Ch03-2-Functions-Library

September 10, 2025

1 Python Standard Libraries

1.1 Topics

- Python standard libraries
- import and use libraries

1.2 3.1 Standard libraries

- Python has several standard libraries (modules) you can readily import
- one can use the names (functions and data/constants) defined in those imported modules
- list of all the Python standard libraries: https://docs.python.org/3/library/index.html
- syntax

```
# first import library
import libraryName
import awesomeLibrary
import libraryName1 as mylib
from libraryName2 import func1, func2 # okay!

# use data and functions provided by the library
libraryName.data
libraryName.function()
func1()
mylib.someFunction()
func2()
```

- according to PEP 8 Guildelines, each import must be on each line
- importing comma separated multiple names from the same library is ok

1.3 3.2 math library

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

- an important library that provides mathematical functions
- run help(moduleName) to get more information about the module

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

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

```
Help on module math:

NAME
    math

MODULE REFERENCE
    https://docs.python.org/3.7/library/math

    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 is always available.  It provides access to the
    mathematical functions defined by the C standard.

FUNCTIONS
    acos(x, /)
        Return the arc cosine (measured in radians) of x.

    acosh(x, /)
        Return the inverse hyperbolic cosine of x.

    asin(x, /)
        Return the arc sine (measured in radians) of x.

    asinh(x, /)
        Return the inverse hyperbolic sine of x.

    atan(x, /)
        Return the arc tangent (measured in radians) of x.

    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.
```

```
    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.

    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(x, y, /)
        greatest common divisor of x and y

    hypot(x, y, /)
        Return the Euclidean distance, sqrt(x*x + y*y).

    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.

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.

pow(x, y, /)
    Return x**y (x to the power of y).

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.

DATA
    e = 2.718281828459045
    inf = inf
    nan = nan
    pi = 3.141592653589793
    tau = 6.283185307179586

FILE
    /Users/rbasnet/anaconda3/lib/python3.7/lib-
dynload/math.cpython-37m-darwin.so
```

[3]:
```
num = 10.5
# math.ceil(x) - return the ceiling (or round up) of x,
# the smallest integer greater than or equal to x
print(math.ceil(num))
```

```
11
```

[4]:
```
# math.floor(x)
# return the floor (or round down) of x, the largest integer less than or equal␣
 ↪to x
print(math.floor(num))
```

```
10
```

[5]:
```
# math.gcd(a, b)
# return the greatest common divisor of the integers a and b
# if both and b are 0, returns 0
```

```
print(math.gcd(0, 0))
print(math.gcd(10, 20))
```

```
0
10
```

[6]:
```
# math.pow(x, y)
# returns x raised to the power y
print(math.pow(2, 10))
```

```
1024.0
```

[7]:
```
# math.sqrt(x, y)
# returns the square root of x
print(math.sqrt(100))
```

```
10.0
```

[8]:
```
# math.radians(x)
# convert and return angle x in degrees to radians
rad = math.radians(90)
```

[9]:
```
# math.sin(x)
# return the sine of x radians
print(math.sin(rad))
```

```
1.0
```

[10]:
```
# Some constants/data defined in math module
math.pi
```

[10]:
```
3.141592653589793
```

[11]:
```
math.inf
```

[11]:
```
inf
```

[12]:
```
math.e
```

[12]:
```
2.718281828459045
```

1.4 3.3 Other common libraries

- all Python libraries: https://docs.python.org/3/library/index.html
- some libraries we'll explore
  - **os** - operating system related
  - **time** - time access and conversion
  - **random** - generate pseudo-random numbers
  - **sys** - system specific data and functions
  - **string** - common string operations and data

[3]:
```
import random
```

[4]:
```
help(random)
```

```
Help on module random:

NAME
    random - Random variable generators.

MODULE REFERENCE
    https://docs.python.org/3.7/library/random

    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
        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.
|
|  randint(self, a, b)
|      Return random integer in range [a, b], including both end points.
|
|  randrange(self, start, stop=None, step=1, _int=<class 'int'>)
|      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)
|      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.
|
|      To choose a sample in a range of integers, use range as an 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 hashable object.
|
|      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.
 |
 |  ----------------------------------------------------------------------
 |  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:
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  getrandbits(…)
 |      getrandbits(k) -> x.  Generates an int with k random bits.
 |
 |  random(…)
 |      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)
```

```
 |
 |  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, _int=<class 'int'>)
|      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)
 |      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.
 |
 |      To choose a sample in a range of integers, use range as an 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.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from Random:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes inherited from Random:
 |
 |  VERSION = 3
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from _random.Random:
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  ----------------------------------------------------------------------
 |  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(…) 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.

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, _int=<class 'int'>) 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) 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.

        To choose a sample in a range of integers, use range as an 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 hashable object.

        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', 'seed', 'random', 'uniform', 'randint', 'choice',…

FILE
    /Users/rbasnet/anaconda3/lib/python3.7/random.py
```

