**[Configuration files in Python](https://martin-thoma.com/configuration-files-in-python/)**

**Contents**

* [Python Configuration File](https://martin-thoma.com/configuration-files-in-python/#python-configuration-file)
* [JSON](https://martin-thoma.com/configuration-files-in-python/#json)
* [YAML](https://martin-thoma.com/configuration-files-in-python/#yaml)
* [INI](https://martin-thoma.com/configuration-files-in-python/#ini)
  + [ConfigParser](https://martin-thoma.com/configuration-files-in-python/#configparser)
* [XML](https://martin-thoma.com/configuration-files-in-python/#xml_1)
* [File Endings](https://martin-thoma.com/configuration-files-in-python/#file-endings)
* [Resources](https://martin-thoma.com/configuration-files-in-python/#resources_1)

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.

**Python Configuration File [¶](https://martin-thoma.com/configuration-files-in-python/" \l "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 [¶](https://martin-thoma.com/configuration-files-in-python/" \l "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 [¶](https://martin-thoma.com/configuration-files-in-python/" \l "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 [¶](https://martin-thoma.com/configuration-files-in-python/" \l "resources)**

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

**INI** [**¶**](https://martin-thoma.com/configuration-files-in-python/#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 [¶](https://martin-thoma.com/configuration-files-in-python/" \l "configparser)**

**Basic example [¶](https://martin-thoma.com/configuration-files-in-python/" \l "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 [¶](https://martin-thoma.com/configuration-files-in-python/" \l "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 [¶](https://martin-thoma.com/configuration-files-in-python/" \l "xml_1)**

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>