MINI-COURSE ON CODE DEVELOPMENT AND PACKAGING

INTRO TO SOME ADVANCED TOPICS

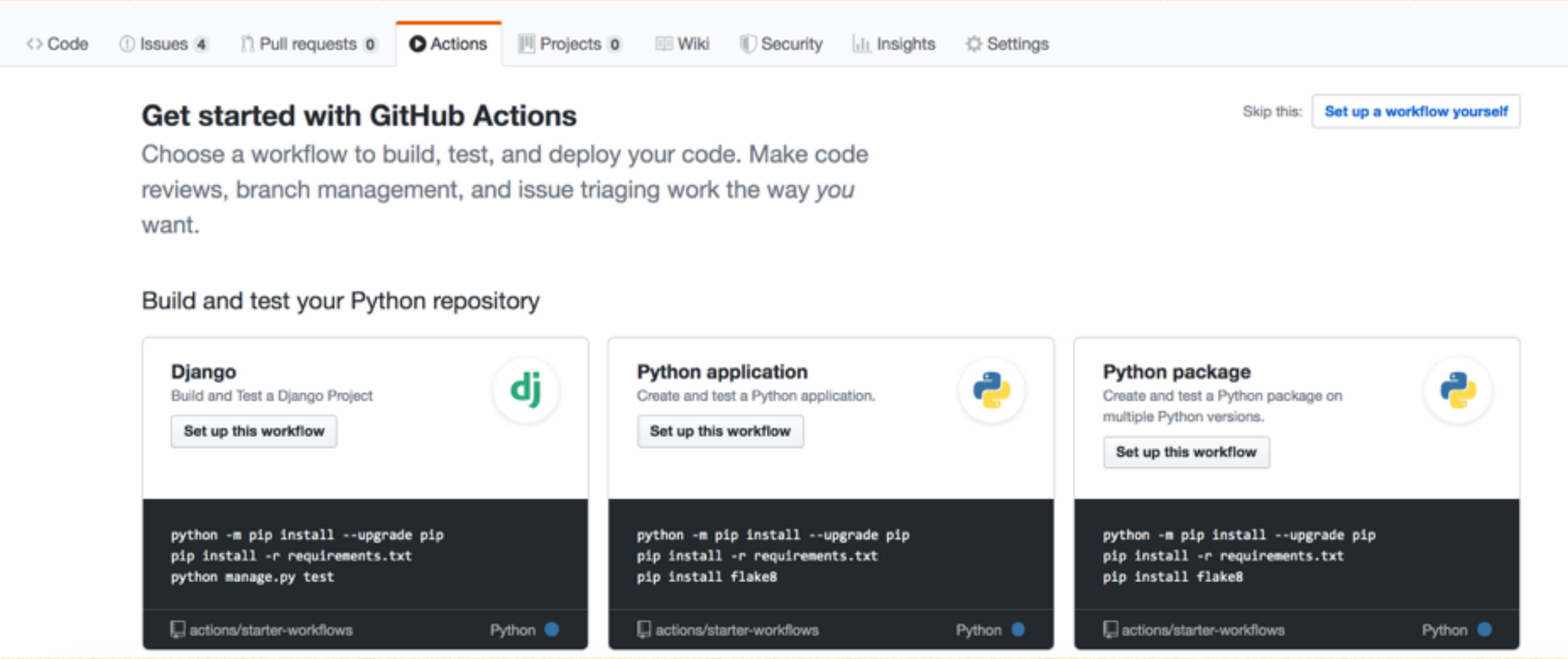
GitHub Actions

GitHub Actions

- Last year, GitHub itself released a CI/CD (continuous-integration/continuous-delivery) service that is automatically part of every GitHub repository —> `GitHub Actions`
- Similar to Travis CI, configured with YAML files (`.yml`), but some big advantages:
  - Easily configure multiple different runs to do (CI tests, building documentation, building a website, responding to issues/pull requests, pushing to AWS S3, …)
  - Re-use atomic steps defined in other GitHub repositories (e.g., setup python, setup Miniconda)
  - More free runners available (up to 20)

