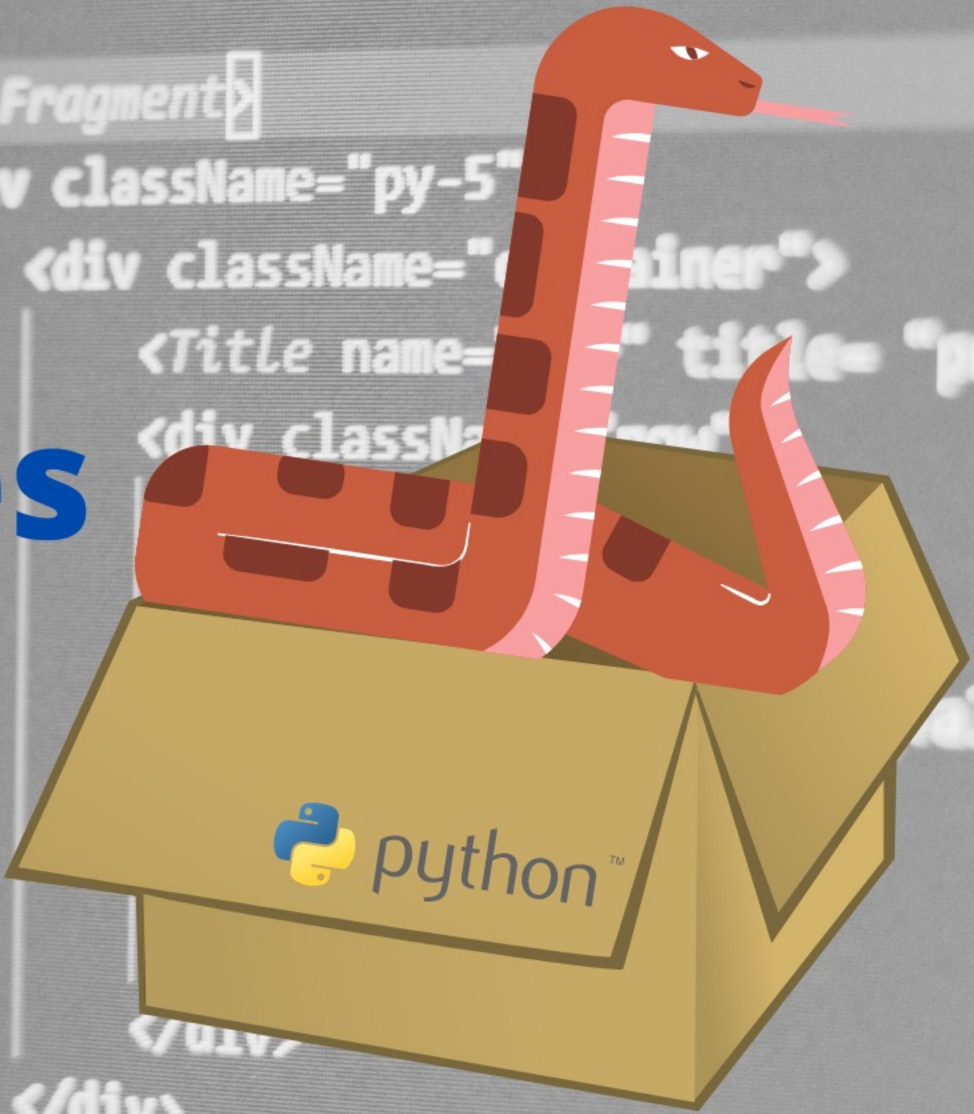
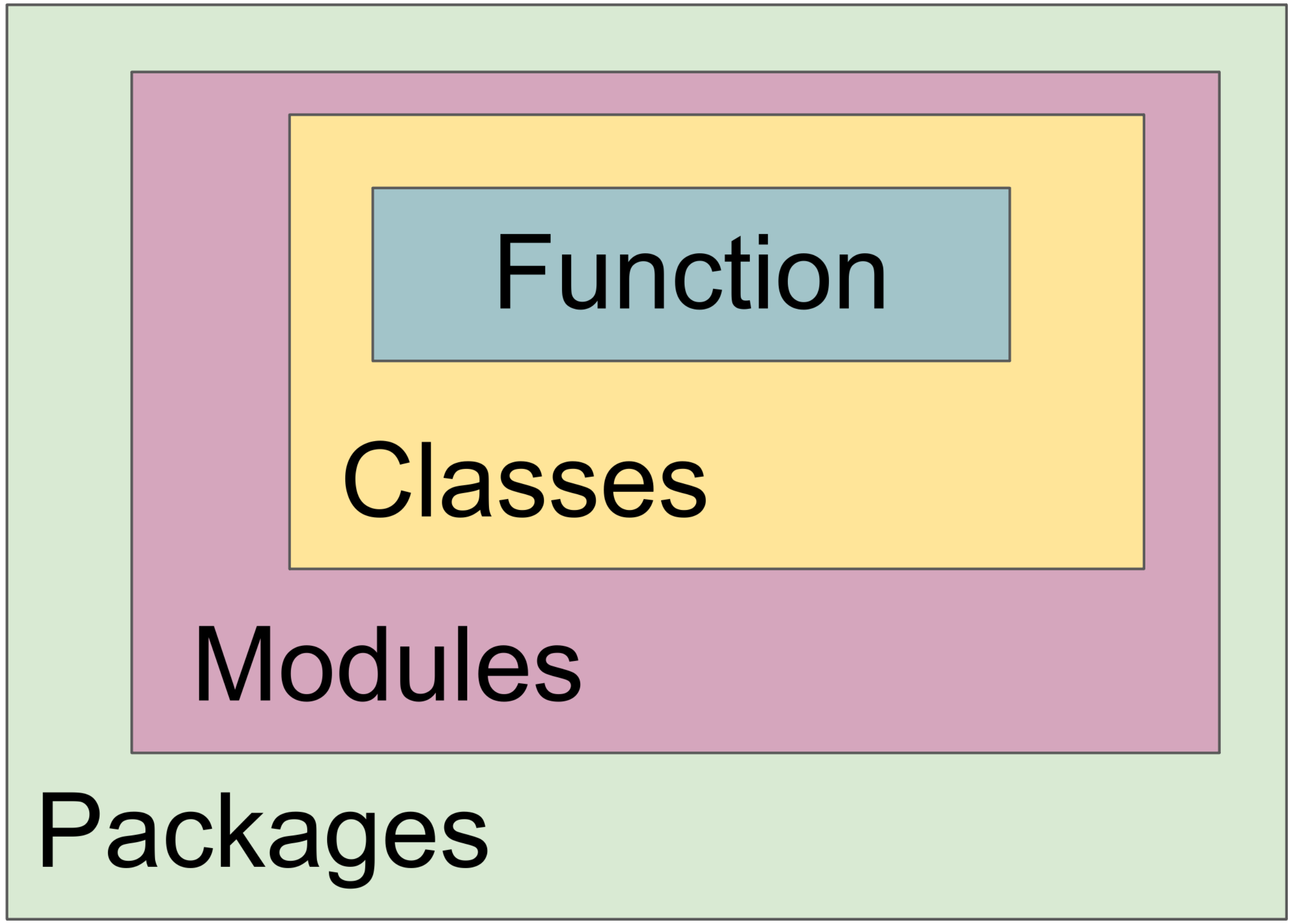
Python Packages

