Institute of Data

2021

Data Science and AI

Module 1
Part 2:

Python for Data Science

Agenda: Module 1 Part 2

- Python Fundamentals
- Software Engineering Best Practices
- Using Git & GitHub for Version Control

Institute of Data

Python Fundamentals

- Programming Data Science in Python
- Developing and running Python
- Data structures in Python
- Writing functions in Python
- Iterating in Python
- numpy, pandas, scikit-learn

Programming Data Science in Python

- Programming is the **process of creating a set of instructions** that tell a computer how to perform a task.

- Python is an Interpreted, *High Level general purpose programming language*.

- Python is easy to learn and use and powerful enough to tackle the most difficult problems in any domain.

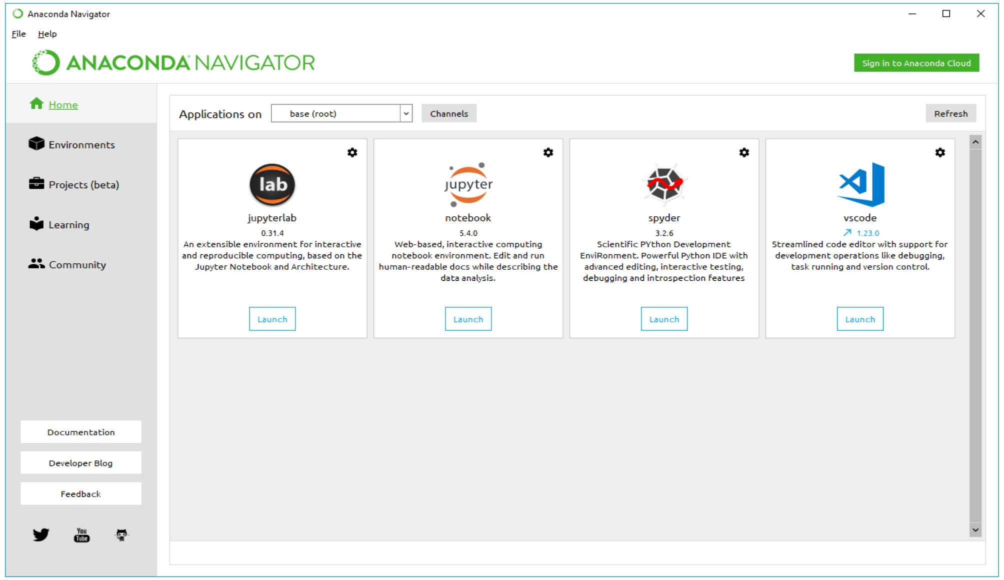
- Python has a very active community with a vast selection of libraries, especially in scientific computing, data analysis and visualisation which makes it **very suitable for Data Science**.

Python versions: 2.7 vs 3.x

- version 2.x
  - large code base
  - last version = 2.7 (no more releases!)
- version 3.x
  - *print* is a function
  - raising & catching exceptions
  - integer division (2.x truncates; 3.x converts to float)
  - short → long integers
  - octal constants: 0*nnn* → 0o*nnn*
  - unicode strings
  - …

Developing and running Python

- Jupyter notebook
- Visual Studio Code (VSC)
    - VSC now has built-in Jupyter notebook support
- Jupyter Lab
- Command prompt
- Anaconda
    - Anaconda distribution is **the recommended way** to configure and manage your Python development and running environment(s).



jupyterlab
0.31.4
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

notebook
5.4.0
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

spyder
3.2.6
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

vscode
↗ 1.23.0
Streamlined code editor with support for development operations like debugging, task running and version control.

Environments

What is an environment?

> a practical way to deal with Python’s packages

Issues:

- Many packages have not been around long enough to be tested with other packages that you might want to use with them
- Packages don’t always get updated quickly in response to updated dependencies

Solution:

