Building and testing Python

In this article

You can create a continuous integration (CI) workflow to build and test your Python project.

> **Note:** GitHub-hosted runners are not currently supported on GitHub Enterprise Server. You can see more information about planned future support on the [GitHub public roadmap]().

Introduction 🔗

This guide shows you how to build, test, and publish a Python package.

GitHub-hosted runners have a tools cache with pre-installed software, which includes Python and PyPy. You don't have to install anything! For a full list of up-to-date software and the pre-installed versions of Python and PyPy, see "[About GitHub-hosted runners]()".

Prerequisites 🔗

You should be familiar with YAML and the syntax for GitHub Actions. For more information, see "[Learn GitHub Actions]()."

We recommend that you have a basic understanding of Python, PyPy, and pip. For more information, see:

- [Getting started with Python]()
- [PyPy]()
- [Pip package manager]()

Using self-hosted runners on GitHub Enterprise Server 🔗

When using setup actions (such as `actions/setup-LANGUAGE`) on GitHub Enterprise Server with self-hosted runners, you might need to set up the tools cache on runners that do not

have internet access. For more information, see "[Setting up the tool cache on self-hosted runners without internet access](#)."

Using the Python starter workflow 🔗

GitHub provides a Python starter workflow that should work for most Python projects. This guide includes examples that you can use to customize the starter workflow. For more information, see the [Python starter workflow](#).

To get started quickly, add the starter workflow to the `.github/workflows` directory of your repository.

```yaml
name: Python package

on: [push]

jobs:
  build:

    runs-on: ubuntu-latest
    strategy:
      matrix:
        python-version: ["3.7", "3.8", "3.9", "3.10", "3.11"]

    steps:
      - uses: actions/checkout@v2
      - name: Set up Python ${{ matrix.python-version }}
        uses: actions/setup-python@v2
        with:
          python-version: ${{ matrix.python-version }}
      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install ruff pytest
          if [ -f requirements.txt ]; then pip install -r requirements.txt; fi
      - name: Lint with ruff
        run: |
          # stop the build if there are Python syntax errors or undefined names
          ruff --format=github --select=E9,F63,F7,F82 --target-version=py37 .
          # default set of ruff rules with GitHub Annotations
          ruff --format=github --target-version=py37 .
      - name: Test with pytest
        run: |
          pytest
```

Specifying a Python version 🔗

To use a pre-installed version of Python or PyPy on a GitHub-hosted runner, use the `setup-python` action. This action finds a specific version of Python or PyPy from the tools cache on each runner and adds the necessary binaries to `PATH`, which persists for the rest of the job. If a specific version of Python is not pre-installed in the tools cache, the `setup-python` action will download and set up the appropriate version from the [python-versions](#) repository.

Using the `setup-python` action is the recommended way of using Python with GitHub Actions because it ensures consistent behavior across different runners and different versions of Python. If you are using a self-hosted runner, you must install Python and add it to `PATH`. For more information, see the [setup-python action](#).

The table below describes the locations for the tools cache in each GitHub-hosted runner.

	Ubuntu	Mac	Windows
Tool Cache Directory	`/opt/hostedtoolcache/*`	`/Users/runner/hostedtoolcache/*`	`C:\hostedtoolcache\windows\*`
Python Tool Cache	`/opt/hostedtoolcache/Python/*`	`/Users/runner/hostedtoolcache/Python/*`	`C:\hostedtoolcache\windows\Python\*`
PyPy Tool Cache	`/opt/hostedtoolcache/PyPy/*`	`/Users/runner/hostedtoolcache/PyPy/*`	`C:\hostedtoolcache\windows\PyPy\*`

If you are using a self-hosted runner, you can configure the runner to use the `setup-python` action to manage your dependencies. For more information, see [using setup-python with a self-hosted runner](#) in the `setup-python` README.
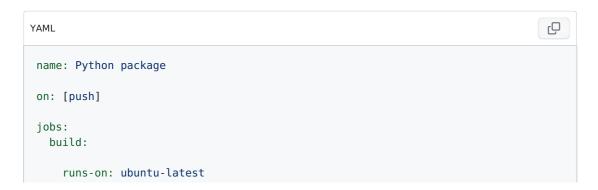
GitHub supports semantic versioning syntax. For more information, see "[Using semantic versioning](#)" and the "[Semantic versioning specification](#)."

