

# Python

A quickstart into the very basics  
Get to know your Python environment

## Thank You

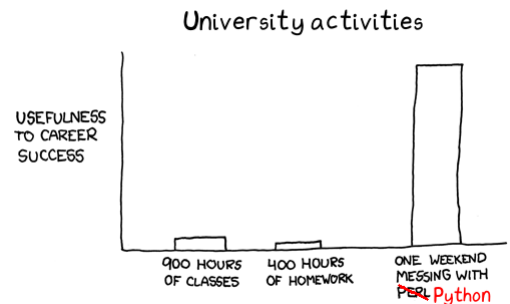
- <https://github.com/gjbex/training-material/tree/master/Python>
- *Whirlwind Tour of Python* by Jake VanderPlas (O'Reilly).  
Copyright 2016 O'Reilly Media, Inc., 978-1-491-96465-1.  
<https://www.oreilly.com/programming/free/files/a-whirlwind-tour-of-python.pdf>
- University of Virginia, Advanced Research Computing Services, Python Quickstart  
<https://arcs.virginia.edu/python-quickstart>
- <http://www.cs.cornell.edu/courses/cs1110/2018sp/>
- [https://fabienmaussion.info/scientific\\_programming/html/00-Introduction.html](https://fabienmaussion.info/scientific_programming/html/00-Introduction.html)
- <https://justinbois.github.io/bootcamp/>

## See also

- <https://www.southampton.ac.uk/~fangohr/teaching/python/book.html>
- <https://www.math.ubc.ca/~pwalls/math-python/>
- <http://troll.cs.ua.edu/ACP-PY/index.html>
- <https://data-flair.training/blogs/python-lambda-expressions/>
- <http://pages.physics.cornell.edu/~myers/teaching/ComputationalMethods/GettingStarted.html>
- <https://anh.cs.luc.edu/python/hands-on/3.1/handsonHtml/index.html>
- <https://www2.cs.duke.edu/courses/spring18/compsci101/index.php>
- <https://github.com/parrt/msan501>
- <https://docs.python-guide.org/intro/learning/>

## Tutorials

- <https://www.python.org/about/gettingstarted/>
- <https://realpython.com/>
- <https://realpython.com/matlab-vs-python/>
- <https://www.learnpython.org/>
- Cheat sheets
- <https://www.datacamp.com/community/data-science-cheatsheets>



[https://fabienmaussion.info/scientific\\_programming/img/00\\_messing\\_python.png](https://fabienmaussion.info/scientific_programming/img/00_messing_python.png)

# Why Programming?

## Why programming?

- Programming is an integral part of research.
- Programming occurs on different levels:
  - Write small scripts,
  - Write complete projects,
  - Need a good understanding of what a software package does.
- All programming languages offer to a certain extend the same building blocks.
  - Understand the basic building blocks.
  - Decompose your problem to fit those blocks.

# Programming?

- It may be (almost) impossible to solve a problem by executing commands at the command prompt.
- What is needed? A **sequence of precise instructions** that, once performed, will complete a **specific task**.
- Computer programs can't do that many things, they can:
  - Assign values to variables (memory locations).
  - Make decisions based on comparisons.
  - Repeat a sequence of instructions over and over.
  - Call subprograms.

# Programming language

- There are many programming languages, with changing popularity
- Check the Tiobe Index: <https://www.tiobe.com/tiobe-index/>
- Consider:
  - it is suited to the problem at hand?
  - is there an active community?
  - is it any good for the job market?

# Key concepts in programming

- Check Isaac Computer Science:  
[https://isaaccomputerscience.org/topics/programming\\_concepts?examBoard=all&stage=all](https://isaaccomputerscience.org/topics/programming_concepts?examBoard=all&stage=all)
- Instructions / Basic Syntax
- Data Types
  - Classification of the type of data being stored or manipulated within a program.
  - Data types are important because they determine the operations that can be performed on the data.
- Variables
  - Named container, held by the computer in a memory location.
  - Has a unique identifier (name) that refers to a value.
- Input / Output

# Key concepts in programming

- Operators
  - Arithmetic
  - Comparison
  - Logical
- Sequence:  
statements are written one after another, will be executed one statement at a time in the order that the statements are written in.
- Selection:  
execute lines of code only if a certain condition is met.
- Iteration (loop):  
repeat a group of statements .
- Subprogram (function):  
is a named sequence of statements, can be repeatedly “called” from different places in the program

