Python at Cognite

> **Work in progress**
> This page is under construction, so some details may be missing. If you find something that's missing or could be improved, feel free to edit this page yourself.

Installing Python

macOS

Install using brew by following the instructions here: https://docs.python-guide.org/starting/install3/osx/

Ubuntu

Install python on Ubuntu

```
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt-get update
sudo apt-get install python3.7 python3.7-dev
# Upgrade pip
python3.7 -m pip install --upgrade pip
```

Windows

Download and install the latest release from here: https://www.python.org/downloads/windows/

Setting up pip to use Artifactory - the Cognite package index

All that is needed is to tell pip to look for packages in our package index as well as the default one.

Create/edit the file

~/.pip/pip.conf

```
[global]
index-url = https://pypi.python.org/simple
extra-index-url =
https://<artifactory-username>:<artifactory-password>@cognite.jfrog.io/cognite/api/pypi/snakepit/simple
```

Artifactory credentials can be created/found by following the instructions here.

On Windows the file should be located in the folder `C:\Users\USERNAME\AppData\Roaming\pip` and have the filename `pip.ini`.

You should also set the artifactory index url as an environment variable with the name PYPI_ARTIFACTORY_URL seeing as our python projects

set up with Pipenv need it.

```
$ export
PYPI_ARTIFACTORY_URL=https://<artifactory-username>:<artifactory-passwor
d>@cognite.jfrog.io/cognite/api/pypi/snakepit/simple
```

Publishing packages to the private package index

To publish packages you first need to make sure it's setup according to the specifications given here: https://packaging.python.org/tutorials/distributing-packages/

To publish we need the twine package

```
pip install twine
```

Then all we need to do is tell twine where to find our privat package index

~/.pypirc

```
[distutils]
index-servers =
  pypi
  artifactory

[pypi]
username=<username-from-lastpass>
password=<password-from-lastpass>

[artifactory]
repository=https://cognite.jfrog.io/cognite/api/pypi/snakepit
username=<artifactory-username>
password=<artifactory-password>
```

Then we can first build a distribution of our package and publish it

```
python setup.py sdist
twine upload -r artifcatory dist/*
```

Using Pipenv

Pipenv automatically creates and manages a virtualenv for your projects, as well as adds/removes packages from your `Pipfile` as you install/uninstall packages. It also generates the ever-important `Pipfile.lock`, which is used to produce deterministic builds. *Pipenv* has become the official Python-*recommended* resource for managing package dependencies and is the tool we use at Cognite.

To install pipenv run the following command:

```
pip install -U pipenv
```

You can then set up pipenv by running the following command in the root directory of your project:

```
pipenv install
```

This will generate a Pipfile and a Pipfile.lock, as well as a virtual environment which can be activated by running this command:

```
pipenv shell
```

This will spawn a shell with the virtualenv activated, it will also load the environment variables specified in a .env file to your virtual environment.

To install packages from an existing Pipfile.lock run the following command:

```
pipenv sync --dev
```

To install new packages to your pipfile run this command:

```
pipenv install <package-name>
```

Here are some other useful commands which pipenv supplies:

- `run` will run a given command from the virtualenv, with any arguments forwarded (e.g. `$ pipenv run python`).
- `check` asserts that PEP 508 requirements are being met by the current environment.
- `graph` will print a pretty graph of all your installed dependencies.

To tell pipenv to point to Artifactory add the following configuration at the top of your Pipfile and set the environment variable PYPI_ARTIFACTORY_URL as described above.

```
[[source]]
url = "https://pypi.python.org/simple"
verify_ssl = true
name = "pypi"

[[source]]
url = "${PYPI_ARTIFACTORY_URL}"
verify_ssl = true
name = "artifactory"
```

Pyenv

In order to seamlessly work with multiple versions of Python we recommend using pyenv. If you have pyenv installed and configured, Pipenv will automatically ask you if you want to install a required version of Python if you don't already have it available.

To install pyenv run this command

```
$ brew install pyenv
```

Then add the following to your .bashrc, .bash_profile, .zshrc, or equivalent.

```
eval "$(pyenv init -)"
```

On newer versions of Mac OS (Mojave +) you can get failed builds and may need install additional headers in your build environment as discussed on the pyenv troubleshooting page.

Code style

At Cognite we use the following tools for formatting and linting our python code:

- iSort for optimizing imports. More information can be found here: https://github.com/timothycrosley/isort
- Black for formatting code. More information can be found here: https://github.com/ambv/black

To integrate these tools with your IDE follow these steps:

Pycharm

1. Install packages

   ```
   pip install black isort
   ```

2. Open External tools in PyCharm with `File -> Settings -> Tools -> External Tools`.
3. Click the + icon to add a new external tool with the following values:
   - Name: Black
   - Description: Black is the uncompromising Python code formatter.
   - Program: $PyInterpreterDirectory$/black
   - Arguments: $FilePath$ --line-length=120
4. Click the + icon to add a new external tool with the following values:
   - Name: iSort
   - Description: iSort is a imports optimizer for Python.
   - Program: $PyInterpreterDirectory$/isort
   - Arguments: $FilePath$ --line-width 120 -m 3 -tc
5. Start recording a new PyCharm macro with `Edit -> Macros -> Start Macro Recording`.
6. Run iSort with `Tools -> External Tools -> iSort`.
7. Run Black with `Tools -> External Tools -> Black`.
8. Save your files with `File -> Save All`.
9. Stop recording your PyCharm macro with `Edit -> Macros -> Stop Macro Recording`.
10. Give your macro a name
11. Open Keymap in PyCharm with `File -> Settings -> Keymap`
12. Open the macros section, select your macro and add the keyboard shortcut you would normally use to save your files.

Now every time you save a file your macro will be run, formatting your code and printing a lint report.

Pre-commit hooks

To enforce this code style in your python project as pre-commit hooks follow the instructions in this repository https://github.com/cognitedata/python-pre-commit-hooks.

Naming conventions

We follow the pep 8 style guide naming conventions. It can be smart to use e.g. flake8 or pylint to enforce this. Some rules are:

- Variables, function, method, arguments and keyword argument names are all camel_case, i.e. lower case letters and words separated by underscore
- Start the name with underscore to indicate that a module variable, module function or method is private
- Class names are CamelCase, i.e. only the first letter of each word is capitalized.
- Constants are upper case with words separated by underscore
- All instance methods have self as the first argument
- All class methods have cls as the first argument
- And for modules: Modules should have short, all-lowercase names. Underscores can be used in the module name if it improves readability.

Naming convention samples

```
# in some file my_module.py

THIS_IS_A_CONSTANT = 1234
this_is_a_variable = ...

def public_module_function(some_argument,
some_keyword_argument="default-value"):
 ...

def _private_module_function(...):
 ...

class SomeClass(object):
 _private_class_variable = ...
 public_class_variable = ...

 def public_instance_method(self, ...):
  ...

 def _private_instance_method(self, ...):
  ...

 @classmethod
 def public_class_method(cls, ...):
  ...

 @classmethod
 def _private_class_method(cls,...):
  ...
```

Docstrings

We use Google Style Python Docstrings to document our code inline. For examples on how to write these docstrings click here.

Other

If it is not already covered by the above items, we follow the pep 8 style guide.

Running python scripts in Google Cloud Platform

For a guide on how to run python scripts in Google Cloud Platform by utilizing Docker and Kubernetes, see this guide on github.