PYTHON APPLICATION LAYOUTS

What you will learn: This video shows you how to organize your python code and associated files. Five examples cover a variety of scenarios:

1. One-off script
2. Installable single package
3. Application with internal packages
4. Django web application
5. Flask web application

PREREQUISITES

This video assumes you have familiarity with python modules and packages.

If you need a refresher:

https://realpython.com/python-modules-packages/

INTRODUCTION

- Python is very flexible on how you structure your applications
  - Brand new project folders can be daunting
- Content is based on article from Kyle Stratis:

  https://realpython.com/python-application-layouts/
- Opinions may differ (see the last section)
- Sample code

ONE-OFF SCRIPT

- Simple case: a single python file containing all your code
- Works for code without dependencies or using a pip/pipenv environment

```
oneoff/
├── helloworld.py
├── tests.py
│
├── .gitignore
│
├── setup.py
├── requirements.txt
│
├── LICENSE
└── README.md
```

.gitignore

- Configuration file that tells **git** to ignore certain file types

```
# simple .gitignore example file
.DS_Store
__pycache__/
*.py[cod]
*$py.class
```

- Lots of .gitignore files:

  https://github.com/github/gitignore

requirements.txt

- Captures dependencies for both tests and code

```
$ pip install -r requirements.txt
```

- Sample file:

```
# These requirements are for running tests, the library itself only
# needs a subset, see setup.py for that list
coverage==5.0.3
pudb==2019.2
requests==2.22.0
```

setup.py

- Lots of ways of doing packaging in Python
- setup.py interacts with many packaging and application tools:
  - **tox** -- harness for testing under multiple version of Python
  - **twine** -- tool for uploading your package to pypi.org
- RealPython article on Pipenv:

  https://realpython.com/pipenv-guide/

LICENSE

- If there is no License file copyright is default
- GitHub lets you choose when creating a new repository

```
MIT License

Copyright (c) [year] [fullname]

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated
documentation files (the "Software"), to deal in the Software without restriction, including without limitation the
rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit
persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the
Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

- Choose a license:

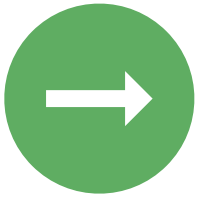
  https://choosealicense.com/

README

- Short description of your project
- If you use Markdown (.md) or reStructuredText (.rst) GitHub will show this automatically on the project's home page
- Hints on writing a good README:

  https://dbader.org/blog/write-a-great-readme-for-your-github-project

NEXT UP...



Installable Single Package

TABLE OF CONTENTS

INSTALLABLE PACKAGE

- Once you have more than one file you'll want a module
- Add a "utils.py" to the Hello World example
- Multiple files usually means multiple test files
  - Start checking test coverage
  - Run a linter

SINGLE PACKAGE APPLICATION

```
single/
├── helloworld/
│   ├── __init__.py
│   ├── helloworld.py
│   └── utils.py
├── tests/
│   ├── __init__.py
│   ├── test_helloworld.py
│   └── test_utils.py
├── runtests.sh
├── .gitignore
├── requirements.txt
├── setup.py
├── LICENSE
└── README.md
```

TEST COVERAGE

- Test coverage tells you how much of your code was exercised by the tests you wrote
- The awesome **coverage** package helps you measure this

```
$ pip install coverage
```

SAMPLE COVERAGE OUTPUT

```
$ ./runtests.sh
..
----------------------------------------------------------------------
Ran 2 tests in 0.001s
OK
================================================
Test Coverage
Name                          Stmts   Miss  Cover
-----------------------------------------------
helloworld/__init__.py            1      0   100%
helloworld/helloworld.py          8      2    75%
helloworld/utils.py               2      0   100%
tests/__init__.py                 0      0   100%
tests/test_helloworld.py          8      0   100%
tests/test_utils.py               8      0   100%
-----------------------------------------------
TOTAL                            27      2    93%

run "coverage html" for full report
```

SAMPLE COVERAGE FULL REPORT

Coverage for **helloworld/helloworld.py** : 75%

8 statements | 6 run | 2 missing | 1 excluded

