

Python Tutorial Series 2019

Basics 3: Packages, Environments and Python Versions

Bruno C. Quint - bquint@gemini.edu

Kenneth Anderson - kanderso@gemini.edu

SUSD - GEMINI South, La Serena, Chile



Previously...

Basics I

- What is Python?
- What will you need?
- Python as a terminal
- Python as a script
- Types of variables

1.
[x,y]
"abc"

Basics II

- Code Flow
- Functions
- Classes
 - Methods
 - Attributes
- Sub-Classes

class
def
self
if
else
for
while



Basics 3 – Table of Contents

- (Re)using code
 - Modules
 - Packages
- Python2 to Python3
- External packages
- Virtual Environments
- Conda and Anaconda

`--main--`

`--init--`

`setup.py`

`pip`

`conda`



(Re)using code

modules

```
#!/usr/bin/env python
""" This module, called my_module.py, contains a function
that simulates a six side dice roll """

import random

def roll_dice():
    _min = 1
    _max = 6
    result = random.randint(_min, _max)
    return result
```



(Re)using code

modules

```
#!/usr/bin/env python
""" This module, called my_module.py, contains a function
that simulates a six side dice roll """

import random

def roll_dice():
    _min = 1
    _max = 6
    result = random.randint(_min, _max)
    return result
```

```
In [1]: import my_module
```

```
In [2]: print(my_module.roll_dice())
2
```

```
In [3]: print(my_module.roll_dice())
5
```



(Re)using code

script

```
#!/usr/bin/env python
""" This module, called my_module.py, contains a function
that simulates a six side dice roll """

import random

def roll_dice():
    _min = 1
    _max = 6
    result = random.randint(_min, _max)
    return result

print('The dice gave me {}'.format(roll_dice()))
```



(Re)using code

script

```
#!/usr/bin/env python
""" This module, called my_module.py, contains a function
that simulates a six side dice roll """

import random

def roll_dice():
    _min = 1
    _max = 6
    result = random.randint(_min, _max)
    return result

print('The dice gave me {}')
```

```
In [1]: import my_module
         .format(roll_dice())
The dice gave me 6
```

```
In [2]: print(my_module.roll_dice())
3
```



(Re)using code

Special Variables

`__myvariable__`

Special variables start and end with double underscore

`__name__`

Stores the name of the scope being run.

`__main__`

Name of the top level scope in Python



(Re)using code

Special Variables

`__myvariable__`

Special variables start and end with double underscore

```
In [1]: print(__name__)
__main__
```

```
In [2]: import my_module
The dice gave me 2
```

```
In [3]: print(my_module.__name__)
my_module
```

`__main__` Name of the top level scope
in Python



(Re)using code

script

```
#!/usr/bin/env python
""" This module, called my_module.py, contains a function
that simulates a six side dice roll """

import random

def roll_dice():
    _min = 1
    _max = 6
    result = random.randint(_min, _max)
    return result

print('The dice gave me {}'.format(roll_dice()))
```



(Re)using code

script

```
#!/usr/bin/env python
""" This module, called my_module.py, contains a function
that simulates a six side dice roll """

import random

def roll_dice():
    _min = 1
    _max = 6
    result = random.randint(_min, _max)
    return result

print('Current scope: {}'.format(__name__))
print('The dice gave me {}'.format(roll_dice()))
```



(Re)using code

script

```
#!/usr/bin/env python
""" This module, called my_module.py, contains a function
that simulates a six side dice roll """

import random

def roll_dice():
    _min = 1
    _max = 6
    result = random.randint(_min, _max)
    return result

print('Current scope: {}'.format(__name__))
print('The dice gave me {}'.format(roll_dice()))
```

```
$ python my_module.py
Current scope: __main__
The dice gave me 1
```



(Re)using code

script

```
#!/usr/bin/env python
""" This module, called my_module.py, contains a function
that simulates a six side dice roll """

import random

def roll_dice():
    _min = 1
    _max = 6
    result = random.randint(_min, _max)
    return result

if __name__ == '__main__':
    print('The dice gave me {}'.format(roll_dice()))
```



(Re)using code

script

```
#!/usr/bin/env python
""" This module, called my_module.py, contains a function
that simulates a six side dice roll """

import random

def roll_dice():
    _min = 1
    _max = 6
    result = random.randint(_min, _max)
    return result

if __name__ == '__main__':
    print('The dice gave me {}'.format(roll_dice()))
```

```
In [1]: import my_module
In [2]: print(my_module.roll_dice())
4
```



(Re)using code good practice

```
#!/usr/bin/env python
""" This module, called my_module.py, contains a function
that simulates a six side dice roll """

import random

def roll_dice():
    _min = 1
    _max = 6
    result = random.randint(_min, _max)
    return result

if __name__ == '__main__':
    print('The dice gave me {}'.format(roll_dice()))
```



(Re)using code good practice

```
#!/usr/bin/env python
""" This module, called my_module.py, contains a function
that simulates a six side dice roll """

import random

def main():
    print('The dice gave me {}'.format(roll_dice()))

def roll_dice():
    _min = 1
    _max = 6
    result = random.randint(_min, _max)
    return result

if __name__ == '__main__':
    main()
```



(Re)using code

my_project/

packages



(Re)using code

packages

```
my_project/  
  └ my_package/
```



(Re)using code

packages

```
my_project/  
  └ my_package/  
    └ __init__.py
```



(Re)using code

packages

my_project/

└ my_package/

- └ __init__.py
- └ first_module.py
- └ second_module.py



(Re)using code

packages

```
my_project/
└── my_package/
    ├── __init__.py
    ├── first_module.py
    └── second_module.py
```

```
In [1]: import my_package
```

```
In [2]: my_package.first_module.my_function()
```

```
In [1]: from my_package import first_module
```

```
In [2]: first_module.my_function()
```



(Re)using code

packages

my_project/

 └ my_package/

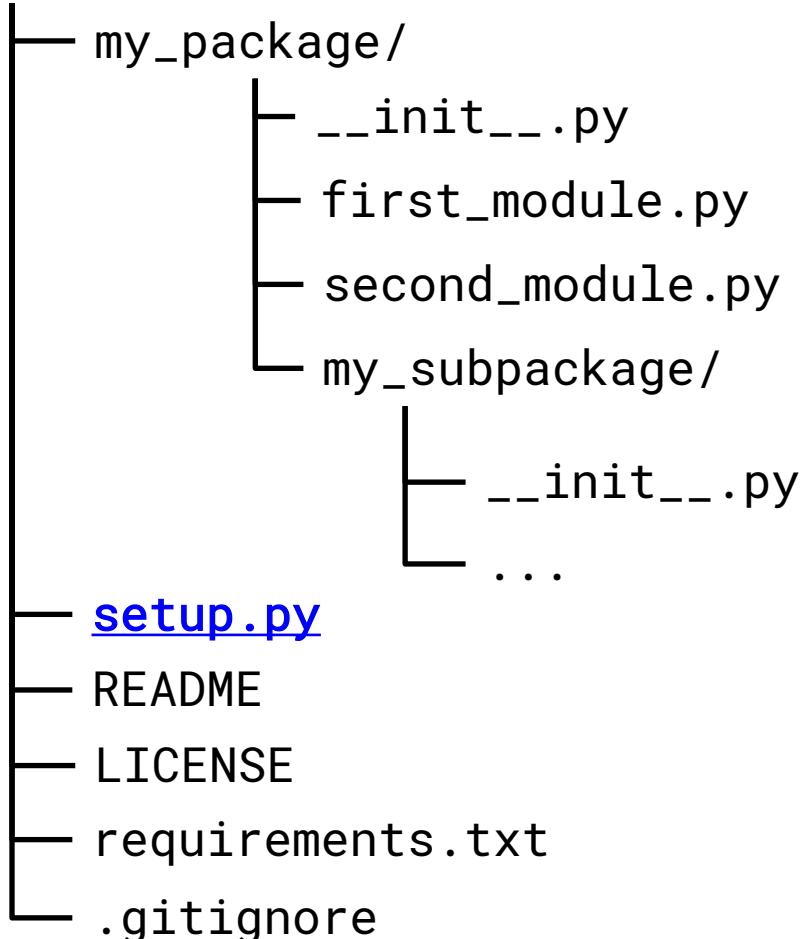
- ├ __init__.py
- ├ first_module.py
- ├ second_module.py
- └ my_subpackage/
 - └ __init__.py
 - ...



