CORE Skills

Delivering Data Science
In Resources & Energy

Preparatory: Introduction to the tools and concepts

Day 1 & 2

Program tools & participant setup

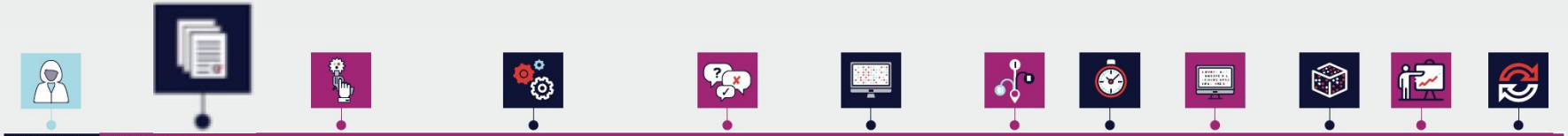
Dr Paul Hancock, Dr Kathryn Napier & Dr Cara Kreck

Curtin Institute for Computation

The material in this tutorial is inspired by & adapted from the ADACS good code etiquette workshop, the Software Carpentry lesson on version control, as well as https://guereslib.github.io/Reproducible-Research-Things/

A CORE partnership

Program Timeline



1 Day Leader Engage	15 Day Professionals Program														
Leading Data Science to Solutions	Data Science Springboard Technical Preparatory		Introduction to Data Projects	Data Analysis			Data & Communication Sandbox	Data Fusion and Machine Learning		Data Fusion Sandbox	Special Data Types - Time-series Data	Special Data Types - Natural Language Processing and Text Mining	Special Data Types - Spatial Data	Capstone Project Development & Presentation	Capstone Propeller
Leader Preparatory	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day7	Day 8	Day 9	Day 10	Day 11	Day 12	Day 13	Day 14	Day 15
Oct 2019	Tues 27 Oct 2020	Wed 28 Oct 2020	Tues 3 Nov 2020	Tues 10 Nov 2020	Wed 11 Nov 2020	Tues 17 Nov 2020	Fri 20 Nov 2020	Tues 24 Nov 2020	Tues 1 Dec 2020	Fri 4 Dec 2020	Tues 8 Dec 2020	Thur 15 Dec 2020	Tues 22 Dec 2020	Tues 12 Jan 2021	Tues 23 Mar 2021
Enabling your people's data science capability in context of the 15 day program	Introduction to the program tools & set up	Introduction to the program tools & set up	Zero to Data Science in a day	Getting to know the program tools - data munging and exploratory data analysis	Simple predictions - regression and statistical model building	Multivariate analysis and model building	Effective data storytelling - communicating results to non-technical audiences	Pros and cons of commonly used statistical and machine learning techniques I	Pros and cons of commonly used statistical and machine learning techniques II	Sandbox - Consolidate approaches covered and test on datasets	The 4th dimension and predictions	Finding needles in wordstacks	Spatial analytics and predictions	Pitching Capstone Projects	Project Review Day

Day 1 – Introducing the Tools & Concepts

Welcome – Preparatory Team

Asking for help

- **Educator/Helpers** You can use the chat function in the ‘Help’ channel in Teams to ask the Educators questions or ask for assistance.
- **Host** Please use the chat function to ask the Host for technical assistance. We will be using the ‘Help’ channel for 1:1 assistance via the direct meeting function.

	Tue 27 Oct	**Wed 28 Oct**
Educator	Kathryn	Cara
Helpers	Ryan Gabriel	Ryan Gabriel
Host	Tamryn and Cara	Paul

Welcome – Material

Materials

- **These Slides with workshop intro:**
  https://tinyurl.com/coreskills00

- **GitHub Repo with the notebooks and data:**
  https://github.com/core-skills/00-Prerequesite

Welcome – Virtual Etiquette

We ask you to please:

- Put yourself on mute to eliminate background noise.
- Turn on your camera if you can.
- If you have a question:
  - Place your question in the chat.
  - Raise your hand virtually using the icon.
  - We will address questions for each section.
- Help Channel
  - If you need help, one on one assistance will be provided in the help channel.
- Be respectful of all participants, through choosing your words purposefully, by giving each other ‘room to speak’ especially in break-out rooms, and by supporting each other.
- We are following the Carpentries’ Code of Conduct.

Set Up – Test your installation

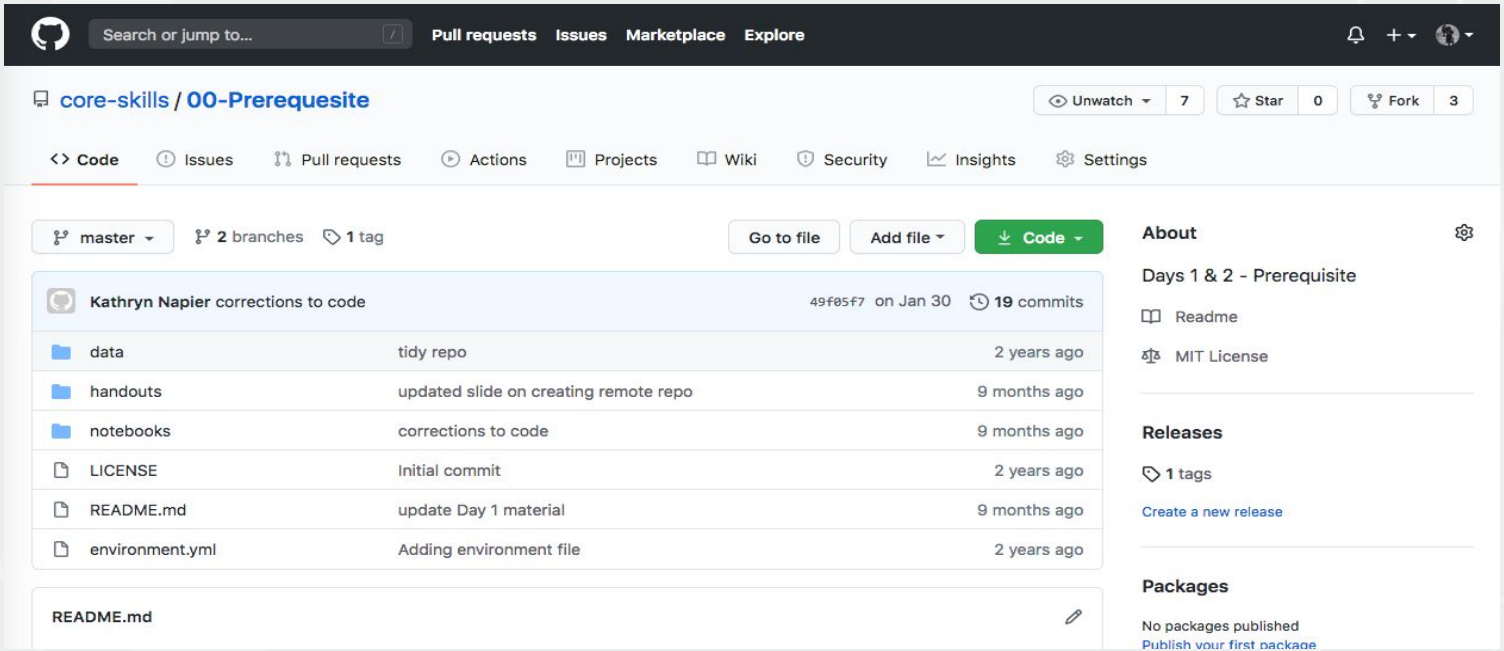
Let's see if our Anaconda installation worked

- Open the **Anaconda Powershell Prompt**:
  - Click Start or Search, and select Anaconda Powershell Prompt
- Open **JupyterLab**:
  - Type: **jupyter lab**
- JupyterLab should open in your web browser
  - Try changing your browser if you have problems

If all this worked for you, we are all set. You can close JupyterLab in the web browser and exit the notebook server in the Anaconda Powershell Prompt with **Ctrl + C**.

