Random Module

In [2]:
```
import random
```

In [2]:
```
random.__dir__()
```

Out [2]:
```
['__name__',
 '__doc__',
 '__package__',
 '__loader__',
 '__spec__',
 '__file__',
 '__cached__',
 '__builtins__',
 '_warn',
 '_log',
 '_exp',
 '_pi',
 '_e',
 '_ceil',
 '_sqrt',
 '_acos',
 '_cos',
 '_sin',
 'TWOPI',
 '_floor',
 '_isfinite',
 '_urandom',
 '_Set',
 '_Sequence',
 '_index',
 '_accumulate',
 '_repeat',
 '_bisect',
 '_os',
 '_random',
 '_sha512',
 '__all__',
 'NV_MAGICCONST',
 'LOG4',
 'SG_MAGICCONST',
 'BPF',
 'RECIP_BPF',
 '_ONE',
 'Random',
 'SystemRandom',
 '_inst',
 'seed',
 'random',
 'uniform',
 'triangular',
 'randint',
 'choice',
 'randrange',
 'sample',
 'shuffle',
 'choices',
 'normalvariate',
 'lognormvariate',
 'expovariate',
 'vonmisesvariate',
 'gammavariate',
 'gauss',
 'betavariate',
 'paretovariate',
 'weibullvariate',
 'getstate',
 'setstate',
 'getrandbits',
 'randbytes',
 '_test_generator',
 '_test']
```

