Python Introduction
Part 2

Aslak Johansen asjo@mmmi.sdu.dk

April 29, 2026

prosa

Part 1:
Files

Files ▷ Mental Model

A file has:

- At least one location on the filesystem.
- Some set of permissions.
- Contents

The contents is a sequence of bytes.

Often, we choose to interpret this sequence as text.

Special characters are used to indicate linebreaks.

Files ▷ Basic Operations

```
fo = open(filename, mode)
# do something with 'fo'
fo.close()
```

Modes:

- `'r'` read mode **(default ⇒ may be omitted)**
- `'w'` write mode (clears file on open)
- `'a'` append mode (continues at end of file)

Read using:

```
lines = fo.readlines()
```

Write using:

```
fo.writelines(lines)
```

Files ▷ Implicit Closing of File Objects

```
with open(input_filename) as fo:
    lines = fo.readlines()

with open(output_filename, 'w') as fo:
    fo.writelines(lines)
```

Files ▷ Reading from a Stream of Lines

```
with open(filename) as fo:
    for line in fo:
        print(line)
```

Files ▷ Directories

```
>>> from os import listdir, path
>>> listdir('/')
['var', 'etc', 'run', 'proc', 'media', 'home', 'sbin', 'srv', 'lost+found',
'boot', 'tmp', 'sys', 'lib32', 'bin', 'dev', 'snap', 'cdrom', 'libx32',
'root', 'usr', 'lib', 'lib64', 'opt', 'mnt']
>>> listdir(path.sep.join(['', 'usr']))
['src', 'local', 'sbin', 'libexec', 'share', 'lib32', 'bin', 'libx32', 'lib',
'lib64', 'include', 'games']
>>> path.isfile('/etc')
False
>>> path.isdir('/etc')
True
>>> path.exists('/etc')
True
>>> path.islink('/etc')
False
>>> import os
>>> os.mkdir('/tmp/dir')
>>> os.unlink('/tmp/somefile')
```

Part 2:
Regular Expressions

Regular Expressions

```
import re

urls = [
    'https://www.gutenberg.org/files/11/11-h/11-h.htm',
    'https://golang.org',
    'http://www.google.com:80/',
    'definitely not a URL',
]
pattern = re.compile('([^:]+)://([^:/]+)(:\d+|)(/.*|)')

for url in urls:
    mo = pattern.match(url)

    if mo:
        print('proto="%s" domain="%s" port="%s" path="%s"' %
              (mo.group(1), mo.group(2), mo.group(3), mo.group(4)))
```

```
proto="https" domain="www.gutenberg.org" port="" path="/files/11/11-h/11-h.htm"
proto="https" domain="golang.org" port="" path=""
proto="http" domain="www.google.com" port=":80" path="/"
```

Regular Expressions

Documentation for the `re` module:
https://docs.python.org/3/library/re.html

A few notes:

- One can match individual characters or character classes.
- A period matches any character.
- One can match the beginning of a string and the end of a string.
- Regular expressions are often used along with string slicing and splits.

Part 3:
Marshalling and JSON

Marshalling and JSON ▷ Marshalling

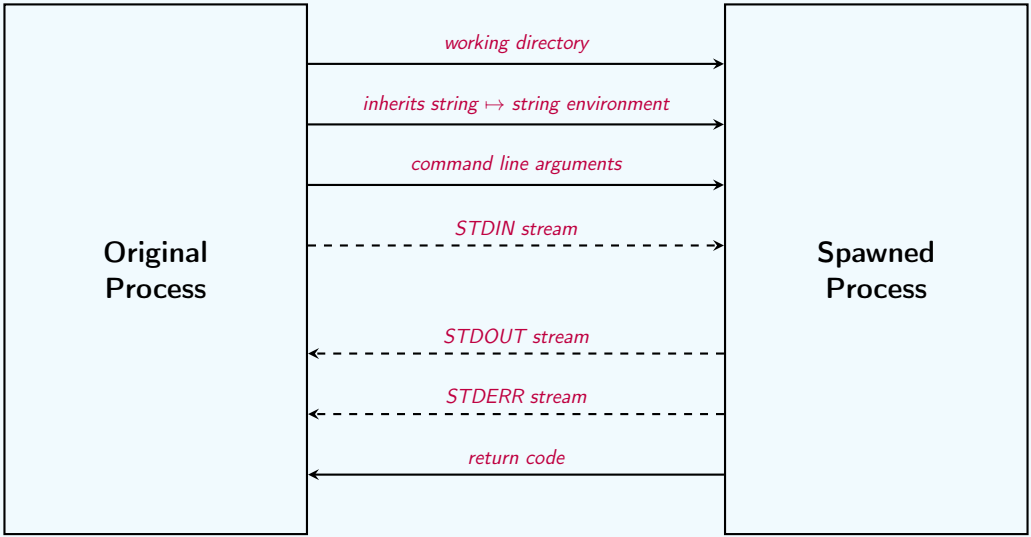
```
>>> import json
>>> data = {'a': 1, 'b': [1,2,3], 'c': 'abc'}
>>> data
{'a': 1, 'b': [1, 2, 3], 'c': 'abc'}

>>> json.dumps(data)
'{"a": 1, "b": [1, 2, 3], "c": "abc"}'
>>> s = json.dumps(data, sort_keys=True, indent=4, separators=(',', ': '))
>>> s
'{\n    "a": 1,\n    "b": [\n        1,\n        2,\n        3\n    ],\n    "c": "abc"\n}'
>>> print(s)
{
    "a": 1,
    "b": [
        1,
        2,
        3
    ],
    "c": "abc"
}
```

