



## Delivering Data Science In Resources & Energy

---

### Preparatory: Introduction to the tools and concepts

#### Day 1 & 2

Program tools & participant setup

---

Dr Paul Hancock, Calvin Pang, Leigh Tyers

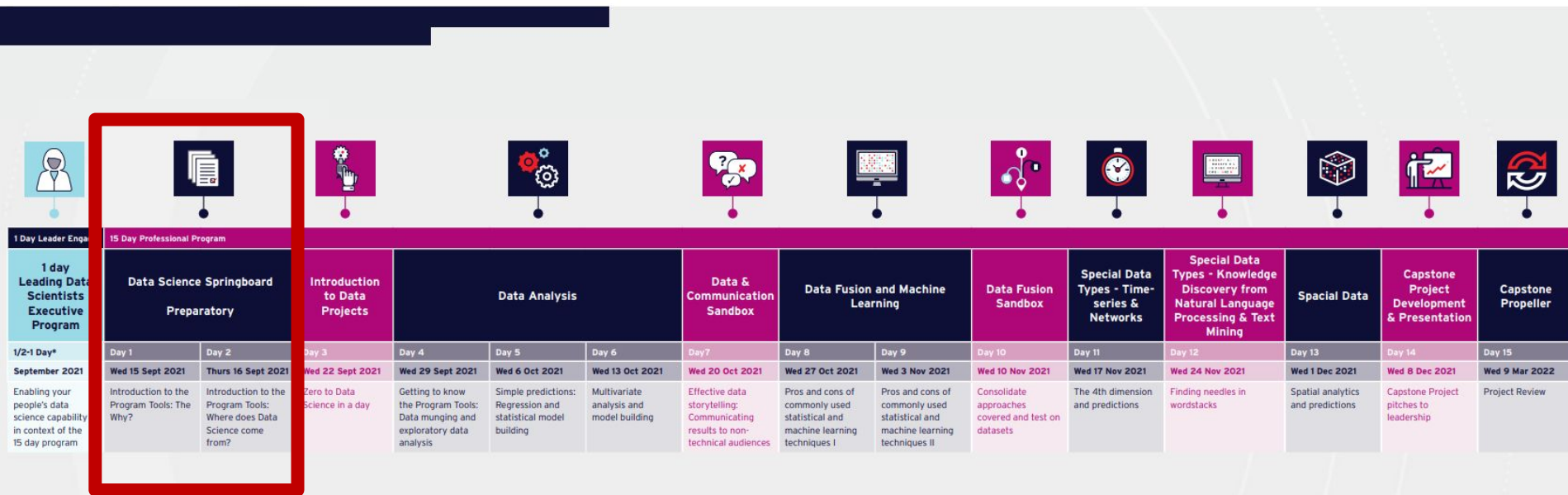
#### Curtin Institute for Computation

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/>





## Program Timeline



## Leading Data Scientists (Executive Program)

To connect leadership thinking to an enabling organisation-wide data science culture, by supporting the 15 Day Data Science Springboard participants' Capstone Project selection, aligning with strategic priorities and ensuring a deployment pathway into the business.

## Data Science Springboard (Professionals Program)

To enhance data science competency of technical, data-centric staff who can execute the organisation's data science strategy, by (a) transforming data into actionable outcomes, (b) evaluating which tool to use, why and when, (c) appreciating good practice in data science, and (d) understanding how to work with data scientists.