Group session #3



Python code structure

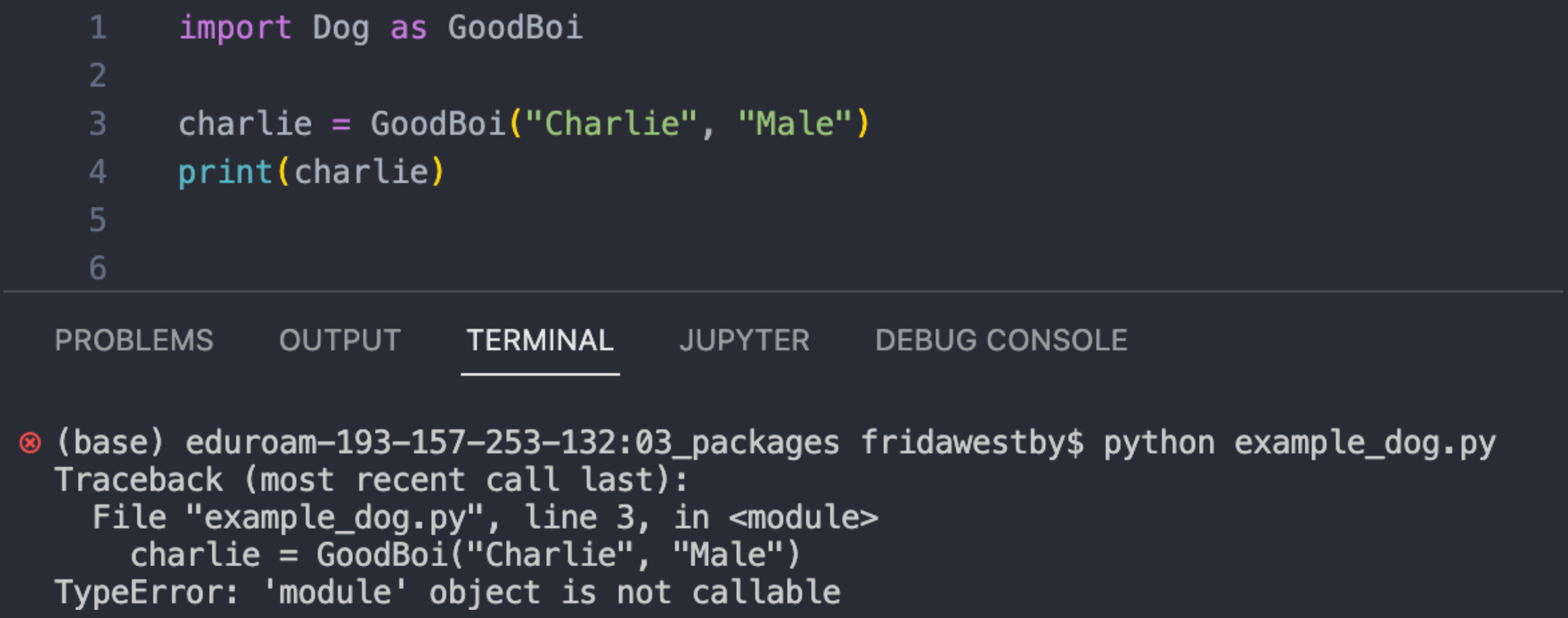


Python module

- A module is a file consisting of Python code.
- A module can define
    - Functions
    - Classes
    - Variables
- A module can also include runnable code.