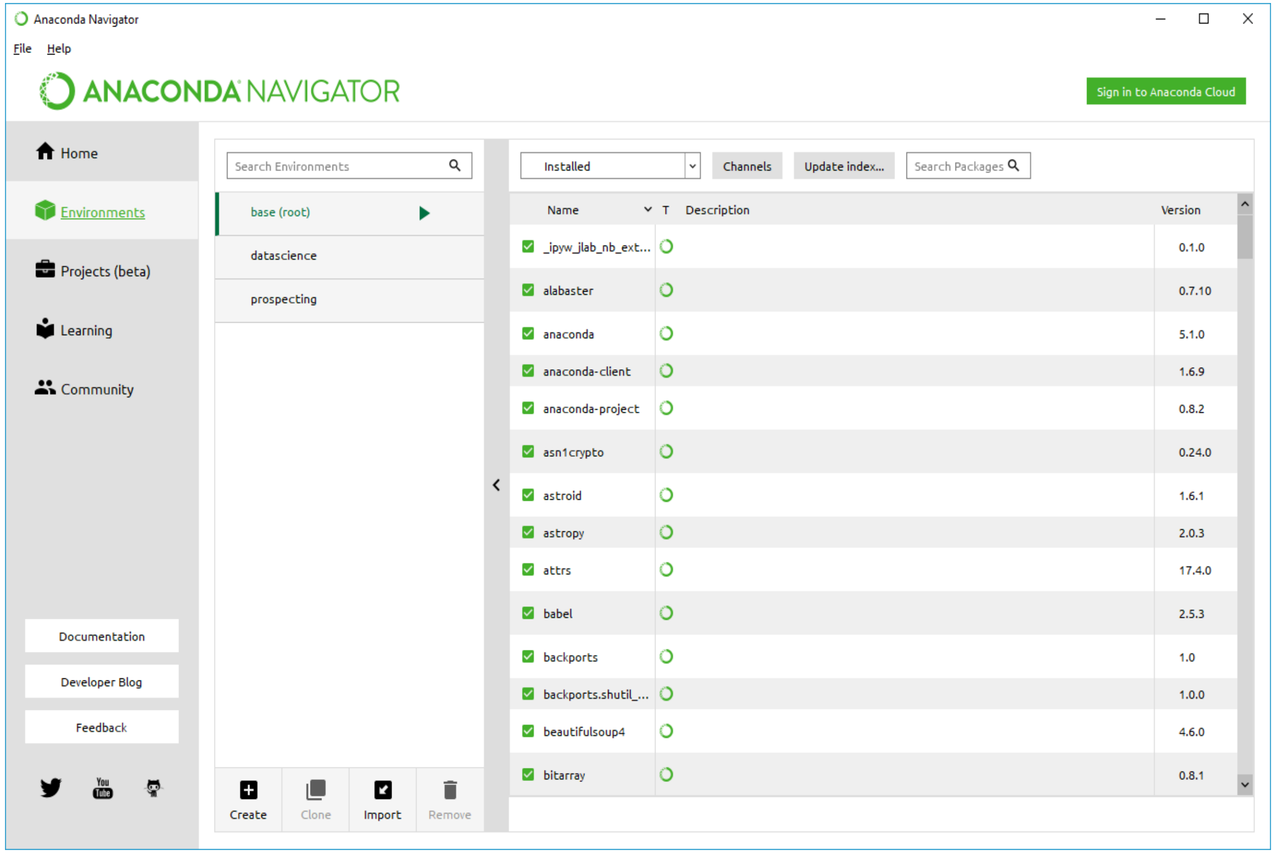
- Create virtual environments for hosting isolated projects using Anaconda Navigator

Environments – cont’d: ***conda***

• create an environment	$ conda create --name myenv1 python
• activate an environment	$ source activate myenv1
• deactivate an environment	$ source deactivate
• install python	$ conda install python=version
• search for available packages	$ conda search searchterm
• install a package	$ conda install anypkg
• list installed packages	$ conda list --name myenv1

Environments – cont’d: ***Anaconda Navigator***

- implements conda via a GUI
  - create envs
  - switch between envs
  - list packages in an env
  - search for packages to add to env
- env-specific app instances
  - set env (e.g. Python27)
  - launch Jupyter notebook to run Python 2.7 code



base (root)

