M269 Overview

M269 Overview Prsntn 2025J

Contents

1 M269 Overview Tutorial Agenda

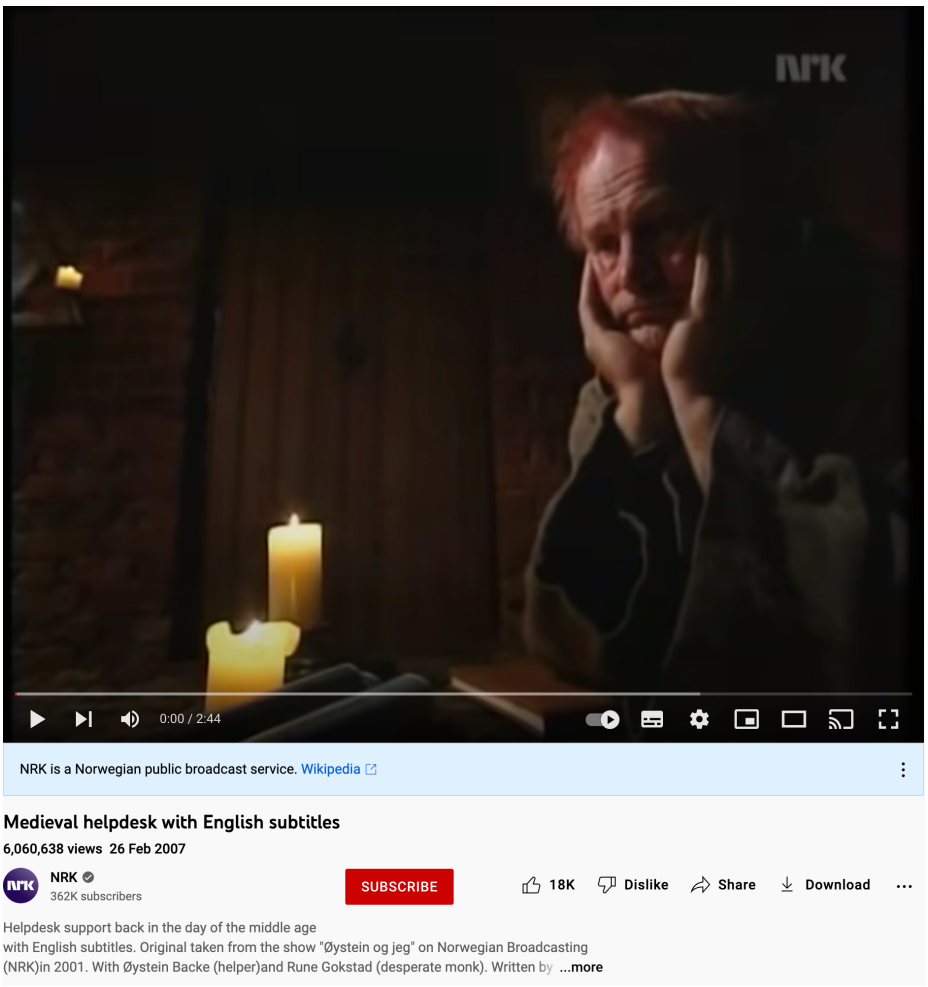
- Introductions
- M269 Overview
- Basic Computational Components
- Course material and software: Jupyter Lab and Python

- Learning Software Packages
- How to Program (in two slides)
- How to survive learning software packages
- *Adobe Connect* — if you or I get cut off, wait till we reconnect (or send you an email)
- Time: about 1 hour
- Do ask questions or raise points.
- Slides/Notes M269Tutorial20241006OverviewPrsntn2024J

Introductions — Me

- *Name* Phil Molyneux
- *Background*
  - Undergraduate: Physics and Maths (Sussex)
  - Postgraduate: Physics (Sussex), Operational Research (Brunel), Computer Science (University College, London)
  - Worked in Operational Research, Business IT, Web technologies, Functional Programming
- *First programming languages* Fortran, BASIC, Pascal
- *Favourite Software*
  - Haskell — pure functional programming language
  - Text editors TextMate, Sublime Text — previously Emacs
  - Word processing in LaTeX — all these slides and notes
  - Mac OS X
- *Learning style* — I read the manual before using the software

Medieval Helpdesk NRK 2001



- Medieval helpdesk with English subtitles

Introductions — You

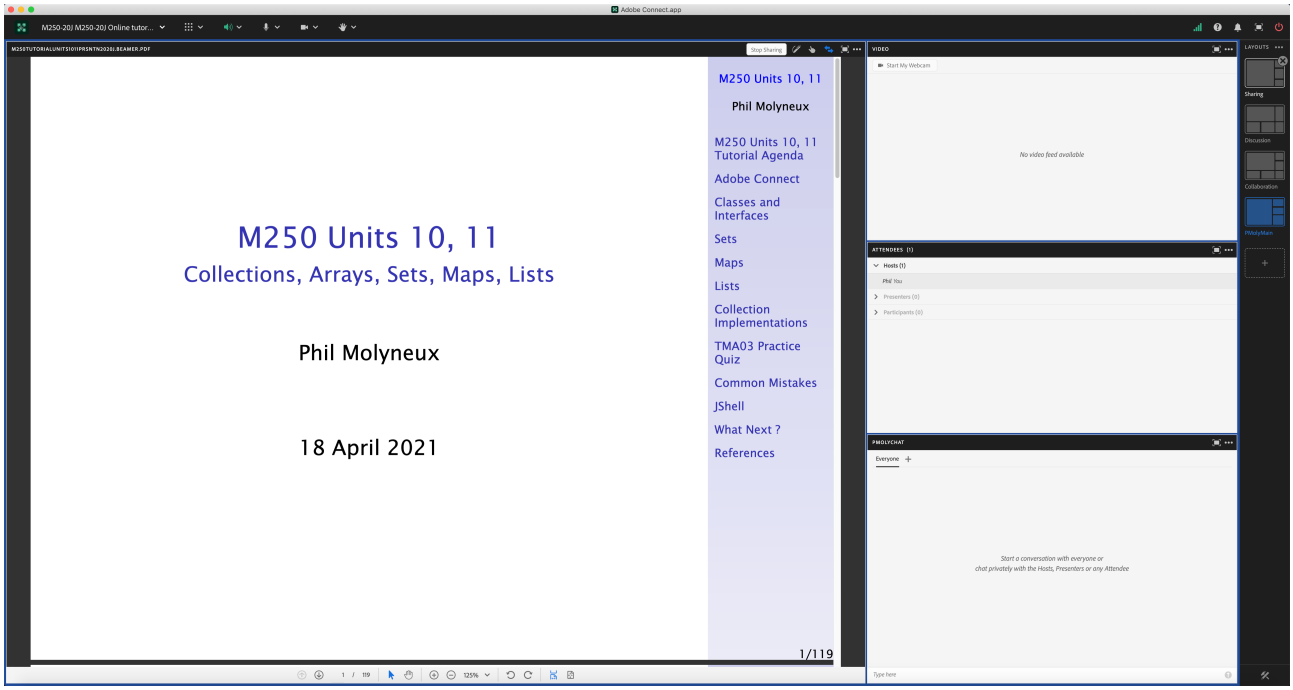
- *Name* ?
- *Favourite software/Programming language* ?
- *Favourite text editor or integrated development environment (IDE)*
- List of text editors, Comparison of text editors and Comparison of integrated development environments
- *Other OU courses* ?
- *Anything else* ?

ToC

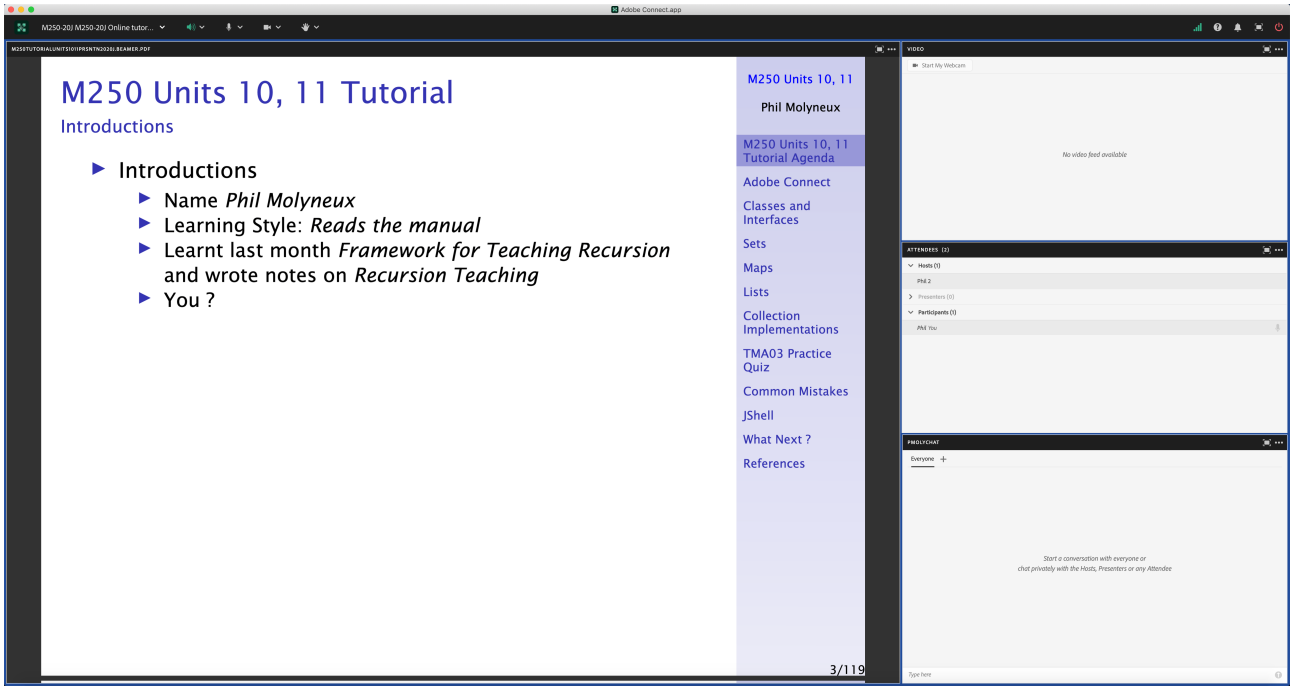
2 Adobe Connect Interface and Settings

2.1 Adobe Connect Interface

Adobe Connect Interface — Host View



Adobe Connect Interface — Participant View



2.2 Adobe Connect Settings

Adobe Connect — Settings

- **Everybody** *Menu bar* › *Meeting* › Speaker & Microphone Setup
- *Menu bar* › *Microphone* › Allow Participants to Use Microphone ✔
- Check Participants see the entire slide including slide numbers bottom right **Workaround**
  - *Disable Draw* *Share pod* › *Menu bar* › *Draw icon*
  - *Fit Width* *Share pod* › *Bottom bar* › *Fit Width icon* ✔

