Python les-materialen

# Math and Random Modules

Python comes with a built in math module and random module. In this lecture we will give a brief tour of their capabilities. Usually you can simply look up the function call you are looking for in the online documentation.

* [Math Module](https://docs.python.org/3/library/math.html)
* [Random Module](https://docs.python.org/3/library/random.html)

We won’t go through every function available in these modules since there are so many, but we will show some useful ones.

## Useful Math Functions

import math

help(math)

Help on built-in module math:  
  
NAME  
 math  
  
DESCRIPTION  
 This module is always available. It provides access to the  
 mathematical functions defined by the C standard.  
  
FUNCTIONS  
 acos(...)  
 acos(x)  
   
 Return the arc cosine (measured in radians) of x.  
   
 acosh(...)  
 acosh(x)  
   
 Return the inverse hyperbolic cosine of x.  
   
 asin(...)  
 asin(x)  
   
 Return the arc sine (measured in radians) of x.  
   
 asinh(...)  
 asinh(x)  
   
 Return the inverse hyperbolic sine of x.  
   
 atan(...)  
 atan(x)  
   
 Return the arc tangent (measured in radians) of x.  
   
 atan2(...)  
 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(...)  
 atanh(x)  
   
 Return the inverse hyperbolic tangent of x.  
   
 ceil(...)  
 ceil(x)  
   
 Return the ceiling of x as an Integral.  
 This is the smallest integer >= x.  
   
 copysign(...)  
 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(...)  
 cos(x)  
   
 Return the cosine of x (measured in radians).  
   
 cosh(...)  
 cosh(x)  
   
 Return the hyperbolic cosine of x.  
   
 degrees(...)  
 degrees(x)  
   
 Convert angle x from radians to degrees.  
   
 erf(...)  
 erf(x)  
   
 Error function at x.  
   
 erfc(...)  
 erfc(x)  
   
 Complementary error function at x.  
   
 exp(...)  
 exp(x)  
   
 Return e raised to the power of x.  
   
 expm1(...)  
 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(...)  
 fabs(x)  
   
 Return the absolute value of the float x.  
   
 factorial(...)  
 factorial(x) -> Integral  
   
 Find x!. Raise a ValueError if x is negative or non-integral.  
   
 floor(...)  
 floor(x)  
   
 Return the floor of x as an Integral.  
 This is the largest integer <= x.  
   
 fmod(...)  
 fmod(x, y)  
   
 Return fmod(x, y), according to platform C. x % y may differ.  
   
 frexp(...)  
 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(...)  
 fsum(iterable)  
   
 Return an accurate floating point sum of values in the iterable.  
 Assumes IEEE-754 floating point arithmetic.  
   
 gamma(...)  
 gamma(x)  
   
 Gamma function at x.  
   
 gcd(...)  
 gcd(x, y) -> int  
 greatest common divisor of x and y  
   
 hypot(...)  
 hypot(x, y)  
   
 Return the Euclidean distance, sqrt(x\*x + y\*y).  
   
 isclose(...)  
 isclose(a, b, \*, rel\_tol=1e-09, abs\_tol=0.0) -> bool  
   
 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(...)  
 isfinite(x) -> bool  
   
 Return True if x is neither an infinity nor a NaN, and False otherwise.  
   
 isinf(...)  
 isinf(x) -> bool  
   
 Return True if x is a positive or negative infinity, and False otherwise.  
   
 isnan(...)  
 isnan(x) -> bool  
   
 Return True if x is a NaN (not a number), and False otherwise.  
   
 ldexp(...)  
 ldexp(x, i)  
   
 Return x \* (2\*\*i).  
   
 lgamma(...)  
 lgamma(x)  
   
 Natural logarithm of absolute value of Gamma function at x.  
   
 log(...)  
 log(x[, base])  
   
 Return the logarithm of x to the given base.  
 If the base not specified, returns the natural logarithm (base e) of x.  
   
 log10(...)  
 log10(x)  
   
 Return the base 10 logarithm of x.  
   
 log1p(...)  
 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(...)  
 log2(x)  
   
 Return the base 2 logarithm of x.  
   
 modf(...)  
 modf(x)  
   
 Return the fractional and integer parts of x. Both results carry the sign  
 of x and are floats.  
   
 pow(...)  
 pow(x, y)  
   
 Return x\*\*y (x to the power of y).  
   
 radians(...)  
 radians(x)  
   
 Convert angle x from degrees to radians.  
   
 sin(...)  
 sin(x)  
   
 Return the sine of x (measured in radians).  
   
 sinh(...)  
 sinh(x)  
   