[5]:
```
from random import randint
```

[8]:
```
randint(0, 20) # Return a random integer between (a, b) inclusive
```

[8]: 16

[1]:
```
import os
```

[2]:
```
help(os)
```

```
Help on module os:
```

```
NAME
    os - OS routines for NT or Posix depending on what system we're on.

MODULE REFERENCE
    https://docs.python.org/3.8/library/os

    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 exports:
      - all functions from posix or nt, e.g. unlink, stat, etc.
      - os.path is either posixpath or ntpath
      - os.name is either 'posix' or 'nt'
      - os.curdir is a string representing the current directory (always '.')
      - os.pardir is a string representing the parent directory (always '..')
      - os.sep is the (or a most common) pathname separator ('/' or '\\')
      - os.extsep is the extension separator (always '.')
      - os.altsep is the alternate pathname separator (None or '/')
      - os.pathsep is the component separator used in $PATH etc
      - os.linesep is the line separator in text files ('\r' or '\n' or '\r\n')
      - os.defpath is the default search path for executables
      - os.devnull is the file path of the null device ('/dev/null', etc.)

    Programs that import and use 'os' stand a better chance of being
    portable between different platforms.  Of course, they must then
    only use functions that are defined by all platforms (e.g., unlink
    and opendir), and leave all pathname manipulation to os.path
    (e.g., split and join).

CLASSES
    builtins.Exception(builtins.BaseException)
        builtins.OSError
    builtins.object
        posix.DirEntry
    builtins.tuple(builtins.object)
        stat_result
        statvfs_result
        terminal_size
        posix.times_result
        posix.uname_result

    class DirEntry(builtins.object)
     |  Methods defined here:
     |
```

```
 |  __fspath__(self, /)
 |      Returns the path for the entry.
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  inode(self, /)
 |      Return inode of the entry; cached per entry.
 |
 |  is_dir(self, /, *, follow_symlinks=True)
 |      Return True if the entry is a directory; cached per entry.
 |
 |  is_file(self, /, *, follow_symlinks=True)
 |      Return True if the entry is a file; cached per entry.
 |
 |  is_symlink(self, /)
 |      Return True if the entry is a symbolic link; cached per entry.
 |
 |  stat(self, /, *, follow_symlinks=True)
 |      Return stat_result object for the entry; cached per entry.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  name
 |      the entry's base filename, relative to scandir() "path" argument
 |
 |  path
 |      the entry's full path name; equivalent to os.path.join(scandir_path,
entry.name)

error = class OSError(Exception)
 |  Base class for I/O related errors.
 |
 |  Method resolution order:
 |      OSError
 |      Exception
 |      BaseException
 |      object
 |
 |  Built-in subclasses:
 |      BlockingIOError
 |      ChildProcessError
 |      ConnectionError
 |      FileExistsError
 |      … and 7 other subclasses
 |
 |  Methods defined here:
```

```
      |
      |  __init__(self, /, *args, **kwargs)
      |      Initialize self.  See help(type(self)) for accurate signature.
      |
      |  __reduce__(…)
      |      Helper for pickle.
      |
      |  __str__(self, /)
      |      Return str(self).
      |
      |  ----------------------------------------------------------------------
      |  Static methods defined here:
      |
      |  __new__(*args, **kwargs) from builtins.type
      |      Create and return a new object.  See help(type) for accurate
signature.
      |
      |  ----------------------------------------------------------------------
      |  Data descriptors defined here:
      |
      |  characters_written
      |
      |  errno
      |      POSIX exception code
      |
      |  filename
      |      exception filename
      |
      |  filename2
      |      second exception filename
      |
      |  strerror
      |      exception strerror
      |
      |  ----------------------------------------------------------------------
      |  Methods inherited from BaseException:
      |
      |  __delattr__(self, name, /)
      |      Implement delattr(self, name).
      |
      |  __getattribute__(self, name, /)
      |      Return getattr(self, name).
      |
      |  __repr__(self, /)
      |      Return repr(self).
      |
      |  __setattr__(self, name, value, /)
      |      Implement setattr(self, name, value).
```

```
 |
 |  __setstate__(…)
 |
 |  with_traceback(…)
 |      Exception.with_traceback(tb) --
 |      set self.__traceback__ to tb and return self.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from BaseException:
 |
 |  __cause__
 |      exception cause
 |
 |  __context__
 |      exception context
 |
 |  __dict__
 |
 |  __suppress_context__
 |
 |  __traceback__
 |
 |  args

class stat_result(builtins.tuple)
 |  stat_result(iterable=(), /)
 |
 |  stat_result: Result from stat, fstat, or lstat.
 |
 |  This object may be accessed either as a tuple of
 |    (mode, ino, dev, nlink, uid, gid, size, atime, mtime, ctime)
 |  or via the attributes st_mode, st_ino, st_dev, st_nlink, st_uid, and so
on.
 |
 |  Posix/windows: If your platform supports st_blksize, st_blocks, st_rdev,
 |  or st_flags, they are available as attributes only.
 |
 |  See os.stat for more information.
 |
 |  Method resolution order:
 |      stat_result
 |      builtins.tuple
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __reduce__(…)
 |      Helper for pickle.
```

```
      |
      |  __repr__(self, /)
      |      Return repr(self).
      |
      |  ----------------------------------------------------------------------
      |  Static methods defined here:
      |
      |  __new__(*args, **kwargs) from builtins.type
      |      Create and return a new object.  See help(type) for accurate
signature.
      |
      |  ----------------------------------------------------------------------
      |  Data descriptors defined here:
      |
      |  st_atime
      |      time of last access
      |
      |  st_atime_ns
      |      time of last access in nanoseconds
      |
      |  st_birthtime
      |      time of creation
      |
      |  st_blksize
      |      blocksize for filesystem I/O
      |
      |  st_blocks
      |      number of blocks allocated
      |
      |  st_ctime
      |      time of last change
      |
      |  st_ctime_ns
      |      time of last change in nanoseconds
      |
      |  st_dev
      |      device
      |
      |  st_flags
      |      user defined flags for file
      |
      |  st_gen
      |      generation number
      |
      |  st_gid
      |      group ID of owner
      |
      |  st_ino
```

```
|      inode
|
|  st_mode
|      protection bits
|
|  st_mtime
|      time of last modification
|
|  st_mtime_ns
|      time of last modification in nanoseconds
|
|  st_nlink
|      number of hard links
|
|  st_rdev
|      device type (if inode device)
|
|  st_size
|      total size, in bytes
|
|  st_uid
|      user ID of owner
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  n_fields = 22
|
|  n_sequence_fields = 10
|
|  n_unnamed_fields = 3
|
|  ----------------------------------------------------------------------
|  Methods inherited from builtins.tuple:
|
|  __add__(self, value, /)
|      Return self+value.
|
|  __contains__(self, key, /)
|      Return key in self.
|
|  __eq__(self, value, /)
|      Return self==value.
|
|  __ge__(self, value, /)
|      Return self>=value.
|
|  __getattribute__(self, name, /)
```

```
 |      Return getattr(self, name).
 |
 |  __getitem__(self, key, /)
 |      Return self[key].
 |
 |  __getnewargs__(self, /)
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __hash__(self, /)
 |      Return hash(self).
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __mul__(self, value, /)
 |      Return self*value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __rmul__(self, value, /)
 |      Return value*self.
 |
 |  count(self, value, /)
 |      Return number of occurrences of value.
 |
 |  index(self, value, start=0, stop=9223372036854775807, /)
 |      Return first index of value.
 |
 |      Raises ValueError if the value is not present.

class statvfs_result(builtins.tuple)
 |  statvfs_result(iterable=(), /)
 |
 |  statvfs_result: Result from statvfs or fstatvfs.
 |
 |  This object may be accessed either as a tuple of
```

```
     |    (bsize, frsize, blocks, bfree, bavail, files, ffree, favail, flag,
namemax),
     |  or via the attributes f_bsize, f_frsize, f_blocks, f_bfree, and so on.
     |
     |  See os.statvfs for more information.
     |
     |  Method resolution order:
     |      statvfs_result
     |      builtins.tuple
     |      builtins.object
     |
     |  Methods defined here:
     |
     |  __reduce__(…)
     |      Helper for pickle.
     |
     |  __repr__(self, /)
     |      Return repr(self).
     |
     |  ----------------------------------------------------------------------
     |  Static methods defined here:
     |
     |  __new__(*args, **kwargs) from builtins.type
     |      Create and return a new object.  See help(type) for accurate
signature.
     |
     |  ----------------------------------------------------------------------
     |  Data descriptors defined here:
     |
     |  f_bavail
     |
     |  f_bfree
     |
     |  f_blocks
     |
     |  f_bsize
     |
     |  f_favail
     |
     |  f_ffree
     |
     |  f_files
     |
     |  f_flag
     |
     |  f_frsize
     |
     |  f_fsid
```

```
|
|  f_namemax
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  n_fields = 11
|
|  n_sequence_fields = 10
|
|  n_unnamed_fields = 0
|
|  ----------------------------------------------------------------------
|  Methods inherited from builtins.tuple:
|
|  __add__(self, value, /)
|      Return self+value.
|
|  __contains__(self, key, /)
|      Return key in self.
|
|  __eq__(self, value, /)
|      Return self==value.
|
|  __ge__(self, value, /)
|      Return self>=value.
|
|  __getattribute__(self, name, /)
|      Return getattr(self, name).
|
|  __getitem__(self, key, /)
|      Return self[key].
|
|  __getnewargs__(self, /)
|
|  __gt__(self, value, /)
|      Return self>value.
|
|  __hash__(self, /)
|      Return hash(self).
|
|  __iter__(self, /)
|      Implement iter(self).
|
|  __le__(self, value, /)
|      Return self<=value.
|
|  __len__(self, /)
```

```
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __mul__(self, value, /)
 |      Return self*value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __rmul__(self, value, /)
 |      Return value*self.
 |
 |  count(self, value, /)
 |      Return number of occurrences of value.
 |
 |  index(self, value, start=0, stop=9223372036854775807, /)
 |      Return first index of value.
 |
 |      Raises ValueError if the value is not present.

class terminal_size(builtins.tuple)
 |  terminal_size(iterable=(), /)
 |
 |  A tuple of (columns, lines) for holding terminal window size
 |
 |  Method resolution order:
 |      terminal_size
 |      builtins.tuple
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __reduce__(…)
 |      Helper for pickle.
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate
signature.
 |
 |  ----------------------------------------------------------------------
```

```
|  Data descriptors defined here:
|
|  columns
|      width of the terminal window in characters
|
|  lines
|      height of the terminal window in characters
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  n_fields = 2
|
|  n_sequence_fields = 2
|
|  n_unnamed_fields = 0
|
|  ----------------------------------------------------------------------
|  Methods inherited from builtins.tuple:
|
|  __add__(self, value, /)
|      Return self+value.
|
|  __contains__(self, key, /)
|      Return key in self.
|
|  __eq__(self, value, /)
|      Return self==value.
|
|  __ge__(self, value, /)
|      Return self>=value.
|
|  __getattribute__(self, name, /)
|      Return getattr(self, name).
|
|  __getitem__(self, key, /)
|      Return self[key].
|
|  __getnewargs__(self, /)
|
|  __gt__(self, value, /)
|      Return self>value.
|
|  __hash__(self, /)
|      Return hash(self).
|
|  __iter__(self, /)
|      Implement iter(self).
```

```
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __mul__(self, value, /)
 |      Return self*value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __rmul__(self, value, /)
 |      Return value*self.
 |
 |  count(self, value, /)
 |      Return number of occurrences of value.
 |
