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`

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 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

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

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)
- `~/.[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
- What is the difference between YAML and JSON? When to prefer one over the other?
- Why do so many projects use XML for configuration files?
- YAML Lint
- TOML: Very similar to INI files.

Francois Nader • 4 years ago
I think it's worth mentioning that to use 'import yaml', you need to install the pyyaml package
25 | • Reply • Share ›

adlgrbz • 2 years ago • edited

Additional info:

Python 2x

```
import ConfigParser
```

Python 3x

```
import configparser
```

5 ⌃ | ⌄ • Reply • Share ›

Marc Richter • 4 years ago

Hi Martin,
great article - thanks for sharing this! I like especially, that you give an detailed example for all introduced kind of possibilities.

5 ⌃ | ⌄ • Reply • Share ›

Martin Thoma Mod ➦ Marc Richter • 4 years ago

You're welcome.

I always like it best when I can see the things in action. It is so much easier then to grasp the concept / quickly skim through the possibilities. I'm glad it helps other, too :-)

see more

⌃ | ⌄ • Reply • Share ›

Bruno Rocha • 3 years ago

dynaconf a layered configuration system for Python applications -
with strong support for 12-factor applications
and extensions for Flask and Django.

https://dynaconf.readthedoc...

4 ⌃ | ⌄ 1 • Reply • Share ›

Zach Jacobs • 4 years ago

Thanks for the resource! I wanted to let you know that your "INI file" link forwarded me to a spam page.

2 ^ | ˅ • Reply • Share ›

Martin Thoma Mod ➦ Zach Jacobs • 4 years ago

Thank you! (I removed the spam-link)

^ | ˅ • Reply • Share ›

Ikem Krueger • 4 years ago

In your ConfigParser example you open a yaml file.

Not sure if that was your intention.

2 ^ | ˅ • Reply • Share ›

Martin Thoma Mod ➦ Ikem Krueger • 4 years ago

Ooops, no, it was not. I fixed it. Thank you :-)

1 ^ | ˅ • Reply • Share ›

John • 2 years ago

.rc is for 'run commands', not 'run common' https://en.wikipedia.org/wi...

1 ^ | ˅ • Reply • Share ›

Colin 't Hart • 4 years ago

There is of course some code missing in the .py config file example; you need to add "cfg." before each parameter:

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

see more

1 ^ | ˅ • Reply • Share ›

Martin Thoma Mod ➦ Colin 't Hart • 4 years ago

Oh, you're right of course. I corrected it.

^ | ˅ • Reply • Share ›

Hans Ginzel • a year ago

See Munch, https://stackoverflow.com/q...

```
import yaml
from munch import munchify
cfg = munchify(yaml.load(…))print(cfg.mysql.user)
```

^ | ˅ • Reply • Share ›

glenfant • a year ago

Hi Martin,
Congrats for this summary. But consider executing the Python config file in some sandbox to prevent potential security issues. I wrote a small blog poste that adds some flexibility and safety to a pure Python configuration file handling, leveraging the "runpy" module from stdlib. Opinions are welcome.

http://glenfant.github.io/s...

see more

^ | ˅ • Reply • Share ›

Anselm Kiefner ➦ glenfant • 2 months ago

runpy is not a sandbox, as mentioned in the first lines of https://docs.python.org/3/l...

^ | ˅ • Reply • Share ›

gk_2000 • a year ago

Other than the python module approach all others bind the config file location to the same location as the python file, which defeats one of the purposes of having a config driven program, which is to have it run differently for different contexts

see more

^ | ⌄ • Reply • Share ›

Martin Thoma ➦ gk_2000 • a year ago

> all others bind the config file location to the same location as the python file

Why do you think that? You can, of course, store all the files at any location. You just need to tell the program where the file is. The two common ways are by defining a fixed location (`~/.somerc`) or by supplying the config as a CLI parameter.

see more

^ | ⌄ • Reply • Share ›

adnan ali • 2 years ago

I need a simple Python Script converted to C#.The zip contains both the main script and .json settings file.

```
{
"Info": [
{
"Website": "http://dextrader.net/#/app/...",
"SleepTime": 2
}
],
"TelegramBot": [
{
"BotToken": "1039495280:AAEKcWO8nsL7UxytKULbUafye4Mp6xWEECY",
"BotChatID": "-1001433084694"
```

```
],
"WebDriver": [
{
"Driver": "Chrome",
```

see more

^ | ˅ • Reply • Share ›

Iam Towrite • 2 years ago

Is `import preprocessing` necessary for this article? If it is not relevant please consider removing it.