Set Up – Download the GitHub Repo

- Go to: https://github.com/core-skills/00-Prerequesite

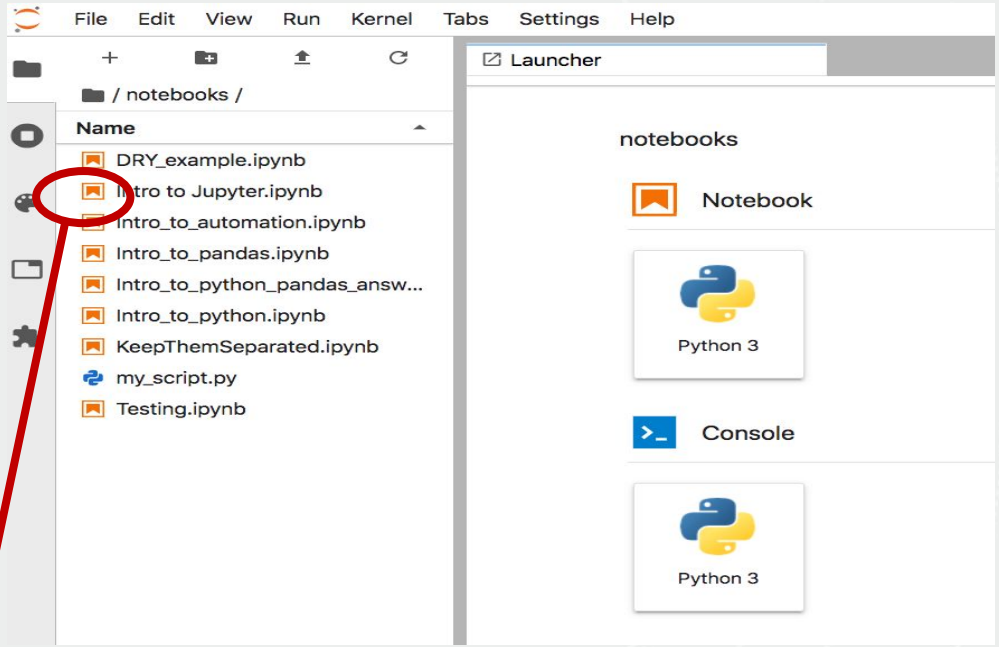


Set Up – If your local installation works

USING ANACODA POWERSHELL PROMPT

If you successfully installed Anaconda and Python and downloaded the Repo:

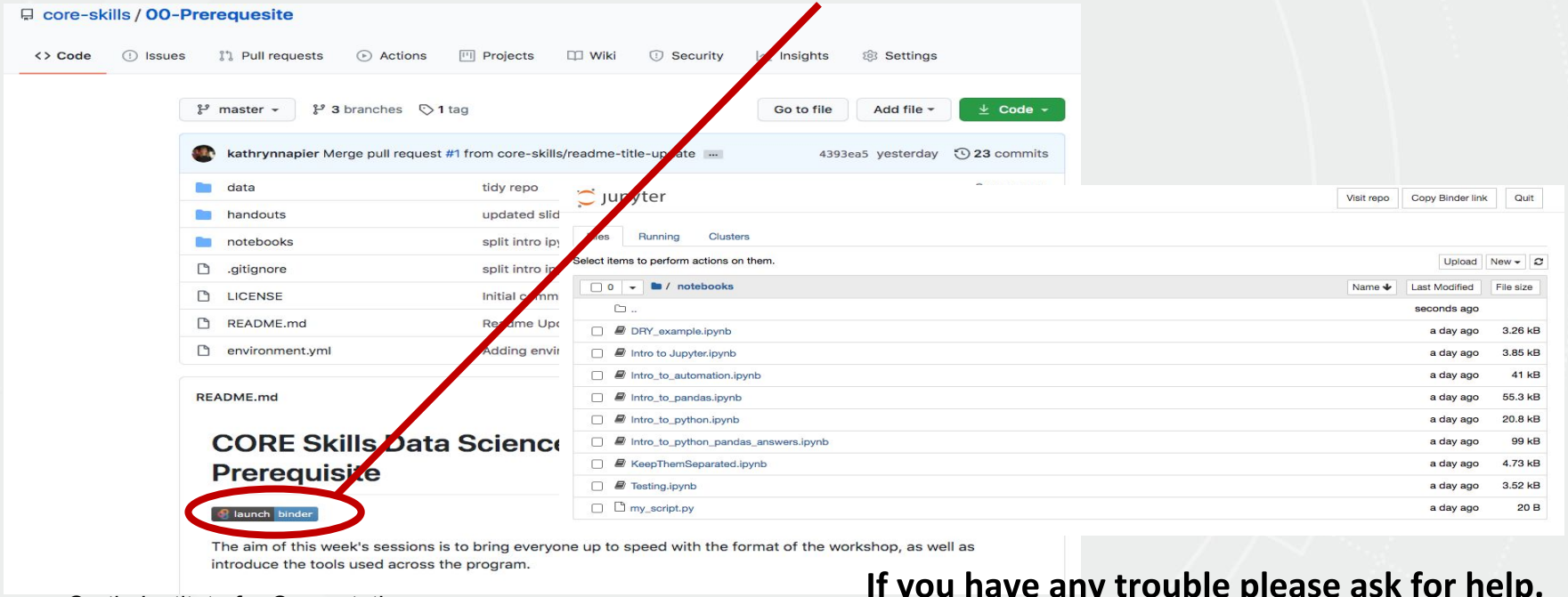
- Unzip the folder to make sure everything downloaded correctly
- Move the folder to a new location if preferred- e.g. to your Desktop
- Open the Anaconda Powershell and launch JupyterLab by typing **jupyter lab**
    - Try changing your browser if you have problems
    - Navigate to the location where you have placed the Repo folder
    - Navigate to the notebooks folder and open the 'Intro to Jupyter.ipynb' notebook



If you have any trouble please ask for help.

Set Up – If your local installation *does not* work

- Navigate to https://github.com/core-skills/00-Prerequesite and scroll down to the repository README and click on the <Launch Binder> button/badge



If you have any trouble please ask for help.

Concepts

1. Overview of the tools used (Anaconda, Python, Github and Git)
2. Intro to Jupyter Notebooks
3. Navigating your computer
    a. File structure
    b. Using jupyter notebooks and Python to move around your filesystem

-> Intro to Jupyter

-> Intro to Navigation

1. Project set up
    b. Package management and Environments
2. Where to find help

-> Intro to Python

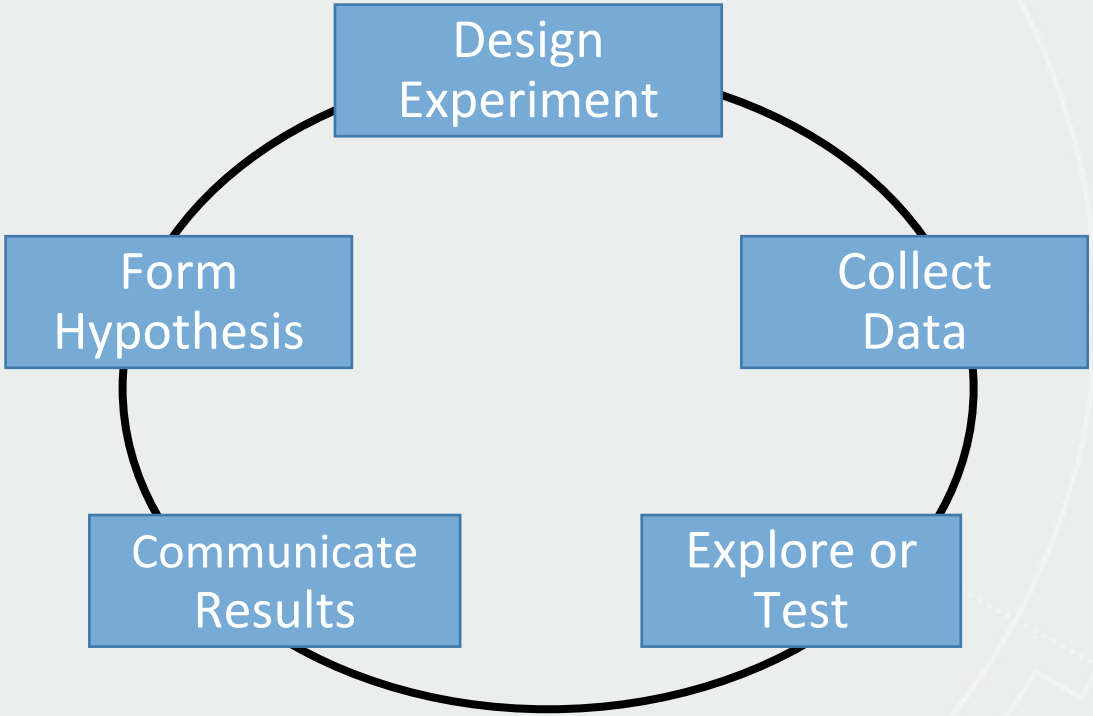
1. Good code etiquette + example notebooks