Marshalling and JSON ▷ Unmarshalling

```
>>> data2 = json.loads(s)
>>> data2
{'a': 1, 'b': [1, 2, 3], 'c': 'abc'}
```

Part 4:
Processes

Processes ▷ Interface



Processes ▷ Working Directory

```
>>> import os
>>> os.getcwd()
'/home/aslak/myproject'
```

```python
from os import environ

for key in environ:
  print("%s -> %s" % (key, environ[key]))
```

```
$ python3 ../src/env.py
SHELL -> /bin/bash
SESSION_MANAGER -> local/gaia:@/tmp/.ICE-unix/3401,unix/gaia:/tmp/.ICE-unix/
QT_ACCESSIBILITY -> 1
COLORTERM -> truecolor
XDG_MENU_PREFIX -> gnome-
GNOME_DESKTOP_SESSION_ID -> this-is-deprecated
HOSTALIASES -> /home/aslak/.hosts
SSH_AUTH_SOCK -> /run/user/1000/keyring/ssh
DOTNET_CLI_TELEMETRY_OPTOUT -> 1
```

```
from sys import argv, exit

if len(argv) != 3:
    print('Syntax: %s INPUT_FILENAME OUTPUT_FILENAME' % argv[0])
    print('        %s log.txt analysis.csv' % argv[0])
    exit(1)
input_filename  = argv[1]
output_filename = argv[2]

print('%s -> %s' % (input_filename, output_filename))
```

Processes ▷ Process Interaction

```
from subprocess import Popen, STDOUT, PIPE

def system (command, err=STDOUT, out=PIPE):
    p = Popen(command, shell=True, stderr=err, stdout=out)
    output = p.communicate()[0]
    return output.decode('utf-8')

output = system('ls /') # perhaps 'dir c:' on windows
print(output)
```

```
$ python3 processexample.py
bin
boot
cdrom
dev
etc
home
lib
...
```

Part 5:
Python Environments

Python Environments ▷ Installing Modules

Python comes with a command-line tool for managing module installation: `pip` (sometimes called `pip3`)

It allows you to install and uninstall python modules from the *Python Package Index* module repository: https://pypi.org

By default, modules are installed in their newest version, but one can also install specific versions for packages.

Version are defined according to PEP 440 (which is not quite semantic versioning): https://peps.python.org/pep-0440/

```
(.venv) aslak@gaia:~/myproject$ pip3 install rdflib
Collecting rdflib
  Downloading rdflib-7.6.0-py3-none-any.whl.metadata (12 kB)
Collecting pyparsing<4,>=2.1.0 (from rdflib)
  Downloading pyparsing-3.3.2-py3-none-any.whl.metadata (5.8 kB)
Downloading rdflib-7.6.0-py3-none-any.whl (615 kB)
Downloading pyparsing-3.3.2-py3-none-any.whl (122 kB)
Installing collected packages: pyparsing, rdflib
Successfully installed pyparsing-3.3.2 rdflib-7.6.0
(.venv) aslak@gaia:~/myproject$ du -h . | tail -n 1
19M         .
```

Python Environments ▷ Module Management ▷ Uninstall

```
(.venv) aslak@gaia:~/myproject$ pip3 uninstall rdflib
Found existing installation: rdflib 7.6.0
Uninstalling rdflib-7.6.0:
  Would remove:
    /home/aslak/myproject/.venv/bin/csv2rdf
    /home/aslak/myproject/.venv/bin/rdf2dot
    /home/aslak/myproject/.venv/bin/rdfgraphisomorphism
    /home/aslak/myproject/.venv/bin/rdfpipe
    /home/aslak/myproject/.venv/bin/rdfs2dot
    /home/aslak/myproject/.venv/bin/sparqlquery
    /home/aslak/myproject/.venv/lib/python3.13/site-packages/rdflib-7.6.0.dist-info/*
    /home/aslak/myproject/.venv/lib/python3.13/site-packages/rdflib/*
Proceed (Y/n)? y
  Successfully uninstalled rdflib-7.6.0
(.venv) aslak@gaia:~/myproject$ du -h . | tail -n 1
14M        .
```

```
(.venv) aslak@gaia:~/myproject$ pip freeze > requirements.txt
(.venv) aslak@gaia:~/myproject$ cat requirements.txt
pyparsing==3.3.2
rdflib==7.6.0
(.venv) aslak@gaia:~/myproject$
```

```
(.venv) aslak@gaia:~/myproject$ pip3 install -r requirements.txt
Requirement already satisfied: pyparsing==3.3.2 in
↪  ./.venv/lib/python3.13/site-packages (from -r requirements.txt
↪  (line 1)) (3.3.2)
Collecting rdflib==7.6.0 (from -r requirements.txt (line 2))
  Using cached rdflib-7.6.0-py3-none-any.whl.metadata (12 kB)
Using cached rdflib-7.6.0-py3-none-any.whl (615 kB)
Installing collected packages: rdflib
Successfully installed rdflib-7.6.0
(.venv) aslak@gaia:~/myproject$
```

Python Environments ▷ Versioning Problem

Python code tend to be rather picky when it comes to module versions.