(Re)using code

packages

my_project/





(Re)using code

packages

```
my_project/
    └── my_package/
        ├── __init__.py
        ├── first_module.py
        ├── second_module.py
        └── ...
    ├── docs/
    ├── tests/
    ├── bin/
    ├── setup.py
    ├── README
    ├── LICENSE
    ├── requirements.txt
    └── .gitignore
```



(Re)using code

packages

my_project/

```
  └── my_package/
        ├── __init__.py
        ├── first_module.py
        ├── second_module.py
        └── my_subpackage/
            ├── __init__.py
            └── ...
  └── setup.py
  └── README
  └── LICENSE
  └── requirements.txt
  └── .gitignore
```

setup.py

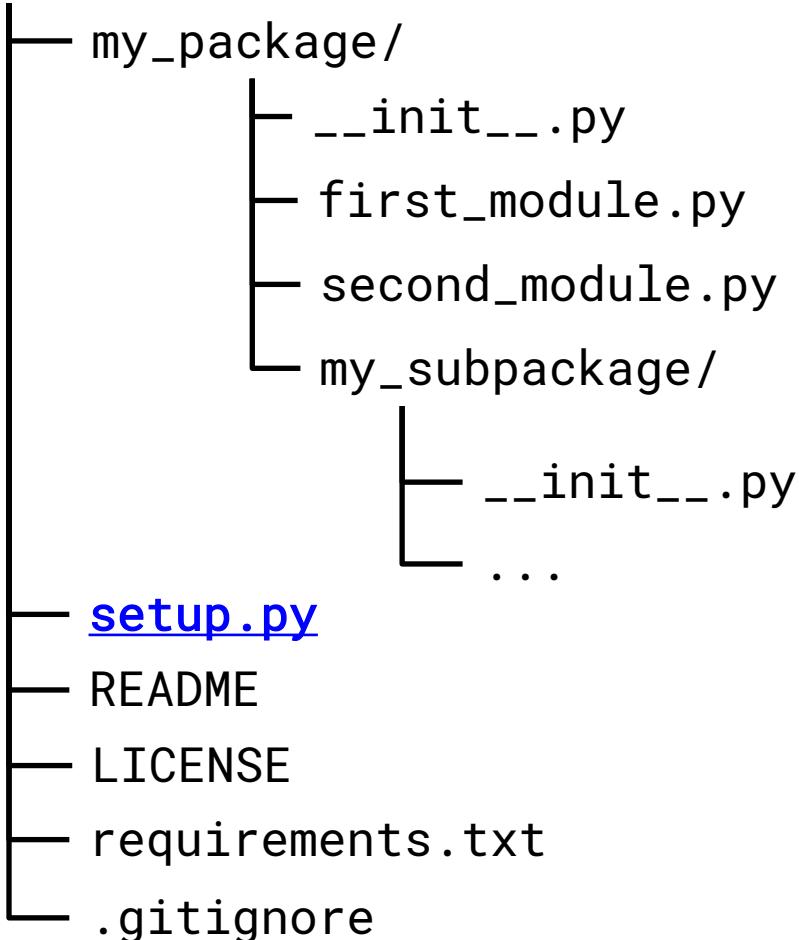
```
$ python setup.py install
```



(Re)using code

packages

my_project/



setup.py

```
$ python setup.py install
```

pip

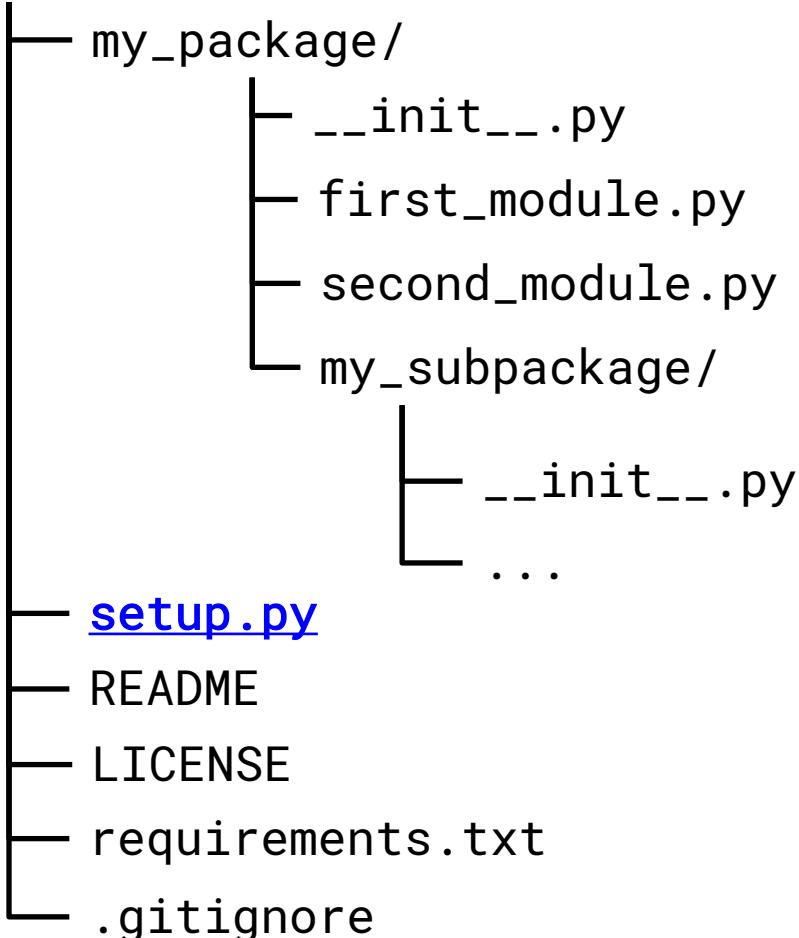
```
$ pip install .
```



(Re)using code

packages

my_project/



setup.py

```
$ python setup.py install
```

pip

editable
\$ pip install -e .
good for development



(Re)using code

packages

my_project/

```
  └── my_package/
        ├── __init__.py
        ├── first_module.py
        ├── second_module.py
        └── my_subpackage/
            ├── __init__.py
            └── ...
  └── setup.py
  └── README
  └── LICENSE
  └── requirements.txt
  └── .gitignore
```

setup.py

\$ python setup.py install

pip

\$ pip --help



Py2/3 Conversion

With Python 2.x EoS slated for end of this year (2019), it is becoming increasingly important to move existing code to Python 3. There are some big differences between Python 2.x and Python 3.x., but there are resources available to help developers move to Py3.