In [3]:
```
help(random) #daha okunaklı
```

```
Help on module random:

NAME
    random - Random variable generators.

MODULE REFERENCE
    https://docs.python.org/3.10/library/random.html

    The following documentation is automatically generated from the Python
    source files.  It may be incomplete, incorrect or include features that
    are considered implementation detail and may vary between Python
    implementations.  When in doubt, consult the module reference at the
    location listed above.

DESCRIPTION
        bytes
        -----
               uniform bytes (values between 0 and 255)

        integers
        --------
               uniform within range

        sequences
        ---------
               pick random element
               pick random sample
               pick weighted random sample
               generate random permutation

        distributions on the real line:
        ------------------------------
               uniform
               triangular
               normal (Gaussian)
               lognormal
               negative exponential
               gamma
               beta
               pareto
               Weibull

        distributions on the circle (angles 0 to 2pi)
        ---------------------------------------------
               circular uniform
               von Mises

    General notes on the underlying Mersenne Twister core generator:

    * The period is 2**19937-1.
    * It is one of the most extensively tested generators in existence.
    * The random() method is implemented in C, executes in a single Python step,
      and is, therefore, threadsafe.

CLASSES
    _random.Random(builtins.object)
        Random
            SystemRandom

    class Random(_random.Random)
     |  Random(x=None)
     |
     |  Random number generator base class used by bound module functions.
     |
     |  Used to instantiate instances of Random to get generators that don't
     |  share state.
     |
     |  Class Random can also be subclassed if you want to use a different basic
     |  generator of your own devising: in that case, override the following
     |  methods:  random(), seed(), getstate(), and setstate().
     |  Optionally, implement a getrandbits() method so that randrange()
     |  can cover arbitrarily large ranges.
     |
     |  Method resolution order:
     |      Random
     |      _random.Random
     |      builtins.object
     |
     |  Methods defined here:
     |
     |  __getstate__(self)
     |      # Issue 17489: Since __reduce__ was defined to fix #759889 this is no
     |      # longer called; we leave it here because it has been here since random was
     |      # rewritten back in 2001 and why risk breaking something.
     |
     |  __init__(self, x=None)
```

```
|      Initialize an instance.
|
|      Optional argument x controls seeding, as for Random.seed().
|
|  __reduce__(self)
|      Helper for pickle.
|
|  __setstate__(self, state)
|
|  betavariate(self, alpha, beta)
|      Beta distribution.
|
|      Conditions on the parameters are alpha > 0 and beta > 0.
|      Returned values range between 0 and 1.
|
|  choice(self, seq)
|      Choose a random element from a non-empty sequence.
|
|  choices(self, population, weights=None, *, cum_weights=None, k=1)
|      Return a k sized list of population elements chosen with replacement.
|
|      If the relative weights or cumulative weights are not specified,
|      the selections are made with equal probability.
|
|  expovariate(self, lambd)
|      Exponential distribution.
|
|      lambd is 1.0 divided by the desired mean.  It should be
|      nonzero.  (The parameter would be called "lambda", but that is
|      a reserved word in Python.)  Returned values range from 0 to
|      positive infinity if lambd is positive, and from negative
|      infinity to 0 if lambd is negative.
|
|  gammavariate(self, alpha, beta)
|      Gamma distribution.  Not the gamma function!
|
|      Conditions on the parameters are alpha > 0 and beta > 0.
|
|      The probability distribution function is:
|
|                  x ** (alpha - 1) * math.exp(-x / beta)
|        pdf(x) =  --------------------------------------
|                    math.gamma(alpha) * beta ** alpha
|
|  gauss(self, mu, sigma)
|      Gaussian distribution.
|
|      mu is the mean, and sigma is the standard deviation.  This is
|      slightly faster than the normalvariate() function.
|
|      Not thread-safe without a lock around calls.
|
|  getstate(self)
|      Return internal state; can be passed to setstate() later.
|
|  lognormvariate(self, mu, sigma)
|      Log normal distribution.
|
|      If you take the natural logarithm of this distribution, you'll get a
|      normal distribution with mean mu and standard deviation sigma.
|      mu can have any value, and sigma must be greater than zero.
|
|  normalvariate(self, mu, sigma)
|      Normal distribution.
|
|      mu is the mean, and sigma is the standard deviation.
|
|  paretovariate(self, alpha)
|      Pareto distribution.  alpha is the shape parameter.
|
|  randbytes(self, n)
|      Generate n random bytes.
|
|  randint(self, a, b)
|      Return random integer in range [a, b], including both end points.
|
|  randrange(self, start, stop=None, step=1)
|      Choose a random item from range(start, stop[, step]).
|
|      This fixes the problem with randint() which includes the
|      endpoint; in Python this is usually not what you want.
|
|  sample(self, population, k, *, counts=None)
|      Chooses k unique random elements from a population sequence or set.
|
|      Returns a new list containing elements from the population while