## And there will be errors...

- Syntax error
  - A mistake against the language rules
  - Program will not run and will return an error message
- Runtime error
  - Usually due to some missing variables, modules,...
- Semantic error
  - A mistake in the reasoning
  - Program is not executing as intended / expected

## Python: setting the scene

get comfortable within the Python universe

# What is Python?

- From [www.python.org](http://www.python.org): "Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance."
- Python is a general purpose programming language used for a huge variety of purposes. It's user community is growing rapidly! (<https://stackoverflow.blog/2017/09/06/incredible-growth-python/>)

# What is Python?

- a **general purpose interpreted** programming language.
- a language that supports multiple approaches to software design, principally **structured** and **object-oriented** programming.
- provides **automatic memory management** and **garbage collection**
- **dynamically** typed.

*Brian Gregor (BU): A Brief Introduction to Using Python for Computational Neuroscience*

# Why Python?

- Python is quick to program in, easy exploration of modeling constructs, and fast analysis of data. (*explorative programming*)
- Python is popular in research, and has lots of libraries for science  
Widely used – extensive capabilities, documentation, and support
- Python interfaces well with faster languages
- Python is free
- Cross-platform (Windows, Mac, Linux)
- Access to advanced math, statistics, and database functions
- *Why write programs for research?*
  - Scripted research can be tested and reproduced
  - Programs are a rigorous way of describing data analysis for other researchers, as well as for computers. By sharing codes, which are much more easy for a non-author to understand than spreadsheets

<http://github-pages.ucl.ac.uk/rsd-engineeringcourse/ch00python/00pythons.html>

# Popular Python?

- Popular programming languages?
- <https://www.tiobe.com/tiobe-index/>
- What is Python used for?
- <https://www.python.org/about/apps/>

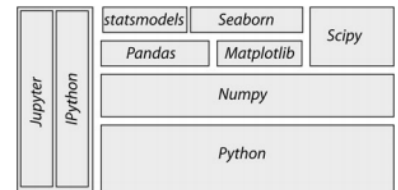


# Python ecosystem

- Large and active ecosystem

- Core Python

- Standard libraries
- third-party packages:
  - *NumPy* for manipulation of homogeneous array-based data,
  - *Pandas* for manipulation of heterogeneous and labeled data,
  - *SciPy* for common scientific computing tasks,
  - *Matplotlib* for publication-quality visualizations,
  - *IPython* for interactive execution and sharing of code, etc. *Python versions*



# Python versions

- Current 3.x
  - More clean than 2.x
  - Python 3.x introduced some backwards-incompatible changes to the language, so code written for 2.7 may not work under 3.x and vice versa.
  - Almost all Python libraries supported
- Version 2.7.x
  - Last of the 2.x releases
  - Many Python 3.x features have been retrofitted
  - All libraries support it

Note: in-application scripting may be stuck at Python 2.7!

Python 2 countdown:  
<https://pythonclock.org/>

- Taken from GJ Bex

# Installing Python

## How do I get Python?

- core Python package
  - <https://www.python.org/downloads/>
  - easy to install but probably *not* the way to go.
- Using a distribution simplifies the process of setting up your python environment, includes core Python, necessary data packages, and integrates useful tools (IDE's, notebooks, etc)  
Python Distributions:
  - **Anaconda distribution** (<https://www.anaconda.com/>)
    - Download: <https://www.anaconda.com/distribution/>
  - WinPython (<https://winpython.github.io/>)
    - Windows specific data science distribution

# Anaconda installation

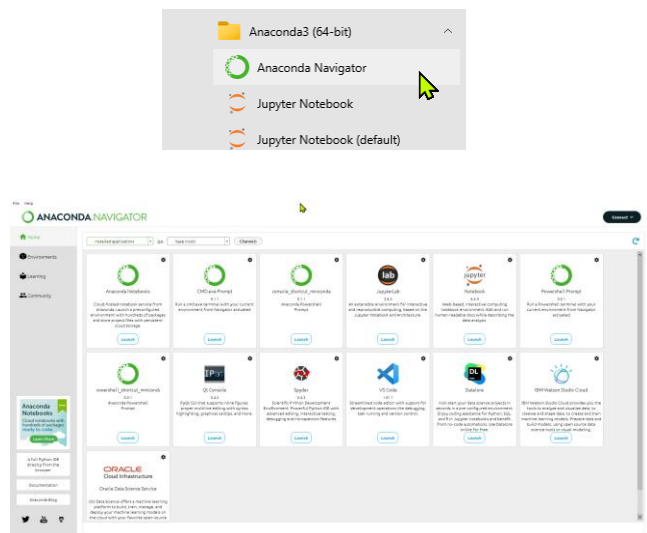
- Installation documentation:  
<https://docs.anaconda.com/anaconda/install/>
- Downloading and installing Anaconda is simple.
- Download the installer – follow the wizard