# Outline

1. Overview of the tools used
  2. Intro to Jupyter Notebooks
  3. Navigating your computer
  4. Project set up
  5. Where to find help
  6. Intro to Python
  7. Good code etiquette
  8. Data processing with pandas
  9. Automation with python
  10. Reproducibility
  11. Version control
- Day 1
- Day 2





# 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.

	<b>Wed 21/28 Sept</b>
<b>Educators</b>	Paul/Calvin/Leigh
<b>Host</b>	Tamryn



# Welcome – Material

## Materials

- These Slides with workshop intro: <https://tinyurl.com/coreskills00>
- GitHub Repo with the notebooks and data: <https://github.com/core-skills/0102-Preparatory>



# 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.



## Set Up – Test docker/Jupyter is working

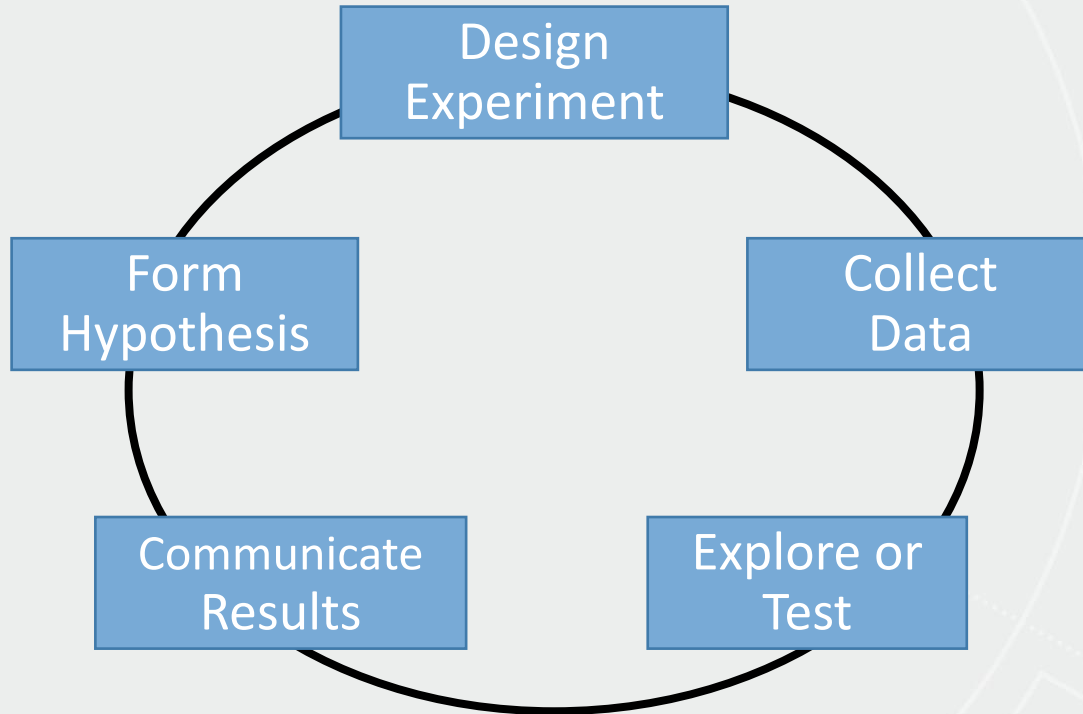
### Check the following:

- That you have docker installed (<https://docs.docker.com/desktop/windows/install/>)
- That you have updated WSL if needed ([link here](#))
- That you have git-bash installed (<https://git-scm.com/downloads>)
- That you have run the scripts in **Core-Skills.zip**
- Finally, run the script called **rio\_cs\_p1.bat**