Using multiple Python versions 🔗

YAML

```yaml
name: Python package

on: [push]

jobs:
  build:

    runs-on: ubuntu-latest
    strategy:
      # You can use PyPy versions in python-version.
      # For example, pypy2 and pypy3
      matrix:
        python-version: ["2.7", "3.7", "3.8", "3.9", "3.10", "3.11"]

    steps:
      - uses: actions/checkout@v2
      - name: Set up Python ${{ matrix.python-version }}
        uses: actions/setup-python@v2
        with:
          python-version: ${{ matrix.python-version }}
      # You can test your matrix by printing the current Python version
      - name: Display Python version
        run: python -c "import sys; print(sys.version)"
```

Using a specific Python version 🔗

You can configure a specific version of Python. For example, 3.10. Alternatively, you can use semantic version syntax to get the latest minor release. This example uses the latest minor release of Python 3.

YAML

```yaml
name: Python package

on: [push]

jobs:
  build:

    runs-on: ubuntu-latest
```

```
    steps:
      - uses: actions/checkout@v2
      - name: Set up Python 3.x
        uses: actions/setup-python@v2
        with:
          # Semantic version range syntax or exact version of a Python version
          python-version: '3.x'
          # Optional - x64 or x86 architecture, defaults to x64
          architecture: 'x64'
      # You can test your matrix by printing the current Python version
      - name: Display Python version
        run: python -c "import sys; print(sys.version)"
```

Excluding a version 🔗

If you specify a version of Python that is not available, `setup-python` fails with an error such as: `##[error]Version 3.6 with arch x64 not found`. The error message includes the available versions.

You can also use the `exclude` keyword in your workflow if there is a configuration of Python that you do not wish to run. For more information, see "[Workflow syntax for GitHub Actions](#)."

YAML

```
name: Python package

on: [push]

jobs:
  build:

    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        os: [ubuntu-latest, macos-latest, windows-latest]
        python-version: ["3.7", "3.8", "3.9", "3.10", "3.11", pypy2, pypy3]
        exclude:
          - os: macos-latest
            python-version: "3.7"
          - os: windows-latest
            python-version: "3.7"
```

Using the default Python version 🔗

We recommend using `setup-python` to configure the version of Python used in your workflows because it helps make your dependencies explicit. If you don't use `setup-python`, the default version of Python set in `PATH` is used in any shell when you call `python`. The default version of Python varies between GitHub-hosted runners, which may cause unexpected changes or use an older version than expected.

GitHub-hosted runner	Description
Ubuntu	Ubuntu runners have multiple versions of system Python installed under `/usr/bin/python` and `/usr/bin/python3`. The Python versions that come packaged with Ubuntu are in addition to the versions that GitHub installs in the tools cache.
Windows	Excluding the versions of Python that are in the tools cache, Windows does not ship with an

	equivalent version of system Python. To maintain consistent behavior with other runners and to allow Python to be used out-of-the-box without the `setup-python` action, GitHub adds a few versions from the tools cache to `PATH`.
macOS	The macOS runners have more than one version of system Python installed, in addition to the versions that are part of the tools cache. The system Python versions are located in the `/usr/local/Cellar/python/*` directory.

Installing dependencies 🔗

GitHub-hosted runners have the pip package manager installed. You can use pip to install dependencies from the PyPI package registry before building and testing your code. For example, the YAML below installs or upgrades the `pip` package installer and the `setuptools` and `wheel` packages.

```yaml
steps:
- uses: actions/checkout@v2
- name: Set up Python
  uses: actions/setup-python@v2
  with:
    python-version: '3.x'
- name: Install dependencies
  run: python -m pip install --upgrade pip setuptools wheel
```

Requirements file 🔗

After you update `pip`, a typical next step is to install dependencies from *requirements.txt*. For more information, see [pip](#).

```yaml
steps:
- uses: actions/checkout@v2
- name: Set up Python
  uses: actions/setup-python@v2
  with:
    python-version: '3.x'
- name: Install dependencies
  run: |
    python -m pip install --upgrade pip
    pip install -r requirements.txt
```

Testing your code 🔗

You can use the same commands that you use locally to build and test your code.

