

Institute of  
Data



2019



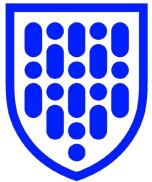
# Data Science and AI

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



# 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**.



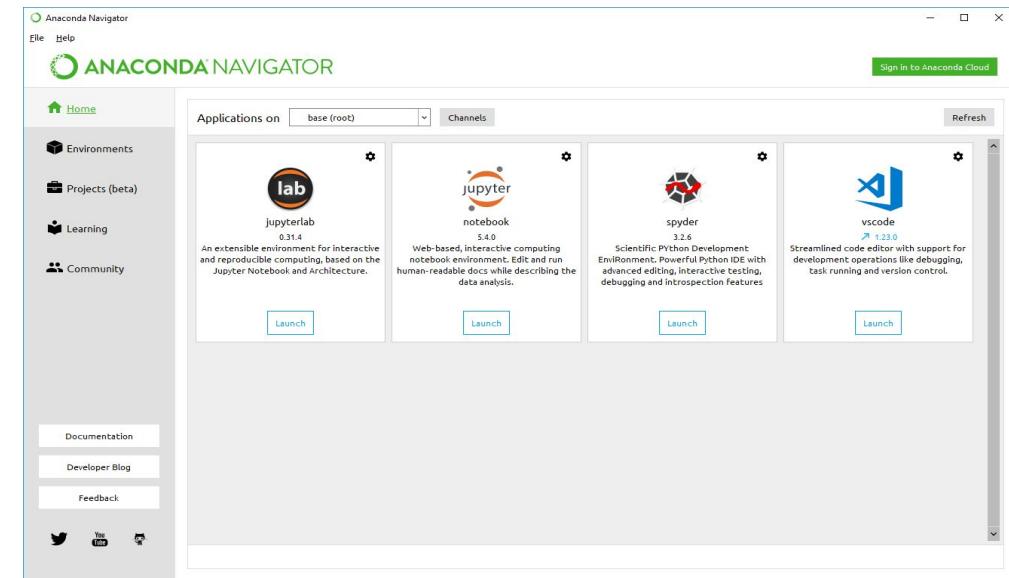
# Python versions: 2.7 vs 3.x

- 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: `0nnn` → `0onnn`
  - unicode strings
  - ...



# Developing and running Python

- Jupyter notebook
- Visual Studio Code (VSC)
  - VSC has now 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).





# Installing Packages with pip

- pip is the package installer for Python. You can use pip to install packages from the Python Package Index and other indexes.
- You can use pip directly in Jupyter notebook or use Anaconda to manage environment configuration (preferred).



## Installing Packages with pip – cont'd

- install a package
- upgrade a package
- install a specific version
- install a set of requirements
- install from an alternate index
- install from a local archive

```
$ pip install anypkg  
$ pip install --upgrade anypkg  
$ pip install anypkg==1.0.4  
$ pip install -r reqsfile.txt  
$ pip install --index-url  
http://my.package.repo/simple/ anypkg  
$ pip install  
./downloads/anypkg-1.0.1.tar.gz
```