- E.g. definitions from a module can be imported into other modules

Example: Dog class from group session #2

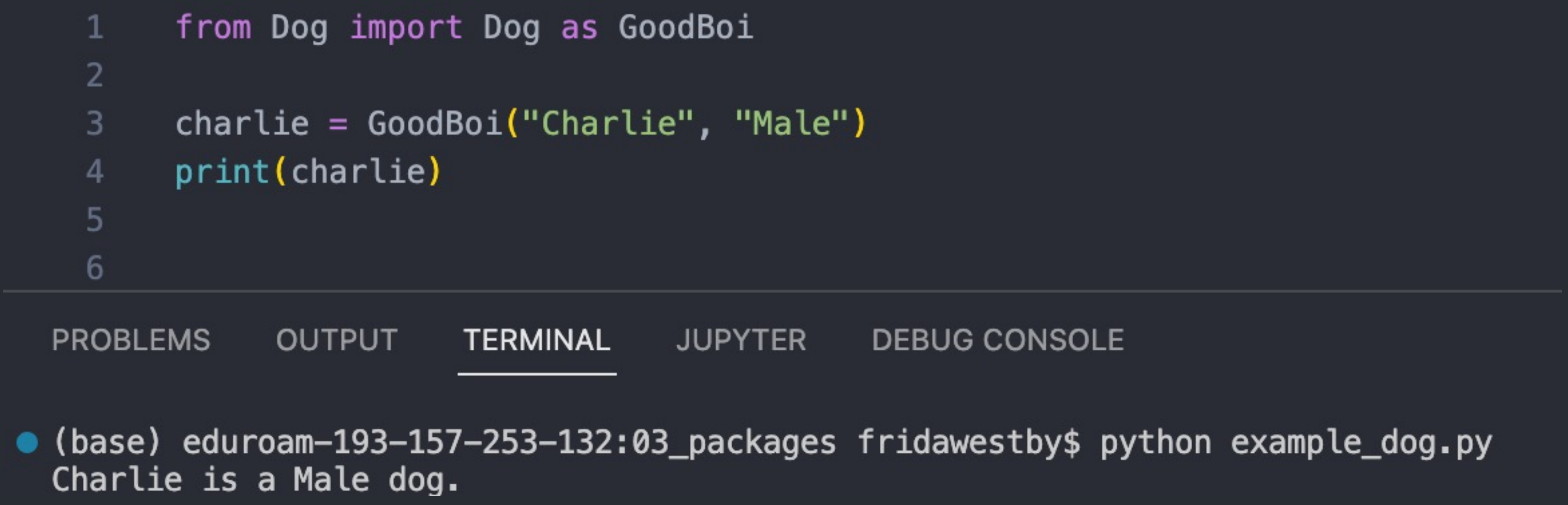


```python
import Dog as GoodBoi

charlie = GoodBoi("Charlie", "Male")
print(charlie)


```

```
(base) eduroam-193-157-253-132:03_packages fridawestby$ python example_dog.py
Traceback (most recent call last):
  File "example_dog.py", line 3, in <module>
    charlie = GoodBoi("Charlie", "Male")
TypeError: 'module' object is not callable
```

Isn´t Dog a module?

- Yes, it is!
- BUT Python could not find it ☹
- So we have to tell it where to find it:



```python
from Dog import Dog as GoodBoi

charlie = GoodBoi("Charlie", "Male")
print(charlie)
```

```
(base) eduroam-193-157-253-132:03_packages fridawestby$ python example_dog.py
Charlie is a Male dog.
```

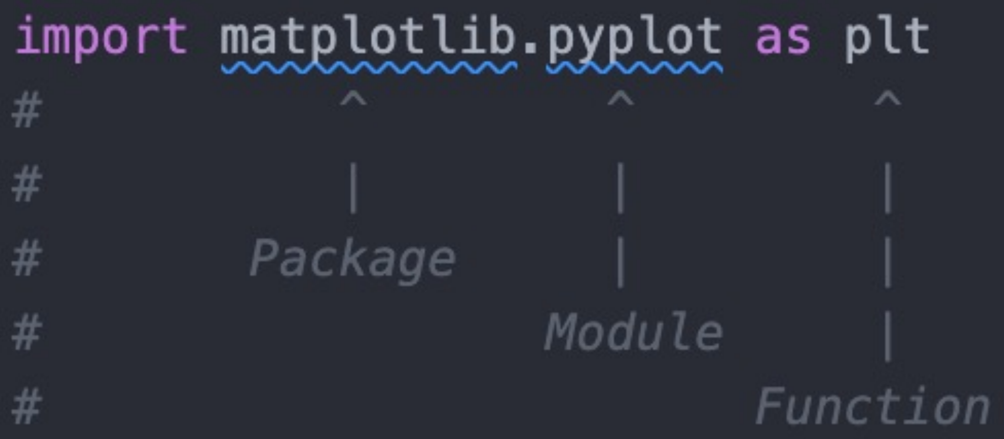
Python package

- Packages are a way of structuring Python’s module namespace by using “dotted module names”.
- For example, the module name matplotlib.pyplot designates a submodule named pyplot in a package named matplotlib.
- Just like the use of modules saves the authors of different modules from having to worry about each other’s global variable names, the use of dotted module names saves the authors of multi-module packages like NumPy or Pillow from having to worry about each other’s module names.

https://docs.python.org/3/tutorial/modules.html

Why Packages?

- Packages allow to organize modules and scripts into single environment
- These can then easily be distributed and imported by name



