Introducing Python Data Types

Gaurav Kr. suman

MIT4

Introducing Python Data Types

They are known as core data types because they are effectively built into the Python language—this implies that there is a specific syntax for generating most of them.

Python’s Core Data Types include:

- Numbers
- Strings
- Lists
- Dictionaries
- Tuples
- Files
- Other types include: Sets, types, None, Booleans

Numbers:

Among Python Data Types, numbers are the most important. The usual object sets may include integers (numbers without a fractional part), floating-point numbers (roughly, numbers with a decimal point in them), and other types (like unlimited-precision “long” integers, complex numbers with imaginary parts, fixed-precision decimals, and sets).

Python’s basic number types support the normal mathematical operations. For instance, the plus sign (+) performs addition, a star (*) is used for multiplication, and two stars (**) are used for exponentiation:

```
>>> 123 + 222# Integer addition
345
>>> 1.5 * 4# Floating-point multiplication
6.0
>>> 2 ** 100# 2 to the power 100
1267650600228229401496703205376L
```

Other data types in Python include more exotic number objects—such as complex numbers, fixed-precision decimal numbers, and sets—and the third-party open-source extension domain has even more (e.g., matrixes and vectors).

Long Data Types in Python/Long Integer Data Types:

Long integers, also known as long or long integer data types in Python, are integers of unlimited size, written like integers and followed by an uppercase or lowercase L. These long data types in Python exist only in Python 2.x.

Integers that are too long to be stored in a 32-bit integer is automatically made into Longs. However, you can explicitly create one by adding an L after the number

Complex Numbers:

Complex data types, also known as complex numbers are of the form a + bJ, where a and b are floats and J (or j) represents the square root of -1 (which is an imaginary number). The real part of the number is a, and the imaginary part is b. Complex numbers are much in use in Python programming.

Strings:

Strings, one of the core data types in Python are used to record textual information as well as arbitrary collections of bytes. Strings are also an example of what we call a sequence in Python—that is, a positionally ordered collection of other objects. Sequences in Python, maintain a left-to-right order among the items they contain: their items are stored and fetched by their relative position. Precisely speaking, strings are sequences of one-character strings; other types of sequences include lists and tuples.

Sequence Operations:

Strings, quite like sequences, support operations that assume a positional ordering among items. For example, if we have a four-character string, we can verify its length with the built-in length function and fetch its components with indexing expressions:

```
>>> S = 'Spam'
>>> len(S)# Length
4
>>> S[0]# The first item in S, indexing by zero-based position
'S'
>>> S[1]# The second item from the left
'p'
```

Every string operation is, actually a sequence operation—that is, these operations will work on other sequences in Python as well, including lists and tuples. In addition to generic sequence operations, though, strings have operations all their own, available as methods (functions attached to the object, which are triggered with a call expression).

Python allows strings to be enclosed in single or double quote characters and it also has a multiline string literal form enclosed in triple quotes (single or double). When this form is used, all the lines are concatenated together, and end-of-line characters are added where line breaks appear.

Python also supports a "raw" string literal that turns off the backslash escape mechanism (they start with the letter r), as well as a Unicode string form that supports internationalization (they begin with the letter u and contain multibyte characters). Technically speaking, the Unicode string, is a different Python data type than normal string. However, this data type supports all the same string operations.

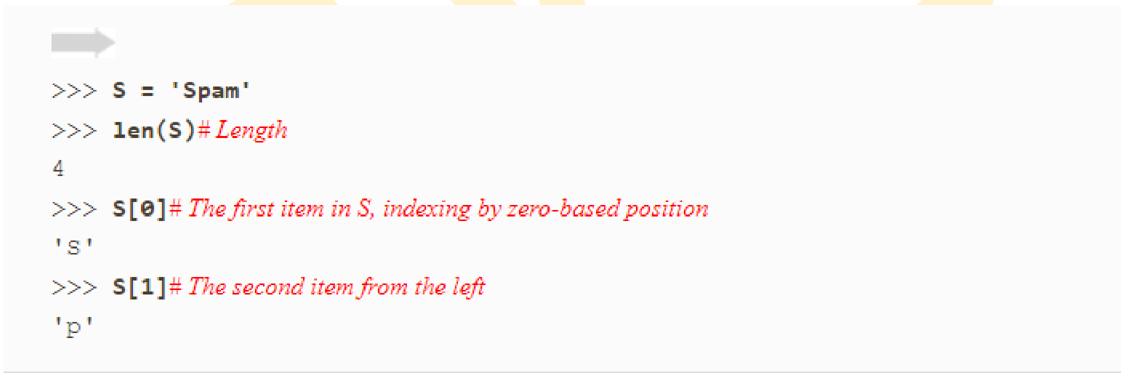
Lists:

The Python list object is the most generic Python Data Type. Lists are positionally ordered collections of arbitrarily typed objects and can be of any length. Lists, like Strings, are also mutable, can be modified in-place by assignment to offsets as well as a variety of list method calls.