# 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*

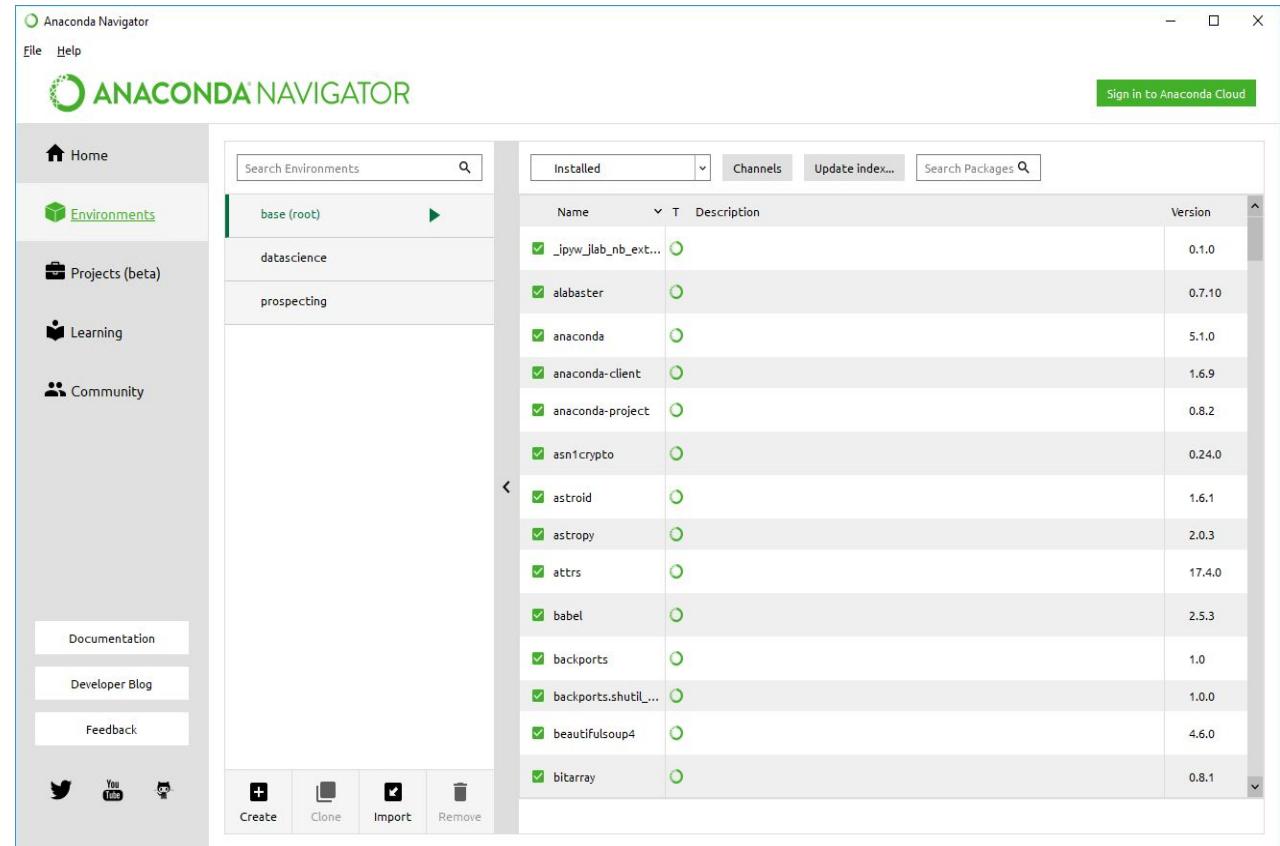
- create an environment
- activate an environment
- deactivate an environment
- install python
- search for available packages
- install a package
- list installed packages

```
$ conda create --name myenv1 python  
$ source activate myenv1  
$ source deactivate  
$ conda install python=version  
$ conda search searchterm  
$ conda install anypkg  
$ 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





# Jupyter Notebooks

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

The screenshot shows the Jupyter Notebook interface. On the left, there's a sidebar with tabs for Files, Running, Commands, Cell Tools, Tabs, and Cell. The Files tab is active, showing a list of files: binder, data, demo, Untitled.ipynb (selected), Module 1 Part 1.ipynb, appveyor.yml, LICENSE, README.md, talks.yml, and tasks.py. The Running tab shows 'Module 1 Part 1.ipynb' as the current notebook. The Commands tab lists various commands. The Cell Tools tab has options like Run, Kernel, and Help. The Tabs tab shows multiple tabs open. The Cell tab has options like New Cell, Insert Cell, and Delete Cell.

In the main area, there are two tabs: 'Untitled.ipynb' and 'Module 1 Part 1.ipynb'. The 'Module 1 Part 1.ipynb' tab is active. Below the tabs, the word 'Code' is followed by a dropdown menu. The status bar at the bottom right shows 'Python 3'.

The code cell contains Python code for creating a donut chart:

```
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='%.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()
```

The output cell shows a donut chart with two segments. The top segment is green and labeled '72.5%'. The bottom segment is yellow and labeled '27.5%'. The labels 'FALSE' and 'TRUE' are positioned above the segments.



# Generic Data Types

Numeric	Text	Other
integer <ul style="list-style-type: none"><li>• signed, unsigned</li></ul>	character <ul style="list-style-type: none"><li>• unicode</li></ul>	Boolean <ul style="list-style-type: none"><li>• true, false</li></ul> Binary <ul style="list-style-type: none"><li>• <math>2^n</math></li></ul>
floating-point ('float') <ul style="list-style-type: none"><li>• double = 2 x float</li></ul>	string <ul style="list-style-type: none"><li>• character array</li><li>• 0-based <i>or</i> 1-based</li><li>• null-terminated <i>or</i> length-encoded</li><li>• usually immutable in OOP</li></ul>	unassigned <ul style="list-style-type: none"><li>• null</li><li>• NA</li></ul> undefined <ul style="list-style-type: none"><li>• NA</li><li>• +, - infinity</li></ul>
complex <ul style="list-style-type: none"><li>• 2 x double (real, imaginary)</li></ul>	document <ul style="list-style-type: none"><li>• key-value pairs (JSON strings)</li></ul>	BLOB <ul style="list-style-type: none"><li>• images, video</li><li>• signals</li></ul>



# Data Structures

- lists
  - ordered, mixed-type, mutable
  - append, extend, insert, remove, pop, clear, index, count, sort, reverse, copy
  - comprehensions
- tuples
  - ordered, mixed-type, immutable
  - support packing, unpacking of variables
- sets
  - unordered, no duplicates
- dictionaries
  - key-value pairs (unordered)



# Functions

```
def funcName(param1, param2, defArg1 = 0, defArg2 = 100):  
    # code here  
    return someResult
```

- optional parameters take default arguments if missing from function call
- arguments are assigned to parameters in defined sequence unless named in call
- return statement
  - optional
  - can return multiple items
- scope is inherited from main (but not from a calling function)



# 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



# Iteration

- *while condition*
- *for iterator in list*
- *continue*
- *break*
- *pass*

```
a = ['Mary', 'had', 'a', 'little', 'lamb']
```

```
for w in a:
```

```
    print(w)
```

```
for i in range(len(a)):
```

```
    print (i, a[i])
```

```
class MyClass(object):
```

```
    def meth_a(self):
```

```
        pass
```

```
    def meth_b(self):
```

```
        print ("I'm meth_b")
```



# 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

The screenshot shows the official SciPy website at <https://www.scipy.org/>. The header features the SciPy logo and the text "Sponsored By ENTHOUGHT". Below the header, there are five circular icons with labels: "Install" (green arrow), "Getting Started" (yellow sun), "Documentation" (blue book), "Report Bugs" (bug), and "Blogs" (RSS feed). A descriptive text block below these icons states: "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:". Below this text, there are six cards, each representing a core package: NumPy (3D array icon), SciPy library (red S icon), Matplotlib (pie chart icon), IP[y]: IPython (ipython logo), Sympy (green S icon), and pandas (panda bear icon).

<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/
```