SETTING UP GITHUB ACTIONS FOR YOUR REPOSITORY

Go to “Actions” tab



SETTING UP GITHUB ACTIONS FOR YOUR REPOSITORY

or just add a `.yml` file under `.github/workflows`

```yaml
name: Test exampy

on: [push]

jobs:
  build:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        python-version: [3.7, 3.8]
        numpy-version: [1.16,1.17,1.18]
        exclude:
          - python-version: 3.8
            numpy-version: 1.16
          - python-version: 3.8
            numpy-version: 1.17
    steps:
    - uses: actions/checkout@v2
    - name: Set up Python ${{ matrix.python-version }}
      uses: actions/setup-python@v1
      with:
        python-version: ${{ matrix.python-version }}
```

```yaml
    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install numpy==${{ matrix.numpy-version }}
    - name: Install package
      run: |
        pip install -e .
    - name: Test with pytest
      run: |
        pip install pytest
        pip install pytest-cov
        pip install scipy
        pytest -v tests/ --cov=exampy/
```

```yaml
name: Test exampy

on: [push]

jobs:
  build:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        python-version: [3.7, 3.8]
        numpy-version: [1.16,1.17,1.18]
        exclude:
          - python-version: 3.8
            numpy-version: 1.16
          - python-version: 3.8
            numpy-version: 1.17
    steps:
    - uses: actions/checkout@v2
    - name: Set up Python ${{ matrix.python-version }}
      uses: actions/setup-python@v1
      with:
        python-version: ${{ matrix.python-version }}
    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install numpy==${{ matrix.numpy-version }}
    - name: Install package
      run: |
        pip install -e .
    - name: Test with pytest
      run: |
        pip install pytest
        pip install pytest-cov
        pip install scipy
        pytest -v tests/ --cov=exampy/
```

<— Informative name

<— When to run the workflow

(e.g., on: [push,pull request])

<— Operating system

(can be ubuntu, mac, windows)

<— Define a matrix of different builds, similar to Travis CI

(can't 'include', so exclude)

<— Arbitrary sequence of steps:

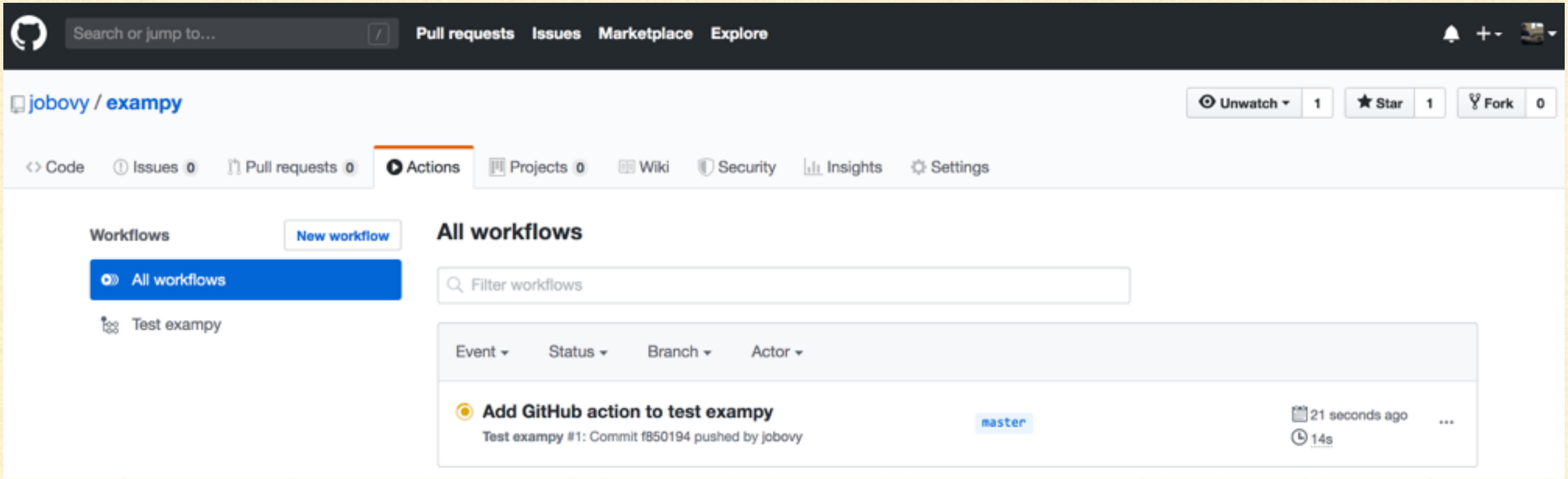
1) check out the repository

<— 2) setup Python

<— 3) install dependencies with pip

<— 4) install package

<— 5) install test dependencies and run tests


Search or jump to...
Pull requests
Issues
Marketplace
Explore
jobovy / exampy
Unwatch 1
Star 1
Fork 0
Code
Issues 0
Pull requests 0
Actions
Projects 0
Wiki
Security
Insights
Settings
Workflows
New workflow
All workflows
Test exampy
All workflows
Filter workflows
Event
Status
Branch
Actor
Add GitHub action to test exampy
Test exampy #1: Commit f850194 pushed by jobovy
master
21 seconds ago
14s

jobovy / exampy

Unwatch 1 | Star 1 | Fork 0

Code | Issues 0 | Pull requests 0 | Actions | Projects 0 | Wiki | Security | Insights | Settings