 Return the hyperbolic sine of x.  
   
 sqrt(...)  
 sqrt(x)  
   
 Return the square root of x.  
   
 tan(...)  
 tan(x)  
   
 Return the tangent of x (measured in radians).  
   
 tanh(...)  
 tanh(x)  
   
 Return the hyperbolic tangent of x.  
   
 trunc(...)  
 trunc(x:Real) -> Integral  
   
 Truncates 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  
 (built-in)

### Rounding Numbers

value = 4.35

math.floor(value)

4

math.ceil(value)

5

round(value)

4

### Mathematical Constants

math.pi

3.141592653589793

from math import pi

pi

3.141592653589793

math.e

2.718281828459045

math.tau

6.283185307179586

math.inf

inf

math.nan

nan

### Logarithmic Values

math.e

2.718281828459045

# Log Base e  
math.log(math.e)

1.0

# Will produce an error if value does not exist mathmatically  
math.log(0)

---------------------------------------------------------------------------  
  
ValueError Traceback (most recent call last)  
  
<ipython-input-12-7563e0a48092> in <module>()  
----> 1 math.log(0)  
  
  
ValueError: math domain error

math.log(10)

2.302585092994046

math.e \*\* 2.302585092994046

10.000000000000002

### Custom Base

# math.log(x,base)  
math.log(100,10)

2.0

10\*\*2

100

### Trigonometrics Functions

# Radians  
math.sin(10)

-0.5440211108893698

math.degrees(pi/2)

90.0

math.radians(180)

3.141592653589793

# Random Module

Random Module allows us to create random numbers. We can even set a seed to produce the same random set every time.

The explanation of how a computer attempts to generate random numbers is beyond the scope of this course since it involves higher level mathmatics. But if you are interested in this topic check out: \* https://en.wikipedia.org/wiki/Pseudorandom\_number\_generator \* https://en.wikipedia.org/wiki/Random\_seed

## Understanding a seed

Setting a seed allows us to start from a seeded psuedorandom number generator, which means the same random numbers will show up in a series. Note, you need the seed to be in the same cell if your using jupyter to guarantee the same results each time. Getting a same set of random numbers can be important in situations where you will be trying different variations of functions and want to compare their performance on random values, but want to do it fairly (so you need the same set of random numbers each time).

import random

random.randint(0,100)

62

random.randint(0,100)

10

# The value 101 is completely arbitrary, you can pass in any number you want  
random.seed(101)  
# You can run this cell as many times as you want, it will always return the same number  
random.randint(0,100)

74

random.randint(0,100)

24

# The value 101 is completely arbitrary, you can pass in any number you want  
random.seed(101)  
print(random.randint(0,100))  
print(random.randint(0,100))  
print(random.randint(0,100))  
print(random.randint(0,100))  
print(random.randint(0,100))

74  
24  
69  
45  
59

### Random Integers

random.randint(0,100)

6

### Random with Sequences

#### Grab a random item from a list

mylist = list(range(0,20))

mylist

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

random.choice(mylist)

12

mylist

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

### Sample with Replacement

Take a sample size, allowing picking elements more than once. Imagine a bag of numbered lottery balls, you reach in to grab a random lotto ball, then after marking down the number, **you place it back in the bag**, then continue picking another one.

random.choices(population=mylist,k=10)

[15, 14, 17, 8, 17, 2, 19, 17, 6, 1]

### Sample without Replacement

Once an item has been randomly picked, it can’t be picked again. Imagine a bag of numbered lottery balls, you reach in to grab a random lotto ball, then after marking down the number, you **leave it out of the bag**, then continue picking another one.

random.sample(population=mylist,k=10)

[17, 19, 11, 14, 1, 3, 4, 10, 5, 15]

### Shuffle a list

**Note: This effects the object in place!**

# Don't assign this to anything!  
random.shuffle(mylist)

mylist

[9, 11, 7, 12, 10, 16, 0, 2, 18, 13, 3, 5, 17, 1, 15, 6, 14, 19, 4, 8]

### Random Distributions

#### [Uniform Distribution](https://en.wikipedia.org/wiki/Uniform_distribution)

# Continuous, random picks a value between a and b, each value has equal change of being picked.  
random.uniform(a=0,b=100)

23.852305703497635

#### [Normal/Gaussian Distribution](https://en.wikipedia.org/wiki/Normal_distribution)

random.gauss(mu=0,sigma=1)

-0.21390381464435643

Final Note: If you find yourself using these libraries a lot, take a look at the NumPy library for Python, covers all these capabilities with extreme efficiency. We cover this library and a lot more in our data science and machine learning courses.