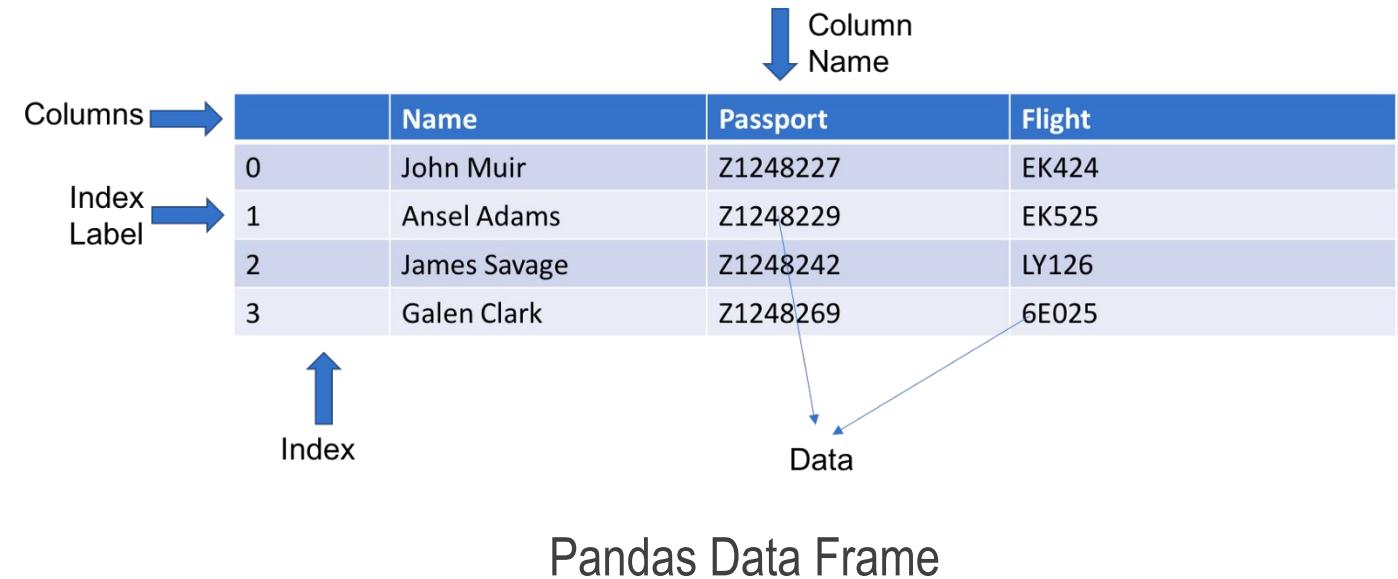
# Data Types in Python and NumPy

Type	Python	Numpy	Usage
byte	b'any string'		<ul style="list-style-type: none"><li>• immutable</li></ul>
byte array	bytearray()		<ul style="list-style-type: none"><li>• mutable</li></ul>
integer	int()	<ul style="list-style-type: none"><li>• 11 types</li></ul>	<ul style="list-style-type: none"><li>• signed, unsigned</li><li>• 8, 16, 32, 64 bits, unlimited</li></ul>
floating-point	float()	<ul style="list-style-type: none"><li>• 3 types</li></ul>	<ul style="list-style-type: none"><li>• 16, 32, 64 bits</li></ul>
complex	complex()	<ul style="list-style-type: none"><li>• 2 types</li></ul>	<ul style="list-style-type: none"><li>• 64, 128 bits</li></ul>
unassigned	None		<ul style="list-style-type: none"><li>• object</li><li>• myVar is not None</li></ul>
missing	nan	isnull(), notnull(), isnan()	<ul style="list-style-type: none"><li>• float, object</li></ul>



# 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





# 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` exercise. Use matplotlib to explore the data

- `max`
- `cumsum`
- `std`



## 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`
2. Stretch exercise. Use matplotlib 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)
```

