Intro to Python — SMM692

Python Objects

Simone Santoni

Bayes Business School

MSc Pre-Course Series

Outline

Scope

In Chapter 3, the attention revolves around the following topics:

- The concept of Python object
- The types of Python objects
- The characteristics of each Python object

Why Shall I Learn About Python Objects?

- 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 knows her business!)

Learning Goals

At the end of the chapter, you will be able to evaluate the various types of Python objects regarding:

- Key features
- Use cases/roles
- Available methods

What is a Python Object?

In essence, Python objects are pieces of data. Mark Lutz, the author of the popular book Learning Python, 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"*

What Are the Main Families of Python Objects?

In Python, there are two families of objects:

- Built-in objects provided by the Python language itself
- Ad-hoc objects — called classes — we can create to accomplish specific goals

What Are the Main Types of Built-In Python Objects?

- Strings
- Numbers
- Data containers
  - Lists
  - Dictionaries
  - Tuples
  - Sets
- Files
- Python statements, syntax, and control flow
- Iterators

Outline

Integers and Floating Points

The most common number types are integers and floating-point numbers:

- Integers are whole numbers such as 0, 4, or -12
- Floating-point numbers represent 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
  - It is worth noticing that any single number with a period '.' is considered a floating point in Python

Snippet 4.1 — doing 'stuff' with numbers

```
# integer addition
>>> 1 + 1
2

# floating-point multiplication
>>> 10 * 0.5
5.0

# 3 to the power 100
>>> 3 ** 100
515377520732011331036461129765621272702107522001
```

Number Type Objects in Python

Number Type Objects in Python

Literal	Interpretation
1234, -24, 0, 99999999999999	Integers (unlimited size)
1.23, 1., 3.14e-10, 4E210, 4.0e+210	Floating-point numbers
0o177, 0x9ff, 0b101010	Octal, hex, and binary literals in 3.X
0177, 0o177, 0x9ff, 0b101010	Octal, octal, hex, and binary literals in 2.X
3+4j, 3.0+4.0j, 3J	Complex number literals
set('spam'), {1, 2, 3, 4}	Sets: 2.X and 3.X construction forms
Decimal('1.0'), Fraction(1, 3)	Decimal and fraction extension types
bool(X), True, False	Boolean type and constants

String Type Fundamentals

What: A Python string is a positionally ordered collection of other objects. Strictly speaking, strings are *immutable sequences* of one-character strings; other, more general sequence types include lists and tuples, covered later.

How and why: Strings are used to record words, contents of text files loaded into memory, Internet addresses, Python source code, and so on.

Snippet 4.6 — Python strings as sequences

```
# the string
>>> S = "Python 3.X"
# check the length of S
>>> len(S)
6
# access the first unitary string in the sequence behind S
>>> S[0]
"P"
# access the last unitary string in the sequence behind S
>>> S[len(S)-1]
"X"
# or, equivalently
>>> S[-1]
"X"
# access the unitary strings between the i-th and j-th positions
# sequence behind S
>>> S[2:5]
"tho"
```

Sting Literals and Operators

Literal/operation	Interpretation
`S = ""`	Empty string
`S = ‘’`	Single quotes, same as double quotes
`S = "spam’s"`	Single quote as a string
`S = ‘spam\’s’`	Escape symbol
`length(S)`	Length
`S[i]`	Index
`S[i:j]`	Slice
`S1 + S2`	Concatenate
`S * 3`	Repeat S *n* times (e.g., three times)
`"text".join(strlist)`	Join multiple strings on a character (e.g., “text”)
`"{}".format()`	String formatting expression

String Literals and Operators (cont'd)

Literal/operation	Interpretation
`S.strip()`	Remove white spaces
`S.replace("pa", "xx")`	Replacement
`S.split(",")`	Split on a character (e.g., ",")
`S.lower()`	Case conversion — to lower case
`S.upper()`	Case conversion — to upper case
`S.find("text")`	Search substring (e.g., "text")
`S.isdigit()`	Test if the string is a digit
`S.endswith("spam")`	End test
`S.startswith("spam")`	Start test
`S = """...multiline..."""`	Triple-quoted block strings

Outline

List Type Fundamentals

What: A Python `list` is an *ordered*, *mutable* array of objects. A list is constructed by specifying the objects, separated by commas, between square brackets, []

How and why: Lists are just places to collect other objects — being numbers, strings, or even other lists — so you can treat them as groups.

Snippet 4.11 — list indexing and slicing

```
# the list
>>> L = [4, ["abc", 8.98]]
# get the first item of L
>>> L[0]
4
# get the second element of L
>>> L[1]
["abc", 8.98]
# get the first item of L's second item
>>> L[1][0]
"abc"
```

Popular List Methods

Method	Synopsis
`L.append(X)`	Append an item to an existing list
`L.insert(i, X)`	Append an item to an existing list in position *i*
`L.extend([X0, X1, X2])`	Extend an existing list with the items from another list
`L.index(X)`	Get the index of the first instance of the argument in an existing list
`L.count(X)`	Get the cardinality of an item in an existing list
`L.sort()`	Sort the items in an existing list
`L.reverse()`	Reverse the order of the items in an existing list
`L.copy()`	Get a copy of an existing list
`L.pop(i)`	Remove the item at the given position in the list, and return it
`L.remove(X)`	Remove the first instance of an item in an existing list
`L.clear()`	Remove all items in an existing list

Dictionary Type Fundamentals

Tuple Type Fundamentals

Set Type Fundamentals

Outline

...

Outline

...

Outline

...

At the End of the Chapter, You Will Be Able to...