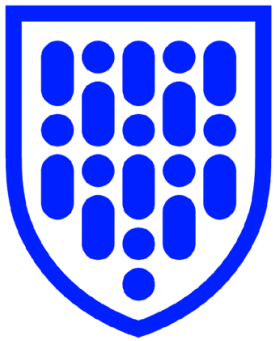


Institute of Data

2024

Data Science and AI

Module 1
Part 1:

Python for Data Science

Institute of Data

Agenda: Module 1 Part 1

- Python Fundamentals
- Python Environments and Libraries
  - NumPy
  - Pandas
- Software Engineering Best Practices
- Appendix – Version Control with Git & GitHub

Institute of Data

Python Fundamentals

- What is programming?
- Importance of programming for data science
- Input and output function options
- Python variables and data types
- If/else, for loops, while loops
- Data structures in Python
- Writing functions in Python

What is programming?

- Programming is:
  - the process of creating a set of instructions that tell a computer how to perform a task.
  - thinking systematically and critically
  - breaking a task into steps. Examples include: a recipe, directions to a destination and mathematical problem solving
- A program usually takes an input and produce an output
- You can think of programming as a way to solve a problem to generate the required output from a given input
- Difference between programming and coding?
  - Programming is the skill to specify a program independent of any programming language
  - Coding is writing the program in a specific programming language

Programming is a fundamental skill for data science

- Data science involves problem solving at many levels and in each step of a project in an implicit (conceptual) or explicit form (programs)

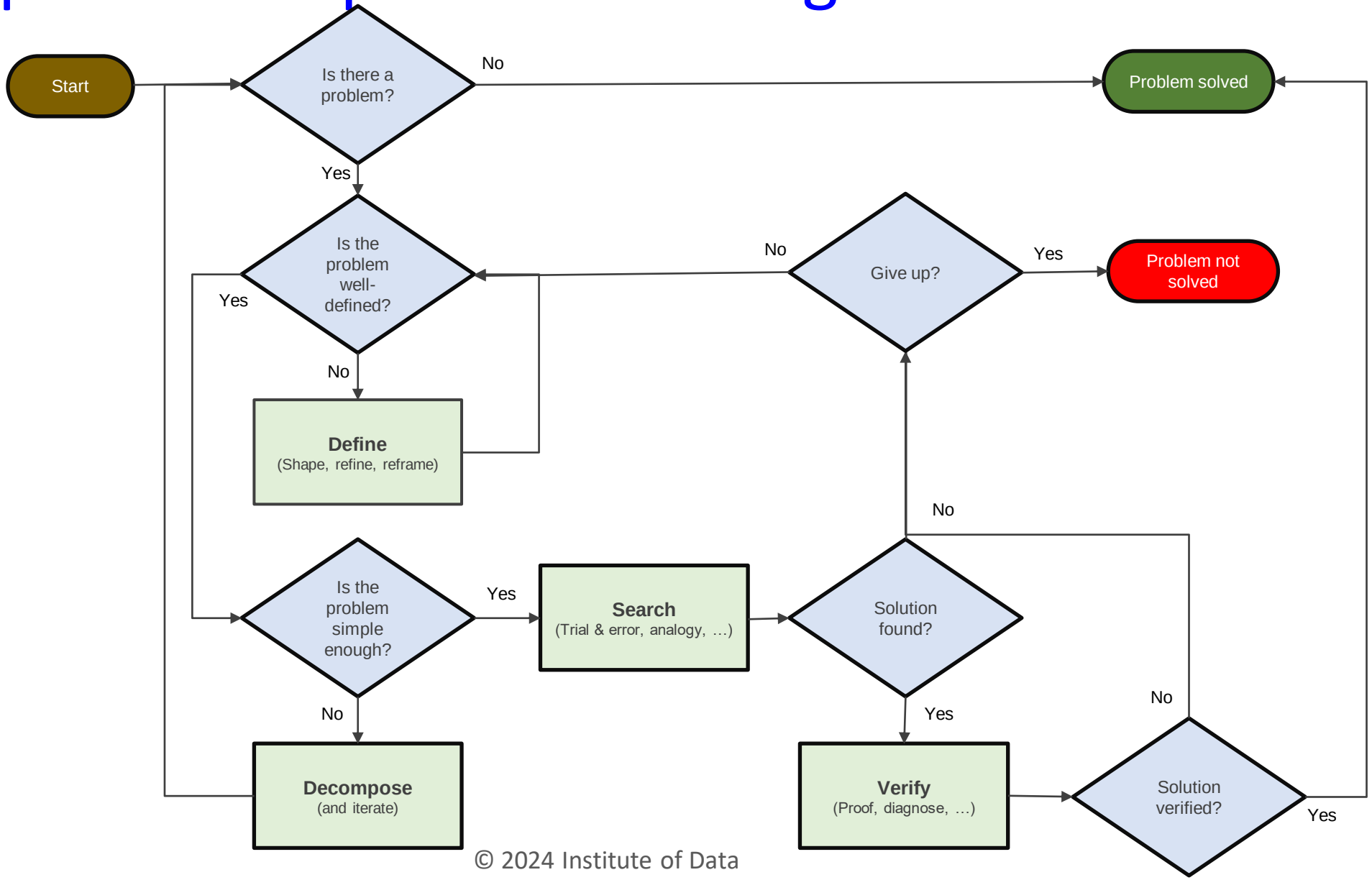
- Programming, which is the main tool for data science, can be defined in its essential form as a problem-solving technique for data-driven problems

