Introduction to Computing

Basics of Python Programming – III

Malay Bhattacharyya

Associate Professor

MIU, CAIML, TIH
Indian Statistical Institute, Kolkata

September, 2024

Functions

```
def <function-name>(<argument 1>, ..., <argument n>):
    Statement 1
    Statement 2
    Statement 3
    return <expression>
```

Functions

```
def <function-name>(<argument 1>, ..., <argument n>):
    Statement 1
    Statement 2
    Statement 3
    return <expression>
```

Note: You may either return a values obtained from an `<expression>` or return nothing based on your requirement.

Functions – Understanding the internals

```
def function(a, b, c):
    print('Inside 1:', hex(id(a)), hex(id(b)), hex(id(c)))
    b += 10
    print('Inside 2:', hex(id(a)), hex(id(b)), hex(id(c)))
x, y, z = 10, 10, 20
print('Outside:', hex(id(x)), hex(id(y)), hex(id(z)))
function(x, y, z)
```

Functions – Understanding the internals

```
def function(a, b, c):
    print('Inside 1:', hex(id(a)), hex(id(b)), hex(id(c)))
    b += 10
    print('Inside 2:', hex(id(a)), hex(id(b)), hex(id(c)))
x, y, z = 10, 10, 20
print('Outside:', hex(id(x)), hex(id(y)), hex(id(z)))
function(x, y, z)
```

Output:

```
Outside: 0x7fc4436a0210 0x7fc4436a0210 0x7fc4436a0350
Inside 1: 0x7fc4436a0210 0x7fc4436a0210 0x7fc4436a0350
Inside 2: 0x7fc4436a0210 0x7fc4436a0350 0x7fc4436a0350
```

Comparing the lengths of two positive integers

```
def compare(m, n):
    while m and n:
        m //= 10
        n //= 10
    return m-n
x, y = input('Enter two numbers: ').split()
f = compare(int(x), int(y))
if f > 0:
    print(x, 'has larger number of digits')
elif f < 0:
    print(y, 'has larger number of digits')
else:
    print('Both have same number of digits')
```

Finding prefixes of a string

```
def prefix(str):
    start, end = 0, 0
    while start < len(str):
        if str[end] == str[end-start]:
            print(str[start:end + 1], end= " ")
            end += 1
            if end == len(str):
                start += 1
                end = start
        else:
            start += 1
            end = start # Index of substring
```

Input: prefix('Python')

Lambda functions

Python provides an anonymous function `lambda` that can take any number of arguments, but can only have one expression.

```
x = lambda a: a + 10
print(x(5))
```

Here, the output will be 15. **Note:** Lambda functions execute

comparatively faster (in general) because they are inline functions.

Lambda functions

```
a, b = 3, 5
maximum = lambda a, b: a if a > b else b
print(f'{maximum(a,b)} is the maximum among', a, 'and', b)
```

Lambda functions

```
a, b = 3, 5
maximum = lambda a, b: a if a > b else b
print(f'{maximum(a,b)} is the maximum among', a, 'and', b)
```

Output:

```
5 is the maximum among 3 and 5
```

Lambda functions

```
l = lambda n: n
q = lambda n: n ** 2
c = lambda n: n ** 3
x = 2
y = c(x) + 3*q(x) + 2*l(x) + 1 # y = x^3 + 3x^2 + 2x + 1
print(y)
```

Lambda functions

```
l = lambda n: n
q = lambda n: n ** 2
c = lambda n: n ** 3
x = 2
y = c(x) + 3*q(x) + 2*l(x) + 1 # y = x^3 + 3x^2 + 2x + 1
print(y)
```

Output:

```
25
```

Late binding closures

Python's closures are late binding. Hence, the values of variables used in closures are looked up when the inner function is called.

```
def increase():
    return [lambda x : i + x for i in range(5)]
for add in increase():
    print(add(10))
```

Here, whenever any of the returned functions are called, the value of i is looked up in the surrounding scope at call time. By then, the loop has completed and i is left with its final value of 4. So, it will print {14, 14, 14, 14, 14} instead of {10, 11, 12, 13, 14}.

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

Homework

- Write a program that takes a string as input and pass it on to a function to print all its suffixes.
- Write a lambda function to return the maximum of three integers.
- Write a lambda function to compute the factorial of a number given as user input.
- The Maclaurin series of the exponential function e^x is given by

$$\sum_{n=0}^{\infty} \frac{x^n}{n!} = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \cdots .$$

  Write a program to find out the value of e^x up to a number of terms given as user input.