Computing Laboratory

Basics of Python - IV

Malay Bhattacharyya

Associate Professor

MIU, CAIML, TIH
Indian Statistical Institute, Kolkata

August, 2023

Importing modules and functions

Importing a module:

```
import <module_name>
```

OR

```
import <module_name> as <custom_name>
```

Importing modules and functions

Importing a module:

```
import <module_name>
```

OR

```
import <module_name> as <custom_name>
```

Using a function:

```
import <module_name>
<module_name>.<function_name>()
```

OR

```
import <module_name> as <custom_name>
<custom_name>.<function_name>()
```

OR

```
from <module_name> import <function_name>
<function_name>()
```

Using built-in methods

There are some functions in Python that does not require to import a module because they work on specific data structures.

- Built-in methods for strings (e.g., `capitalize()`, `strip()`, `zfill()`, etc.)
- Built-in methods for lists/arrays (e.g., `sort()`, `reverse()`, `clear()`, etc.)
- Built-in methods for dictionaries (e.g., `keys()`, `values()`, `update()`, etc.)
- Built-in methods for tuples (e.g., `count()`)

Note: The `sort()` method in Python implements the hybrid algorithm *Timsort*, derived from merge sort and insertion sort.

Mathematical functions for real numbers

Using the `math` module:

```
import math
math.<function_name>()
```

Mathematical functions for real numbers

Using the `math` module:

```
import math
math.<function_name>()
```

ceil(x)	– Ceiling of x
comb(n, r)	– Number of ways to choose r from n (${}^nC_r$)
copysign(x, y)	– Float with magnitude of x but sign of y
fabs(x)	– Absolute value of x
factorial(x)	– Factorial of x
floor(x)	– Floor of x

Table: Functions in `math` module

Mathematical functions for real numbers

Library code of the `factorial()` function:

```
static PyObject * math_factorial(PyObject *module, PyObject *arg){
    ...
    /* use lookup table if x is small */
    if (x < (long)Py_ARRAY_LENGTH(SmallFactorials))
        return PyLong_FromUnsignedLong(SmallFactorials[x]);
    /* else express in the form odd_part * 2**two_valuation,
    and compute as odd_part << two_valuation. */
    odd_part = factorial_odd_part(x);
    if (odd_part == NULL)
        return NULL;
    two_valuation = x - count_set_bits(x);
    result = _PyLong_Lshift(odd_part, two_valuation);
    Py_DECREF(odd_part);
    return result;
}
```

Source: `github.com/python/cpython/blob/main/Modules/mathmodule.c`

Mathematical functions for real numbers

fmod(x, y)	– x % y (preferable for integers)
frexp(x)	– Mantissa and exponent of x as a pair (m, e)
fsum(iterable)	– Accurate floating point sum of values in iterable
gcd(x, y)	– GCD of the integers x and y
isclose(x, y, *, rel_tol =1e-09, abs_tol=0.0)	– Whether x is close to y w.r.t max/min allowed tolerance
isfinite(x)	– Whether x is finite (or ∞/NaN)
isinf(x)	– Whether x is infinite
isnan(x)	– Whether x is NaN ("not a number")
isqrt(x)	– Integer square root of x
ldexp(x, i)	– $x * 2^i$

Table: Functions in `math` module

Mathematical functions for real numbers

```
import math
print(sum([.1, .1, .1, .1, .1, .1, .1, .1]))
print(math.fsum([.1, .1, .1, .1, .1, .1, .1, .1]))
ls = [1/7, 1/7, 1/7, 1/7, 1/7, 1/7, 1/7]
print(sum(ls))
print(math.fsum(ls))
```

Output:

```
0.7999999999999999
0.8
0.9999999999999998
1.0
```

Mathematical functions for real numbers

modf(x)	– Fractional and integer parts of x
perm(n, r=None)	– Number of ways to choose k from n without repetition (${}^nP_r$)
prod(iterable, *, start=1)	– Product of values in iterable
remainder(x, y)	– Remainder of x when divided by y
trunc(x)	– Value of x truncated to an integral
exp(x)	– e^x
expm1(x)	– $e^x - 1$ (provides a better precision)
log(x[, base])	– Natural logarithm of x (base e)
log1p(x)	– Natural logarithm of 1+x (base e)
log2(x)	– Base-2 logarithm of x
log10(x)	– Base-10 logarithm of x
pow(x, y)	– x^y

Table: Functions in `math` module

Mathematical functions for real numbers

```
import math
print(math.exp(0.8) - 1)
print(math.expm1(0.8))
print(math.log(math.exp(0.8)))
print(math.log(math.expm1(0.8)+1))
```

Output:

```
1.2255409284924679
1.2255409284924677
0.8000000000000002
0.7999999999999999
```