-> Intro to Pandas

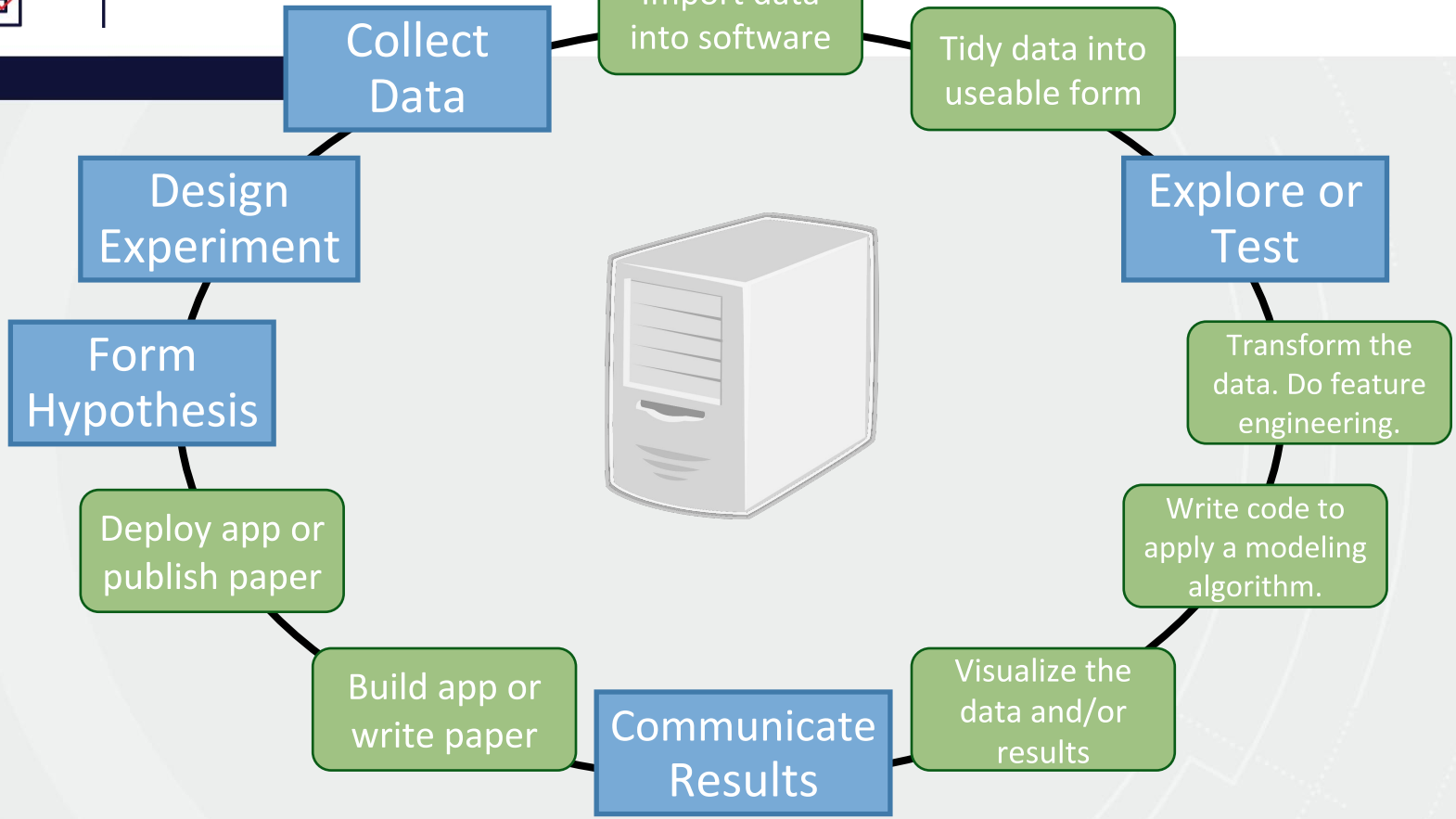
-> Intro to Automation

1. Reproducibility
2. Version control

Data Science

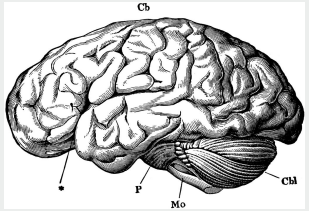


Data Science

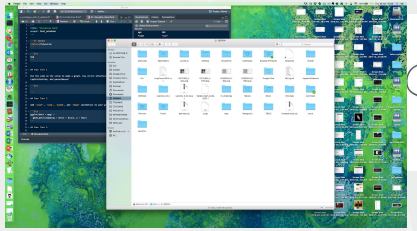


How to communicate with your PC

1. run programs
2. store data
3. communicate with each other, and
4. interact with us

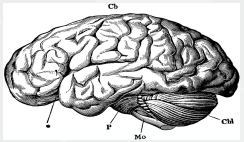


Human thought



- Simple tasks
- One at a time
- Hard to automate or reproduce

Machine language



C++

Human thought

Machine language

Tools used throughout the program

- Distribution of Python and R with commonly used packages and tools
- Includes package and environment management system *conda*

- Web-based, interactive computational environment
- Write code and markdown to share results and work

- High level, interpreted programming language

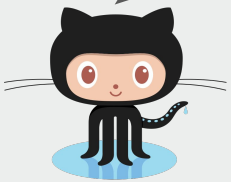
Conda

- Create environments for projects which use incompatible/different versions of packages
- Use it to download and update packages from a central repo

Version Control System

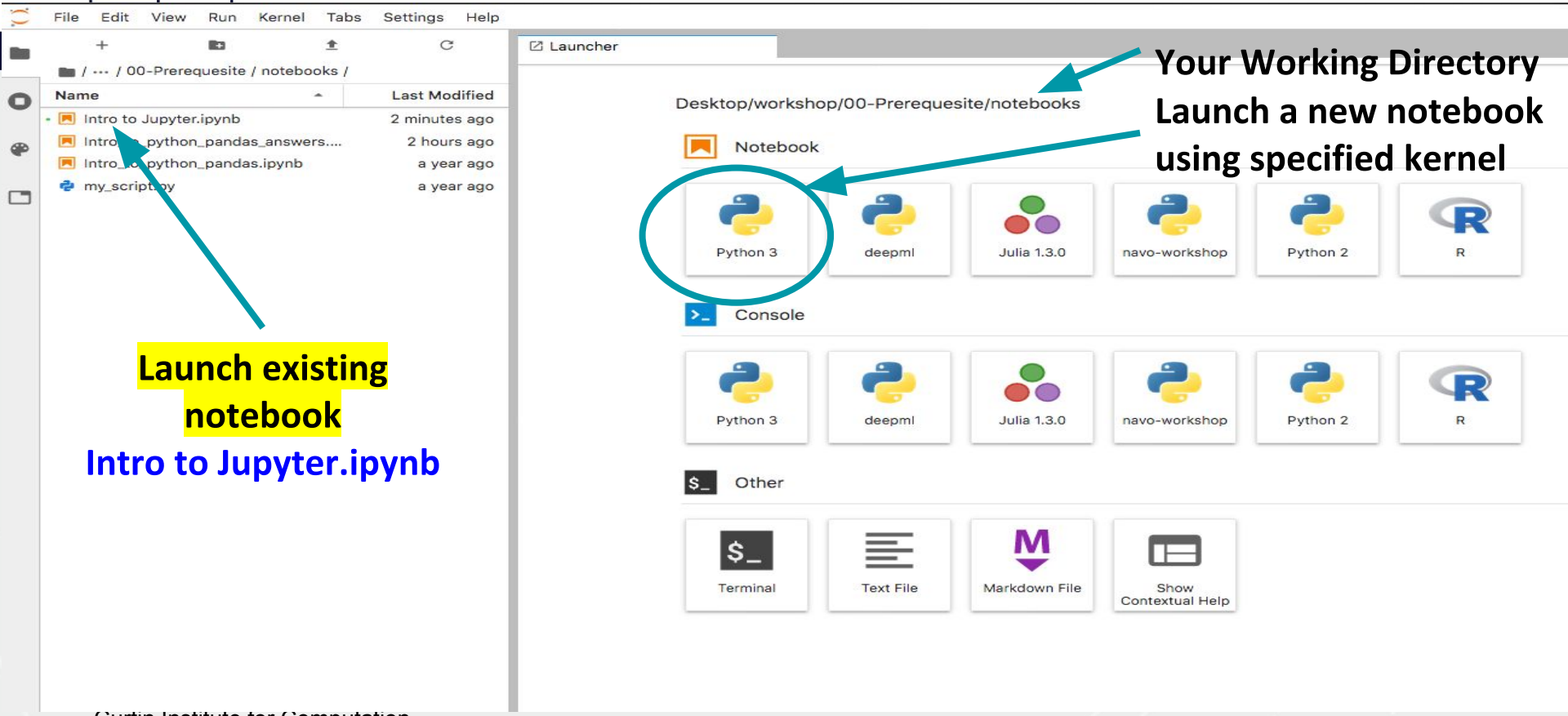
- Keep track of your evolving code

- Github is a remote server where repos can be hosted and shared



- Git Kraken, Github Desktop, etc are user interfaces to git
- They can connect to your local and remote repos
- Many IDEs offer git connectivity too
- List of Git GUI clients