 |  index(self, value, start=0, stop=9223372036854775807, /)
 |      Return first index of value.
 |
 |      Raises ValueError if the value is not present.

class times_result(builtins.tuple)
 |  times_result(iterable=(), /)
 |
 |  times_result: Result from os.times().
 |
 |  This object may be accessed either as a tuple of
 |    (user, system, children_user, children_system, elapsed),
 |  or via the attributes user, system, children_user, children_system,
 |  and elapsed.
 |
 |  See os.times for more information.
 |
 |  Method resolution order:
 |      times_result
 |      builtins.tuple
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __reduce__(…)
 |      Helper for pickle.
```

```
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate
signature.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  children_system
 |      system time of children
 |
 |  children_user
 |      user time of children
 |
 |  elapsed
 |      elapsed time since an arbitrary point in the past
 |
 |  system
 |      system time
 |
 |  user
 |      user time
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  n_fields = 5
 |
 |  n_sequence_fields = 5
 |
 |  n_unnamed_fields = 0
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from builtins.tuple:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __eq__(self, value, /)
```

```
|      Return self==value.
|
|  __ge__(self, value, /)
|      Return self>=value.
|
|  __getattribute__(self, name, /)
|      Return getattr(self, name).
|
|  __getitem__(self, key, /)
|      Return self[key].
|
|  __getnewargs__(self, /)
|
|  __gt__(self, value, /)
|      Return self>value.
|
|  __hash__(self, /)
|      Return hash(self).
|
|  __iter__(self, /)
|      Implement iter(self).
|
|  __le__(self, value, /)
|      Return self<=value.
|
|  __len__(self, /)
|      Return len(self).
|
|  __lt__(self, value, /)
|      Return self<value.
|
|  __mul__(self, value, /)
|      Return self*value.
|
|  __ne__(self, value, /)
|      Return self!=value.
|
|  __rmul__(self, value, /)
|      Return value*self.
|
|  count(self, value, /)
|      Return number of occurrences of value.
|
|  index(self, value, start=0, stop=9223372036854775807, /)
|      Return first index of value.
|
|      Raises ValueError if the value is not present.
```

```
    class uname_result(builtins.tuple)
     |  uname_result(iterable=(), /)
     |
     |  uname_result: Result from os.uname().
     |
     |  This object may be accessed either as a tuple of
     |    (sysname, nodename, release, version, machine),
     |  or via the attributes sysname, nodename, release, version, and machine.
     |
     |  See os.uname for more information.
     |
     |  Method resolution order:
     |      uname_result
     |      builtins.tuple
     |      builtins.object
     |
     |  Methods defined here:
     |
     |  __reduce__(…)
     |      Helper for pickle.
     |
     |  __repr__(self, /)
     |      Return repr(self).
     |
     |  ----------------------------------------------------------------------
     |  Static methods defined here:
     |
     |  __new__(*args, **kwargs) from builtins.type
     |      Create and return a new object.  See help(type) for accurate
signature.
     |
     |  ----------------------------------------------------------------------
     |  Data descriptors defined here:
     |
     |  machine
     |      hardware identifier
     |
     |  nodename
     |      name of machine on network (implementation-defined)
     |
     |  release
     |      operating system release
     |
     |  sysname
     |      operating system name
     |
     |  version
     |      operating system version
```

```
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  n_fields = 5
|
|  n_sequence_fields = 5
|
|  n_unnamed_fields = 0
|
|  ----------------------------------------------------------------------
|  Methods inherited from builtins.tuple:
|
|  __add__(self, value, /)
|      Return self+value.
|
|  __contains__(self, key, /)
|      Return key in self.
|
|  __eq__(self, value, /)
|      Return self==value.
|
|  __ge__(self, value, /)
|      Return self>=value.
|
|  __getattribute__(self, name, /)
|      Return getattr(self, name).
|
|  __getitem__(self, key, /)
|      Return self[key].
|
|  __getnewargs__(self, /)
|
|  __gt__(self, value, /)
|      Return self>value.
|
|  __hash__(self, /)
|      Return hash(self).
|
|  __iter__(self, /)
|      Implement iter(self).
|
|  __le__(self, value, /)
|      Return self<=value.
|
|  __len__(self, /)
|      Return len(self).
|
```

```
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __mul__(self, value, /)
 |      Return self*value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __rmul__(self, value, /)
 |      Return value*self.
 |
 |  count(self, value, /)
 |      Return number of occurrences of value.
 |
 |  index(self, value, start=0, stop=9223372036854775807, /)
 |      Return first index of value.
 |
 |      Raises ValueError if the value is not present.

FUNCTIONS
    WCOREDUMP(status, /)
        Return True if the process returning status was dumped to a core file.

    WEXITSTATUS(status)
        Return the process return code from status.

    WIFCONTINUED(status)
        Return True if a particular process was continued from a job control
stop.

        Return True if the process returning status was continued from a
        job control stop.

    WIFEXITED(status)
        Return True if the process returning status exited via the exit() system
call.

    WIFSIGNALED(status)
        Return True if the process returning status was terminated by a signal.

    WIFSTOPPED(status)
        Return True if the process returning status was stopped.

    WSTOPSIG(status)
        Return the signal that stopped the process that provided the status
value.
```

```
    WTERMSIG(status)
        Return the signal that terminated the process that provided the status
value.

    _exit(status)
        Exit to the system with specified status, without normal exit
processing.

    abort()
        Abort the interpreter immediately.

        This function 'dumps core' or otherwise fails in the hardest way
possible
        on the hosting operating system.  This function never returns.

    access(path, mode, *, dir_fd=None, effective_ids=False,
follow_symlinks=True)
        Use the real uid/gid to test for access to a path.

          path
            Path to be tested; can be string, bytes, or a path-like object.
          mode
            Operating-system mode bitfield.  Can be F_OK to test existence,
            or the inclusive-OR of R_OK, W_OK, and X_OK.
          dir_fd
            If not None, it should be a file descriptor open to a directory,
            and path should be relative; path will then be relative to that
            directory.
          effective_ids
            If True, access will use the effective uid/gid instead of
            the real uid/gid.
          follow_symlinks
            If False, and the last element of the path is a symbolic link,
            access will examine the symbolic link itself instead of the file
            the link points to.

        dir_fd, effective_ids, and follow_symlinks may not be implemented
          on your platform.  If they are unavailable, using them will raise a
          NotImplementedError.

        Note that most operations will use the effective uid/gid, therefore this
          routine can be used in a suid/sgid environment to test if the invoking
user
          has the specified access to the path.

    chdir(path)
        Change the current working directory to the specified path.
```

```
        path may always be specified as a string.
        On some platforms, path may also be specified as an open file
descriptor.
          If this functionality is unavailable, using it raises an exception.

    chflags(path, flags, follow_symlinks=True)
        Set file flags.

        If follow_symlinks is False, and the last element of the path is a
symbolic
          link, chflags will change flags on the symbolic link itself instead of
the
          file the link points to.
        follow_symlinks may not be implemented on your platform.  If it is
        unavailable, using it will raise a NotImplementedError.

    chmod(path, mode, *, dir_fd=None, follow_symlinks=True)
        Change the access permissions of a file.

          path
            Path to be modified.  May always be specified as a str, bytes, or a
path-like object.
            On some platforms, path may also be specified as an open file
descriptor.
            If this functionality is unavailable, using it raises an exception.
          mode
            Operating-system mode bitfield.
          dir_fd
            If not None, it should be a file descriptor open to a directory,
            and path should be relative; path will then be relative to that
            directory.
          follow_symlinks
            If False, and the last element of the path is a symbolic link,
            chmod will modify the symbolic link itself instead of the file
            the link points to.

        It is an error to use dir_fd or follow_symlinks when specifying path as
          an open file descriptor.
        dir_fd and follow_symlinks may not be implemented on your platform.
          If they are unavailable, using them will raise a NotImplementedError.

    chown(path, uid, gid, *, dir_fd=None, follow_symlinks=True)
        Change the owner and group id of path to the numeric uid and gid.\

          path
            Path to be examined; can be string, bytes, a path-like object, or
open-file-descriptor int.
          dir_fd
```

```
            If not None, it should be a file descriptor open to a directory,
            and path should be relative; path will then be relative to that
            directory.
          follow_symlinks
            If False, and the last element of the path is a symbolic link,
            stat will examine the symbolic link itself instead of the file
            the link points to.

        path may always be specified as a string.
        On some platforms, path may also be specified as an open file
descriptor.
          If this functionality is unavailable, using it raises an exception.
        If dir_fd is not None, it should be a file descriptor open to a
directory,
          and path should be relative; path will then be relative to that
directory.
        If follow_symlinks is False, and the last element of the path is a
symbolic
          link, chown will modify the symbolic link itself instead of the file
the
          link points to.
        It is an error to use dir_fd or follow_symlinks when specifying path as
          an open file descriptor.
        dir_fd and follow_symlinks may not be implemented on your platform.
          If they are unavailable, using them will raise a NotImplementedError.

    chroot(path)
        Change root directory to path.

    close(fd)
        Close a file descriptor.

    closerange(fd_low, fd_high, /)
        Closes all file descriptors in [fd_low, fd_high), ignoring errors.

    confstr(name, /)
        Return a string-valued system configuration variable.

    cpu_count()
        Return the number of CPUs in the system; return None if indeterminable.

        This number is not equivalent to the number of CPUs the current process
can
        use.  The number of usable CPUs can be obtained with
        ``len(os.sched_getaffinity(0))``

    ctermid()
        Return the name of the controlling terminal for this process.
```

```
device_encoding(fd)
    Return a string describing the encoding of a terminal's file descriptor.

    The file descriptor must be attached to a terminal.
    If the device is not a terminal, return None.

dup(fd, /)
    Return a duplicate of a file descriptor.

dup2(fd, fd2, inheritable=True)
    Duplicate file descriptor.

execl(file, *args)
    execl(file, *args)

    Execute the executable file with argument list args, replacing the
    current process.

execle(file, *args)
    execle(file, *args, env)

    Execute the executable file with argument list args and
    environment env, replacing the current process.

execlp(file, *args)
    execlp(file, *args)

    Execute the executable file (which is searched for along $PATH)
    with argument list args, replacing the current process.

execlpe(file, *args)
    execlpe(file, *args, env)

    Execute the executable file (which is searched for along $PATH)
    with argument list args and environment env, replacing the current
    process.

execv(path, argv, /)