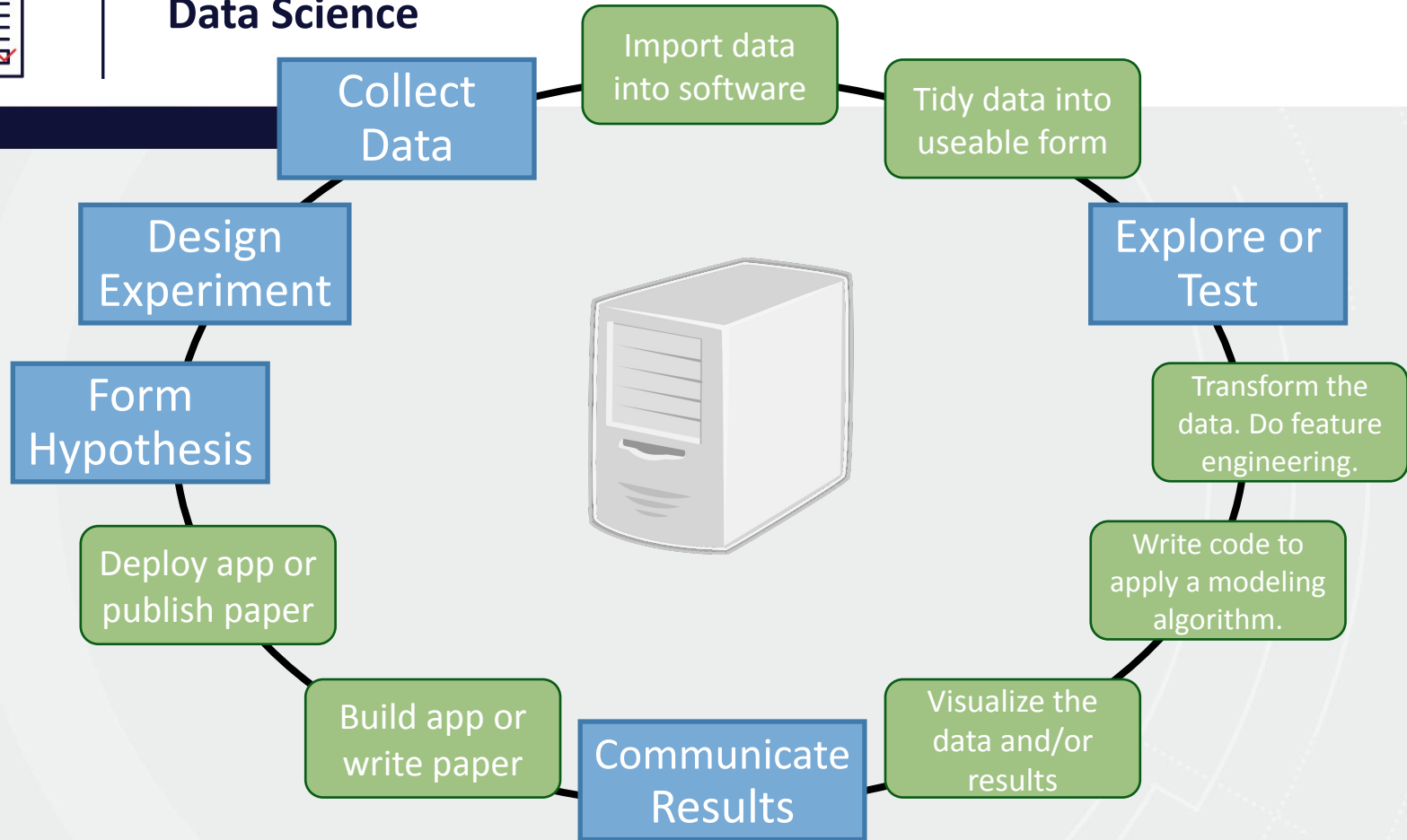


# Data Science





# Data Science

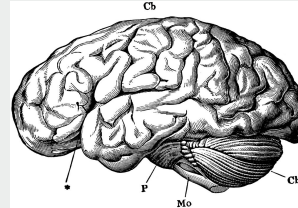




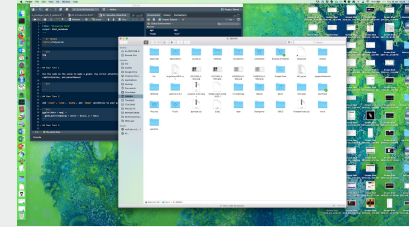
# How to communicate with your PC



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



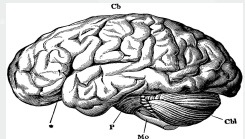
Human thought



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



Machine language



Human thought

Curtin Institute for Computation



C



Machine language



## Tools used throughout the program



### Jupyter Lab

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



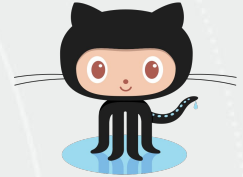
### Python

- High level, interpreted programming language



### Version Control System

- Keep track of your evolving text based documents / code



### GitHub

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



# Jupyter Notebooks

File Edit View Run Kernel Tabs Settings Help