- *Meeting* > Preferences > General > **Host Cursor** > Show to all attendees
- *Menu bar* > *Video* > Enable **Webcam** for Participants ✔
- Do not *Enable single speaker mode*
- Cancel hand tool
- Do not enable green pointer
- **Recording** Meeting > Record Session ✔
- **Documents** Upload PDF with drag and drop to share pod
- Delete *Meeting* > Manage Meeting Information > Uploaded Content and *check filename* > *click on delete*

Adobe Connect — Access

- **Tutor Access**

  TutorHome > M269 Website > Tutorials

  Cluster Tutorials > M269 Online tutorial room

  Tutor Groups > M269 Online tutor group room

  Module-wide Tutorials > M269 Online module-wide room

- **Attendance**

  TutorHome > Students > View your tutorial timetables

- **Beamer Slide Scaling** 440% (422 x 563 mm)
- **Clear Everyone's Status**

  Attendee Pod > Menu > Clear Everyone's Status

- **Grant Access** and send link via email

  Meeting > Manage Access & Entry > Invite Participants...

- **Presenter Only Area**

  Meeting > Enable/Disable Presenter Only Area

Adobe Connect — Keystroke Shortcuts

- Keyboard shortcuts in Adobe Connect
- **Toggle Mic** ⌘ + M (Mac), Ctrl + M (Win) (On/Disconnect)
- **Toggle Raise-Hand status** ⌘ + E
- **Close dialog box** ⎋ (Mac), Esc (Win)
- **End meeting** ⌘ + \

2.3 Adobe Connect — Sharing Screen & Applications

- Share My Screen > Application tab > Terminal for Terminal
- Share menu > Change View > Zoom in for mismatch of screen size/resolution (Participants)

- (Presenter) Change to 75% and back to 100% (solves participants with smaller screen image overlap)
- Leave the application on the original display
- Beware blued hatched rectangles — from other (hidden) windows or contextual menus
- Presenter screen pointer affects viewer display — beware of moving the pointer away from the application
- First time: System Preferences › Security & Privacy › Privacy › Accessibility

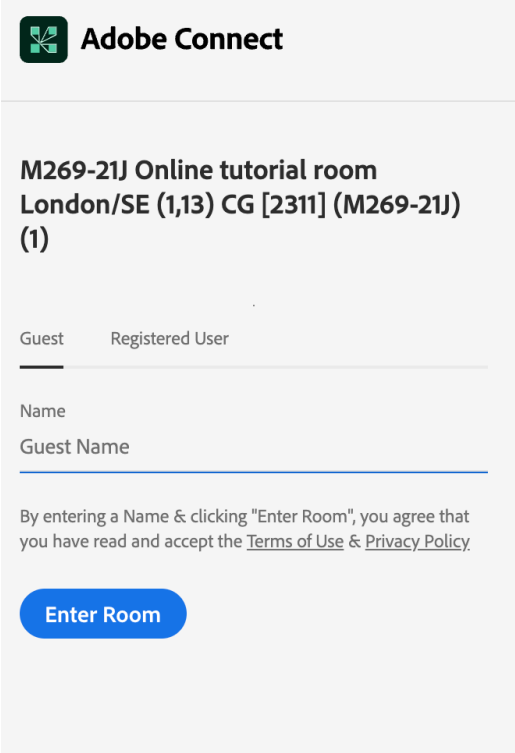
2.4 Adobe Connect — Ending a Meeting

- *Notes for the tutor only*
- **Student:** Meeting › Exit Adobe Connect
- **Tutor:**
- **Recording** Meeting › Stop Recording ✔
- **Remove Participants** Meeting › End Meeting... ✔
  - Dialog box allows for message with default message:
  - *The host has ended this meeting. Thank you for attending.*
- **Recording availability** *In course Web site for joining the room, click on the eye icon in the list of recordings under your recording* — edit description and name
- **Meeting Information** Meeting › Manage Meeting Information — can access a range of information in Web page.
- **Delete File Upload** Meeting › Manage Meeting Information › Uploaded Content tab select file(s) and click Delete
- **Attendance Report** see course Web site for joining room

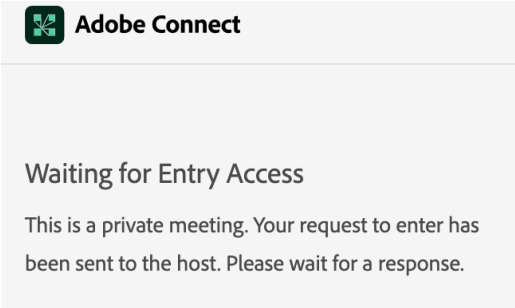
2.5 Adobe Connect — Invite Attendees

- **Provide Meeting URL** Menu › Meeting › Manage Access & Entry › Invite Participants...
- **Allow Access without Dialog** Menu › Meeting › Manage Meeting Information provides new browser window with *Meeting Information* Tab bar › Edit Information
- Check *Anyone who has the URL for the meeting can enter the room*
- Default *Only registered users and accepted guests may enter the room*
- **Reverts to default next session but URL is fixed**
- Guests have blue icon top, registered participants have yellow icon top — same icon if URL is open
- See Start, attend, and manage Adobe Connect meetings and sessions
- Click on the link sent in email from the Host
- Get the following on a Web page

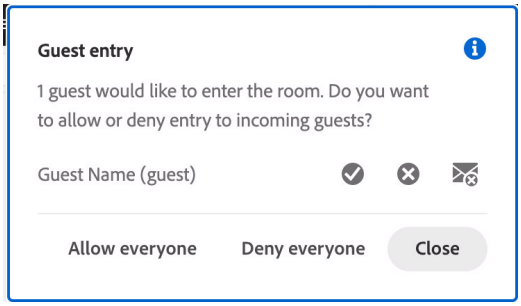
- As *Guest* enter your name and click on Enter Room



- See the *Waiting for Entry Access* for *Host* to give permission



- *Host* sees the following dialog in *Adobe Connect* and grants access



2.6 Layouts

- **Creating new layouts** example *Sharing* layout
- Menu > Layouts > Create New Layout. . . > Create a New Layout dialog > Create a new blank layout and name it *PMolyMain*
- New layout has no Pods but does have Layouts Bar open (see Layouts menu)

- **Pods**
- Menu › Pods › Share › Add New Share and resize/position — initial name is *Share n* — rename *PMolyShare*
- **Rename Pod** Menu › Pods › Manage Pods... › Manage Pods › Select › Rename or Double-click & rename
- Add Video pod and resize/reposition
- Add Attendance pod and resize/reposition
- Add Chat pod — rename it *PMolyChat* — and resize/reposition
- Dimensions of **Sharing** layout (on 27-inch iMac)
  - Width of Video, Attendees, Chat column 14 cm
  - Height of Video pod 9 cm
  - Height of Attendees pod 12 cm
  - Height of Chat pod 8 cm
- **Duplicating Layouts** does *not* give new instances of the Pods and is probably not a good idea (apart from local use to avoid delay in reloading Pods)
- **Auxiliary Layouts** name *PMolyAux0n*
  - Create new Share pod
  - Use existing Chat pod
  - Use same Video and Attendance pods

2.7 Chat Pods

- **Format Chat text**
- Chat Pod › menu icon › My Chat Color
- Choices: Red, Orange, Green, Brown, Purple, Pink, Blue, Black
- Note: Color reverts to Black if you switch layouts
- Chat Pod › menu icon › Show Timestamps

2.8 Graphics Conversion for Web

- Conversion of the screen snaps for the installation of Anaconda on 1 May 2020
- Using GraphicConverter 11
- File › Convert & Modify › Conversion › Convert
- Select files to convert and destination folder
- Click on Start selected Function or ⌘+⏎

2.9 Adobe Connect Recordings

- *Menu bar* > *Meeting* > Preferences > Video
- Aspect ratio > Standard (4:3) (not Wide screen (16:9) default)
- Video quality > Full HD (1080p not High default 480p)
- **Recording** *Menu bar* > *Meeting* > Record Session ✔
- **Export Recording**
- *Menu bar* > *Meeting* > Manage Meeting Information
- *New window* > Recordings > *check Tutorial* > Access Type button
- check Public > check Allow viewers to download
- **Download Recording**
- *New window* > Recordings > *check Tutorial* > Actions > Download File

3 M269 Overview

M269 Algorithms, data structures and computability
Aims

- Ideas of *computational thinking*
- Introduction to algorithms and data structures (using *Python*)
- Logic and the limits of computation
- Computability
- Complexity

M269 Algorithms, data structures and computability
Topics

- Numbers and sequences — functions, complexity, data types
- Booleans and selection — Abstract Data Type (ADT), Decision problems
- Sequences and iteration — control structures for iteration, lists, tuples
- Implementing sequences — arrays as primitives, lists in arrays
- Stack and queues — example ADTs
- Unordered collections — maps, dictionaries, hash tables, sets, bags
- Exhaustive search
- Recursion — some historical context
- Divide and conquer
- Sorting
- Tree data structures
- Graph algorithms 1