- Python is the most popular programming language for data science

Problem solving

- Problem solving is the process of finding solutions to difficult or complex issue
- Scientific method involves stating problems in a manner that facilitate solving them *mathematically* and verify them *empirically*
- Problems can be solved using techniques that include a combination of the following actions:
  - Define
  - Decompose
  - Search
  - Validate

An approach for problem solving



Python programming language

- Python is a high-level programming language, and its core design philosophy is all about code readability and a syntax which allows programmers to express concepts in a few lines of code
- Python is used for developing many different types of computer programs including:
  - Data analysis
  - Data visualisation
  - Machine learning

Python comments

- Single line comment

```
# This is a comment
```

- Multiple line comments

```
"""
Multiple line comment
"""
```

Python input and output functions

- Input function receives an input from the user

```
Data = input('Please enter your name:')
```

- Print function prints formatted text and variables

```
A = 100
print(f"This is a text and embedded variable {A}")
```

Python variables and data types

- Variables are used to store information to be referenced and manipulated in a computer program.
- Common data types
  - Integer <int> examples: 1, 1095, -2
  - Float <float> examples: 1.2, - 2974.074
  - String <str> examples: ‘Bob, “This is a longer string \t with special char’s”
  - Boolean <bool> examples: True, False
- Python allows you to convert variables between these types when needed
- Type command
  - type(12.65) -> <class 'float'>

Python operations

- Math operations
  - + plus - minus / divide * multiply

    < less-than > greater-than <= less-than-equal

    >= greater-than-equal
- Logic operations
  - and, or, not

If/else, for loops, while loops

- The **if/else** statement executes a block of code if a **specified condition** is true. If condition is not met, another block of code can be executed.
- Loops through a block of code a number of times
- Loops through a block of code while a specified condition is met
- continue
- break
- pass

```
var = 10
if (var >= 5):
   print('var is greater than or equal 5')
elif (var < 0):
   print('var is negative')
elif (var == 0):
   print('var is zero')
else:
   print('var is less than 5')
```

```
for i in range(10):
   print(i)
```

```
var = 10
while(var < 20):
   print('var is less than 20')
   var+=2
```

Data structures

- lists
  - A list is the Python equivalent of an array, but is resizable and can contain elements of different types
  - Functions: append, extend, insert, remove, pop, clear, index, count, sort, reverse, copy
  - comprehensions
- tuples
  - A tuple is an (immutable) ordered list of values.
- sets
  - A set is an unordered collection with no duplicate elements.
- dictionaries
  - A dictionary stores (key, value) pairs

```
Tuple_x = (2, 7)

List_y = [2, 4, 6, 8]

Dictionary_z = {"id": 123,
"name": "Item 123"}
```

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)

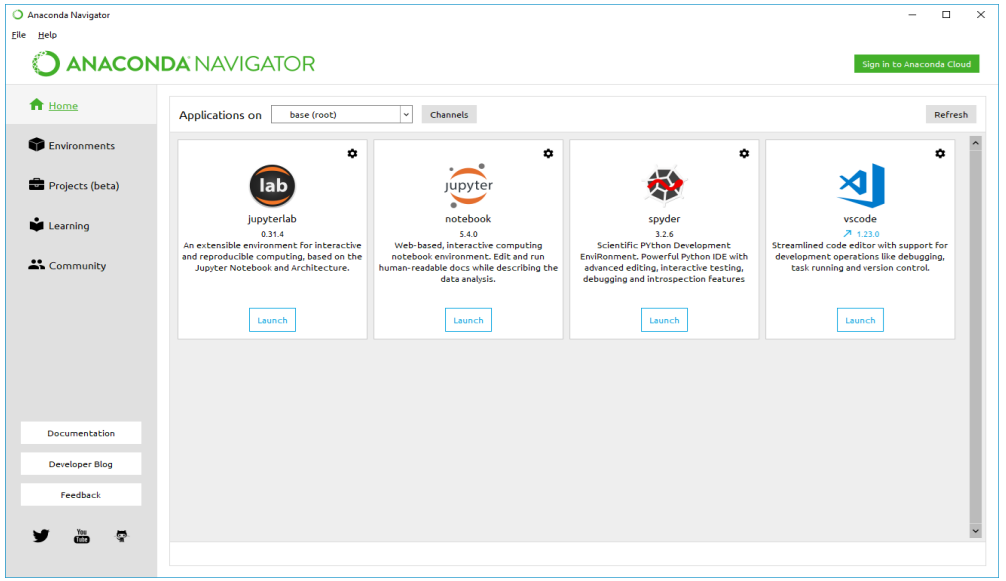
Institute of Data

Python Environments and Libraries

- Developing and running Python
- Python environments
- Python libraries
  - NumPy
  - Visualisation libraries
  - Pandas

