Configuration files in Python (../configuration-files-in-python/)

Contents

Most interesting programs need some kind of configuration:

- Content Management Systems like WordPress blogs, WikiMedia and Joomla need to store the information where the database server is (the hostname) and how to login (username and password)
- Proprietary software might need to store if the software was registered already (the serial key)
- Scientific software could store the path to BLAS libraries

For very simple tasks you might choose to write these configuration variables directly into the source code. But this is a bad idea when you upload the code to GitHub.

I will explain some alternatives I got to know for Python.

I've created a module for configuration handling: `cfg_load` **(https://github.com/MartinThoma/cfg_load)**

Python Configuration File

The simplest way to write configuration files is to simply write a separate file that contains Python code. You might want to call it something like `databaseconfig.py`. Then you could add the line `*config.py` to your `.gitignore` file to avoid uploading it accidentally.

A configuration file could look like this:

```
#!/usr/bin/env python
import preprocessing
mysql = {'host': 'localhost',
         'user': 'root',
         'passwd': 'my secret password',
         'db': 'write-math'}
preprocessing_queue = [preprocessing.scale_and_center,
                       preprocessing.dot_reduction,
                       preprocessing.connect_lines]
use_anonymous = True
```

Within the actual code, you can use it like this:

```
#!/usr/bin/env python
import databaseconfig as cfg
connect(cfg.mysql['host'], cfg.mysql['user'], cfg.mysql['password'])
```

The way you include the configuration might feel very convenient at a first glance, but imagine what happens when you get more configuration variables. You definitely need to provide an example configuration file. And it is hard to resist the temptation to include code within the configuration file.

JSON

JSON (https://en.wikipedia.org/wiki/JSON) is short for JavaScript Object Notation. It is widespread and thus has good support for many programming languages.

The configuration might look like this:

```
{
    "mysql":{
        "host":"localhost",
        "user":"root",
        "passwd":"my secret password",
        "db":"write-math"
    },
    "other":{
        "preprocessing_queue":[
            "preprocessing.scale_and_center",
            "preprocessing.dot_reduction",
            "preprocessing.connect_lines"
            ],
        "use_anonymous":true
    }
}
```

You can read it like this:

```
import json

with open('config.json') as json_data_file:
    data = json.load(json_data_file)
print(data)
```

which outputs

```
{u'mysql': {u'db': u'write-math',
            u'host': u'localhost',
            u'passwd': u'my secret password',
            u'user': u'root'},
 u'other': {u'preprocessing_queue': [u'preprocessing.scale_and_center',
                                     u'preprocessing.dot_reduction',
                                     u'preprocessing.connect_lines'],
            u'use_anonymous': True}}
```

Writing JSON files is also easy. Just build up the dictionary and use

```
import json
with open('config.json', 'w') as outfile:
    json.dump(data, outfile)
```

YAML

YAML (https://en.wikipedia.org/wiki/YAML) is a configuration file format. Wikipedia says:

> *YAML (rhymes with camel) is a human-readable data serialization format that takes concepts from programming languages such as C, Perl, and Python, and ideas from XML and the data format of electronic mail (RFC 2822). YAML was first proposed by Clark Evans in 2001, who designed it together with Ingy döt Net and Oren Ben-Kiki. It is available for several programming languages.*

The file itself might look like this:

```
mysql:
    host: localhost
    user: root
    passwd: my secret password
    db: write-math
other:
    preprocessing_queue:
        - preprocessing.scale_and_center
        - preprocessing.dot_reduction
        - preprocessing.connect_lines
    use_anonymous: yes
```

You can read it like this:

```
import yaml

with open("config.yml", 'r') as ymlfile:
    cfg = yaml.load(ymlfile)

for section in cfg:
    print(section)
print(cfg['mysql'])
print(cfg['other'])
```

It outputs:

```
other
mysql
{'passwd': 'my secret password',
 'host': 'localhost',
 'db': 'write-math',
 'user': 'root'}
{'preprocessing_queue': ['preprocessing.scale_and_center',
                         'preprocessing.dot_reduction',
                         'preprocessing.connect_lines'],
 'use_anonymous': True}
```

There is a `yaml.dump` method, so you can write the configuration the same way. Just build up a dictionary.

YAML is used by the Blender project.