Fix exclusion syntax

master 93cf2da

Test exampy
on: push

- build (3.7, 1.16)
- build (3.7, 1.17)
- build (3.7, 1.18)
- build (3.8, 1.18)

Test exampy / **build (3.7, 1.16)**
succeeded 24 seconds ago in 21s

Search logs

Set up job	2s
Run actions/checkout@v2	1s
Set up Python 3.7	0s
Install dependencies	8s
Install package	1s
Test with pytest	9s
Post actions/checkout@v2	0s
Complete job	0s

Easy to use multiple operating systems:

```yaml
name: Test exampy

on: [push]

jobs:
  build:
    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        os: [ubuntu-latest, windows-latest, macos-latest]
        python-version: [3.7, 3.8]
        numpy-version: [1.16,1.17,1.18]
        exclude:
          - os: ubuntu-latest
            python-version: 3.8
            numpy-version: 1.16
          - os: ubuntu-latest
            python-version: 3.8
            numpy-version: 1.17
          - os: windows-latest
            python-version: 3.8
            numpy-version: 1.16
          - os: windows-latest
            python-version: 3.8
            numpy-version: 1.17
          - os: macos-latest
            python-version: 3.8
            numpy-version: 1.16
          - os: macos-latest
            python-version: 3.8
            numpy-version: 1.17
    steps:
```

jobovy / exampy

Unwatch 1 | Star 1 | Fork 0

Code | Issues 0 | Pull requests 0 | Actions | Projects 0 | Wiki | Security | Insights | Settings

Run integration tests on all three major OS using Act…

master 6c26f8e

Test exampy
on: push

- build (ubuntu-latest, 3.7, …
- build (ubuntu-latest, 3.7, …
- build (ubuntu-latest, 3.7, …
- build (ubuntu-latest, 3.8, …
- build (windows-latest, 3.…
- build (windows-latest, 3.…
- build (windows-latest, 3.…
- build (windows-latest, 3.…
- build (macos-latest, 3.7, …
- build (macos-latest, 3.7, …
- build (macos-latest, 3.7, …
- build (macos-latest, 3.8, …

Test exampy

This run | Workflow file

Create status badge

12 completed jobs in 1m 34s

Artifacts

No artifacts were generated.

Annotations

RE-USING OTHER ACTIONS

- GitHub users can define their own ‘actions’ that you can re-use in your own workflows: e.g.,
  - `- uses: actions/checkout@v2`
  -
```
- name: Set up Python ${{ matrix.python-version }}
  uses: actions/setup-python@v1
  with:
    python-version: ${{ matrix.python-version }}
```
- Take arguments in the `with:` section
- Make sure to a) use a released version (e.g., `@v2`), b) check that you can trust the action if you give it passwords or other permissions

USING SECRETS

- If as part of a workflow, you need to authenticate, you can add ‘secrets’ to your GitHub repository that can be used by actions (usernames, passwords, SSH keys, etc.)
- Use as `${{ secrets.SECRET_NAME }}`
- Not shared with forks, quite secure
- But always try to make secrets specific to GitHub, so can be easily revoked (e.g., set up special SSH key for GitHub use rather than using your normal one)
- SSH keys: set up special GitHub private key with *no passphrase*

jobovy / exampy

Unwatch 1 | Star 2 | Fork 0

Code | Issues 0 | Pull requests 0 | Actions | Projects 0 | Wiki | Security | Insights | Settings

Options

- Manage access
- Branches
- Webhooks
- Notifications
- Integrations & services
- Deploy keys
- Autolink references
- Secrets

Moderation

- Interaction limits

Settings

Repository name

exampy | Rename

Template repository

Template repositories let users generate new repositories with the same directory structure and files. Indicate if **jobovy/exampy** can be used as a template for creating other repositories.

Social preview

Upload an image to customize your repository's social media preview.

Images should be at least 640×320px (1280×640px for best display).

Download template

jobovy / exampy

Unwatch 1 | Star 2 | Fork 0

Code | Issues 0 | Pull requests 0 | Actions | Projects 0 | Wiki | Security | Insights | Settings

- Options
- Manage access
- Branches
- Webhooks
- Notifications
- Integrations & services
- Deploy keys
- Autolink references
- **Secrets**
- Actions

Moderation

- Interaction limits

Secrets

Secrets are environment variables that are **encrypted** and only exposed to selected actions. Anyone with **collaborator** access to this repository can use these secrets in a workflow.