- Greedy algorithms
- Graphs 2
- Backtracking
- Dynamic Programming
- Complexity classes
- Computability

ToC

4 Basic Computational Components

Computational Components — Imperative

Imperative or procedural programming has statements which can manipulate global memory, have explicit control flow and can be organised into procedures (or functions)

- **Sequence** of statements

```
stmnt ; stmnt
```

- **Iteration** to repeat statements

```
while expr :
  suite

for targetList in exprList :
  suite
```

- **Selection** choosing between statements

```
if expr : suite
elif expr : suite
else : suite
```

- Some references on structured programming and its history in the 1960s,1970s
- Böhm and Jacopini (1966) *Flow diagrams, Turing Machines and Languages with Only Two Formation Rules*
- Structured program theorem, Structured programming
- Dijkstra (1968) *Letters to the editor: Go To Statement Considered Harmful* See Goto
- Knuth (1974) *Structured Programming with go to Statements*
- Rubin (1987) *GOTO considered harmful considered harmful* Rubin's article created a large correspondence ending in the August CACM (No. 8) with further responses by Rubin and Dijkstra
- The controversy continued with arguments about the use of `break` and `continue`

Functional programming treats computation as the evaluation of expressions and the definition of functions (in the mathematical sense)

- **Function composition** to combine the application of two or more functions — like sequence but from right to left (notation accident of history)

```
(f . g) x = f (g x)
```

- **Recursion** — function definition defined in terms of calls to itself (with *smaller* arguments) and base case(s) which do not call itself.
- **Conditional expressions** choosing between alternatives expressions

```
if expr then expr else expr
```

- The author of these notes regards Recursion as easier than all those dreadful For loops — he may be in a minority on this one but glance at the following
- *Recursion is the GoTo of Functional Programming* was probably a term from Erik Meijer (Meijer et al. (1991) *Functional programming with bananas, lenses, envelopes and barbed wire*)
- By that, Erik meant that a limited number of *Recursion Schemes* are sensible to use just as a limited number patterns of usage of *GoTo* in structured programming are captured by several syntactic constructs such as for loops and *if. . . then. . . else*
- *Structural recursion* is probably the easiest point to start using recursion but often you may get introduced through some *generative* (tricksy) recursion — see How does structural recursion differ from generative recursion?

4.1 Computation, Programming, Programming Languages

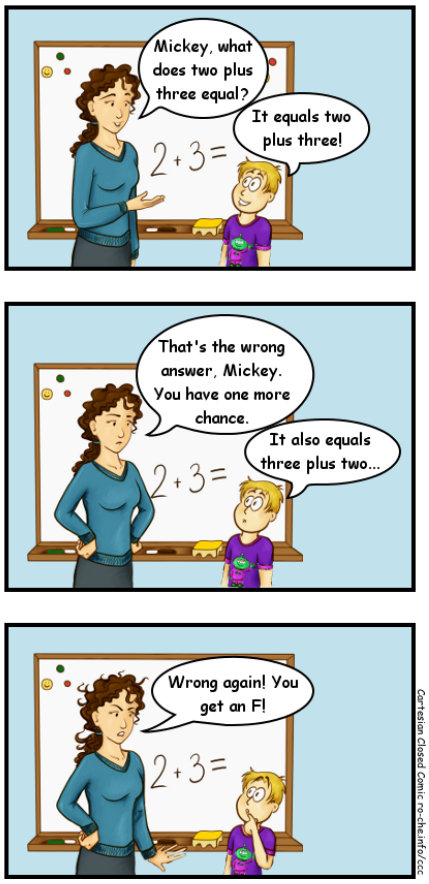
- M269 is not a programming course but . . .
- The course uses Python to illustrate various algorithms and data structures
- The final unit addresses the question:
- *What is an algorithm* ? What is programming ? What is a programming language ?
- So it *is* a programming course (sort of)

Computation, Syntax and Semantics

- **Syntax and Semantics (1)**
- What is each of the following — *first reaction !*
- $4 + 6$
- $4 + 6 \times 3$
- 4
- $19\,370\,721 \times 761\,838\,257\,287$
- The above are *expressions in arithmetic*
  - Most students read *what is* as *evaluate*
  - Not easy for the last one
  - *But* you can say:
  - *They are expressions which when evaluated, evaluate to some number*
  - $19{,}370{,}721 \times 761{,}838{,}257{,}287$
  - $= 147{,}573{,}952{,}589{,}676{,}412{,}927 = 2^{67} - 1$

- demonstrated in a famous meeting of the New York AMS in October 1903 by F.N.Cole (Cole, 1903)

Computation — Cartesian Close Comic Cartoon



Sad fact: many math teachers do not know the difference between equality and reduction.

- **Syntax and Semantics (2)**
- **Evaluate**
- $6 + 4 \times 3$
- $6 - 4 - 1$
- False or True (in Python)
- 5 // 3 (integer division in Python)
- 1 // 0 (in Python)
- False or True or 1 // 0 (in Python)
- **Syntax and Semantics (2a)**

```
Python3>>> 6 + 4 * 3
18  # Why not 30 ?
Python3>>> 6 - 4 - 1
1   # Why not 3 ?
Python3>>> False or True
True
Python3>>> 5 // 3
1
Python3>>> 1 // 0
```

```
Traceback (most recent call last):
  File <stdin>, line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
Python3>>> False or True or 1 // 0
True  # Why did it not crash as before ?
Python3>>>
```

Syntax and Sematics — Elementary Concepts

- An *expression* can be thought of as a program (and vice versa)
- A set of instructions to find a value.
- Operator *precedence* and *associativity* are there to get rid of some brackets
- (to make the code more *user friendly*!)
- **Precedence** — which operator to use first. This is also called *binding power* or operator *fixity*
- **Associativity** — for the same operator, whether to evaluate from left to right or right to left (or it doesn't matter)
- **Lazy Evaluation** — don't do today what you can put off til tomorrow, because you might never have to do it (useful in computation — not useful for doing TMAs)
- **Sharp edges**
- Evaluate (in Maths) 2^2 and 2^{2^2} and $2^{2^{2^2}}$
- In Python `2**2**2**2`
- Alternate in Python `pow(2,pow(2,pow(2,2)))`
- Microsoft Excel `=2^2^2^2`
- or use LibreOffice, Numbers, . . .
- **Sharp edges**
- Evaluate (in Maths) 2^2 and 2^{2^2} and $2^{2^{2^2}}$
- $2^{2^2} = 16$ and $2^{2^{2^2}} = 2^{16} = 65536$ (or 64K in computing)
- Python `2**2**2**2 == 65536`
- Python `pow(2,pow(2,pow(2,2))) == 65536`
- Casio fx-85GT Plus `2^2^2^2` shows `65536`
- Haskell `2^2^2^2 == 65536`
- Microsoft Excel `=2^2^2^2 == 256`
- *Beware language semantics*
- Microsoft Excel `=2^2^2^2^2 == 65536`
- Haskell `length (show (2^2^2^2^2)) == 19729`
- $2^{2^{2^2}}$ has 19729 digits
- What is Excel doing differently ?

ToC

4.2 Programming Languages

- Add a tick on the slide next to languages used
- FORTRAN
- BASIC
- Pascal
- SASL
- C
- Miranda
- Prolog
- JavaScript
- Java
- Haskell
- Add names of other languages used
- Are the following programming languages ?
- Excel
- HTML
- Word
- LaTeX
- SQL
- **Excel**
- Excel has conditional expressions and indirections (so can have loops)
- An Excel Turing Machine is described in Felienne's blog
- Excel see Improving the world's most popular functional language: user-defined functions in Excel
- Announcing LAMBDA: Turn Excel formulas into custom functions (3 December 2020)
- **HTML**
- HyperText Markup Language is the standard markup language for Web pages — it describes the structure of the content.
- It can contain CSS (for describing appearance) and
- JavaScript (for describing behaviour)
- HTML is not a programming language
- JavaScript is a Turing complete programming language but embedded in a host environment.

- CSS could be extended to be Turing complete — see Is CSS Turing complete
- **Word**
- Microsoft Word interface to text formatting
- Serialised with the markup language Office Open XML
- Visual Basic for Applications is embedded and is a programming language
- **LaTeX**
- LaTeX is a format of TeX
- Markup technology for typesetting documents — oriented towards mathematics and technical documents.
- Is also a Turing complete programming language
- Used in MST125 Essential Mathematics 2 Unit 2 *Mathematical typesetting*
- **SQL**
- Structured Query Language based on relational algebra and tuple relational calculus
- Syntactic sugar for first order logic
- Originally not a Turing complete programming language
- but extensions are Turing complete
- **Turing completeness** is not everything
- Data languages such as XML, HTML, JSON
- Regular languages for regular expressions in your favourite text editor (and some programming languages)
- Pushdown automata and Context-free grammars used in program compiling.
- Total Functional Programming requires all programs to be provably terminating.