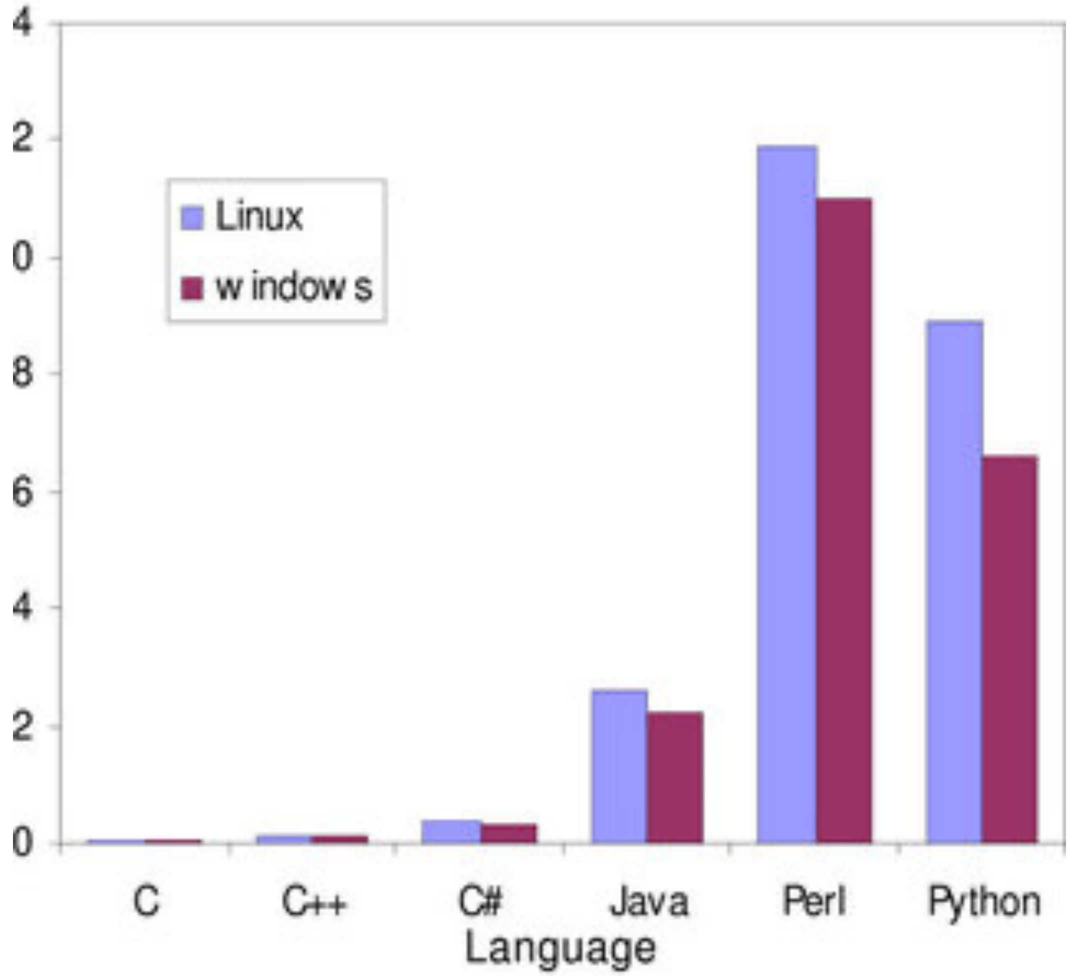
```
import matplotlib.pyplot as plt
#            ^          ^         ^
#            |          |         |
#         Package       |         |
#                    Module       |
#                              Function
```

Example: NumPy

- NumPy is a package for Python
- The name is an acronym for "Numeric Python" or "Numerical Python"
- It mostly written in C

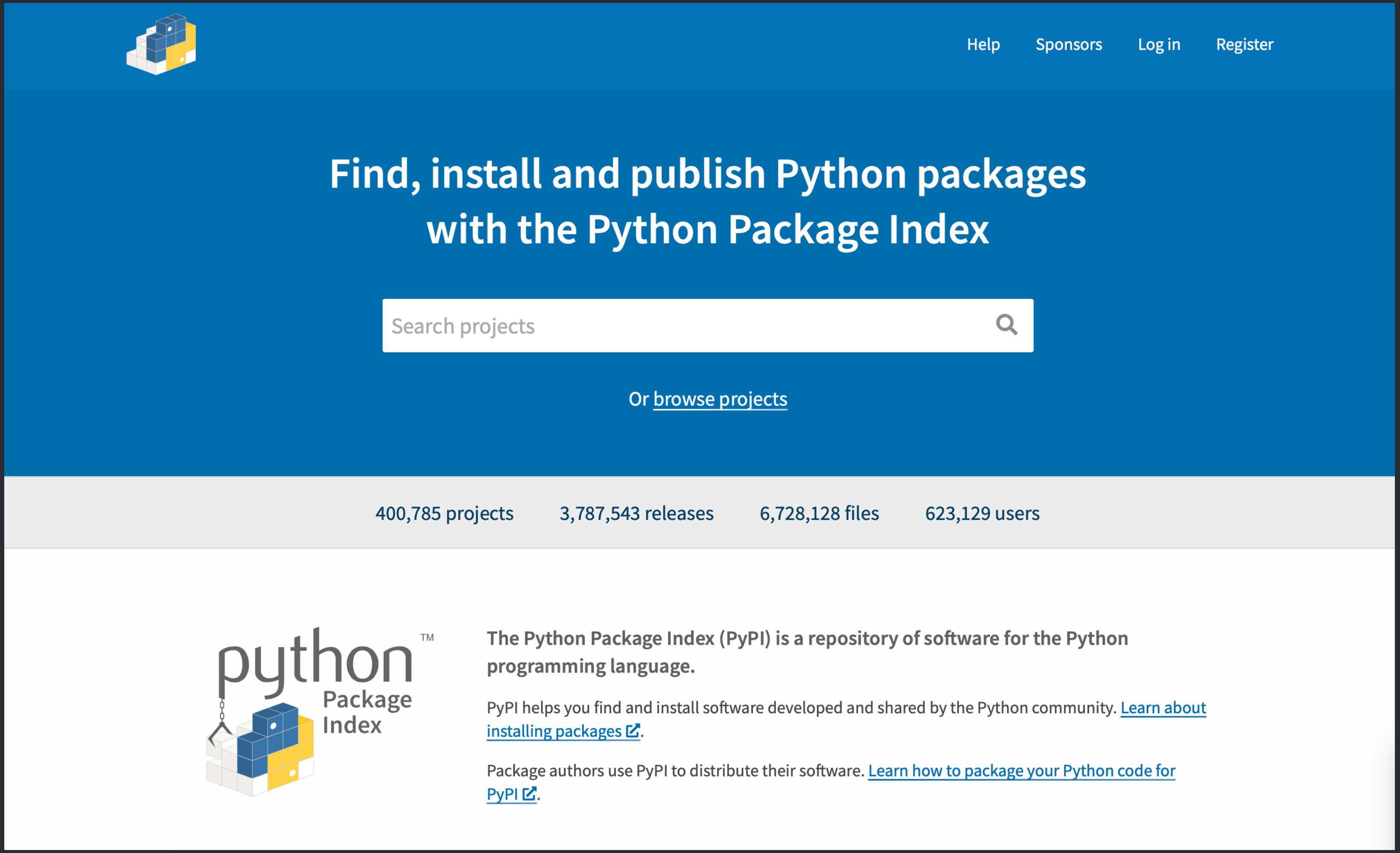
```python
import numpy as np

import numpy
# Access pi variable from the NumPy module
numpy.pi

# Access pi variable from the NumPy module with alias
import numpy as np
np.pi

# Access pi variable from the NumPy with bad practice
from numpy import *
# Just don´t
```

Why C?