    Execute an executable path with arguments, replacing current process.

    path
      Path of executable file.
    argv
      Tuple or list of strings.

execve(path, argv, env)
    Execute an executable path with arguments, replacing current process.
```

```
    path
      Path of executable file.
    argv
      Tuple or list of strings.
    env
      Dictionary of strings mapping to strings.

execvp(file, args)
    execvp(file, args)

    Execute the executable file (which is searched for along $PATH)
    with argument list args, replacing the current process.
    args may be a list or tuple of strings.

execvpe(file, args, env)
    execvpe(file, args, env)

    Execute the executable file (which is searched for along $PATH)
    with argument list args and environment env, replacing the
    current process.
    args may be a list or tuple of strings.

fchdir(fd)
    Change to the directory of the given file descriptor.

    fd must be opened on a directory, not a file.
    Equivalent to os.chdir(fd).

fchmod(fd, mode)
    Change the access permissions of the file given by file descriptor fd.

    Equivalent to os.chmod(fd, mode).

fchown(fd, uid, gid)
    Change the owner and group id of the file specified by file descriptor.

    Equivalent to os.chown(fd, uid, gid).

fdopen(fd, *args, **kwargs)
    # Supply os.fdopen()

fork()
    Fork a child process.

    Return 0 to child process and PID of child to parent process.

forkpty()
```

```
        Fork a new process with a new pseudo-terminal as controlling tty.

        Returns a tuple of (pid, master_fd).
        Like fork(), return pid of 0 to the child process,
        and pid of child to the parent process.
        To both, return fd of newly opened pseudo-terminal.

    fpathconf(fd, name, /)
        Return the configuration limit name for the file descriptor fd.

        If there is no limit, return -1.

    fsdecode(filename)
        Decode filename (an os.PathLike, bytes, or str) from the filesystem
        encoding with 'surrogateescape' error handler, return str unchanged. On
        Windows, use 'strict' error handler if the file system encoding is
        'mbcs' (which is the default encoding).

    fsencode(filename)
        Encode filename (an os.PathLike, bytes, or str) to the filesystem
        encoding with 'surrogateescape' error handler, return bytes unchanged.
        On Windows, use 'strict' error handler if the file system encoding is
        'mbcs' (which is the default encoding).

    fspath(path)
        Return the file system path representation of the object.

        If the object is str or bytes, then allow it to pass through as-is. If
the
        object defines __fspath__(), then return the result of that method. All
other
        types raise a TypeError.

    fstat(fd)
        Perform a stat system call on the given file descriptor.

        Like stat(), but for an open file descriptor.
        Equivalent to os.stat(fd).

    fstatvfs(fd, /)
        Perform an fstatvfs system call on the given fd.

        Equivalent to statvfs(fd).

    fsync(fd)
        Force write of fd to disk.

    ftruncate(fd, length, /)
```

```
        Truncate a file, specified by file descriptor, to a specific length.

    fwalk(top='.', topdown=True, onerror=None, *, follow_symlinks=False,
dir_fd=None)
        Directory tree generator.

        This behaves exactly like walk(), except that it yields a 4-tuple

            dirpath, dirnames, filenames, dirfd

        `dirpath`, `dirnames` and `filenames` are identical to walk() output,
        and `dirfd` is a file descriptor referring to the directory `dirpath`.

        The advantage of fwalk() over walk() is that it's safe against symlink
        races (when follow_symlinks is False).

        If dir_fd is not None, it should be a file descriptor open to a
directory,
          and top should be relative; top will then be relative to that
directory.
          (dir_fd is always supported for fwalk.)

        Caution:
        Since fwalk() yields file descriptors, those are only valid until the
        next iteration step, so you should dup() them if you want to keep them
        for a longer period.

        Example:

        import os
        for root, dirs, files, rootfd in os.fwalk('python/Lib/email'):
            print(root, "consumes", end="")
            print(sum(os.stat(name, dir_fd=rootfd).st_size for name in files),
                  end="")
            print("bytes in", len(files), "non-directory files")
            if 'CVS' in dirs:
                dirs.remove('CVS')  # don't visit CVS directories

    get_blocking(fd, /)
        Get the blocking mode of the file descriptor.

        Return False if the O_NONBLOCK flag is set, True if the flag is cleared.

    get_exec_path(env=None)
        Returns the sequence of directories that will be searched for the
        named executable (similar to a shell) when launching a process.

        *env* must be an environment variable dict or None.  If *env* is None,
```

```
    os.environ will be used.

get_inheritable(fd, /)
    Get the close-on-exe flag of the specified file descriptor.

get_terminal_size(…)
    Return the size of the terminal window as (columns, lines).

    The optional argument fd (default standard output) specifies
    which file descriptor should be queried.

    If the file descriptor is not connected to a terminal, an OSError
    is thrown.

    This function will only be defined if an implementation is
    available for this system.

    shutil.get_terminal_size is the high-level function which should
    normally be used, os.get_terminal_size is the low-level implementation.

getcwd()
    Return a unicode string representing the current working directory.

getcwdb()
    Return a bytes string representing the current working directory.

getegid()
    Return the current process's effective group id.

getenv(key, default=None)
    Get an environment variable, return None if it doesn't exist.
    The optional second argument can specify an alternate default.
    key, default and the result are str.

getenvb(key, default=None)
    Get an environment variable, return None if it doesn't exist.
    The optional second argument can specify an alternate default.
    key, default and the result are bytes.

geteuid()
    Return the current process's effective user id.

getgid()
    Return the current process's group id.

getgrouplist(…)
    getgrouplist(user, group) -> list of groups to which a user belongs
```

```
        Returns a list of groups to which a user belongs.

            user: username to lookup
            group: base group id of the user

    getgroups()
        Return list of supplemental group IDs for the process.

    getloadavg()
        Return average recent system load information.

        Return the number of processes in the system run queue averaged over
        the last 1, 5, and 15 minutes as a tuple of three floats.
        Raises OSError if the load average was unobtainable.

    getlogin()
        Return the actual login name.

    getpgid(pid)
        Call the system call getpgid(), and return the result.

    getpgrp()
        Return the current process group id.

    getpid()
        Return the current process id.

    getppid()
        Return the parent's process id.

        If the parent process has already exited, Windows machines will still
        return its id; others systems will return the id of the 'init' process
(1).

    getpriority(which, who)
        Return program scheduling priority.

    getsid(pid, /)
        Call the system call getsid(pid) and return the result.

    getuid()
        Return the current process's user id.

    initgroups(…)
        initgroups(username, gid) -> None

        Call the system initgroups() to initialize the group access list with
all of
```

```
        the groups of which the specified username is a member, plus the
specified
        group id.

    isatty(fd, /)
        Return True if the fd is connected to a terminal.

        Return True if the file descriptor is an open file descriptor
        connected to the slave end of a terminal.

    kill(pid, signal, /)
        Kill a process with a signal.

    killpg(pgid, signal, /)
        Kill a process group with a signal.

    lchflags(path, flags)
        Set file flags.

        This function will not follow symbolic links.
        Equivalent to chflags(path, flags, follow_symlinks=False).

    lchmod(path, mode)
        Change the access permissions of a file, without following symbolic
links.

        If path is a symlink, this affects the link itself rather than the
target.
        Equivalent to chmod(path, mode, follow_symlinks=False)."

    lchown(path, uid, gid)
        Change the owner and group id of path to the numeric uid and gid.

        This function will not follow symbolic links.
        Equivalent to os.chown(path, uid, gid, follow_symlinks=False).

    link(src, dst, *, src_dir_fd=None, dst_dir_fd=None, follow_symlinks=True)
        Create a hard link to a file.

        If either src_dir_fd or dst_dir_fd is not None, it should be a file
          descriptor open to a directory, and the respective path string (src or
dst)
          should be relative; the path will then be relative to that directory.
        If follow_symlinks is False, and the last element of src is a symbolic
          link, link will create a link to the symbolic link itself instead of
the
          file the link points to.
        src_dir_fd, dst_dir_fd, and follow_symlinks may not be implemented on
```

```
your
          platform.  If they are unavailable, using them will raise a
          NotImplementedError.

    listdir(path=None)
        Return a list containing the names of the files in the directory.

        path can be specified as either str, bytes, or a path-like object.  If
path is bytes,
          the filenames returned will also be bytes; in all other circumstances
          the filenames returned will be str.
        If path is None, uses the path='.'.
        On some platforms, path may also be specified as an open file
descriptor;\
          the file descriptor must refer to a directory.
          If this functionality is unavailable, using it raises
NotImplementedError.

        The list is in arbitrary order.  It does not include the special
        entries '.' and '..' even if they are present in the directory.

    lockf(fd, command, length, /)
        Apply, test or remove a POSIX lock on an open file descriptor.

        fd
          An open file descriptor.
        command
          One of F_LOCK, F_TLOCK, F_ULOCK or F_TEST.
        length
          The number of bytes to lock, starting at the current position.

    lseek(fd, position, how, /)
        Set the position of a file descriptor.  Return the new position.

        Return the new cursor position in number of bytes
        relative to the beginning of the file.

    lstat(path, *, dir_fd=None)
        Perform a stat system call on the given path, without following symbolic
links.

        Like stat(), but do not follow symbolic links.
        Equivalent to stat(path, follow_symlinks=False).

    major(device, /)
        Extracts a device major number from a raw device number.

    makedev(major, minor, /)
```

```
        Composes a raw device number from the major and minor device numbers.

    makedirs(name, mode=511, exist_ok=False)
        makedirs(name [, mode=0o777][, exist_ok=False])

        Super-mkdir; create a leaf directory and all intermediate ones.  Works
like
        mkdir, except that any intermediate path segment (not just the
rightmost)
        will be created if it does not exist. If the target directory already
        exists, raise an OSError if exist_ok is False. Otherwise no exception is
        raised.  This is recursive.

    minor(device, /)
        Extracts a device minor number from a raw device number.

    mkdir(path, mode=511, *, dir_fd=None)
        Create a directory.

        If dir_fd is not None, it should be a file descriptor open to a
directory,
          and path should be relative; path will then be relative to that
directory.
        dir_fd may not be implemented on your platform.
          If it is unavailable, using it will raise a NotImplementedError.

        The mode argument is ignored on Windows.

    mkfifo(path, mode=438, *, dir_fd=None)
        Create a "fifo" (a POSIX named pipe).

        If dir_fd is not None, it should be a file descriptor open to a
directory,
          and path should be relative; path will then be relative to that
directory.