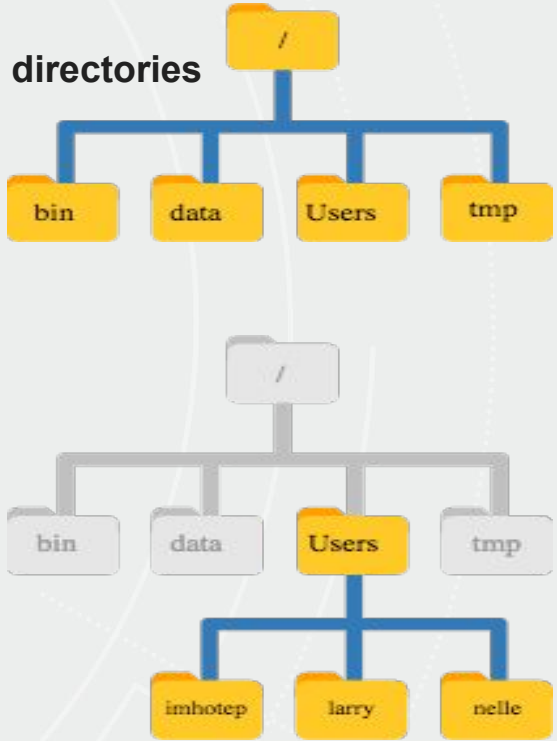
Jupyter Notebooks



Navigating your computer

The part of the operating system responsible for managing files and directories is called the file system.

- The file system is responsible for managing information on the disk.
- Information is stored in files, which are stored in directories (folders).
- Directories can also store other directories, which forms a directory tree.



Every user on a computer will have a **home directory**.

The home directory path will look different on different operating systems. On Linux it may look like /home/nelle, and on Windows it will be similar to C:\Documents and Settings\nelle or C:\Users\nelle

Navigating your computer

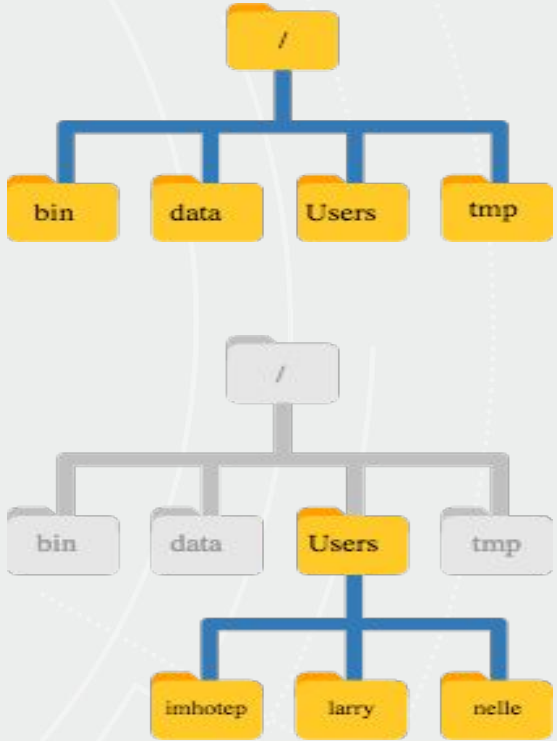
Directory Tree:

At the top is the **root directory** that holds everything else. We refer to it using a slash character, **/, on its own**.

Inside that directory are several other directories, in which are other directories, and so on.

Directories can also contain executables, files and links to directories/executables/files.

Most files' names are **something.extension**. The extension isn't required, and doesn't guarantee anything, but is normally used to indicate the type of data in the file, e.g., **.txt**.

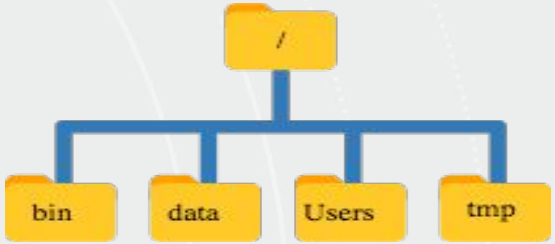


Navigating your computer

Creating a path:

When creating a **path** (i.e. address) to a file we use its location within the directory tree to locate it. To separate directory names in the path name we use / or \:

C:\Users\nelle\report.txt

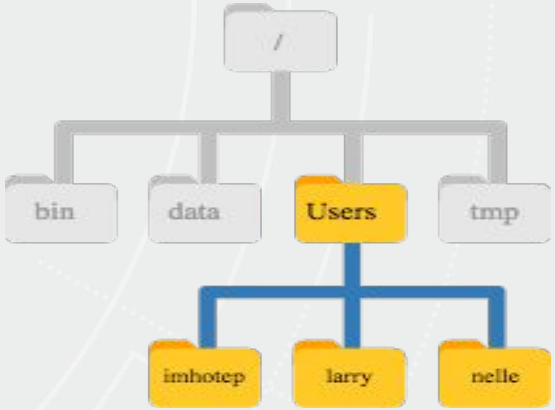


There are two types of paths: **relative path** and **absolute path:**

- A relative path specifies a location starting from the current location.
- An absolute path specifies a location from the root of the file system.

There are also special characters to describe locations in the directory tree:

- .. means ‘the directory above the current one’;
- . on its own means ‘the current directory’.



Navigating your computer

Intro_to_Navigation.ipynb

Project Set up

Folder structures/workflow:

Discuss how you currently organise your files, feel free to talk about a specific dataset you are working on.

Consider the following:

- How do you handle data? Where is it stored?
- How do you keep track of your workflow?
- Do you have a naming convention for directories and files?
- Can you draw your proposed folder structure?

Additional:

See if you can devise a better naming convention or note one or two improvements you could make to how you name your files

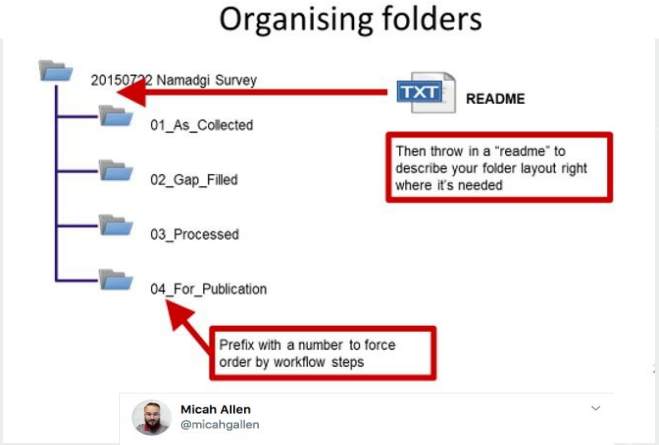
Project Set up

Folder structures:

a. Structure folders **hierarchically** - start with a limited number of folders for the broader topics, and then create more specific folders within these

b. **Separate ongoing and completed work** - as you start to create lots of folders and files, it is a good idea to start thinking about separating your older documents from those you are currently working on

c. Probably the simplest way to **document your structure** - for your future reference - is to add a “README” file - a text file outlining the contents of the folder.



