Math Module Functions in Python

- Python **math module** is defined as the most famous mathematical functions, which includes **trigonometric functions, representation functions, logarithmic functions**, etc.
- Furthermore, it also defines two mathematical constants, i.e., **Pie and Euler number**, etc.
- **Pie (n):** It is a well-known mathematical constant and defined as the ratio of circumstance to the diameter of a circle. Its value is 3.141592653589793.
- **Euler's number(e):**It is defined as the base of the natural logarithmic, and its value is 2.718281828459045.
- The **math** module has a set of methods and constants.

In [1]:

```
# Import math module and functions
import math
from math import *
```

In [2]:

```
# Many functions regarding math modules in python can be find using helf(math) method.
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, /)
```

acos() function

- Return the arc cosine (measured in radians) of x.
- The result is between 0 and pi.
- The parameter must be a double value between -1 and 1.

In [54]:

```
nlis = []
nlis.append(math.acos(1))
nlis.append(math.acos(-1))
print(nlis)
```

[0.0, 3.141592653589793]

acosh() function

- It is a built-in method defined under the math module to calculate the hyperbolic arc cosine of the given parameter in radians.
- For example, if x is passed as an acosh function (acosh(x)) parameter, it returns the hyperbolic arc cosine value.

In [56]:

```
print(math.acosh(1729))
```

8.148445582615551

asin() function

- Return the arc sine (measured in radians) of x.
- The result is between -pi/2 and pi/2.

In [52]:

```
nlis = []
nlis.append(math.asin(1))
nlis.append(math.asin(-1))
print(nlis)

```

[1.5707963267948966, -1.5707963267948966]

asinh() function

- Return the inverse hyperbolic sine of x.

In [55]:

```
print(math.asinh(1729))
```

8.1484457498709

atan() function

- Return the arc tangent (measured in radians) of x.
- The result is between -pi/2 and pi/2.

In [66]:

```
nlis = []
nlis.append(math.atan(math.inf))      # pozitive infinite
nlis.append(math.atan(-math.inf))     # negative infinite
print(nlis)
```

[1.5707963267948966, -1.5707963267948966]

atan2() function

- Return the arc tangent (measured in radians) of y/x.
- Unlike atan(y/x), the signs of both x and y are considered.

In [74]:

```
print(math.atan2(1729, 37))
print(math.atan2(1729, -37))
print(math.atan2(-1729, -37))
print(math.atan2(-1729, 37))
print(math.atan2(math.pi, math.inf))
print(math.atan2(math.inf, math.e))
print(math.atan2(math.tau, math.pi))
```

1.5493999395414435
1.5921927140483498
-1.5921927140483498
-1.5493999395414435
0.0
1.5707963267948966
1.1071487177940904

atanh() function

- Return the inverse hyperbolic tangent of x.

In [91]:

```
nlis=[]
nlis.append(math.atanh(-0.9999))
nlis.append(math.atanh(0))
nlis.append(math.atanh(0.9999))
print(nlis)

```

[-4.951718775643098, 0.0, 4.951718775643098]

ceil() function

- Rounds a number up to the nearest integer
- Returns the smalles integer greater than or equal to variable.

In [22]:

```
pi_number = math.pi       # math.pi is equal to pi_number 3.14.
print(f'The nearest integer greater than pi number is {math.ceil(pi_number)}.')
```

The nearest integer greater than pi number is 4.

comb() function

- Number of ways to choose k items from n items without repetition and without 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.

In [19]:

```
print(f'The combination of 6 with 2 is {math.comb(6, 2)}.')
print(math.comb(10, 3.14))       # It returns a TypeError
```

The combination of 6 with 2 is 15.