- 2to3 – available directly with your Py3 conda environment.
`$ which 2to3
~/anaconda2/envs/dragons36/bin/2to3`
- Porting Code to Python 3 with 2to3 (Dive into Python).
<https://www.diveinto.org/python3/porting-code-to-python-3-with-2to3.html>
- Python docs also provides help on 2to3.<https://docs.python.org/3.7/library/2to3.html>



Py2/3 Conversion

Things to be prepared for:

print statement – not a “statement” but now a function.

e.g., `print("foo", "bar")` → `print("foo", "bar")`

strings – Py2: unicode, string, and byte

Py3: unicode, byte (all strings in Py3 are unicode).

long int – Py2 only. There is *only one* `int` type in Py3.

dict – Py3, methods on dictionaries return “views”, which are not usually what you want. To maintain the iterator behaviour of `.items()`, `.values()`, etc., methods just wrap in a list.

`a_dict.keys()` → `list(a_dict.keys())`



Py2/3 Conversion

Python 2>3 (cont'd)

More on dict types – method `.has_key()` is gone on Py3. Use nominal Python idiom “in”. E.g.,

`a_dict.has_key('foobar')` → ‘foobar’ in `a_dict`

In Python 3, several modules have been reorganized, and/or eliminated.

Notables:

http libraries

urllib libraries

(dbm libraries)

For example,

`urlparse` → `urllib.parse`

Py2 division is replaced. Use “//” for integer division. “/” is purely float div.

py2: `1/3` → 0

py3: `1/3` → 0.3333333... `1//3` → 0



Py2/3 Conversion

Demo Time

Run 2to3 against some existing scripts.