ToC

5 Python & Jupyter

5.1 M269 Software Installation

- Chapter 1 of the M269 book directs you to **installation instructions** at dsa-ou.github.io/m269-installer/
- **macOS installation** dsa-ou.github.io/m269-installer/install-mac.html
- **Windows installation** dsa-ou.github.io/m269-installer/install-windows.html
- **macOS Jupyter Lab usage**, **Windows Jupyter Lab usage** see the links in the above
- You will need to use PowerShell (Windows) or Terminal (macOS) or a terminal (Unix/Linux)
- Note that the install will add some commands to your startup shell
- nb is created as a shortcut for `jupyter lab &` launched in a particular folder

```
<~><1009> alias m269-25j
m269-25j='cd "{Path to base folder}/M269-25J";
source ~/venvs/m269-25j/bin/activate;
unalias allowed 2> /dev/null'
```

- <~> is my Terminal prompt
- cd changes to given directory (folder)
- source execute the following shell script in the same environment
- activate sets up your Jupyter Lab and Python environment to be separate for other Python environments
- deactivate is defined in the activate script

```
<~><1010> alias nb
nb='source ~/venvs/m269-25j/bin/activate;
unalias allowed 2> /dev/null;
jupyter lab --custom-css &'
```

- This calls jupyter lab
- Note that there are lots of shell initialisation files
- A little knowledge of your shell commands will be useful
- For Zsh see zsh.sourceforge.io, Moving to zsh
- Bash www.gnu.org/software/bash/manual/bash.html
- Windows PowerShell Documentation
- which python3.12 shows the difference

```
%load_ext algoesup.magics
%allowed on
%ruff on
%run -i m269_test
```

- The above are IPython Magics — see Magic functions
- See Built-in magic commands
- The M269 Book section 5.3.2 describe the ones used here
- Intended to check your coding style is in line with various style rules
- Ruff is a Python linter and code formatter written in Rust
- See PEP 8 — Style Guide for Python Code
- PEP 257 — Docstring Conventions
- PEP 20 — The Zen of Python
- PEP 484 — Type Hints

ToC

5.2 Files & Folders

- Some files and folders in macOS are not displayed by default in Finder or Terminal

- These are files with names starting with a dot (.), Library folders (there are several) and some system folders
- Here are several ways of making these files visible
- **Finder (1)** with Finder selected, type ⌘ + ⇧ + . — the keystroke command is a toggle so to turn viewing off just re-type the same
- **Finder (2)** to make the change permanent, type the following in Terminal

```
defaults write com.apple.Finder AppleShowAllFiles true
killall Finder
```

- **Finder (3)** to make a permanent change without using Terminal have a look at TinkerTool (free) — the *Finder* tab first item is *Show hidden and system files*
- Also make sure you display filename extensions with Finder > Preferences > Advanced and check Show all filename extensions
- Obtaining a folder or file path in Finder:
- **(1)** Drag the folder or file into a Terminal window

  the file path is displayed at the command prompt

  most often used with cd to change to a new folder
- **(2)** Ensure you have the *Path Bar* visible

  View > Show Path Bar

  Right-click on the folder in the Path Bar and go Copy *Folder* as Pathname
- **(3)** In Terminal use the pwd command
- **Question** What folders are represented by

  `(.)`

  `(..)`

  `./SomeFolder`

  `/SomeFolder`

  `../SomeFolder`

5.3 Standalone Python Workflow

5.3.1 Learning Python

Learning Python

- Python 3 Documentation
- Python Tutorial
- Python Language Reference
- Python Library Reference
- Stackoverflow on Python
- Lutz (2025) *Learning Python* 6th edition — see About *Learning Python, 6th Edition*

- Martelli et al. (2022) *Python in a Nutshell* (fourth edition)

ToC

5.3.2 Python Workflow

- How do you enter an interactive Python shell ?
- How do you exit Python in *Terminal* (Mac) or *Command prompt* (Windows) ?
- How do you get help in a shell ?
- How do you exit the *interactive help utility* ?
- How do you enter an interactive Python shell ?

  Windows python3 in *Command Prompt*; *Mac python3* in *Terminal*; or *idle3* in either
- How do you exit Python in *Terminal* (Mac) or *Command prompt* (Windows) ?

  `quit()`
- How do you get help in a shell ?

  `help()`
- How do you exit the *interactive help utility* ?

  `quit`

Sequences Indexing, Slices

- `xs[i:j:k]` is defined to be the sequence of items from index i to (j-1) with step k.
- If k is omitted or None, it is treated as `1`.
- If i or j are negative then they are relative to the end.
- If i is omitted or None use 0.
- If j is omitted or None use `len(xs)`
- Th *i* or *j* are omitted or None, they become end values (which end depends on *k*)

Python Quiz — Lists

Given the following definitions

```
xs = [10.9,25,"Phil",3.14,42,1985]
ys = [[5]] * 3
```

Evaluate

```
xs[1]
xs[0]
xs[5]
ys
xs[1:3]
xs[::2]
xs[1:-1]
xs[-3]
xs[:]
ys[0].append(4)
xs1 = xs[::-1]
```

Python Quiz — Lists — Answers

Given the following definitions

```
xs = [10.9,25,"Phil",3.14,42,1985]
ys = [[5]] * 3
```

Evaluate

```
xs[1]            == 25
xs[0]            == 10.9
xs[5]            == 1985
ys               == [[5],[5],[5]]
xs[1:3]          == [25, 'Phil']
xs[::2]          == [10.9, 'Phil', 42]
xs[1:-1]         == [25, 'Phil', 3.14, 42]
xs[-3]           == 3.14
xs[:]            == [10.9, 25, 'Phil', 3.14, 42, 1985]
ys[0].append(4)  == [[5, 4], [5, 4], [5, 4]]
xs[::-1]         == [1985, 42, 3.14, 'Phil', 25, 10.9]
```

Command Line Python Workflow

1. Create *someProgram.py* with assignment statements defining variables and other data along with function definitions.
2. There may be auxiliary files with other definitions, for example *someOtherDefinitions.py* — this uses the `import` statement in *someProgram.py*

   ```
   from someOtherDefinitions import someIdentifier
   ```

3. In *someProgram.py* you can then use `someIdentifier` as a local identifier
4. To import everything

   ```
   from someOtherDefinitions import *
   ```

- See Python Tutorial: Modules
- See Python Language Reference: The `import` statement

1. Create *someDefinitions.py* with assignment statements defining variables and function definitions.
2. In *Terminal* (Mac) or *Command Prompt* (Windows), navigate to *someDefinitions.py* and invoke the *Python 3* interpreter
3. Load *someDefinitions.py* into the *Python 3* with the command

   ```
   import someDefinitions as sdf
   ```

   The `as sdf` gives a shorter qualifier for the namespace — names in the file are now `sdf.x`

   Note that the commands are executed — any print statement will execute, for example
4. At the *Python 3* interpreter prompt, evaluate expressions (remember that they may have side effects and alter the current definitions)

1. For further results, edit the file in *Your Favourite Editor* and use one of the following commands:

```
reload(sdf)

import imp
imp.reload(sdf)
```

Note the use of the name sdf as opposed to the original name.

Read the following references about the dangers of reloading as compared to recycling *Python 3*

- How do I unload (reload) a Python module?
- How to re import an updated package while in Python Interpreter? [duplicate]
- Reloading Python modules
- How to dynamically import and reimport a file containing definition of a global variable which may change anytime

ToC

5.4 Notebook File Format

- Optional topic from Notebook file format
- This is not part of the course for may be of interest
- **Top-level structure**
- metadata (dict)
- nbformat (int)
- nbformat_minor (int)
- cells (list)
- **Top-level structure**

```
{
  "metadata" : {
    "kernel_info": {
        # if kernel_info is defined, its name field is required.
        "name" : "the name of the kernel"
    },
    "language_info": {
        # if language_info is defined, its name field is required.
        "name" : "the programming language of the kernel",
        "version": "the version of the language",
        "codemirror_mode": "The name of the codemirror mode to use [optional]"
    }
  },
  "nbformat": 4,
  "nbformat_minor": 0,
  "cells" : [
      # list of cell dictionaries, see below
  ],
}
```

- **Cell Types**
- Basic structure

```
{
  "cell_type" : "type",
  "metadata" : {},
  "source" : "single string or [list, of, strings]",
```

```
}
```

- Several basic cell types
- Markdown cells
- Code cells
- Raw NBConvert cells
- **Markdown Cells**
- Markdown cells are used for body-text, and contain markdown, as defined in GitHub-flavored markdown, and implemented in marked.

```
{
  "cell_type" : "markdown",
  "metadata" : {},
  "source" : "[multi-line␣*markdown*]",
}
```

- It would be useful to learn some Markdown — and HTML, CSS
- **Mastering Markdown**
- **Daring Fireball: Markdown** the original reference
- **MultiMarkdown** there are lots of extensions
- Markdown Guide and references
- Python Markdown see Fenced Code Blocks
- **Code Cells**

```
{
  "cell_type" : "code",
  "execution_count": 1, # integer or null
  "metadata" : {
      "collapsed" : True, # whether the output of the cell is collapsed
      "scrolled": False, # any of true, false or "auto"
  },
  "source" : "[some␣multi-line␣code]",
  "outputs": [{
      # list of output dicts (described below)
      "output_type": "stream",
      ...
  }],
}
```

- **Code Cell Outputs**
- The `output_type` field defines the output
- **stream** output for text
- **display_data** data keyed by mime-type
- **execute_result** gives results of executing a cell
- **error** messages and traceback
- **Raw NBConvert Cells**
- content that should be included unmodified in nbconvert output

```
{
  "cell_type" : "raw",
  "metadata" : {
    # the mime-type of the target nbconvert format.
    # nbconvert to formats other than this will exclude this cell.
    "format" : "mime/type"
  },
  "source" : "[some nbformat output text]"
}
```

- There are a lot more features than can be covered here
- The main usage here will be adding effects to a cell
- this will require some Markdown, HTML and CSS knowledge (but not much)
- See the documentation

ToC

6 Software & Programming

6.1 Learning Software Packages

Key questions

1. Where is the package source ?
2. What version are you using ?
3. What documentation is available ?
4. What are the *names* for the parts of the interface ?
5. How do you leave the package ? How do you enter the package ?
6. Is there any on-line help and, if so, how is it used ?
7. Are there any initialisation files, configuration or preferences and how are they used ?
8. How do you import and export data from the package ?
9. *When all else fails*, how can you obtain advice ?

- Answer the *Key Questions* for Jupyter Lab
- Where is the package source ?
- What version are you using ?
- Where is the package source ?

  See Installing the Jupyter Software

  Python Packaging User Guide

  pip documentation

  https://jupyterlab.readthedocs.io/en/4.0.x/getting_started/installation.html

  venv — Creation of virtual environments

  M269 Software: Installing and changing the software

- What version are you using ?

```
((m269-25j) ) <M269-25J><1013> jupyter --version
Selected Jupyter core packages...
IPython          : 8.37.0
ipykernel        : 6.29.5
ipywidgets       : 8.1.7
jupyter_client   : 7.4.9
jupyter_core     : 5.8.1
jupyter_server   : 2.16.0
jupyterlab       : 4.4.6
nbclient         : 0.10.2
nbconvert        : 7.16.6
nbformat         : 5.10.4
notebook         : 6.5.7
qtconsole        : not installed
traitlets        : 5.14.3
```

- Answer the *Key Questions* for Jupyter Lab
- What documentation is available ?
- Answer the *Key Questions* for Jupyter Lab
- What documentation is available ?

