Bit(Binary number system)

```
In [ ]:   '''
          binary (base 2)  -- 0& 1 (0b)
          octal (base 8) -- 1,2,3,4,5,6,7(0o)
          decimal (base 10)--0x
          hexadecimal (base 16) -- 0x a-10,b-11,c-12,d-13,e-14,f-15 (0-9 a-f)
          '''
```

```
In [2]:   25
```

```
Out[2]:   25
```

```
In [3]:   bin(25)
```

```
Out[3]:   '0b11001'
```

```
In [5]:   int(0b11001)
```

```
Out[5]:   25
```

```
In [6]:   oct(25)
```

```
Out[6]:   '0o31'
```

```
In [7]:   hex(25)
```

```
Out[7]:   '0x19'
```

```
In [8]:   hex(9)
```

```
Out[8]:   '0x9'
```

```
In [ ]:   # how to swap 2 variable in python
```

```
In [9]:   a=5
          b=6
```

```
In [10]:  temp =a
          a=b
          b=temp
          print(a)
          print(b)

          6
          5
```

```
In [11]:  a2=5
          b2=6
```

```
In [12]:  a2=a2+b2
          b2=a2-b2
          a2=a2-b2
```

In [13]:
```python
print(a2)
print(b2)
```

6
5

In [14]:
```python
a1 =7
b1=8
```

In [15]:
```python
a1,b1=b1,a1
```

In [41]:
```python
print(a1)
print(b1)
```

8
7

In [33]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Bitwise operator

complement varibale ~

and &

or |

xor (exclusive or ^)

left shift <<

right shift >>

```
In [17]:   ~12
```

```
Out[17]:   -13
```

```
In [18]:   ~45
```

```
Out[18]:   -46
```

```
In [19]:   bin(12)
```

```
Out[19]:   '0b1100'
```

```
In [20]:   bin(13)
```

```
Out[20]:   '0b1101'
```

```
In [22]:   12 & 13   #  1 1 0 0
                     #  1 1 0 1
                     #& 1 1 0 0 = 12
```

```
Out[22]:   12
```

```
In [24]:   12 | 13   #   1 1 0 0
                     #   1 1 0 1
                     # | 1 1 0 1 = 13
```

```
Out[24]:   13
```

```
In [26]:   12 ^ 13  #   1 1 0 0
                    #   1 1 0 1
                    # ^ 0 0 0 1 =
```

```
Out[26]:   1
```

```
In [27]:   12 << 2  #   1 1 0 0 0 0   we are ganing the bit
```

```
Out[27]:   48
```

```
In [29]:   13 << 3 # 1101000
```

```
Out[29]:   104
```

```
In [30]:   12>>2  # 1100    we will lossing the bit  11
```

```
Out[30]:   3
```

```
In [31]:   13>> 1  # 1101 -> 110
```

```
Out[31]:   6
```

```
In [ ]:
```

In [ ]:

In [42]:
```python
x=sqrt(25)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[42], line 1
----> 1 x=sqrt(25)

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

In [44]:
```python
import math
math.sqrt(25)
```

Out[44]: 5.0

In [45]:
```python
x1 = math.sqrt(15)
```

In [46]:
```python
math(help)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[46], line 1
----> 1 math(help)

TypeError: 'module' object is not callable
```

In [47]:
```python
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!.

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

    This is the largest integer <= x.

fma(x, y, z, /)
    Fused multiply-add operation.

    Compute (x * y) + z with a single round.
```

```
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)
```

In [48]:
```python
# floor & ceil
print(math.floor(3.87))
```

```
3
```

In [49]:
```python
print(math.ceil(3.87))
```

```
4
```

In [50]:
```python
round(3.87)
```

Out[50]: 4

In [51]:
```python
round(3.37)
```

Out[51]: 3

In [52]:
```python
3*2
```

Out[52]: 6

In [53]:
```python
3**2
```

Out[53]: 9

In [54]:
```python
print(math.pow(3,2))
```

```
9.0
```

In [55]:
```python
math.pi
```

Out[55]: 3.141592653589793

In [56]:
```python
pi=5.89
```

In [57]:
```python
pi #  python we don't  have concet called constant& we have concepy only variabl
```

Out[57]: 5.89

In [58]:
```python
math.e
```

Out[58]: 2.71828182845904

In [59]:
```python
math.E
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[59], line 1
----> 1 math.E

AttributeError: module 'math' has no attribute 'E'
```

In [61]:
```python
import math as m

m.pow(3,2)
```

Out[61]: 9.0

In [62]:
```python
import keyword
keyword.kwlist
```

Out[62]:
```
['False',
 'None',
 'True',
 'and',
 'as',
 'assert',
 'async',
 'await',
 'break',
 'class',
 'continue',
 'def',
 'del',
 'elif',
 'else',
 'except',
 'finally',
 'for',
 'from',
 'global',
 'if',
 'import',
 'in',
 'is',
 'lambda',
 'nonlocal',
 'not',
 'or',
 'pass',
 'raise',
 'return',
 'try',
 'while',
 'with',
 'yield']
```

In [63]:
```python
from math import sqrt,pow,ceil,floor
print(pow(2,3))
print(m.sqrt(10))
print(m.floor(3.5))
print(m.ceil(3.6))
```

```
8.0
3.1622776601683795
3
4
```

In [64]:
```python
from math import *
print(pow(2,3))
print(m.sqrt(10))
print(m.floor(3.5))
print(m.ceil(3.6))
```

```
8.0
3.1622776601683795
3
4
```

In [ ]:
```python
# input function in python
```

In [65]:
```python
x = input()
x
```

Out[65]: '2'

In [ ]:
```python
type(x)
```

In [67]:
```python
x=input()
y=input()
z=x+y
print(z)
```

```
23
```

In [69]:
```python
x=int(input())   # input string convert to int
y=int(input())
z=x+y
print(z)
```

```
5
```

In [70]:
```python
x1 = int(input('Enter the 1st number'))
y1 = int(input('Enter the 2nd number'))
z = x1+y1
print(z)
```

```
9
```

In [71]:
```python
st = input('enter a string')
print(st)
```

```
Hello
```

In [72]:
```python
print(st[0])
```

```
H
```

In [73]:
```python
print(st[0:3])
```

```
Hel
```

In [74]:
```python
st = input('enter a string')[1]
print(st)
```

i

```
In [75]: st = input('enter a string')[5:8]
         print(st)
```

hit

```
In [76]: st = input('enter a string')
         print(st)
```

2+3-1*4

```
In [77]: st = eval(input('enter a string'))
         print(st)
```

1

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```