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_9084/1573976203.py in <module>
      1 print(f'The combination of 6 with 2 is {math.comb(6, 2)}.')
----> 2 print(math.comb(10, 3.14))       # It returns a TypeError

TypeError: 'float' object cannot be interpreted as an integer
```

copysign() function

- Returns a float consisting of the value of the first parameter and the sign of the second parameter.

In [18]:

```
print(f'The copysign of thes two numbers -3.14 and 2.718 is {math.copysign(-3.14, 2.718)}.')
print(f'The copysign of thes two numbers 1729 and -0.577 is {math.copysign(1729, -0.577)}.')
```

The copysign of thes two numbers -3.14 and 2.718 is 3.14.
The copysign of thes two numbers 1729 and -0.577 is -1729.0.

cos() function

- Return the cosine of x (measured in radians).

In [105]:

```
print(math.cos(0))
print(math.cos(math.pi/6))
print(math.cos(-1))
print(math.cos(1))
print(math.cos(1729))
print(math.cos(90))
```

```
1.0
0.8660254037844387
0.5403023058681398
0.5403023058681398
0.43204202084333315
-0.4480736161291701
```

cosh() function

- Return the hyperbolic cosine of x.

In [114]:

```
nlis = []
nlis.append(math.cosh(1))
nlis.append(math.cosh(0))
nlis.append(math.cosh(-5))
print(nlis)
```

```
[1.5430806348152437, 1.0, 74.20994852478785]
```

degrees() function

- Convert angle x from radians to degrees.

In [122]:

```
nlis = []
nlis.append(math.degrees(math.pi/2))
nlis.append(math.degrees(math.pi))
nlis.append(math.degrees(math.pi/4))
nlis.append(math.degrees(-math.pi))
print(nlis)
```

```
[90.0, 180.0, 45.0, -180.0]
```

dist() function

- 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)))**

In [127]:

```
print(math.dist([30], [60]))
print(math.dist([0.577, 1.618], [3.14, 2.718]))
x = [0.577, 1.618, 2.718]
y = [6, 28, 37]
print(math.dist(x, y))
```

```
30.0
2.7890803143688783
43.59672438383416
```

erf() function

- Error function at x.
- This method accepts a value between - inf and + inf, and returns a value between - 1 to + 1.

In [136]:

```
nlis = []
nlis.append(math.erf(math.inf))
nlis.append(math.erf(math.pi))
nlis.append(math.erf(math.e))
nlis.append(math.erf(math.tau))
nlis.append(math.erf(0))
nlis.append(math.erf(6))
nlis.append(math.erf(1.618))
nlis.append(math.erf(0.577))
nlis.append(math.erf(-math.inf))
print(nlis)
```

```
[1.0, 0.9999911238536323, 0.9998790689599072, 1.0, 0.0, 1.0, 0.9778739803135315, 0.585500565194
3818, -1.0]
```

erfc() function

- Complementary error function at x.
- This method accepts a value between - inf and + inf, and returns a value between 0 and 2.

In [137]:

```
nlis = []
nlis.append(math.erfc(math.inf))
nlis.append(math.erfc(math.pi))
nlis.append(math.erfc(math.e))
nlis.append(math.erfc(math.tau))
nlis.append(math.erfc(0))
nlis.append(math.erfc(6))
nlis.append(math.erfc(1.618))
nlis.append(math.erfc(0.577))
nlis.append(math.erfc(-math.inf))
print(nlis)
```

```
[0.0, 8.876146367641612e-06, 0.00012093104009276267, 6.348191705159502e-19, 1.0, 2.1519736712
498913e-17, 0.022126019686468514, 0.41449943480561824, 2.0]
```

exp() function

- The **math.exp()** method returns **E** raised to the power of x (Ex).
- **E** is the base of the natural system of logarithms (approximately 2.718282) and x is the number passed to it.

In [139]:

```
nlis = []
nlis.append(math.exp(math.inf))
nlis.append(math.exp(math.pi))
nlis.append(math.exp(math.e))
nlis.append(math.exp(math.tau))
nlis.append(math.exp(0))
nlis.append(math.exp(6))
nlis.append(math.exp(1.618))
nlis.append(math.exp(0.577))
nlis.append(math.exp(-math.inf))
print(nlis)
```

```
[inf, 23.140692632779267, 15.154262241479262, 535.4916555247646, 1.0, 403.4287934927351, 5.042
994235377287, 1.780688344599613, 0.0]
```

expm1() function

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

In [141]:

```
nlis = []
nlis.append(math.expm1(math.inf))
nlis.append(math.expm1(math.pi))
nlis.append(math.expm1(math.e))
nlis.append(math.expm1(math.tau))
nlis.append(math.expm1(0))
nlis.append(math.expm1(6))
nlis.append(math.expm1(1.618))
nlis.append(math.expm1(0.577))
nlis.append(math.expm1(-math.inf))
print(nlis)
```

[inf, 22.140692632779267, 14.154262241479262, 534.4916555247646, 0.0, 402.4287934927351, 4.042
994235377287, 0.7806883445996128, 6.38905609893065, -1.0]

fabs() function

- Returns the absolute value of a number

In [14]:

```
print(f'The absolute value of the number -1.618 is {math.fabs(-1.618)}.')
```

The absolute value of the number -1.618 is 1.618.

factorial() function

- Returns the factorial of a number.

In [28]:

```
print(f'The factorial of the number 6 is {math.factorial(6)}.')
```

The factorial of the number 6 is 720.

In [29]:

```
# Factorial of negative numbers returns a ValueError.
print(math.factorial(-6))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_9084/3052214312.py in <module>
      1 # Factorial of negative numbers returns a ValueError.
----> 2 print(math.factorial(-6))

ValueError: factorial() not defined for negative values
```

In [30]:

```
# Factorial of non-unteger numbers returns a TypeError.
print(math.factorial(3.14))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_9084/3264451390.py in <module>
      1 # Factorial of non-unteger numbers returns a TypeError.
----> 2 print(math.factorial(3.14))

TypeError: 'float' object cannot be interpreted as an integer
```

floor() functions:

- Rounds a number down to the nearest integer

In [34]:

```
print(math.floor(3.14))
```

```
3
```

fmod() function

- Returns the remainder of x/y

In [37]:

```
print(math.fmod(37, 6))
print(math.fmod(1728, 37))
```

```
1.0
26.0
```

frexp() function

- Returns the mantissa and the exponent, of a specified number

In [31]:

```
print(math.frexp(2.718))
```

```
(0.6795, 2)
```

fsum() function

- Returns the sum of all items in any iterable (tuples, arrays, lists, etc.)

In [142]:

```
special_nums = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
print(math.fsum(special_nums))
```

1808.053

gamma() function

- Returns the gamma function at x.
- You can find more information about **gamma function** from this Link. (https://en.wikipedia.org/wiki/Gamma_function)

In [143]:

```
print(math.gamma(3.14))
print(math.gamma(6))
print(math.gamma(2.718))
```

2.2844806338178008
120.0
1.5671127417668826

gcd() function

- Returns the greatest common divisor of two integers

In [144]:

```
print(math.gcd(3, 10))
print(math.gcd(4, 8))
print(math.gcd(0, 0))
```

1
4
0

hypot() function

- Returns the Euclidean norm.
- 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* + *yy*).

In [148]:

```
print(math.hypot(3, 4))
print(math.hypot(5, 12))
print(math.hypot(8, 15))
```

```
5.0
13.0
17.0
```

isclose() function

- It checks whether two values are close to each other, or not.
- Returns True if the values are close, otherwise False.
- This method uses a relative or absolute tolerance, to see if the values are close.
- **Tip:** It uses the following formula to compare the values: abs(a-b) <= max(rel_tol * max(abs(a), abs(b)), abs_tol)

In [11]:

```
print(math.isclose(math.pi, math.tau))     # tau number is 2 times higher than pi number
print(math.isclose(3.14, 2.718))
print(math.isclose(3.14, 1.618))
print(math.isclose(10, 5, rel_tol = 3, abs_tol=0))
print(math.isclose(3.14, 3.1400000000001))
```

```
False
False
False
True
True
```

isfinite() function

- Return *True* if x is neither an **infinity** nor a **NaN**, and *False* otherwise.

In [155]:

```
nlis = []
nlis.append(math.isfinite(math.inf))
nlis.append(math.isfinite(math.pi))
nlis.append(math.isfinite(math.e))
nlis.append(math.isfinite(math.tau))
nlis.append(math.isfinite(0))
nlis.append(math.isfinite(6))
nlis.append(math.isfinite(1.618))
nlis.append(math.isfinite(0.577))
nlis.append(math.isfinite(-math.inf))
nlis.append(math.isfinite(float('NaN')))
nlis.append(math.isfinite(float('inf')))
print(nlis)
```