  Jupyter Documentation

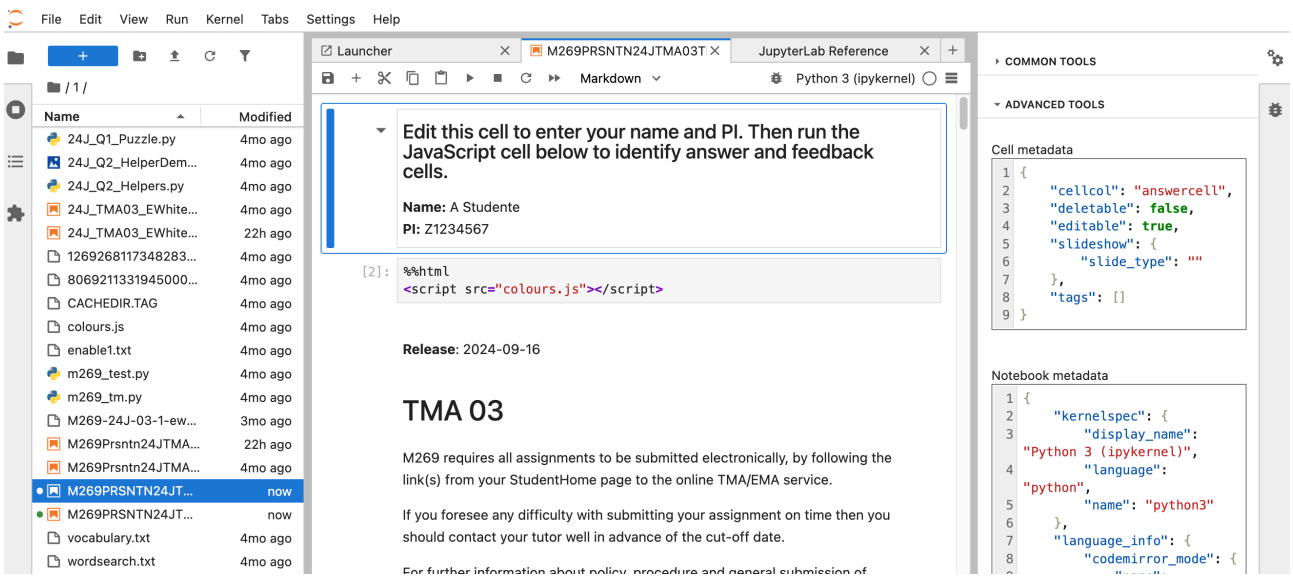
  Jupyter Lab Documentation

  Jupyter Notebook Documentation The Jupyter Notebook
- Jupyter Notebook Format
- The JSON Data Interchange Standard
- Answer the *Key Questions* for Jupyter Notebook
- What are the *names* for the parts of the interface ?
- Answer the *Key Questions* for Jupyter Notebook
- What are the *names* for the parts of the interface ?

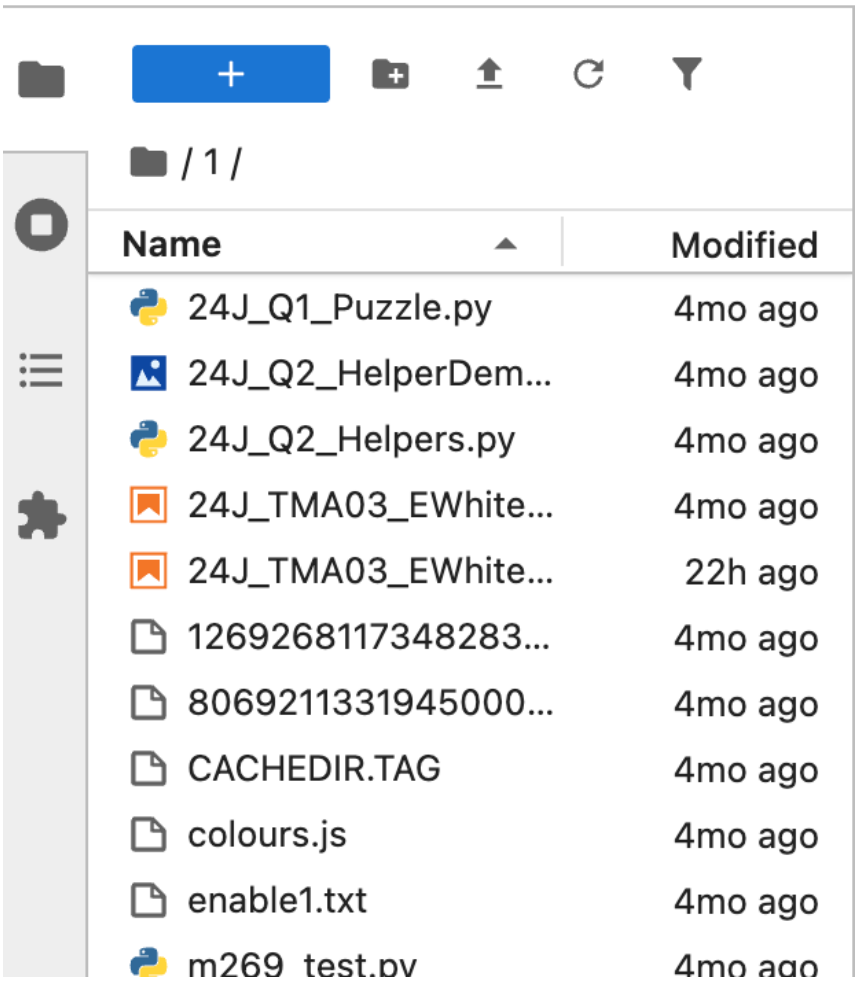
  JupyterLab Interface

  See below screen snap

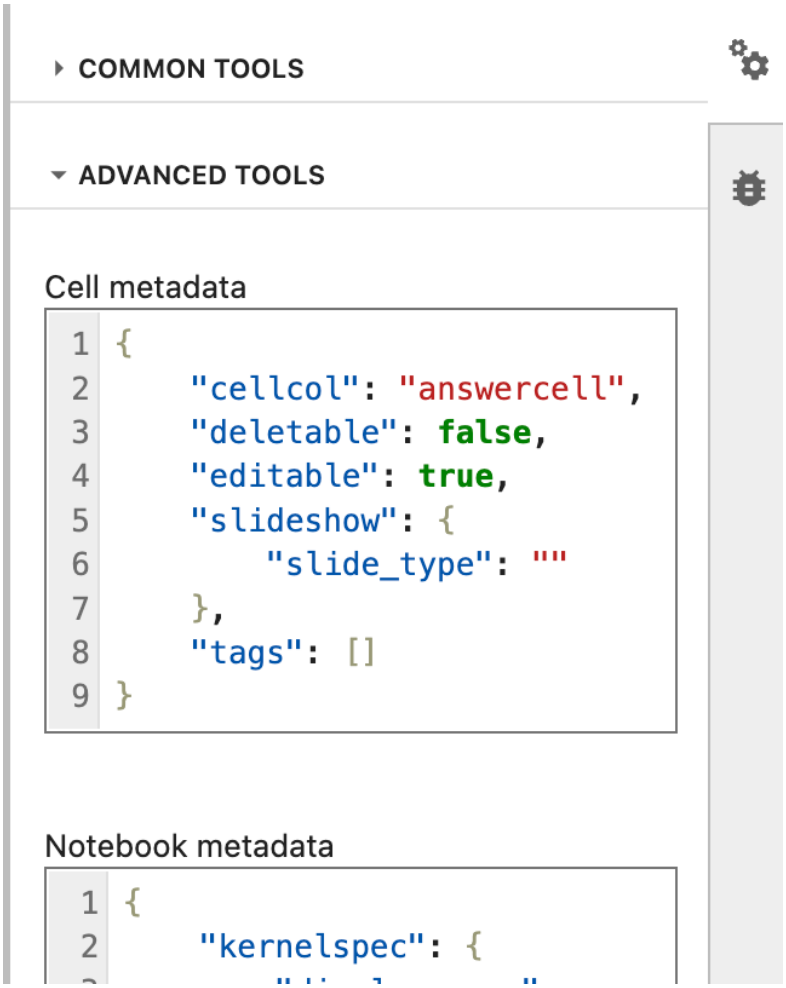
  User interface components
- *Notebook Dashboard* and *Notebook Editor*
- *Command mode* and *Edit mode*