Speed comparing for different programming languages (lower is better!)

Python Package Index (PyPI)



Find, install and publish Python packages with the Python Package Index

400,785 projects 3,787,543 releases 6,728,128 files 623,129 users

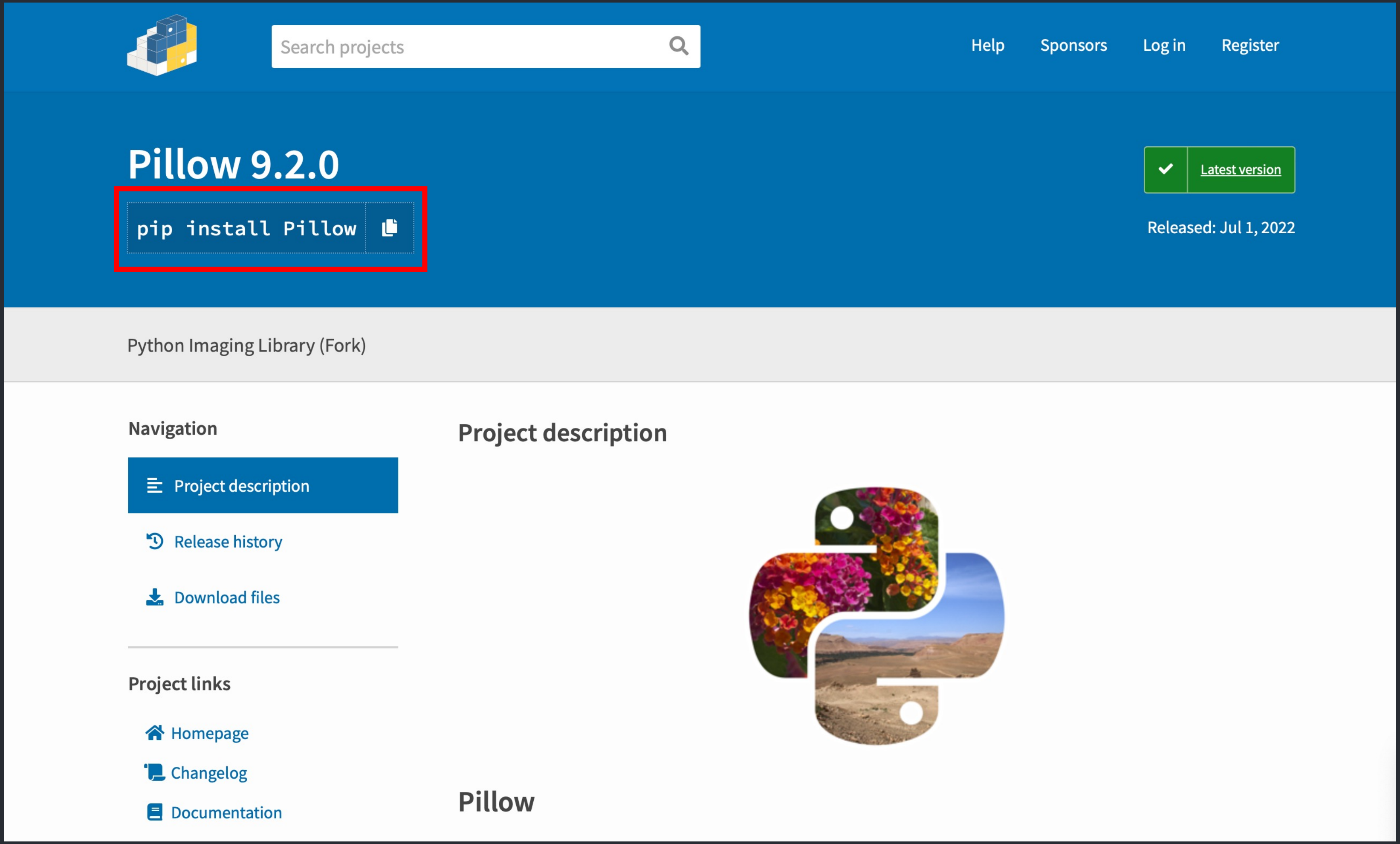
The Python Package Index (PyPI) is a repository of software for the Python programming language.

PyPI helps you find and install software developed and shared by the Python community. Learn about installing packages.

Package authors use PyPI to distribute their software. Learn how to package your Python code for PyPI.

https://pypi.org

Python Package Index (PyPI)



Search projects

Help Sponsors Log in Register

Pillow 9.2.0

```
pip install Pillow
```

Latest version

Released: Jul 1, 2022

Python Imaging Library (Fork)

Navigation

- Project description
- Release history
- Download files

Project links

- Homepage
- Changelog
- Documentation

Project description

Pillow

https://pypi.org/project/Pillow/

pip

- pip is a package manager for Python packages

```
pip --version  # pip version installed
pip --upgrade <package_name>  # update package
pip list  # list installed packages
pip list --outdated  # list outdated packages
pip install <package_name>  # install <package_name>
pip install <package_name>==<version>
pip uninstall <package_name>  # uninstall <package_name>
```

Make package editeble

- Install your project in “editable” or “develop” mode while you’re working on it.
- This will just link the package to the original location, basically meaning any changes to the original package would reflect directly in your environment.

```
pip install -e .
```