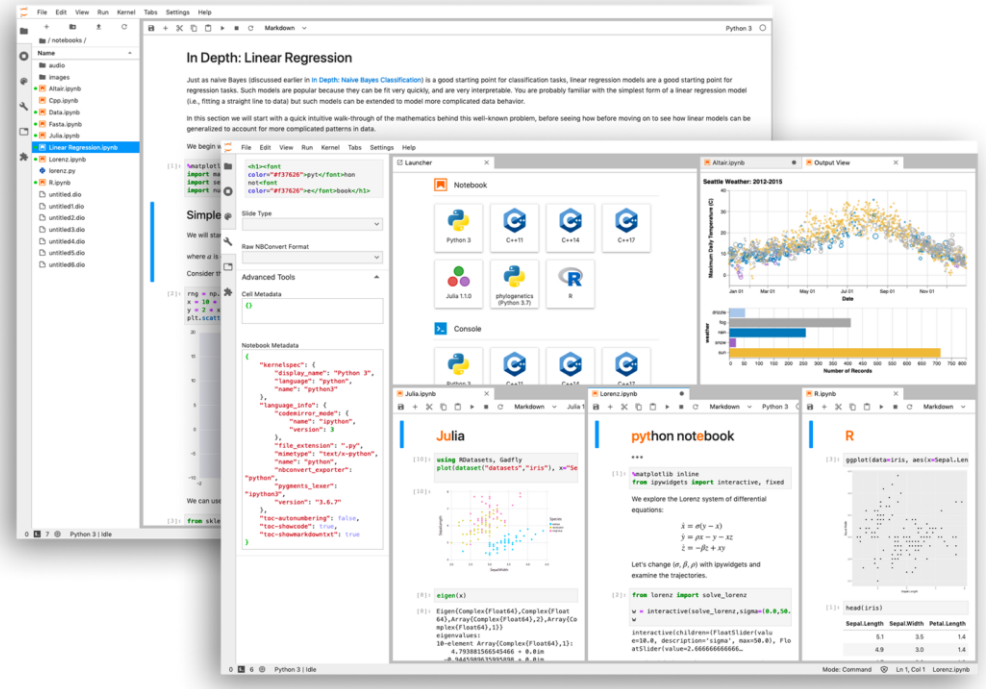
Developing and running Python

- 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).



Jupyter notebook

- The **Jupyter** notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualisations and narrative text
- Interactive or batch execution
- Support for >40 languages including Big Data
  - Python, R, Scala, Spark, …
- We will use Jupyter notebooks for exercises in this course



Environments

What is an environment?

A practical way to deal with Python's packages (libraries)

