What makes a bunch of scripts a package?

From personal to going public!

Ellert van der Velden

ADACS Astro Hack Week 2020

“I have made a package… Now what?”

“I have made a package… Now what?”

- Please check below if you have everything you need:

“I have made a package… Now what?”

- Please check below if you have everything you need:
  - Package is in a GitHub repo;

“I have made a package… Now what?”

- Please check below if you have everything you need:
  - Package is in a GitHub repo;
  - Package is installable (setup.py);

“I have made a package… Now what?”

- Please check below if you have everything you need:
  - Package is in a GitHub repo;
  - Package is installable (setup.py);
  - Package is importable;

“I have made a package… Now what?”

- Please check below if you have everything you need:
  - Package is in a GitHub repo;
  - Package is installable (setup.py);
  - Package is importable;
  - Scripts are commented and documented;

“I have made a package… Now what?”

- Please check below if you have everything you need:
  - Package is in a GitHub repo;
  - Package is installable (setup.py);
  - Package is importable;
  - Scripts are commented and documented;
  - GitHub repo contains basic example use.

“I have made a package… Now what?”

- Please check below if you have everything you need:
  - Package is in a GitHub repo;
  - Package is installable (setup.py);
  - Package is importable;
  - Scripts are commented and documented;
  - GitHub repo contains basic example use.
- Got everything? Great! Then you can make it publicly available…

t?”

- Ple
  - Pa
  - Scrir
  - C
- Go

Have you really made a package?

Have you really made a package?

- Did you also include:

Have you really made a package?

- Did you also include:
  - Installation instructions and description of package (README.rst)?

Have you really made a package?

- Did you also include:
  - Installation instructions and description of package (README.rst)?
  - An OSI-approved license (LICENSE)?

Have you really made a package?

- Did you also include:
  - Installation instructions and description of package (README.rst)?
  - An OSI-approved license (LICENSE)?
  - List of package requirements (requirements.txt)?

Have you really made a package?

- Did you also include:
  - Installation instructions and description of package (README.rst)?
  - An OSI-approved license (LICENSE)?
  - List of package requirements (requirements.txt)?
  - Online API documentation (ReadTheDocs/GitHub Pages)?

Have you really made a package?

- Did you also include:
  - Installation instructions and description of package (README.rst)?
  - An OSI-approved license (LICENSE)?
  - List of package requirements (requirements.txt)?
  - Online API documentation (ReadTheDocs/GitHub Pages)?
  - (Automated) tests (unit tests/pytests)?

Have you really made a package?

- Did you also include:
  - Installation instructions and description of package (README.rst)?
  - An OSI-approved license (LICENSE)?
  - List of package requirements (requirements.txt)?
  - Online API documentation (ReadTheDocs/GitHub Pages)?
  - (Automated) tests (unit tests/pytests)?
  - Other things I have forgotten about…?

Code requirements

Code requirements

- Python is designed to be an open-source programming language;

Code requirements

- Python is designed to be an open-source programming language;
- Code sharing and recycling is encouraged;

Code requirements

- Python is designed to be an open-source programming language;
- Code sharing and recycling is encouraged;
- So, your code probably relies on a few (or more) third-party packages.

Code requirements

```
colorspacious>=1.1.0
matplotlib>=2.2.4
numpy>=1.8
six>=1.10.0
```

Code requirements

```
colorspacious>=1.1.0
matplotlib>=2.2.4
numpy>=1.8
six>=1.10.0
```

```
cmasher>=1.0.5
e13tools>=0.6.17
h5py>=2.8.0
hickle>=3.4.0
matplotlib>=2.2.4
mlxtend>=0.16.0
mpi4pyd>=0.2.4
numpy>=1.12.0
pyqt5==5.12.*
qtpy>=1.9.0
scikit-learn>=0.22
scipy>=1.0.0
sortedcontainers>=1.5.9
threadpoolctl>=1.0.0
tqdm>=4.7.6
```

Code requirements

- Two common mistakes:

```
colorspacious>=1.1.0
matplotlib>=2.2.4
numpy>=1.8
six>=1.10.0
```

```
cmasher>=1.0.5
e13tools>=0.6.17
h5py>=2.8.0
hickle>=3.4.0
matplotlib>=2.2.4
mlxtend>=0.16.0
mpi4pyd>=0.2.4
numpy>=1.12.0
pyqt5==5.12.*
qtpy>=1.9.0
scikit-learn>=0.22
scipy>=1.0.0
sortedcontainers>=1.5.9
threadpoolctl>=1.0.0
tqdm>=4.7.6
```