# install the current directory (i.e. your project) in editable mode.

SETUPTOOLS

- Starting with **PEP 621** (22-Jun-2020), the Python community selected pyproject.toml as a standard way of specifying *project metadata*.
- Setuptools is a fully-featured, actively-maintained, and stable library designed to facilitate packaging Python projects.
- It helps developers to easily share reusable code (in the form of a library) and programs (e.g., CLI/GUI tools implemented in Python), that can be installed with pip and uploaded to PyPI.

```toml
[build-system]
requires = ["setuptools", "setuptools-scm"]
build-backend = "setuptools.build_meta"

[project]
name = "my_package"
description = "My package description"
readme = "README.rst"
requires-python = ">=3.7"
keywords = ["one", "two"]
license = {text = "BSD 3-Clause License"}
classifiers = [
    "Framework :: Django",
    "Programming Language :: Python :: 3",
]
dependencies = [
    "requests",
    'importlib-metadata; python_version<"3.8"',
]
dynamic = ["version"]

[project.optional-dependencies]
pdf = ["ReportLab>=1.2", "RXP"]
rest = ["docutils>=0.3", "pack ==1.1, ==1.3"]

[project.scripts]
my-script = "my_package.module:function"
```

This section declares what are your build system dependencies, and which library will be used to actually do the packaging

```
[build-system]
requires = ["setuptools", "setuptools-scm"]
build-backend = "setuptools.build_meta"
```

This section is where the package configuration happens

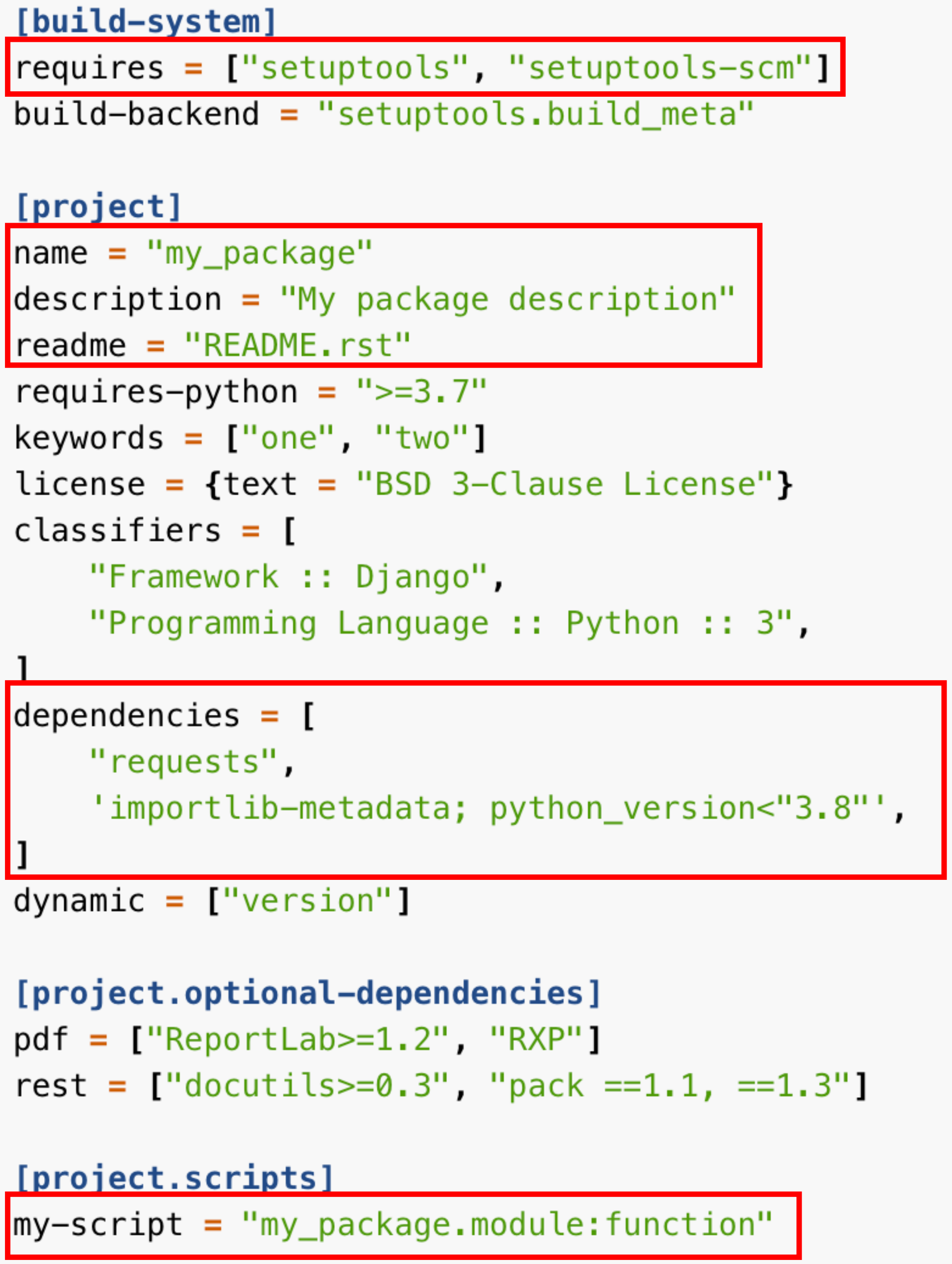
```
[project]
name = "my_package"
description = "My package description"
readme = "README.rst"
requires-python = ">=3.7"
keywords = ["one", "two"]
license = {text = "BSD 3-Clause License"}
classifiers = [
    "Framework :: Django",
    "Programming Language :: Python :: 3",
]
dependencies = [
    "requests",
    'importlib-metadata; python_version<"3.8"',
]
dynamic = ["version"]
```

This section is optional

```
[project.optional-dependencies]
pdf = ["ReportLab>=1.2", "RXP"]
rest = ["docutils>=0.3", "pack ==1.1, ==1.3"]
```