Mathematical functions for real numbers

sqrt(x)	– Square root of x
acos(x)	– Arc cosine of x (result in radians)
asin(x)	– Arc sine of x (result in radians)
atan(x)	– Arc tangent of x (result in radians)
atan2(x, y)	– Arc tangent of x/y (in radians)
cos(x)	– Cosine of x
dist(iterable1, iterable1)	– Euclidean distance between iterable 1 and iterable2
hypot(*coordinates)	– Euclidean norm sqrt(x*x + y*y) for the point (x, y), sqrt(sum(x**2 for x in coordinates))
sin(x)	– Sine of x
tan(x)	– Tangent of x

Table: Functions in `math` module

Mathematical functions for real numbers

degrees(x)	– Convert angle x from radians to degrees
radians(x)	– Convert angle x from degrees to radians.
acosh(x)	– Inverse hyperbolic cosine of x
asinh(x)	– Inverse hyperbolic sine of x
atanh(x)	– Inverse hyperbolic tangent of x
cosh(x)	– Hyperbolic cosine of x
sinh(x)	– Hyperbolic sine of x
tanh(x)	– Hyperbolic tangent of x
erf(x)	– Error function at x
erfc(x)	– Complementary error function at x
gamma(x)	– Gamma function at x
lgamma(x)	– Natural logarithm of absolute value of Gamma function at x

Table: Functions in `math` module

Mathematical functions for real numbers

pi	– $\pi = 3.14...$, to available precision
e	– $e = 2.71...$, to available precision
tau	– $\tau = 6.28...$, to available precision
inf	– Floating-point positive infinity
nan	– Floating-point NaN

Table: Functions in `math` module

Mathematical functions for complex numbers

Using the `cmath` module:

```
import cmath
cmath.<function_name>()
```

OR

```
from cmath import <function_name>
<function_name>()
```

Mathematical functions for complex numbers

Using the `cmath` module:

```
import cmath
cmath.<function_name>()
```

OR

```
from cmath import <function_name>
<function_name>()
```

phase(x)	– Phase of x (in float)
polar(x)	– Representation of x in polar coordinates as (r, phi)
rect(r, phi)	– Complex number x with polar coordinates r and phi
exp(x)	– e^x
log(x[, base])	– Natural logarithm of x (base e)
log10(x)	– Base-10 logarithm of x

Table: Functions in `cmath` module

Mathematical functions for complex numbers

sqrt(x)	– Square root of x
acos(x)	– Arc cosine of x
asin(x)	– Arc sine of x
atan(x)	– Arc tangent of x
cos(x)	– Cosine of x
sin(x)	– Sine of x
tan(x)	– Tangent of x
acosh(x)	– Inverse hyperbolic cosine of x
asinh(x)	– Inverse hyperbolic sine of x
atanh(x)	– Inverse hyperbolic tangent of x
cosh(x)	– Hyperbolic cosine of x
sinh(x)	– Hyperbolic sine of x
tanh(x)	– Hyperbolic tangent of x

Table: Functions in `cmath` module

Mathematical functions for complex numbers

isclose(x, y, *, rel_tol =1e-09, abs_tol=0.0)	– Whether x is close to y w.r.t max/min allowed tolerance
isfinite(x)	– Whether x is finite (or ∞/NaN)
isinf(x)	– Whether x is infinite
isnan(x)	– Whether x is NaN
pi	– $\pi = 3.14...$, to available precision
e	– $e = 2.71...$, to available precision
tau	– $\tau = 6.28...$, to available precision
inf	– Floating-point positive infinity
infj	– Complex number with zero real and positive infinity imaginary parts
nan	– Floating-point NaN
nanj	– Complex number with zero real part and NaN imaginary part

Table: Functions in `cmath` module

Statistical functions

Using the `statistics` module:

```
import statistics
statistics.<function_name>()
```

Statistical functions

Using the `statistics` module:

```
import statistics
statistics.<function_name>()
```

mean(X)	– Arithmetic mean of the data in X
fmean(X)	– Arithmetic mean of the data (converted to float) in X
geometric_mean(X)	– Geometric mean of the data (converted to float) in X
harmonic_mean(X)	– Harmonic mean of the data in X

Table: Functions in `statistics` module

Statistical functions

median(X)	– Median of the data in X
median_low(X)	– Low median of the data in X
median_high(X)	– High median of the data in X
median_grouped(X, interval)	– Median of the grouped data in X
mode(X)	– Most frequent data item in X
multimode(X)	– Most frequent data items in the order they appear in X

Table: Functions in `statistics` module

Statistical functions

pstdev(X, mu=None)	– Population standard deviation of the data in X
pvariance(X, mu=None)	– Population variance of the data in X
stdev(X, xbar=None)	– Sample standard deviation of the data in X
variance(X, xbar=None)	– Sample variance of the data in X
quantiles(X, *, n, method)	– Divide data in X into n continuous intervals with equal probability

