How to Package and Publish Your Python Code

Winter 2026 Research Computing Workshops

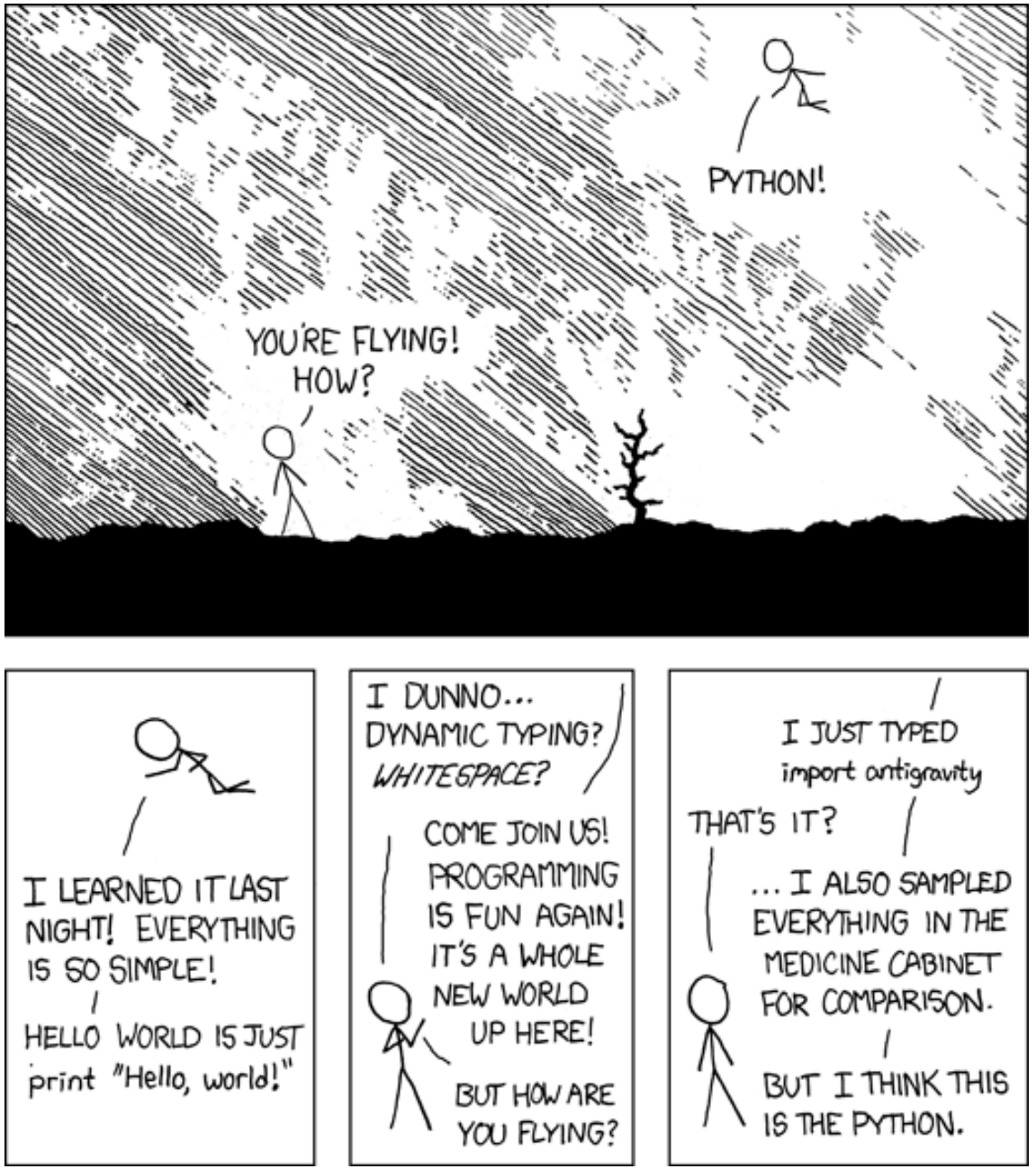
Presented by Giannis Paraskevakos Princeton RSE II

Outline

- Python Packages
- Using a python package
- How to create a python package
  - Organization
  - Hands-on
- Summary

Python Packages: What are they & why do we care

- A reusable way to share code
- Dramatically accelerate development
- Make large projects manageable



https://xkcd.com/353/

How do we use a python package

- We install the package by using tools such as `pip`, `conda`, `uv`, `hatch`, etc.
  - Installed in the loaded environment
- Best use a virtual environment
  - Separates system and user environment
  - Provides reproducible environments
- After installation is complete, import package

Build systems

Setuptools	Poerty	Hatch/ hatchling	meson-python	scikit-build-core	uv
Industry standard	Modern	Modern build backend	Very fast builds, Esp. C/C++ ext.	Modern Cmake intergation	Extremely fast
pure Python and extension modules	Integrated env management	Both python and extensions	Good for complex projects	Good for C/C++ extensions	All-in-one tool
setup.py, setup.cfg, and pyproject.toml	Only pyproject.toml	Integrated environment management	Used by Numpy and SciPy	Requires Cmake knowledge	Grows rapidly
Multiversion	Offers lockfile	Small community	Need to learn meson	Complex for simple projects	Breaking changes possibles
Slow and complex	Can be slow	Still evolving	Maybe an overkill	Still evolving0	Features are still added

