A whirlwind tour of SETUPTOOLS

“big ideas”

- Goal is to give you a roadmap, not a comprehensive understanding
- `setuptools` User Guide is fantastic (if not thrilling) reading: https://setuptools.pypa.io/en/latest/userguide/index.html
- `setuptools` is a tool for building distributable packages for Python software
  - Including metadata about dependencies, etc.
- Can be used in imperative or declarative styles
- Capable of building components written in not-Python

“small ideas”

- `setuptools` is a tool that ‘eats’ metadata and spits out an installable package `.whl` or `.tar.gz`
- `setuptools` is one tool among many

setuptools - overview

- The `setuptools` user fills in two distinct types of information:
- Project metadata
- Build-time executable code

`setuptools` - overview

- Two types of information:
- Project metadata
  - `setup()` function in `setup.py`, OR static data in `setup.cfg`, `pyproject.toml`
  - The name of the distribution (what you `pip install`), version number, dependencies, license, etc.
- Build-time executable code

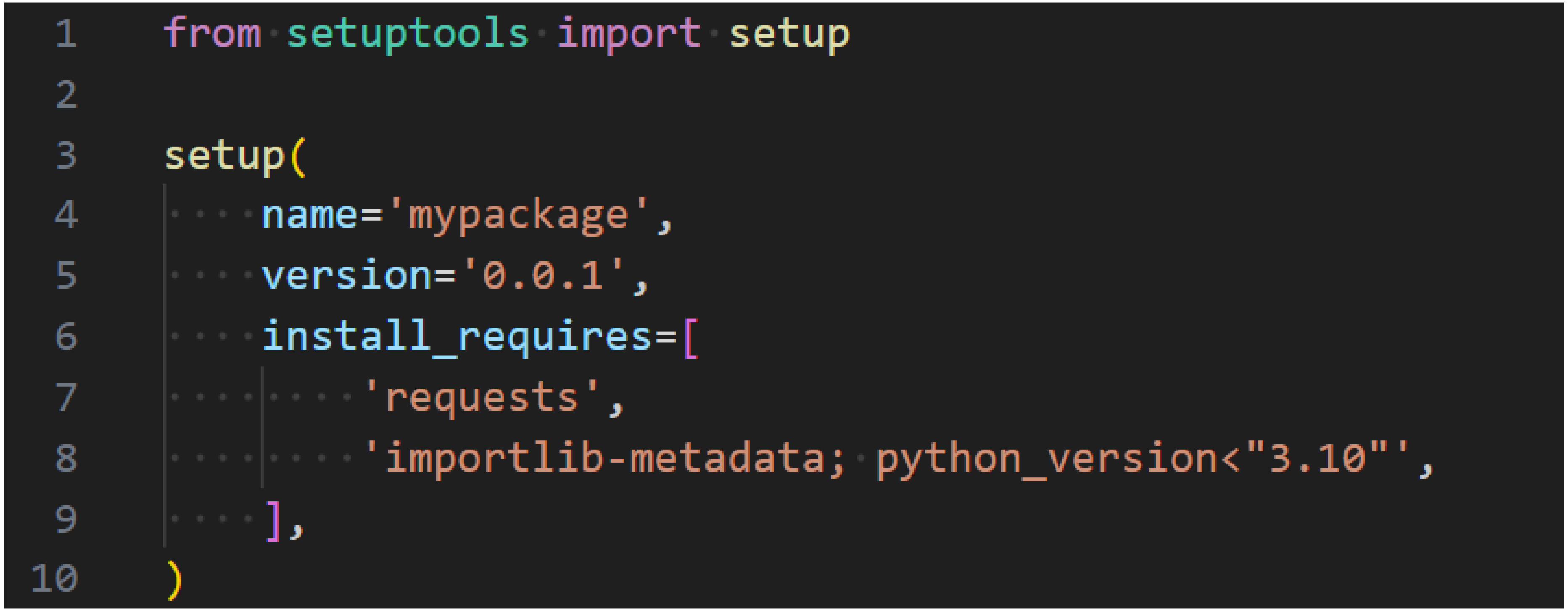
`setuptools` - overview

- Two types of information:
- Project metadata
- Build-time executable code
  - Does any number of things at (<u>only</u>) build time
  - Truly arbitrary! Powerful, but easy to write obscure things
  - Exclusively in `setup.py` or things run by it

`setuptools` - overview

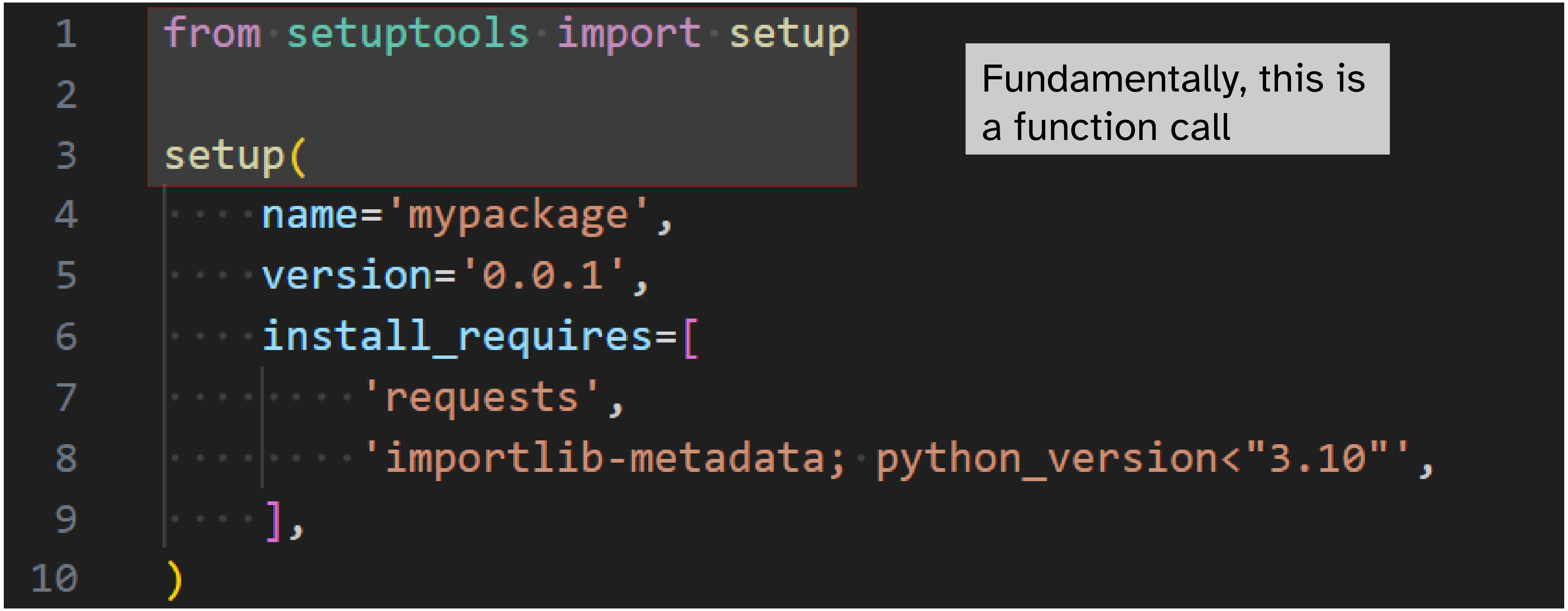
- Important note: once upon a time, users ran `setup.py` directly, e.g.
  - `python3 setup.py install`
  - `python3 setup.py develop`
- **New software SHOULD NOT do this!**
  - You can probably use `pip` or `build` instead

- Anatomy of a `setup.py`



```python
from setuptools import setup

setup(
    name='mypackage',
    version='0.0.1',
    install_requires=[
        'requests',
        'importlib-metadata; python_version<"3.10"',
    ],
)
```

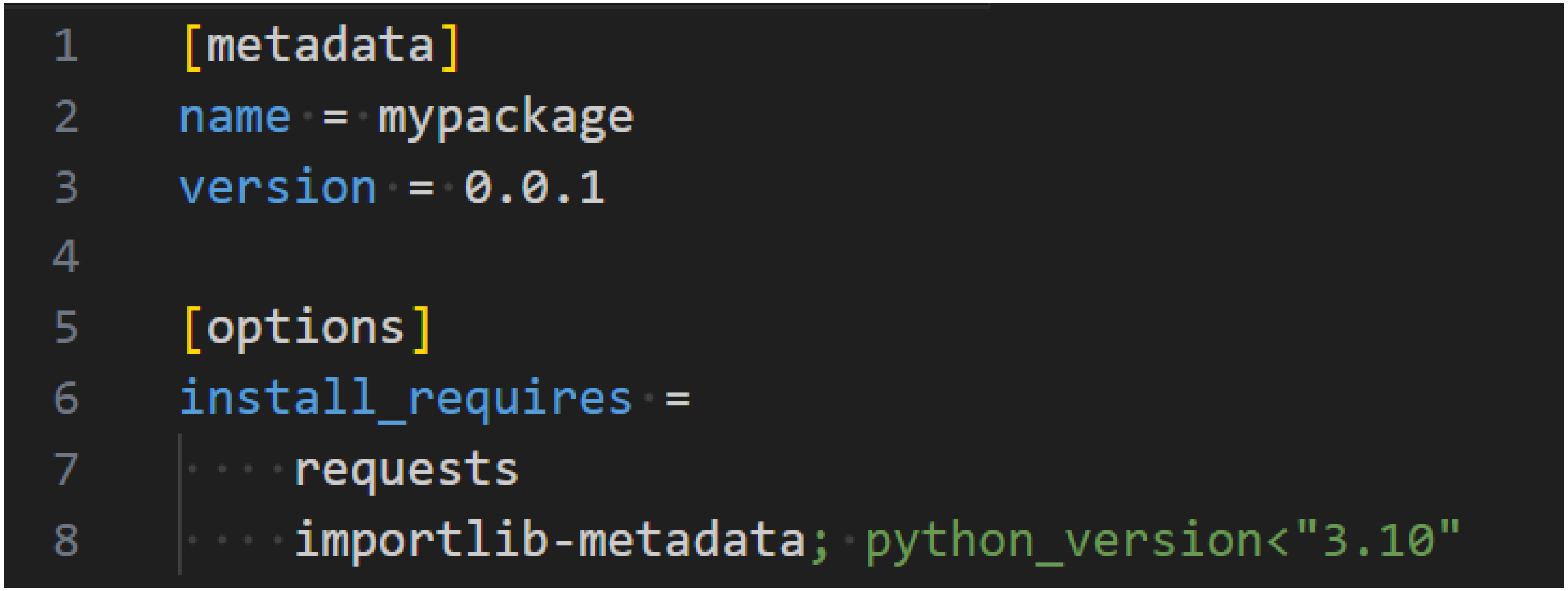
- Anatomy of a `setup.py`



```python
from setuptools import setup

setup(
    name='mypackage',
    version='0.0.1',
    install_requires=[
        'requests',
        'importlib-metadata; python_version<"3.10"',
    ],
)
```

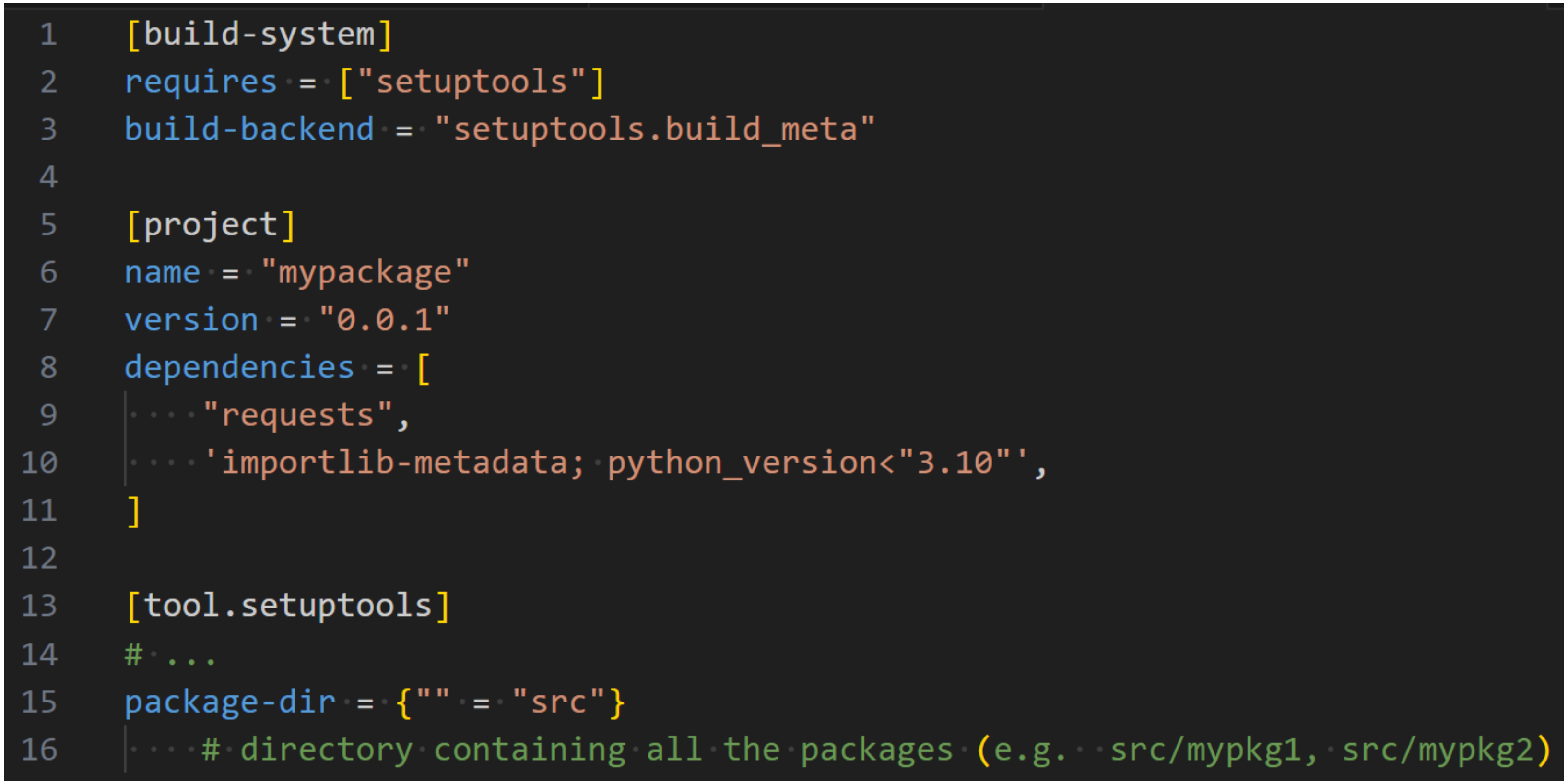
Fundamentally, this is a function call

- Anatomy of a `setup.cfg`



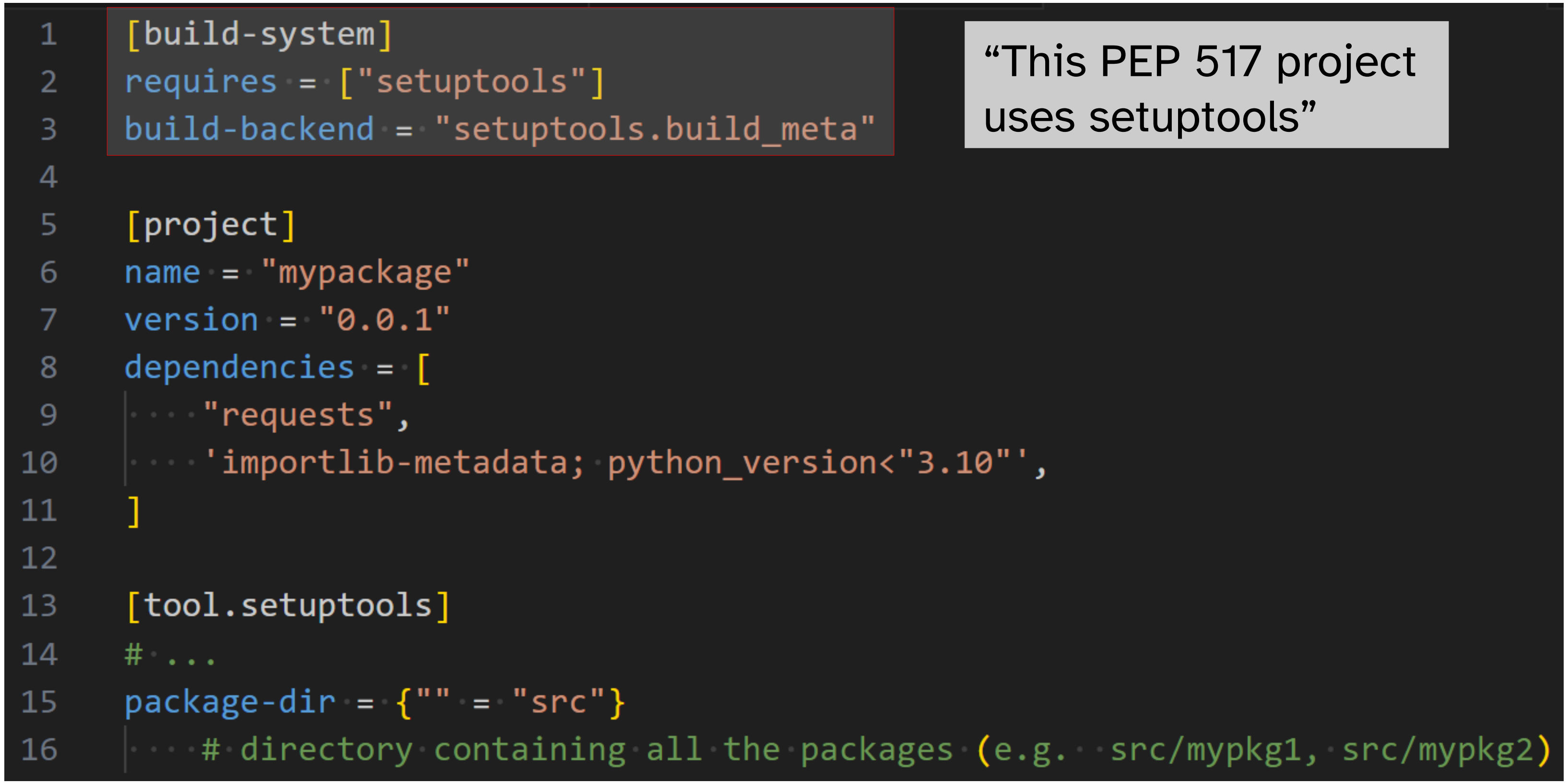
```
[metadata]
name = mypackage
version = 0.0.1

[options]
install_requires =
    requests
    importlib-metadata; python_version<"3.10"
```

- Anatomy of a `pyproject.toml`



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

[project]
name = "mypackage"
version = "0.0.1"
dependencies = [
    "requests",
    'importlib-metadata; python_version<"3.10"',
]

[tool.setuptools]
# ...
package-dir = {"" = "src"}
    # directory containing all the packages (e.g.  src/mypkg1, src/mypkg2)
```

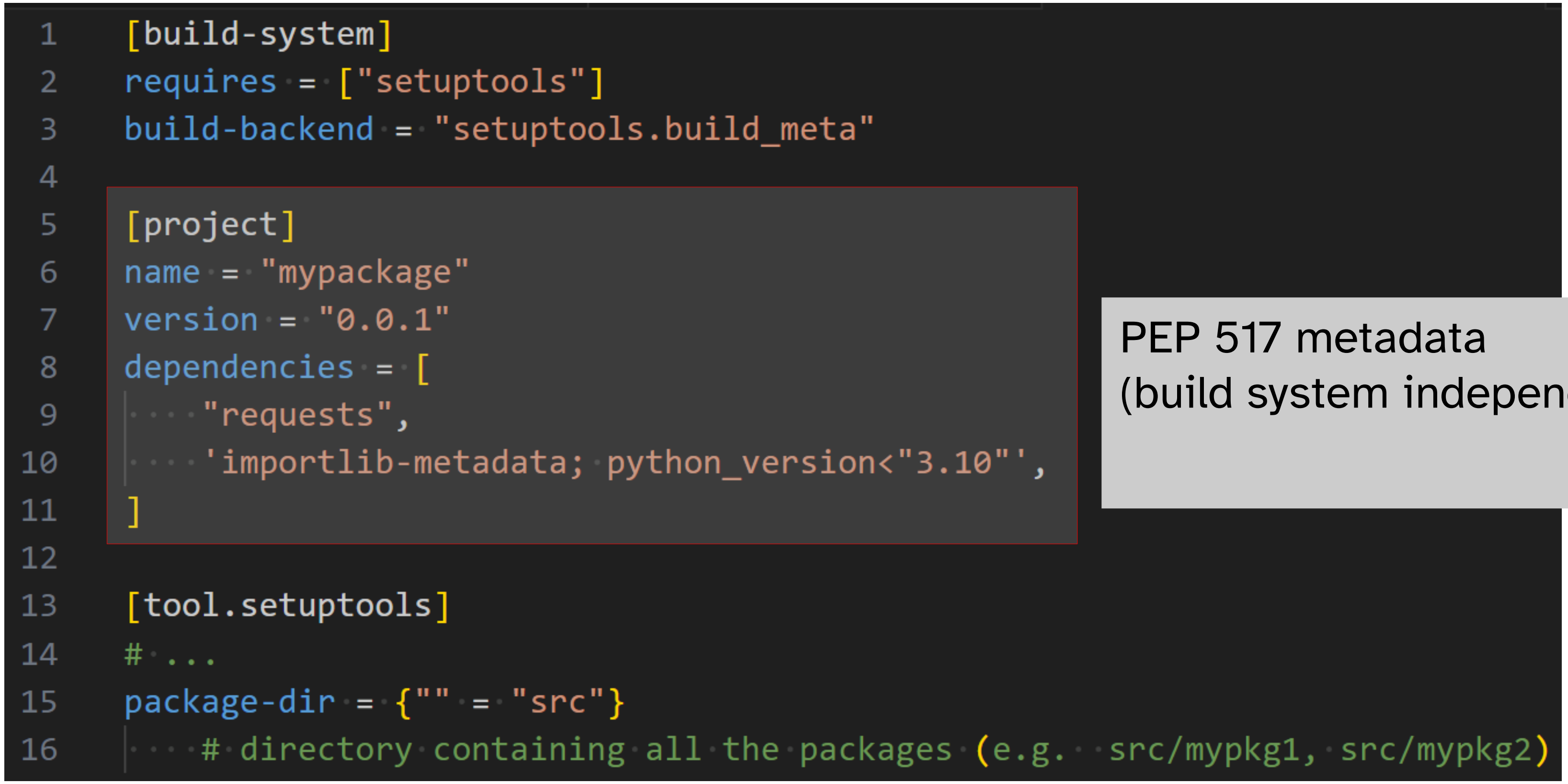
- Anatomy of a `pyproject.toml`



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

[project]
name = "mypackage"
version = "0.0.1"
dependencies = [
    "requests",
    'importlib-metadata; python_version<"3.10"',
]

[tool.setuptools]
# ...
package-dir = {"" = "src"}
    # directory containing all the packages (e.g.  src/mypkg1, src/mypkg2)
```

"This PEP 517 project uses setuptools"

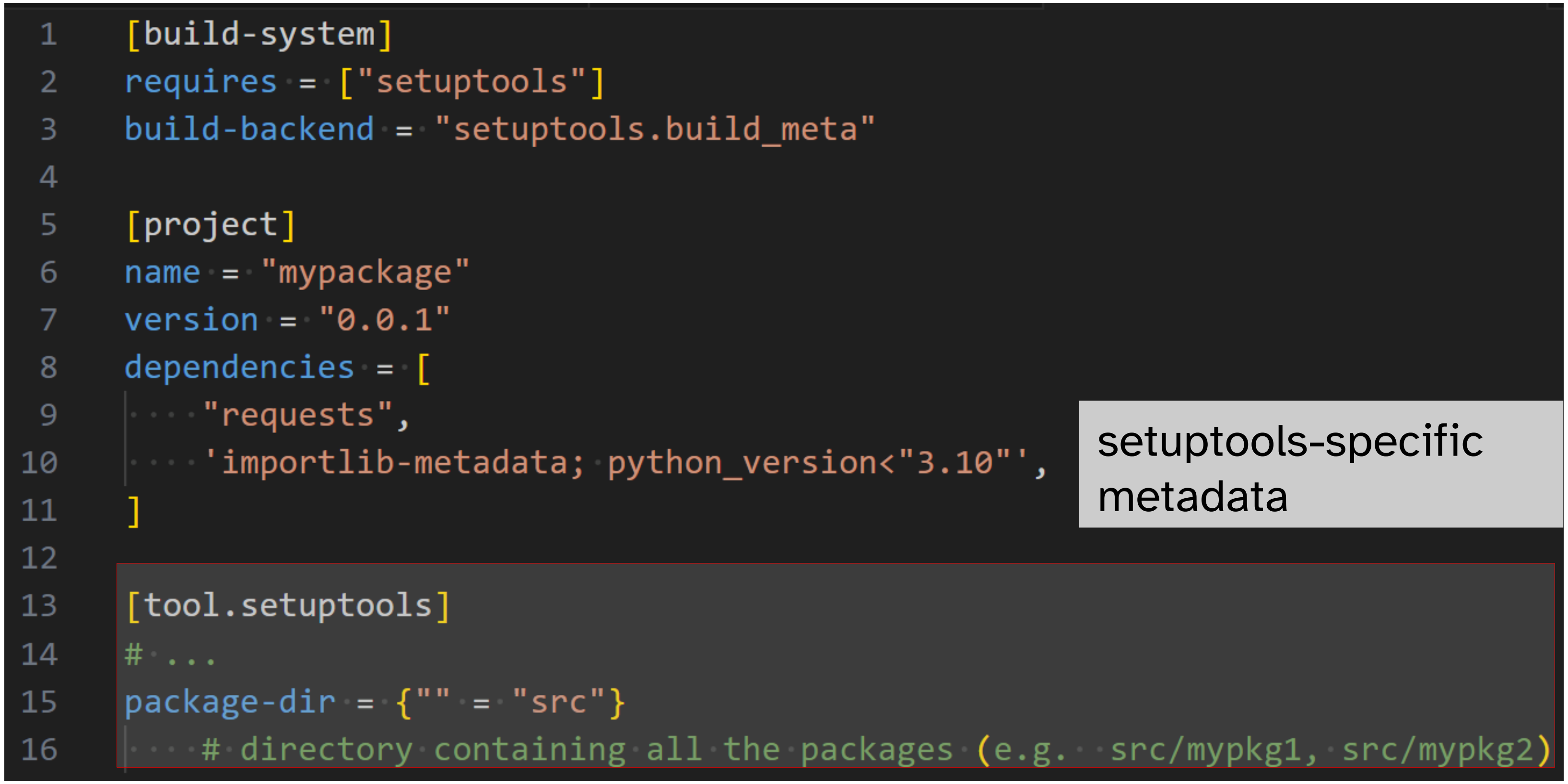
- Anatomy of a `pyproject.toml`



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

[project]
name = "mypackage"
version = "0.0.1"
dependencies = [
    "requests",
    'importlib-metadata; python_version<"3.10"',
]

[tool.setuptools]
# ...
package-dir = {"" = "src"}
    # directory containing all the packages (e.g.  src/mypkg1, src/mypkg2)
```

PEP 517 metadata
(build system independent)

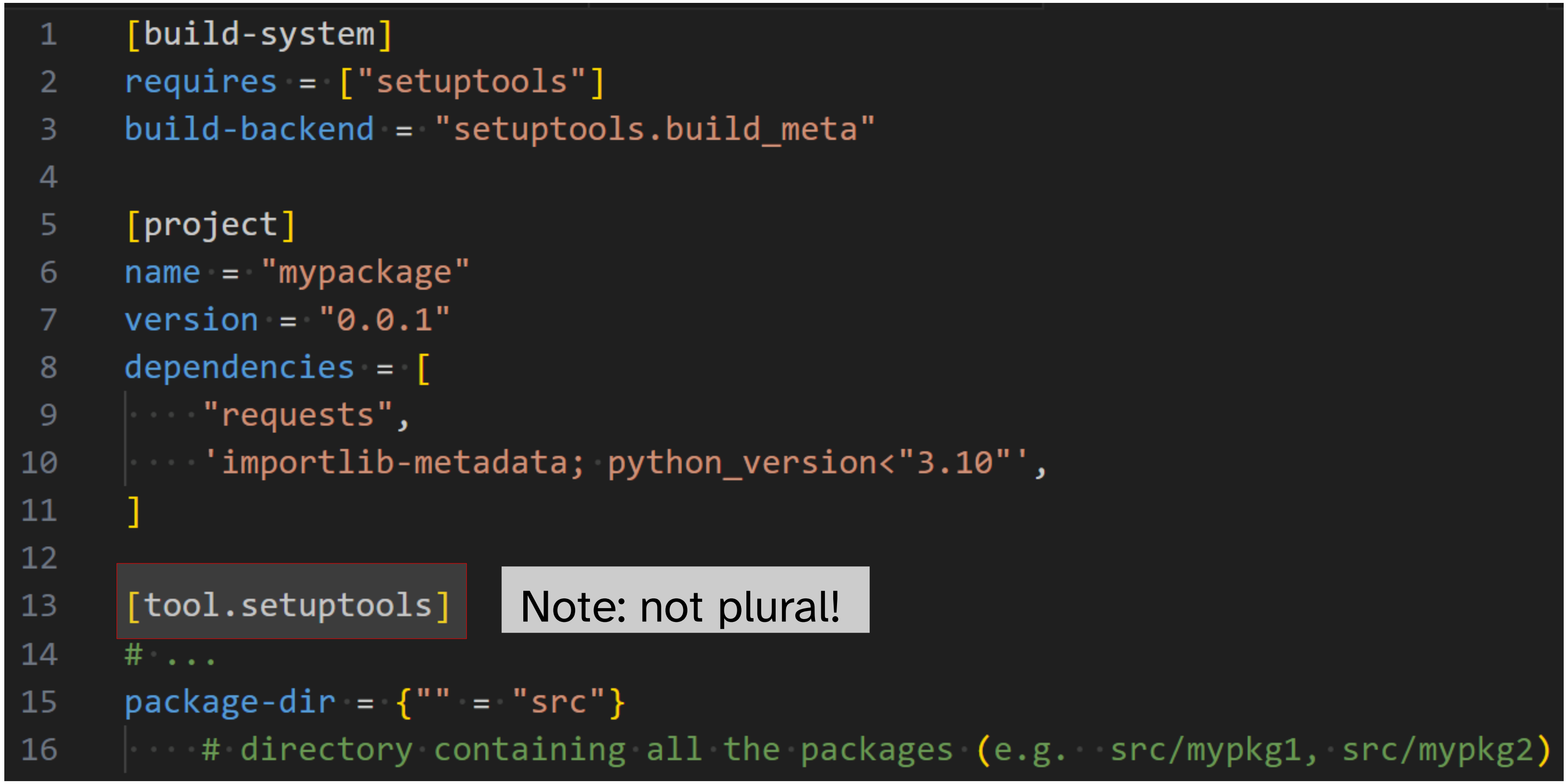
- Anatomy of a `pyproject.toml`



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

[project]
name = "mypackage"
version = "0.0.1"
dependencies = [
    "requests",
    'importlib-metadata; python_version<"3.10"',
]

[tool.setuptools]
# ...
package-dir = {"" = "src"}
    # directory containing all the packages (e.g.  src/mypkg1, src/mypkg2)
```

setuptools-specific metadata

- Anatomy of a `pyproject.toml`



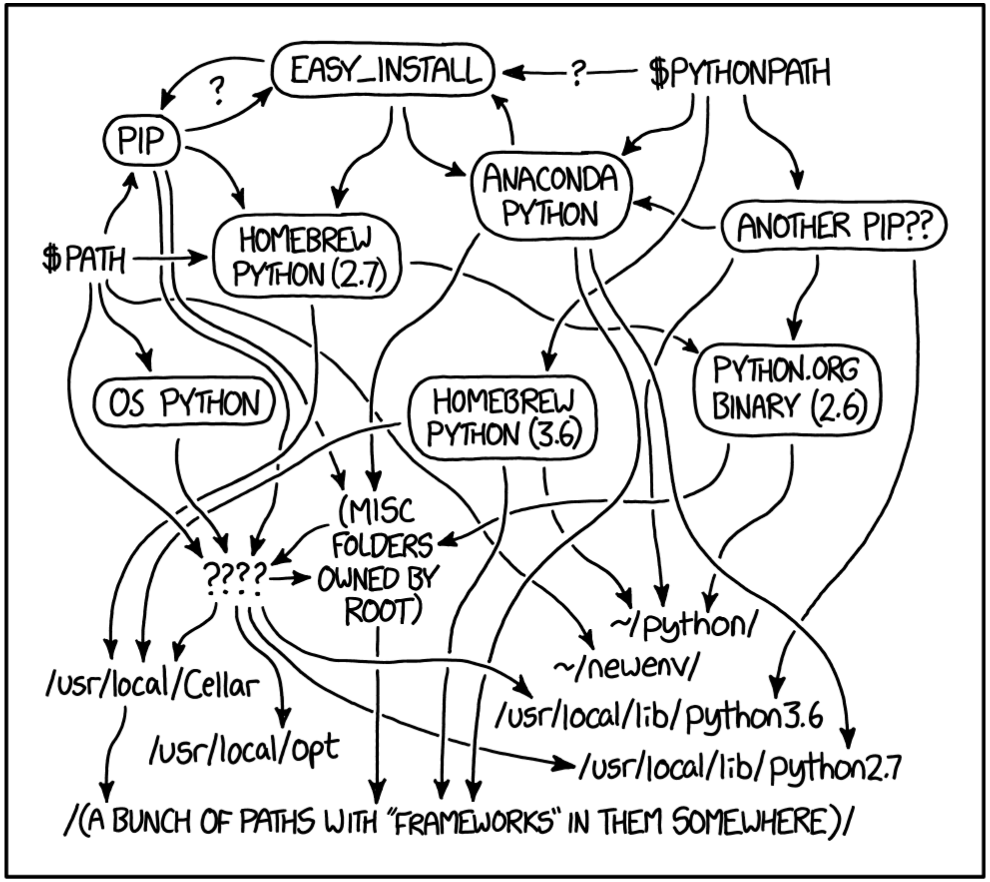
```toml
[build-system]
requires = ["setuptools"]
build-backend = "setuptools.build_meta"

[project]
name = "mypackage"
version = "0.0.1"
dependencies = [
    "requests",
    'importlib-metadata; python_version<"3.10"',
]

[tool.setuptools]
# ...
package-dir = {"" = "src"}
    # directory containing all the packages (e.g.  src/mypkg1, src/mypkg2)
```

Note: not plural!

`setuptools` metadata fields

- The gimmes:
  - `name, version, author, description, install_requires`
- Package discovery:
  - `packages, package_dir, package_data`
- The weirdos:
  - `cmdclass, ext_modules`
- https://setuptools.pypa.io/en/latest/references/keywords.html

Intermezzo:
why so many ways to do it?!



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

A brief history of Python packaging

- Ancient pre-history:
  - No formal concept of packaging!
- A bunch of `.py` goes from `[HERE]` to `[THERE]`
- What if we could formalize that

A brief history of Python packaging

- `distutils` is born, quickly added to the stdlib
- `setup.py, setup.cfg` come into existence here
- The concept of a "distribution" formalizes "a bunch of `.py`"
    - But some problems are left unsolved, many of them about package metadata

A brief history of Python packaging

- `setuptools` is born
- Based on `distutils`, but substantially more capable
  - Package metadata!
  - Extension modules! (i.e. written in not-Python)
- Not part of stdlib, but this allows it to move quickly
  - Eventually, it even absorbed `distutils`

A brief history of Python packaging

- Concurrently, tools for ‘installing’ and otherwise juggling distributions also come into existence
  - `easy_install`
  - `wheel`
  - `pip`
- These influence and are influenced by `setuptools`

A brief history of Python packaging

- For a ‘long’ time, `setuptools` is the way things get done
- Desire for some more flexibility leads to PEP 517/518
  - Formal division between building distributions (“build backend”) and installing them (“integration frontend”)
- `setuptools` is compatible with the modern way, it is **not** obsolete as is often claimed
  - “Reports of my death are greatly exaggerated” - `setuptools`

A taste of advanced `setuptools`

Advanced `setuptools`

- Loading requirements from a file
- https://github.com/nedbat/scriv/blob/603f8e760ca4a2ab6011c02f3b5cc6dcaaf8c7dc/setup.py#L72

Advanced `setuptools`

- Building an extension module
- https://git.snoopj.dev/SnoopJ/unicodedata2/src/commit/a7ef92c6dbffb5a3bfe198156e5a924d476880eb/setup.py#L21-L25

Advanced `setuptools`

- Custom `cmdclass`
- https://github.com/numpy/numpy/blob/4adc87dff15a247e417d50f10cc4def8e1c17a03/setup.py#L389

Thank you!

Questions?