+ -

/ ... / 00-Prerequisite / notebooks /

Name	Last Modified
Intro to Jupyter.ipynb	2 minutes ago
Intro_to_python_pandas_answers....	2 hours ago
Intro_to_python_pandas.ipynb	a year ago
my_script.py	a year ago

**Launch existing notebook**

**Intro to Jupyter.ipynb**

Launcher

Desktop/workshop/00-Prerequisite/notebooks

Notebook

Python 3 deepml Julia 1.3.0 navo-workshop Python 2 R

Console

Python 3 deepml Julia 1.3.0 navo-workshop Python 2 R

Other

Terminal Text File Markdown File Show Contextual Help

**Your Working Directory**

**Launch a new notebook using specified kernel**

# Intro to Jupyter

Open the notebook called:

**00\_Intro\_to\_Jupyter.ipynb**





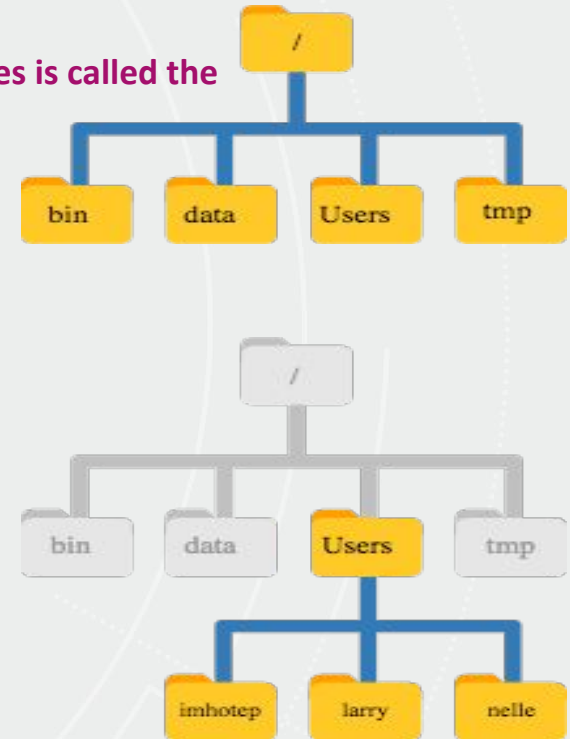
# Navigating your computer

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

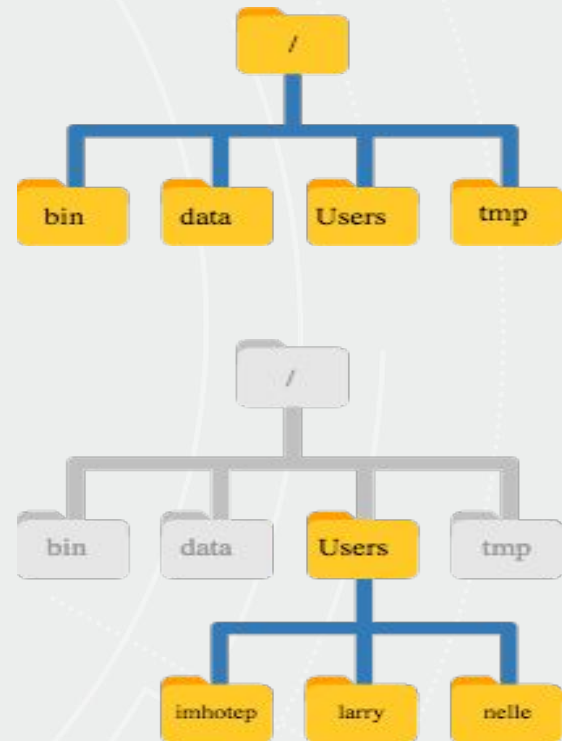
## Directory Tree:

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**:

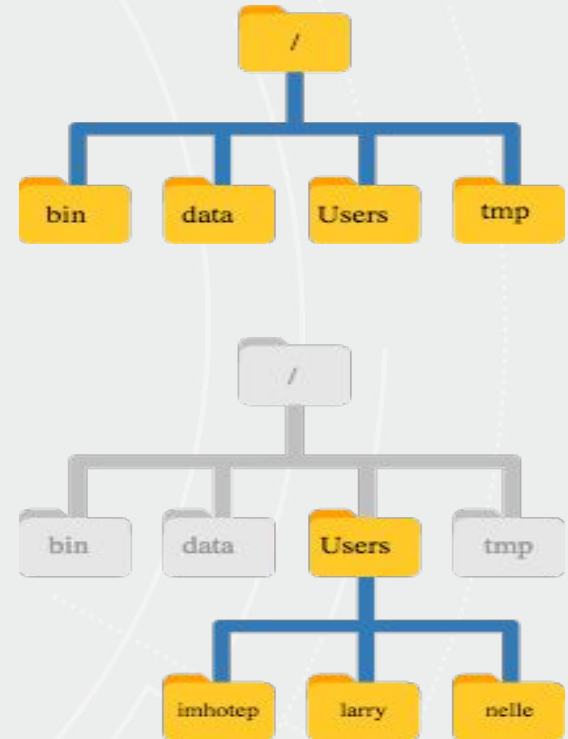
- A relative path specifies a location starting from the current location.
- An absolute path specifies a location from the root of the file system.

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'.
- `~` is the current user's home directory, has to be at the start of specified path

`C:\Users\nelle\report.txt` is equivalent to `~\report.txt`

Curtin Institute for Computation



## Navigating your computer

Open the notebook called:

**01\_Intro\_to\_Navigation.ipynb**



## Project Set up

**5:00**

Stop

### 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

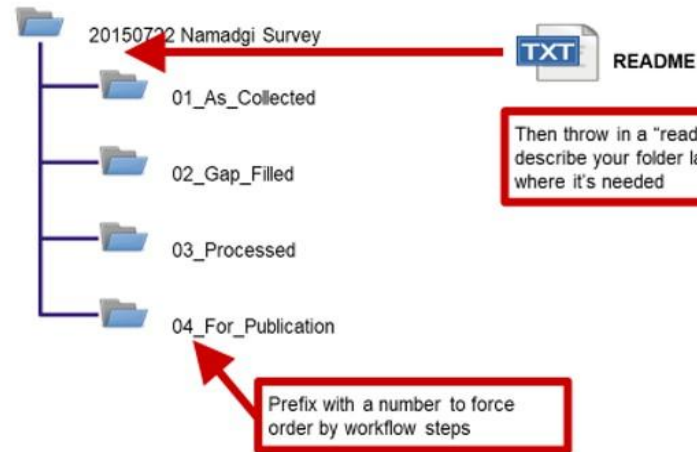


## Project Set up

### Folder structures:

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

## Organising folders



Micah Allen  
@micahgallen

My universal directory structure, forged from years of bad organization:

```
/Projects
.../InProgress
...../ProjectName
...../docs
...../code
...../data
...../figures
.../published
.../submitted
```

You're welcome.





## 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/computing discussion group
  - Meetup
  - Hacky hour

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

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

reversed()

Init signature: reversed(sequence, /)  
Docstring: Return a reverse iterator over the values of the given sequence.  
Type: type  
Subclasses:



# Stack overflow

Stack Overflow

Tags

Users

Jobs

TEAMS

What's this?