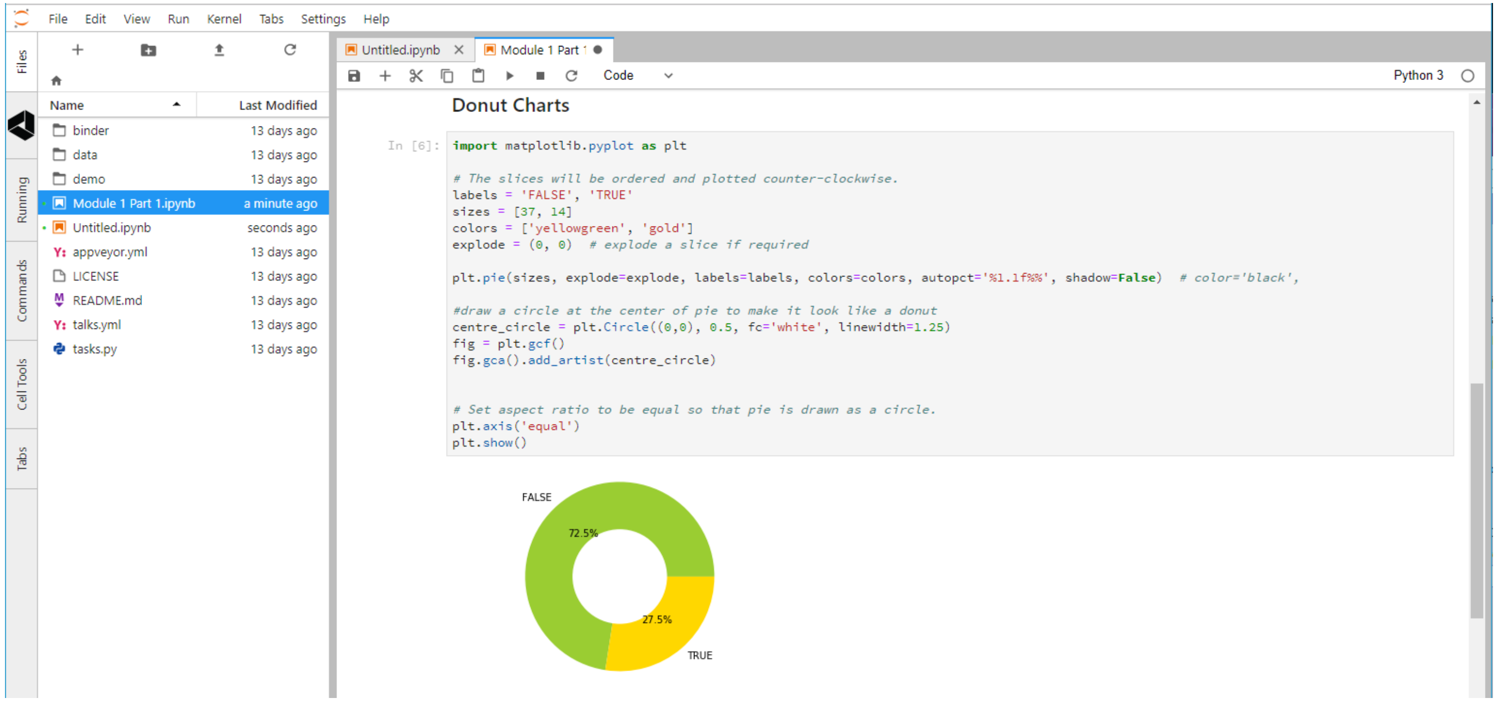
datascience

prospecting

Name	T	Description	Version
☑ _ipyw_jlab_nb_ext...	○		0.1.0
☑ alabaster	○		0.7.10
☑ anaconda	○		5.1.0
☑ anaconda-client	○		1.6.9
☑ anaconda-project	○		0.8.2
☑ asn1crypto	○		0.24.0
☑ astroid	○		1.6.1
☑ astropy	○		2.0.3
☑ attrs	○		17.4.0
☑ babel	○		2.5.3
☑ backports	○		1.0
☑ backports.shutil_...	○		1.0.0
☑ beautifulsoup4	○		4.6.0
☑ bitarray	○		0.8.1

Jupyter Notebooks

- shareable
- environment-based
- interactive or batch execution
- > 40 languages
  - Python, R, Scala, …
- Big Data support
  - Spark



Name	Last Modified
binder	13 days ago
data	13 days ago
demo	13 days ago
Module 1 Part 1.ipynb	a minute ago
Untitled.ipynb	seconds ago
appveyor.yml	13 days ago
LICENSE	13 days ago
README.md	13 days ago
talks.yml	13 days ago
tasks.py	13 days ago

Donut Charts

In [6]:

```
import matplotlib.pyplot as plt

# The slices will be ordered and plotted counter-clockwise.
labels = 'FALSE', 'TRUE'
sizes = [37, 14]
colors = ['yellowgreen', 'gold']
explode = (0, 0)  # explode a slice if required

plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%', shadow=False)  # color='black',

#draw a circle at the center of pie to make it look like a donut
centre_circle = plt.Circle((0,0), 0.5, fc='white', linewidth=1.25)
fig = plt.gcf()
fig.gca().add_artist(centre_circle)


# Set aspect ratio to be equal so that pie is drawn as a circle.
plt.axis('equal')
plt.show()
```

Generic Data Types

Numeric	Text	Other
integer • signed, unsigned	character • unicode	Boolean • true, false Binary • 2^n
floating-point (‘float’) • double = 2 x float	string • character array • 0-based *or* 1-based • null-terminated *or* length-encoded • usually immutable in OOP	unassigned • null • NA undefined • NA • +, − infinity
complex • 2 x double (real, imaginary)	document • key-value pairs (JSON strings)	BLOB • images, video • signals

Classes

```
class phasor:
    def __init__(self, r=0, p=0):
        self.r = r
        self.p = p
    def real(self):
        return (self.r * math.cos(self.p))
    def imag(self):
        return (self.r * math.sin(self.p))

z = phasor(2.7, 0.4 * math.pi)
```

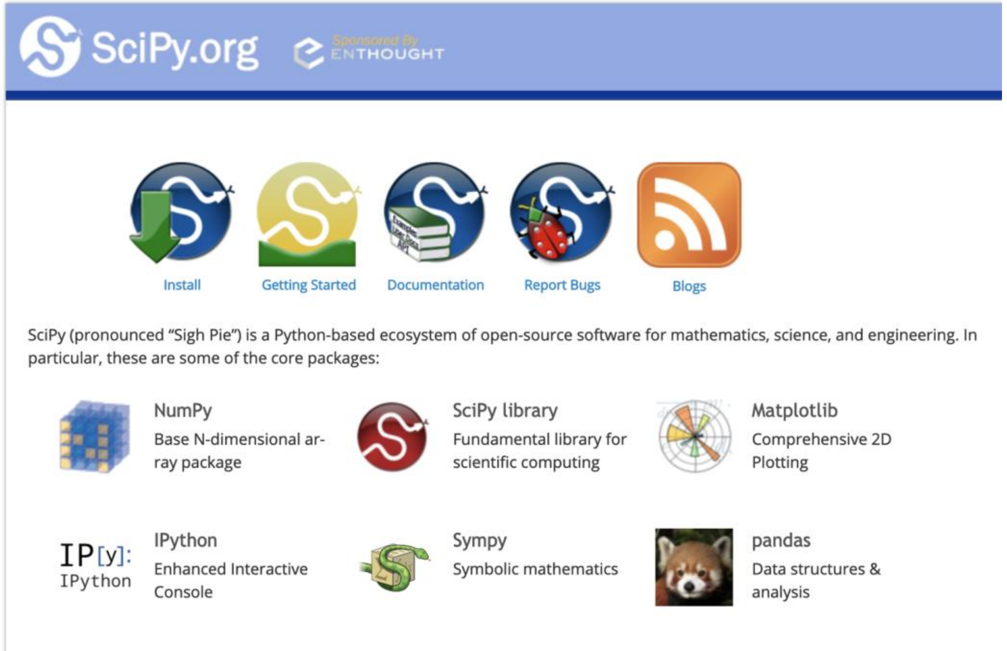
- 2 underscores before/after init

- the **self** parameter is not explicitly mapped to the function call

SciPy

- SciPy (pronounced “Sigh Pie”) is a Python-based ecosystem of open-source software for mathematics, science, and engineering. In particular.