```
1  #!/usr/bin/env python
2  # helloworld.py
3
4  import re
5  import requests
6
7  from helloworld.utils import show_message
8
9  URL = 'https://en.wikipedia.org/wiki/"Hello,_World!"_program'
10
11
12 def do_hello():
13     result = requests.get(URL)
14     show_message(re.findall('<title>(.*?)</title>', result.text)[0])
15
16
17 if __name__ == '__main__':
18     do_hello() # pragma: no cover
```

« index coverage.py v5.0.3, created at 2020-02-25 15:21

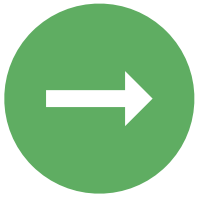
runtests.sh

```
#!/bin/bash

find . -name "*.pyc" -exec rm {} \;
coverage run -p --source=tests,helloworld -m unittest
if [ "$?" = "0" ]; then
    coverage combine
    echo -e "\n\n========================================================"
    echo "Test Coverage"
    coverage report
    echo -e "\nrun \"coverage html\" for full report"
    echo -e "\n"

    # pyflakes or its like should go here
fi
```

NEXT UP...



Larger applications with multiple modules

TABLE OF CONTENTS

LARGER APPLICATION

- Bigger apps probably mean multiple modules
  - Particularly anything GUI
- Possibly add:
  - “bin” directory
  - “data” directory
- Documentation!

SINGLE PACKAGE APPLICATION

```
app/
├── helloworld/
│   ├──__init__.py
│   ├── hello/
│   │   ├── __init__.py
│   │   ├── hello.py
│   │   └── utils.py
│   └── world/
│       ├──__init__.py
│       └── world.py
├── tests/
│   ├──__init__.py
│   ├── hello/
│   │   ├──__init__.py
│   │   ├── test_hello.py
│   │   └── test_utils.py
│   ├── world/
│   │   ├──__init__.py
│   │   └── test_world.py
```