Secrets are not passed to workflows that are triggered by a pull request from a fork. Learn more.

Add a new secret

Code | Issues 0 | Pull requests 0 | Actions | Projects 0 | Wiki | Security | Insights | Settings

- Options
- Manage access
- Branches
- Webhooks
- Notifications
- Integrations & services
- Deploy keys
- Autolink references
- **Secrets**
- Actions

Moderation

- Interaction limits

Secrets

Secrets are environment variables that are **encrypted** and only exposed to selected actions. Anyone with **collaborator** access to this repository can use these secrets in a workflow.

Secrets are not passed to workflows that are triggered by a pull request from a fork. Learn more.

Add a new secret

Name

SSH_PRIVATE_KEY

Value

PRIVATE KEY HERE (get with cat private_key | pbcopy)

Add secret

Once you hit 'Add secret',
you can never see the value again

PRACTICAL EXAMPLE: UPDATE YOUR WEBSITE ON LEPUS

```yaml
name: Update website

on: [push]

jobs:
  build_website:
    name: Build main website
    runs-on: ubuntu-latest
    if: "!contains(github.event.head_commit.message, 'ci skip')"
    steps:
    # check-out this repository
    - uses: actions/checkout@v2
    # Build main website
    - name: Build website
      shell: bash -l {0}
      run: |
          mkdir build
          make web ONLINEPATH=build
    # Setup SSH agent
    - uses: webfactory/ssh-agent@v0.2.0
      with:
        ssh-private-key: ${{ secrets.SSH_PRIVATE_KEY }}
    # Upload to lepus, to quickly upload new website
    - name: Upload
      shell: bash -l {0}
      working-directory: build
      run: rsync -e"ssh -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null" -azv ./ bovy@lepus.astro.utoronto.ca:/home/bovy/web/
```

```yaml
  build_cv:
    name: Build CV
    runs-on: ubuntu-latest
    if: "!contains(github.event.head_commit.message, 'ci skip')"
    steps:
    # check-out this repository
    - uses: actions/checkout@v2
    # Setup SSH agent
    - uses: webfactory/ssh-agent@v0.2.0
      with:
        ssh-private-key: ${{ secrets.SSH_PRIVATE_KEY }}
    # Install LaTeX for CV building
    - name: Install LaTeX
      shell: bash -l {0}
      run: |
          sudo paperconfig -p letter
          sudo apt-get install -qq --no-install-recommends dvipng texlive-latex-base texlive-latex-extra texlive-fonts-recommended graphviz
    # Build CV
    - name: Build CV
      shell: bash -l {0}
      working-directory: CV
      run: |
          mkdir ../build
          make web ONLINEPATH=../build
    # Upload to lepus, including CV now
    - name: Upload
      shell: bash -l {0}
      working-directory: build
      run: rsync -e"ssh -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null" -azv ./ bovy@lepus.astro.utoronto.ca:/home/bovy/web/
```

PRACTICAL EXAMPLE: UPDATE WEBSITE ON AWS S3

```yaml
name: Update website

# Update on every push
on: [push]

jobs:
  build:
    name: Update website
    runs-on: ubuntu-latest
    if: "!contains(github.event.head_commit.message, 'ci skip')"
    steps:
      # check-out this repository
      - uses: actions/checkout@v2
      # Specify Python version as >=3
      - uses: actions/setup-python@v1
        with:
          python-version: '>3.6'
          architecture: 'x64'
      # Verify the JSON file
      - name: verify papers JSON
        working-directory: py
        shell: bash -l {0}
        run: python verify_papers_json.py
      # Push to Amazon S3
      - name: Upload website
        uses: jakejarvis/s3-sync-action@v0.5.1
        with:
          args: --acl public-read --follow-symlinks --delete
        env:
          AWS_S3_BUCKET: ${{ secrets.AWS_S3_BUCKET }}
          AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
          AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          AWS_REGION: 'us-east-2'
          SOURCE_DIR: 'src'      # optional: defaults to entire repository
```

ADDING BADGES TO YOUR README

WHY BADGES?

