SMM692
Introduction to Programming in Python

Contents

Chapter 1

Organization of the Module

...

Chapter 2

Getting Started with Python

...

Chapter 3

Managing Python Environments

...

Chapter 4

Collaborative and Versioning Tools for Python

...

Chapter 5

Python Objects

What is a Python object?

In essence, Python objects are pieces of data. Mark Lutz, the author of the popular book Learning Python[1], points out

> *in Python we do things with stuff. "Things" take the form of operations like addition and concatenation, and "stuff" refers to the objects on which we perform those operations.*

Built-in and ad-hoc objects

In Python, there are two families of objects: built-in objects provided by the Python language itself and ad-hoc objects — called classes — we can create to accomplish specific goals.

Why do built-in Python objects matter?

Typically, we do not need to create ad-hoc objects. Python provides us with diverse built-in objects that make our job easier:

- built-in objects make coding efficient and easy. For example, using the string object, we can represent and manipulate a piece of text — e.g., a newspaper article — without loading any module
- built-in objects are flexible. For example, we can deploy built-in objects to create a class
- built-in objects have been created and refined over time by a large community of expert developers. Hence, they are often more efficient than ad-hoc objects (unless the creator of the ad-hoc object really knows her business!)

The core built-in Python objects

Table I illustrates the types of built-in Python objects. For example, Numbers and strings objects are used to represent numeric and textual data respectively. Lists and dictionaries are — likely as not — the two most popular data structures in Python. Lists are ordered collections of other objects such (any type!!). Dictionaries are pairs of keys (e.g., a product identifier) and objects (e.g., the price of the product). No worries: we will go through each built-in type in the following sections of this document. Caveat: in the interest of logical coherence, the various built-in types will not be presented in the order adopted Table I.

TABLE I
Buil-In Objects in Python

Object type	Example literals/creation
Numbers	1234, 3.1415, 3+4j, 0b111, Decimal(), Fraction()
Strings	'spam', "Bob's", b'a\x01c', u'sp\xc4m'
Lists	[1, [2, 'three'], 4.5], list(range(10))
Dictionaries	{'food': 'spam', 'taste': 'yum'}, dict(hours=10)
Tuples	(1, 'spam', 4, 'U'), tuple('spam'), namedtuple
Files	open('eggs.txt'), open(r'C:\ham.bin', 'wb')
Sets	set('abc'), {'a', 'b', 'c'}
Other core types	Booleans, types, None
Program unit types	Functions, modules, classes
Implementation types	Compiled code, stack tracebacks

5.1 Number Type Fundamentals

Types of 'number' objects

Example 1, "Doing stuff with numbers," highlights a sample of three 'number' instances in Python: integers, floating-point, and complex numbers. Integers are whole numbers such as 0, 4, or -12. Floating-point numbers are the representation of real numbers such as 0.5, 3.1415, or -1.6e-19. However, floating points in Python do not have — in general — the same value as the real number they represent.[2] It is worth noticing that any single number with a period '.' is considered a floating point in Python. Also, Example 1 shows that the multiplication of an integer by a floating point yields a floating point. Complex numbers such as $2 + 3j$ consist of a real and the imaginary part j, each of which is a floating point number. Besides integers, floating points, and complex numbers, Python includes fixed-precision, rational numbers, Booleans, and sets instances.

Example 1 — Doing 'stuff' with numbers

```
# let us assign the string "Python 3.X" to the variable S
In [1]: S = "Python 3.X"

# check the length of S
In [2]: len(S)
Out[2]: 6

# access the first string object in the sequence behind S
In [3]: S[0]
Out[3]: "P"

# access the last string object in the sequence behind S
In [4]: S[-1]
Out[4]: "X"

# or, equivalently
In [5]: S[len(S)-1]
Out[5]: "X"

# access the i-th, e.g., 3rd, string object in the sequence behind S
In [6]: S[3]
Out[6]: "X"

# access string objects between the i-th and j-th positions in the sequence
# behind S
In [7]: S[2:5]
Out[7]: "tho"

# access string objects following the i-th position in the sequence behind S
In [8]: S[-3:]
Out[8]: "3.X"
```

Basic arithmetic operations in Python

Numbers in Python support the usual mathematical operations shown in Table II. To use these operations, it is sufficient to launch a Python or IPython session without any modules loaded (see Example 1).