```

```
|      leaving the original population unchanged.  The resulting list is
|      in selection order so that all sub-slices will also be valid random
|      samples.  This allows raffle winners (the sample) to be partitioned
|      into grand prize and second place winners (the subslices).
|
|      Members of the population need not be hashable or unique.  If the
|      population contains repeats, then each occurrence is a possible
|      selection in the sample.
|
|      Repeated elements can be specified one at a time or with the optional
|      counts parameter.  For example:
|
|          sample(['red', 'blue'], counts=[4, 2], k=5)
|
|      is equivalent to:
|
|          sample(['red', 'red', 'red', 'red', 'blue', 'blue'], k=5)
|
|      To choose a sample from a range of integers, use range() for the
|      population argument.  This is especially fast and space efficient
|      for sampling from a large population:
|
|          sample(range(10000000), 60)
|
|  seed(self, a=None, version=2)
|      Initialize internal state from a seed.
|
|      The only supported seed types are None, int, float,
|      str, bytes, and bytearray.
|
|      None or no argument seeds from current time or from an operating
|      system specific randomness source if available.
|
|      If *a* is an int, all bits are used.
|
|      For version 2 (the default), all of the bits are used if *a* is a str,
|      bytes, or bytearray.  For version 1 (provided for reproducing random
|      sequences from older versions of Python), the algorithm for str and
|      bytes generates a narrower range of seeds.
|
|  setstate(self, state)
|      Restore internal state from object returned by getstate().
|
|  shuffle(self, x, random=None)
|      Shuffle list x in place, and return None.
|
|      Optional argument random is a 0-argument function returning a
|      random float in [0.0, 1.0); if it is the default None, the
|      standard random.random will be used.
|
|  triangular(self, low=0.0, high=1.0, mode=None)
|      Triangular distribution.
|
|      Continuous distribution bounded by given lower and upper limits,
|      and having a given mode value in-between.
|
|      http://en.wikipedia.org/wiki/Triangular_distribution
|
|  uniform(self, a, b)
|      Get a random number in the range [a, b) or [a, b] depending on rounding.
|
|  vonmisesvariate(self, mu, kappa)
|      Circular data distribution.
|
|      mu is the mean angle, expressed in radians between 0 and 2*pi, and
|      kappa is the concentration parameter, which must be greater than or
|      equal to zero.  If kappa is equal to zero, this distribution reduces
|      to a uniform random angle over the range 0 to 2*pi.
|
|  weibullvariate(self, alpha, beta)
|      Weibull distribution.
|
|      alpha is the scale parameter and beta is the shape parameter.
|
|  ----------------------------------------------------------------------
|  Class methods defined here:
|
|  __init_subclass__(**kwargs) from builtins.type
|      Control how subclasses generate random integers.
|
|      The algorithm a subclass can use depends on the random() and/or
|      getrandbits() implementation available to it and determines
|      whether it can generate random integers from arbitrarily large
|      ranges.
|
|  ----------------------------------------------------------------------
```

```
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  VERSION = 3
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from _random.Random:
 |
 |  getrandbits(self, k, /)
 |      getrandbits(k) -> x.  Generates an int with k random bits.
 |
 |  random(self, /)
 |      random() -> x in the interval [0, 1).
 |
 |  ----------------------------------------------------------------------
 |  Static methods inherited from _random.Random:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.

class SystemRandom(Random)
 |  SystemRandom(x=None)
 |
 |  Alternate random number generator using sources provided
 |  by the operating system (such as /dev/urandom on Unix or
 |  CryptGenRandom on Windows).
 |
 |   Not available on all systems (see os.urandom() for details).
 |
 |  Method resolution order:
 |      SystemRandom
 |      Random
 |      _random.Random
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  getrandbits(self, k)
 |      getrandbits(k) -> x.  Generates an int with k random bits.
 |
 |  getstate = _notimplemented(self, *args, **kwds)
 |
 |  randbytes(self, n)
 |      Generate n random bytes.
 |
 |  random(self)
 |      Get the next random number in the range [0.0, 1.0).
 |
 |  seed(self, *args, **kwds)
 |      Stub method.  Not used for a system random number generator.
 |
 |  setstate = _notimplemented(self, *args, **kwds)
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from Random:
 |
 |  __getstate__(self)
 |      # Issue 17489: Since __reduce__ was defined to fix #759889 this is no
 |      # longer called; we leave it here because it has been here since random was
 |      # rewritten back in 2001 and why risk breaking something.
 |
 |  __init__(self, x=None)
 |      Initialize an instance.
 |
 |      Optional argument x controls seeding, as for Random.seed().
 |
 |  __reduce__(self)
 |      Helper for pickle.
 |
 |  __setstate__(self, state)
 |
 |  betavariate(self, alpha, beta)
 |      Beta distribution.
 |
 |      Conditions on the parameters are alpha > 0 and beta > 0.
 |      Returned values range between 0 and 1.
 |
 |  choice(self, seq)
```

```
|      Choose a random element from a non-empty sequence.
|
|  choices(self, population, weights=None, *, cum_weights=None, k=1)
|      Return a k sized list of population elements chosen with replacement.