```
obj1 = myclass(arg1, ...)
```

- 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
- 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>



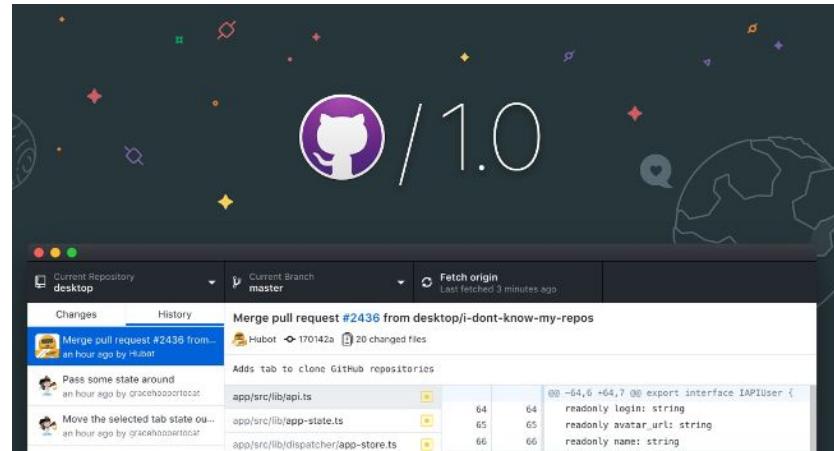
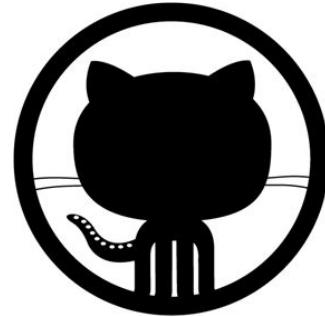
# Version Control with Git & GitHub

- 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





# 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>

adahajari / spyre

a web application framework for python

287 commits 2 branches 0 releases 16 contributors MIT

Branch: master New pull request Create new file Upload files Find file Clone or download

adahajari Merge pull request #89 from adamhajari/versionbump ...  
docs update  
examples fixed issues with stock example  
spyre if -> elif  
tests remove bad unit test

Latest commit 5dd9f6d on Apr 28 3 years ago  
3 months ago 3 months ago 3 months ago

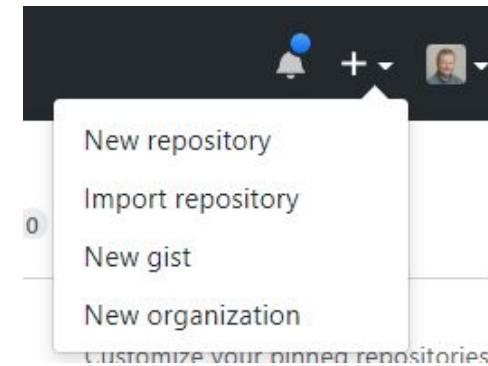
- command-line procedure:

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

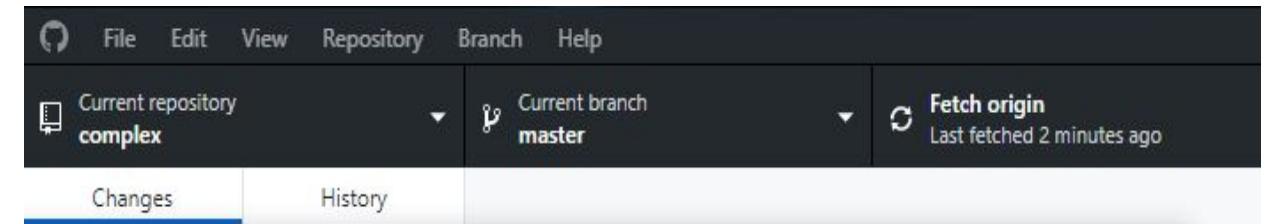




# GitHub: Refreshing Local Repo from Source

## Desktop

- <Fetch origin>