Table: Functions in `statistics` module

Shuffling of data

```
import random
ls = [10, 20, 30, 40, 50]
random.shuffle(ls)
print(ls)
```

Output: [40, 20, 10, 50, 30]

Randomization of data

```
import random
random.randint(1, 100) # The interval is [1, 100]
```

Output: 15

```
import random
random.random() # The interval is [0.0, 1.0)
```

Output: 0.4289859005273219

Random sampling with numpy

```
import numpy as np
np.random.randint(1, 100) # From uniform distribution
```

Output: 97

The interval is [1, 100)

Random sampling with numpy

```
import numpy as np
np.random.randint(1, 100) # From uniform distribution
```

Output: 97 The interval is [1, 100)

```
import numpy as np
np.random.random(5) # From uniform distribution
```

Output: array([0.37076725, 0.10547203, 0.21298417, 0.96296838, 0.15390583]) The interval is [0.0, 1.0)

Random sampling with numpy

```
import numpy as np
np.random.randint(1, 100) # From uniform distribution
```

Output: 97 The interval is [1, 100)

```
import numpy as np
np.random.random(5) # From uniform distribution
```

Output: array([0.37076725, 0.10547203, 0.21298417, 0.96296838, 0.15390583]) The interval is [0.0, 1.0)

Note: Unlike the random() function in random module, the one in the random submodule of numpy module optionally takes size of the array.

Random sampling with numpy

```
import numpy as np
np.random.randn(5) # From normal distribution
```

Output: array([1.62029576, 0.29112406, 1.21198839, 0.26851418, -0.46712281])

```
import numpy as np
np.random.randn(3, 3) # From normal distribution
```

Output: array([[1.13217437, 0.56093212, -2.36792091], [1.42652606, 0.68953983, 0.521175], [1.36766496, 0.9488446 , -1.10042531]])

Effect of the seed value on random sampling

Code 1:

```
import numpy as np
print(np.random.randint(1, 100))
print(np.random.randint(1, 100))
print(np.random.randint(1, 100))
```

Effect of the seed value on random sampling

Code 1:

```
import numpy as np
print(np.random.randint(1, 100))
print(np.random.randint(1, 100))
print(np.random.randint(1, 100))
```

Code 2:

```
import numpy as np
np.random.seed(20)
print(np.random.randint(1, 100))
print(np.random.randint(1, 100))
print(np.random.randint(1, 100))
```

Effect of the seed value on random sampling

Code 1:

```
import numpy as np
print(np.random.randint(1, 100))
print(np.random.randint(1, 100))
print(np.random.randint(1, 100))
```

Code 2:

```
import numpy as np
np.random.seed(20)
print(np.random.randint(1, 100))
print(np.random.randint(1, 100))
print(np.random.randint(1, 100))
```

Note: Unlike Code 1, Code 2 will generate the same set of random values whenever executed.

Drawing samples from different distributions using `numpy`

beta(a, b[, size])	– Samples from Beta distribution
binomial(n, p[, size])	– Samples from binomial distribution
chisquare(df[, size])	– Samples from chi-square distribution
dirichlet(alpha[, size])	– Samples from Dirichlet distribution
exponential([scale, size])	– Samples from exponential distribution
f(dfnum, dfden[, size])	– Samples from F distribution
gamma(shape[, scale, size])	– Samples from Gamma distribution
geometric(p[, size])	– Samples from geometric distribution
gumbel([loc, scale, size])	– Samples from Gumbel distribution

Table: Functions in `random` submodule of `numpy`

Drawing samples from different distributions using `numpy`

hypergeometric(ngood, nbad, nsample[, size])	– Samples from hypergeometric distribution
laplace([loc, scale, size])	– Samples from Laplace distribution
logistic([loc, scale, size])	– Samples from logistic distribution
lognormal([mean, sigma, size])	– Samples from lognormal distribution
logseries(p[, size])	– Samples from logarithmic series distribution

Table: Functions in `random` submodule of `numpy`

Drawing samples from different distributions using `numpy`

multinomial(n, pvals[, size])	– Samples from multinomial distribution
multivariate_normal(mean, cov[, size, ...)	– Samples from multivariate normal distribution
negative_binomial(n, p[, size])	– Samples from negative binomial distribution
noncentral_chisquare(df, nonc[, size])	– Samples from noncentral chi-square distribution
noncentral_f(dfnum, dfden, nonc[, size])	– Samples from noncentral F distribution
normal([loc, scale, size])	– Samples from normal distribution

