

Programming and Database Fundamentals for Data Scientists

Python Lists

Varun Chandola

School of Engineering and Applied Sciences
State University of New York at Buffalo
Buffalo, NY, USA
chandola@buffalo.edu



Outline

Recap

Basic Data Types

Lists

What do we know so far?

- ▶ Python is awesome!!
- ▶ Forced indentations, simultaneous assignments, swapping values in one line
- ▶ Read - *The Zen of Python*
- ▶ What next?

Basic Data Types

- ▶ float - real numbers
- ▶ int - integers
- ▶ str - string, text
- ▶ bool - true, false

Basic Data Types

- ▶ float - real numbers
- ▶ int - integers
- ▶ str - string, text
- ▶ bool - true, false
- ▶ Each type refers to one value

Numeric Data Types

- ▶ How do I figure out the *type* of a variable?

Numeric Data Types

- ▶ How do I figure out the *type* of a variable?
 - ▶ Use in-built function `type`
- ▶ Why not use `float` instead of using `float` and `int`?

Numeric Data Types

- ▶ How do I figure out the *type* of a variable?
 - ▶ Use in-built function `type`
- ▶ Why not use `float` instead of using `float` and `int`?
 - ▶ Most mathematical algorithms are very efficient with integers
- ▶ Quick question
 - ▶ What is $7/3$?
 - ▶ What is $7//3$?
 - ▶ Also known as *integer division*.

Type Conversions

- ▶ Python supports valid type conversions
- ▶ `float(3)`
- ▶ `int('3')`
- ▶ A good way to validate inputs
- ▶ try-catch primer

Errors and Exceptions

- ▶ Typically two types of errors are encountered: *syntax errors* and *exceptions*
- ▶ Syntax errors are reported by the parser and are (relatively) easier to fix
- ▶ Exceptions occur during run time
- ▶ Unhandled exceptions crash the application
- ▶ How to explicitly handle exceptions?
 - ▶ `try` and `except` (with optional `else`)
- ▶ Cleaning up
 - ▶ Use `finally`

- ▶ In Data Science we work with many data points (recall the Chicago Crime example)
- ▶ Creating one variable for each data point is inefficient
- ▶ Introducing - *Python Lists*

Python Lists

- ▶ A collection of values
- ▶ Can contain any type
- ▶ Can contain several different types

Indexing Lists

- ▶ Zero based indexing
- ▶ Accessing from the end of the list

Manipulating Lists

- ▶ Allows adding, appending, deleting elements

List Storage

- ▶ A new list is created everytime we use []
- ▶ But what about?
 - ▶ `A = list(range(10))`
 - ▶ `B = A`

List Storage

- ▶ A new list is created everytime we use []
- ▶ But what about?
 - ▶ `A = list(range(10))`
 - ▶ `B = A`
- ▶ A and B *point* to the same collection

Some more operations on lists

- ▶ **list.extend**
- ▶ **list.insert**
- ▶ **list.remove**
- ▶ **list.pop**
- ▶ **list.clear**
- ▶ **list.index**
- ▶ **list.count**
- ▶ **list.reverse**
- ▶ **list.sort**

Lists of Lists

- ▶ Nested lists

To Summarize

- ▶ The list object stores pointers to objects, not the actual objects themselves.
 - ▶ The size of a list in memory depends on the number of objects in the list, not the size of the objects.
- ▶ Getting or setting elements is $O(1)$
- ▶ Appending is more expensive but not $O(n^2)$, where n is the length of the list
- ▶ Removing items is similar in complexity as adding elements
- ▶ The time needed to reverse a list is proportional to the list size ($O(n)$).
- ▶ The time needed to sort a list varies; the worst case is $O(n \log n)$, but typical cases are often a lot better than that.

Other Data Structures in Python

- ▶ Strings
- ▶ Tuples
- ▶ Dictionaries
- ▶ Sets

- ▶ A string is a *sequence* of a **Unicode** characters
- ▶ Create a new string using single, double, or even triple quotes
 - ▶ No difference between a single and a double quote

Immutable Objects

- ▶ An object whose *state* cannot be modified after it is modified
- ▶ `str` - immutable, `list` - mutable

Why?

- ▶ Performance
- ▶ Ease of programming

Many methods defined for strings

- ▶ Accessing `str` elements is same as that for `list`
- ▶ Many in-built Python functions that operate on sequences (such as lists), will work on `str` too (`len`, `in`, `for`)
- ▶ Check out `str.is_`

Tuples - Another Immutable Object

- ▶ Another way to define sequences of arbitrary objects in Python
- ▶ Cannot change after creating
- ▶ Support standard Python sequence operations:
 1. Length (`len`)
 2. Concatentation (+)
 3. Repetition (*)
 4. Membership (`in`)
 5. Iteration (`for in`)
- ▶ And many other built-in Python core library functions

Creating Tuples

- ▶ Just create a comma separated “list”
- ▶ Parentheses optional
- ▶ One can convert tuple to a list, and vice-versa (use `tuple(l)` and `list(t)`)

Associative Arrays in Python - Dictionaries

- ▶ Different from traditional sequences
 - ▶ Indexed by *keys* instead of *range* of integers
 - ▶ Hashes
- ▶ Needed to create maps
- ▶ Great for a quick look-up
- ▶ Mutable

- ▶ An unordered collection with *no duplicate elements*
- ▶ Supports many set operations

References