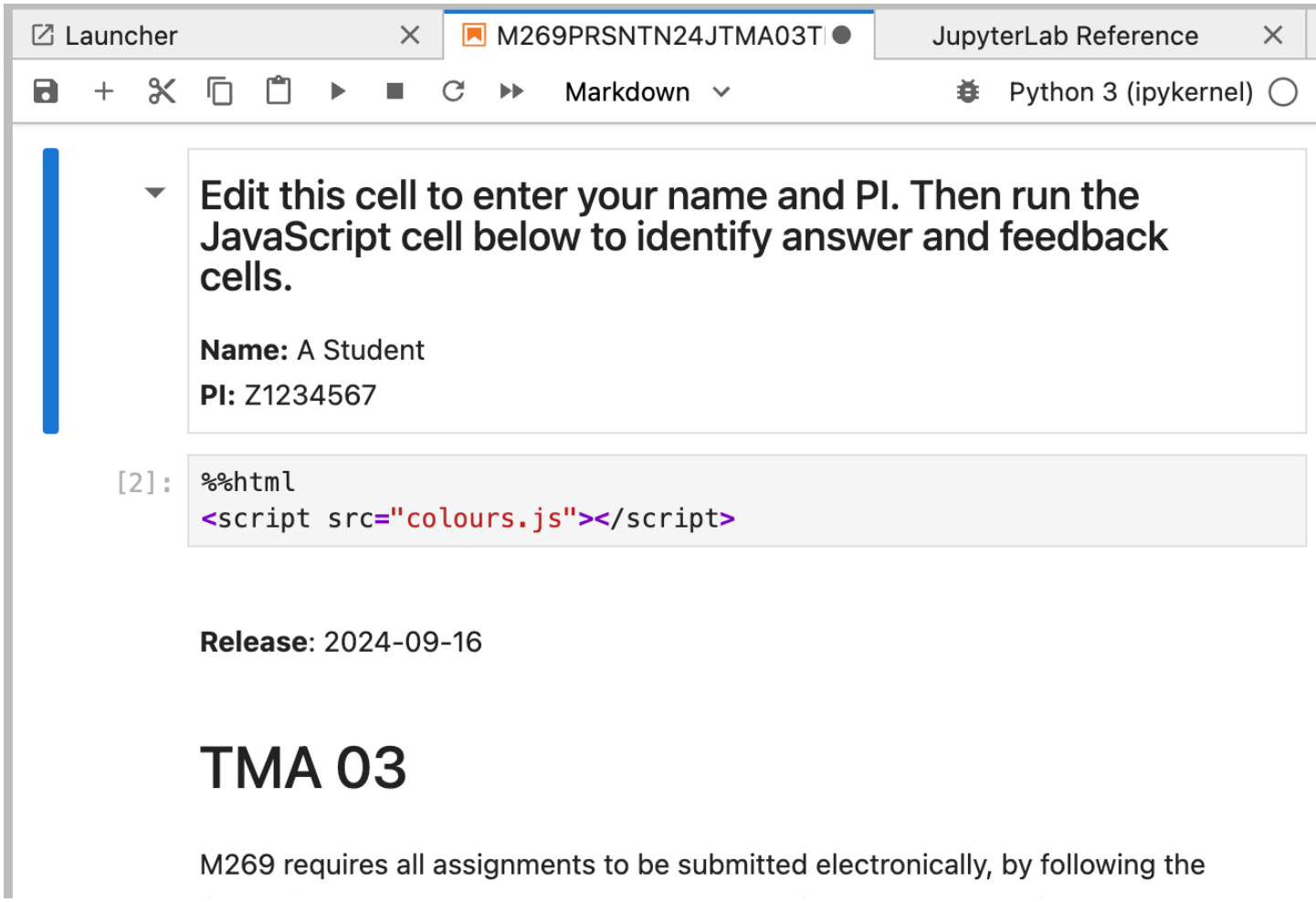
- **File**: actions related to files and folders
- **Edit**: actions related to editing documents and other activities
- **View**: actions that alter the appearance of JupyterLab
- **Run**: actions for running code in different activities such as notebooks and code consoles
- **Kernel**: actions for managing kernels, which are separate processes for running code
- **Tabs**: a list of the open documents and activities in the dock panel
- **Settings**: common settings and an advanced settings editor
- **Help**: a list of JupyterLab and kernel help links
- Left sidebar: Side tabs with various roles
- File browser
- Running terminals and kernels
- Table of Contents
- Extension manager
- View > Appearance > Show Left Sidebar or ⌘ + B



- Right sidebar: Side tabs with various roles
- Property Inspector
- Debugger
- View > Appearance > Show Right Sidebar or ⌘ + J



- Can contain documents (notebooks, files ...) and other activities (terminals, code consoles ...)



- Answer the *Key Questions* for Jupyter Lab &Notebook
- How do you leave the package ? How do you enter the package ?
- Answer the *Key Questions* for Jupyter Notebook
- How do you leave the package ? How do you enter the package ?
- **Enter**
- **Command line**

```
cd whatEverFolder
nb
```

- **GUI** see Windows but beware slow launch and having to navigate folders a lot
- **Leave**
- **Notebook** File > Close and Shut Down Notebook... ctrl + ⇧ + Q
- **Jupyter Lab** File > Shut Down
- Note File > Logout does something else and will lead to an odd request to login
- Answer the *Key Questions* for Jupyter Lab
- Is there any on-line help and, if so, how is it used ?
- Answer the *Key Questions* for Jupyter Lab
- Is there any on-line help and, if so, how is it used ?

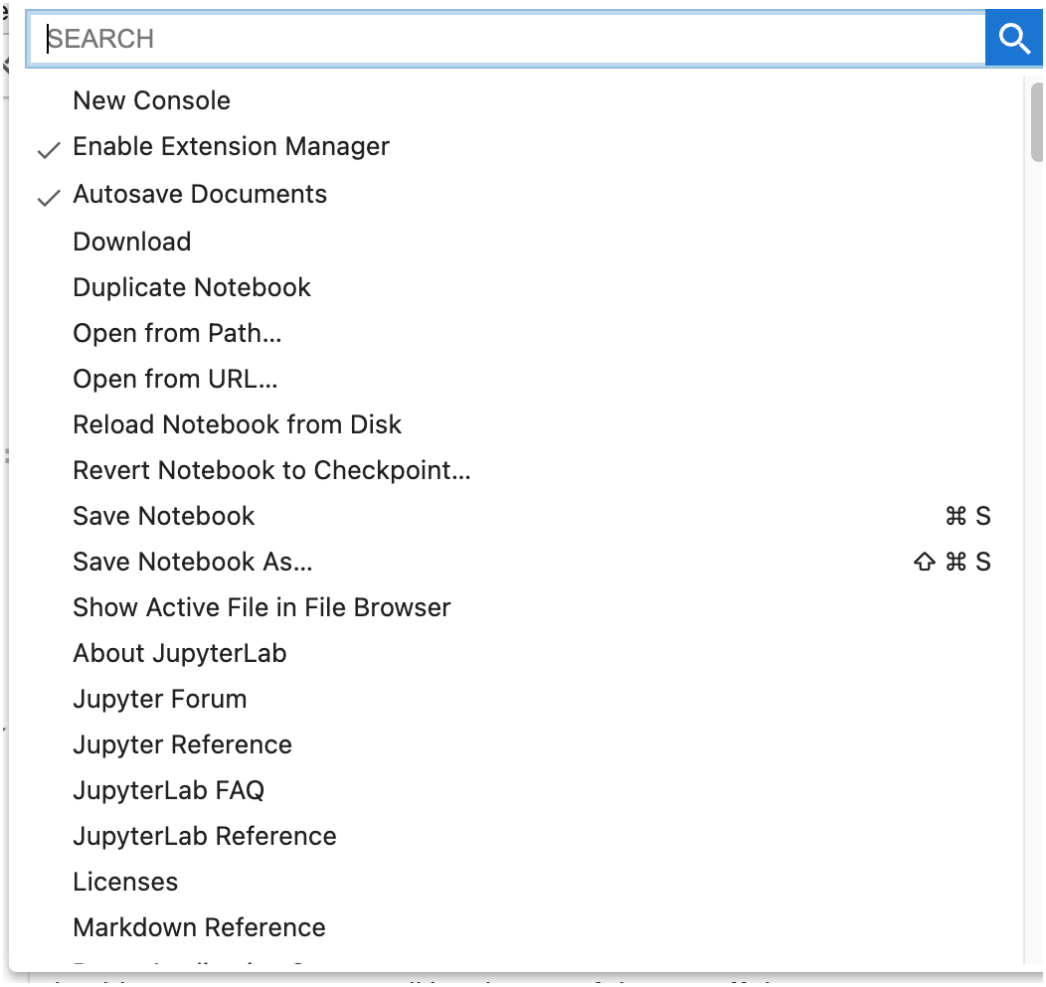
```
jupyter --help
```

but you will have to read the documentation

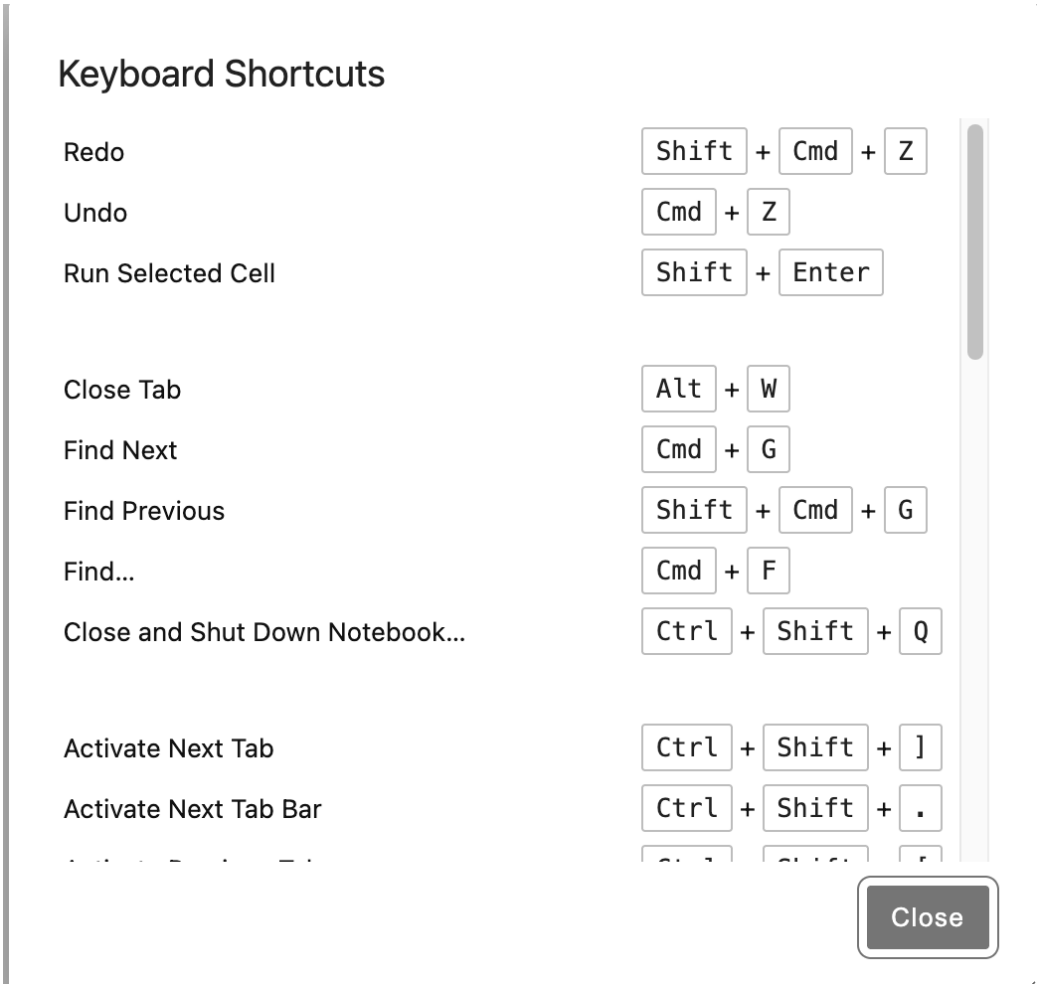
- Answer the *Key Questions* for Jupyter Lab
- Are there any initialisation files, configuration or preferences and how are they used ?
- Answer the *Key Questions* for Jupyter Lab
- Are there any initialisation files, configuration or preferences and how are they used ?
- Jupyter Lab: Advanced Usage
- See Config file and command line options
- Set in `jupyter_notebook_config.py` in `~/.jupyter` (see earlier)
- See also

```
((m269-25j) ) <M269Prsntn2024JTMAs><1010> jupyter --paths
config:
    /Users/molyneux/venvs/m269-25j/etc/jupyter
    /Users/molyneux/.jupyter
    /usr/local/etc/jupyter
    /etc/jupyter
data:
    /Users/molyneux/venvs/m269-25j/share/jupyter
    /Users/molyneux/Library/Jupyter
    /usr/local/share/jupyter
    /usr/share/jupyter
runtime:
    /Users/molyneux/Library/Jupyter/runtime
((m269-25j) ) <M269Prsntn2024JTMAs><1011>
```