        dir_fd may not be implemented on your platform.
          If it is unavailable, using it will raise a NotImplementedError.

    mknod(path, mode=384, device=0, *, dir_fd=None)
        Create a node in the file system.

        Create a node in the file system (file, device special file or named
pipe)
        at path.  mode specifies both the permissions to use and the
        type of node to be created, being combined (bitwise OR) with one of
        S_IFREG, S_IFCHR, S_IFBLK, and S_IFIFO.  If S_IFCHR or S_IFBLK is set on
mode,
        device defines the newly created device special file (probably using
```

```
        os.makedev()).  Otherwise device is ignored.

        If dir_fd is not None, it should be a file descriptor open to a
directory,
          and path should be relative; path will then be relative to that
directory.
        dir_fd may not be implemented on your platform.
          If it is unavailable, using it will raise a NotImplementedError.

    nice(increment, /)
        Add increment to the priority of process and return the new priority.

    open(path, flags, mode=511, *, dir_fd=None)
        Open a file for low level IO.  Returns a file descriptor (integer).

        If dir_fd is not None, it should be a file descriptor open to a
directory,
          and path should be relative; path will then be relative to that
directory.
        dir_fd may not be implemented on your platform.
          If it is unavailable, using it will raise a NotImplementedError.

    openpty()
        Open a pseudo-terminal.

        Return a tuple of (master_fd, slave_fd) containing open file descriptors
        for both the master and slave ends.

    pathconf(path, name)
        Return the configuration limit name for the file or directory path.

        If there is no limit, return -1.
        On some platforms, path may also be specified as an open file
descriptor.
          If this functionality is unavailable, using it raises an exception.

    pipe()
        Create a pipe.

        Returns a tuple of two file descriptors:
          (read_fd, write_fd)

    popen(cmd, mode='r', buffering=-1)
        # Supply os.popen()

    posix_spawn(…)
        Execute the program specified by path in a new process.
```

```
        path
          Path of executable file.
        argv
          Tuple or list of strings.
        env
          Dictionary of strings mapping to strings.
        file_actions
          A sequence of file action tuples.
        setpgroup
          The pgroup to use with the POSIX_SPAWN_SETPGROUP flag.
        resetids
          If the value is `true` the POSIX_SPAWN_RESETIDS will be activated.
        setsid
          If the value is `true` the POSIX_SPAWN_SETSID or POSIX_SPAWN_SETSID_NP
will be activated.
        setsigmask
          The sigmask to use with the POSIX_SPAWN_SETSIGMASK flag.
        setsigdef
          The sigmask to use with the POSIX_SPAWN_SETSIGDEF flag.
        scheduler
          A tuple with the scheduler policy (optional) and parameters.

    posix_spawnp(…)
        Execute the program specified by path in a new process.

        path
          Path of executable file.
        argv
          Tuple or list of strings.
        env
          Dictionary of strings mapping to strings.
        file_actions
          A sequence of file action tuples.
        setpgroup
          The pgroup to use with the POSIX_SPAWN_SETPGROUP flag.
        resetids
          If the value is `True` the POSIX_SPAWN_RESETIDS will be activated.
        setsid
          If the value is `True` the POSIX_SPAWN_SETSID or POSIX_SPAWN_SETSID_NP
will be activated.
        setsigmask
          The sigmask to use with the POSIX_SPAWN_SETSIGMASK flag.
        setsigdef
          The sigmask to use with the POSIX_SPAWN_SETSIGDEF flag.
        scheduler
          A tuple with the scheduler policy (optional) and parameters.

    pread(fd, length, offset, /)
```

```
        Read a number of bytes from a file descriptor starting at a particular
offset.

        Read length bytes from file descriptor fd, starting at offset bytes from
        the beginning of the file.  The file offset remains unchanged.

    putenv(name, value, /)
        Change or add an environment variable.

    pwrite(fd, buffer, offset, /)
        Write bytes to a file descriptor starting at a particular offset.

        Write buffer to fd, starting at offset bytes from the beginning of
        the file.  Returns the number of bytes writte.  Does not change the
        current file offset.

    read(fd, length, /)
        Read from a file descriptor.  Returns a bytes object.

    readlink(path, *, dir_fd=None)
        Return a string representing the path to which the symbolic link points.

        If dir_fd is not None, it should be a file descriptor open to a
directory,
        and path should be relative; path will then be relative to that
directory.

        dir_fd may not be implemented on your platform.  If it is unavailable,
        using it will raise a NotImplementedError.

    readv(fd, buffers, /)
        Read from a file descriptor fd into an iterable of buffers.

        The buffers should be mutable buffers accepting bytes.
        readv will transfer data into each buffer until it is full
        and then move on to the next buffer in the sequence to hold
        the rest of the data.

        readv returns the total number of bytes read,
        which may be less than the total capacity of all the buffers.

    register_at_fork(…)
        Register callables to be called when forking a new process.

          before
            A callable to be called in the parent before the fork() syscall.
          after_in_child
            A callable to be called in the child after fork().
```

```
          after_in_parent
            A callable to be called in the parent after fork().

        'before' callbacks are called in reverse order.
        'after_in_child' and 'after_in_parent' callbacks are called in order.

    remove(path, *, dir_fd=None)
        Remove a file (same as unlink()).

        If dir_fd is not None, it should be a file descriptor open to a
directory,
          and path should be relative; path will then be relative to that
directory.
        dir_fd may not be implemented on your platform.
          If it is unavailable, using it will raise a NotImplementedError.

    removedirs(name)
        removedirs(name)

        Super-rmdir; remove a leaf directory and all empty intermediate
        ones.  Works like rmdir except that, if the leaf directory is
        successfully removed, directories corresponding to rightmost path
        segments will be pruned away until either the whole path is
        consumed or an error occurs.  Errors during this latter phase are
        ignored -- they generally mean that a directory was not empty.

    rename(src, dst, *, src_dir_fd=None, dst_dir_fd=None)
        Rename a file or directory.

        If either src_dir_fd or dst_dir_fd is not None, it should be a file
          descriptor open to a directory, and the respective path string (src or
dst)
          should be relative; the path will then be relative to that directory.
        src_dir_fd and dst_dir_fd, may not be implemented on your platform.
          If they are unavailable, using them will raise a NotImplementedError.

    renames(old, new)
        renames(old, new)

        Super-rename; create directories as necessary and delete any left
        empty.  Works like rename, except creation of any intermediate
        directories needed to make the new pathname good is attempted
        first.  After the rename, directories corresponding to rightmost
        path segments of the old name will be pruned until either the
        whole path is consumed or a nonempty directory is found.

        Note: this function can fail with the new directory structure made
        if you lack permissions needed to unlink the leaf directory or
```

```
        file.

    replace(src, dst, *, src_dir_fd=None, dst_dir_fd=None)
        Rename a file or directory, overwriting the destination.

        If either src_dir_fd or dst_dir_fd is not None, it should be a file
          descriptor open to a directory, and the respective path string (src or
dst)
          should be relative; the path will then be relative to that directory.
        src_dir_fd and dst_dir_fd, may not be implemented on your platform.
          If they are unavailable, using them will raise a NotImplementedError.

    rmdir(path, *, dir_fd=None)
        Remove a directory.

        If dir_fd is not None, it should be a file descriptor open to a
directory,
          and path should be relative; path will then be relative to that
directory.
        dir_fd may not be implemented on your platform.
          If it is unavailable, using it will raise a NotImplementedError.

    scandir(path=None)
        Return an iterator of DirEntry objects for given path.

        path can be specified as either str, bytes, or a path-like object.  If
path
        is bytes, the names of yielded DirEntry objects will also be bytes; in
        all other circumstances they will be str.

        If path is None, uses the path='.'.

    sched_get_priority_max(policy)
        Get the maximum scheduling priority for policy.

    sched_get_priority_min(policy)
        Get the minimum scheduling priority for policy.

    sched_yield()
        Voluntarily relinquish the CPU.

    sendfile(…)
        sendfile(out, in, offset, count) -> byteswritten
        sendfile(out, in, offset, count[, headers][, trailers], flags=0)
                    -> byteswritten
        Copy count bytes from file descriptor in to file descriptor out.

    set_blocking(fd, blocking, /)
```

```
    Set the blocking mode of the specified file descriptor.

    Set the O_NONBLOCK flag if blocking is False,
    clear the O_NONBLOCK flag otherwise.

set_inheritable(fd, inheritable, /)
    Set the inheritable flag of the specified file descriptor.

setegid(egid, /)
    Set the current process's effective group id.

seteuid(euid, /)
    Set the current process's effective user id.

setgid(gid, /)
    Set the current process's group id.

setgroups(groups, /)
    Set the groups of the current process to list.

setpgid(pid, pgrp, /)
    Call the system call setpgid(pid, pgrp).

setpgrp()
    Make the current process the leader of its process group.

setpriority(which, who, priority)
    Set program scheduling priority.

setregid(rgid, egid, /)
    Set the current process's real and effective group ids.

setreuid(ruid, euid, /)
    Set the current process's real and effective user ids.

setsid()
    Call the system call setsid().

setuid(uid, /)
    Set the current process's user id.

spawnl(mode, file, *args)
    spawnl(mode, file, *args) -> integer

    Execute file with arguments from args in a subprocess.
    If mode == P_NOWAIT return the pid of the process.
    If mode == P_WAIT return the process's exit code if it exits normally;
    otherwise return -SIG, where SIG is the signal that killed it.
```

```
spawnle(mode, file, *args)
    spawnle(mode, file, *args, env) -> integer

    Execute file with arguments from args in a subprocess with the
    supplied environment.
    If mode == P_NOWAIT return the pid of the process.
    If mode == P_WAIT return the process's exit code if it exits normally;
    otherwise return -SIG, where SIG is the signal that killed it.

spawnlp(mode, file, *args)
    spawnlp(mode, file, *args) -> integer

    Execute file (which is looked for along $PATH) with arguments from
    args in a subprocess with the supplied environment.
    If mode == P_NOWAIT return the pid of the process.
    If mode == P_WAIT return the process's exit code if it exits normally;
    otherwise return -SIG, where SIG is the signal that killed it.

spawnlpe(mode, file, *args)
    spawnlpe(mode, file, *args, env) -> integer

    Execute file (which is looked for along $PATH) with arguments from
    args in a subprocess with the supplied environment.
    If mode == P_NOWAIT return the pid of the process.
    If mode == P_WAIT return the process's exit code if it exits normally;
    otherwise return -SIG, where SIG is the signal that killed it.

spawnv(mode, file, args)
    spawnv(mode, file, args) -> integer

    Execute file with arguments from args in a subprocess.
    If mode == P_NOWAIT return the pid of the process.
    If mode == P_WAIT return the process's exit code if it exits normally;
    otherwise return -SIG, where SIG is the signal that killed it.

spawnve(mode, file, args, env)
    spawnve(mode, file, args, env) -> integer