Code requirements

- Two common mistakes:
  - Not specifying all requirements;

```
colorspacious>=1.1.0
matplotlib>=2.2.4
numpy>=1.8
six>=1.10.0
```

```
cmasher>=1.0.5
e13tools>=0.6.17
h5py>=2.8.0
hickle>=3.4.0
matplotlib>=2.2.4
mlxtend>=0.16.0
mpi4pyd>=0.2.4
numpy>=1.12.0
pyqt5==5.12.*
qtpy>=1.9.0
scikit-learn>=0.22
scipy>=1.0.0
sortedcontainers>=1.5.9
threadpoolctl>=1.0.0
tqdm>=4.7.6
```

Code requirements

- Two common mistakes:
  - Not specifying all requirements;
    - Requirement is very common, like NumPy;

```
colorspacious>=1.1.0
matplotlib>=2.2.4
numpy>=1.8
six>=1.10.0
```

```
cmasher>=1.0.5
e13tools>=0.6.17
h5py>=2.8.0
hickle>=3.4.0
matplotlib>=2.2.4
mlxtend>=0.16.0
mpi4pyd>=0.2.4
numpy>=1.12.0
pyqt5==5.12.*
qtpy>=1.9.0
scikit-learn>=0.22
scipy>=1.0.0
sortedcontainers>=1.5.9
threadpoolctl>=1.0.0
tqdm>=4.7.6
```

Code requirements

- Two common mistakes:
  - Not specifying all requirements;
    - Requirement is very common, like NumPy;
    - Requirement is statisfied by another requirement.

```
colorspacious>=1.1.0
matplotlib>=2.2.4
numpy>=1.8
six>=1.10.0
```

```
cmasher>=1.0.5
e13tools>=0.6.17
h5py>=2.8.0
hickle>=3.4.0
matplotlib>=2.2.4
mlxtend>=0.16.0
mpi4pyd>=0.2.4
numpy>=1.12.0
pyqt5==5.12.*
qtpy>=1.9.0
scikit-learn>=0.22
scipy>=1.0.0
sortedcontainers>=1.5.9
threadpoolctl>=1.0.0
tqdm>=4.7.6
```

Code requirements

- Two common mistakes:
  - Not specifying all requirements;
    - Requirement is very common, like NumPy;
    - Requirement is statisfied by another requirement.

This is wrong as you cannot guarantee these assumptions.

```
colorspacious>=1.1.0
matplotlib>=2.2.4
numpy>=1.8
six>=1.10.0
```

```
cmasher>=1.0.5
e13tools>=0.6.17
h5py>=2.8.0
hickle>=3.4.0
matplotlib>=2.2.4
mlxtend>=0.16.0
mpi4pyd>=0.2.4
numpy>=1.12.0
pyqt5==5.12.*
qtpy>=1.9.0
scikit-learn>=0.22
scipy>=1.0.0
sortedcontainers>=1.5.9
threadpoolctl>=1.0.0
tqdm>=4.7.6
```

Code requirements

- Two common mistakes:
  - Not specifying all requirements;
  - Not specifying minimum required versions.

```
colorspacious>=1.1.0
matplotlib>=2.2.4
numpy>=1.8
six>=1.10.0
```

```
cmasher>=1.0.5
e13tools>=0.6.17
h5py>=2.8.0
hickle>=3.4.0
matplotlib>=2.2.4
mlxtend>=0.16.0
mpi4pyd>=0.2.4
numpy>=1.12.0
pyqt5==5.12.*
qtpy>=1.9.0
scikit-learn>=0.22
scipy>=1.0.0
sortedcontainers>=1.5.9
threadpoolctl>=1.0.0
tqdm>=4.7.6
```

Code requirements

- Two common mistakes:
  - Not specifying all requirements;
  - Not specifying minimum required versions.
- Both are annoying and tedious to deal with as a user, especially the latter.

```
colorspacious>=1.1.0
matplotlib>=2.2.4
numpy>=1.8
six>=1.10.0
```

```
cmasher>=1.0.5
e13tools>=0.6.17
h5py>=2.8.0
hickle>=3.4.0
matplotlib>=2.2.4
mlxtend>=0.16.0
mpi4pyd>=0.2.4
numpy>=1.12.0
pyqt5==5.12.*
qtpy>=1.9.0
scikit-learn>=0.22
scipy>=1.0.0
sortedcontainers>=1.5.9
threadpoolctl>=1.0.0
tqdm>=4.7.6
```

Online API documentation

CMasher: Scientific colormaps for making accessible, informative and *cmashing* plots

This is the documentation for the *CMasher* package, a collection of scientific colormaps for making accessible, informative and *cmashing* plots. It is written in pure Python 2/3 and publicly available on GitHub.