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

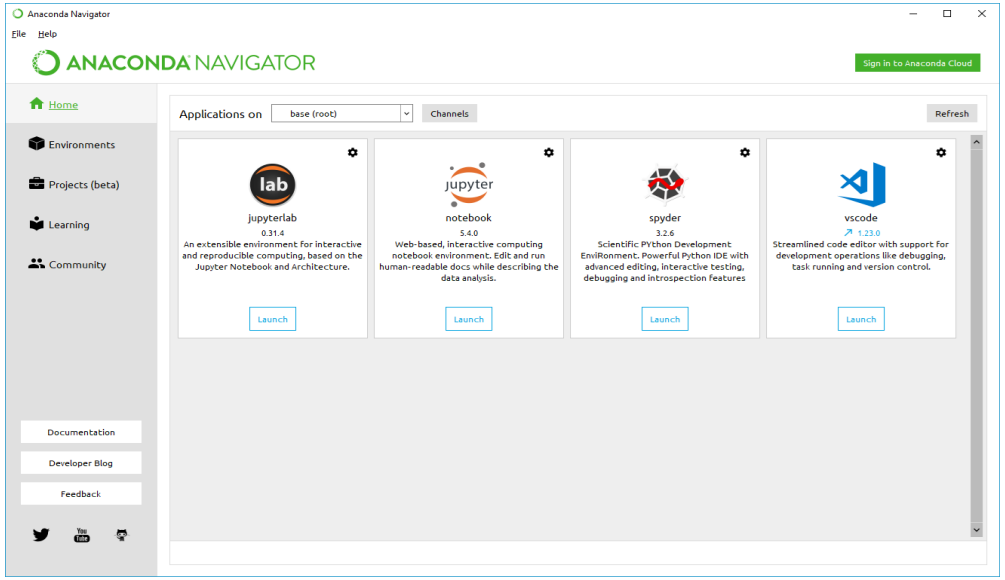
Installing Packages with pip

- 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
```

Anaconda

Anaconda Distribution is the recommended way to configure and manage your Python development and running environment(s).



SciPy

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

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

https://www.scipy.org/

NumPy

- NumPy is the fundamental package for scientific computing with Python
- A powerful N-dimensional array object
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities and many, many more

Generic Data Types (native Python)

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

NumPy Data Types

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

Visualisation libraries

matplotlib

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

seaborn

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

Pandas

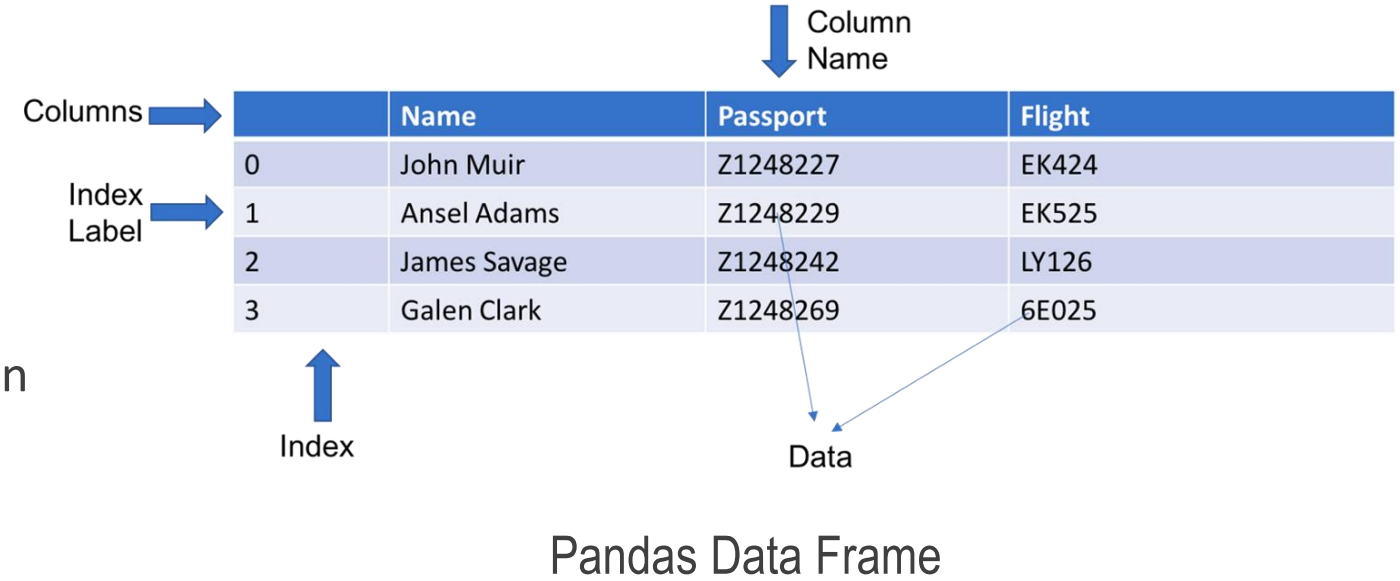
- high-performance, easy-to-use data structures and data analysis tools
  - DataFrame class
  - IO tools
  - data alignment
  - handling of missing data
  - manipulating data sets
    - reshaping, pivoting
    - slicing, dicing, subsetting
    - merging, joining

```
import pandas as pd
```

https://pandas.pydata.org/

Pandas library

- 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



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

Pandas Data Frame

Loading and exploring data

- Pandas can load data from many sources including csv files, websites and databases
- Pandas load data into a data structure called a Data Frame which looks like a spreadsheet

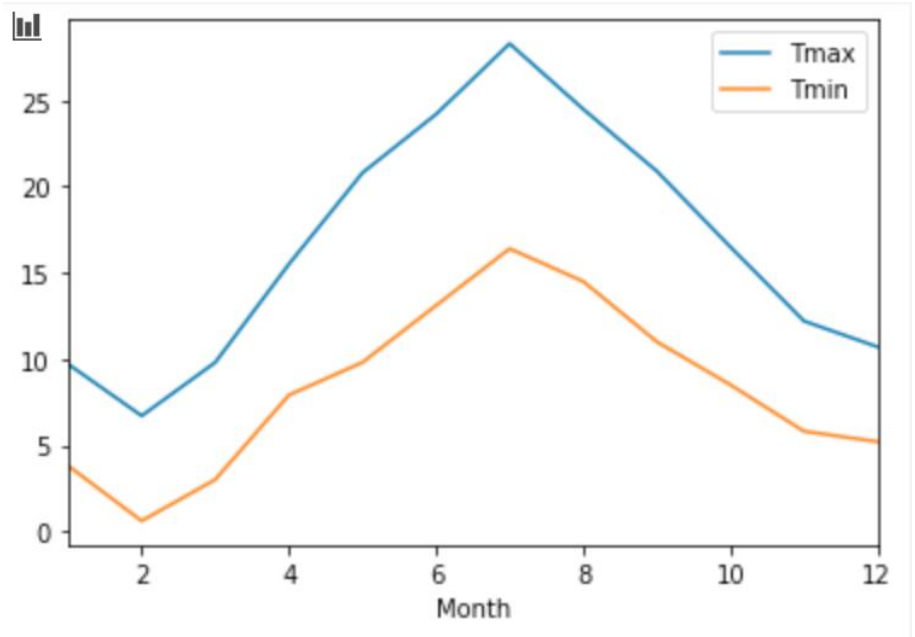
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
weather =
pd.read_csv('https://raw.githubusercontent.com
/alanjones2/dataviz/master/london2018.csv')
print(weather.head())
```

```
   Year  Month  Tmax  Tmin  Rain   Sun
0  2018      1   9.7   3.8  58.0  46.5
1  2018      2   6.7   0.6  29.0  92.0
2  2018      3   9.8   3.0  81.2  70.3
3  2018      4  15.5   7.9  65.2 113.4
4  2018      5  20.8   9.8  58.4 248.3
```

