

import math module

In [1]: `x=sqrt(25)`

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

In [2]: `import math`

In [3]: `x=math.sqrt(25)`
`x`

Out[3]: 5.0

In [4]: `x1=math.sqrt(15)`
`x1`

Out[4]: 3.872983346207417

In [25]: `math.floor(3.2)`

Out[25]: 3

In [5]: `print(math.floor(2.9))`

2

In [6]: `print(math.ceil(2.9))`

3

In [27]: `print(math.pow(3,2))`

9.0

In [8]: `3**2`

Out[8]: 9

In [30]: `math.pi`

Out[30]: 3.141592653589793

In [10]: `print(math.e)`

2.718281828459045

In [11]: `import math as m`

In [36]: `sqrt(10)`

Out[36]: 3.1622776601683795

In [34]: `round(pow(2,2))`

Out[34]: 4

In [13]: `from math import sqrt,pow`
`pow(2,3)`

Out[13]: 8.0

In [14]: `round(pow(2,3))`

Out[14]: 8

In [40]: `sqrt(225)`

Out[40]: 15.0

In [42]: `math.ceil(3.2)`

Out[42]: 4

In [38]: `math.floor(3.2)`

Out[38]: 3

In [15]: `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.

`-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 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)

In []:

In [16]: `from import *`

```
Cell In[16], line 1
  from import *
      ^
SyntaxError: invalid syntax
```

In [17]: `import*`

```
Cell In[17], line 1
  import*
      ^
SyntaxError: invalid syntax
```

In [18]: `from math import *`

In [19]: `import * from math`

```
Cell In[19], line 1
  import * from math
      ^
SyntaxError: invalid syntax
```

In [20]: `from math import *`

In [21]: `sqrt(25)`

Out[21]: 5.0

In [22]: `pow(3,3)`

Out[22]: 27.0

In [23]: `sin(60)`

Out[23]: -0.3048106211022167

In [24]: `cos(30)`

Out[24]: 0.15425144988758405

user input function in python || command line input


```
In [45]: x=input()  
y=input()  
z=x+y  
print(z)
```

naresh it

```
In [47]: x1=input('Enter the 1st number')  
x2=input('Enter the 2nd number')  
z1=x1+x2  
print(z1)
```

44

```
In [48]: type(x1)  
type(x2)
```

Out[48]: str

```
In [49]: x1=input('Enter the 1st number')  
a1=int(x1)  
y1=input('Enter the 2nd number')  
b1=int(y1)  
z1=a1+b1  
print(z1)
```

8

```
In [66]: x2=int(input('Enter the 1st number'))  
y2=int(input('Enter the 2nd number'))  
print(x2+y2)
```

8

```
In [51]: ch=input('enter a char')  
print(ch)
```

nit

```
In [52]: print(ch[0])
```

n

```
In [53]: print(ch[1])
```

i

```
In [54]: print(ch[-1])
```

t

```
In [71]: ch=input('enter a char')[0:2]  
print(ch)
```

ni

```
In [73]: ch[0:3]
```

Out[73]: 'ni'

```
In [67]: print(ch[1:4])
```

+6-

```
In [68]: print(ch[1:3])
```

+6

```
In [59]: ch[0:3]
```

```
Out[59]: 'n'
```

```
In [69]: print(ch[0:2])
```

2+

```
In [61]: ch=input('enter a char')[1:3]
         print(ch)
```

it

```
In [76]: ch=input('enter a char')
         print(ch)
```

2+2*-2

Eval function using input

```
In [64]: result=(eval(input('enter a expr')))
         print(result)
```

```
Cell In[64], line 1
      result=(eval(input('enter a expr')))
              ^
```

SyntaxError: '(' was never closed

```
In [65]: result=eval(input('enter a expr'))
         print(result)
```

6

```
In [ ]:
```