External Packages

pip

Default package manager

```
$ pip --help
```

Search

```
$ pip search <package_name>
```

Install

```
$ pip install scipy
```

Upgrade

```
$ pip install scipy --upgrade
```

List

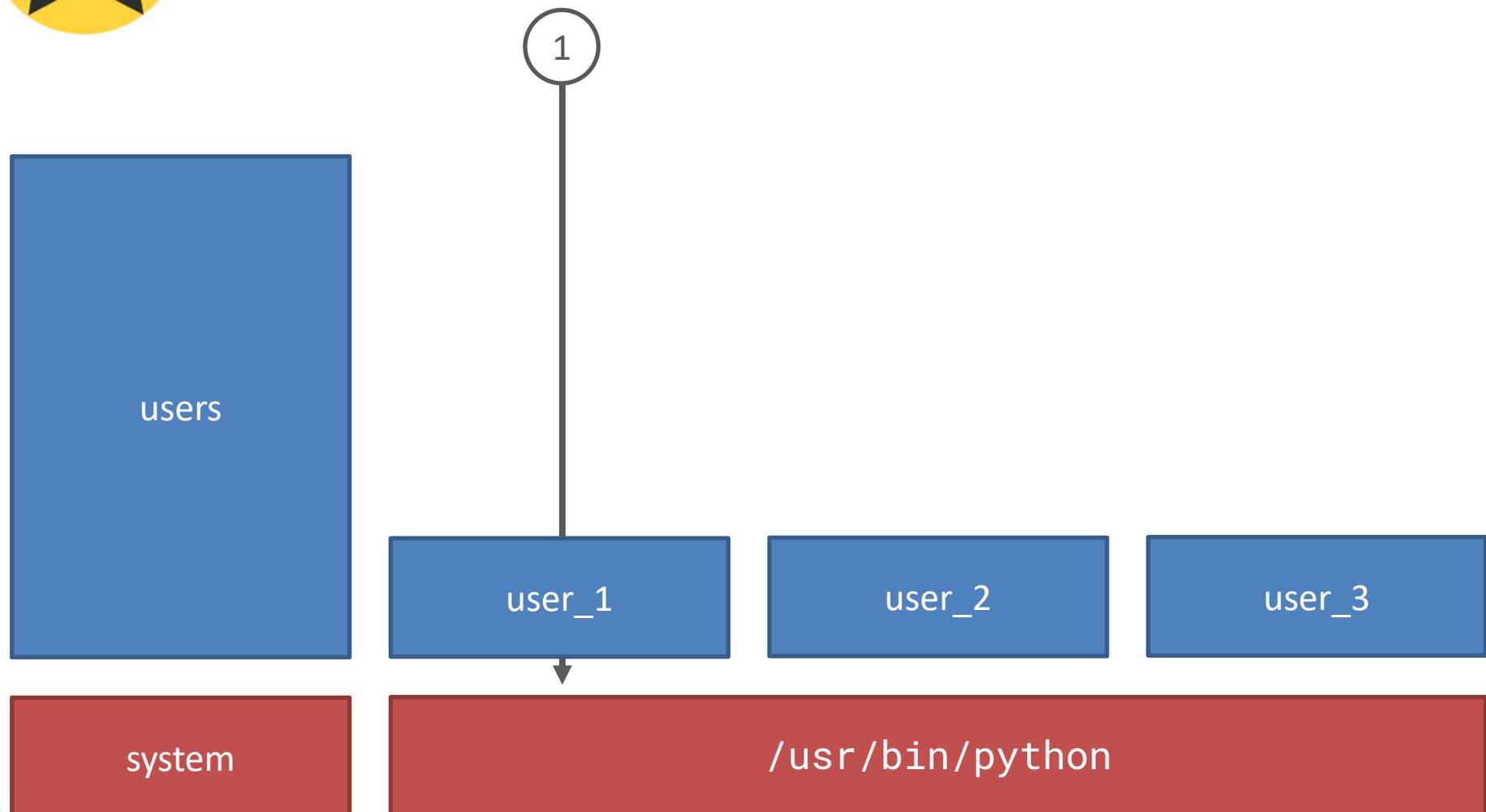
```
$ pip list
```

Uninstall

```
$ pip uninstall pyfits
```

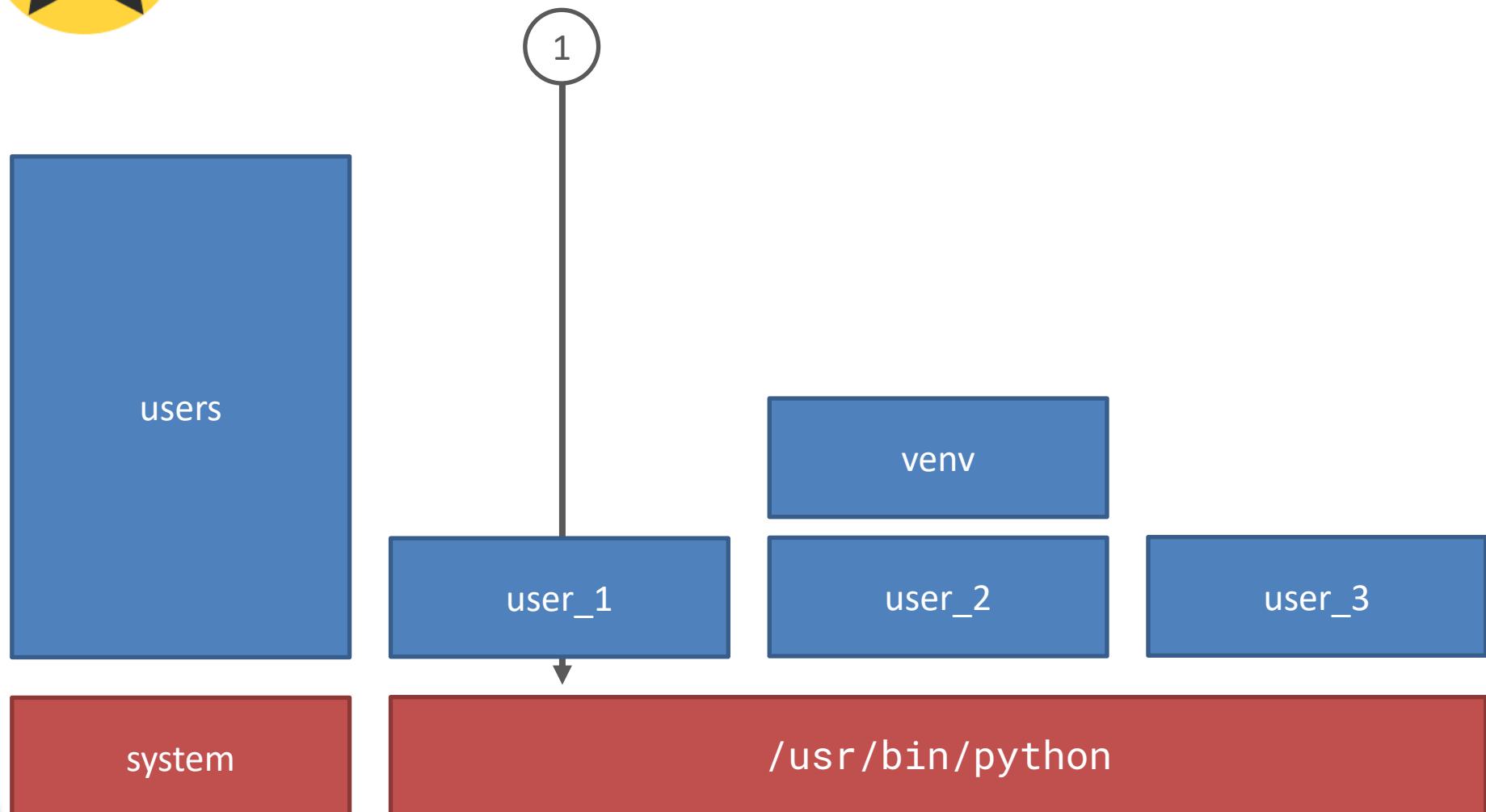