Data visualisation

- Data can be plotted directly from pandas' data frames using matplotlib
- There are many plot types available including:
  - Line chart
  - Bar chart
  - Scatter plot
  - Pie chart
  - Histograms
  - etc

```
weather.plot(y=['Tmax','Tmin'], x='Month')
```



Data statistics

- Pandas provides many functions that allow you to explore statistics of the data including:
  - Count
  - Mean
  - Standard deviation
  - Minimum
  - Maximum

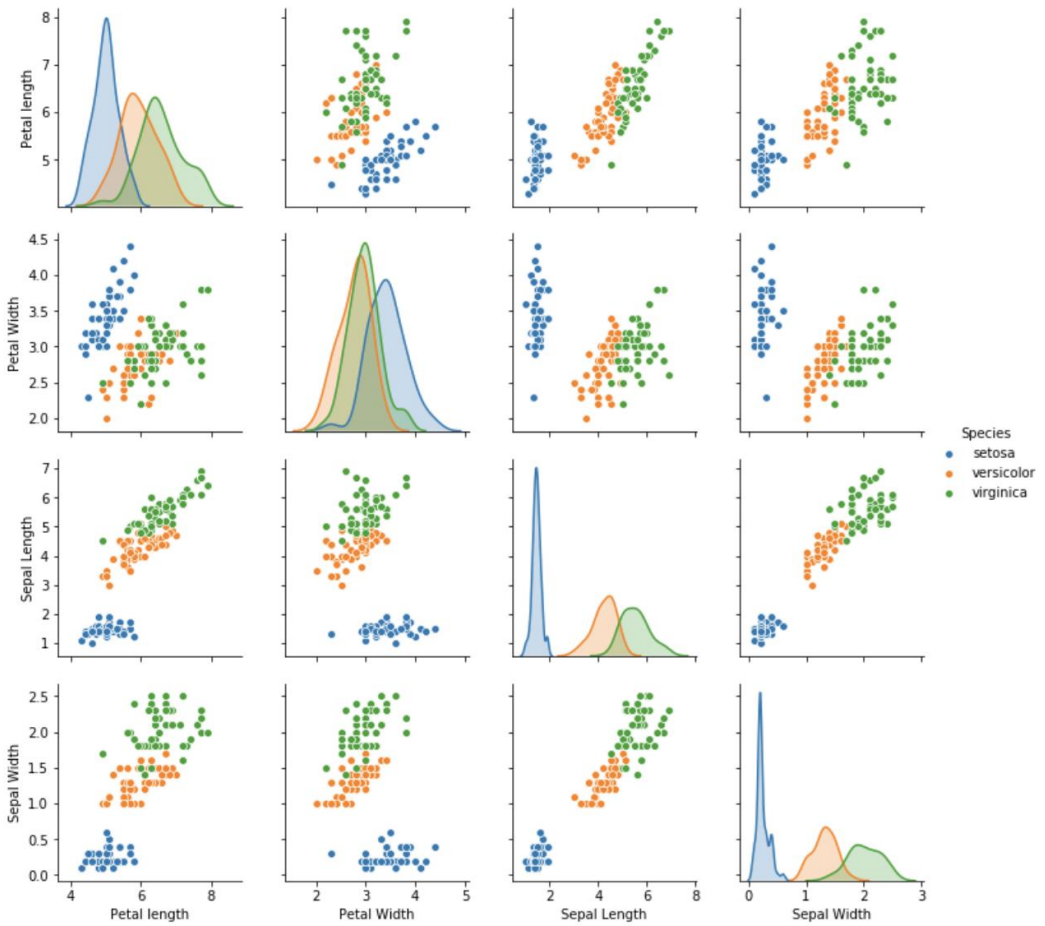
```
from sklearn.datasets import load_iris
dataset=load_iris()
data=pd.DataFrame(dataset["data"],columns=["Petal length","Petal Width","Sepal Length","Sepal Width"])
data["Species"]=dataset["target"]
data["Species"]=data["Species"].apply(lambda x: dataset["target_names"][x])
print(data.head())
print(data.describe())
```

```
Petal length  Petal Width  Sepal Length  Sepal Width Species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
       Petal length  Petal Width  Sepal Length  Sepal Width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.057333      3.758000     1.199333
std        0.828066     0.435866      1.765298     0.762238
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
50%        5.800000     3.000000      4.350000     1.300000
75%        6.400000     3.300000      5.100000     1.800000
max        7.900000     4.400000      6.900000     2.500000
```

Analytical insights

- Using pandas, numpy and matplotlib you can not just describe and visualise the data. You can obtain insights that show deeper relationships between various data elements.

```
sns.pairplot(data, hue="Species")
```



Lab 1.1.1

This lab provides practice in:

1. Creating and manipulating lists
2. For loops
3. Creating dictionaries
4. Importing modules and functions
5. Plotting line, bar, and histogram charts using `matplotlib`
6. Simple customisations when creating visualisations
7. Using the `pandas` module for data import and analysis
8. Using the `seaborn` module for data visualisation

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
```

Institute of Data

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**

An example of a class

```
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

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/

More on NumPy

- Numpy arrays have elements of the same type, so are faster to manipulate than lists which may have elements of mixed types and are stored differently in memory.