- What are status badges? Images like 

- Helpful way to get an overview of your package's status on the various services that you are using:
  - Continuous-integration services like `Travis CI`, `AppVeyor`, `GitHub Actions`
  - Test coverage statistics from `Codecov`
  - Documentation's status from `readthedocs.io`
  - Package version on PyPI
  - …

README.md	Add various badges to README	now
setup.py	Add long_description	5 days ago

README.md

exampy

This is an example Python package to illustrate a set of notes on code development and packaging.

build passing | build passing | Test exampy passing | codecov 89% | docs passing | license MIT

jobovy / exampy

Unwatch 1 | Star 1 | Fork 0

Code | Issues 0 | Pull requests 0 | Actions | Projects 0 | Wiki | Security | Insights | Settings

exampy / README.md Cancel

Edit file | Preview changes Spaces 2 Soft wrap

```
# exampy

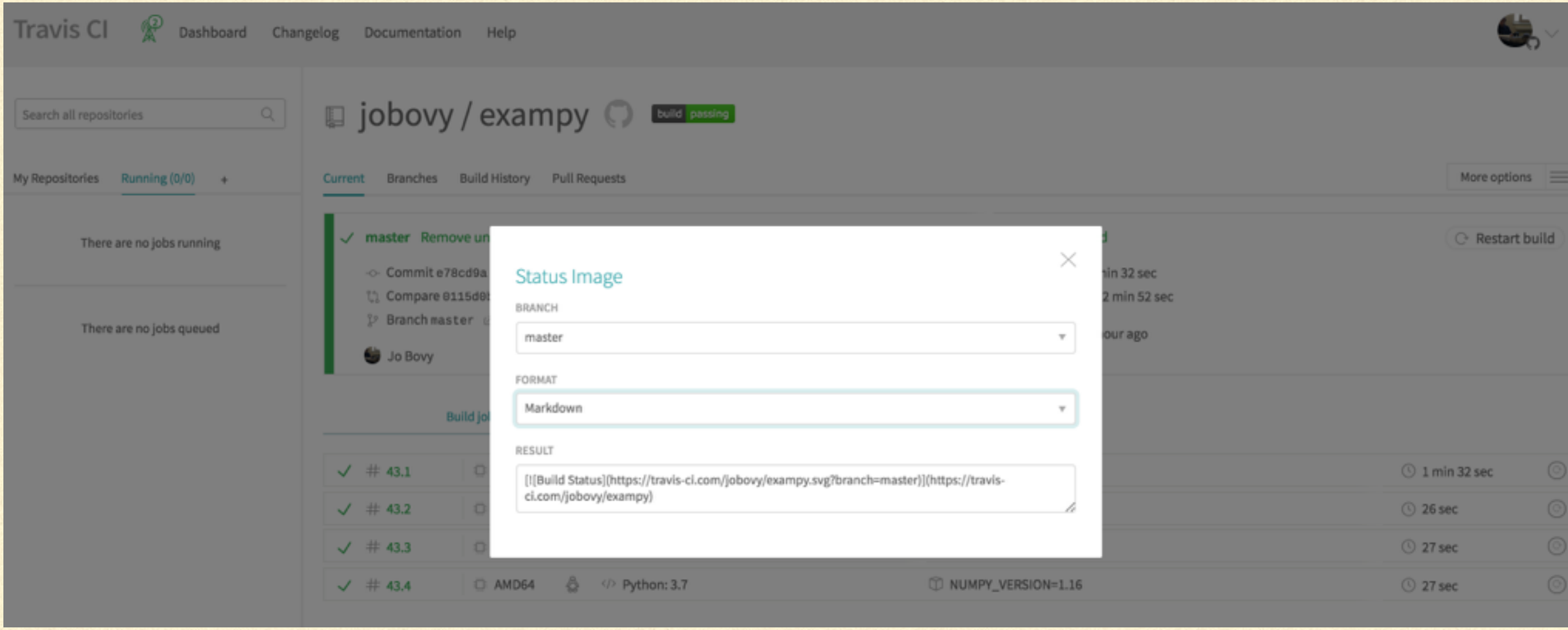
This is an example Python package to illustrate a set of notes on code
development and packaging.

[![Build Status](https://travis-ci.com/jobovy/exampy.svg?branch=master)](https://travis-ci.com/jobovy/exampy)
[![Build status](https://ci.appveyor.com/api/projects/status/7hybo3b6t0rrxnio?svg=true)](https://ci.appveyor.com/project/jobovy/exampy)
![Test exampy](https://github.com/jobovy/exampy/workflows/Test%20exampy/badge.svg)
[![codecov](https://codecov.io/gh/jobovy/exampy/branch/master/graph/badge.svg)](https://codecov.io/gh/jobovy/exampy)
[![Documentation Status](https://readthedocs.org/projects/exampy/badge/?version=latest)](https://exampy.readthedocs.io/en/latest/?badge=latest)
[![image](http://img.shields.io/badge/license-MIT-brightgreen.svg)](https://github.com/jobovy/exampy/blob/master/LICENSE)

```

Travis CI

Click on the badge!