- Main libraries (packages) include numpy, scipy, matplotlib, ipython, jupyter, pandas, sympy, nose



SciPy (pronounced “Sigh Pie”) is a Python-based ecosystem of open-source software for mathematics, science, and engineering. In particular, these are some of the core packages:

NumPy
Base N-dimensional array package

SciPy library
Fundamental library for scientific computing

Matplotlib
Comprehensive 2D Plotting

IPython
Enhanced Interactive Console

Sympy
Symbolic mathematics

pandas
Data structures & analysis

https://www.scipy.org/

NumPy

- the fundamental package for scientific computing with Python
  - a powerful N-dimensional array object
  - sophisticated (broadcasting) functions
  - tools for integrating C/C++ and Fortran code
  - useful linear algebra, Fourier transform, and random number capabilities

```
import numpy as np
```

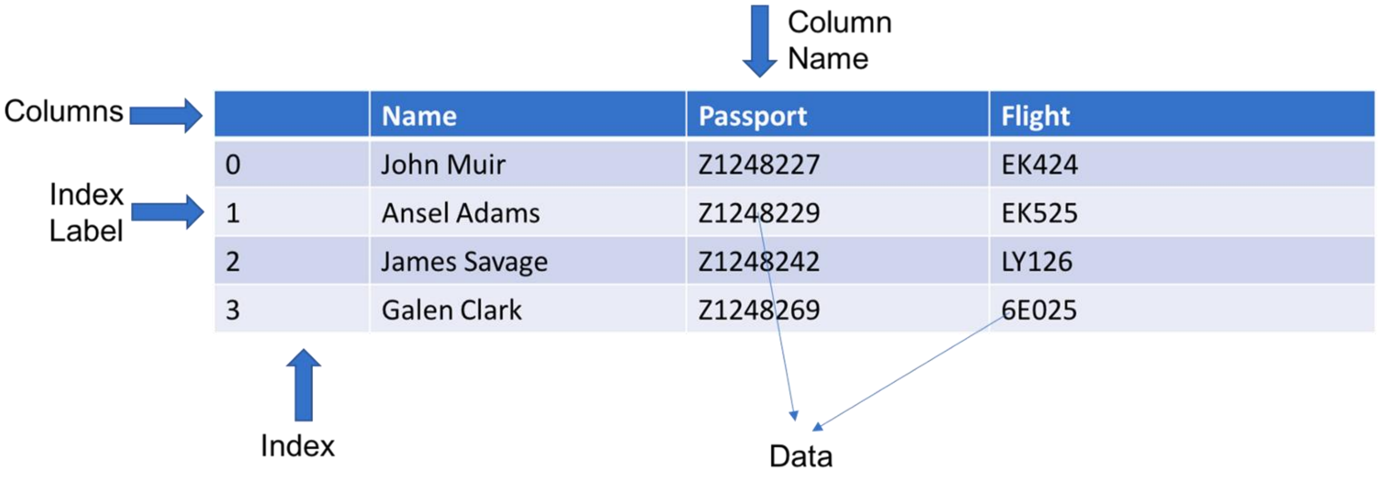
http://www.numpy.org/

Data Types in Python and NumPy

Type	Python	Numpy	Usage
byte byte array	b'any string' bytearray()		- immutable - mutable
integer	int()	- 11 types	- signed, unsigned - 8, 16, 32, 64 bits, unlimited
floating-point	float()	- 3 types	- 16, 32, 64 bits
complex	complex()	- 2 types	- 64, 128 bits
unassigned	None		- object - myVar is not None
missing	nan	isnull(), notnull(), isnan()	- float, object

Pandas

- Rich relational data analysis tool built on top of NumPy
- Easy to use and highly performing APIs
- A foundation for data wrangling, munging, preparation, etc in Python



Column Name

Columns

Index Label

	Name	Passport	Flight
0	John Muir	Z1248227	EK424
1	Ansel Adams	Z1248229	EK525
2	James Savage	Z1248242	LY126
3	Galen Clark	Z1248269	6E025

Index

Data

Pandas Data Frame

Pandas

- high-performance, easy-to-use data structures and data analysis tools
    - DataFrame class
    - IO tools
    - data alignment
    - handling of missing data
    - manipulating data sets
        - reshaping, pivoting
        - slicing, dicing, subsetting
        - merging, joining

```
import pandas as pd
```

https://pandas.pydata.org/

Scikit-learn

- biggest library of ML functions for Python
  - classification
  - regression
  - clustering
  - dimensional reduction
  - model selection & tuning
  - preprocessing

```
$ pip install -U scikit-learn
```

or

```
$ conda install scikit-learn
```

http://scikit-learn.org/stable/

Other Python Packages for Data Science

- statsmodels
  - statistical modelling & testing
  - R-style formulae

```
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

- BeautifulSoup
  - reading & parsing XML & HTML data