The more python codebases you have lying around the more likely you are to run into a versioning conflict. That is, **one piece of python code depends on a version of a module while a different piece of python code depends of an incompatible version of the same module**.

These codebases cannot (successfully) coexist in the same environment: At most one can be executed.

The trick is multiple environments. By default you have:

- The system-wide environment.
- A per-user environment.

But ...

```
aslak@gaia:~/myproject$ python -m venv .venv
aslak@gaia:~/myproject$ ls -a
.  ..  .venv
aslak@gaia:~/myproject$ ls -a .venv
.  ..  bin  .gitignore  include  lib  lib64  pyvenv.cfg
```

Linux/macOS (bash/zsh):

```
source .venv/bin/activate
```

Windows (cmd):

```
.venv\Scripts\activate.bat
```

Windows (PowerShell):

```
.venv\Scripts\Activate.ps1
```

```
aslak@gaia:~/myproject$ source .venv/bin/activate
(.venv) aslak@gaia:~/myproject$ deactivate
aslak@gaia:~/myproject$
```

`pip` has a number of downsides:

- A *freeze* generates platform-specific list of module versions.
- It operates at a low granularity of control.
- It can be slow and reource hungry in CI/CD pipelines.

The `uv` project tries to address this: https://docs.astral.sh/uv/

It claims to be compatible with `pip`.

Part 6:
Exercises

Exercises ▷ Marshalling

Make some complex data structure that consists of integers, floats, booleans, strings, lists, tuples and dictionaries.

Marshall that to JSON and store the resulting string in a file.

Unmarshall the contents of that file to a new datastructure.

How similar is the new datastructure to the old?

Exercises ▷ Regular Expressions

You input text: *"Avocados are considered expensive but delicious. In the US they are about 1$. In Denmark they are approx 15 kr and in Holland they are roughly 1.5€. In the UK the price is 1.5 £."*

Here we have a string containing 4 different prices for avocados. The exercise is to find the average price.

Specifically:

1. Define a dictionary that maps currencies to conversion factors.
2. Define a regular expression that matches a number followed by a currency.
3. Find all matches of this regular expression.
4. For each match, look up the conversions factor, parse the number and multiply these.
5. Find the average value.

Exercises ▷ CSV Corrections

The CSV file format can hold tabular data. The format:

- Rows are separated by linebreaks.
- Columns are separated by commas.

Excel is infamous for misinterpreting the CSV format. It uses semicolons instead of commas for separation.

The exercise:

1. Make a script that takes three parameters so that it can be called like this:
   ```
   python3 csv_reformatter.py fix input.csv output.csv
   python3 csv_reformatter.py unfix input.csv output.csv
   ```
2. Decide on a reasonable intermediate format.
3. Write functions for fundamental operations like reading, adjusting and writing data.
4. Write the main code that parses the command line arguments, reads the input, adjusts it and writes the output.
5. What could go wrong when converting between the two versions of the format?

Exercises ▷ Offloading to Other Processes

Write a script called `read.py` that takes a filename as a parameter and prints out the contents of that file.

Write a script called `write.py` that takes a filename as a parameter and write whatever it gets on STDIN to this file.

Write a script called `filter.py` that takes a word as a parameter and forwards every line that gets from STDIN to STDOUT if the word is present.

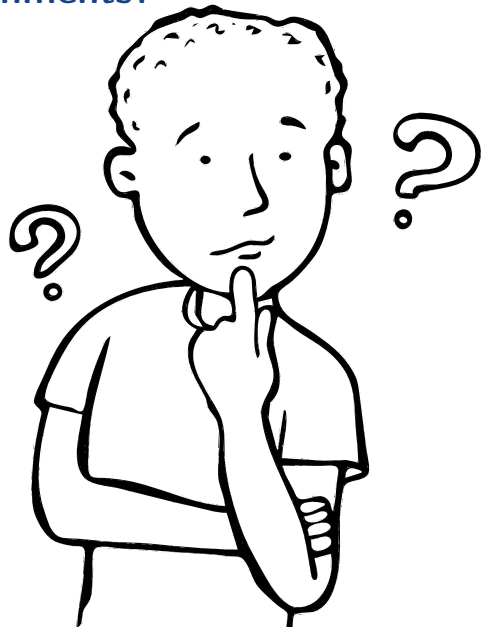
Write a script that calls `read.py`, uses `filter.py` to filter the result and finally uses `write.py` to store the result.

Hint:
https://docs.python.org/3/library/subprocess.html#replacing-shell-pipeline

Where to Find

https://python.asjo.dk

Questions and Comments?



https://openclipart.org/detail/238687/boy-thinking-of-question