AppVeyor

jobovy Projects Environments BYOC Account Support Jo Bovy

exampy

Current build History Deployments Events **Settings**

- General
- Environment
- Build
- Tests
- Artifacts
- Deployment
- NuGet
- Notifications
- Permissions
- **Badges**
- Export YAML
- Delete project

Project status badge

Current status

build passing

SVG image URL

```
https://ci.appveyor.com/api/projects/status/7hybo3b6t0rrxnio?svg=true
```

Raster image URL

```
https://ci.appveyor.com/api/projects/status/7hybo3b6t0rrxnio
```

Sample markdown code

```
[![Build status](https://ci.appveyor.com/api/projects/status/7hybo3b6t0rrxnio?
```

master branch status badge

SVG branch image URL

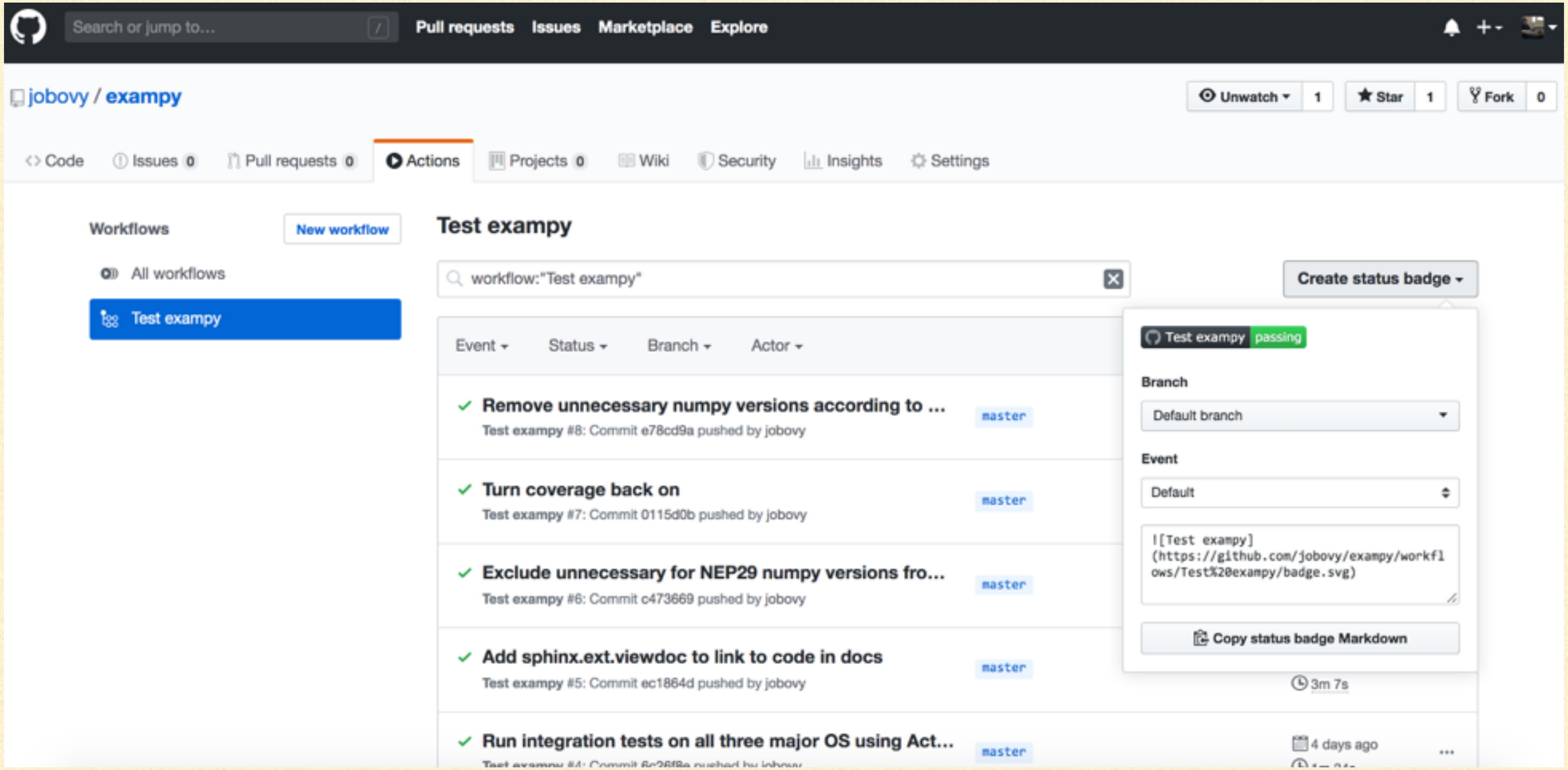
```
https://ci.appveyor.com/api/projects/status/7hybo3b6t0rrxnio/branch/master?svg=true
```