Free 30 Day Trial

Products

[python]

Questions tagged [python]

Ask Question

Python is a multi-paradigm, dynamically typed, multipurpose programming language, designed to be quick (to learn, to use, and to understand), and to enforce a clean and uniform syntax. Two similar but incompatible versions of Python are commonly in use, Python 2.7 and 3.x. For version-specific Python questions, add the [python-2.7] or [python-3.x] tag. When using a Python variant or library (e.g. Jython, PyPy, Pandas, Numpy), please include it in the tags.

Unwatch Tag Ignore Tag

Learn more... Improve tag info Top users Synonyms (4) python jobs

131,090 questions

Newest Active Bountied 48 Unanswered More Filter

612 votes

24 answers

207k views

How to test multiple variables against a value?

I'm trying to make a function that will compare multiple variables to an integer and output a string of three letters. I was wondering if there was a way to translate this into Python. So say: `x = 0` ...

python if-statement comparison match boolean-logic

asked Feb 27 '13 at 12:26

user1877442

6,311 3 10 5

3095 votes

32 answers

1.6m views

Understanding slice notation

I need a good explanation (references are a plus) on Python's slice notation. To me, this notation needs a bit of picking up. It looks extremely powerful, but I haven't quite got my head around it...

python list slice iterable

asked Feb 3 '09 at 22:31

Simon

62k 24 80 116

533 votes

19 answers

Asking the user for input until they give a valid response

I am writing a program that accepts an input from the user. #note: Python 2.7 users should use `'raw_input'`, the equivalent of 3.X's `'input'` `age = int(input("Please enter your age: "))` if `age >= 18`...

python validation loops python-3.x user-input

community wiki

11 revs, 8 users 67%



# 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.

Jupyter Python Playground (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [1]: `import tensorflow as tf`

In [ ]: `X = tf.placeholder()`

**Signature:** `tf.placeholder(dtype, shape=None, name=None)`

**Docstring:**  
Inserts a placeholder for a tensor that will be always fed.

**\*\*Important\*\*:** This tensor will produce an error if evaluated. Its value must be fed using the ``feed_dict`` optional argument to ``Session.run()``, ``Tensor.eval()``, or ``Operation.run()``.

