Python Fundamentals
A Gentle Introduction to Built-In Objects

Contents

1 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. 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

2 Number Type Fundamentals

Doing ‘stuff’ with numbers

```
# integer addition
In [1]: 1 + 1
Out[1]: 2

# floating-point multiplication
In [2]: 10 * 0.5
Out[2]: 5.0

# 3 to the power 1000
In [3]: 3 ** 1000
Out[3]: 515377520732011331036461129765621272702107522001
```

Types of ‘number’ objects

The above-displayed Python snippet “Doing stuff with numbers” highlights two

3 String Type Fundamentals

...

4 List and Dictionaries

...

5 Tuples, Files, and Everything Else

...

6 Python Statements

...

7 If Test

...

8 While and For Loops

...

9 Iterations and Comprehensions

...

Notes

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