Project Set up

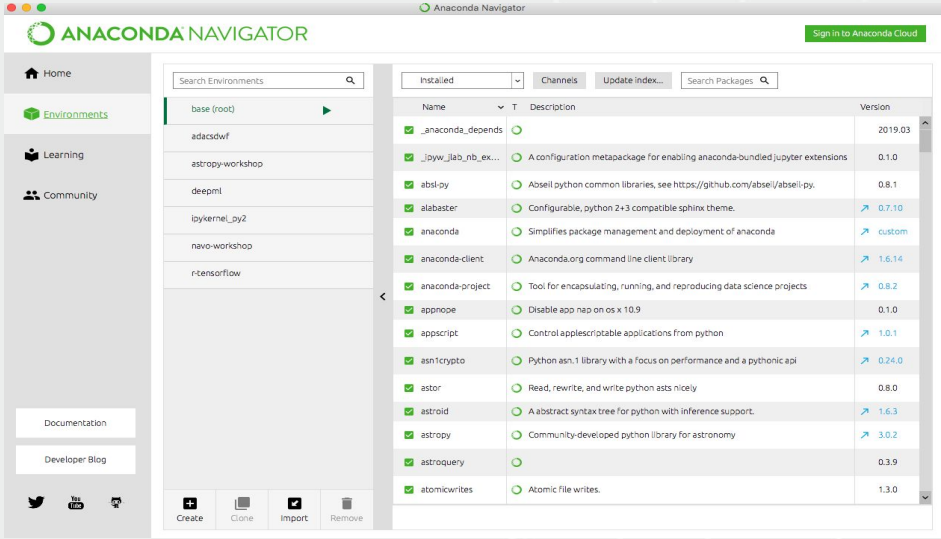
Package and Environment Management:

Package and Environment Management

1. Software and packages have **different versions**
2. Different software and packages can have **different conflicting dependencies**

It is good practice to manage these dependencies by setting up an **environment**.

`conda` is an environment manager which can be used via the command line/Anaconda Powershell Prompt or the anaconda navigator.



Environment set up

Anaconda Powershell Prompt:

Create a new environment using the *environment.yml* file you downloaded with the repo.

1. Open the Anaconda Powershell Prompt
2. Navigate to the folder that contains the Repo, e.g:
   cd Desktop/00-Prerequesite/
3. Create the environment by typing:
   conda env create -n core00 --file environment.yml
4. Activate the environment by typing:
   conda activate core00
5. Deactivate the environment by typing:
   conda deactivate core00

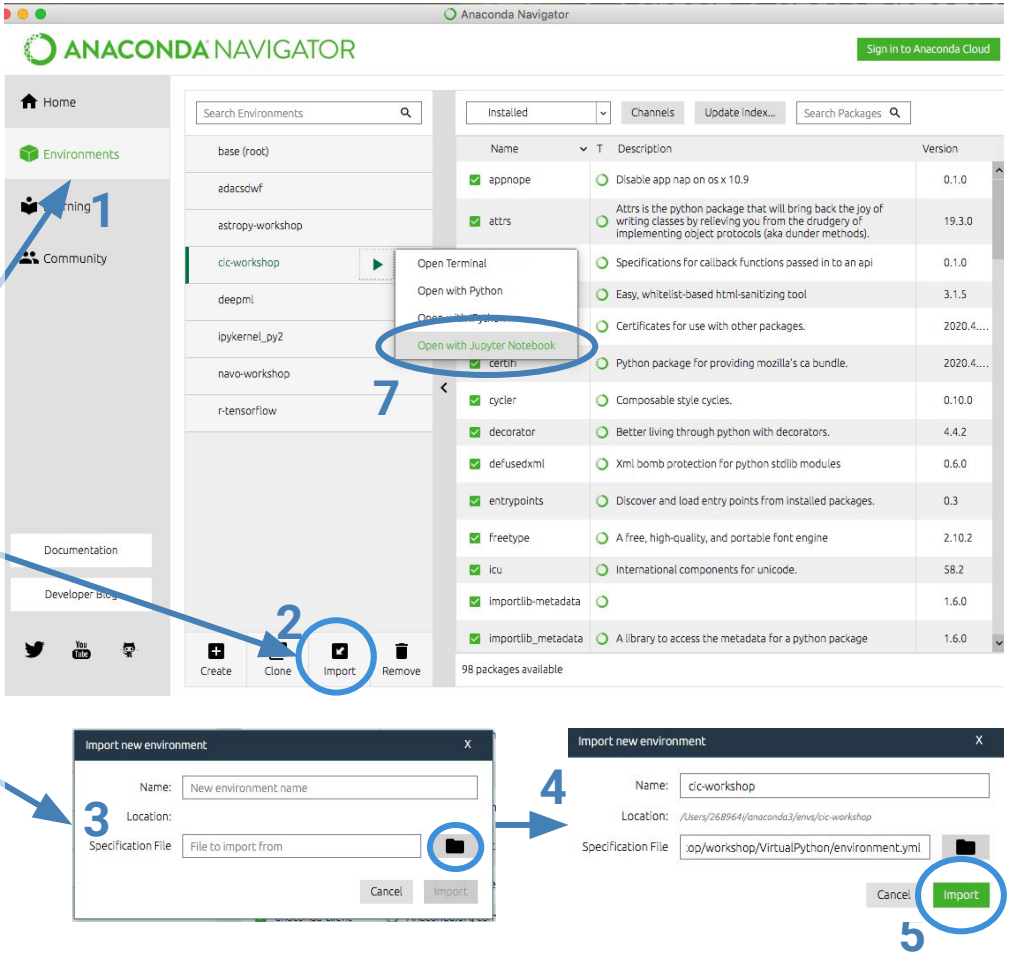
https://conda.io/docs/user-guide/tasks/manage-environments.html

Environment set up

Anaconda Navigator:

Create a new environment by importing the *environment.yml* file you downloaded with the repo:

1. Click on the Environment tab in the left-hand menu
2. Choose the Import button at the bottom of the page
3. In the new dialog window click on the folder icon and search for the *environment.yml* file in the workshop material you downloaded.
4. This should populate the Name, Location and Specification File fields in the dialog box.
5. Click Import
6. Conda is now installing the required version of Python and packages for the workshop
7. Launch a jupyter notebook by selecting the environment and clicking the play button and choosing “Open with Jupyter Notebook”



