values to be considered close, the difference between them
    must be smaller than at least one of the tolerances.

    -inf, inf and NaN behave similarly to the IEEE 754 Standard.  That
    is, NaN is not close to anything, even itself.  inf and -inf are
    only close to themselves.

isfinite(x, /)
    Return True if x is neither an infinity nor a NaN, and False otherwise.

isinf(x, /)
    Return True if x is a positive or negative infinity, and False otherwise.
```

```
    isnan(x, /)
        Return True if x is a NaN (not a number), and False otherwise.

    isqrt(n, /)
        Return the integer part of the square root of the input.

    lcm(*integers)
        Least Common Multiple.

    ldexp(x, i, /)
        Return x * (2**i).

        This is essentially the inverse of frexp().

    lgamma(x, /)
        Natural logarithm of absolute value of Gamma function at x.

    log(...)
        log(x, [base=math.e])
        Return the logarithm of x to the given base.

        If the base is not specified, returns the natural logarithm (base e) of
x.

    log10(x, /)
        Return the base 10 logarithm of x.

    log1p(x, /)
        Return the natural logarithm of 1+x (base e).

        The result is computed in a way which is accurate for x near zero.

    log2(x, /)
        Return the base 2 logarithm of x.

    modf(x, /)
        Return the fractional and integer parts of x.

        Both results carry the sign of x and are floats.

    nextafter(x, y, /, *, steps=None)
        Return the floating-point value the given number of steps after x towards
y.

        If steps is not specified or is None, it defaults to 1.

        Raises a TypeError, if x or y is not a double, or if steps is not an inte
ger.
        Raises ValueError if steps is negative.

    perm(n, k=None, /)
        Number of ways to choose k items from n items without repetition and with
order.

        Evaluates to n! / (n - k)! when k <= n and evaluates
        to zero when k > n.

        If k is not specified or is None, then k defaults to n
        and the function returns n!.
```

```
    Raises TypeError if either of the arguments are not integers.
    Raises ValueError if either of the arguments are negative.

pow(x, y, /)
    Return x**y (x to the power of y).

prod(iterable, /, *, start=1)
    Calculate the product of all the elements in the input iterable.

    The default start value for the product is 1.

    When the iterable is empty, return the start value.  This function is
    intended specifically for use with numeric values and may reject
    non-numeric types.

radians(x, /)
    Convert angle x from degrees to radians.

remainder(x, y, /)
    Difference between x and the closest integer multiple of y.

    Return x - n*y where n*y is the closest integer multiple of y.
    In the case where x is exactly halfway between two multiples of
    y, the nearest even value of n is used. The result is always exact.

sin(x, /)
    Return the sine of x (measured in radians).

sinh(x, /)
    Return the hyperbolic sine of x.

sqrt(x, /)
    Return the square root of x.

sumprod(p, q, /)
    Return the sum of products of values from two iterables p and q.

    Roughly equivalent to:

        sum(itertools.starmap(operator.mul, zip(p, q, strict=True)))

    For float and mixed int/float inputs, the intermediate products
    and sums are computed with extended precision.

tan(x, /)
    Return the tangent of x (measured in radians).

tanh(x, /)
    Return the hyperbolic tangent of x.

trunc(x, /)
    Truncates the Real x to the nearest Integral toward 0.

    Uses the __trunc__ magic method.

ulp(x, /)
    Return the value of the least significant bit of the float x.

DATA
    e = 2.718281828459045
```

```
    inf = inf
    nan = nan
    pi = 3.141592653589793
    tau = 6.283185307179586

FILE
    (built-in)
```

user input function in python || command line input

In [323…
```
x = input()
y = input()
z = x + y
print(z) # console is waiting for user to enter input
# also if you work in idle
```

```
45
```

In [325…
```
x1 = input('Enter the 1st number') #whenevery you works in input function it alw
y1 = input('Enter the 2nd number') # it wont understand as arithmetic operator
z1 = x1 + y1
print(z1)
```

```
45
```

In [327…
```
type(x1)
type(y1)
```

Out[327…
```
str
```

In [329…
```
x1 = input('Enter the 1st number') #whenevery you works in input function it alw
a1 = int(x1)
y1 = input('Enter the 2nd number') # it wont understand as arithmetic operator
b1 = int(y1)
z1 = a1 + b1
print(z1)
```

```
405
```

In [331…
```
x2 = int(input('Enter the 1st number'))
y2 = int(input('Enter the 2nd number'))
z2 = x2 + y2
z2
```

Out[331…
```
405
```

In [333…
```
ch = input('enter a char')
print(ch)
```

```
Sider
```

In [335…
```
print(ch[0])
```

```
S
```

In [337…
```
print(ch[1])
```

```
i
```

In [339…
```
print(ch[-1])
```

```
r
```

In [343…
```
ch = input('enter a char')[0]
print(ch)
```

```
B
```

In [345…
```
ch = input('enter a char')[1:3]
print(ch)
```

```
us
```

In [347…
```
ch = input('enter a char')
print(ch) # if you enter as 2 + 6 -1 we get output as 2 + 6-1 only
```

```
2+6-4
```

In [351…
```
result = eval(input('enter an expr'))
print(result)
```

```
9
```