What will we use

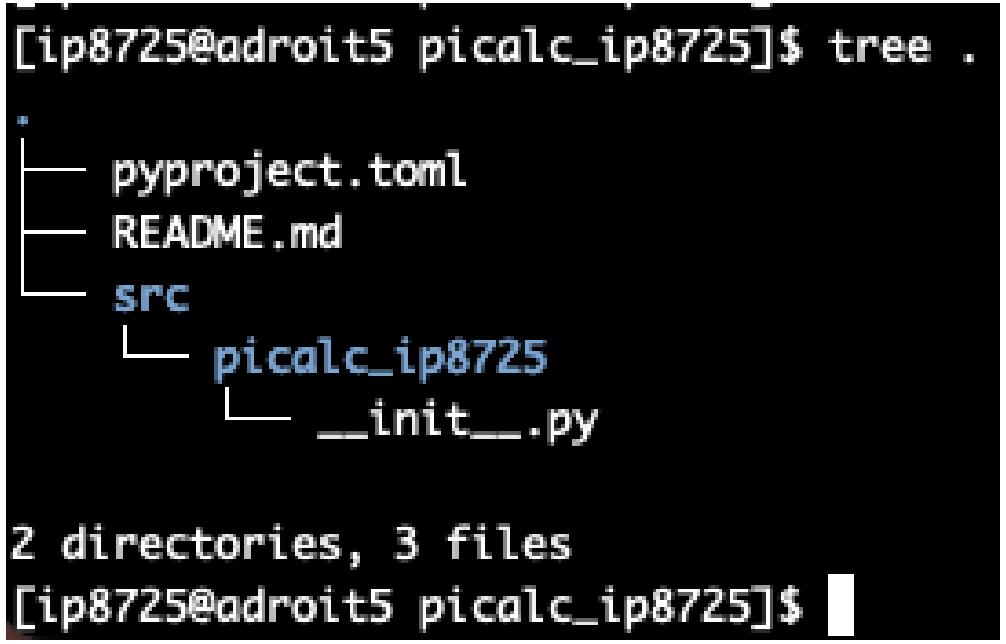
- Github
  - Please create a github project named picalc_[netid]
- Uv
- Adroit
- https://test.pypi.org
  - Please create an account
  - Please create an API key
  - Copy the key and save it in a text document for now

How to create a python package

- `uv` can be installed easily on nearly any system and doesn't rely on python
  - https://docs.astral.sh/uv/getting-started/installation/
- Clone your empty repo first
- The following command will create a new directory called `picalc_ip8725` for our package
  - `uv init --package picalc_ip8725 -build_backend setuptools`

How to create a python package

- pyproject.toml file
- Specific folder structure
- Tools like `uv` does it automatically




```
[build-system]
requires = ["setuptools>=61.0.0"]
build-backend =
"setuptools.build_meta"

[project]
name = " picalc_ip8725"
version = "0.1.0"
description = "A short description
of my project"
readme = "README.md"
requires-python = ">=3.8"
```

Adding depencencies

- Through uv
- By changing pyproject.toml
- Optional dependencies
  - We will use later

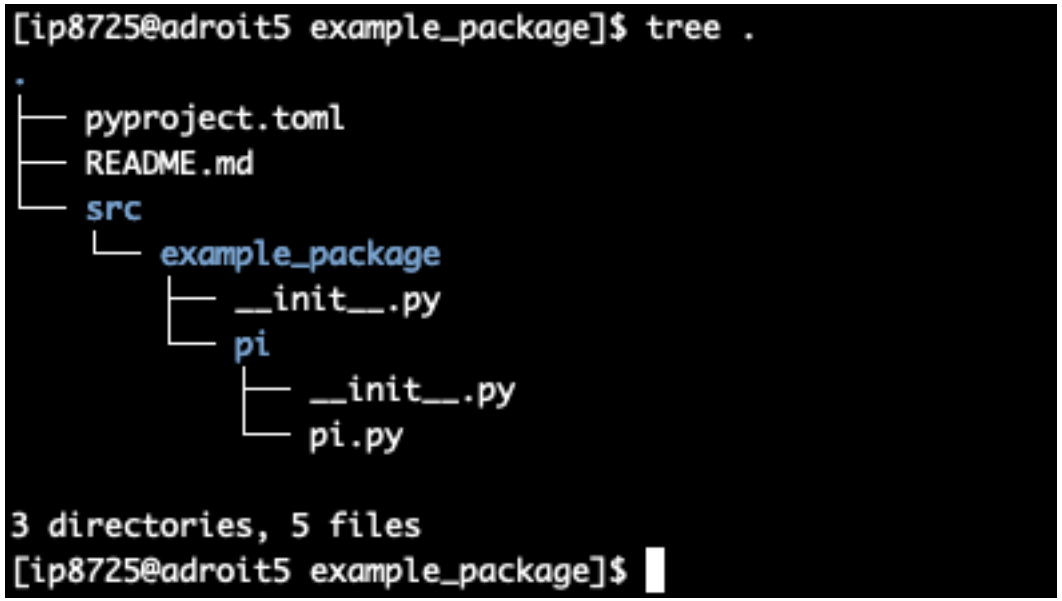
```
[project]
name = "picalc-ip8725”
version = "0.1.0"
description = "Add your description here"
readme = "README.md"
requires-python = ">=3.9"
dependencies = [
    "numpy>=2.0.2",
]
[project.scripts]
example-package = "example_package:main"
[build-system]
requires = ["setuptools>=61"]
build-backend = "setuptools.build_meta”
[dependency-groups]
dev = [
    "pytest>=8.4.2",
]
```

How do we add more code? Build?

- Let's add a pi calculation
- We need to add an `__init__.py`
- Add a script in pyproject:

```
[project.scripts]
picalc-ip8725 = "picalc_ip8725:main"
example-pi = "picalc_ip8725.pi.pi:main"
```

- Install it
  - Through uv
  - With pip
- `uv build`

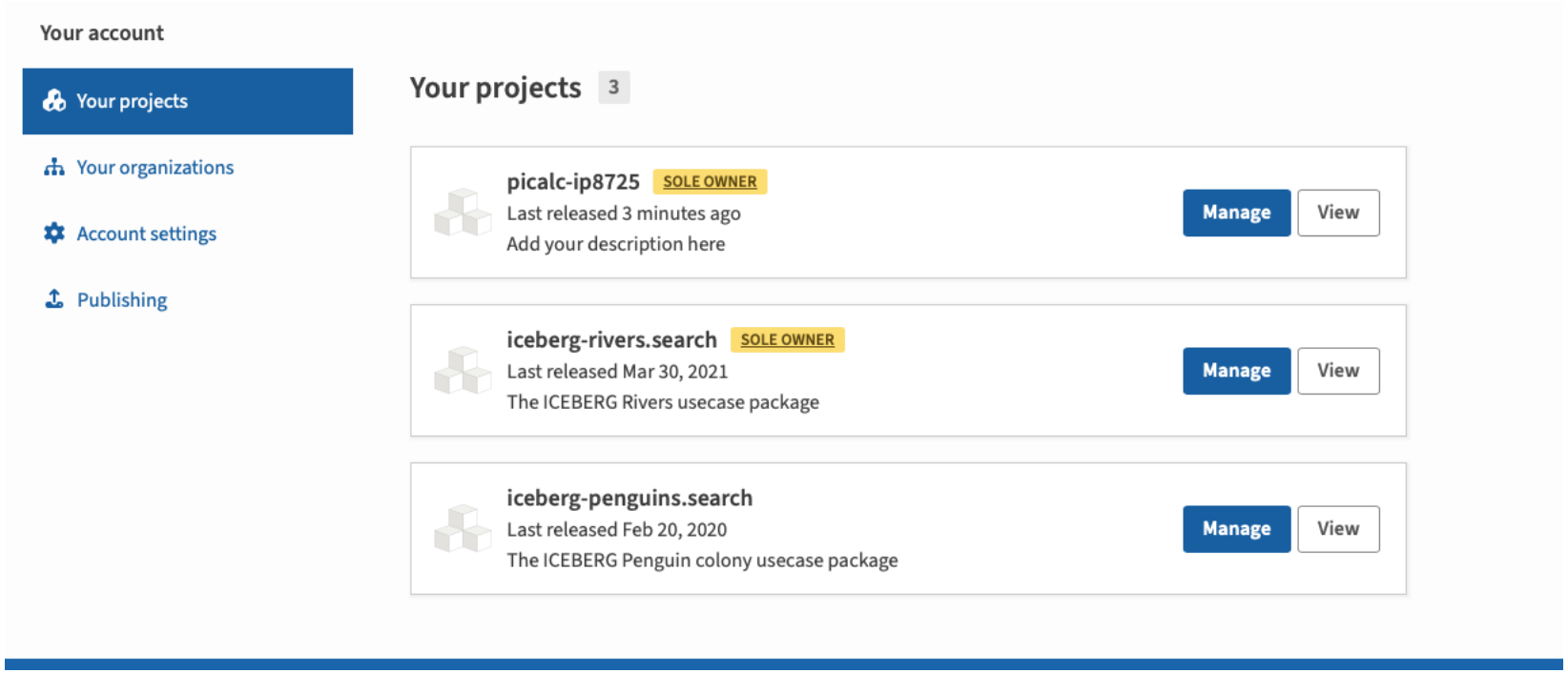


Let’s use the optional dependencies

- We will create a test
- Install everything
  - Through uv
- Run pytests

Let’s publish it

- Setup pypirc first
- `uv tool install twine`
- `uv tool run twine upload --repository testpypi dist/*`



From Github workflows

- Create a release.yaml github workflow
- Add a publisher on pypi
- Create a github release