```
from bs4 import BeautifulSoup
```

- Natural Language Toolkit
  - tokenising, tagging, analysing text

```
import nltk
```

Visualisation

matplotlib

- histograms
- bars
- curves
- surfaces
- contours
- maps
- legends
- annotations
- primitives

https://matplotlib.org/gallery.html

seaborn

- based on matplotlib
- prettier
- more informative
- more specialised

https://seaborn.pydata.org/examples/index.html

Lab 1.2.1: Numpy

1. Explain the following NumPy methods and create working examples in Jupyter notebook using the data created for you in the beginning of the Lab notebook:

2. Structure your code using functions (prepare to discuss the value of using functions).

- ndim
- shape
- Size
- itemsize
- data
- linspace
- mean
- min
- max
- cumsum
- std
- sum

…

3. Stretch exercise. Use matplotlib to explore the data

Lab 1.2.2: Pandas

1. Explore and download Employee Attrition file from Kaggle (https://www.kaggle.com/HRAnalyticRepository/employee-attrition-data)
2. Explain the following Pandas methods and create working examples in the lab Jupyter notebook.
3. Structure your code using functions (prepare to discuss the value of using functions.

- read_csv
- describe
- loc
- iloc
- Index
- sort_index
- set_index
- sample
- …

4. Stretch exercise. Use matplot to explore some of the data in the data frame

Software Engineering Best Practices

- Object-Oriented Programming
- Refactoring
- Coding for readability
- Coding for testability
- Documenting

Object-Oriented Programming

- an *object* encapsulates
  - data (*attributes*)
  - procedures (*methods*)
- a *class* is a prototype for an object
  - *instantiation*: creating an object (in memory) from a class definition

def: **encapsulation**

- attributes of the class should only be accessible by methods of the class
  - get()
  - set()

Creating and Using a Class in Python

```
class myclass:
    def __init__(self, param1, …):
        # initialise class attributes

    def method1(self, ):
        # do something
        return (method1result)
```

- define class by name
  - initialisation code
    - only **self** is mandatory
    - may use arguments passed from caller
  - define methods
    - only **self** is mandatory
    - may use arguments passed from caller
    - may use attributes
    - may return a value

```
obj1 = myclass(arg1, …)
```

- invoke class name in assignment to instantiate an object
    - omit **self**

Other OOP Concepts

def: **abstraction**

- data and procedures that do not need to be accessible to the caller should be hidden within the class

def: **inheritance**

- new classes can be based on and extend an existing class

def: **polymorphism**

- a class can implement multiple methods with the same name and function, but which operate on different parameters (type and/or number)

Refactoring

def: Restructuring existing code without changing its behaviour

Examples

- abstract reused code to functions
  - generalise functions (polymorphism?)
- use get, set methods
- simplify structure of nested loops, logic
- minimise use of global variables
  - in Python, this includes all variables defined in main program

Coding for Readability (Maintainability)

Examples

- indent blocks
  - mandatory in Python
- white space
  - between groups of lines
  - between symbols
- comments: inline (to explain logic, return values, etc.)
  - sectional (to explain functional blocks)
  - header (to explain program or module)
    - purpose, authors, date
    - dependences, assumptions

- comments are for coders
  - maintaining or extending your code
- documentation is for users
  - explaining what the application is for and how to use it

Coding for Testability

Examples

- avoid side-effects in functions
- enable testing via compiler flags

