A Short and Incomplete Introduction to Python

Part 5: File I/O and string processing

Riccardo Murri <riccardo.murri@uzh.ch>,
Sergio Maffioletti <sergio.maffioletti@uzh.ch>
S3IT: Services and Support for Science IT,
University of Zurich

January 25–26, 2018

Strings

In Python, Strings are sequences of characters:

- They can be indexed and sliced like `list`'s and other sequences:

```
In [1]: s = 'python'
In [2]: s[:2]
Out[2]: 'py'
```

- They are **homogeneous**: items of a string are always characters.
- They are **immutable**: you can only alter a string through functions that make a (modified) copy:

```
In [3]: s[0] = 'c'
...
TypeError: 'str' object does not support
item assignment
```

Operations on strings, I

`s.capitalize()`, `s.lower()`, `s.upper()`
Return a *copy* of the string capitalized / turned all lowercase / turned all uppercase.

`s.split(t)`
Split `s` at every occurrence of `t` and return a list of parts. If `t` is omitted, split on whitespace.

`s.startswith(t)`, `s.endswith(t)`
Return `True` if `t` is the initial/final substring of `s`.

Reference: http://docs.python.org/library/stdtypes.html#string-methods

Operations on strings, II

`s.replace(old, new)`
Return a *copy* of string `s` with all occurrences of substring `old` replaced by `new`.

`s.lstrip(), s.rstrip(), s.strip()`
Return a *copy* of the string with the leading (resp. trailing, resp. leading *and* trailing) whitespace removed.

Reference: http://docs.python.org/library/stdtypes.html#string-methods

Exercise 5.A: Write a function `split_comma(s)` which, given a string `s` (containing comma-separated items) returns a *list* of the items. For example:

```
In [4]: split_comma("a,b,c")
Out[4]: ['a', 'b', 'c']
```

Exercise 5.B: Modify `split_comma` to remove whitespace around the returned items, so that `split_comma("a, b, c")` and `split_comma("a,b,c")` return the same result `['a', 'b', 'c']`.

Exercise 5.C: Write a function `unquotes(s)` which, given a string `s` returns a copy of `s` with: *1.* All leading and trailing whitespace removed, *2.* Initial and final double quote “`"`” characters removed (if any). For example:

```
In [5]: unquote('  "abc"')
Out[5]: 'abc'
```

File I/O

File I/O

Code for processing a text file usually looks like this:

```
with open(filename, 'r') as stream:
  # prepare for processing
  for line in stream:
    # process each line
```

File I/O

```
with open(filename, 'r') as stream:
  # prepare for processing
  for line in stream:
    # process each line
```

The **open**(path, mode) function opens the file located at path and returns a “file object” that can be used for reading and/or writing.

Mode is one of 'r', 'w' or 'a' for reading, writing (truncates on open), appending. You can add a ‘+’ character to enable read+write (other effects being the same).

File I/O

```
with open(filename, 'r') as stream:
  # prepare for processing
  for line in stream:
    # process each line
```

This is equivalent to `stream = open(...)` but in addition *closes* the file when the code in the `with`-block is done.

There are many more uses of the `with` statement besides automatically closing files, check out https://jeffknupp.com/blog/2016/03/07/python-with-context-managers/

File I/O

```
with open(filename, 'r') as stream:
  # prepare for processing
  for line in stream:
    # process each line
```

A `for`-loop can be used to process all lines in a file, as if the file were a list.

More on File I/O

The `.read()` method can be used to read the *whole* contents of a file in one go as a single string:

```
>>> s = stream.read()
```

Method `.readlines()` returns a list of all lines in the file:

```
>>> L = stream.readlines()
```

Reference: http://docs.python.org/library/stdtypes.html#file-objects

Type conversions

str(x)	Converts the argument x to a string; for numbers, the base 10 representation is used.
int(x)	Converts its argument x (a number or a string) to an integer; if x is a a floating-point literal, decimal digits are truncated.
float(x)	Converts its argument x (a number or a string) to a floating-point number.

Exercise 5.D: Write a function `load_data(filename)` that reads a file containing one integer number per line, and return a list of the integer values.

Test it with the values.txt file:

```
>>> load_data('values.dat')
[299850, 299740, 299900, 300070, 299930]
```

Exercise 5.E: Write a function `fgrep(s, p)` which returns a list of all lines in file `p` which contain string `s`.

Exercise 5.F: Write a function `read_csv(p)` which reads a CSV (*Comma-sSparated Values*) file and returns a list of all rows in it. A *row* will be represented as a Python list of (string) items.

Filesystem operations, I

These functions are available from the `os` module.

`os.getcwd(), os.chdir(path)`
Return the path to the current working directory / Change the current working directory to `path`.

`os.listdir(dir)`
Return list of entries in directory `dir` (omitting '`.`' and '`..`')

`os.makedirs(path)`
Create a directory; no-op if the directory already exists. Creates all the intermediate-level directories needed to contain the leaf.

`os.rename(old,new)`
Rename a file or directory from `old` to `new`.

Reference: http://docs.python.org/library/os.html

Filesystem operations, II

These functions are available from the `os.path` module.

`os.path.exists(path)`, `os.path.isdir(path)`, `os.path.isfile(path)`
Return `True` if `path` exists / is a directory / is a regular file.

`os.path.basename(path)`, `os.path.dirname(path)`
Return the base name (the part after the last '`/`' character) or the directory name (the part before the last `/` character).

`os.path.abspath(path)`
Make `path` absolute (i.e., start with a `/`).

Reference: http://docs.python.org/library/os.path.html