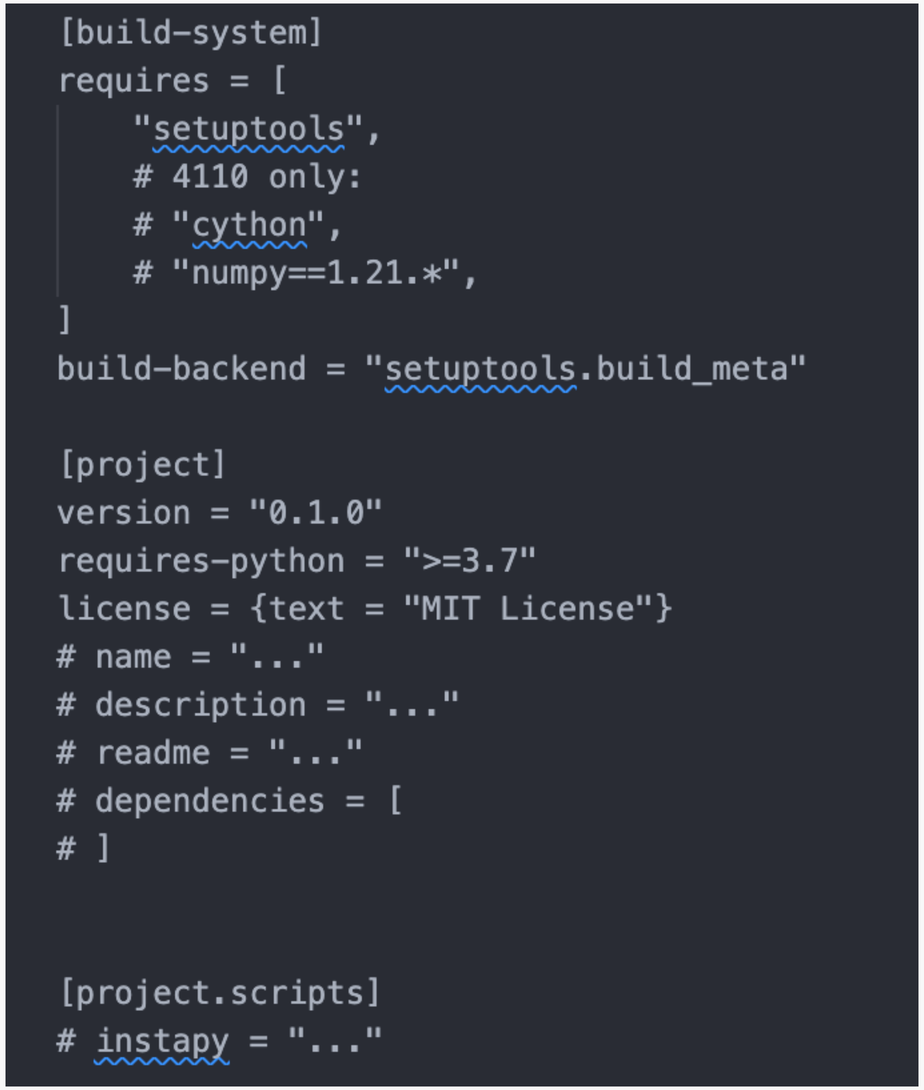
When this project is installed, a cli-name executable will be created. cli-name will invoke the function some_func in the mypkg/mymodule.py file when called by the user. E.g. dog-script = ”dog.cli:main"

```
[project.scripts]
my-script = "my_package.module:function"
```



```toml
[build-system]
requires = ["setuptools", "setuptools-scm"]
build-backend = "setuptools.build_meta"

[project]
name = "my_package"
description = "My package description"
readme = "README.rst"
requires-python = ">=3.7"
keywords = ["one", "two"]
license = {text = "BSD 3-Clause License"}
classifiers = [
    "Framework :: Django",
    "Programming Language :: Python :: 3",
]
dependencies = [
    "requests",
    'importlib-metadata; python_version<"3.8"',
]
dynamic = ["version"]

[project.optional-dependencies]
pdf = ["ReportLab>=1.2", "RXP"]
rest = ["docutils>=0.3", "pack ==1.1, ==1.3"]

[project.scripts]
my-script = "my_package.module:function"
```