```
##define TEST_MODE
#if TEST_MODE
print(“test mode activated”)
#endif
```

- write tests *before* functions
  - specify return type(s) supported
  - test return type(s), validity
  - pass sample data as arguments
  - print result

- test *frequently*
  - avoid marathon coding sessions
- code top-down
  - create wireframe code to test logic, structures
  - fill in the details later

pytest

https://docs.pytest.org/en/latest/

Homework

1. Create a GitHub account (if you don’t already have one).
2. Optional: Install GitHub Desktop

   url: https://desktop.github.com

Questions?

Appendices

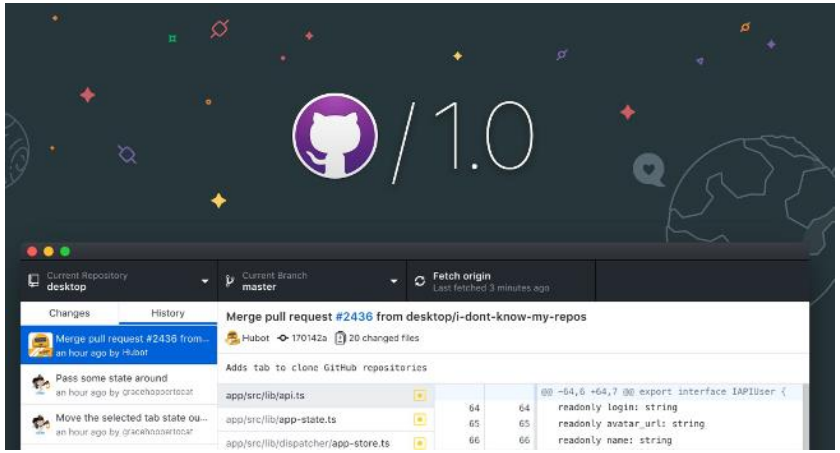
Institute of Data

Version Control with Git & GitHub

- Forking
- Cloning
- Communicating issues
- Managing notifications
- Creating branches
- Making commits
- Introducing changes with Pull Requests

Git & GitHub

- web-based, API
- host code, data, resources
- version control
  - integrates with open-source and commercial IDE tools
- share, collaborate
  - branching
- showcase achievements
- command line & desktop versions



Merge pull request #2436 from...
an hour ago by Hubot

Pass some state around
an hour ago by gracehoppertocat

Move the selected tab state ou...
an hour ago by gracehoppertocat

Merge pull request #2436 from desktop/i-dont-know-my-repos

Hubot 170142a 20 changed files

Adds tab to clone GitHub repositories

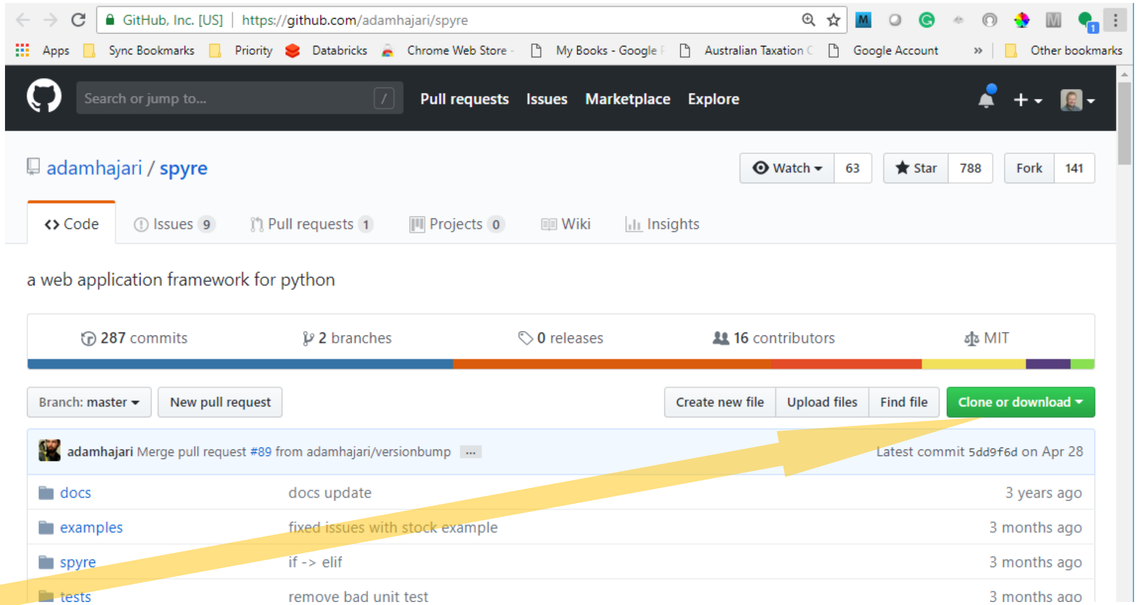
app/src/lib/api.ts
app/src/lib/app-state.ts
app/src/lib/dispatcher/app-store.ts