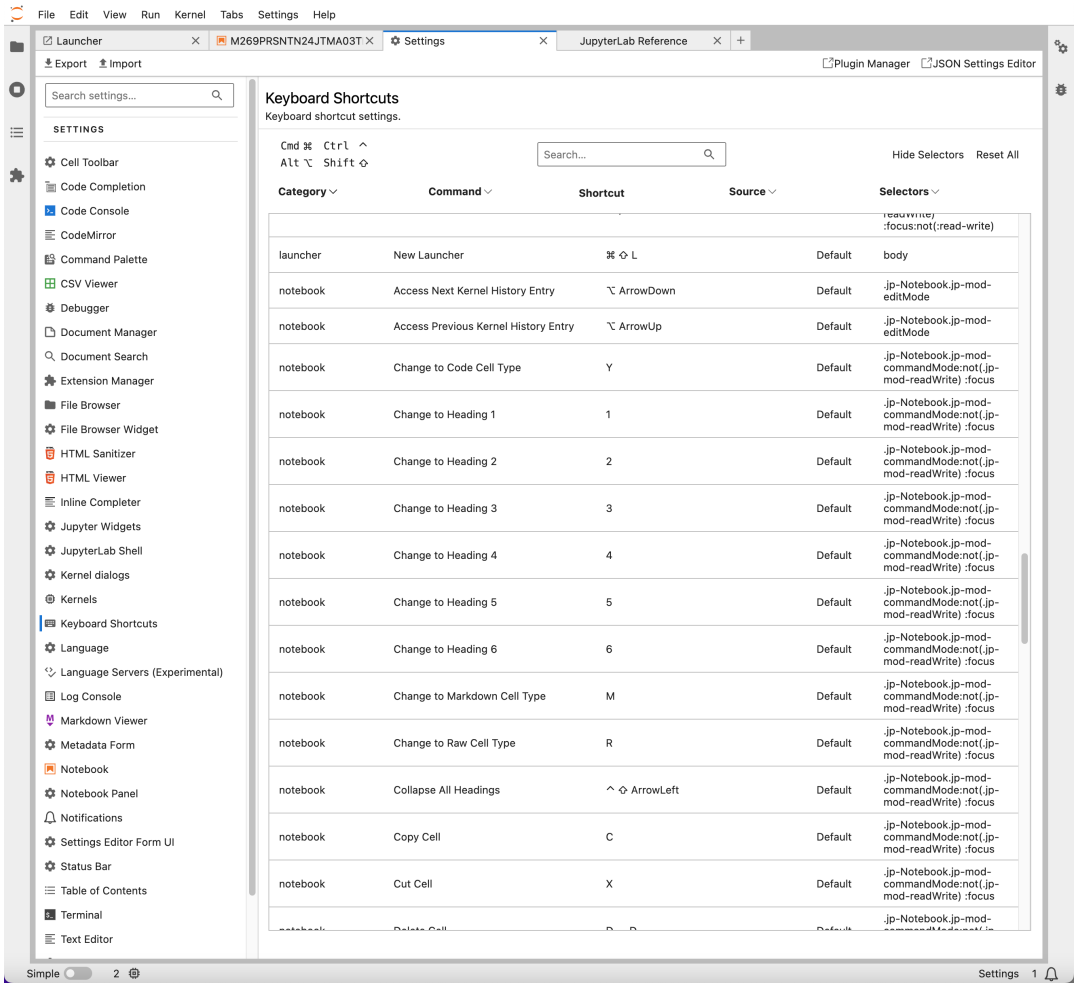
- View > Activate Command Palette or ⇧ + ⌘ + C
- Some commands have keystroke shortcuts



- Help > Show Keyboard Shortcuts or ⇧ + ⌘ + H



- Settings > Settings Editor or ⌘ + ,
- Keyboard Shortcut settings below — part — there are lots


File Edit View Run Kernel Tabs Settings Help
Launcher
M269PRSNTN24JTMA03T
Settings
JupyterLab Reference
Export
Import
Plugin Manager
JSON Settings Editor
Search settings...
SETTINGS
Cell Toolbar
Code Completion
Code Console
CodeMirror
Command Palette
CSV Viewer
Debugger
Document Manager
Document Search
Extension Manager
File Browser
File Browser Widget
HTML Sanitizer
HTML Viewer
Inline Completer
Jupyter Widgets
JupyterLab Shell
Kernel dialogs
Kernels
Keyboard Shortcuts
Language
Language Servers (Experimental)
Log Console
Markdown Viewer
Metadata Form
Notebook
Notebook Panel
Notifications
Settings Editor Form UI
Status Bar
Table of Contents
Terminal
Text Editor
Keyboard Shortcuts
Keyboard shortcut settings.
Cmd ⌘ Ctrl ^
Alt ⌥ Shift ⇧
Search...
Hide Selectors
Reset All
Category
Command
Shortcut
Source
Selectors
launcher New Launcher ⌘ ⇧ L Default body
notebook Access Next Kernel History Entry ⌥ ArrowDown Default .jp-Notebook.jp-mod-editMode
notebook Access Previous Kernel History Entry ⌥ ArrowUp Default .jp-Notebook.jp-mod-editMode
notebook Change to Code Cell Type Y Default .jp-Notebook.jp-mod-commandMode:not(.jp-mod-readWrite) :focus
notebook Change to Heading 1 1 Default .jp-Notebook.jp-mod-commandMode:not(.jp-mod-readWrite) :focus
notebook Change to Heading 2 2 Default .jp-Notebook.jp-mod-commandMode:not(.jp-mod-readWrite) :focus
notebook Change to Heading 3 3 Default .jp-Notebook.jp-mod-commandMode:not(.jp-mod-readWrite) :focus
notebook Change to Heading 4 4 Default .jp-Notebook.jp-mod-commandMode:not(.jp-mod-readWrite) :focus
notebook Change to Heading 5 5 Default .jp-Notebook.jp-mod-commandMode:not(.jp-mod-readWrite) :focus
notebook Change to Heading 6 6 Default .jp-Notebook.jp-mod-commandMode:not(.jp-mod-readWrite) :focus
notebook Change to Markdown Cell Type M Default .jp-Notebook.jp-mod-commandMode:not(.jp-mod-readWrite) :focus
notebook Change to Raw Cell Type R Default .jp-Notebook.jp-mod-commandMode:not(.jp-mod-readWrite) :focus
notebook Collapse All Headings ^ ⇧ ArrowLeft Default .jp-Notebook.jp-mod-commandMode:not(.jp-mod-readWrite) :focus
notebook Copy Cell C Default .jp-Notebook.jp-mod-commandMode:not(.jp-mod-readWrite) :focus
notebook Cut Cell X Default .jp-Notebook.jp-mod-commandMode:not(.jp-mod-readWrite) :focus
Simple
2
Settings
1

Jupyter Notebook Command Mode		macOS	
F	Find & replace	Y	Change cell to code
⏎, ⌅	Enter edit mode	M	Change cell to markdown
⌘+⇧+F	Open command palette	R	Change cell to raw
⌘+⇧+P	Open command palette	1	Change cell to heading 1
P	Open command palette	2	Change cell to heading 2
⇧+⏎, ⇧+⌅	Run cell, select below	3	Change cell to heading 3
^+⌅	Run selected cells	4	Change cell to heading 4
⌥+⏎, ⌥+⌅	Run cell, insert below	5	Change cell to heading 5
⇧+K, ⇧+↑	Extend selected cells above	6	Change cell to heading 6
⇧+J, ⇧+↓	Extend selected cells below	K, ↑	Select cell above
⌘+A	Select all cells	J, ↓	Select cell below
A	Insert cell above	B	Insert cell below
X	Cut selected cells	C	Copy selected cells
⇧+V	Paste cells above	V	Paste cells below
Z	Undo cell deletion	D , D	Delete selected cells
S, ⌘+S	Save and Checkpoint	⇧+M	Merge selected cells
L	Toggle line numbers	⇧+L	Toggle all cells line numbers
O	Toggle selected cells output	⇧+O	Toggle selected cells output scrolling
I , I	Interrupt the kernel	0 , 0	Restart the kernel (with dialog)
Esc, ⎋, Q	Close the pager	H	Show keyboard shortcuts
⇧+␣	Scroll notebook up	␣	Scroll notebook down

- Answer the *Key Questions* for Jupyter Lab
- How do you import and export data from the package ?
- Answer the *Key Questions* for Jupyter Lab
- How do you import and export data from the package ?
- **Export**
- See nbconvert — Using as a command line tool

```
jupyter nbconvert --to FORMAT myNotebook.ipynb
```

- Output formats — HTML, LaTeX, PDF, Markdown, WebPDF, Reveal.js HTML slideshow and others
- **Import**
- Embedding images — use Markdown syntax

```
    ![title][Images/myPicture.png]
```

- Convert notebook to slides

```
jupyter nbconvert --to slides --post serve myNotebook.ipynb
```

- Convert slide `myNotebook.slides.html` to PDF version — replace `#` at end of URL to `?print-pdf`

- Answer the *Key Questions* for Jupyter Lab
- *When all else fails*, how can you obtain advice ?
- Answer the *Key Questions* for Jupyter Lab
- *When all else fails*, how can you obtain advice ?