- Anaconda installation – managed PC - Windows
- Install as Admin under C:\Workdir\MyApps\Anaconda3

## Anaconda Navigator

- <https://docs.anaconda.com/navigator/index.html>
- The Navigator is a main landing page for working with your python environment.
- Launch editors (spyder, jupyter notebook, etc.) to develop python code
- Manage (install packages, etc.) the python environment





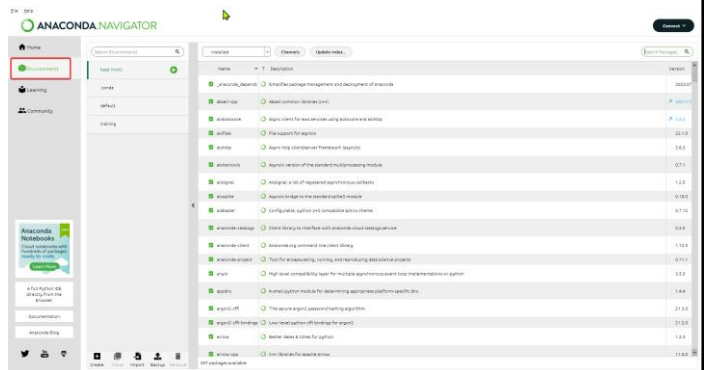
## A note on Environments

- Python virtual environments help decouple and isolate Python installs. This allows end-users to install and manage their own set of packages.
- What?
  - A Python virtual environment is a folder structure that gives you everything you need to run a lightweight yet isolated Python environment, an independent collection of software.
- Check: <https://realpython.com/python-virtual-environments-a-primer/>



# Why Environments?

- Avoid System Pollution
- Sidestep Dependency Conflicts.  
A major barrier to reproducibility in the world of computational science is package dependencies.
- Minimize Reproducibility Issues  
When sharing code, we want to be able to setup a computer ecosystem that is exactly the same as when we were developing the code.

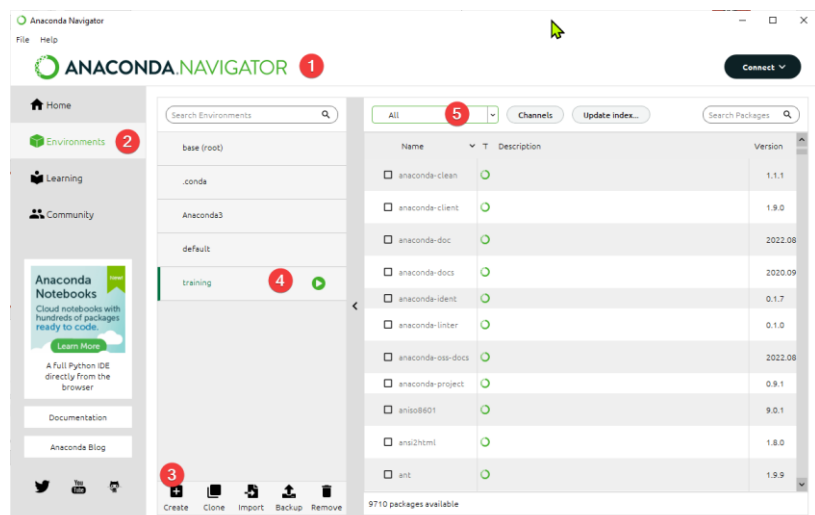


# How to create an anaconda environment



1. Open anaconda navigator
2. Click on “environment” located on the left side of the screen
3. Click on “create” located on the bottom of the screen.
4. Type in the name of the new environment—>choose python and/or R installation—>and click create.
  - Make sure the name you give the new environment has never been used before.
5. Add packages to the new environment by clicking the environment name
  - Search for the package you want in the search bar—>click on “ALL” from the drop down menu.
  - Select the name of the package you want to install, then click “apply” located in the lower right side of the screen.
  - Clicking apply installs the selected package.