^ | ˅ • Reply • Share ›

Andre Oliveira Dias • 3 years ago

Very comprehensive article. Thank you from a perl programmer in Brazil

^ | ˅ • Reply • Share ›

bronco • 3 years ago

Excelent!
Thanks!

^ | ˅ • Reply • Share ›

Martin Thoma Mod • 3 years ago

I should include https://github.com/MartinTh... in this article.

^ | ˅ • Reply • Share ›

Tim Bramlett • 3 years ago

Great tutorial!

^ | ˅ • Reply • Share ›

Martin Thoma Mod • 4 years ago

See also: https://github.com/MartinTh...

^ | ˅ • Reply • Share ›

Jiyuan Zheng • 4 years ago • edited

Is there a configuration format that supports variables and simple expression like "1+1"? (other python)

^ | ˅ • Reply • Share ›

Martin Thoma Mod ➦ Jiyuan Zheng • 3 years ago

From the formats above, only Python can do that. However, most often when I had the need to do so, it was rather because of bad software design. Configuration files should be static and not compute something.

see more

Reply • Share ›

Colin 't Hart • 4 years ago

I'd love to hear from anyone using plain Python .py configuration files and have an elegant solution for being able to override configuration values for use on different environments. We have some ugly solution in place right now.
Also I'd love to hear tips on the use of centralised configuration files; we have many config.py files, one in each project, with many duplicated values...

see more

Reply • Share ›

kpd Colin 't Hart • 4 years ago

Redchrom has a nice solution posted in the comments here:
https://www.reddit.com/r/Py...

Reply • Share ›

Martin Thoma Mod Colin 't Hart • 4 years ago

By override, do you mean similar to https://martin-thoma.com/cf... ? So that you have a "hierarchy" of configuration file (e.g. a default one of the program, a system wide one and a user-specific one)?

see more

^ | ˅ • Reply • Share ›

Colin 't Hart ➦ Martin Thoma • 4 years ago

Yes. But we have just two levels: one for the program that contains values that work in production, and an optional, local one that is used when running the same program in a different environment (development, test, etc).

see more

1 ^ | ˅ • Reply • Share ›

Stoneland ➦ Colin 't Hart • 4 years ago

Would love to hear of a solution here as well. I'm currently doing a 'config.py' for the program values and a 'config.yml' for the optional, local one. Feels hacky though....

see more

^ | ˅ • Reply • Share ›

shekhar ➦ Stoneland • 3 years ago

I always liked python config files. So converge https://pypi.org/project/co... is my attempt address it. Still WIP but stable enough as we in production.

^ | ˅ • Reply • Share ›

Martin Thoma Mod ➦ shekhar • 3 years ago

For PyPI, you might want to look at https://github.com/MartinTh...

^ | ˅ • Reply • Share ›

Martin Thoma Mod ➦ Colin 't Hart • 4 years ago

Is there any reason not to use something similar to what I've described in the article linked above? (You can read the data from the Python module into a dictionary.)

see more

^ | ˅ • Reply • Share ›

Andrés • 5 years ago

Great post! Thank you.

^ | ˅ • Reply • Share ›

Polinompol • 5 years ago

So do you know what is and advantage of using simple python file vs external file format?

^ | ˅ • Reply • Share ›

Martin Thoma Mod → Polinompol • 5 years ago

Pro Python file: easy to use for Python programmers, flexible

Contra Python file: hard to use for other languages / non-python programmers; flexible

^ | ˅ • Reply • Share ›

face_facts • 6 years ago

Good overview of file-based options. For adherents to the 12-Factor App philosophy (http://12factor.net/), a good option is to pass parameters as environment variables, and to define each variable as a single file in a folder, allowing use of the very awesome envdir utility to translate files --> environment at runtime.

see more

^ | ˅ • Reply • Share ›

Published

Jul 27, 2014
by Martin Thoma

Category

Code

Tags

Configuration 2

INI 1

JSON 4

Python 139

XML 1

YAML 2

Contact