Virtual Environments



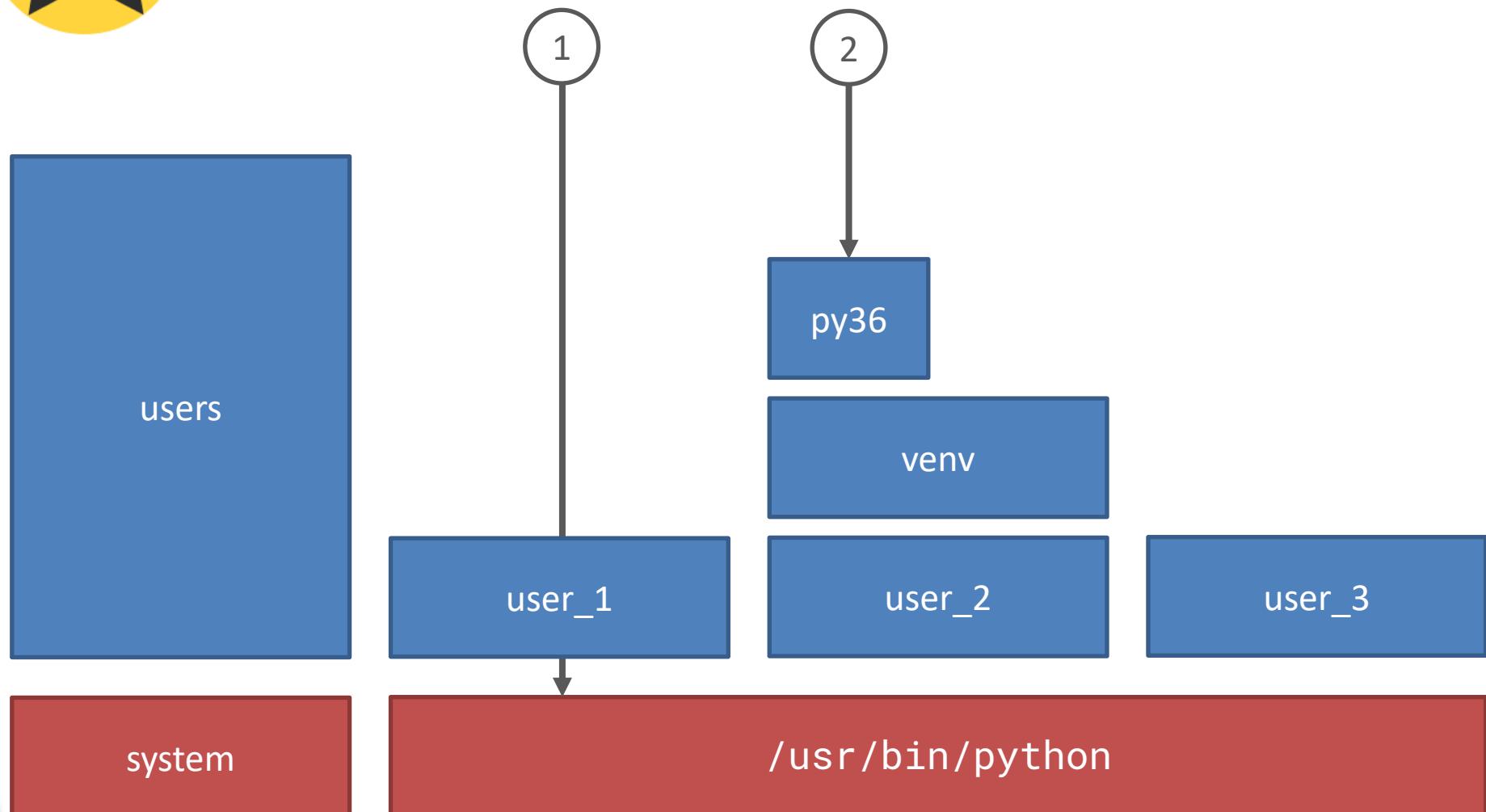


Virtual Environments



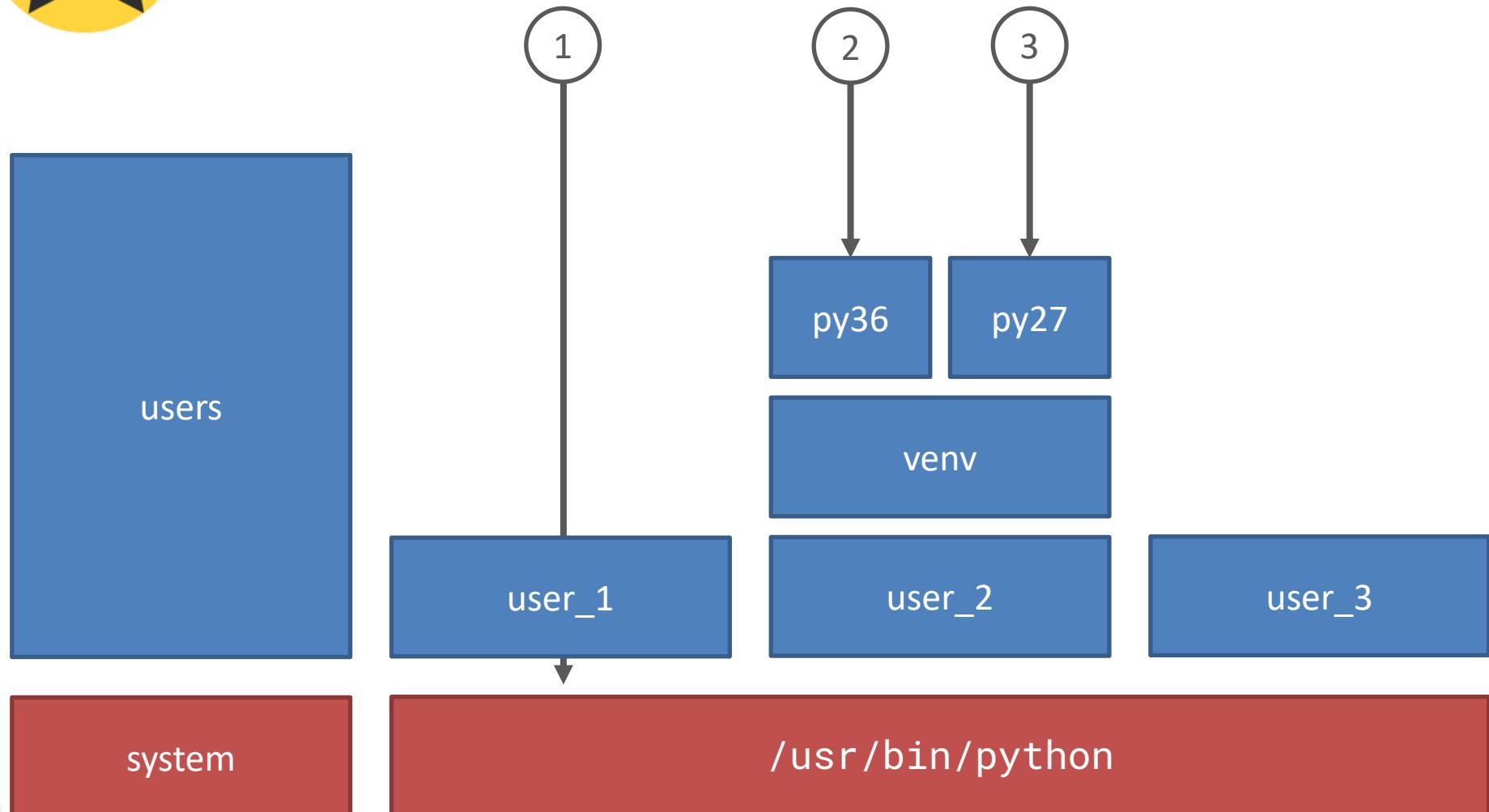


Virtual Environments



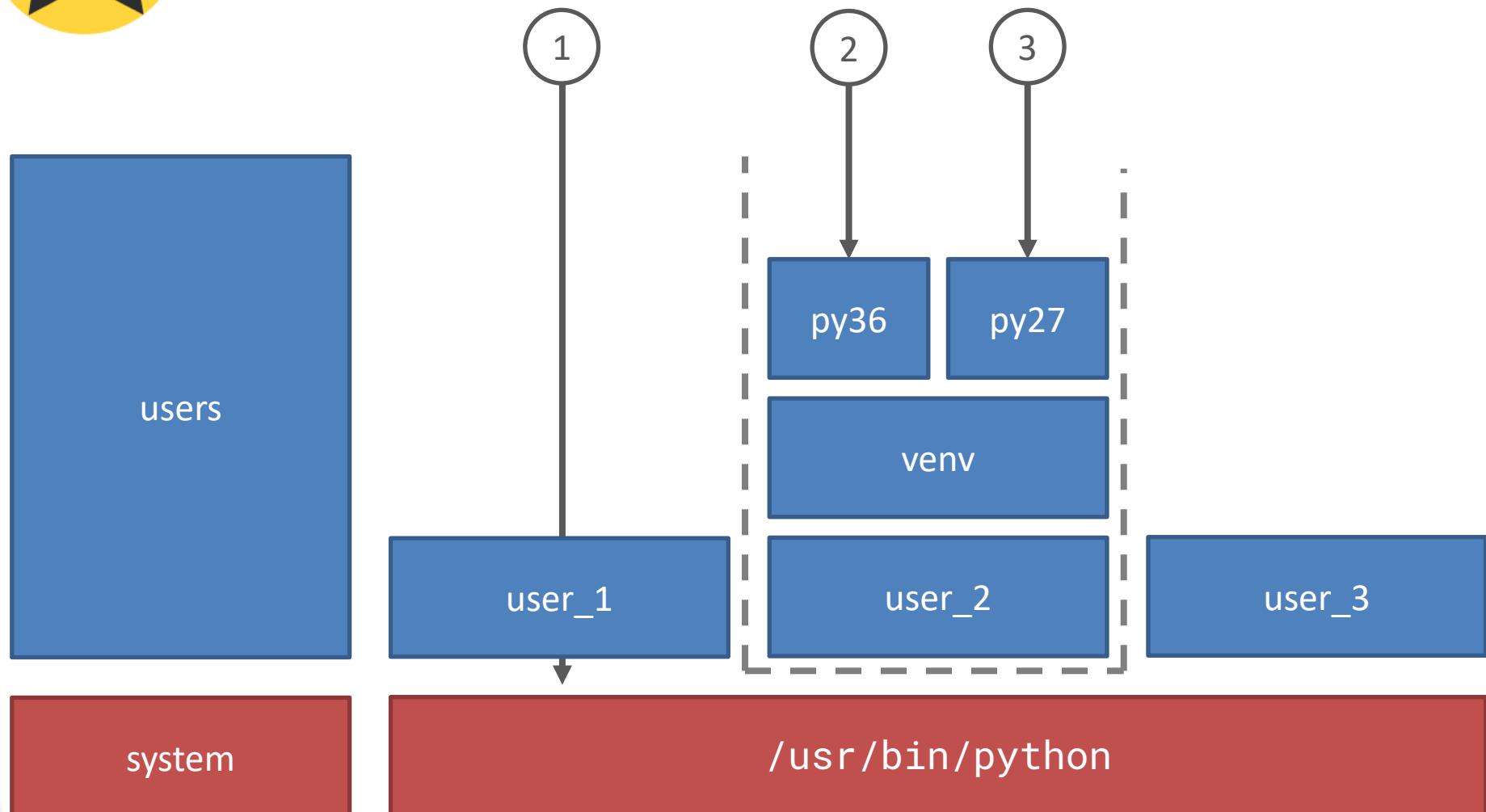


Virtual Environments





Virtual Environments





Conda and Anaconda



ANACONDA®



Conda and Anaconda