```
      @@ -64,6 +64,7 @@ export interface IAPIUser {
64  64   readonly login: string
65  65   readonly avatar_url: string
66  66   readonly name: string
```

GitHub: Forking & Cloning a Repo

- *fork:* make your own copy of someone else's repo, on GitHub
  1. click <Fork>
- *clone:* create a (working) copy of the repo on your computer

- GitHub Desktop procedure:
  1. click <Clone or download>
  2. click <Open in Desktop>
  3. navigate to target (local) folder
  4. click <Clone>



adamhajari / spyre

a web application framework for python

287 commits | 2 branches | 0 releases | 16 contributors | MIT

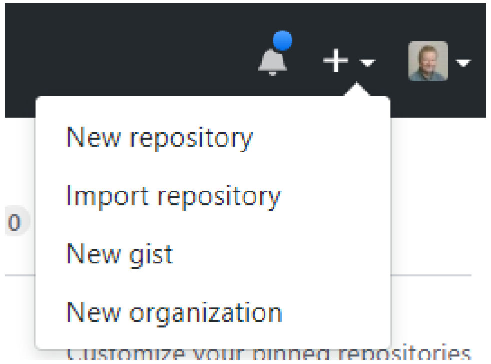
adamhajari Merge pull request #89 from adamhajari/versionbump … Latest commit 5dd9f6d on Apr 28

docs	docs update	3 years ago
examples	fixed issues with stock example	3 months ago
spyre	if -> elif	3 months ago
tests	remove bad unit test	3 months ago

- command-line procedure:
  1. $ cd yourpath
  2. $ git clone https://github.com/yourgithubname/yourgithubrepo

GitHub: Creating a New Repo

- from your GitHub home page
  1. <New repository>
  2. clone the repo to your local drive
  3. copy files, folders into it
  4. commit changes
  5. generate a *pull* request

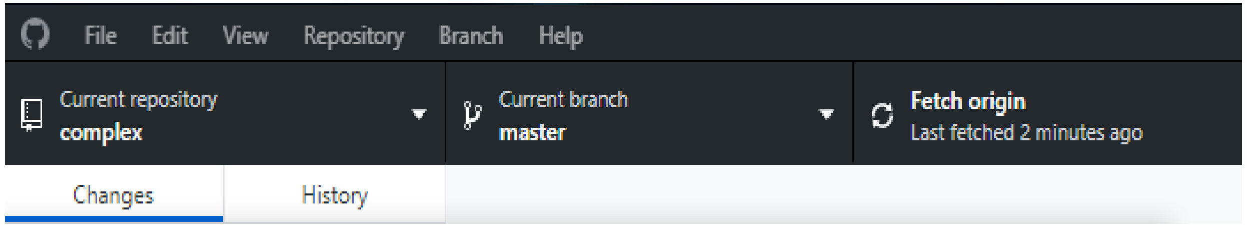


- Creating a branch
  - to allow development in isolation from source repo
    - protects your changes from changes to source
    - rejoin main branch when ready

GitHub: Refreshing Local Repo from Source

Desktop

- <Fetch origin>



Command-line

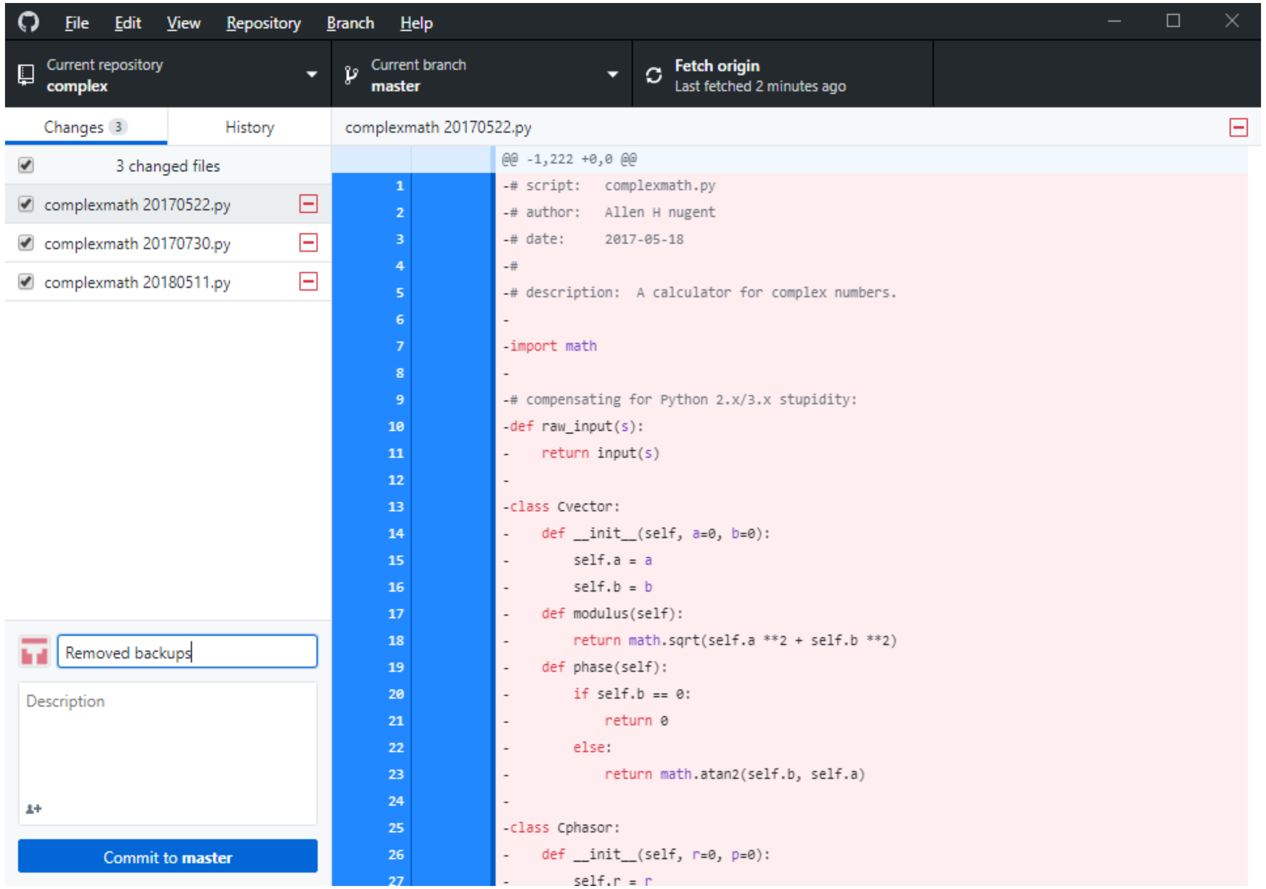
```
$ git checkout master
$ git fetch upstream
$ git merge upstream/master
```

- Ensure you’re in the master branch
- Grab the latest changes from the master
- Merge the master changes with your repo

GitHub: Commit & Pull Request

Desktop

- enter comments in text box
- <Commit to master>
- Repository > Push
  or
  <Push origin>



☑ 3 changed files

☑ complexmath 20170522.py

☑ complexmath 20170730.py

☑ complexmath 20180511.py

Removed backups

Description

complexmath 20170522.py

```
@@ -1,222 +0,0 @@
-# script:   complexmath.py
-# author:   Allen H nugent
-# date:     2017-05-18
-#
-# description:  A calculator for complex numbers.
-
-import math
-
-# compensating for Python 2.x/3.x stupidity:
-def raw_input(s):
-    return input(s)
-
-class Cvector:
-    def __init__(self, a=0, b=0):
-        self.a = a
-        self.b = b
-    def modulus(self):
-        return math.sqrt(self.a **2 + self.b **2)
-    def phase(self):
-        if self.b == 0:
-            return 0
-        else:
-            return math.atan2(self.b, self.a)
-
-class Cphasor:
-    def __init__(self, r=0, p=0):
-        self.r = r
```

GitHub: Commit & Pull Request

Command-line

- commit