  M269 Forums

  StackOverflow: Questions tagged [jupyter]

ToC

6.2 Writing Programs & Thinking

The Steps

1. Invent a *name* for the program (or function)
2. What is the *type* of the function ? What sort of *input* does it take and what sort of *output* does it produce ? In Python a type is implicit; in other languages such as Haskell a type signature can be explicit.
3. Invent *names* for the input(s) to the function (*formal parameters*) — this can involve thinking about possible *patterns* or *data structures*
4. What restrictions are there on the input — state the preconditions.
5. What must be true of the output — state the postconditions.
6. *Think* of the definition of the function body.

The *Think* Step

- **How to Think**
  1. Think of an example or two — what should the program/function do ?
  2. Break the inputs into separate cases.
  3. Deal with simple cases.
  4. Think about the result — try your examples again.
- **Thinking Strategies**
  1. Don't think too much at one go — break the problem down. Top down design, step-wise refinement.
  2. What are the inputs — describe all the cases.
  3. Investigate choices. What data structures ? What algorithms ?
  4. Use common tools — bottom up synthesis.
  5. Spot common function application patterns — generalise & then specialise.
  6. Look for good *glue* — to combine functions together.

ToC

7 What Next ?

Programming, Debugging, Psychology

Although programming techniques have improved immensely since the early days, the process of finding and correcting errors in programming — known graphically if inelegantly, as *debugging* — still remains a most, difficult, confused and unsatisfactory operation. The chief impact of this state of affairs is psychological. Although we are all happy to pay lip-service to the adage that to err is human, most of us like to make a small private reservation about our own performance on special occasions when we really try. It is somewhat deflating to be shown publicly and incontrovertibly by a machine that even when we do try, we in fact make just as many mistakes as other people. If your pride cannot recover from this blow, you will never make a programmer.

Christopher Strachey, Systems Analysis and Programming. Scientific American, *215(3):112–124, 1966.*

- To err is human, to really foul things up requires a computer.
- Attributed to Paul R. Ehrlich in 101 Great Programming Quotes
- Attributed to Bill Vaughn in Quote Investigator
- Derived from Alexander Pope (1711, An Essay on Criticism)
- *To Err is Humane; to Forgive, Divine*
- This also contains

  A little learning is a dangerous thing;

  Drink deep, or taste not the Pierian Spring
- In programming, this means you have to *read the fabulous manual* (RTFM)

Python Data Structures and Abstract Data Types

- Basic Python — selection and iteration
- Basic data types — arrays, sequences, lists, tuples
- Example Algorithm Design
- Writing Programs & Thinking — *The Steps*
- Abstract Data Types
- Tutorial online (PM) 10:00 Sunday 26 November 2023
- TMA01 Thursday 21 December 2023

ToC

8 Web Links & References

- The *offside rule* (using layout to determine the start and end of code blocks) comes originally from Landin (1966) — see Wikipedia: Off-side rule for other programming languages that use this.

- The *step-by-step* approach to writing programs is described in Glaser et al. (2000)
- The difficulty in learning programming is described in many articles — see, for example, Dehnadi and Bornat (2006)
- UTF-8 is Unicode (or Universal Coded Character Set) Transformation Format — 8-bit — one of the character encodings for the Unicode characters or code points
- Python 3.11 Documentation
- Project Jupyter Documentation
- IPython Documentation
- Martelli et al. (2022) *Python in a Nutshell*
- Ramalho (2022) *Fluent Python*
- Lutz (2013) *Learning Python* — a sixth edition is due out in 2025 (see Web site in bibliography)
- Guttag (2021) *Introduction to Computation and Programming Using Python*
- **Exercism** exercism.org
- **Rosetta Code** rosettacode.org/wiki/Rosetta_Code
- **Rosetta Code Category: Programming Tasks** rosettacode.org/wiki/Category:Programming_Tasks

References

Böhm, Corrado and Giuseppe Jacopini (1966). Flow diagrams, Turing Machines and Languages with Only Two Formation Rules. *Communications of the ACM*, 9(5):366–371. 10

Cole, Frank N (1903). On the factoring of large numbers. *Bulletin of the American Mathematical Society*, 10(3):134–137. 12

Dehnadi, Saeed and Richard Bornat (2006). The camel has two humps. Web (Last checked 22 October 2015). URL http://www.eis.mdx.ac.uk/research/PhDArea/saeed/paper1.pdf. 35

Dijkstra, Edsger W (1968). Letters to the editor: Go To Statement Considered Harmful. *Communications of the ACM*, 11(3):147–148. 10

Glaser, H; P J Hartel; and P W Garratt (2000). Programming by numbers: a programming method for complete novices. *The Computer Journal*, 43(4):252–265. A functional approach to learning programming. 35

Guttag, John V (2013). *Introduction to Computation and Programming Using Python*. MIT Press. ISBN 0262525003. URL https://mitpress.mit.edu/9780262525008/introduction-to-computation-and-programming-using-python/.

Guttag, John V (2016). *Introduction to Computation and Programming Using Python*. MIT Press. ISBN 0262529629. URL https://mitpress.mit.edu/9780262529624/introduction-to-computation-and-programming-using-python/.

Guttag, John V (2021). *Introduction to Computation and Programming Using Python*. MIT Press, third edition. ISBN 9780262542364. URL https://mitpress.mit.edu/books/

introduction-computation-and-programming-using-python-third-edition. 35

Knuth, Donald E (1974). Structured programming with go to statements. *ACM Computing Surveys (CSUR)*, 6(4):261–301. 10

Landin, Peter J. (1966). The next 700 programming languages. *Communications of the Association for Computing Machinery*, 9:157–166. 34

Lutz, Mark (2011). *Programming Python*. O'Reilly, fourth edition. ISBN 0596158106. URL http://learning-python.com/books/about-pp4e.html.

Lutz, Mark (2013). *Learning Python*. O'Reilly, fifth edition. ISBN 1449355730. URL http://learning-python.com/books/about-lp5e.html. 35

Lutz, Mark (2025). *Learning Python*. O'Reilly Media, sixth edition. ISBN 1098171306. 17

Martelli, Alex; Anna Ravenscroft; and Steve Holden (2017). *Python in a Nutshell: A Desktop Quick Reference*. O'Reilly, third edition. ISBN 144939292X.

Martelli, Alex; Anna Martelli Ravenscroft; Steve Holden; and Paul McGuire (2022). *Python in a Nutshell: A Desktop Quick Reference*. O'Reilly, fourth edition. ISBN 1098113551. 18, 35

Meijer, Erik; Maarten Fokkinga; and Ross Paterson (1991). Functional programming with bananas, lenses, envelopes and barbed wire. In *Functional Programming Languages and Computer Architecture*, pages 124–144. Springer. 11

Miller, Bradley W. and David L. Ranum (2011). *Problem Solving with Algorithms and Data Structures Using Python*. Franklin, Beedle Associates Inc, second edition. ISBN 1590282574. URL http://interactivepython.org/courselib/static/pythonds/index.html.

Pirnat, Mike (2015). *How to Make Mistakes in Python*. O'Reilly. ISBN 978-1-491-93447-0. URL http://www.oreilly.com/programming/free/how-to-make-mistakes-in-python.csp.

Ramalho, Luciano (2022). *Fluent Python: Clear, Concise, and Effective Programming*. O'Reilly. ISBN 1492056359. 35

Rubin, Frank (1987). Goto considered harmful considered harmful. *Communications of the ACM*, 30(3):195–196. 10

Strachey, Christopher (1966). Systems Analysis and Programming. *Scientific American*, 215(3):112–124.

Tollervey, Nicholas H. (2015). *Python in Education*. O'Reilly. ISBN 978-1-491-92462-4. URL http://www.oreilly.com/programming/free/python-in-education.csp.

van Rossum, Guido and Fred Drake (2003a). *An Introduction to Python*. Network Theory Limited. ISBN 0954161769.

van Rossum, Guido and Fred Drake (2003b). *The Python Language Reference Manual*. Network Theory Limited. ISBN 0954161785.

van Rossum, Guido and Fred Drake (2011a). *An Introduction to Python*. Network Theory Limited, revised edition. ISBN 1906966133.

van Rossum, Guido and Fred Drake (2011b). *The Python Language Reference Manual*. Network Theory Limited, revised edition. ISBN 1906966141.

VanderPlas, Jake (2016). *A Whirlwind Tour of Python*. O'Reilly. ISBN 978-1-491-96465-1. URL http://www.oreilly.com/programming/free/a-whirlwind-tour-of-python.csp.

Wirth, Niklaus (1975). *Algorithms Plus Data Structures Equals Programs*. Prentice Hall. ISBN 0130224189.

ToC