Conda and Anaconda





Conda and Anaconda

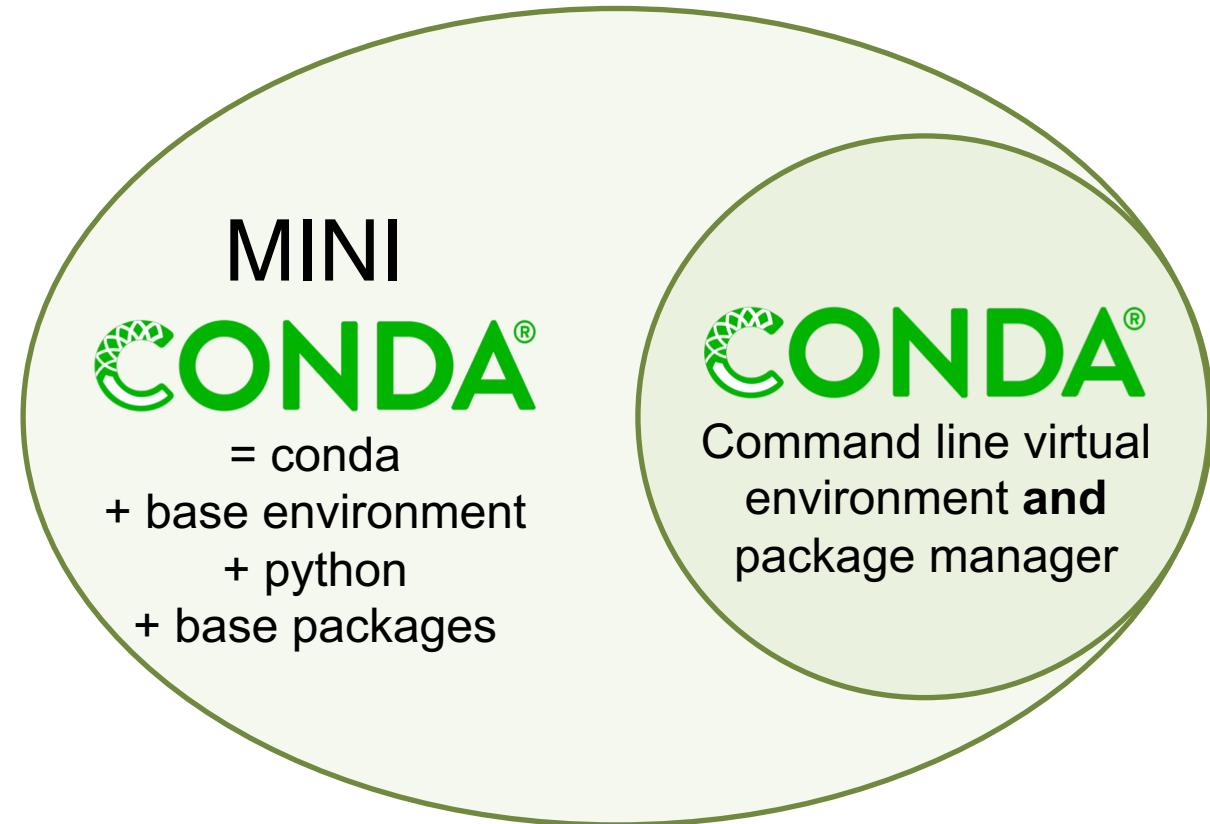


MINI
CONDA®





Conda and Anaconda





Conda and Anaconda



= miniconda
+ more than 100
packages



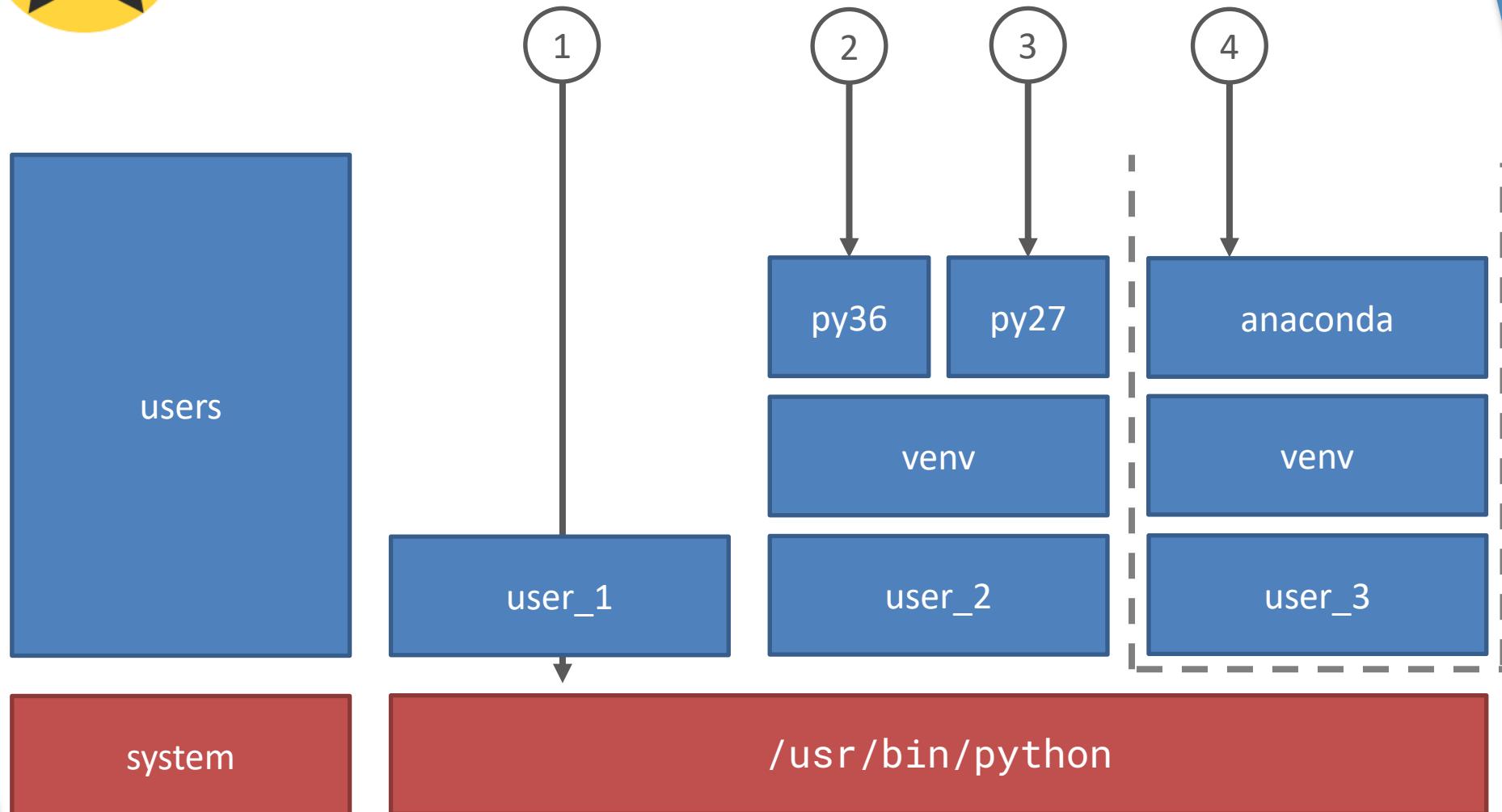
= conda
+ python
+ base packages



Command line virtual
environment **and**
package manager