Raster branch image URL

```
https://ci.appveyor.com/api/projects/status/7hybo3b6t0rrxnio/branch/master
```

GitHub Actions

Test exampy passing



Codecov

codecov 89%

gh jobovy exampy Docs Support

Overview Commits Branches Pulls Compare Settings

General
Yaml
Badge

Markdown Copy

```
[![codecov](https://codecov.io/gh/jobovy/exampy/branch/master/graph/badge.svg)](https://codecov.
```

HTML Copy

```
<a href="https://codecov.io/gh/jobovy/exampy">
  <img src="https://codecov.io/gh/jobovy/exampy/branch/master/graph/badge.svg" />
</a>
```

RST Copy

```
.. image:: https://codecov.io/gh/jobovy/exampy/branch/master/graph/badge.svg
  :target: https://codecov.io/gh/jobovy/exampy
```

readthedocs.io

docs passing

Read the Docs

bovy

Projects >

exampy

View Docs

Overview | Downloads | Search | Builds | Versions | Admin

reStructuredText

```
.. image:: https://readthedocs.org/projects/exampy/badge/?version=l
:target: https://exampy.readthedocs.io/en/latest/?badge=latest
:alt: Documentation Status
```

Markdown

```
[![Documentation Status](https://readthedocs.org/projects/exampy/ba
```

HTML

```
<a href='https://exampy.readthedocs.io/en/latest/?badge=latest'>
    <img src='https://readthedocs.org/projects/exampy/badge/?versic
</a>
```

Repository
https://github.com/jobovy/exampy.git

Project Slug
exampy

Last Built
1 hour, 37 minutes ago passed

Maintainers

Badge
docs passing

Tags
Project has no tags. Add some in your project settings.

Project Privacy Level
Public

PyPI

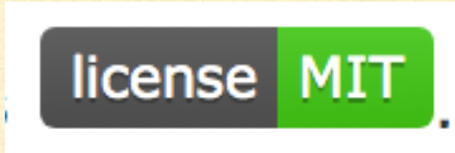

```
[![image](http://img.shields.io/pypi/v/exampy.svg)](https://pypi.python.org/pypi/exampy/)
```

which gives 

.

MAKE YOUR OWN WITH `shields.io`

```
[![image](http://img.shields.io/badge/license-MIT-brightgreen.svg)](https://github.com/jobovy/exampy/blob/master/LICENSE)
```



README.md	Add various badges to README	now
setup.py	Add long_description	5 days ago

README.md

exampy

This is an example Python package to illustrate a set of notes on code development and packaging.

build passing | build passing | Test exampy passing | codecov 89% | docs passing | license MIT

ADDING A C EXTENSION TO YOUR PACKAGE

WHY YOU MIGHT WANT TO ADD A C EXTENSION

- Python is very convenient to code in, but can be slow for computationally-expensive applications
- *Olden times*: write entire application in C or Fortran to get speed (e.g., Gadget, MESA)
- Now: provide interaction API in Python for easy code interaction, results analysis, plotting; write slow parts in compiled language
- Could use other compiled languages, but Python is itself written in C (CPython), so C is the easiest option

OPTIONS FOR C EXTENSION

- Write an entire module in C (e.g., some parts of the standard library):
  - Cumbersome, difficult, and not very portable
- Wrap existing C library/code in Python with no changes to C code:
  - Advantage is that you can wrap existing code or allow C code to interact with other languages (e.g., wrap same C library in multiple languages)
  - `ctypes`, or wrapper generators like SWIG (SWIG not recommended)
- Write C code specific to Python application, but use C only to speed up computationally-expensive parts of the code:
  - Convenient, good option for speeding-up highly package-specific code
  - `cython`
- Don't write any C code! Just use an automatic Python —> C/machine-code compiler: works well for simple functions