## Command-line

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

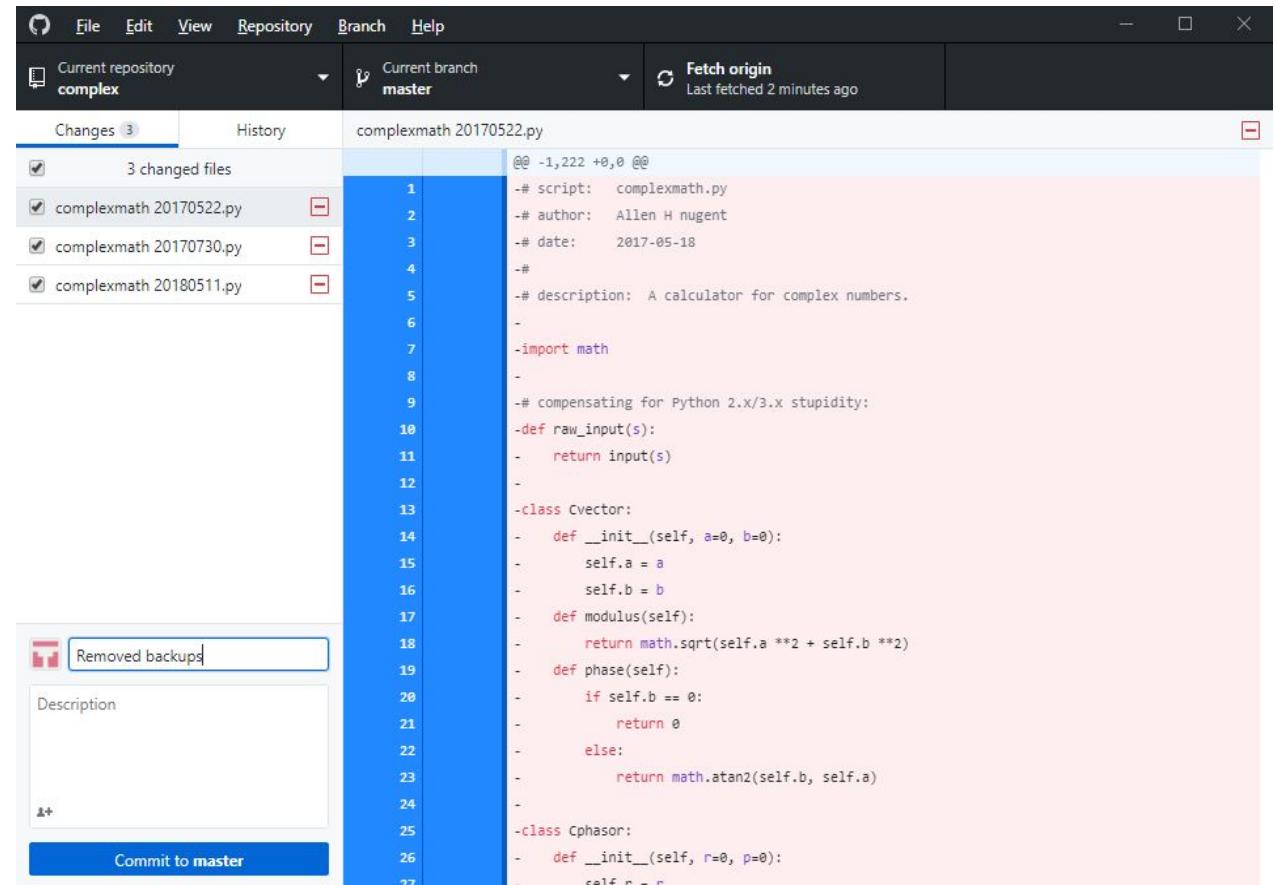
- Ensure you're in the master branch
- Grab the latest changes from the master
- Merge the master changes with your repo



# GitHub: Commit & Pull Request

## Desktop

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



The screenshot shows the GitHub desktop application interface. At the top, there's a menu bar with File, Edit, View, Repository, Branch, and Help. Below the menu, it says "Current repository complex" and "Current branch master". A "Fetch origin" status indicates it was last fetched 2 minutes ago. The main area has tabs for Changes (3) and History. Under Changes, three files are listed: complexmath 20170522.py, complexmath 20170730.py, and complexmath 20180511.py. The History tab shows the commit message "complexmath 20170522.py" which includes a detailed description of the file's purpose and its contents. Below the history, there's a "Removed backups" section with a "Description" input field and a "Commit to master" button.