```
[False, True, True, True, True, True, True, True, False, False, False]
```

isinf() function

- Return *True* if x is a **positive or negative infinity**, and *False* otherwise.

In [161]:

```
nlis = []
nlis.append(math.isinf(math.inf))
nlis.append(math.isinf(math.pi))
nlis.append(math.isinf(math.e))
nlis.append(math.isinf(math.tau))
nlis.append(math.isinf(0))
nlis.append(math.isinf(6))
nlis.append(math.isinf(1.618))
nlis.append(math.isinf(0.577))
nlis.append(math.isinf(-math.inf))
print(nlis)
```

[True, False, False, False, False, False, False, False, True]

isnan() function

- Return *True* if x is a **NaN (not a number)**, and *False* otherwise.

In [162]:

```
nlis = []
nlis.append(math.isnan(float('NaN')))
nlis.append(math.isnan(math.inf))
nlis.append(math.isnan(math.pi))
nlis.append(math.isnan(math.e))
nlis.append(math.isnan(math.tau))
nlis.append(math.isnan(0))
nlis.append(math.isnan(6))
nlis.append(math.isnan(1.618))
nlis.append(math.isnan(0.577))
nlis.append(math.isnan(-math.inf))
nlis.append(math.isnan(math.nan))
print(nlis)
```

[True, False, False, False, False, False, False, False, False, False, True]

isqrt() function

- Rounds a square root number downwards to the nearest integer.
- The returned square root value is the floor value of square root of a non-negative integer number.
- It gives a **ValueError** and **TypeError** when **a negative integer number** and **a float number** are used, respectively.

In [15]:

```
print(math.isqrt(4))
print(math.isqrt(5))
print(math.isqrt(-5))
```

```
2
2

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_20068/2393595883.py in <module>
      1 print(math.isqrt(4))
      2 print(math.isqrt(5))
----> 3 print(math.isqrt(-5))

ValueError: isqrt() argument must be nonnegative
```

In [16]:

```
print(math.isqrt(3.14))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_20068/4116779010.py in <module>
----> 1 print(math.isqrt(3.14))

TypeError: 'float' object cannot be interpreted as an integer
```

lcm() function

- Least Common Multiple.

In [168]:

```
nlis = []
nlis.append(math.lcm(3, 5, 25))
nlis.append(math.lcm(9, 6, 27))
nlis.append(math.lcm(21, 27, 54))
print(nlis)
```

```
[75, 54, 378]
```

ldexp() function

- Returns the inverse of **math.frexp()** which is x*(2^i) of the given numbers x and i

In [19]:

```
print(math.ldexp(20, 4))
print(20*(2**4))
```

320.0
320

lgamma() function

- Returns the log gamma value of x

In [26]:

```
print(math.gamma(6))
print(math.lgamma(6))
print(math.log(120))      # print(math.gamma(6)) = 120
```

120.0
4.787491742782047
4.787491742782046

log() function

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

In [174]:

```
nlis = []
nlis.append(math.log(90))
nlis.append(math.log(1))
nlis.append(math.log(math.e))
nlis.append(math.log(math.pi))
nlis.append(math.log(math.tau))
nlis.append(math.log(math.inf))
nlis.append(math.log(math.nan))
print(nlis)
```

[4.499809670330265, 0.0, 1.0, 1.1447298858494002, 1.8378770664093453, inf, nan]

log10() function

- Return the base 10 logarithm of x.

In [177]:

```
nlis = []
nlis.append(math.log10(90))
nlis.append(math.log10(1))
nlis.append(math.log10(math.e))
nlis.append(math.log10(math.pi))
nlis.append(math.log10(math.tau))
nlis.append(math.log10(math.inf))
nlis.append(math.log10(math.nan))
print(nlis)
```

[1.954242509439325, 0.0, 0.4342944819032518, 0.49714987269413385, 0.798179868358115, inf, nan]

log1p() function

- Return the natural logarithm of 1+x (base e).

In [179]:

```
nlis = []
nlis.append(math.log1p(90))
nlis.append(math.log1p(1))
nlis.append(math.log1p(math.e))
nlis.append(math.log1p(math.pi))
nlis.append(math.log1p(math.tau))
nlis.append(math.log1p(math.inf))
nlis.append(math.log1p(math.nan))
print(nlis)
```

[4.51085950651685, 0.6931471805599453, 1.3132616875182228, 1.4210804127942926, 1.985568308
7099187, inf, nan]

log2() function

- Return the base 2 logarithm of x.

In [183]:

```
nlis = []
nlis.append(math.log2(90))
nlis.append(math.log2(2))
nlis.append(math.log2(1))
nlis.append(math.log2(math.e))
nlis.append(math.log2(math.pi))
nlis.append(math.log2(math.tau))
nlis.append(math.log2(math.inf))
nlis.append(math.log2(math.nan))
print(nlis)
```

[6.491853096329675, 1.0, 0.0, 1.4426950408889634, 1.6514961294723187, 2.651496129472319, inf, n
an]

modf() function

- It returns the frwactional and integer parts of the certain number. Both the outputs carry the sign of x and are of type float.

In [29]:

```
print(math.modf(math.pi))
print(math.modf(math.e))
print(math.modf(1.618))
```

(0.14159265358979312, 3.0)
(0.7182818284590451, 2.0)
(0.6180000000000001, 1.0)

nextafter() function

- Return the next floating-point value after x towards y.
- if x is equal to y then y is returned.

In [191]:

```
nlis = []
nlis.append(math.nextafter(3.14, 90))
nlis.append(math.nextafter(6, 2.718))
nlis.append(math.nextafter(3, math.e))
nlis.append(math.nextafter(28, math.inf))
nlis.append(math.nextafter(1.618, math.nan))
nlis.append(math.nextafter(1, 1))
nlis.append(math.nextafter(0, 0))
print(nlis)
```

[3.1400000000000006, 5.999999999999999, 2.9999999999999996, 28.000000000000004, nan, 1.0, 0.
0]

perm() function

- Returns the number of ways to choose k items from n items with order and without repetition.

In [31]:

```
print(math.perm(6, 2))
print(math.perm(6, 6))
```

30
720

pow() function

- Returns the value of x to the power of y.

In [34]:

```
print(math.pow(10, 2))
print(math.pow(math.pi, math.e))
```

100.0
22.45915771836104

prod() function

- Returns the product of all the elements in an iterable

In [32]:

```
special_nums = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
print(math.prod(special_nums))
```

85632659.07026622

radians() function

- Convert angle x from degrees to radians.

In [193]:

```
nlis = []
nlis.append(math.radians(0))
nlis.append(math.radians(30))
nlis.append(math.radians(45))
nlis.append(math.radians(60))
nlis.append(math.radians(90))
nlis.append(math.radians(120))
nlis.append(math.radians(180))
nlis.append(math.radians(270))
nlis.append(math.radians(360))
print(nlis)
```

[0.0, 0.5235987755982988, 0.7853981633974483, 1.0471975511965976, 1.5707963267948966, 2.0943
951023931953, 3.141592653589793, 4.71238898038469, 6.283185307179586]

remainder() function

- Difference between x and the closest integer multiple of y.
- Return x - n*y* *where* *n*y is the closest integer multiple of y.
- ReturnIn 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.

In [196]:

```
nlis = []
nlis.append(math.remainder(3.14, 2.718))
nlis.append(math.remainder(6, 28))
nlis.append(math.remainder(5, 3))
nlis.append(math.remainder(1729, 37))
print(nlis)
```

[0.42200000000000015, 6.0, -1.0, -10.0]

sin() function

- Return the sine of x (measured in radians).
- **Note:** To find the sine of degrees, it must first be converted into radians with the **math.radians() method.**

In [204]:

```
nlis = []
nlis.append(math.sin(math.pi))
nlis.append(math.sin(math.pi/2))
nlis.append(math.sin(math.e))
nlis.append(math.sin(math.nan))
nlis.append(math.sin(math.tau))
nlis.append(math.sin(30))
nlis.append(math.sin(-5))
nlis.append(math.sin(37))
print(nlis)
```

[1.2246467991473532e-16, 1.0, 0.41078129050290885, nan, -2.4492935982947064e-16, -0.988031624
0928618, 0.9589242746631385, -0.6435381333569995]

sinh() function

- Return the hyperbolic sine of x.

In [213]:

```
nlis = []
nlis.append(math.sinh(1))
nlis.append(math.sinh(0))
nlis.append(math.sinh(-5))
nlis.append(math.sinh(math.pi))
nlis.append(math.sinh(math.e))
nlis.append(math.sinh(math.tau))
nlis.append(math.sinh(math.nan))
nlis.append(math.sinh(math.inf))
print(nlis)
```

[1.1752011936438014, 0.0, -74.20321057778875, 11.548739357257746, 7.544137102816975, 267.744
89404101644, nan, inf]

sqrt() function

- Return the square root of x.

In [210]:

```
nlis = []
nlis.append(math.sqrt(1))
nlis.append(math.sqrt(0))
nlis.append(math.sqrt(37))
nlis.append(math.sqrt(math.pi))
nlis.append(math.sqrt(math.e))
nlis.append(math.sqrt(math.tau))
nlis.append(math.sqrt(math.nan))
nlis.append(math.sqrt(math.inf))
print(nlis)
```

```
[1.0, 0.0, 6.082762530298219, 1.7724538509055159, 1.6487212707001282, 2.5066282746310002, na
n, inf]
```

tan() function

- Return the tangent of x (measured in radians).

In [212]:

```
nlis = []
nlis.append(math.tan(0))
nlis.append(math.tan(30))
nlis.append(math.tan(45))
nlis.append(math.tan(60))
nlis.append(math.tan(90))
nlis.append(math.tan(120))
nlis.append(math.tan(180))
nlis.append(math.tan(270))
nlis.append(math.tan(360))
print(nlis)
```

```
[0.0, -6.405331196646276, 1.6197751905438615, 0.320040389379563, -1.995200412208242, 0.71312
30097859091, 1.3386902103511544, -0.17883906379845224, -3.380140413960958]
```

tanh() function

- Return the hyperbolic tangent of x.

In [214]:

```
nlis = []
nlis.append(math.tanh(1))
nlis.append(math.tanh(0))
nlis.append(math.tanh(-5))
nlis.append(math.tanh(math.pi))
nlis.append(math.tanh(math.e))
nlis.append(math.tanh(math.tau))
nlis.append(math.tanh(math.nan))
nlis.append(math.tanh(math.inf))
print(nlis)
```

```
[0.7615941559557649, 0.0, -0.9999092042625951, 0.99627207622075, 0.9913289158005998, 0.99999
30253396107, nan, 1.0]
```

trunc() function

- Truncates the Real x to the nearest Integral toward 0.
- Returns the truncated integer parts of different numbers

In [218]:

```
nlis = []
nlis.append(math.trunc(1))
nlis.append(math.trunc(0))
nlis.append(math.trunc(-5))
nlis.append(math.trunc(0.577))
nlis.append(math.trunc(1.618))
nlis.append(math.trunc(math.pi))
nlis.append(math.trunc(math.e))
nlis.append(math.trunc(math.tau))
print(nlis)
```

```
[1, 0, -5, 0, 1, 3, 2, 6]
```

ulp() function

- Return the value of the least significant bit of the float x.

In [224]:

```
import sys
nlis = []
nlis.append(math.ulp(1))
nlis.append(math.ulp(0))
nlis.append(math.ulp(-5))
nlis.append(math.ulp(0.577))
nlis.append(math.ulp(1.618))
nlis.append(math.ulp(math.pi))
nlis.append(math.ulp(math.e))
nlis.append(math.ulp(math.tau))
nlis.append(math.ulp(math.nan))
nlis.append(math.ulp(math.inf))
nlis.append(math.ulp(-math.inf))
nlis.append(math.ulp(float('nan')))
nlis.append(math.ulp(float('inf')))
x = sys.float_info.max
nlis.append(math.ulp(x))
print(nlis)
```

```
[2.220446049250313e-16, 5e-324, 8.881784197001252e-16, 1.1102230246251565e-16, 2.2204460492
50313e-16, 4.440892098500626e-16, 4.440892098500626e-16, 8.881784197001252e-16, nan, inf, inf, n
an, inf, 1.99584030953472e+292]
```