`ctypes`

- My own preferred method, part of standard library
- Sequence:
  - Compile C module into *shared library* or *DLL* (can be done automatically in `setup.py`)
  - Load shared library in Python code, declare function signature
  - Call functions in the shared library directly from Python

ctypes: example

```
// C extension for exampy
#include <_math.h>
double square(double x){
  return x*x;
}
-uuu:**-F1  _math.c        Top L1     (C/l Abbrev)------------------------------
#ifndef __EXAMPY_MATH_H__
#define __EXAMPY_MATH_H__
double square(double);
#endif /* _math.h */

-uuu:---F1  _math.h        All L5     (C/l Flymake:! Abbrev)--------------------
```

ctypes: example

```
import setuptools

with open("README.md", "r") as fh:
    long_description = fh.read()

setuptools.setup(
    name="exampy_c",
    version="0.2",
    author="Jo Bovy",
    author_email="bovy@astro.utoronto.ca",
    description="A small example Python package",
    long_description=long_description,
    long_description_content_type="text/markdown",
    packages=["exampy_c","exampy_c/integrate"],
    install_requires=["numpy"],
    ext_modules=[setuptools.Extension('libexampy',sources=['exampy_c/_math.c'],
                                      include_dirs=['exampy_c'])]
)

-uu-:---F1  setup.py       All L12    (Python Flymake Pyflakes)----------------
```

ctypes: example

```
import os
import ctypes
import numpy.ctypeslib as ctl
import sysconfig
math_lib= ctl.load_library('libexampy'+sysconfig.get_config_var('EXT_SUFFIX'),
                           os.path.abspath(\
                os.path.join(os.path.dirname(__file__),'..')))
square_c= math_lib.square
square_c.argtypes= [ctypes.c_double]
square_c.restype= ctypes.c_double

def square(x):
    """The square of a number

-uu-:---F1  _math.py       Top L14    (Python Flymake Pyflakes)----------------
```

`ctypes`

- Annoying:
  - Can be confusing to find the library (built for system libraries)
  - You have to declare all functions and their parameters and return types
  - Complicated to pass objects, need to define them as `structs` in C
- Good!
  - Can easily pass arrays as pointers, other pointers (but make sure arrays are `C_CONTIGUOUS`)
  - Very fast

`ctypes` on Windows

- Things are a bit more complicated on Windows: You need to explicitly export the C functions you want to expose, leads to code such as

  - ```
    #ifdef _WIN32
    #include <Python.h>
    #define EXPORT __declspec(dllexport)
    #if PY_MAJOR_VERSION >= 3
    PyMODINIT_FUNC PyInit_libexampy(void) { // Python 3
      return NULL;
    }
    #else
    PyMODINIT_FUNC initlibexampy(void) {} // Python 2
    #endif
    #endif
    #else
    #if defined(__GNUC__)
    #define EXPORT __attribute__((visibility("default")))
    #else
    #define EXPORT
    #endif
    ```

  - Then put EXPORT in front of functions you want to expose

- Since Python 3.8, you need to specify `winmode=0x008` in the CDLL call
    ```

cython

- Essentially, a way to write Python-style code that allows fast compiled C code to be generated
- So keep many of the advantages of coding in Python, combined with speed of C
- Install with `pip install cython`

cython: example

```
from libc.math cimport pow

def square(double x):
    """The square of a number

Parameters
----------
x: float
    Number to square

Returns
-------
float
    Square of x
"""
    return pow(x,2.0)

-uu-:---F1  _math.pyx       All L17    (Fundamental)----------------------------
```

Note the docstring!

cython: example

```
import setuptools
from Cython.Build import cythonize

with open("README.md", "r") as fh:
    long_description = fh.read()

setuptools.setup(
    name="exampy_c",
    version="0.2",
    author="Jo Bovy",
    author_email="bovy@astro.utoronto.ca",
    description="A small example Python package",
    long_description=long_description,
    long_description_content_type="text/markdown",
    packages=["exampy_c","exampy_c/integrate"],
    install_requires=["numpy"],
    ext_modules=cythonize(setuptools.Extension("exampy_c.math_cython",
                                               sources=['exampy_c/_math.pyx']))
)

-uu-:---F1  setup.py       Top L21    (Python Flymake Pyflakes)----------------
```

Can now `import exampy_c.math_cython` and use functions

`cython`

- Typical workflow:
  - Figure out which parts of your code are slow
  - Move them to a .pyx file, add static types (double, etc.)
  - use `def` for functions available in Python, `cdef` for functions only available in C, `cpdef` for functions available in both
- Quite complicated in the end, because need to almost learn another language and create non-portable code

ALSO CHECK OUT

- numba: Just-In-Time (JIT) compilation of functions that use numpy into fast machine code
- cupy: run numpy code on a GPU with minimal changes

```
from numba import jit
import random

@jit(nopython=True)
def monte_carlo_pi(nsamples):
    acc = 0
    for i in range(nsamples):
        x = random.random()
        y = random.random()
        if (x ** 2 + y ** 2) < 1.0:
            acc += 1
    return 4.0 * acc / nsamples
```

PRESENTATIONS