```
app/
├── bin/
│   └── helloworld*
├── data/
│   └── translate.csv
├── docs/
│   ├── Makefile
│   ├── conf.py
│   ├── index.rst
│   ├── hello.rst
│   └── world.rst
├── runtests.sh
├──.gitignore
├── requirements.txt
├── setup.py
├── LICENSE
└── README.md
```

bin/

- Holds the programs the user will execute
- Scripts typically drop the ".py" ending
- Should have very little code logic, just a wrapper for your main module's entry point
- Can configure "setup.py" to package this if you build a wheel, script will be put on the path

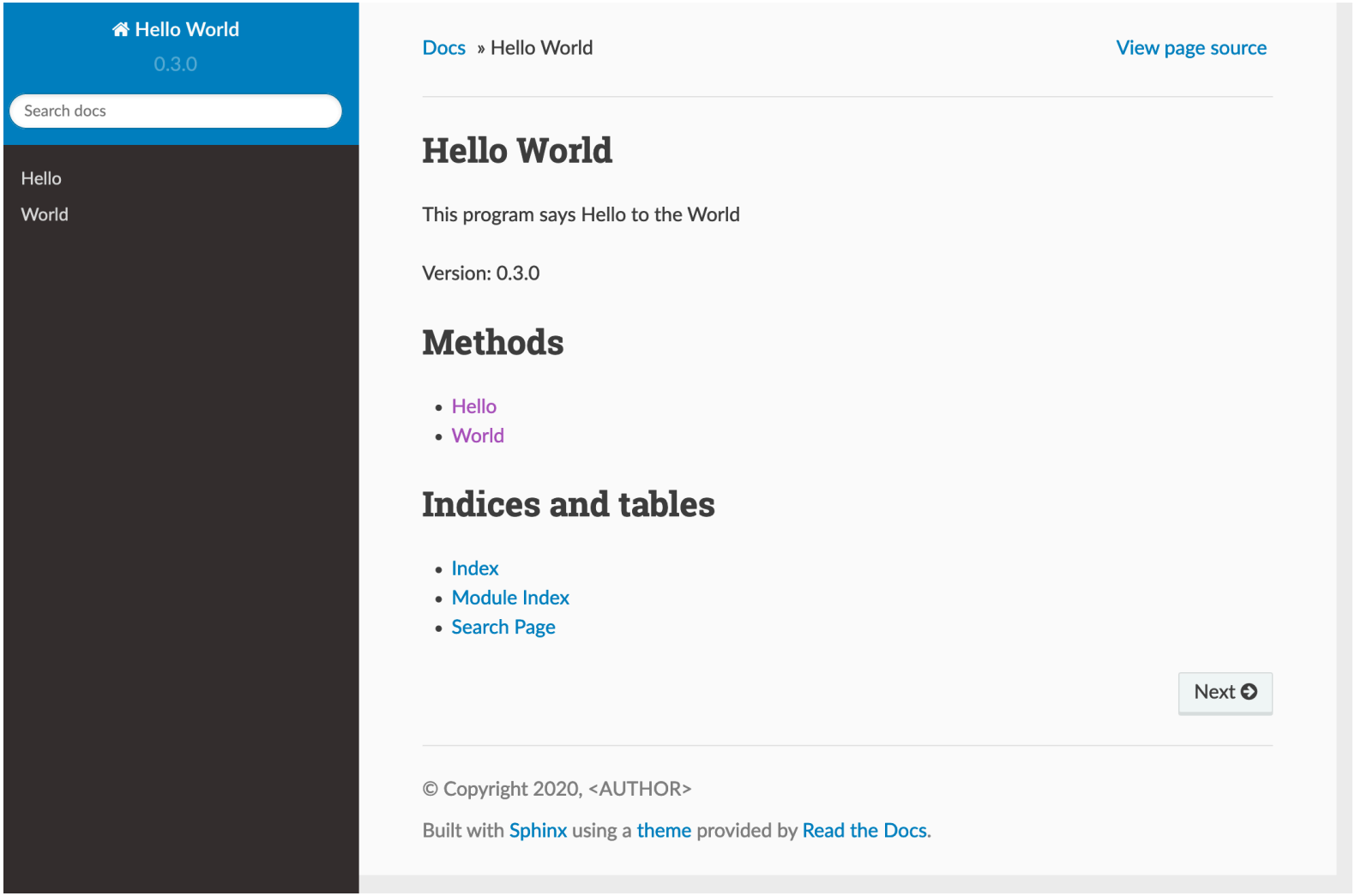
data/

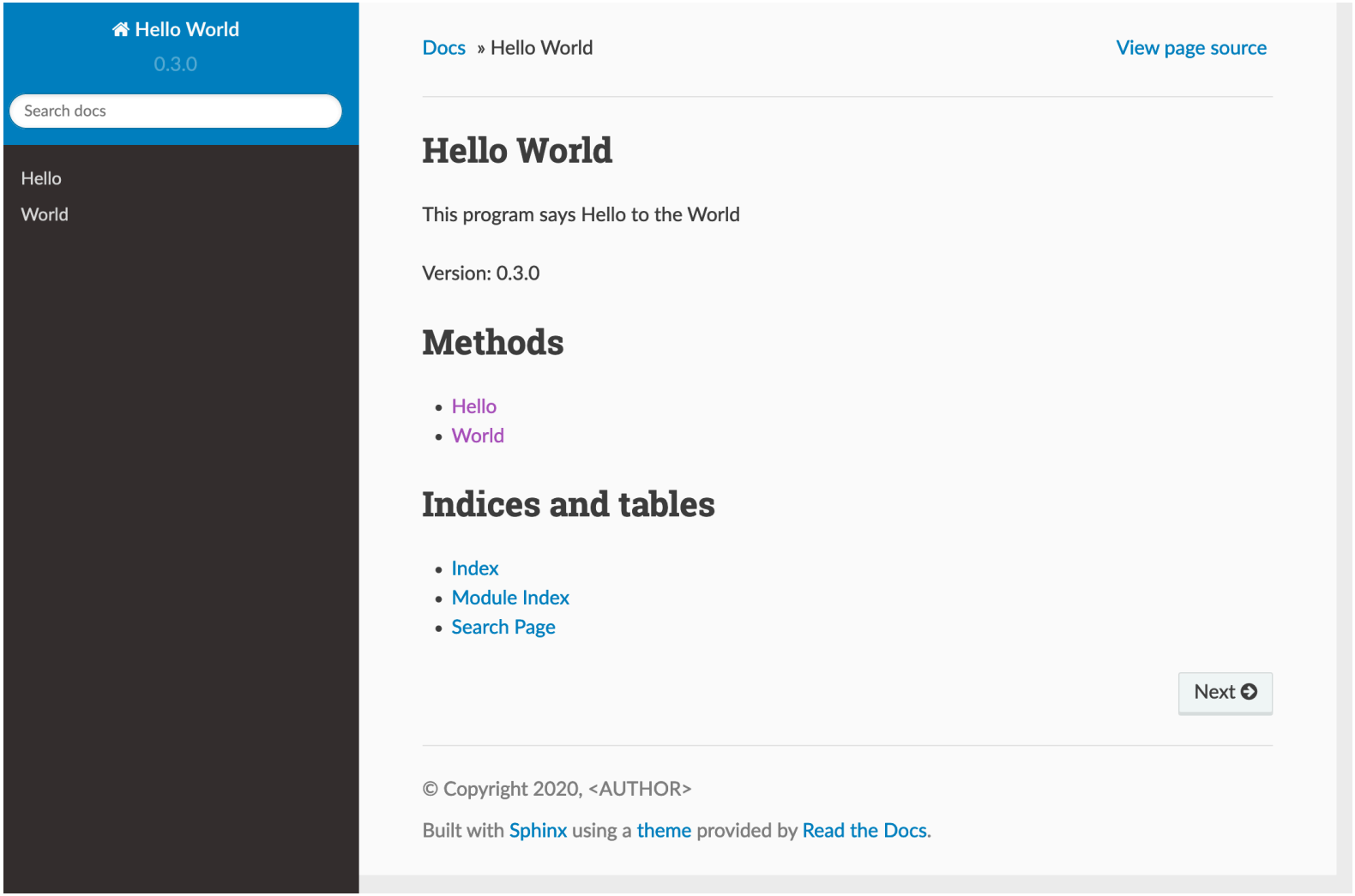
- If your program has files it loads they can go here
- Also useful for test data
- Keeps data separate from the code
- Can use multiple sub-directories for production and test data

docs/

- Documentation is often over looked and is an important part of package software
- **Sphinx** package helps you document using pydoc comments