The documentation of *CMasher* is spread out over several sections:

- User Documentation
- API Reference

User Documentation

- Introduction
  - Description
  - Colormap overview
  - How to install
  - Example use
  - Using custom colormaps
- Background
- Sequential
  - Amber
  - Apple
  - Arctic
  - Chroma
  - Dusk
  - Eclipse
  - Ember
  - Flamingo

CMasher

Scientific colormaps for making accessible, informative and *cmashing* plots.

`cmasher.create_cmap_overview`(*cmaps=None, savefig=None, use_types=True, sort='alphabetical'*) [source]

Creates an overview plot containing all colormaps defined in the provided *cmaps*.

Other Parameters:

- **cmaps** (list of {str; `Colormap` objects}, dict of lists or None. Default: None) – A list of all colormaps that must be included in the overview plot. If dict of lists, the keys define categories for the colormaps. If *None*, all colormaps defined in *CMasher* are used instead.
- **savefig** (*str or None. Default: None*) – If not *None*, the path where the overview plot must be saved to. Else, the plot will simply be shown.
- **use_types** (*bool. Default: True*) – Whether all colormaps in *cmaps* should be categorized into their colormap types (sequential; diverging; cyclic; qualitative; misc). If *cmaps* is a dict, this value is ignored.
- **sort** (*{'alphabetical'/'name'; 'lightness'}. Default: 'alphabetical'*) – String indicating how the colormaps should be sorted in the overview. If 'alphabetical', the colormaps are sorted alphabetically on their name. If 'lightness', the colormaps are sorted on their starting lightness and their lightness range.

Note

The colormaps in *cmaps* can either be provided as their registered name in *MPL*, or their corresponding `Colormap` object. Any provided reversed colormaps (colormaps that end their name with '_r') are ignored.

`cmasher.import_cmaps`(*cmap_path*) [source]

Reads in custom colormaps from a provided file or directory *cmap_path*; transforms them into

naps for making
ıd *cmashing* plots

ge, a collection of scientific colormaps for making
ritten in pure Python 2/3 and publicly available on

r several sections:

- Dusk
- Eclipse
- Ember
- Flamingo

CMasher

Scientific colormaps for making accessible, informative a

`cmasher.create_cmap_overview`(*cmaps=None, savefig=Non*
[source]

Creates an overview plot containing all colormaps de

Other Parameters:

- **cmaps** (list of {str; `Colormap` objects}, dict of list
colormaps that must be included in the overvie
categories for the colormaps. If *None*, all colorn
- **savefig** (*str or None. Default: None*) – If not *None*
saved to. Else, the plot will simply be shown.
- **use_types** (*bool. Default: True*) – Whether all co
their colormap types (sequential; diverging; cyc
value is ignored.
- **sort** ({*'alphabetical'/'name'; 'lightness'}. Default: '*
colormaps should be sorted in the overview. If
alphabetically on their name. If 'lightness', the c
lightness and their lightness range.

Note

The colormaps in *cmaps* can either be provided as t
corresponding `Colormap` object. Any provided rever
name with '_r') are ignored.

`cmasher.import_cmaps`(*cmap_path*) [source]