- Arrays having more than one dimension have axes: a two-dimensional array has axis 0 running downwards along rows, axis 1 running horizontally across columns.

- Boolean indexing allows arrays satisfying a True/False condition to be returned.

  e.g.

```
arr = np.array([5,3,7,2])
arr > 3                  # returns array([True, False, True, False])
arr[arr > 3]             # returns array([5, 7])
```

- np.where returns the indices where a True/False condition is satisfied

  e.g.

```
np.where(arr > 3)        # returns array([0, 2])
```

Lab 1.1.2: 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

More on Pandas

- The `set_index` function enables one to assign the index column to another list or series
- The `reset_index` function resets the index to the default (numbering starts at 0)
- loc vs iloc: `loc` enables indexing by label, `iloc` enables indexing by integer
- Discuss: what's the difference between df.loc[1:3] and df.iloc[1:3]?

Lab 1.1.3: Pandas

1. Explore 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 matplotlib to explore some of the data in the data frame.

Questions?

Appendices

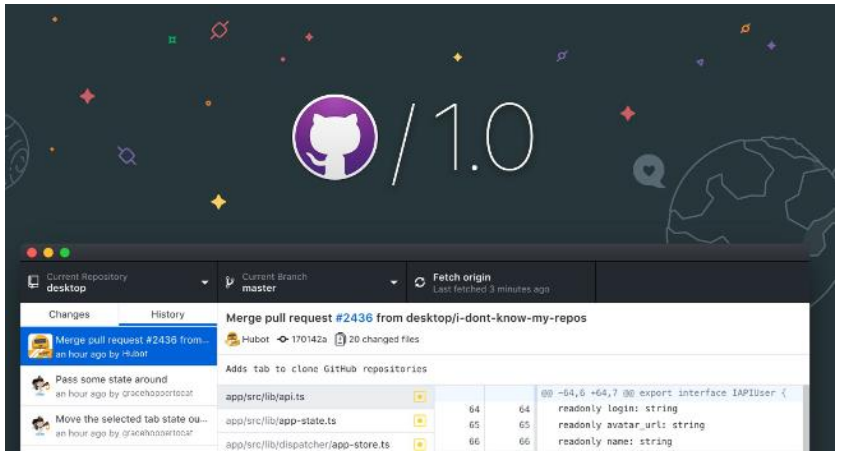
Institute of Data

Version Control with Git & GitHub

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

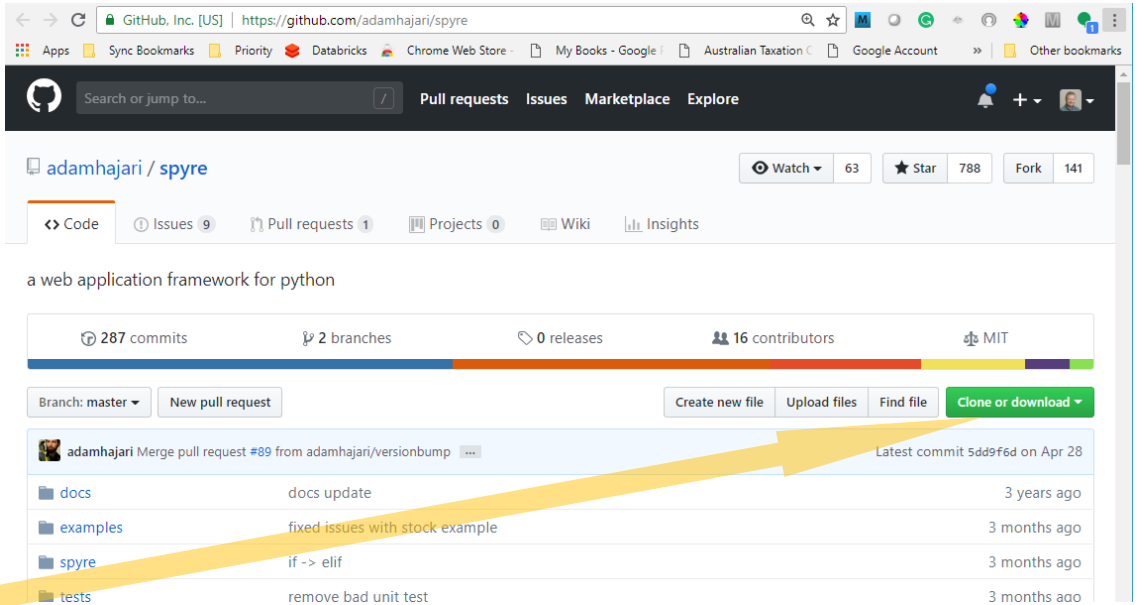
Git & GitHub

- 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



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

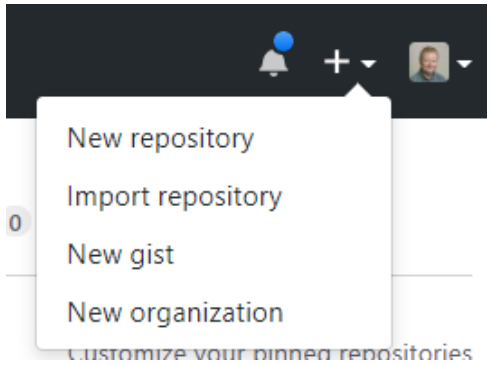


- GitHub Desktop procedure:
  1. click <Clone or download>
  2. click <Open in Desktop>
  3. navigate to target (local) folder
  4. click <Clone>

- 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

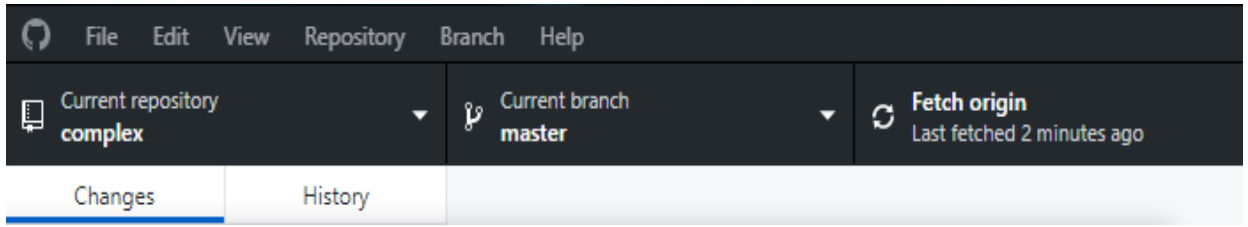


- Creating a branch
  - to allow development in isolation from source repo
    - protects your changes from changes to source
    - rejoin main branch when ready

GitHub: Refreshing Local Repo from Source

Desktop

- <Fetch origin>



Command-line

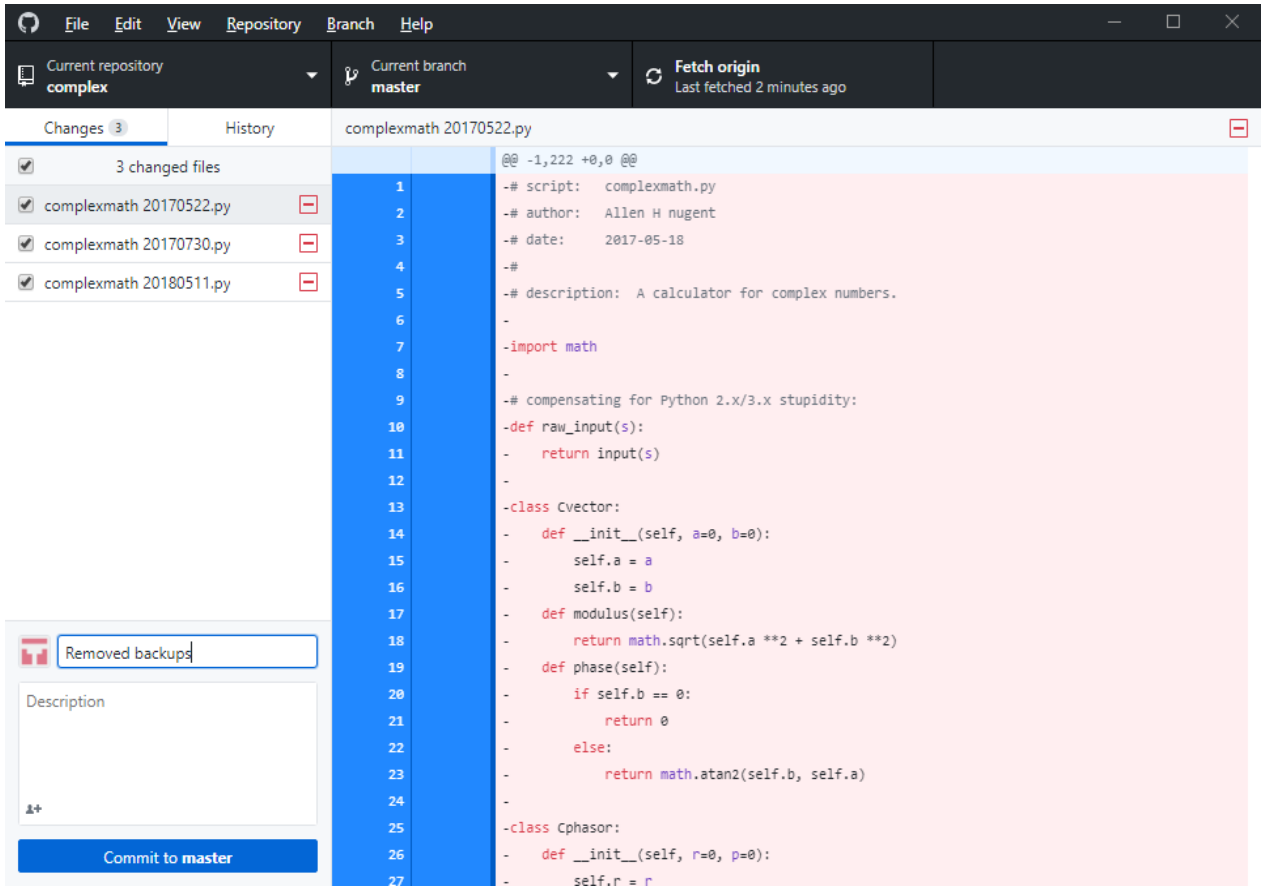
```
$ git checkout master
$ git fetch upstream
$ git merge upstream/master
```