    Execute file with arguments from args in a subprocess with the
    specified environment.
    If mode == P_NOWAIT return the pid of the process.
    If mode == P_WAIT return the process's exit code if it exits normally;
    otherwise return -SIG, where SIG is the signal that killed it.

spawnvp(mode, file, args)
    spawnvp(mode, file, args) -> integer
```

```
        Execute file (which is looked for along $PATH) with arguments from
        args in a subprocess.
        If mode == P_NOWAIT return the pid of the process.
        If mode == P_WAIT return the process's exit code if it exits normally;
        otherwise return -SIG, where SIG is the signal that killed it.

    spawnvpe(mode, file, args, env)
        spawnvpe(mode, file, args, env) -> integer

        Execute file (which is looked for along $PATH) with arguments from
        args in a subprocess with the supplied environment.
        If mode == P_NOWAIT return the pid of the process.
        If mode == P_WAIT return the process's exit code if it exits normally;
        otherwise return -SIG, where SIG is the signal that killed it.

    stat(path, *, dir_fd=None, follow_symlinks=True)
        Perform a stat system call on the given path.

          path
            Path to be examined; can be string, bytes, a path-like object or
            open-file-descriptor int.
          dir_fd
            If not None, it should be a file descriptor open to a directory,
            and path should be a relative string; path will then be relative to
            that directory.
          follow_symlinks
            If False, and the last element of the path is a symbolic link,
            stat will examine the symbolic link itself instead of the file
            the link points to.

        dir_fd and follow_symlinks may not be implemented
          on your platform.  If they are unavailable, using them will raise a
          NotImplementedError.

        It's an error to use dir_fd or follow_symlinks when specifying path as
          an open file descriptor.

    statvfs(path)
        Perform a statvfs system call on the given path.

        path may always be specified as a string.
        On some platforms, path may also be specified as an open file
descriptor.
          If this functionality is unavailable, using it raises an exception.

    strerror(code, /)
        Translate an error code to a message string.
```

```
    symlink(src, dst, target_is_directory=False, *, dir_fd=None)
        Create a symbolic link pointing to src named dst.

        target_is_directory is required on Windows if the target is to be
          interpreted as a directory.  (On Windows, symlink requires
          Windows 6.0 or greater, and raises a NotImplementedError otherwise.)
          target_is_directory is ignored on non-Windows platforms.

        If dir_fd is not None, it should be a file descriptor open to a
directory,
          and path should be relative; path will then be relative to that
directory.
        dir_fd may not be implemented on your platform.
          If it is unavailable, using it will raise a NotImplementedError.

    sync()
        Force write of everything to disk.

    sysconf(name, /)
        Return an integer-valued system configuration variable.

    system(command)
        Execute the command in a subshell.

    tcgetpgrp(fd, /)
        Return the process group associated with the terminal specified by fd.

    tcsetpgrp(fd, pgid, /)
        Set the process group associated with the terminal specified by fd.

    times()
        Return a collection containing process timing information.

        The object returned behaves like a named tuple with these fields:
          (utime, stime, cutime, cstime, elapsed_time)
        All fields are floating point numbers.

    truncate(path, length)
        Truncate a file, specified by path, to a specific length.

        On some platforms, path may also be specified as an open file
descriptor.
          If this functionality is unavailable, using it raises an exception.

    ttyname(fd, /)
        Return the name of the terminal device connected to 'fd'.

        fd
```

```
          Integer file descriptor handle.

    umask(mask, /)
        Set the current numeric umask and return the previous umask.

    uname()
        Return an object identifying the current operating system.

        The object behaves like a named tuple with the following fields:
          (sysname, nodename, release, version, machine)

    unlink(path, *, dir_fd=None)
        Remove a file (same as remove()).

        If dir_fd is not None, it should be a file descriptor open to a
directory,
          and path should be relative; path will then be relative to that
directory.
        dir_fd may not be implemented on your platform.
          If it is unavailable, using it will raise a NotImplementedError.

    unsetenv(name, /)
        Delete an environment variable.

    urandom(size, /)
        Return a bytes object containing random bytes suitable for cryptographic
use.

    utime(…)
        Set the access and modified time of path.

        path may always be specified as a string.
        On some platforms, path may also be specified as an open file
descriptor.
          If this functionality is unavailable, using it raises an exception.

        If times is not None, it must be a tuple (atime, mtime);
            atime and mtime should be expressed as float seconds since the
epoch.
        If ns is specified, it must be a tuple (atime_ns, mtime_ns);
            atime_ns and mtime_ns should be expressed as integer nanoseconds
            since the epoch.
        If times is None and ns is unspecified, utime uses the current time.
        Specifying tuples for both times and ns is an error.

        If dir_fd is not None, it should be a file descriptor open to a
directory,
          and path should be relative; path will then be relative to that
```

```
directory.
        If follow_symlinks is False, and the last element of the path is a
symbolic
          link, utime will modify the symbolic link itself instead of the file
the
          link points to.
        It is an error to use dir_fd or follow_symlinks when specifying path
          as an open file descriptor.
        dir_fd and follow_symlinks may not be available on your platform.
          If they are unavailable, using them will raise a NotImplementedError.

    wait()
        Wait for completion of a child process.

        Returns a tuple of information about the child process:
            (pid, status)

    wait3(options)
        Wait for completion of a child process.

        Returns a tuple of information about the child process:
          (pid, status, rusage)

    wait4(pid, options)
        Wait for completion of a specific child process.

        Returns a tuple of information about the child process:
          (pid, status, rusage)

    waitpid(pid, options, /)
        Wait for completion of a given child process.

        Returns a tuple of information regarding the child process:
            (pid, status)

        The options argument is ignored on Windows.

    walk(top, topdown=True, onerror=None, followlinks=False)
        Directory tree generator.

        For each directory in the directory tree rooted at top (including top
        itself, but excluding '.' and '..'), yields a 3-tuple

            dirpath, dirnames, filenames

        dirpath is a string, the path to the directory.  dirnames is a list of
        the names of the subdirectories in dirpath (excluding '.' and '..').
        filenames is a list of the names of the non-directory files in dirpath.
```

```
        Note that the names in the lists are just names, with no path
components.
        To get a full path (which begins with top) to a file or directory in
        dirpath, do os.path.join(dirpath, name).

        If optional arg 'topdown' is true or not specified, the triple for a
        directory is generated before the triples for any of its subdirectories
        (directories are generated top down).  If topdown is false, the triple
        for a directory is generated after the triples for all of its
        subdirectories (directories are generated bottom up).

        When topdown is true, the caller can modify the dirnames list in-place
        (e.g., via del or slice assignment), and walk will only recurse into the
        subdirectories whose names remain in dirnames; this can be used to prune
the
        search, or to impose a specific order of visiting.  Modifying dirnames
when
        topdown is false has no effect on the behavior of os.walk(), since the
        directories in dirnames have already been generated by the time dirnames
        itself is generated. No matter the value of topdown, the list of
        subdirectories is retrieved before the tuples for the directory and its
        subdirectories are generated.

        By default errors from the os.scandir() call are ignored.  If
        optional arg 'onerror' is specified, it should be a function; it
        will be called with one argument, an OSError instance.  It can
        report the error to continue with the walk, or raise the exception
        to abort the walk.  Note that the filename is available as the
        filename attribute of the exception object.

        By default, os.walk does not follow symbolic links to subdirectories on
        systems that support them.  In order to get this functionality, set the
        optional argument 'followlinks' to true.

        Caution:  if you pass a relative pathname for top, don't change the
        current working directory between resumptions of walk.  walk never
        changes the current directory, and assumes that the client doesn't
        either.

        Example:

        import os
        from os.path import join, getsize
        for root, dirs, files in os.walk('python/Lib/email'):
            print(root, "consumes", end="")
            print(sum(getsize(join(root, name)) for name in files), end="")
            print("bytes in", len(files), "non-directory files")
            if 'CVS' in dirs:
```

```
                dirs.remove('CVS')  # don't visit CVS directories

    write(fd, data, /)
        Write a bytes object to a file descriptor.

    writev(fd, buffers, /)
        Iterate over buffers, and write the contents of each to a file
descriptor.

        Returns the total number of bytes written.
        buffers must be a sequence of bytes-like objects.

DATA
    CLD_CONTINUED = 6
    CLD_DUMPED = 3
    CLD_EXITED = 1
    CLD_TRAPPED = 4
    EX_CANTCREAT = 73
    EX_CONFIG = 78
    EX_DATAERR = 65
    EX_IOERR = 74
    EX_NOHOST = 68
    EX_NOINPUT = 66
    EX_NOPERM = 77
    EX_NOUSER = 67
    EX_OK = 0
    EX_OSERR = 71
    EX_OSFILE = 72
    EX_PROTOCOL = 76
    EX_SOFTWARE = 70
    EX_TEMPFAIL = 75
    EX_UNAVAILABLE = 69
    EX_USAGE = 64
    F_LOCK = 1
    F_OK = 0
    F_TEST = 3
    F_TLOCK = 2
    F_ULOCK = 0
    NGROUPS_MAX = 16
    O_ACCMODE = 3
    O_APPEND = 8
    O_ASYNC = 64
    O_CLOEXEC = 16777216
    O_CREAT = 512
    O_DIRECTORY = 1048576
    O_DSYNC = 4194304
    O_EXCL = 2048
    O_EXLOCK = 32
```

```
O_NDELAY = 4
O_NOCTTY = 131072
O_NOFOLLOW = 256
O_NONBLOCK = 4
O_RDONLY = 0
O_RDWR = 2
O_SHLOCK = 16
O_SYNC = 128
O_TRUNC = 1024
O_WRONLY = 1
POSIX_SPAWN_CLOSE = 1
POSIX_SPAWN_DUP2 = 2
POSIX_SPAWN_OPEN = 0
PRIO_PGRP = 1
PRIO_PROCESS = 0
PRIO_USER = 2
P_ALL = 0
P_NOWAIT = 1
P_NOWAITO = 1
P_PGID = 2
P_PID = 1
P_WAIT = 0
RTLD_GLOBAL = 8
RTLD_LAZY = 1
RTLD_LOCAL = 4
RTLD_NODELETE = 128
RTLD_NOLOAD = 16
RTLD_NOW = 2
R_OK = 4
SCHED_FIFO = 4
SCHED_OTHER = 1
SCHED_RR = 2
SEEK_CUR = 1
SEEK_END = 2
SEEK_SET = 0
ST_NOSUID = 2
ST_RDONLY = 1
TMP_MAX = 308915776
WCONTINUED = 16
WEXITED = 4
WNOHANG = 1
WNOWAIT = 32
WSTOPPED = 8
WUNTRACED = 2
W_OK = 2
X_OK = 1
__all__ = ['altsep', 'curdir', 'pardir', 'sep', 'pathsep', 'linesep', …
altsep = None
```

```
    confstr_names = {'CS_PATH': 1, 'CS_XBS5_ILP32_OFF32_CFLAGS': 20, 'CS_X…
    curdir = '.'
    defpath = '/bin:/usr/bin'
    devnull = '/dev/null'
    environ = environ({'TERM_PROGRAM': 'Apple_Terminal', 'SHEL…END': 'mo…
    environb = environ({b'TERM_PROGRAM': b'Apple_Terminal', b'S…ND': b'm…
    extsep = '.'
    linesep = '\n'
    name = 'posix'
    pardir = '..'
    pathconf_names = {'PC_ALLOC_SIZE_MIN': 16, 'PC_ASYNC_IO': 17, 'PC_CHOW…
    pathsep = ':'
    sep = '/'
    supports_bytes_environ = True
    sysconf_names = {'SC_2_CHAR_TERM': 20, 'SC_2_C_BIND': 18, 'SC_2_C_DEV'…

FILE
    /Users/rbasnet/miniconda3/lib/python3.8/os.py
```