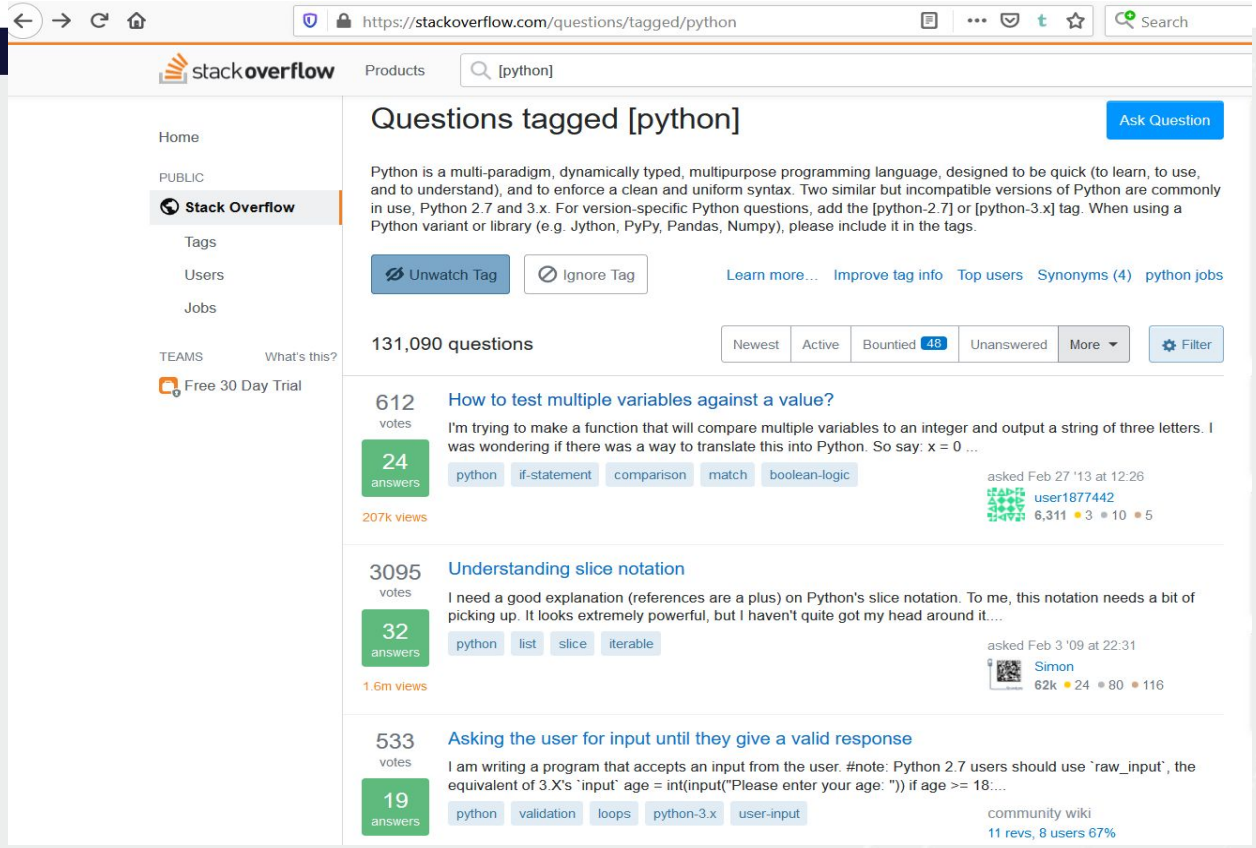
Where to get help

- Read error messages, they are there to assist you
- Google is your friend
- Stackoverflow is your bible
  - Basically Yahoo answers/Quora for code
  - Most questions already exist
- Documentation
  - Inbuilt doc strings
  - Online manuals
  - Tutorials
  - Awesome lists on github
- Join or start a code review group
  - Meetup

```
[2]: # this is a code cell
     2+

       File "<ipython-input-2-557ca13e2cdc>", line 2
         2+
           ^
     SyntaxError: invalid syntax
```

Stack overflow



Inbuilt doc strings

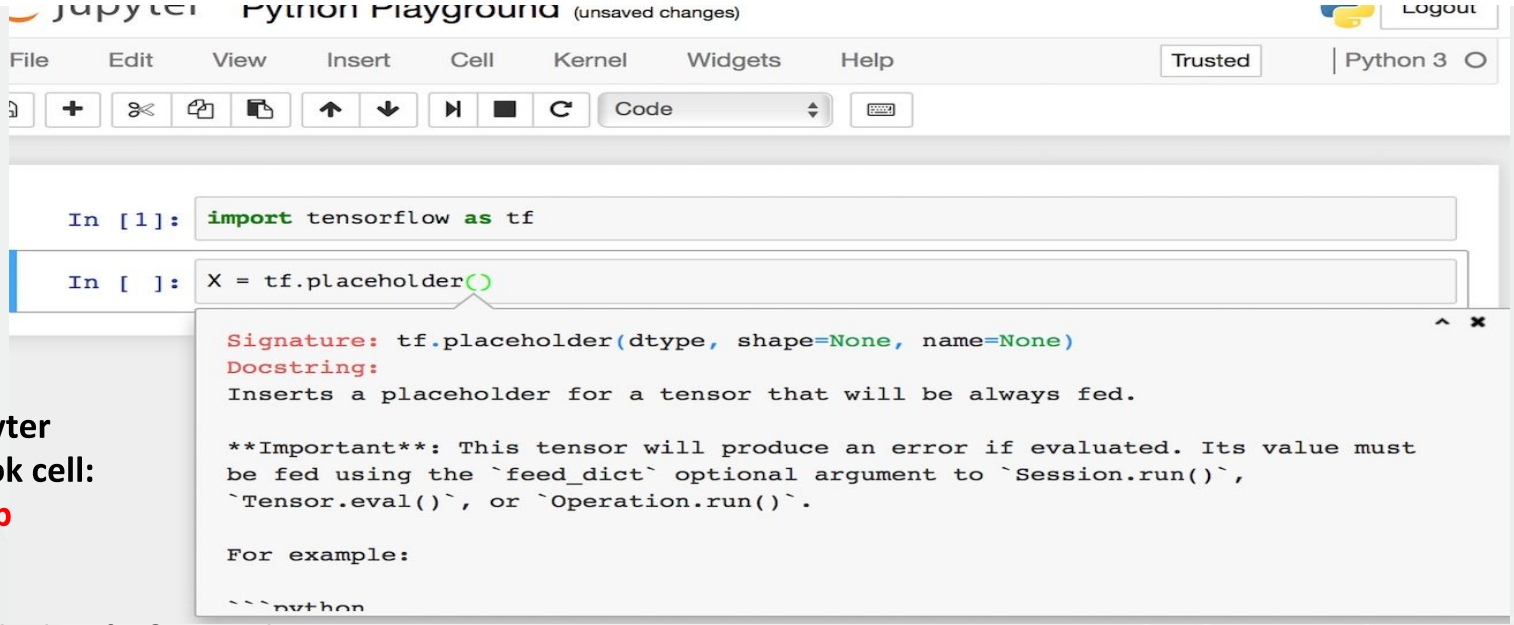
```
In [1]: help(len)
Help on built-in function len in module builtins:

len(...)
    len(object) -> integer

    Return the number of items of a sequence or mapping.
```

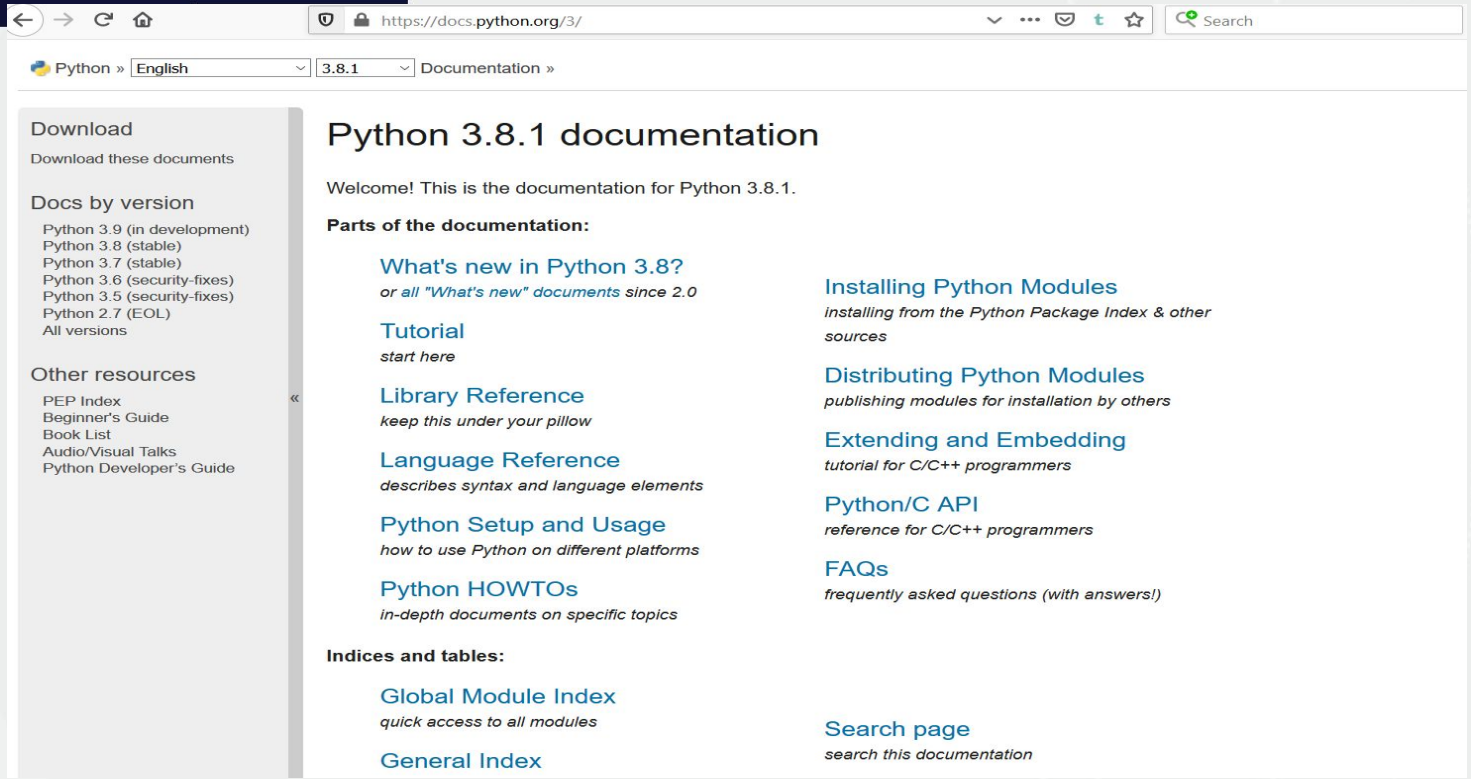
```
In [2]: len?
Type:        builtin_function_or_method
String form: <built-in function len>
Namespace:   Python builtin
Docstring:
len(object) -> integer

Return the number of items of a sequence or mapping.
```