Testing with pytest and pytest-cov 🔗

This example installs or upgrades `pytest` and `pytest-cov`. Tests are then run and output in JUnit format while code coverage results are output in Cobertura. For more information, see [JUnit](#) and [Cobertura](#).

```yaml
steps:
- uses: actions/checkout@v2
- name: Set up Python
  uses: actions/setup-python@v2
  with:
    python-version: '3.x'
- name: Install dependencies
  run: |
    python -m pip install --upgrade pip
    pip install -r requirements.txt
- name: Test with pytest
  run: |
    pip install pytest pytest-cov
    pytest tests.py --doctest-modules --junitxml=junit/test-results.xml --cov=com --
```

Using Ruff to lint code 🔗

The following example installs or upgrades `ruff` and uses it to lint all files. For more information, see [Ruff](#).

```yaml
steps:
- uses: actions/checkout@v2
- name: Set up Python
  uses: actions/setup-python@v2
  with:
    python-version: '3.x'
- name: Install dependencies
  run: |
    python -m pip install --upgrade pip
    pip install -r requirements.txt
- name: Lint with Ruff
  run: |
    pip install ruff
    ruff --format=github --target-version=py37 .
  continue-on-error: true
```

The linting step has `continue-on-error: true` set. This will keep the workflow from failing if the linting step doesn't succeed. Once you've addressed all of the linting errors, you can remove this option so the workflow will catch new issues.

Running tests with tox 🔗

With GitHub Actions, you can run tests with tox and spread the work across multiple jobs. You'll need to invoke tox using the `-e py` option to choose the version of Python in your `PATH`, rather than specifying a specific version. For more information, see [tox](#).

```yaml
name: Python package

on: [push]

jobs:
  build:

    runs-on: ubuntu-latest
    strategy:
```

```yaml
    matrix:
      python: ["3.9", "3.10", "3.11"]

    steps:
      - uses: actions/checkout@v2
      - name: Setup Python
        uses: actions/setup-python@v2
        with:
          python-version: ${{ matrix.python }}
      - name: Install tox and any other packages
        run: pip install tox
      - name: Run tox
        # Run tox using the version of Python in `PATH`
        run: tox -e py
```

Packaging workflow data as artifacts 🔗

You can upload artifacts to view after a workflow completes. For example, you may need to save log files, core dumps, test results, or screenshots. For more information, see "Storing workflow data as artifacts."

The following example demonstrates how you can use the `upload-artifact` action to archive test results from running `pytest`. For more information, see the `upload-artifact` action.

```yaml
YAML                                                              ⎘

name: Python package

on: [push]

jobs:
  build:

    runs-on: ubuntu-latest
    strategy:
      matrix:
        python-version: ["3.7", "3.8", "3.9", "3.10", "3.11"]

    steps:
      - uses: actions/checkout@v2
      - name: Setup Python # Set Python version
        uses: actions/setup-python@v2
        with:
          python-version: ${{ matrix.python-version }}
      # Install pip and pytest
      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install pytest
      - name: Test with pytest
        run: pytest tests.py --doctest-modules --junitxml=junit/test-results-${{ mat
      - name: Upload pytest test results
        uses: actions/upload-artifact@v2
        with:
          name: pytest-results-${{ matrix.python-version }}
          path: junit/test-results-${{ matrix.python-version }}.xml
          # Use always() to always run this step to publish test results when there a
        if: ${{ always() }}
```

Publishing to package registries 🔗

You can configure your workflow to publish your Python package to a package registry once your CI tests pass. This section demonstrates how you can use GitHub Actions to upload your package to PyPI each time you [publish a release](#).

For this example, you will need to create two [PyPI API tokens](#). You can use secrets to store the access tokens or credentials needed to publish your package. For more information, see "[Encrypted secrets](#)."

```yaml
# This workflow uses actions that are not certified by GitHub.
# They are provided by a third-party and are governed by
# separate terms of service, privacy policy, and support
# documentation.

# GitHub recommends pinning actions to a commit SHA.
# To get a newer version, you will need to update the SHA.
# You can also reference a tag or branch, but the action may change without warning

name: Upload Python Package

on:
  release:
    types: [published]

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Set up Python
        uses: actions/setup-python@v2
        with:
          python-version: '3.x'
      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install build
      - name: Build package
        run: python -m build
      - name: Publish package
        uses: pypa/gh-action-pypi-publish@27b31702a0e7fc50959f5ad993c78deac1bdfc29
        with:
          user: __token__
          password: ${{ secrets.PYPI_API_TOKEN }}
```

For more information about the starter workflow, see `python-publish` .

Legal