[5]:
```
os.sep
```

[5]:
```
'/'
```

[14]:
```
# create someDir in current folder
os.mkdir('someDir')
```

[15]:
```
# remove directory and its contents
os.rmdir('someDir')
```

[16]:
```
import time
```

[17]:
```
help(time)
```

```
Help on built-in module time:

NAME
    time - This module provides various functions to manipulate time values.

DESCRIPTION
    There are two standard representations of time.  One is the number
    of seconds since the Epoch, in UTC (a.k.a. GMT).  It may be an integer
    or a floating point number (to represent fractions of seconds).
    The Epoch is system-defined; on Unix, it is generally January 1st, 1970.
    The actual value can be retrieved by calling gmtime(0).

    The other representation is a tuple of 9 integers giving local time.
```

```
    The tuple items are:
      year (including century, e.g. 1998)
      month (1-12)
      day (1-31)
      hours (0-23)
      minutes (0-59)
      seconds (0-59)
      weekday (0-6, Monday is 0)
      Julian day (day in the year, 1-366)
      DST (Daylight Savings Time) flag (-1, 0 or 1)
    If the DST flag is 0, the time is given in the regular time zone;
    if it is 1, the time is given in the DST time zone;
    if it is -1, mktime() should guess based on the date and time.

CLASSES
    builtins.tuple(builtins.object)
        struct_time

    class struct_time(builtins.tuple)
     |  struct_time(iterable=(), /)
     |
     |  The time value as returned by gmtime(), localtime(), and strptime(), and
     |  accepted by asctime(), mktime() and strftime().  May be considered as a
     |  sequence of 9 integers.
     |
     |  Note that several fields' values are not the same as those defined by
     |  the C language standard for struct tm.  For example, the value of the
     |  field tm_year is the actual year, not year - 1900.  See individual
     |  fields' descriptions for details.
     |
     |  Method resolution order:
     |      struct_time
     |      builtins.tuple
     |      builtins.object
     |
     |  Methods defined here:
     |
     |  __reduce__(…)
     |      Helper for pickle.
     |
     |  __repr__(self, /)
     |      Return repr(self).
     |
     |  ----------------------------------------------------------------------
     |  Static methods defined here:
     |
     |  __new__(*args, **kwargs) from builtins.type
     |      Create and return a new object.  See help(type) for accurate
```

```
signature.
     |
     |  ----------------------------------------------------------------------
     |  Data descriptors defined here:
     |
     |  tm_gmtoff
     |      offset from UTC in seconds
     |
     |  tm_hour
     |      hours, range [0, 23]
     |
     |  tm_isdst
     |      1 if summer time is in effect, 0 if not, and -1 if unknown
     |
     |  tm_mday
     |      day of month, range [1, 31]
     |
     |  tm_min
     |      minutes, range [0, 59]
     |
     |  tm_mon
     |      month of year, range [1, 12]
     |
     |  tm_sec
     |      seconds, range [0, 61])
     |
     |  tm_wday
     |      day of week, range [0, 6], Monday is 0
     |
     |  tm_yday
     |      day of year, range [1, 366]
     |
     |  tm_year
     |      year, for example, 1993
     |
     |  tm_zone
     |      abbreviation of timezone name
     |
     |  ----------------------------------------------------------------------
     |  Data and other attributes defined here:
     |
     |  n_fields = 11
     |
     |  n_sequence_fields = 9
     |
     |  n_unnamed_fields = 0
     |
     |  ----------------------------------------------------------------------
```

```
|  Methods inherited from builtins.tuple:
|
|  __add__(self, value, /)
|      Return self+value.
|
|  __contains__(self, key, /)
|      Return key in self.
|
|  __eq__(self, value, /)
|      Return self==value.
|
|  __ge__(self, value, /)
|      Return self>=value.
|
|  __getattribute__(self, name, /)
|      Return getattr(self, name).
|
|  __getitem__(self, key, /)
|      Return self[key].
|
|  __getnewargs__(self, /)
|
|  __gt__(self, value, /)
|      Return self>value.
|
|  __hash__(self, /)
|      Return hash(self).
|
|  __iter__(self, /)
|      Implement iter(self).
|
|  __le__(self, value, /)
|      Return self<=value.
|
|  __len__(self, /)
|      Return len(self).
|
|  __lt__(self, value, /)
|      Return self<value.
|
|  __mul__(self, value, /)
|      Return self*value.
|
|  __ne__(self, value, /)
|      Return self!=value.
|
|  __rmul__(self, value, /)
|      Return value*self.
```

```
 |
 |  count(self, value, /)
 |      Return number of occurrences of value.
 |
 |  index(self, value, start=0, stop=9223372036854775807, /)
 |      Return first index of value.
 |
 |      Raises ValueError if the value is not present.

FUNCTIONS
    asctime(…)
        asctime([tuple]) -> string

        Convert a time tuple to a string, e.g. 'Sat Jun 06 16:26:11 1998'.
        When the time tuple is not present, current time as returned by
localtime()
        is used.

    ctime(…)
        ctime(seconds) -> string

        Convert a time in seconds since the Epoch to a string in local time.
        This is equivalent to asctime(localtime(seconds)). When the time tuple
is
        not present, current time as returned by localtime() is used.

    get_clock_info(…)
        get_clock_info(name: str) -> dict

        Get information of the specified clock.

    gmtime(…)
        gmtime([seconds]) -> (tm_year, tm_mon, tm_mday, tm_hour, tm_min,
                              tm_sec, tm_wday, tm_yday, tm_isdst)

        Convert seconds since the Epoch to a time tuple expressing UTC (a.k.a.
        GMT).  When 'seconds' is not passed in, convert the current time
instead.

        If the platform supports the tm_gmtoff and tm_zone, they are available
as
        attributes only.

    localtime(…)
        localtime([seconds]) -> (tm_year,tm_mon,tm_mday,tm_hour,tm_min,
                                 tm_sec,tm_wday,tm_yday,tm_isdst)

        Convert seconds since the Epoch to a time tuple expressing local time.
```

```
    When 'seconds' is not passed in, convert the current time instead.

mktime(…)
    mktime(tuple) -> floating point number

    Convert a time tuple in local time to seconds since the Epoch.
    Note that mktime(gmtime(0)) will not generally return zero for most
    time zones; instead the returned value will either be equal to that
    of the timezone or altzone attributes on the time module.

monotonic(…)
    monotonic() -> float

    Monotonic clock, cannot go backward.

monotonic_ns(…)
    monotonic_ns() -> int

    Monotonic clock, cannot go backward, as nanoseconds.

perf_counter(…)
    perf_counter() -> float

    Performance counter for benchmarking.

perf_counter_ns(…)
    perf_counter_ns() -> int

    Performance counter for benchmarking as nanoseconds.

process_time(…)
    process_time() -> float

    Process time for profiling: sum of the kernel and user-space CPU time.

process_time_ns(…)
    process_time() -> int

    Process time for profiling as nanoseconds:
    sum of the kernel and user-space CPU time.

sleep(…)
    sleep(seconds)

    Delay execution for a given number of seconds.  The argument may be
    a floating point number for subsecond precision.

strftime(…)
```

```
        strftime(format[, tuple]) -> string

        Convert a time tuple to a string according to a format specification.
        See the library reference manual for formatting codes. When the time
tuple
        is not present, current time as returned by localtime() is used.

        Commonly used format codes:

        %Y  Year with century as a decimal number.
        %m  Month as a decimal number [01,12].
        %d  Day of the month as a decimal number [01,31].
        %H  Hour (24-hour clock) as a decimal number [00,23].
        %M  Minute as a decimal number [00,59].
        %S  Second as a decimal number [00,61].
        %z  Time zone offset from UTC.
        %a  Locale's abbreviated weekday name.
        %A  Locale's full weekday name.
        %b  Locale's abbreviated month name.
        %B  Locale's full month name.
        %c  Locale's appropriate date and time representation.
        %I  Hour (12-hour clock) as a decimal number [01,12].
        %p  Locale's equivalent of either AM or PM.

        Other codes may be available on your platform.  See documentation for
        the C library strftime function.

    strptime(...)
        strptime(string, format) -> struct_time

        Parse a string to a time tuple according to a format specification.
        See the library reference manual for formatting codes (same as
        strftime()).

        Commonly used format codes:

        %Y  Year with century as a decimal number.
        %m  Month as a decimal number [01,12].
        %d  Day of the month as a decimal number [01,31].
        %H  Hour (24-hour clock) as a decimal number [00,23].
        %M  Minute as a decimal number [00,59].
        %S  Second as a decimal number [00,61].
        %z  Time zone offset from UTC.
        %a  Locale's abbreviated weekday name.
        %A  Locale's full weekday name.
        %b  Locale's abbreviated month name.
        %B  Locale's full month name.
        %c  Locale's appropriate date and time representation.
```

```
        %I  Hour (12-hour clock) as a decimal number [01,12].
        %p  Locale's equivalent of either AM or PM.

        Other codes may be available on your platform.  See documentation for
        the C library strftime function.

    time(…)
        time() -> floating point number

        Return the current time in seconds since the Epoch.
        Fractions of a second may be present if the system clock provides them.

    time_ns(…)
        time_ns() -> int

        Return the current time in nanoseconds since the Epoch.

    tzset(…)
        tzset()

        Initialize, or reinitialize, the local timezone to the value stored in
        os.environ['TZ']. The TZ environment variable should be specified in
        standard Unix timezone format as documented in the tzset man page
        (eg. 'US/Eastern', 'Europe/Amsterdam'). Unknown timezones will silently
        fall back to UTC. If the TZ environment variable is not set, the local
        timezone is set to the systems best guess of wallclock time.
        Changing the TZ environment variable without calling tzset *may* change
        the local timezone used by methods such as localtime, but this behaviour
        should not be relied on.

DATA
    altzone = 21600
    daylight = 1
    timezone = 25200
    tzname = ('MST', 'MDT')

FILE
    (built-in)
```

[18]:
```
time.localtime()
```

[18]:
```
time.struct_time(tm_year=2021, tm_mon=9, tm_mday=7, tm_hour=21, tm_min=15,
tm_sec=5, tm_wday=1, tm_yday=250, tm_isdst=1)
```