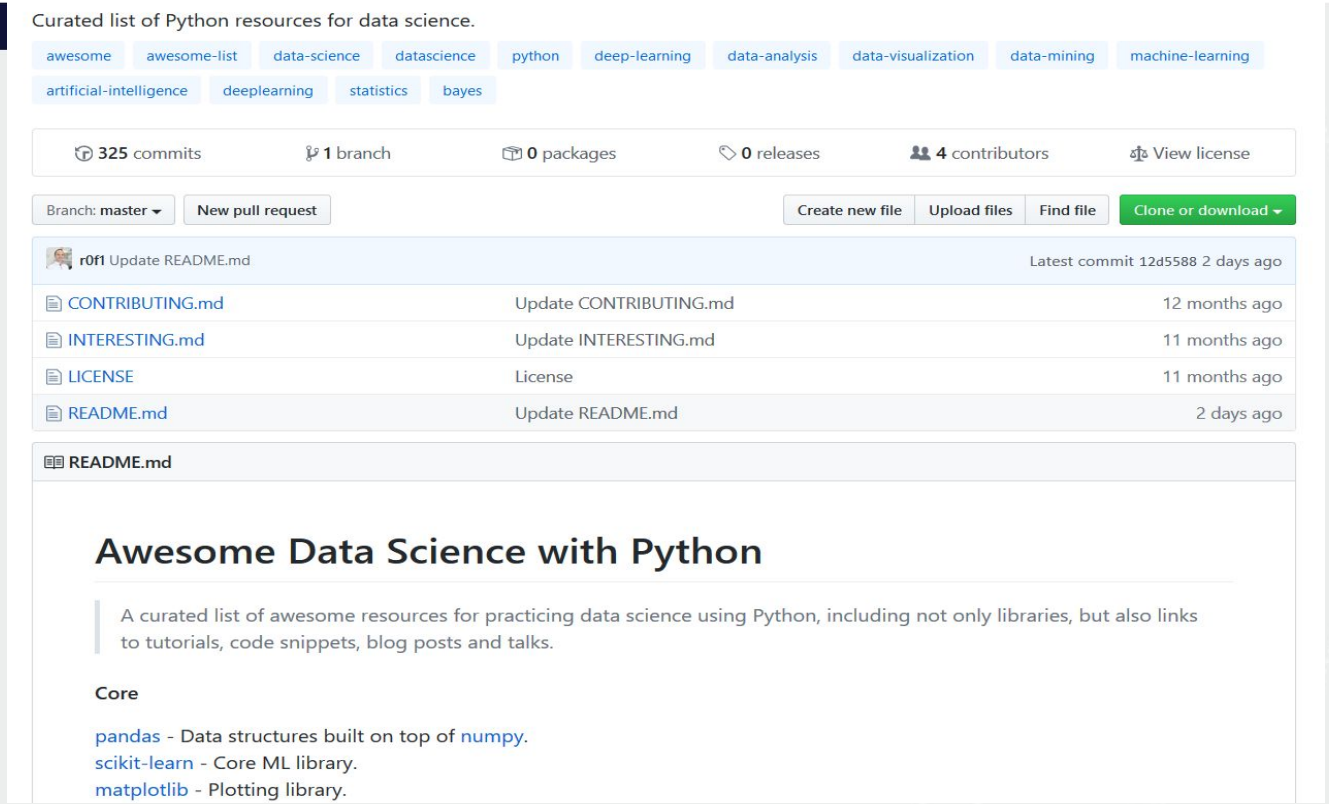


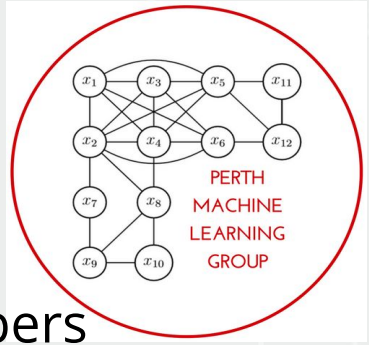
In a jupyter notebook cell:
shift+tab

Online documentation



Github: Awesome lists





Perth Django and Python Developers

Western Australian R Group


Day 1 – Intro to Python

Concepts

1. Overview of the tools used (Anaconda, Python, Github and Git)
2. Intro to Jupyter Notebooks
3. Navigating your computer
    a. File structure
    b. Using jupyter notebooks and Python to move around your filesystem

-> Intro to Jupyter

-> Intro to Navigation

1. Project set up
    b. Package management and Environments
2. Where to find help

-> Intro to Python

1. Good code etiquette + example notebooks

-> Intro to Pandas

-> Intro to Automation

1. Reproducibility
2. Version control

Programming ‘Rules of Thumb’

Programming Rules of Thumb

1. K.I.S.S. (Keep It Simple, Stupid)
   a. Subprograms should do precisely ONE conceptual task and no more.
   b. If a problem can be decomposed into two or more independently solvable problems, do so.
2. Rule of Three
   a. When you copy/paste a piece of code 3 or more times turn it into a function.
3. 90-90 rule (failure to anticipate the hard parts)
   a. "The first 90 percent of the code accounts for the first 90 percent of the development time. The remaining 10 percent of the code accounts for the other 90 percent of the development time."
   b. —Tom Cargill, Bell Labs
4. Efficiency vs clarity (chasing false efficiency)
   a. Never sacrifice clarity for some perceived efficiency.
5. Naming of things
   a. Naming conventions are there to make code easier to read

Style Guides

"Programs must be written for people to read, and only incidentally for machines to execute."
– Harold Abelson, Structure and Interpretation of Computer Program

A style guide is about **consistency**.

Consistency with this style guide is important.
Consistency within a project is more important.
Consistency within one module or function is the most important

[PEP8 style guide]

Why care?

- provides consistency
- makes code easier to read
- makes code easier to write
- makes it easier to collaborate

[Beautify your R code by Saskia Freytag]

Python

- Python Enhancement Proposals https://www.python.org/dev/peps/#numerical-index
- PEP 8 -- Style Guide for Python Code

R

- tidyverse style guide
  - ➢ most comprehensive, underscore for naming conventions
- Advanced R style guide
  - ➢ fairly comprehensive, underscore for naming conventions
- Google style guide
  - ➢ first of its kind, CamelCase for naming conventions

Create readable code

- Python was designed to be readable
- Code-blocks are defined by indentation
- Line continuations are not required
- Syntax is human readable

```
a="""Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
"""

lines = a.split('\n') # \n is the newline character
num_lines = len(lines)

nwords = 0
for line in lines:
    words = line.split()
    nwords += len(words)
```

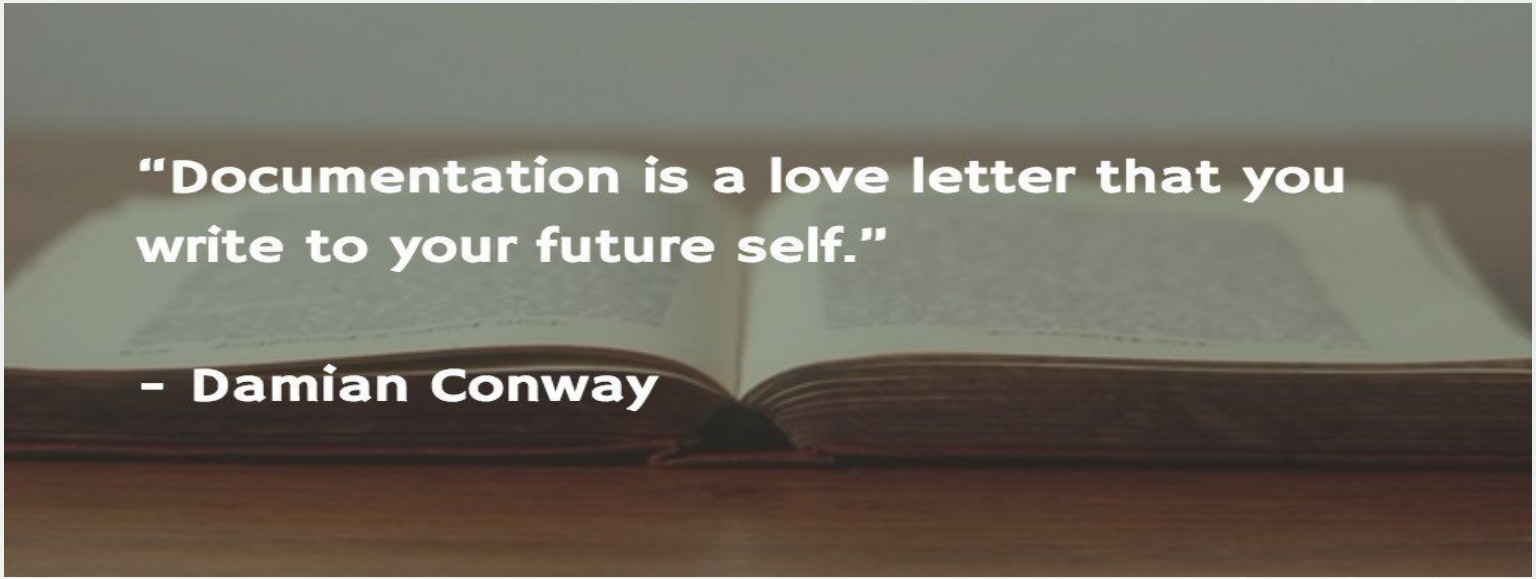
```
def do_lots_of_things(option1, # required
                      option2=0, # not required, has default value
                      option3=1,
                      labels=(),
                      sqlconnecion=None,
                      retries=3,
                      verbose=False,
                      do_print=True):
    pass
```