```
$ git status
$ git add filename
$ git add .
$ git commit -m your_comments
$ git status
```

- pull request

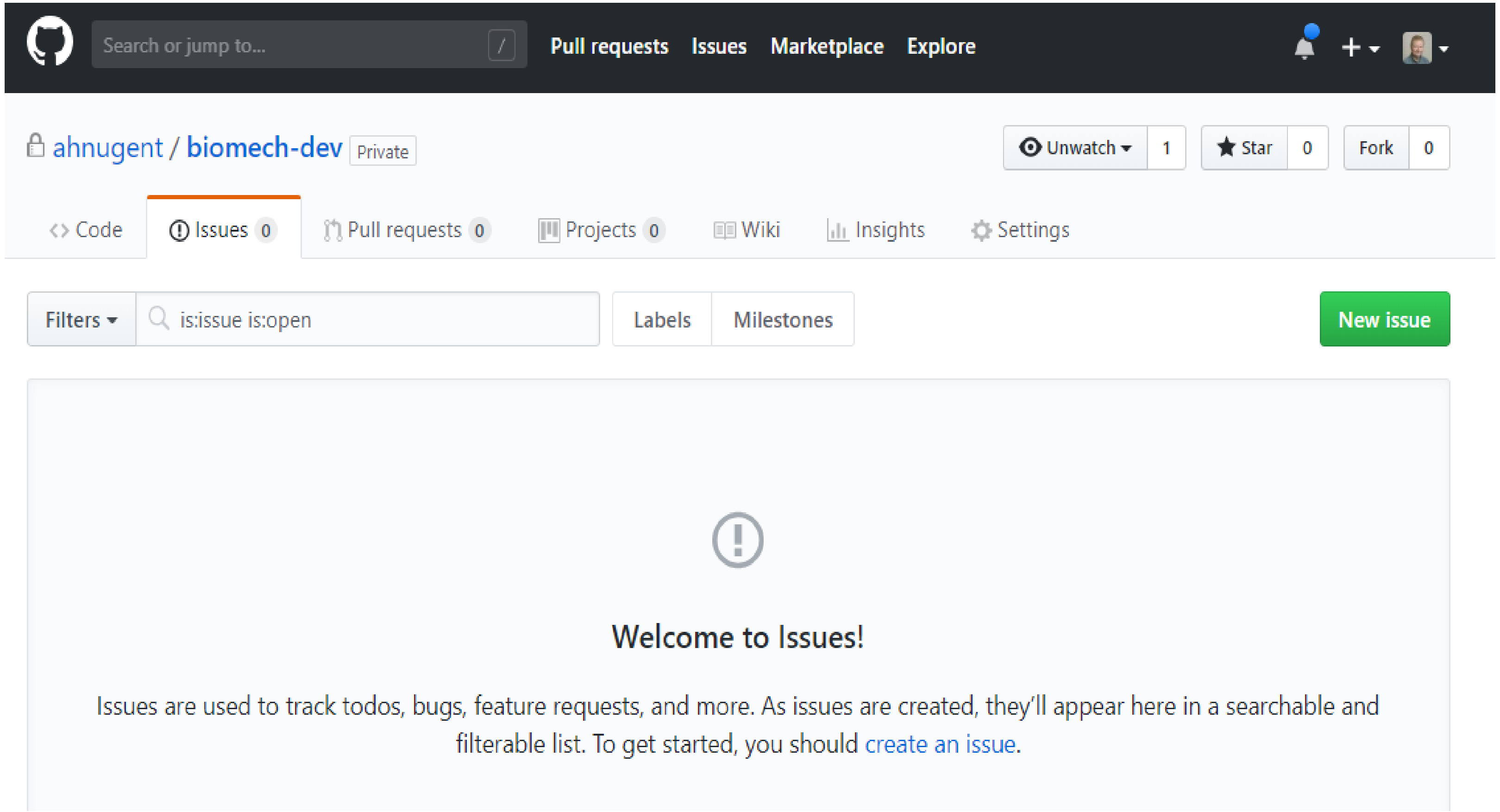
```
$ git push origin master
```

- show changes
- stage one file
- stage all change
- commit file(s), with comments

- origin = your GitHub repo (forked from source repo)
- master = source repo

GitHub: Issues

- track
  - issues / bugs
  - to-do items
  - feature requests
- search
- filter



is:issue is:open

Welcome to Issues!

Issues are used to track todos, bugs, feature requests, and more. As issues are created, they'll appear here in a searchable and filterable list. To get started, you should create an issue.

GitHub: Notifications

Triggers

- you, a team member, or a parent team are mentioned
- you're assigned to an issue or pull request
- a comment is added in a conversation you're subscribed to
- a commit is made to a pull request you're subscribed to
- you open, comment on, or close an issue or pull request
- a review is submitted that approves or requests changes to a pull request you're subscribed to
- you or a team member are requested to review a pull request
- you or a team member are the designated owner of a file affected by a pull request
- you create or reply to a team discussion

End of Presentation!