|
|      If the relative weights or cumulative weights are not specified,
|      the selections are made with equal probability.
|
|  expovariate(self, lambd)
|      Exponential distribution.
|
|      lambd is 1.0 divided by the desired mean.  It should be
|      nonzero.  (The parameter would be called "lambda", but that is
|      a reserved word in Python.)  Returned values range from 0 to
|      positive infinity if lambd is positive, and from negative
|      infinity to 0 if lambd is negative.
|
|  gammavariate(self, alpha, beta)
|      Gamma distribution.  Not the gamma function!
|
|      Conditions on the parameters are alpha > 0 and beta > 0.
|
|      The probability distribution function is:
|
|                  x ** (alpha - 1) * math.exp(-x / beta)
|        pdf(x) =  --------------------------------------
|                    math.gamma(alpha) * beta ** alpha
|
|  gauss(self, mu, sigma)
|      Gaussian distribution.
|
|      mu is the mean, and sigma is the standard deviation.  This is
|      slightly faster than the normalvariate() function.
|
|      Not thread-safe without a lock around calls.
|
|  lognormvariate(self, mu, sigma)
|      Log normal distribution.
|
|      If you take the natural logarithm of this distribution, you'll get a
|      normal distribution with mean mu and standard deviation sigma.
|      mu can have any value, and sigma must be greater than zero.
|
|  normalvariate(self, mu, sigma)
|      Normal distribution.
|
|      mu is the mean, and sigma is the standard deviation.
|
|  paretovariate(self, alpha)
|      Pareto distribution.  alpha is the shape parameter.
|
|  randint(self, a, b)
|      Return random integer in range [a, b], including both end points.
|
|  randrange(self, start, stop=None, step=1)
|      Choose a random item from range(start, stop[, step]).
|
|      This fixes the problem with randint() which includes the
|      endpoint; in Python this is usually not what you want.
|
|  sample(self, population, k, *, counts=None)
|      Chooses k unique random elements from a population sequence or set.
|
|      Returns a new list containing elements from the population while
|      leaving the original population unchanged.  The resulting list is
|      in selection order so that all sub-slices will also be valid random
|      samples.  This allows raffle winners (the sample) to be partitioned
|      into grand prize and second place winners (the subslices).
|
|      Members of the population need not be hashable or unique.  If the
|      population contains repeats, then each occurrence is a possible
|      selection in the sample.
|
|      Repeated elements can be specified one at a time or with the optional
|      counts parameter.  For example:
|
|          sample(['red', 'blue'], counts=[4, 2], k=5)
|
|      is equivalent to:
|
|          sample(['red', 'red', 'red', 'red', 'blue', 'blue'], k=5)
|
|      To choose a sample from a range of integers, use range() for the
|      population argument.  This is especially fast and space efficient
|      for sampling from a large population:
|
```

```
 |          sample(range(10000000), 60)
 |
 |  shuffle(self, x, random=None)
 |      Shuffle list x in place, and return None.
 |
 |      Optional argument random is a 0-argument function returning a
 |      random float in [0.0, 1.0); if it is the default None, the
 |      standard random.random will be used.
 |
 |  triangular(self, low=0.0, high=1.0, mode=None)
 |      Triangular distribution.
 |
 |      Continuous distribution bounded by given lower and upper limits,
 |      and having a given mode value in-between.
 |
 |      http://en.wikipedia.org/wiki/Triangular_distribution
 |
 |  uniform(self, a, b)
 |      Get a random number in the range [a, b) or [a, b] depending on rounding.
 |
 |  vonmisesvariate(self, mu, kappa)
 |      Circular data distribution.
 |
 |      mu is the mean angle, expressed in radians between 0 and 2*pi, and
 |      kappa is the concentration parameter, which must be greater than or
 |      equal to zero.  If kappa is equal to zero, this distribution reduces
 |      to a uniform random angle over the range 0 to 2*pi.
 |
 |  weibullvariate(self, alpha, beta)
 |      Weibull distribution.
 |
 |      alpha is the scale parameter and beta is the shape parameter.
 |
 |  ----------------------------------------------------------------------
 |  Class methods inherited from Random:
 |
 |  __init_subclass__(**kwargs) from builtins.type
 |      Control how subclasses generate random integers.
 |
 |      The algorithm a subclass can use depends on the random() and/or
 |      getrandbits() implementation available to it and determines
 |      whether it can generate random integers from arbitrarily large
 |      ranges.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from Random:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes inherited from Random:
 |
 |  VERSION = 3
 |
 |  ----------------------------------------------------------------------
 |  Static methods inherited from _random.Random:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.

FUNCTIONS
    betavariate(alpha, beta) method of Random instance
        Beta distribution.

        Conditions on the parameters are alpha > 0 and beta > 0.
        Returned values range between 0 and 1.

    choice(seq) method of Random instance
        Choose a random element from a non-empty sequence.

    choices(population, weights=None, *, cum_weights=None, k=1) method of Random instance
        Return a k sized list of population elements chosen with replacement.