Naming conventions

Use words! Be verbose but not needlessly so.

- nouns for classes and variables,
- verbs for functions,
- (adjectives for decorators?)
- Underscores_for_functions
- CamelCaseForClasses
- ALL_CAPS_FOR_STATIC_VARIABLES

Document your work



Comments are not documentation

Writing code is not a story that unfolds and entertains people with twists and character developments. It's a **recipe**.

1. Ingredients for the shopping list ⇒ modules to import
2. Description of techniques ⇒ functions
3. Directions ⇒ code in main scope

- Documentation is for people **using** the code (regular folks)
- Documentation describes the ingredients and what kind of cakes are made.
- Comments are for people **reading** the code (ie developers and future you)
- Comments are about the cake making process.

DRY or DIE!

Don't **R**epeat **Y**ourself (**D**uplication **I**s **E**vil)

- Duplicated code means duplicated errors and bugs
- Write a function, call it many times
- Better still,
  - write a **module** in Python and **import** this, or
  - save your collection of **functions** in a separate .R script and **source** it

The DRY principle - II (or DRO maybe?)

Don't **R**epeat **O**thers

- (re-) implementing code often means going through the same growth/development curve of bugs and corner cases
- Common problems have common solutions, use them!
- **'import'** / **'library'** your way to success
- DRY_examples.ipynb

Separate code and data

Having a script that needs to be edited every time it runs is just asking for trouble.

KeepThemSeparated.ipynb

Test code

"Finding your bug is a process of confirming the many things that you believe are true — until you find one which is not true." — Norm Matloff

Use words! Be verbose but not needlessly so.

The only thing that people write less than documentation is test code.

Pro-tip: Both documentation and test code is easier to write if you do it as part of the development process.

1. Write function definition and basic docstring
2. Write function contents
3. Write test to ensure that function does what the docstring claims.
4. Update code and/or docstring until (3) is true.

What to test?

- Whatever you currently do to convince yourself that your code works is a test!
- Everytime you find a bug or some corner case, write a test that will check it.
- Making mistakes doesn't make you a bad person, making the **same mistake** over and over does.

Testing in Python:
https://docs.python-guide.org/writing/tests/

Testing.ipynb

Interactive Development Environment

An IDE is like a text editor but with lots of extra fancy-ness added on.
In fact, you can take your favourite text editor (emacs or vim) and give it an upgrade with plugins that will turn it into more of an IDE.

- Syntax Highlighting and Checking
- Auto Indentation
- Spell Checking (language aware)

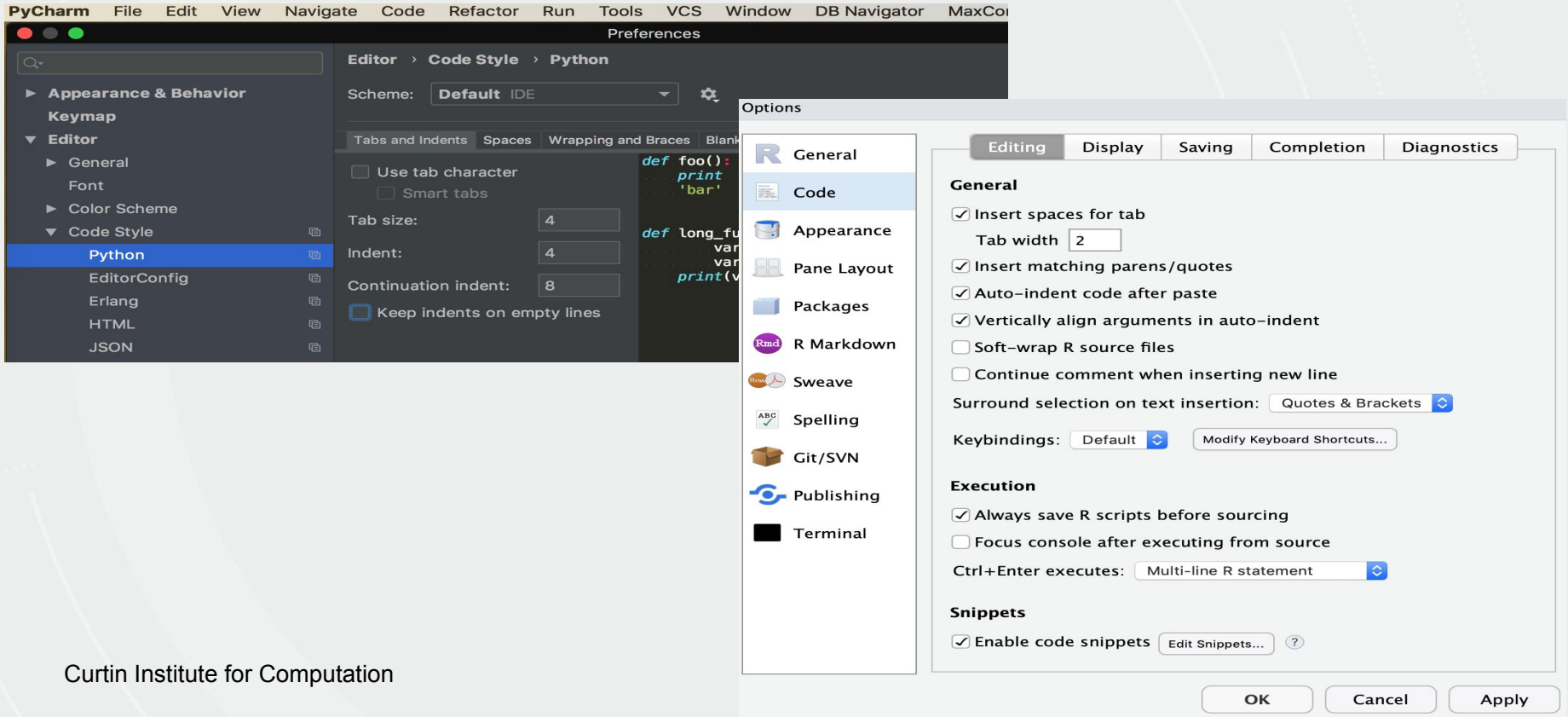
Get a 'real' IDE
Includes: debugging tools, integration with version control, refactoring tools, templates for new modules/files and docstrings.

- PyCharm (not just for python)
- Visual Studio Code
- RStudio (not just for R, RStudio 1.4 will have better support for Python)

Others:

- Spyder (part of anaconda install)
- Emacs and Vim have lots of plug-ins and extensions for this
- Many others -> see what people around you are using

IDEs can help



Recap

Hadley Wickham 
@hadleywickham

Follow

The only way to write good code is to write tons of shitty code first. Feeling shame about bad code stops you from getting to good code

6:11 AM - 17 Apr 2015

- Writing good code takes practice.
- Reuse things that work for you.
- Develop a support group you can call on for help.
  - We have weekly meet-up groups like hacky-hour
- Share your code on GitHub or similar, with documentation, so others can benefit from your work.
  - People can help you debug by reporting issues and submitting bug fixes via pull requests
  - Remember sharing your code on github does not mean you can be held accountable for its maintenance.

*Publish your code and cite that of others.

Day 1 – Reflections

CORE Skills

A CORE partnership

COREHUB.COM.AU/**SKILLS**