Reads in custom colormaps from a provided file or di

CMasher

Search docs

USER DOCUMENTATION

Introduction

Background

Sequential

Amber

Apple

Arctic

Chroma

Dusk

Eclipse

Ember

Flamingo

Freeze

Gem

Gothic

Heat

Horizon

Jungle

Lavender

Neutral

Nuclear

Rainforest

Sunburst

Voltage

Diverging

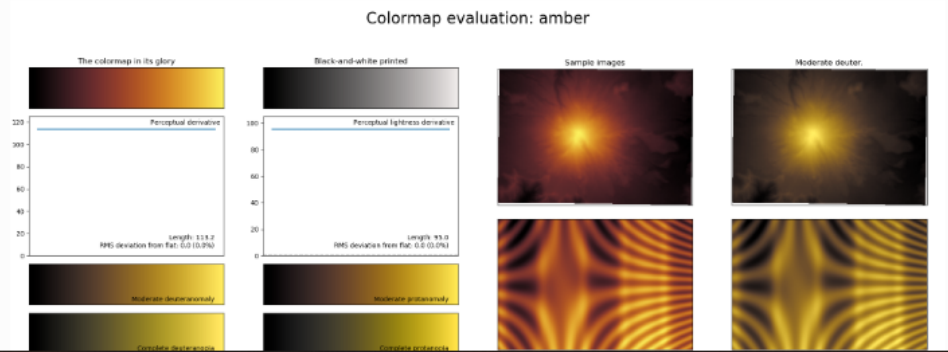
Community guidelines

Read the Docs v: latest

Sequential

Sequential colormaps (that are perceptually uniform of course) are basic colormaps that start at a reasonably low lightness value and uniformly increase to a higher value. They are commonly used to represent information that is ordered. The *matplotlib* package already has a few great sequential colormaps readily available for the user, mainly the colormaps named *viridis*; *plasma*; *inferno*; *magma*; and *cividis*. However, three of these colormaps use the color red as their main color and none of them uses the full lightness range. As it might sometimes be desirable to use a different main color or maximize the lightness range of the colormap, *CMasher* provides a few sequential colormaps that do exactly that. These colormaps are shown below.

Amber



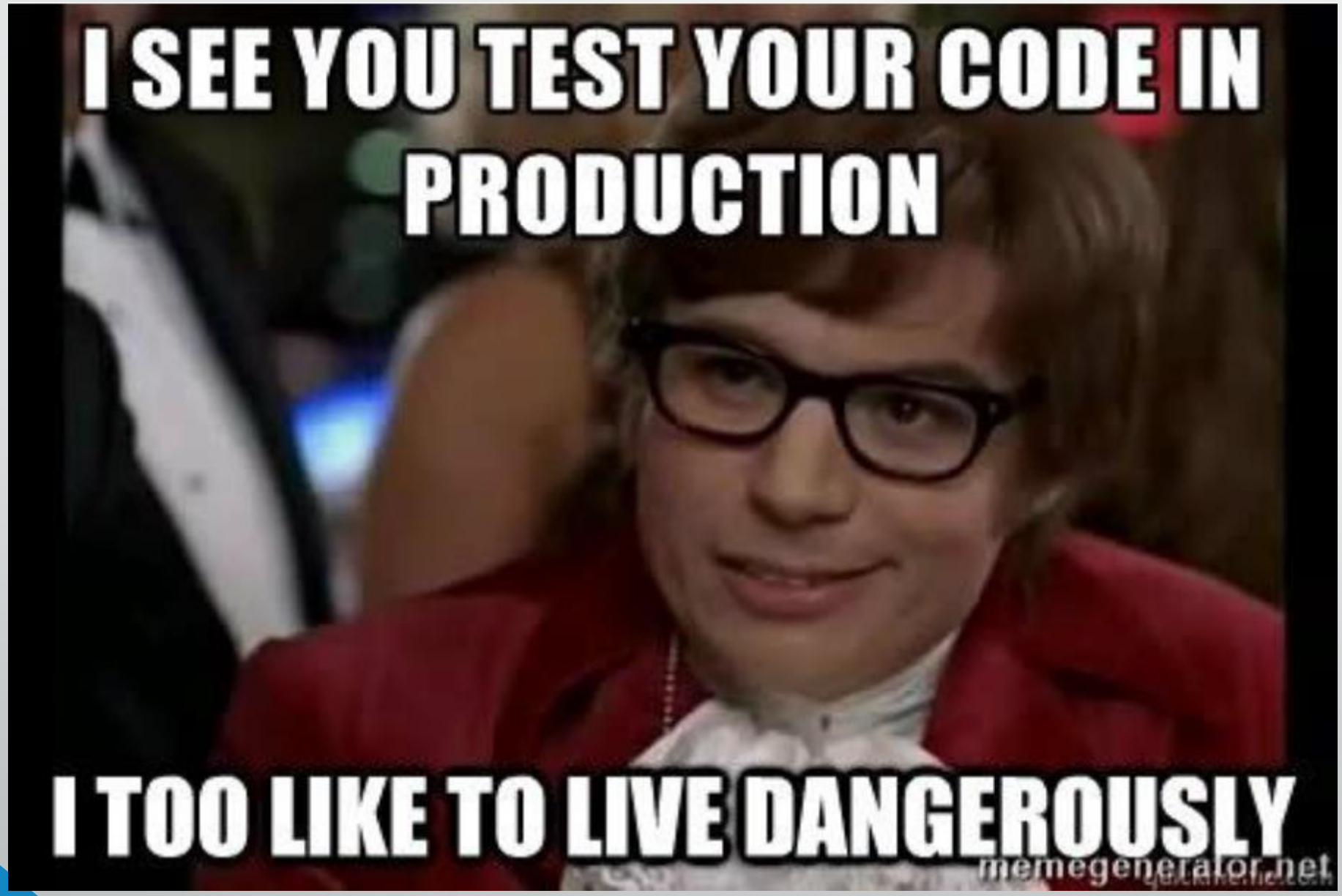
Testing: Why should I do it?