Resources

- **Documentation (https://docs.python.org/3/library/configparser.html)**

INI

INI files look like this:

```
[mysql]
host=localhost
user=root
passwd=my secret password
db=write-math

[other]
preprocessing_queue=["preprocessing.scale_and_center",
                     "preprocessing.dot_reduction",
                     "preprocessing.connect_lines"]
use_anonymous=yes
```

ConfigParser

Basic example

The file can be loaded and used like this:

```
#!/usr/bin/env python

import ConfigParser
import io

# Load the configuration file
with open("config.ini") as f:
    sample_config = f.read()
config = ConfigParser.RawConfigParser(allow_no_value=True)
config.readfp(io.BytesIO(sample_config))

# List all contents
print("List all contents")
for section in config.sections():
    print("Section: %s" % section)
    for options in config.options(section):
        print("x %s:::%s:::%s" % (options,
                                  config.get(section, options),
                                  str(type(options))))

# Print some contents
print("\nPrint some contents")
print(config.get('other', 'use_anonymous'))  # Just get the value
print(config.getboolean('other', 'use_anonymous'))  # You know the datatype?
```

which outputs

```
List all contents
Section: mysql
x host:::localhost:::<type 'str'>
x user:::root:::<type 'str'>
x passwd:::my secret password:::<type 'str'>
x db:::write-math:::<type 'str'>
Section: other
x preprocessing_queue:::["preprocessing.scale_and_center",
"preprocessing.dot_reduction",
"preprocessing.connect_lines"]:::<type 'str'>
x use_anonymous:::yes:::<type 'str'>

Print some contents
yes
True
```

As you can see, you can use a standard data format that is easy to read and write. Methods like `getboolean` and `getint` allow you to get the datatype instead of a simple string.

Writing configuration

```
import os
configfile_name = "config.ini"

# Check if there is already a configurtion file
if not os.path.isfile(configfile_name):
    # Create the configuration file as it doesn't exist yet
    cfgfile = open(configfile_name, 'w')

    # Add content to the file
    Config = ConfigParser.ConfigParser()
    Config.add_section('mysql')
    Config.set('mysql', 'host', 'localhost')
    Config.set('mysql', 'user', 'root')
    Config.set('mysql', 'passwd', 'my secret password')
    Config.set('mysql', 'db', 'write-math')
    Config.add_section('other')
    Config.set('other',
               'preprocessing_queue',
               ['preprocessing.scale_and_center',
                'preprocessing.dot_reduction',
                'preprocessing.connect_lines'])
    Config.set('other', 'use_anonymous', True)
    Config.write(cfgfile)
    cfgfile.close()
```

results in

```
[mysql]
host = localhost
user = root
passwd = my secret password
db = write-math

[other]
preprocessing_queue = ['preprocessing.scale_and_center', 'preprocessing.dot_reduction', 'preprocessing.connect_lines']
use_anonymous = True
```

XML

Seems not to be used at all for configuration files by the Python community. However, parsing / writing XML is easy and there are plenty of possibilities to do so with Python. One is BeautifulSoup:

```
from BeautifulSoup import BeautifulSoup

with open("config.xml") as f:
    content = f.read()

y = BeautifulSoup(content)
print(y.mysql.host.contents[0])
for tag in y.other.preprocessing_queue:
    print(tag)
```

where the config.xml might look like this:

```
<config>
    <mysql>
        <host>localhost</host>
        <user>root</user>
        <passwd>my secret password</passwd>
        <db>write-math</db>
    </mysql>
    <other>
        <preprocessing_queue>
            <li>preprocessing.scale_and_center</li>
            <li>preprocessing.dot_reduction</li>
            <li>preprocessing.connect_lines</li>
        </preprocessing_queue>
        <use_anonymous value="true" />
    </other>
</config>
```

File Endings

File Endings give the user and the system an indicator about the content of a file. Reasonable file endings for configuration files are

- `*config.py` for Python files
- `*.yaml` or `*.yml` if the configuration is done in YAML format
- `*.json` for configuration files written in JSON format
- `*.cfg` or `*.conf` to indicate that it is a configuration file
- `*.ini` for "initialization" are quite widespread (see **Wiki (https://en.wikipedia.org/wiki/INI_file)**)
- `~/.[my_app_name]rc` is a VERY common naming scheme for configuration files on Linux systems. RC is a reference to an old computer system and means "run common".

That said, I think I prefer `*.conf`. I think it is a choice that users understand.

But you might also consider that `*.ini` might get opened by standard in a text editor. For the other options, users might get asked which program they want to use.

Resources

- **JSON Online Parser (http://json.parser.online.fr/)**
- **What is the difference between YAML and JSON? When to prefer one over the other? (https://stackoverflow.com/q/1726802/562769)**
- **Why do so many projects use XML for configuration files? (https://stackoverflow.com/q/791761/562769)**
- **YAML Lint (http://yamllint.com/)**
- **TOML (https://en.wikipedia.org/wiki/TOML)**: Very similar to INI files.

Published

Jul 27, 2014
by **Martin Thoma (../author/martin-thoma/)**

Category

Code (../categories.html#code-ref)

Tags

- **Configuration 2 (../tags.html#configuration-ref)**
- **INI 1 (../tags.html#ini-ref)**
- **JSON 3 (../tags.html#json-ref)**
- **Python 98 (../tags.html#python-ref)**
- **XML 1 (../tags.html#xml-ref)**
- **YAML 1 (../tags.html#yaml-ref)**

Contact

(https://twitter.com/themoosemind) **(mailto:info@martin-thoma.de)** **(https://github.com/MartinThoma)**
(https://stackoverflow.com/users/562769/martin-thoma)