```
$ pip install sphinx
$ pip install sphinx-rtd-theme
$ cd docs
$ sphinx-quickstart
$ make html
```


Hello World
0.3.0
Search docs
Hello
World
Docs » Hello World
View page source
Hello World
This program says Hello to the World
Version: 0.3.0
Methods
Hello
World
Indices and tables
Index
Module Index
Search Page
Next
© Copyright 2020, <AUTHOR>
Built with Sphinx using a theme provided by Read the Docs.


Hello World
0.3.0
Search docs
Hello
World
Docs » Hello World
View page source
Hello World
This program says Hello to the World
Version: 0.3.0
Methods
Hello
World
Indices and tables
Index
Module Index
Search Page
Next
© Copyright 2020, <AUTHOR>
Built with Sphinx using a theme provided by Read the Docs.

Hello World

0.3.0

Search docs

Hello

World

Docs » Hello View page source

Hello

This is the documentation on the methods in the module `hello`

helloworld.hello.hello.do_hello()

Main entry point for the program, does the look-up and translation

helloworld.hello.utils.show_message(*msg*)

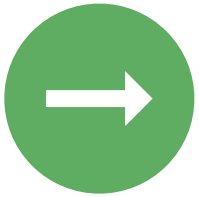
Prints the given message to the screen

Parameters

msg – message to be printed

Previous Next

NEXT UP...



Django web application

TABLE OF CONTENTS

DJANGO

- Django is opinionated about project structure
- Use the **django-admin** command to create projects

```
$ pip install Django
$ django-admin startproject django_world
```

```
django_world/
├── django_world
│   ├── __init__.py
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── manage.py
```

DJANGO APPS

- Application logic goes in a Django app, also created using **django-admin**