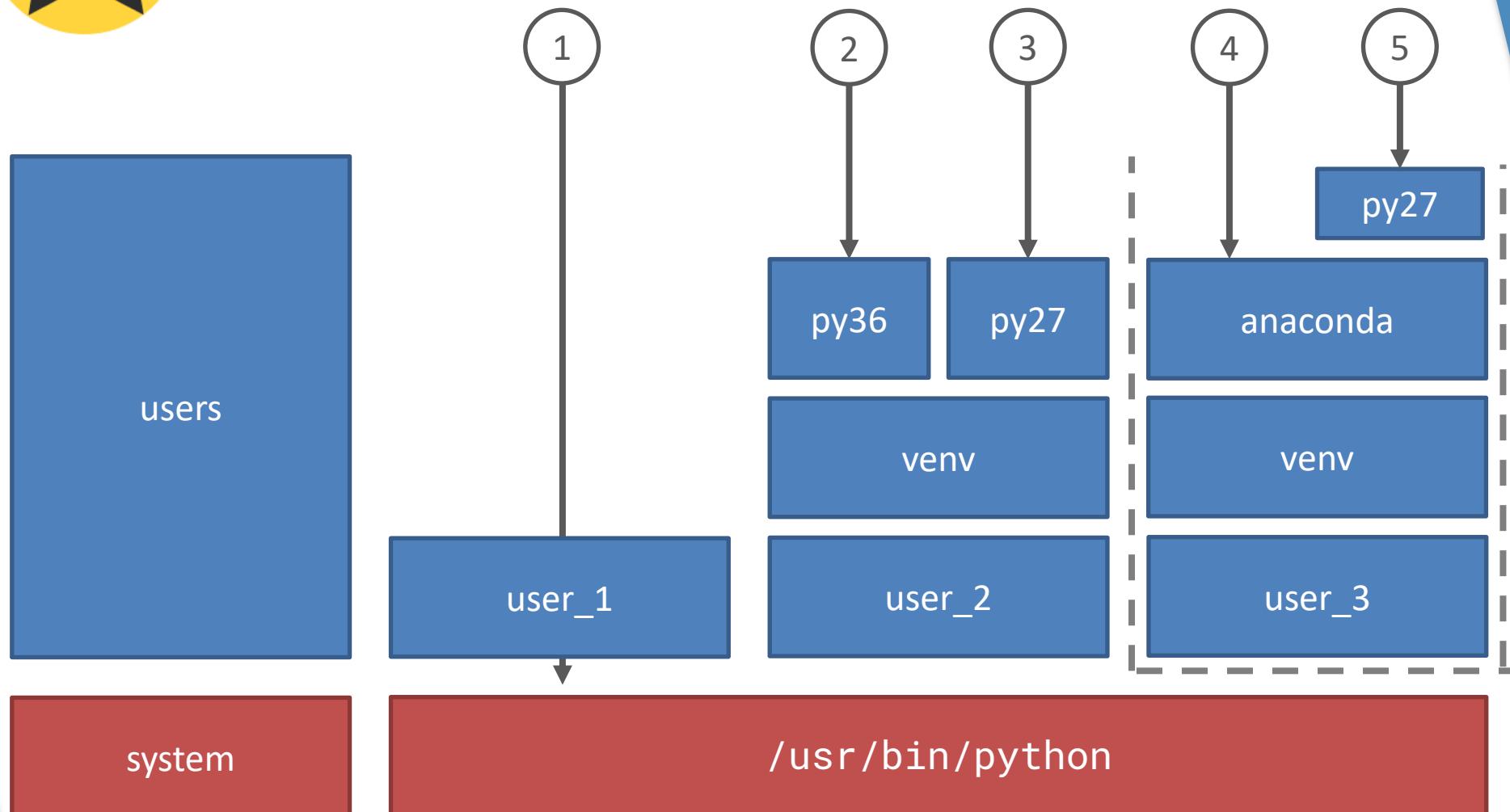


Virtual Environments



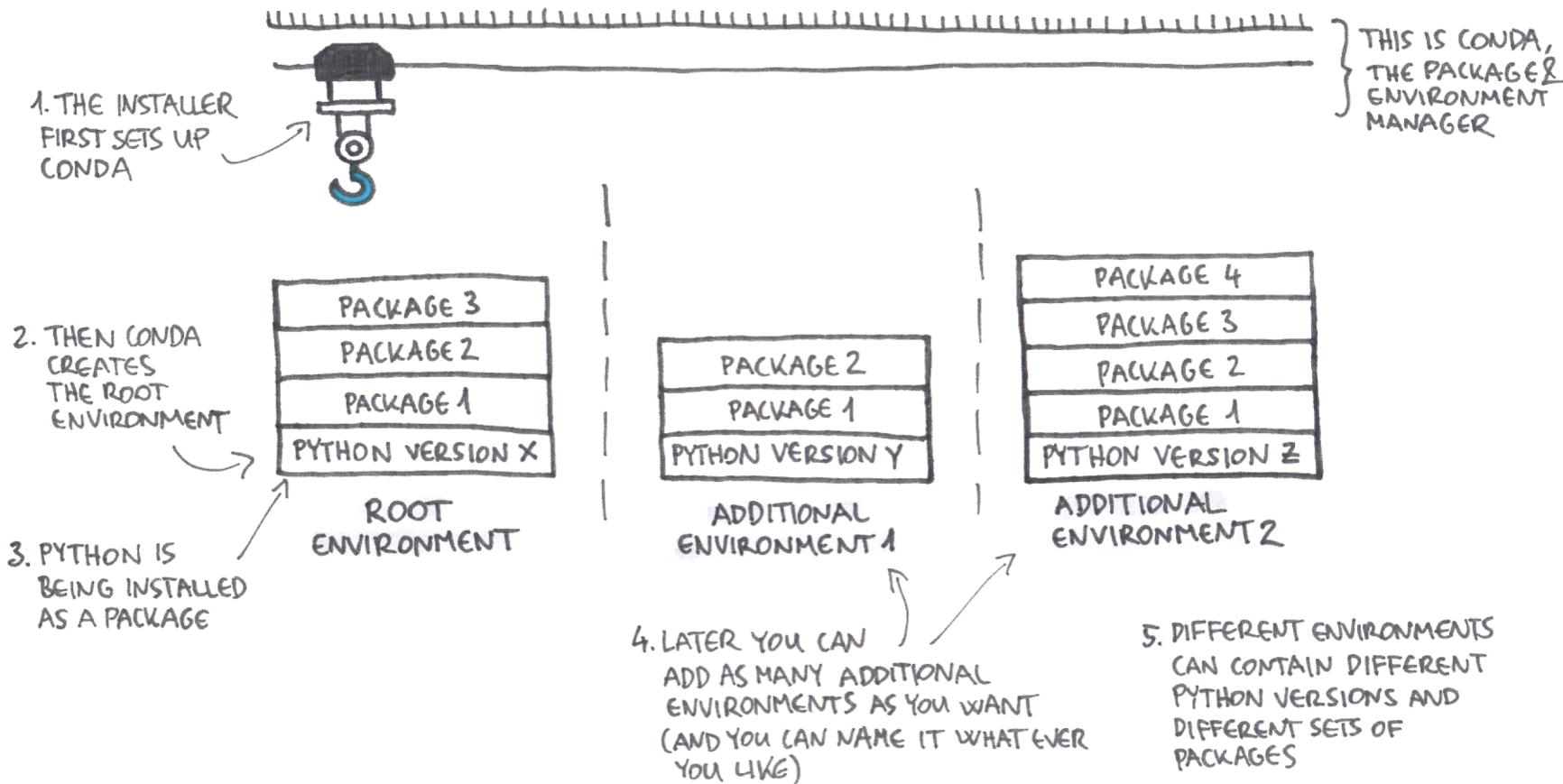


Virtual Environments





Conda and Anaconda





Conda and Anaconda



ANACONDA®
Download and Install



Download and Install



Conda and Anaconda

Let's check **which** Python we are using.

```
$ which python  
/usr/bin/python
```