```
@@ -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: For labs, mini projects & capstone

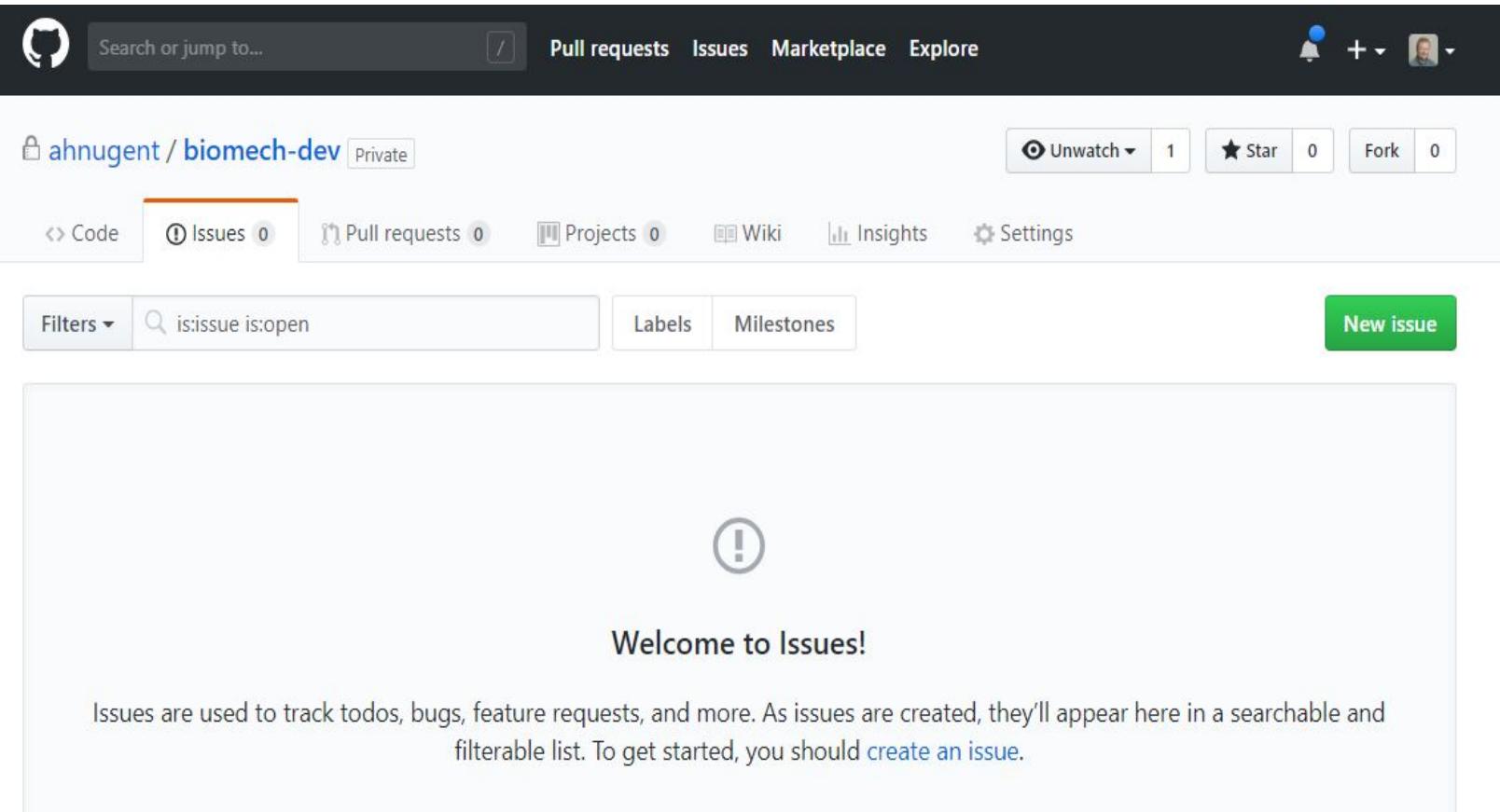
## Workflow

- Please refer to “GIT Workflow.pdf” under “Module X”



# GitHub: Issues

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



A screenshot of a GitHub repository page for 'ahnugent / biomech-dev'. The repository is private. The top navigation bar includes 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. On the right, there are buttons for 'Unwatch', 'Star', 'Fork', and a profile icon. Below the navigation, tabs for 'Code', 'Issues 0', 'Pull requests 0', 'Projects 0', 'Wiki', 'Insights', and 'Settings' are shown, with 'Issues' being the active tab. A search bar contains the query 'is:issue is:open'. To the right of the search bar are 'Labels' and 'Milestones' buttons, and a green 'New issue' button. The main content area displays a large exclamation mark icon and the text 'Welcome to Issues!'. It explains that issues are used to track todos, bugs, feature requests, and more, and encourages users to 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



# Lab 1.2.3: Setting Up GitHub

## Purpose:

- To establish a GitHub repo and develop basic skills for collaborating and maintaining projects.

## Tools & Resources:

- GitHub / GitHub Desktop

## Materials:

- ‘Lab 1.2.3.docx’



# Questions?

# End of Presentation!