TABLE II
Arithmetic Operations in Python

Symbol	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Floating point division
//	Integer division
%	Modulus (remainder)
**	Exponentiation

Advanced mathematical operations

Besides the mathematical operations included in Table II, there are many modules shipped with Python that carry out advanced/specific numerical analysis. For example, the `math` module provides access to the mathematical functions defined by the C standard.[3] Table III reports a sample of these functions. To use them `math`, we have to import the module as shown in Example 2. Another popular module shipped with Python is `random`, implementing pseudo-random number generators for various distributions (see the lower section of Example 2).

Example 2 — Advanced mathematical operations with the modules shipped with Python

```
# import the math module
In [1]: import math

# base-y log of x
In [2]: math.log(12, 8)
Out[2]: 1.1949875002403856

# base-10 log of x
In [3]: math.log10(12)
Out[3]: 1.0791812460476249

# import the random module
In [4]: import random

# a draw from a normal distribution with mean = 0 and standard deviation = 1
In [5]: random.normalvariate(0, 1)
Out[5]: -0.136017752991189
```

TABLE III
A Sample of Functions Provided by the `math` Module

Function name	Expression
math.sqrt(x)	$\sqrt{x}$
math.exp(x)	e^x
math.log(x)	lnx
math.log(x, b)	$log_b(x)$
math.log10(x)	$log_{10}(x)$
math.sin(x)	$sin(x)$
math.cos(x)	$cos(x)$
math.tan(x)	$tan(x)$
math.asin(x)	$arcsin(x)$
math.acos(x)	$arccos(x)$
math.atan(x)	$arctan(x)$
math.sinh(x)	$sinh(x)$
math.cosh(x)	$cosh(x)$
math.tanh(x)	$tanh(x)$
math.asinh(x)	$arsinh(x)$
math.acosh(x)	$arcosh(x)$
math.atanh(x)	$artanh(x)$
math.hypot(x, y)	The Euclidean norm, $\sqrt{x^2 + y^2}$
math.factorial(x)	$x!$
math.erf(x)	The error function at x
math.gamma(x)	The gamma function at x, $\omega(x)$
math.degrees(x)	Converts x from radians to degrees
math.radians(x)	Converts x from degrees to radians

5.2 String Type Fundamentals

What is a string?

A Python string is a positionally ordered collection of other objects. Sequences maintain a left-to-right order among the items they contain: their items are stored and fetched by their relative positions. Strictly speaking, strings are sequences of one-character strings; other, more general sequence types include lists and tuples, covered later.

Is abc a Python string?

Nope. Pythons strings are enclosed in single quotes ('...') *or* double quotes ("...") with the same result. Example

Example 3 — Python strings as sequences

```
# let us assign the string "Python 3.X" to the variable S
In [1]: S = "Python 3.X"

# check the length of S
In [2]: len(S)
Out[2]: 6

# access the first string object in the sequence behind S
In [3]: S[0]
Out[3]: "P"

# access the last string object in the sequence behind S
In [4]: S[-1]
Out[4]: "X"

# or, equivalently
In [5]: S[len(S)-1]
Out[5]: "X"

# access the i-th, e.g., 3rd, string object in the sequence behind S
In [6]: S[3]
Out[6]: "X"

# access string objects between the i-th and j-th positions in the sequence
# behind S
In [7]: S[2:5]
Out[7]: "tho"

# access string objects following the i-th position in the sequence behind S
In [8]: S[-3:]
Out[8]: "3.X"
```

How do we use strings?

> Strings are used to record both textual information (your name, for instance) as well as arbitrary collections of bytes (such as an image file's contents).

5.3 List and Dictionaries

...

5.4 Tuples, Files, and Everything Else

...

5.5 Python Statements

...

5.6 If Test

...

5.7 While and For Loops

...

5.8 Iterations and Comprehensions

...

Notes

[1]Lutz, Mark. *Learning Python: Powerful object-oriented programming.* O'Reilly Media, Inc., 2013.

[2]Floating numbers are stored in binaries with an assigned level of precision that is typically equivalent to 15 or 16 decimals.

[3]As per the documentation of the Python programming language, `math` cannot be used with complex numbers.

Chapter 6

Technical and Scientific Computation with NumPy and SciPy

...

Chapter 7

Data Management with Pandas and Dask

...

Chapter 8

Coda

...