For example:

```
```python
```

In a jupyter  
notebook cell:  
**shift+tab**



# Online documentation

← → ↻ 🏠 <https://docs.python.org/3/> 🔍 Search

Python » English 3.8.1 Documentation »

## Download

Download these documents

## Docs by version

- [Python 3.9 \(in development\)](#)
- [Python 3.8 \(stable\)](#)
- [Python 3.7 \(stable\)](#)
- [Python 3.6 \(security-fixes\)](#)
- [Python 3.5 \(security-fixes\)](#)
- [Python 2.7 \(EOL\)](#)
- [All versions](#)

## Other resources

- [PEP Index](#)
- [Beginner's Guide](#)
- [Book List](#)
- [Audio/Visual Talks](#)
- [Python Developer's Guide](#)

## Python 3.8.1 documentation

Welcome! This is the documentation for Python 3.8.1.

### Parts of the documentation:

#### What's new in Python 3.8?

*or all "What's new" documents since 2.0*

#### Tutorial

*start here*

#### Library Reference

*keep this under your pillow*

#### Language Reference

*describes syntax and language elements*

#### Python Setup and Usage

*how to use Python on different platforms*

#### Python HOWTOs

*in-depth documents on specific topics*

#### Installing Python Modules

*installing from the Python Package Index & other sources*

#### Distributing Python Modules

*publishing modules for installation by others*

#### Extending and Embedding

*tutorial for C/C++ programmers*

#### Python/C API

*reference for C/C++ programmers*

#### FAQs

*frequently asked questions (with answers!)*

### Indices and tables:

#### Global Module Index

*quick access to all modules*

#### General Index

#### Search page

*search this documentation*



# Github: Awesome lists

Curated list of Python resources for data science.

[awesome](#) [awesome-list](#) [data-science](#) [datascience](#) [python](#) [deep-learning](#) [data-analysis](#) [data-visualization](#) [data-mining](#) [machine-learning](#)  
[artificial-intelligence](#) [deeplearning](#) [statistics](#) [bayes](#)

325 commits

1 branch

0 packages

0 releases

4 contributors

[View license](#)

Branch: master ▾

[New pull request](#)

[Create new file](#)

[Upload files](#)

[Find file](#)

[Clone or download ▾](#)



r0f1 Update README.md

Latest commit 12d5588 2 days ago

[CONTRIBUTING.md](#)

Update CONTRIBUTING.md

12 months ago

[INTERESTING.md](#)

Update INTERESTING.md

11 months ago

[LICENSE](#)

License

11 months ago

[README.md](#)

Update README.md

2 days ago

[README.md](#)

## Awesome Data Science with Python

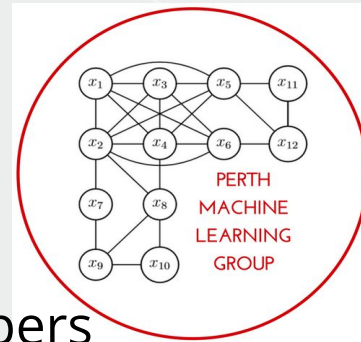
A curated list of awesome resources for practicing data science using Python, including not only libraries, but also links to tutorials, code snippets, blog posts and talks.

### Core

[pandas](#) - Data structures built on top of [numpy](#).

[scikit-learn](#) - Core ML library.

[matplotlib](#) - Plotting library.



## Perth Django and Python Developers Western Australian R Group





# Intro to Python

Open the notebook called:

**02\_Intro\_to\_python.ipynb**



# Programming 'Rules of Thumb'

## Programming Rules of Thumb

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



# Style Guides

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

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

## Python

- Python Enhancement Proposals  
<https://www.python.org/dev/peps/>
- PEP 8 -- Style Guide for Python Code



## Create readable code

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

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

    pass
```

```
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)
```



# Naming conventions

**Use words! Be verbose but not needlessly so.**

- nouns for classes and variables,
- verbs for functions,
- underscores\_for\_functions
- CamelCaseForClasses
- ALL\_CAPS\_FOR\_STATIC\_VARIABLES



## Document your work

**“Documentation is a love letter that you write to your future self.”**

**- Damian Conway**





# 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  $\Rightarrow$  modules to import
  2. Description of techniques  $\Rightarrow$  functions
  3. Directions  $\Rightarrow$  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 Repeat Yourself (Duplication Is Evil)

- 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 Repeat Others

- (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

**Don't repeat yourself or others**

Open the notebook called:

**03\_Using\_functions.ipynb**

## Separate Code and Data

Open the notebook called:

**04\_Separate\_data\_and\_code.ipynb**



## Test code

### Test your code

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.