What you need for Assignment 3 ☺



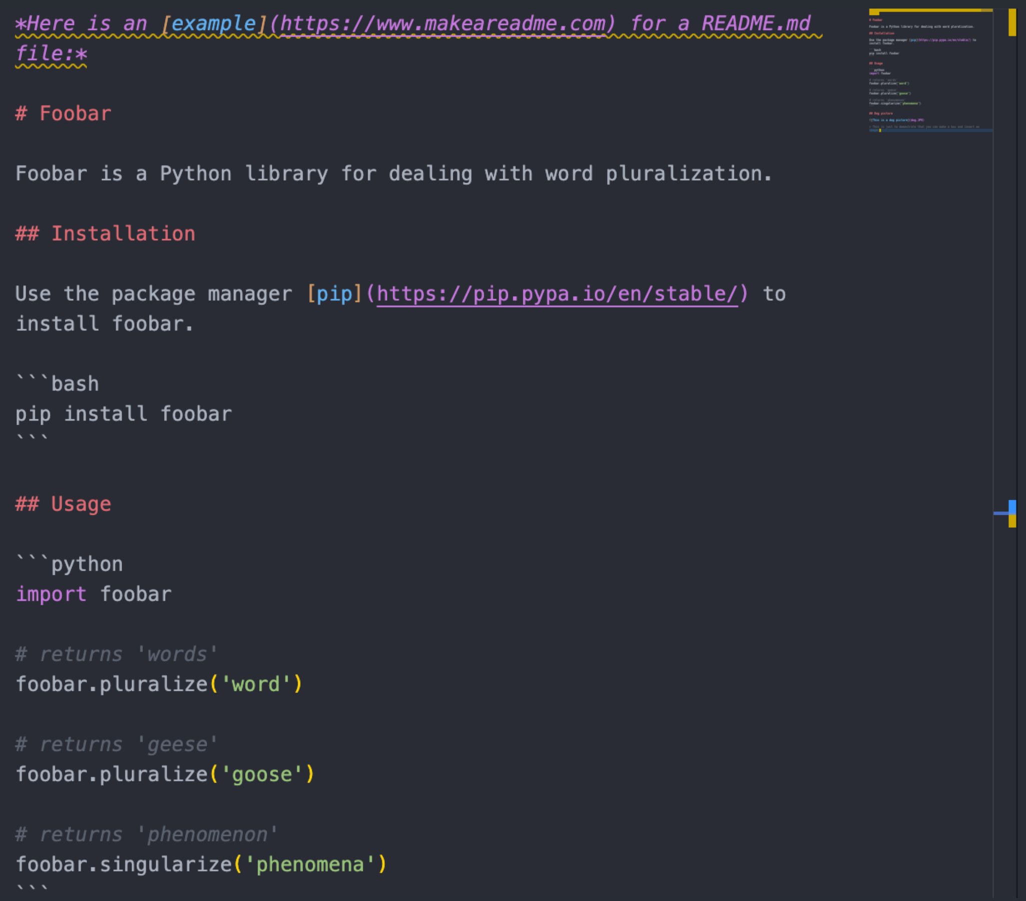
```toml
[build-system]
requires = [
    "setuptools",
    # 4110 only:
    # "cython",
    # "numpy==1.21.*",
]
build-backend = "setuptools.build_meta"

[project]
version = "0.1.0"
requires-python = ">=3.7"
license = {text = "MIT License"}
# name = "..."
# description = "..."
# readme = "..."
# dependencies = [
# ]


[project.scripts]
# instapy = "..."
```

`build-system.requires` vs `project.dependencies`

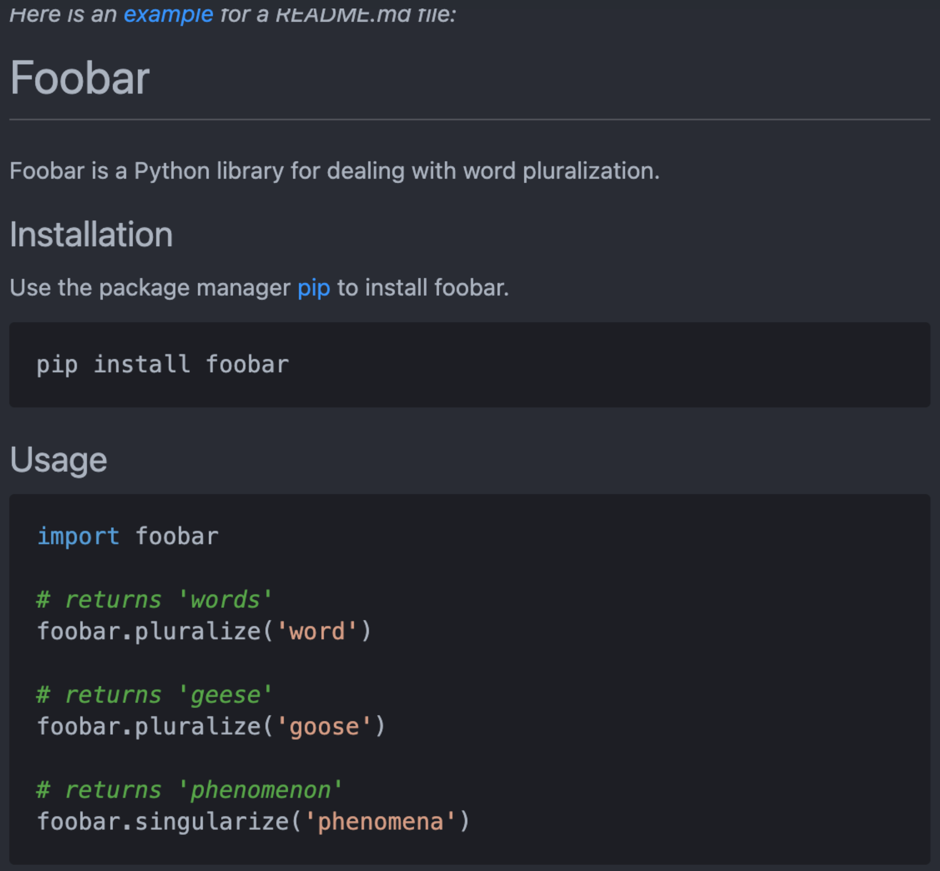
- `build-system.requires` is what’s required to build the package from source and install it. Effectively, these are the dependencies of `setup.py`.
  - These are not required when you are using the package.
  - Most users of your package will not get these packages when you publish a package on PyPI.

- `project.dependencies` are required when your package is actually used.
  - These are the dependencies (I.e. imports) of instapy.

How to make a more readable README.md



~~~markdown
*Here is an [example](https://www.makeareadme.com) for a README.md
file:*

# Foobar

Foobar is a Python library for dealing with word pluralization.

## Installation

Use the package manager [pip](https://pip.pypa.io/en/stable/) to
install foobar.

```bash
pip install foobar
```

## Usage

```python
import foobar

# returns 'words'
foobar.pluralize('word')

# returns 'geese'
foobar.pluralize('goose')

# returns 'phenomenon'
foobar.singularize('phenomena')
```
~~~



Here is an example for a README.md file:

Foobar

Foobar is a Python library for dealing with word pluralization.

Installation

Use the package manager pip to install foobar.

```
pip install foobar
```

Usage

```python
import foobar

# returns 'words'
foobar.pluralize('word')

# returns 'geese'
foobar.pluralize('goose')

# returns 'phenomenon'
foobar.singularize('phenomena')
```