Testing: Why should I do it?

- You can convince yourself that the code you just wrote works properly.

Testing: Tests are like scientific experiments

Experiment	Tests
Setup your experiment (environment, apparatus, etc)	Put your system under test in a known state
Run the experiment	Execute the test on your system
Analyse the results • Did you get what was expected?	Validate that your result is what you expected
Repeat experiment, perhaps with different parameters	Repeat, perhaps with different states


I SEE YOU TEST YOUR CODE IN PRODUCTION
I TOO LIKE TO LIVE DANGEROUSLY
memegenerator.net

Automated testing: What are they?

- “Repeatable experiments (tests) that have been codified such that they can be executed on or by a computer.”.

Automated testing: Why should I do it?

Automated testing: Why should I do it?

- Modifying code equals introducing “new features”;

Automated testing: Why should I do it?

- Modifying code equals introducing “new features”;
- Humans are lazy! (Read: Manual testing is tedious);

Automated testing: Why should I do it?

- Modifying code equals introducing “new features”;
- Humans are lazy! (Read: Manual testing is tedious);
- Manual testing the exact same way every time is even more tedious;

Automated testing: Why should I do it?

- Modifying code equals introducing “new features”;
- Humans are lazy! (Read: Manual testing is tedious);
- Manual testing the exact same way every time is even more tedious;
- Open-source projects are used by other people, who may require maintaining or testing your code;

Automated testing: Why should I do it?

- Modifying code equals introducing “new features”;
- Humans are lazy! (Read: Manual testing is tedious);
- Manual testing the exact same way every time is even more tedious;
- Open-source projects are used by other people, who may require maintaining or testing your code;
- Future you is a different person, so this includes you as well.

Automated testing: Benefits

Automated testing: Benefits

- It helps to document the expected behaviour of your code;

Automated testing: Benefits

- It helps to document the expected behaviour of your code;
- Tests are always ran the same way;

Automated testing: Benefits

- It helps to document the expected behaviour of your code;
- Tests are always ran the same way;
- Early detection of ~~bugs~~, new features;

Automated testing: Benefits

- It helps to document the expected behaviour of your code;
- Tests are always ran the same way;
- Early detection of ~~bugs~~, new features;
- Feedback is provided in a faster fashion;

Automated testing: Benefits

- It helps to document the expected behaviour of your code;
- Tests are always ran the same way;
- Early detection of ~~bugs~~, new features;
- Feedback is provided in a faster fashion;
- It is harder to forget to test certain situations;

Automated testing: Benefits

- It helps to document the expected behaviour of your code;
- Tests are always ran the same way;
- Early detection of ~~bugs~~, new features;
- Feedback is provided in a faster fashion;
- It is harder to forget to test certain situations;
- Automated tests can be run anytime and anywhere (e.g., during a morning break).

Automated testing: How to write them?

Automated testing: How to write them?

- Use manual testing for testing **new** code;

Automated testing: How to write them?

- Use manual testing for testing **new** code;
- If you can manually test a piece of code, then it can be automated;

Automated testing: How to write them?

- Use manual testing for testing **new** code;
- If you can manually test a piece of code, then it can be automated;
- Convert manual test to automated test;

Automated testing: How to write them?

- Use manual testing for testing **new** code;
- If you can manually test a piece of code, then it can be automated;
- Convert manual test to automated test;
- Profit!

What makes a bunch of scripts a package?

What makes a bunch of scripts a package?

- Depends on:
  - Accessibility;

What makes a bunch of scripts a package?

- Depends on:
  - Accessibility;
  - Installability;

What makes a bunch of scripts a package?

- Depends on:
  - Accessibility;
  - Installability;
  - Usability;

What makes a bunch of scripts a package?

- Depends on:
  - Accessibility;
  - Installability;
  - Usability;
  - Readability;

What makes a bunch of scripts a package?

- Depends on:
  - Accessibility;
  - Installability;
  - Usability;
  - Readability;
  - Reliability;

What makes a bunch of scripts a package?

- Depends on:
  - Accessibility;
  - Installability;
  - Usability;
  - Readability;
  - Reliability;
  - And all the other Python ‘abilities’.

Coming up: How to be even more lazy!