        If the relative weights or cumulative weights are not specified,
        the selections are made with equal probability.

    expovariate(lambd) method of Random instance
        Exponential distribution.

        lambd is 1.0 divided by the desired mean.  It should be
        nonzero.  (The parameter would be called "lambda", but that is
        a reserved word in Python.)  Returned values range from 0 to
```

```
    positive infinity if lambd is positive, and from negative
    infinity to 0 if lambd is negative.

gammavariate(alpha, beta) method of Random instance
    Gamma distribution.  Not the gamma function!

    Conditions on the parameters are alpha > 0 and beta > 0.

    The probability distribution function is:

                x ** (alpha - 1) * math.exp(-x / beta)
      pdf(x) =  --------------------------------------
                  math.gamma(alpha) * beta ** alpha

gauss(mu, sigma) method of Random instance
    Gaussian distribution.

    mu is the mean, and sigma is the standard deviation.  This is
    slightly faster than the normalvariate() function.

    Not thread-safe without a lock around calls.

getrandbits(k, /) method of Random instance
    getrandbits(k) -> x.  Generates an int with k random bits.

getstate() method of Random instance
    Return internal state; can be passed to setstate() later.

lognormvariate(mu, sigma) method of Random instance
    Log normal distribution.

    If you take the natural logarithm of this distribution, you'll get a
    normal distribution with mean mu and standard deviation sigma.
    mu can have any value, and sigma must be greater than zero.

normalvariate(mu, sigma) method of Random instance
    Normal distribution.

    mu is the mean, and sigma is the standard deviation.

paretovariate(alpha) method of Random instance
    Pareto distribution.  alpha is the shape parameter.

randbytes(n) method of Random instance
    Generate n random bytes.

randint(a, b) method of Random instance
    Return random integer in range [a, b], including both end points.

random() method of Random instance
    random() -> x in the interval [0, 1).

randrange(start, stop=None, step=1) method of Random instance
    Choose a random item from range(start, stop[, step]).

    This fixes the problem with randint() which includes the
    endpoint; in Python this is usually not what you want.

sample(population, k, *, counts=None) method of Random instance
    Chooses k unique random elements from a population sequence or set.

    Returns a new list containing elements from the population while
    leaving the original population unchanged.  The resulting list is
    in selection order so that all sub-slices will also be valid random
    samples.  This allows raffle winners (the sample) to be partitioned
    into grand prize and second place winners (the subslices).

    Members of the population need not be hashable or unique.  If the
    population contains repeats, then each occurrence is a possible
    selection in the sample.

    Repeated elements can be specified one at a time or with the optional
    counts parameter.  For example:

        sample(['red', 'blue'], counts=[4, 2], k=5)

    is equivalent to:

        sample(['red', 'red', 'red', 'red', 'blue', 'blue'], k=5)

    To choose a sample from a range of integers, use range() for the
    population argument.  This is especially fast and space efficient
    for sampling from a large population:

        sample(range(10000000), 60)
```

```
    seed(a=None, version=2) method of Random instance
        Initialize internal state from a seed.

        The only supported seed types are None, int, float,
        str, bytes, and bytearray.

        None or no argument seeds from current time or from an operating
        system specific randomness source if available.

        If *a* is an int, all bits are used.

        For version 2 (the default), all of the bits are used if *a* is a str,
        bytes, or bytearray.  For version 1 (provided for reproducing random
        sequences from older versions of Python), the algorithm for str and
        bytes generates a narrower range of seeds.

    setstate(state) method of Random instance
        Restore internal state from object returned by getstate().

    shuffle(x, random=None) method of Random instance
        Shuffle list x in place, and return None.

        Optional argument random is a 0-argument function returning a
        random float in [0.0, 1.0); if it is the default None, the
        standard random.random will be used.

    triangular(low=0.0, high=1.0, mode=None) method of Random instance
        Triangular distribution.

        Continuous distribution bounded by given lower and upper limits,
        and having a given mode value in-between.

        http://en.wikipedia.org/wiki/Triangular_distribution

    uniform(a, b) method of Random instance
        Get a random number in the range [a, b) or [a, b] depending on rounding.

    vonmisesvariate(mu, kappa) method of Random instance
        Circular data distribution.

        mu is the mean angle, expressed in radians between 0 and 2*pi, and
        kappa is the concentration parameter, which must be greater than or
        equal to zero.  If kappa is equal to zero, this distribution reduces
        to a uniform random angle over the range 0 to 2*pi.

    weibullvariate(alpha, beta) method of Random instance
        Weibull distribution.

        alpha is the scale parameter and beta is the shape parameter.

DATA
    __all__ = ['Random', 'SystemRandom', 'betavariate', 'choice', 'choices...

FILE
    /Users/yavuzsebe/anaconda3/lib/python3.10/random.py
```