Table: Functions in `random` submodule of `numpy`

Drawing samples from different distributions using `numpy`

pareto(a[, size])	– Samples from pareto distribution
poisson([lam, size])	– Samples from Poisson distribution
power(a[, size])	– Samples from power distribution
rayleigh([scale, size])	– Samples from Rayleigh distribution
standard_cauchy([size])	– Samples from standard Cauchy distribution
standard_exponential([size])	– Samples from standard exponetial distribution
standard_gamma(shape[, size])	– Samples from standard Gamma distribution
standard_normal([size])	– Samples from standard normal distribution

Table: Functions in `random` submodule of `numpy`

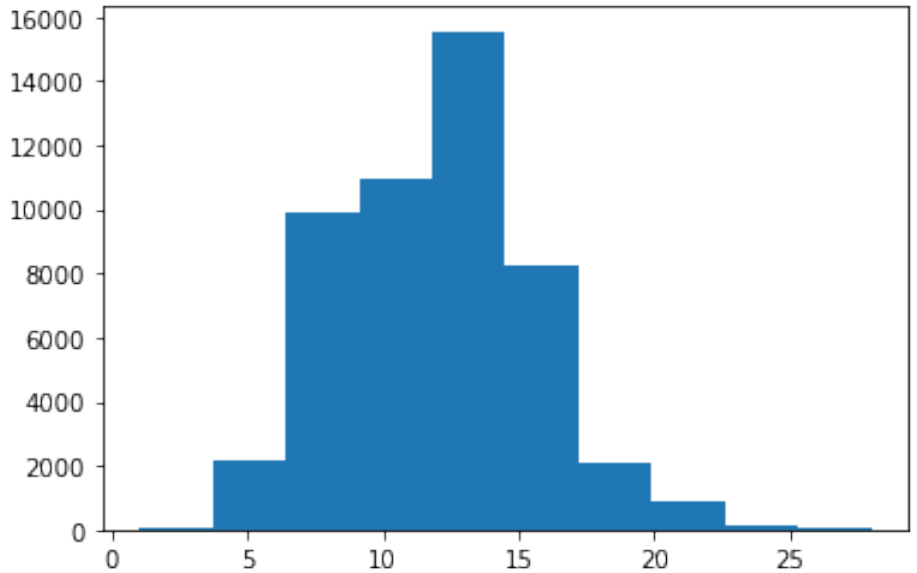
Drawing samples from different distributions using `numpy`

standard_t(df[, size])	– Samples from standard Student's t distribution
triangular(left, mode, right[, size])	– Samples from triangular distribution
uniform([low, high, size])	– Samples from uniform distribution
vonmises(mu, kappa[, size])	– Samples from von Mises distribution
wald(mean, scale[, size])	– Samples from Wald distribution
weibull(a[, size])	– Samples from Weibull distribution
zipf(a[, size])	– Samples from Zipf distribution

Table: Functions in `random` submodule of `numpy`

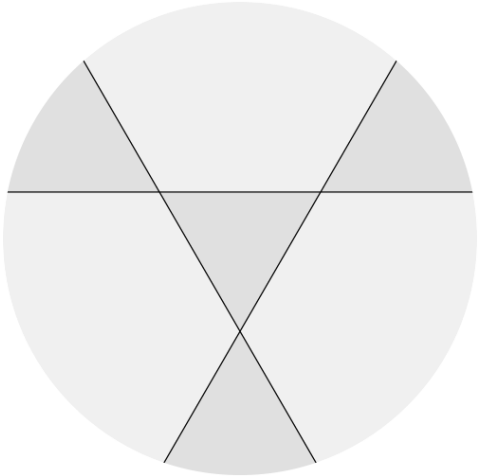
Visualizing a distribution

```
import numpy as np
sample = np.random.poisson(10, 50000)
import matplotlib.pyplot as plt
plt.hist(sample, 10)
plt.show()
```



Problems – Day 5

1. Suppose a caterer needs to cut a circular pankake into maximum number of pieces (not necessarily of the same shape or size) with a given number of straight cuts. For example, 3 straight cuts across a pancake will produce 6 pieces if the cuts are made intersecting at a common point, but 7 if they do not. Write a program to print the maximum number of pieces that can be obtained with a given number of straight cuts as input.



Problems – Day 5

2. Write a program that will print '0' with a probability of 0.5 and '1' in rest of the cases. Now write a function `RUN` to count the number of runs in the binary stream generated with your program. The number of runs will simply explore and count dichotomous (reflecting contrast between two things) sequence of values.
   Note: The following sequence of 0's and 1's has a run count of nine.
   0 0 1 1 0 1 0 0 0 0 1 0 0 1 1 1 1 1 0 0
3. Write a program to find out the summation of the following series given n as the user input.

$$\frac{1}{1} + \frac{11}{12} + \frac{111}{123} + \cdots + \frac{111 \ldots n \text{ times}}{123 \ldots n}$$