```
$ django-admin startapp hello
```

```
django_world/
├── hello
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── tests.py
│   └── views.py
```

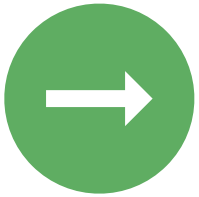
DJANGO STRUCTURE

```
django_world/
├── docs/
├── static
│   └── style.css
└── templates
    └── base.html
├── resetdb.sh
├── runserver.sh
├── django_world/
├── hello/
└── manage.py
```

MORE INFO

- Packaging for Django installable apps is different
- Django is huge:
  - https://realpython.com/courses/django-portfolio-project/
  - https://realpython.com/tutorials/django/
  - https://docs.djangoproject.com/en/3.0/intro/tutorial01/
- More on packaging choices:
  - https://django-project-skeleton.readthedocs.io/en/latest/structure.html
  - https://stackoverflow.com/questions/22841764/

NEXT UP...



Flask web application

TABLE OF CONTENTS

FLASK

- Flask is lighter weight than Django
- Less opinionated about structure
- Build useful applications in less than 10 lines of code
- Documentation includes sample application “Flaskr”

  https://flask.palletsprojects.com/en/1.1.x/tutorial/layout/

FLASKR

```
flaskr/
│
├── flaskr/
│   ├── __init__.py
│   ├── db.py
│   ├── schema.sql
│   ├── auth.py
│   ├── blog.py
│   ├── templates/
│   │   ├── base.html
│   │   ├── auth/
│   │   │   ├── login.html
│   │   │   └── register.html
│   │   └── blog/
│   │       ├── create.html
│   │       ├── index.html
│   │       └── update.html
│   └── static/
│       └── style.css
```

```
flaskr/
│
├── tests/
│   ├── conftest.py
│   ├── data.sql
│   ├── test_factory.py
│   ├── test_db.py
│   ├── test_auth.py
│   └── test_blog.py
│
├── venv/
│
├── .gitignore
├── setup.py
└── MANIFEST.in
```

MORE INFO

- Flask is also huge:

  https://realpython.com/tutorials/flask/

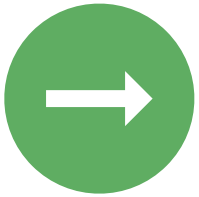
  https://palletsprojects.com/p/flask/

- Flask boilerplate:

  https://github.com/realpython/flask-boilerplate

  http://www.flaskboilerplate.com/

NEXT UP...



Conclusion and Variations

TABLE OF CONTENTS

VARIATIONS

- Python isn’t opinionated about the structure
- Desktop applications have their own intricacies
- Packaging is weak in Python
  - Lots of options
  - Changed a lot over the years

THE “src” DEBATE

- Online debate about “src” directories:
  - Because current directory is in the “sys.path”, it is easy to make mistakes
  - Using a “src” directory forces you to test installment
  - Some IDEs have problems with this structure
  - Python Packaging Authority does not use this mechanism

USING “setup.cfg”

- Instead of (or in addition to) writing a “setup.py” you can write a “setup.cfg” file
- Uses “INI” style declaration instead of a big dict
- Easier way of providing defaults when giving installers choices

LOCATION OF tests/

- Some programmers prefer to put test directories inside of the module directories
- Works fine enough
- If you don’t want to include the tests in your packaged object it means more work
- Unit vs integration tests in larger applications

WHERE TO LOCATE VIRTUAL ENVIRONMENT?

- Short answer: it depends
- Different tooling has different expectations

  https://realpython.com/pipenv-guide/

TOOLS

- Testing and Packaging:
  - **tox** -- harness for testing under multiple version of Python
  - **twine** -- tool for uploading your package to pypi.org
- Linting / Formatting:
  - **pyflakes, pylint, pychecker, pep8, flake8** -- linters / error checkers
  - **black** -- code reformatter
- Project Scaffolding:
  - **pyscaffold** -- https://pyscaffold.org
  - **cookiecutter** -- https://cookiecutter.readthedocs.io

SAMPLE CODE

Don’t forget the sample code