- 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>



GitHub: Commit & Pull Request

Command-line

- commit

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

- pull request

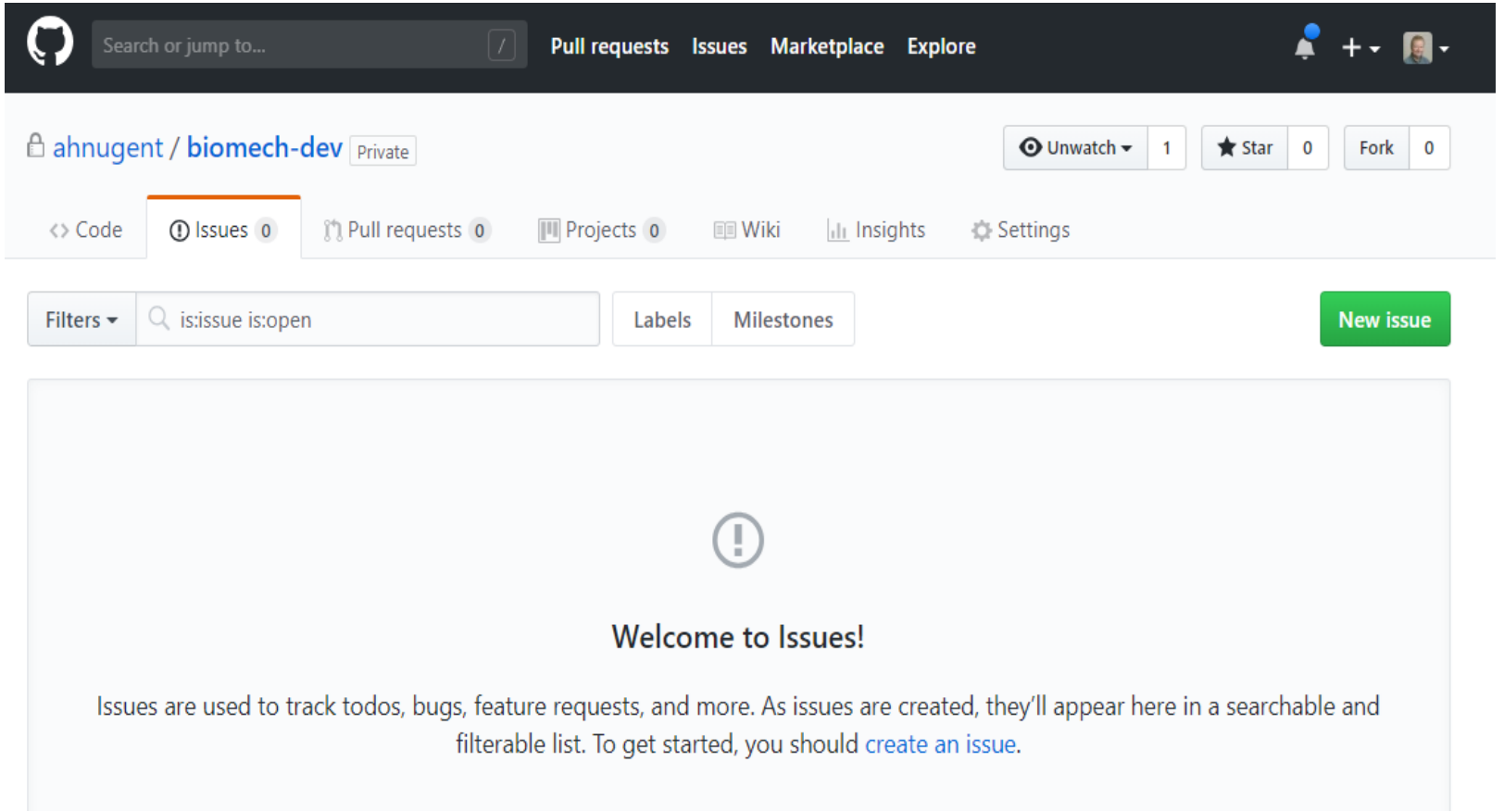
```
$ git push origin master
```

- 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



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!