[19]:
```
import sys
```

[20]:
```
help(sys)
```

```
Help on built-in module sys:

NAME
    sys

MODULE REFERENCE
    https://docs.python.org/3.8/library/sys

    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 some objects used or maintained by the
    interpreter and to functions that interact strongly with the interpreter.

    Dynamic objects:

    argv -- command line arguments; argv[0] is the script pathname if known
    path -- module search path; path[0] is the script directory, else ''
    modules -- dictionary of loaded modules

    displayhook -- called to show results in an interactive session
    excepthook -- called to handle any uncaught exception other than SystemExit
      To customize printing in an interactive session or to install a custom
      top-level exception handler, assign other functions to replace these.

    stdin -- standard input file object; used by input()
    stdout -- standard output file object; used by print()
    stderr -- standard error object; used for error messages
      By assigning other file objects (or objects that behave like files)
      to these, it is possible to redirect all of the interpreter's I/O.

    last_type -- type of last uncaught exception
    last_value -- value of last uncaught exception
    last_traceback -- traceback of last uncaught exception
      These three are only available in an interactive session after a
      traceback has been printed.

    Static objects:

    builtin_module_names -- tuple of module names built into this interpreter
    copyright -- copyright notice pertaining to this interpreter
    exec_prefix -- prefix used to find the machine-specific Python library
```

```
    executable -- absolute path of the executable binary of the Python
interpreter
    float_info -- a named tuple with information about the float implementation.
    float_repr_style -- string indicating the style of repr() output for floats
    hash_info -- a named tuple with information about the hash algorithm.
    hexversion -- version information encoded as a single integer
    implementation -- Python implementation information.
    int_info -- a named tuple with information about the int implementation.
    maxsize -- the largest supported length of containers.
    maxunicode -- the value of the largest Unicode code point
    platform -- platform identifier
    prefix -- prefix used to find the Python library
    thread_info -- a named tuple with information about the thread
implementation.
    version -- the version of this interpreter as a string
    version_info -- version information as a named tuple
    __stdin__ -- the original stdin; don't touch!
    __stdout__ -- the original stdout; don't touch!
    __stderr__ -- the original stderr; don't touch!
    __displayhook__ -- the original displayhook; don't touch!
    __excepthook__ -- the original excepthook; don't touch!

    Functions:

    displayhook() -- print an object to the screen, and save it in builtins._
    excepthook() -- print an exception and its traceback to sys.stderr
    exc_info() -- return thread-safe information about the current exception
    exit() -- exit the interpreter by raising SystemExit
    getdlopenflags() -- returns flags to be used for dlopen() calls
    getprofile() -- get the global profiling function
    getrefcount() -- return the reference count for an object (plus one :-)
    getrecursionlimit() -- return the max recursion depth for the interpreter
    getsizeof() -- return the size of an object in bytes
    gettrace() -- get the global debug tracing function
    setcheckinterval() -- control how often the interpreter checks for events
    setdlopenflags() -- set the flags to be used for dlopen() calls
    setprofile() -- set the global profiling function
    setrecursionlimit() -- set the max recursion depth for the interpreter
    settrace() -- set the global debug tracing function

FUNCTIONS
    __breakpointhook__ = breakpointhook(…)
        breakpointhook(*args, **kws)

        This hook function is called by built-in breakpoint().

    __displayhook__ = displayhook(object, /)
        Print an object to sys.stdout and also save it in builtins._
```

```
__excepthook__ = excepthook(exctype, value, traceback, /)
    Handle an exception by displaying it with a traceback on sys.stderr.

__unraisablehook__ = unraisablehook(unraisable, /)
    Handle an unraisable exception.

    The unraisable argument has the following attributes:

    * exc_type: Exception type.
    * exc_value: Exception value, can be None.
    * exc_traceback: Exception traceback, can be None.
    * err_msg: Error message, can be None.
    * object: Object causing the exception, can be None.

addaudithook(hook)
    Adds a new audit hook callback.

audit(…)
    audit(event, *args)

    Passes the event to any audit hooks that are attached.

breakpointhook(…)
    breakpointhook(*args, **kws)

    This hook function is called by built-in breakpoint().

call_tracing(func, args, /)
    Call func(*args), while tracing is enabled.

    The tracing state is saved, and restored afterwards.  This is intended
    to be called from a debugger from a checkpoint, to recursively debug
    some other code.

callstats()
    Return a tuple of function call statistics.

    A tuple is returned only if CALL_PROFILE was defined when Python was
    built.  Otherwise, this returns None.

    When enabled, this function returns detailed, implementation-specific
    details about the number of function calls executed. The return value
    is a 11-tuple where the entries in the tuple are counts of:
    0. all function calls
    1. calls to PyFunction_Type objects
    2. PyFunction calls that do not create an argument tuple
    3. PyFunction calls that do not create an argument tuple
```

```
       and bypass PyEval_EvalCodeEx()
    4. PyMethod calls
    5. PyMethod calls on bound methods
    6. PyType calls
    7. PyCFunction calls
    8. generator calls
    9. All other calls
    10. Number of stack pops performed by call_function()

exc_info()
    Return current exception information: (type, value, traceback).

    Return information about the most recent exception caught by an except
    clause in the current stack frame or in an older stack frame.

exit(status=None, /)
    Exit the interpreter by raising SystemExit(status).

    If the status is omitted or None, it defaults to zero (i.e., success).
    If the status is an integer, it will be used as the system exit status.
    If it is another kind of object, it will be printed and the system
    exit status will be one (i.e., failure).

get_asyncgen_hooks()
    Return the installed asynchronous generators hooks.

    This returns a namedtuple of the form (firstiter, finalizer).

get_coroutine_origin_tracking_depth()
    Check status of origin tracking for coroutine objects in this thread.

getallocatedblocks()
    Return the number of memory blocks currently allocated.

getcheckinterval()
    Return the current check interval; see sys.setcheckinterval().

getdefaultencoding()
    Return the current default encoding used by the Unicode implementation.

getdlopenflags()
    Return the current value of the flags that are used for dlopen calls.

    The flag constants are defined in the os module.

getfilesystemencodeerrors()
    Return the error mode used Unicode to OS filename conversion.
```

```
getfilesystemencoding()
    Return the encoding used to convert Unicode filenames to OS filenames.

getprofile()
    Return the profiling function set with sys.setprofile.

    See the profiler chapter in the library manual.

getrecursionlimit()
    Return the current value of the recursion limit.

    The recursion limit is the maximum depth of the Python interpreter
    stack.  This limit prevents infinite recursion from causing an overflow
    of the C stack and crashing Python.

getrefcount(object, /)
    Return the reference count of object.

    The count returned is generally one higher than you might expect,
    because it includes the (temporary) reference as an argument to
    getrefcount().

getsizeof(…)
    getsizeof(object [, default]) -> int

    Return the size of object in bytes.

getswitchinterval()
    Return the current thread switch interval; see sys.setswitchinterval().

gettrace()
    Return the global debug tracing function set with sys.settrace.

    See the debugger chapter in the library manual.

intern(string, /)
    ``Intern'' the given string.

    This enters the string in the (global) table of interned strings whose
    purpose is to speed up dictionary lookups. Return the string itself or
    the previously interned string object with the same value.

is_finalizing()
    Return True if Python is exiting.

set_asyncgen_hooks(…)
    set_asyncgen_hooks(* [, firstiter] [, finalizer])
```

```
    Set a finalizer for async generators objects.

set_coroutine_origin_tracking_depth(depth)
    Enable or disable origin tracking for coroutine objects in this thread.

    Coroutine objects will track 'depth' frames of traceback information
    about where they came from, available in their cr_origin attribute.

    Set a depth of 0 to disable.

setcheckinterval(n, /)
    Set the async event check interval to n instructions.

    This tells the Python interpreter to check for asynchronous events
    every n instructions.

    This also affects how often thread switches occur.

setdlopenflags(flags, /)
    Set the flags used by the interpreter for dlopen calls.

    This is used, for example, when the interpreter loads extension
    modules. Among other things, this will enable a lazy resolving of
    symbols when importing a module, if called as sys.setdlopenflags(0).
    To share symbols across extension modules, call as
    sys.setdlopenflags(os.RTLD_GLOBAL).  Symbolic names for the flag
    modules can be found in the os module (RTLD_xxx constants, e.g.
    os.RTLD_LAZY).

setprofile(…)
    setprofile(function)

    Set the profiling function.  It will be called on each function call
    and return.  See the profiler chapter in the library manual.

setrecursionlimit(limit, /)
    Set the maximum depth of the Python interpreter stack to n.

    This limit prevents infinite recursion from causing an overflow of the C
    stack and crashing Python.  The highest possible limit is platform-
    dependent.

setswitchinterval(interval, /)
    Set the ideal thread switching delay inside the Python interpreter.

    The actual frequency of switching threads can be lower if the
    interpreter executes long sequences of uninterruptible code
    (this is implementation-specific and workload-dependent).
```

```
        The parameter must represent the desired switching delay in seconds
        A typical value is 0.005 (5 milliseconds).

    settrace(…)
        settrace(function)

        Set the global debug tracing function.  It will be called on each
        function call.  See the debugger chapter in the library manual.

    unraisablehook(unraisable, /)
        Handle an unraisable exception.

        The unraisable argument has the following attributes:

        * exc_type: Exception type.
        * exc_value: Exception value, can be None.
        * exc_traceback: Exception traceback, can be None.
        * err_msg: Error message, can be None.
        * object: Object causing the exception, can be None.

DATA
    __stderr__ = <_io.TextIOWrapper name='<stderr>' mode='w' encoding='utf…
    __stdin__ = <_io.TextIOWrapper name='<stdin>' mode='r' encoding='utf-8…
    __stdout__ = <_io.TextIOWrapper name='<stdout>' mode='w' encoding='utf…
    abiflags = ''
    api_version = 1013
    argv = ['/Users/rbasnet/miniconda3/lib/python3.8/site-packages/ipykern…
    base_exec_prefix = '/Users/rbasnet/miniconda3'
    base_prefix = '/Users/rbasnet/miniconda3'
    builtin_module_names = ('_abc', '_ast', '_codecs', '_collections', '_f…
    byteorder = 'little'
    copyright = 'Copyright (c) 2001-2021 Python Software Foundati…ematis…
    displayhook = <ipykernel.displayhook.ZMQShellDisplayHook object>
    dont_write_bytecode = False
    exec_prefix = '/Users/rbasnet/miniconda3'
    executable = '/Users/rbasnet/miniconda3/bin/python'
    flags = sys.flags(debug=0, inspect=0, interactive=0, opt…ation=1, is…
    float_info = sys.float_info(max=1.7976931348623157e+308, max_…epsilo…
    float_repr_style = 'short'
    hash_info = sys.hash_info(width=64, modulus=2305843009213693…iphash2…
    hexversion = 50858992
    implementation = namespace(_multiarch='darwin', cache_tag='cpytho…no…
    int_info = sys.int_info(bits_per_digit=30, sizeof_digit=4)
    last_value = FileExistsError(17, 'File exists')
    maxsize = 9223372036854775807
    maxunicode = 1114111
    meta_path = [<class '_frozen_importlib.BuiltinImporter'>, <class '_fro…
```

```
    modules = {'IPython': <module 'IPython' from '/Users/rbasnet/miniconda…
    path = ['/Users/rbasnet/CMU/projects/Python-Fundamentals', '/Users/rba…
    path_hooks = [<class 'zipimport.zipimporter'>, <function FileFinder.pa…
    path_importer_cache = {'/Users/rbasnet/.ipython': FileFinder('/Users/r…
    platform = 'darwin'
    prefix = '/Users/rbasnet/miniconda3'
    ps1 = 'In : '
    ps2 = '…: '
    ps3 = 'Out: '
    pycache_prefix = None
    stderr = <ipykernel.iostream.OutStream object>
    stdin = <_io.TextIOWrapper name='<stdin>' mode='r' encoding='utf-8'>
    stdout = <ipykernel.iostream.OutStream object>
    thread_info = sys.thread_info(name='pthread', lock='mutex+cond', versi…
    version = '3.8.11 (default, Aug  6 2021, 08:56:27) \n[Clang 10.0.0 ]'
    version_info = sys.version_info(major=3, minor=8, micro=11, releaselev…
    warnoptions = []

FILE
    (built-in)
```

[21]:
```
import string
```

[22]:
```
help(string)
```

```
Help on module string:

NAME
    string - A collection of string constants.

MODULE REFERENCE
    https://docs.python.org/3.8/library/string

    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
    Public module variables:

    whitespace -- a string containing all ASCII whitespace
    ascii_lowercase -- a string containing all ASCII lowercase letters
    ascii_uppercase -- a string containing all ASCII uppercase letters
    ascii_letters -- a string containing all ASCII letters
```

```
    digits -- a string containing all ASCII decimal digits
    hexdigits -- a string containing all ASCII hexadecimal digits
    octdigits -- a string containing all ASCII octal digits
    punctuation -- a string containing all ASCII punctuation characters
    printable -- a string containing all ASCII characters considered printable

CLASSES
    builtins.object
        Formatter
        Template

    class Formatter(builtins.object)
     |  Methods defined here:
     |
     |  check_unused_args(self, used_args, args, kwargs)
     |
     |  convert_field(self, value, conversion)
     |
     |  format(self, format_string, /, *args, **kwargs)
     |
     |  format_field(self, value, format_spec)
     |
     |  get_field(self, field_name, args, kwargs)
     |      # given a field_name, find the object it references.
     |      #  field_name:   the field being looked up, e.g. "0.name"
     |      #                 or "lookup[3]"
     |      #  used_args:    a set of which args have been used
     |      #  args, kwargs: as passed in to vformat
     |
     |  get_value(self, key, args, kwargs)
     |
     |  parse(self, format_string)
     |      # returns an iterable that contains tuples of the form:
     |      # (literal_text, field_name, format_spec, conversion)
     |      # literal_text can be zero length
     |      # field_name can be None, in which case there's no
     |      #  object to format and output
     |      # if field_name is not None, it is looked up, formatted
     |      #  with format_spec and conversion and then used
     |
     |  vformat(self, format_string, args, kwargs)
     |
     |  ----------------------------------------------------------------------
     |  Data descriptors defined here:
     |
     |  __dict__
     |      dictionary for instance variables (if defined)
     |
```

```
 |  __weakref__
 |      list of weak references to the object (if defined)

class Template(builtins.object)
 |  Template(template)
 |
 |  A string class for supporting $-substitutions.
 |
 |  Methods defined here:
 |
 |  __init__(self, template)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  safe_substitute(self, mapping={}, /, **kws)
 |
 |  substitute(self, mapping={}, /, **kws)
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  braceidpattern = None
 |
 |  delimiter = '$'
 |
 |  flags = re.IGNORECASE
 |
 |  idpattern = '(?a:[_a-z][_a-z0-9]*)'
 |
 |  pattern = re.compile('\n    \\$(?:\n
(?P<escaped>\\$)…ced>(?a:[…

FUNCTIONS
    capwords(s, sep=None)
        capwords(s [,sep]) -> string

        Split the argument into words using split, capitalize each
        word using capitalize, and join the capitalized words using
        join.  If the optional second argument sep is absent or None,
        runs of whitespace characters are replaced by a single space
```

```
        and leading and trailing whitespace are removed, otherwise
        sep is used to split and join the words.

DATA
    __all__ = ['ascii_letters', 'ascii_lowercase', 'ascii_uppercase', 'cap…
    ascii_letters = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
    ascii_lowercase = 'abcdefghijklmnopqrstuvwxyz'
    ascii_uppercase = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    digits = '0123456789'
    hexdigits = '0123456789abcdefABCDEF'
    octdigits = '01234567'
    printable = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTU…
    punctuation = '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
    whitespace = ' \t\n\r\x0b\x0c'

FILE
    /Users/rbasnet/miniconda3/lib/python3.8/string.py
```

[23]:
```
string.ascii_letters
```

[23]:
```
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

[ ]:
```

```