A list is a typical example of the mutual data type in Python. It can contain mixed data types. A list and a tuple share many common features. Because a list is a modifiable data type, it has some additional operations. A whole chapter is dedicated to the Python list.

Sequence Operations:

Lists generally support all the sequence operations for strings; the only difference is that results usually list instead of strings.



```
>>> S = 'Spam'
>>> len(S)# Length
4
>>> S[0]# The first item in S, indexing by zero-based position
'S'
>>> S[1]# The second item from the left
'p'
```

Nesting:

Nesting is perhaps the best feature of Lists, one of Python's core data types. Lists support arbitrary nesting. They can be nested in any combination, and as deeply as required. For example, you can have a list containing a dictionary, which leads to another list, and so on. multidimensional arrays in Python are a classic application of this feature.

Dictionaries:

Python dictionaries are known as mappings. Mappings may also be described as collections of other objects, but they store objects by key instead of by relative position. mappings do not maintain any reliable left-to-right order; they simply map keys to associated values. Dictionaries, the only mapping type in Python's core objects set, are also mutable. These data types may be changed in-place and can grow and shrink on demand, just as lists do.

Mapping Operations:

When written as literals, dictionaries are coded in curly braces and consist of a series of "key: value" pairs. Dictionaries are useful anytime we need to associate a set of values with keys—to describe the properties of something, for instance. As an example, consider the following three-item dictionary (with keys "food," "quantity," and "color"):

Tuples:

A tuple is an immutable sequence Python data type. The tuple may contain mixed data types.

For example:

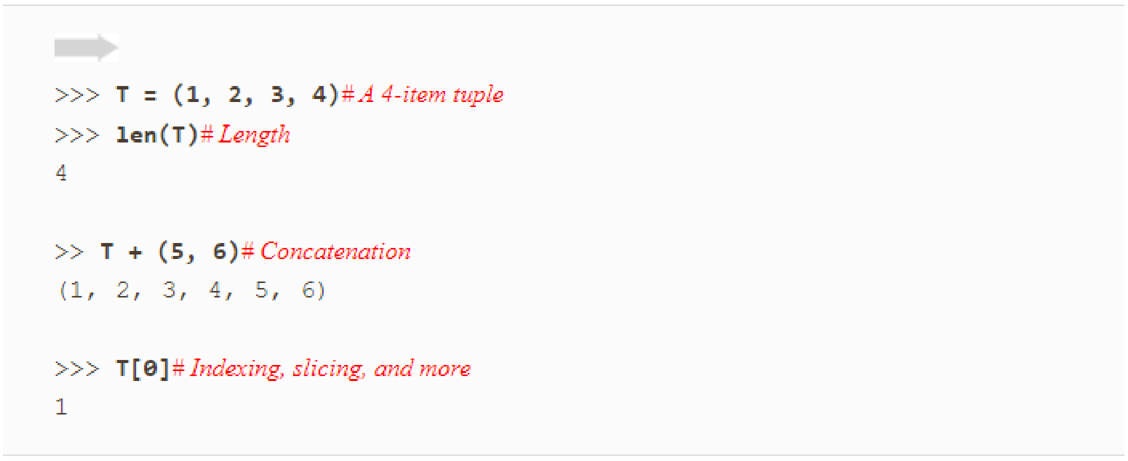
fruits = ("oranges", "apples", "bananas")

Tuples are created using round brackets. Here we have a tuple consisting of three fruit types.

fruits = “apples”, “oranges”, “bananas”

print(fruits) # prints (‘apples’, ‘oranges’, ‘bananas’)

The tuple object may be grossly explained as a list that cannot be changed. Tuples are, in fact, sequences, like lists, but they are also immutable, like strings. Syntactically, tuples, as core Python data types, are coded in parentheses instead of square brackets, and they support arbitrary types, nesting, and the usual sequence operations:



```
>>> T = (1, 2, 3, 4)#A 4-item tuple
>>> len(T)# Length
4

>> T + (5, 6)# Concatenation
(1, 2, 3, 4, 5, 6)

>>> T[0]# Indexing, slicing, and more
1
```

Files:

File objects may be described as Python code’s main interface to external files on your computer. They are one of the popular core data types in Python, but without any specific literal syntax for creating them. To create a file object, one needs to call the built-in open function, passing in an external filename as a string, and a processing mode string. For example, to create an output file, you would pass in its name and the ‘w’ processing mode string to write data:

```
>>> f = open('data.txt', 'w')# Make a new file in output mode

>>> f.write('Hello\n')# Write strings of bytes to it
>>> f.write('world\n')
>>> f.close(   )# Close to flush output buffers to disk
```

Other Core Types:

Other Python data types, may or may not qualify for membership, depending on how broad the category is defined to be. Sets, for example, are a recent addition to the language. Sets may be defined as containers of other objects created by calling the built-in set function, and they support the usual mathematical set operations. Sets are available as one of the standard data type of Python, since Python 2.6.

Among other data types in Python, decimal numbers (fixed-precision floating-point numbers) and Booleans (with predefined True and False objects that are essentially just the integers 1 and 0 with custom display logic), are important. Booleans have long supported a special placeholder object called None.