System

```
$ which python  
/Users/bquint/miniconda3/bin/python
```

Miniconda

```
$ which python  
/Users/bquint/anaconda3/bin/python
```

Anaconda



Conda and Anaconda

HELP!!!

```
$ conda help
usage: conda [-h] [-V] command ...
...
...
...
```



Conda and Anaconda

HELP!!!

```
$ conda help
usage: conda [-h] [-V] command ...
...
...
...
```

info create activate
list remove deactivate



Conda and Anaconda

HELP!!!

```
$ conda help
usage: conda [-h] [-V] command ...
...
...
...
```

info create activate
list remove deactivate

```
$ conda create --help
```



Conda and Anaconda

[Manage Packages](#)

Search

```
$ conda search <package_name>
```

Install

```
$ conda install --channel astroconda sextractor
```

```
$ conda install ds9=7.5
```

List

```
$ conda list
```

```
$ conda list --explicit > list_of_packages.txt
```

Remove

```
$ conda remove seaborn
```

Update

```
$ conda update numpy
```



Conda and Anaconda

[Manage Environments](#)

Create

```
$ conda create --name myenv --python=3.6 pkg1 pkg2 pkg3 ...
```

```
$ conda create -n myenv --file list_of_packages.txt
```

(De)Activate

```
$ source activate myenv
```

```
$ source deactivate
```

Conda >= 4.6

```
$ conda activate myenv
```

```
$ conda deactivate
```

List environments

```
$ conda info --envs
```

Remove

```
$ conda remove --name myenv --all
```



Conda and Anaconda

HELP!!!

```
$ conda help
usage: conda [-h] [-V] command ...
...
...
...
conda commands available from other packages:
  env
```



Conda and Anaconda

HELP!!!

```
$ conda help
usage: conda [-h] [-V] command ...
...
...
...
conda commands available from other packages:
    env
```





Conda and Anaconda

HELP!!!

```
$ conda help  
usage: conda [-h] [-V] command ...  
...  
...  
...  
conda commands available from other packages:  
→ env ←
```

```
$ conda list
```

!=

```
$ conda env list
```

```
$ conda remove -n myenv --all
```

!=

```
$ conda env remove -n myenv
```



Conda and Anaconda

HELP!!!

```
$ conda help  
usage: conda [-h] [-V] command ...  
...  
...  
...  
conda commands available from other packages:  
→ env ←
```

```
$ conda list
```

!=

```
$ conda env list
```

```
$ conda remove -n myenv --all
```

!=

```
$ conda env remove -n myenv
```

conda vs conda env has too much confusion #5253



Conda and Anaconda

[Manage Environments](#)

Anaconda Navigator

Search Environments

Installed Channels Search Packages

Name	Description	Version
anaconda-client	Anaconda.org command line client library	1.7.2
asn1crypto	Python asn.1 library with a focus on performance and a pythonic api	0.24.0
attrs	Attrs is the python package that will bring back the joy of writing classes by relieving you from the drudgery of implementing object protocols (aka dunder methods).	19.1.0
ca-certificates	Certificates for use with other packages.	2019.1.23
certifi	Python package for providing mozilla's ca bundle.	2019.3.9
cffi	Foreign function interface for python calling c code.	1.11.5
chardet	Universal character encoding detector	3.0.4
client	Command line client library for windows and posix	1.2.2
conda	Os-agnostic, system-level binary package and environment manager.	4.6.11
conda-env	Tools for interacting with conda environments.	2.6.0
cryptography	Provides cryptographic recipes and primitives to python developers	2.4.2
dbus	Simple message bus system for applications to talk to one another	1.13.6

66 packages available



Conda and Anaconda

[Manage Environments](#)

Anaconda Navigator

Sign in to Anaconda Cloud

Home

Environments

Learning

Community

Documentation

Developer Blog

[Twitter](#) [YouTube](#) [GitHub](#)

Search Environments

Installed Channels Update index... Search Packages

base (root)

- astroconda
- dragons
- iraf27

Name	Description	Version
anaconda-client	Anaconda.org command line client library	1.7.2
asn1crypto	Python asn.1 library with a focus on performance and a pythonic api	0.24.0
attrs	Attrs is the python package that will bring back the joy of writing classes by relieving you from the drudgery of implementing object protocols (aka dunder methods).	19.1.0
ca-certificates	Certificates for use with other packages.	2019.1.23
certifi	Python package for providing mozilla's ca bundle.	2019.3.9
cffi	Foreign function interface for python calling c code.	1.11.5
chardet	Universal character encoding detector	3.0.4
client	Command line client library for windows and posix	1.2.2
conda	Os-agnostic, system-level binary package and environment manager.	4.6.11
conda-env	Tools for interacting with conda environments.	2.6.0
cryptography	Provides cryptographic recipes and primitives to python developers	2.4.2
dbus	Simple message bus system for applications to talk to one another	1.13.6

Create Clone Import Remove

66 packages available



Preferences

Project: DRAGONS > Project Interpreter For current project

Project Interpreter: Python 3.7 (dragons) ~/miniconda3/envs/dragons/bin/python

Package	Version	Latest version
acstools	3.0.0	
alabaster	0.7.12	0.7.12
appnope	0.1.0	0.1.0
asdf	2.3.2	
asn1crypto	0.24.0	0.24.0
asteval	0.9.13	
astroimtools	0.2	
astropy	3.1.2	3.1.2
astroquery	0.3.9	
atomicwrites	1.3.0	1.3.0
attrs	19.1.0	19.1.0
babel	2.6.0	2.6.0
backcall	0.1.0	0.1.0
beautifulsoup4	4.7.1	4.7.1
blas	1.0	1.0
bleach	3.1.0	3.1.0
bokeh	1.0.4	1.0.4
bottleneck	1.2.1	1.2.1
ca-certificates	2019.1.23	2019.1.23
calcos	3.3.5	
certifi	2019.2.28	2019.2.28

Editor
Font
Color Scheme
Code Style
Inspections
File and Code Templates
File Encodings
Live Templates
File Types
Copyright
Emmet
Images
Intentions
Language Injections
Spelling
TextMate Bundles
TODO
Vim Emulation
Plugins
Version Control
Project: DRAGONS
Project Interpreter
Project Structure
Build, Execution, Deployment
Languages & Frameworks

Cancel Apply OK



Now what?

The Exercise 3 holds a simple example of a Python Project. It includes three Python files:

- `my_module.py` that can be invoked from a terminal and return a random value between 1 and 6,
- `dices.py` that contains classes with dices,
- `setup.py` which contains the configuration for installation of the package.

Exercise

Check the GitHub Page with the exercises:

<https://github.com/b1quint/Python-Tutorial-Series/tree/master/Exercises>



Questions?