Think of testing as an investment

`"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`



## 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

Open the notebook called:  
**05\_Testing.ipynb**



# Integrated 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

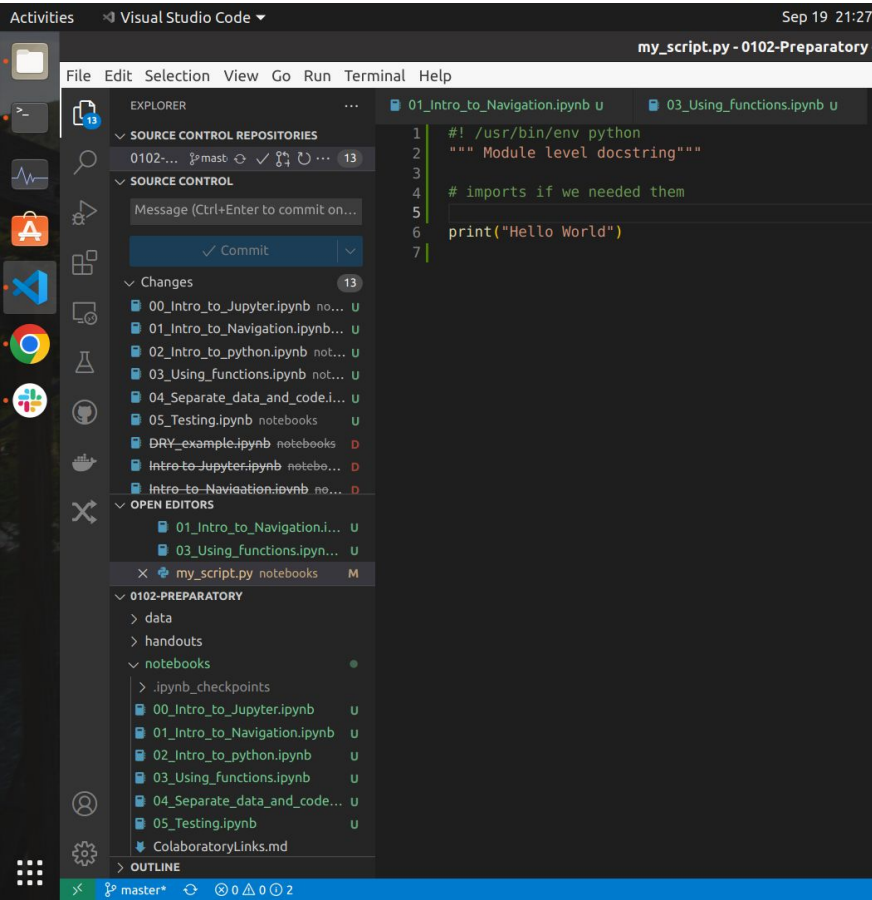
Includes: debugging tools, integration with version control, refactoring tools, templates for new modules/files and docstrings.

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





# IDEs can help



Syntax highlighting (see mistakes early!)

Auto indenting and formatting

Code aware spell checker

Integrated tools like version control



## 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

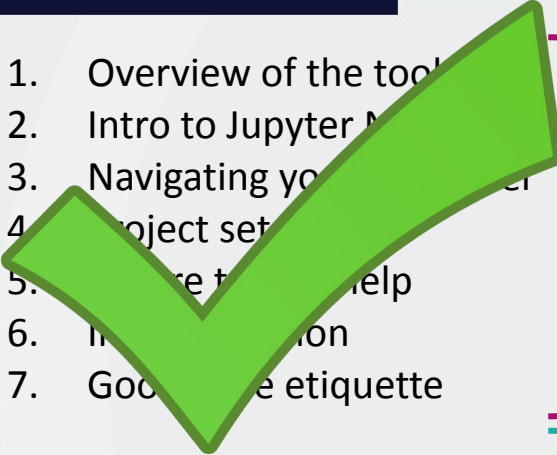
- Writing good code takes practice.
- Reuse things that work for you.
- Develop a support group you can call on for help.
  - We have weekly meetup groups like hacky-hour
- Share your code on GitHub or similar, with documentation, so others can benefit from your work.
  - People can help you debug by 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.

# Day 1 – Reflections

The background is a dark purple gradient. It features a complex network of thin, light purple lines connecting small squares, creating a web-like or molecular structure. There are also larger, fainter squares scattered throughout. On the left side, there are white curved lines and a dotted line. On the right side, there is a large, thick, grey curved line that resembles a stylized arrow or a bracket.



# Outline

- 
- 1. Overview of the tool
  - 2. Intro to Jupyter Notebook
  - 3. Navigating your environment
  - 4. Project setup
  - 5. Getting help
  - 6. Installation
  - 7. Good code etiquette
- Day 1
- 8. Data processing with pandas
  - 9. Automation with python
  - 10. Reproducibility
  - 11. Version control
- Day 2



**COREHUB.COM.AU/SKILLS**