In [6]:
```
random.random() #0-1 arasında random sayı verir FLOAT
```

Out [6]: 0.645859844368483

In [7]:
```
random.uniform(3,10) #3-10 arasında random sayı verir FLOAT
```

Out [7]: 5.020726474861165

In [9]:
```
random.randint(5,10) #random integer verir
```

Out [9]: 7

In [10]:
```
random.choice(range(10)) #listelerden random seçim yapar sayı olmak zorunda değil ço
```

Out [10]: 9

In [11]:
```
random.sample(range(10), k=4) # belirtilen kadar öğeyi rastgele çeker
```

Out [11]: [1, 7, 5, 6]

In [19]:
```
random.shuffle([*range(10)]) #çıktı vermedi
print(random.shuffle([*range(10)])) #None verdi
```

None

In [18]:
```
liste = [*range(10)]
random.shuffle(liste)
print(liste)
```

[1, 7, 8, 4, 5, 2, 0, 6, 3, 9]

In [74]:
```
#Original Shuffle Match
import random
def shuffleMatch(a,b):
    x=1
    while True:
            try:
                list1 = [*range(a,b)]
                random.shuffle(list1)
                listOriginal=[*range(a,b)]
                if list1==listOriginal:
                    print("{} defa karıştırıldı.".format(x))
                    print("Karıştırılmış liste: {},\nOrjinal liste: {}".format(list1
                    break
                else:
                    x+=1
                    continue
            finally:
                pass
```

In [79]:
```
import random
def shuffleMatch(a,b):
    x=1
    while True:
            try:
                list1 = [*range(a,b)]
                random.shuffle(list1)
                listOriginal=[*range(a,b)]
                if list1==listOriginal:
                    print(x)
                    break
                else:
                    x+=1
                    continue
            finally:
                pass
```

In [76]:
```
shuffleMatch(1,6)
```

317

In [77]:
```
y=0
z=[]
tryOuts=[]
```

```
while True:
    try:
        z = shuffleMatch(1,6)
        tryOuts.append(z)
        if len(tryOuts)==len(set(tryOuts)):
            print(y)
            break
        else:
            y+=1
            continue
    finally:
        pass
```

```
3
0
```

In [8]:
```
import random

def shuffleMatch(a, b):
    x = 1
    while [*range(a, b)] != (shuffled := random.sample(range(a, b), b - a)):
        x += 1
    print(x)
```

In [16]:
```
shuffleMatch(1, 6)
```

```
227
```

In [18]:
```
import random

def shuffleMatch(a, b):
    x = 1
    while True:
        list1 = list(range(a, b))
        random.shuffle(list1)
        listOriginal = list(range(a, b))
        if list1 == listOriginal or len(set(list1)) < len(list1):
            break
        else:
            x += 1

    return x

y = 0
tryOuts = set()

while True:
    try:
        attempts = shuffleMatch(1, 6)
        y += 1
    except KeyboardInterrupt:
        break
    except:
        continue

    if y == len(tryOuts):
        print("Could not find a unique shuffle.")
```

```
        break

    tryOuts.add(y)

print(f"Number of attempts to get a unique shuffle: {attempts}")
```

```
Number of attempts to get a unique shuffle: 81
```

Math Module

In [1]:
```
import math
```

In [2]:
```
help(math)
```

```
Help on module math:

NAME
    math

MODULE REFERENCE
    https://docs.python.org/3.10/library/math.html

    The following documentation is automatically generated from the Python
    source files.  It may be incomplete, incorrect or include features that
    are considered implementation detail and may vary between Python
    implementations.  When in doubt, consult the module reference at the
    location listed above.

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.

    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 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.

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.

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(x, /)
    Find x!.

    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 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, /)
    Return the next floating-point value after x towards y.

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.

    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
    /Users/yavuzsebe/anaconda3/lib/python3.10/lib-dynload/math.cpython-310-darwin.so
```

In [3]:
```
round(7.1) #pythona dahil
```

Out [3]: 7

In [4]:
```
math.ceil(7.1) #her daim yukarı yuvarlar
```

Out [4]: 8

In [5]:
```
math.floor(7.9) #her daim aşağı yuvarlar
```

Out [5]: 7

In [7]:
```
math.factorial(6) #faktöriyel hesaplama
```

Out [7]: 720

In [9]:
```
math.pow(3,2) #kuvvet alır, float verir
```

Out [9]: 9.0

In [ ]: