Workflow basics

Tips to improve productivity

Jason DeBacker

January 17, 2025

Roadmap

Why do you want one of these?

- Syntax highlighting
- Autocorrect/completion of code
- Previews of data objects
- Previews of compiled code (e.g., Markdown, HTML, TeX)
- Integrated terminal to execute code
- Coding copilot

Text editor or IDE

- Text Editors: Lightweight, fast, extensible
- IDEs: Full-featured, integrated tools like debugging, typically software-specific

Which ones to look at for programming in Python?

- Text Editors:
  - → **Visual Studio Code**
    - Free, open-source, excellent Python extension, git integration, integrated terminal, large extension ecosystem, lightweight yet powerful
  - → Atom
    - Close to VS Code, but not quite as good
- IDEs:
  - → Pycharm
    - Advanced debugging, database tools, scientific computing support, but is resource intensive
  - → Spyder
    - Scientific computing focus, IPython integration variable explorer, built for data science

Extensions for Text Editor/IDE

- Code linters (for language-specific errors: Python, Stata, R, etc.)

- Git integration

- DataWrangler (or similar for data preview)

- Jupyter Notebook display

- Preview tools (for Markdown, HTML, LaTeX, etc.)

- TeX compiler

Roadmap

Why use the command line?

- More efficient than GUI
- Essential for development
- Required for many tools
- Enables automation

Essential commands

- Navigation
  - → `pwd` - Print working directory
  - → `ls` / `dir` - List contents
  - → `cd` - Change directory
  - → `cd ..` - Move up a directory level
  - → `cd ~` - Home directory
- File Operations
  - → `mkdir` - Create directory
  - → `touch` / `echo` - Create file
  - → `cp` - Copy
  - → `mv` - Move/rename
  - → `rm` / `rmdir` - Remove

Power user moves

- Tab completion
- Command history (↑/↓)
- `ctrl + r` - Search history
- Wildcards: `*, ?, []`
- `gzip` / `gunzip` - Compress/decompress
- Writing bash scripts
- Git workflow (see Git slides)

Python-specific commands

- `python --version` - Check Python version
- `pip list` / `conda list` - List installed packages
- `pip install package_name` / `conda install package_name` - Install package
- `conda activate venv_name` - Activate conda virtual environment called `venv_name` (More on this later)

Roadmap

Why format code?

- Improves readability/maintainability
- Ensures consistency
- Prevents conflicts
- Speeds up code reviews/debugging
- Reduces errors

Who says what is good formatting?

- **PEP 8** - Python Enhancement Proposal 8
- Official style guide for Python
- Some thought went into this, and many follow
- But who reads?

Thoughtless code formatting

- Two packages (pick one or the other) that can help you format your code without any work:
  - → `black`
    - ■ "Uncompromising" formatter, zero configuration, PEP8 compliant
  - → `autopep8`
    - ■ More configurable, PEP8 based
- I use `black`
- Both can integrate with VS Code or PyCharm (e.g., you can configure to format on save)
- Both can be run from the command line, e.g., `black my_file.py`

Roadmap

Text Editors/IDEs

Navigating the command line

Code formatting

Unit Testing

Continuous Integration

Virtual environments

What is unit testing?

- Testing individual units of code
- Typically functions or classes
- But can also adapt for data analysis (e.g., create data and see if estimator can recover parameters used to create data)

Why test?

- Catch bugs early
- Document behavior
- Enable refactoring
- Improve design

Testing in Python with `pytest`

- `pytest` is the industry standard
- Simple syntax
- Powerful fixtures (e.g., create temporary directories, parameterize tests)
- Rich plugin ecosystem

Testing example

```python
# example_test.py
def test_add_numbers():
    result = add(2, 3)
    assert result == 5

def test_empty_list():
    my_list = []
    assert len(my_list) == 0
    assert not my_list

@pytest.fixture
def sample_data():
    return {"name": "Test", "value": 42}
```

Running unit tests

- Run all tests in a directory: `pytest`
  - $\rightarrow$ will run all files that start with `test_` or end with `_test.py`
  - $\rightarrow$ will run all functions/classes that start with `test_` in those files
  - $\rightarrow$ can specify options (e.g., markers to skip certain tests)
- Run a specific test module in a directory:
  `pytest test_example.py::test_add`
- Run a specific test: `pytest test_example.py::test_add`

Test Best Practices

- Test one thing per test
- Use descriptive names
- Arrange-Act-Assert pattern
- Keep tests independent
- Don't test implementation
- Use fixtures for setup
- Mock external dependencies

Test Coverage

- Coverage.py tool – or Codecov

- Aim for meaningful coverage

- Common targets: 80-90%

- Focus on critical paths

Roadmap

Continuous Integration (CI) CI Fundamentals

- Automated build & test
- Early issue detection
- Quality assurance
- Faster development

GH Actions

- Most popular CI tool for open-source projects
- Free for public repos
- YAML configuration
- Runs on push, pull request, schedule – many triggers you can customize

Example GitHub Actions YAML file

```
1 name: Python CI
2
3 on: [push, pull_request]
4
5 jobs:
6   test:
7     runs-on: ubuntu-latest
8     steps:
9       - uses: actions/checkout@v2
10      - name: Set up Python
11        uses: actions/setup-python@v2
12        with:
13          python-version: '3.x'
14      - name: Install dependencies
15        run: |
16          pip install -r requirements.txt
17          pip install pytest pytest-cov
18      - name: Run tests
19        run: |
20          pytest --cov=./ --cov-report=xml
21      - name: Check formatting
22        run: |
23          black --check .
24          isort --check-only .
```

Example GitHub Actions

Working examples on GitHub: Cost-of-Capital-Calculator workflows

CI Best Practices

- Fast builds
- Cache dependencies
- Matrix testing
- Clear error reporting
- Automated formatting
- Security scanning

CI in GitLab

- Similar to GitHub Actions
- More built-in features
- More complex configuration
- More control over runners
- Still YAML-based, but syntax differs
  - → `.gitlab-ci.yml` used by default
  - → See this comparison of syntax: GitLab CI vs GitHub Actions

Complete Development Workflow

Daily Workflow:

- Pull latest changes
- Create feature branch
- Write tests
- Implement features
- Run local tests
- Format code
- Commit and push
- CI pipeline runs
- Code review
- Merge

Roadmap

Text Editors/IDEs

Navigating the command line

Code formatting

Unit Testing

Continuous Integration

Virtual environments

What is an environment?

- **A Python environment**: a self-contained directory that contains a Python installation for a particular version of Python, plus a number of additional packages.
- **A virtual environment**: a Python environment that is isolated from other Python environments on the same machine.

Why use a virtual environment?

- Open-source projects often have many dependencies, for which new updates may be released at any time
- Virtual environments solve several issues related to this:
  - $\rightarrow$ Isolation
  - $\rightarrow$ Dependency management
  - $\rightarrow$ Reproducibility
  - $\rightarrow$ Avoid conflicts

How to create virtual environment

One way: with conda an a `environment.yml` file, e.g.,

```
 1  name: usitc-env
 2  channels:
 3  - conda-forge
 4  dependencies:
 5  - python >=3.9, <3.13
 6  - numpy
 7  - ipython
 8  - scipy >=1.7.1
 9  - pandas >=1.2.5
10  - numba
11  - jax
12  - matplotlib
13  - scikit-learn
14  - dask >=2.30.0
15  - dask-core >=2.30.0
16  - distributed >=2.30.1
17  - paramtools >=0.15.0
18  - sphinx >=3.5.4
19  - sphinx-argparse
20  - sphinxcontrib-bibtex >=2.0.0
21  - sphinx-math-dollar
22  - pydata-sphinx-theme
23  - jupyter-book >=0.11.3
```

How to create and use virtual environment

- `conda create --name myenv --file environment.yml`
- `conda activate myenv`
- To use in a Jupyter Notebook, install `ipykernel` and run
  `python -m ipykernel install --user --name=myenv`
- To deactivate, run `conda deactivate`

Special cases:

- Create a completely empty environment (named `myenv`):
    → `conda create --name myenv --no-default-packages`
- Dump all installed packages in an environment to a file:
    → `conda env export > environment.yml`

Tools Summary

- Editor/IDE: VS Code or PyCharm
- Formatting: `black` or `autopep8`
- Testing: `pytest`
- CI: GitHub Actions
- VCS: Git
- Virtual Environments: Conda