

Number system coverstion (bit-binary digit)

```
In [1]: 20
```

```
Out[1]: 20
```

```
In [2]: bin(24)
```

```
Out[2]: '0b11000'
```

```
In [3]: int(0b11000)
```

```
Out[3]: 24
```

```
In [4]: bin(25)
```

```
Out[4]: '0b11001'
```

```
In [6]: type(bin(25))
```

```
Out[6]: str
```

```
In [7]: int(0b11001)
```

```
Out[7]: 25
```

```
In [8]: oct(25)
```

```
Out[8]: '0o31'
```

```
In [9]: int(0o31)
```

```
Out[9]: 25
```

```
In [10]: oct(40)
```

```
Out[10]: '0o50'
```

```
In [11]: int(0o50)
```

```
Out[11]: 40
```

```
In [12]: 0o50
```

```
Out[12]: 40
```

```
In [13]: bin(2)
```

```
Out[13]: '0b10'
```

```
In [14]: int(0b10)
```

Out[14]: 2

In [17]: `hex(24)`

Out[17]: '0x18'

In [19]: `int(0x18)`

Out[19]: 24

In [20]: `0xc`

Out[20]: 12

In [22]: `0xd`

Out[22]: 13

In [23]: `hex(25)`

Out[23]: '0x19'

In [24]: `0x15`

Out[24]: 21

Swap 2-Variable in python

In [74]: `a=5`
`b=6`

In [75]: `print(a)`
`print(b)`

5
6

In [76]: `a1=7` *# swapping the values by using swap method*
`b1=8`

In [77]: `temp=a1`
`a1=b1`
`b1=temp`

In [78]: `print(a1)`
`print(b1)`

8
7

Without using Third Variable

In [90]: `a2=5`
`b2=6`

In [91]: `a2=a2+b2`
`b2=a2-b2`

```
a2=a2-b2  
print(a2)  
print(b2)
```

6
5

In [92]: `0b110`

Out[92]: 6

```
print(0b110)  
print(0b101)
```

6
5

```
print(bin(11))  
print(0b1011)
```

0b1011
11

```
print(a2)  
print(b2)
```

6
5

In [96]: `a2,b2=b2,a2`

```
print(a2)  
print(b2)
```

5
6

```
In [103... a3=4  
b3=5
```

```
In [104... a3 = a3 ^ b3  
b3 = a3 ^ b3  
a3 = a3 ^ b3
```

```
In [105... print(a3)  
print(b3)
```

5
4

Bitwise and Operator

```
In [45]: # here the AND funtion is 1.if both are having '1' and ' 1' is '1' remaining all  
# here the AND funtion is 1.if both are having '0' and '0' is '0' remaining all
```

In [46]: `12&13`

Out[46]: 12

In [47]: `12|14`

Out[47]: 14

In [48]: `1&0`

Out[48]: 0

In [49]: `1|0`

Out[49]: 1

In [54]: `bin(13)`
`print(bin(35))`
`print(bin(40))`

0b100011

0b101000

In [55]: `35&40`

Out[55]: 32

In [56]: `35|40`

Out[56]: 43

In [57]: `12^13`

Out[57]: 1

In [58]: `print(bin(25))`

0b11001

In [60]: `print(bin(30))`
`print(bin(40))`

0b11110

0b101000

In [61]: `bin(7)`

Out[61]: '0b111'

In [62]: `bin(30)`

Out[62]: '0b11110'

In [63]: `10<<1`

Out[63]: 20

In [65]: `10<<2`

Out[65]: 40

```
In [66]: 20<<4
```

```
Out[66]: 320
```

BITWIASE RIGHTSHIFT OPERATOR

```
In [67]: bin(10)
```

```
Out[67]: '0b1010'
```

```
In [68]: 10>>1
```

```
Out[68]: 5
```

```
In [69]: 10>>3
```

```
Out[69]: 1
```

```
In [70]: 3<<10
```

```
Out[70]: 3072
```

```
In [71]: 3>>100
```

```
Out[71]: 0
```

```
In [72]: bin(20)
```

```
Out[72]: '0b10100'
```

```
In [ ]:
```

import math Module

<https://docs.python.org/3/library/math.html>

```
In [106... x=sqrt(25)
```

```
-----  
NameError                                Traceback (most recent call last)  
Cell In[106], line 1  
----> 1 x=sqrt(25)  
  
NameError: name 'sqrt' is not defined
```

```
In [108... import math    # math is module
```

```
In [109... x=math.sqrt(25)  
x
```

```
Out[109... 5.0
```

```
In [110... x1=math.sqrt(30)
```

```
x1
```

```
Out[110...] 5.477225575051661
```

```
In [112...] x2=int(math.sqrt(30))  
x2
```

```
Out[112...] 5
```

```
In [113...] print(math.floor(2.9)) # floor- minimum or least value
```

```
2
```

```
In [114...] print(math.ceil(2.9)) # ceil- maximum or highest value
```

```
3
```

```
In [115...] print(math.pow(3,2))
```

```
9.0
```

```
In [117...] print(int(math.pow(3,2)))
```

```
9
```

```
In [118...] print(math.pi) #these values ar constant
```

```
3.141592653589793
```

```
In [119...] print(math.e) # these value are constant
```

```
2.718281828459045
```

```
In [120...] import math as m  
m.sqrt(24)
```

```
Out[120...] 4.898979485566356
```

```
In [121...] m.sqrt(25)
```

```
Out[121...] 5.0
```

```
In [122...] from math import sqrt,pow # math has many func if you want to call specific fun  
pow(3,2)
```

```
Out[122...] 9.0
```

```
In [123...] round(pow(3,2))
```

```
Out[123...] 9
```

```
In [124...] 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 $\geq x$.

`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 $k \leq n$ 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 evaluation 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 $0.5 \leq \text{abs}(m) < 1.0$.

`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.

$-\text{inf}$, inf and NaN behave similarly to the IEEE 754 Standard. That is, NaN is not close to anything, even itself. inf and $-\text{inf}$ 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 integer.

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 $k \leq n$ 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

```

In [125... x=input()
y=input()    # now here we console is open we can gie the values
z=x+y        # user can enter the values in that console
print(z)

```

2030

```

In [1]: x1=input("Enter your 1st number :")
y1=input("Enter your 2nd number :") #here it can't understand the arthematic op
z1=x1+y1
print(z1)

```

2030

```

In [3]: x1=input("Enter your 1st number :")
a1=int(x1)
y1=input("Enter your 2nd number :") # here we assign the a1 and b1 in place of
b1=int(y1)
z1=a1+b1
print(z1)

```

50

```

In [5]: x2=int(input("Enter the 1st number :")) #in the above code taking space or memo
y2=int(input("Enter the 2nd Number :")) # by using the int
z2=x2+y2
z2

```

Out[5]: 60

```

In [8]: ch=input('Enter a char :')
print(ch)

```

vinay

```

In [9]: print(ch[0])

```

v

```

In [10]: print(ch[1])

```

i

```

In [11]: print(len(ch))

```

5

```

In [12]: print(ch[-1])

```

y

```
In [14]: print(ch[1:])
```

inay

```
In [15]: print(ch[:-1])
```

vina

```
In [16]: ch=input("Enter a char:")[0]  
print(ch)
```

R

```
In [17]: ch=input("Enter a char:")[1:3]  
print(ch)
```

AN

```
In [20]: ch=input("Enter a char:")  
print(ch)  
print(len(ch))
```

linga

5

```
In [26]: result=eval(input("Enter an expr:"))  
print(result)  
print(float(result))
```

59

59.0

```
In [ ]:
```