# How to create an anaconda environment



# Environment: General Workflow



- **Create the environment**, either from scratch or based off an existing environment.
  - Create environment : `conda create --name myenv python=3.9`
    - where myenv is the name of the environment you would like to create
- **Activate the environment**, either using the command line or using the Anaconda Navigator graphical user interface. Anaconda starts by default in the base environment.
  - Via Command Line
    - Open the Anaconda Prompt.
    - Activate the environment using: `conda activate myenv`
      - where myenv is the name of your environment.
  - Via Anaconda Navigator
    - Open the Anaconda Navigator.
    - Switch to the desired environment by clicking the drop-down menu next to the “Applications on” text.

# Environment: General Workflow



- Do whatever you would like to do with Python.
  - Develop code and install the required packages for the code as you go, using either :
    - `conda install --name myenv <package>` or
    - `pip install <package>`
    - where <package> is the name of the python package you wish to install.
- Either **close** the programs you were working on and the Anaconda programs or **deactivate** the environment with `conda deactivate`.



## Environment: extra

- Export a current environment to a .yaml file:
  - `conda env export --name myenv > myenv.yaml`
- Create an exact copy of an existing environment:
  - `conda create --clone myenv --name myenv_bis`
- List all current existing environments:
  - `conda env list`
- List all current existing environments:
  - `conda list`
- To completely delete an environment, use the command:
  - `conda remove --name ENV_NAME --all`

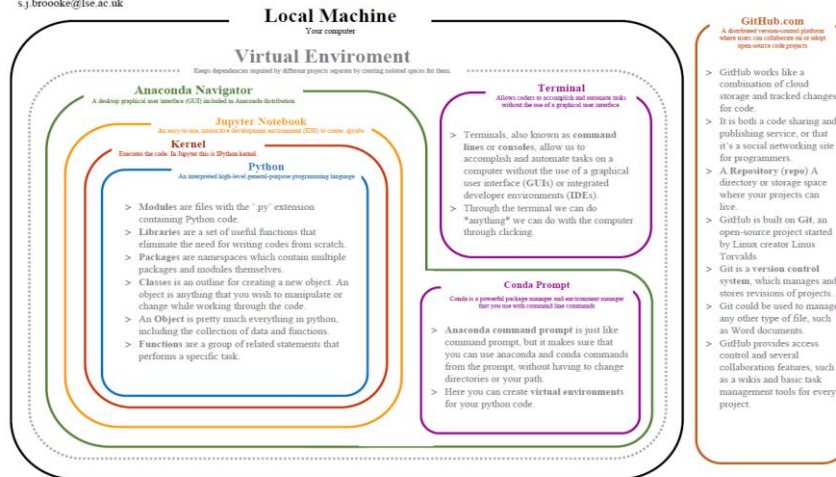
## Check this out

- <https://www.sianbrooke.co.uk/dr-brookes-blog/coding-in-python-managing-packages-with-conda-and-pip>
- <https://www.earthdatascience.org/courses/intro-to-earth-data-science/python-code-fundamentals/use-python-packages/introduction-to-python-conda-environments/>
- Quick intro into
  - Packages, modules
  - Why working with environments: keep dependencies
  - Installing with conda, pip

# How everything fits together

## Coding in Python: Understanding How Everything Fits Together

Dr Siân Brooke  
s.j.brooke@lse.ac.uk



<https://www.sianbrooke.co.uk/dr-brookes-blog/coding-in-python-understanding-how-everything-fits-together>

## Running Python Code



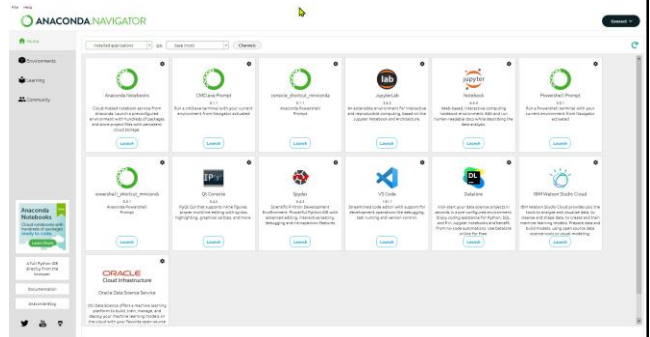
- <https://docs.anaconda.com/anaconda/user-guide/getting-started/>
- <https://realpython.com/run-python-scripts/>
- <https://plot.ly/python/ipython-vs-python/>
- <https://yihui.name/en/2018/09/notebook-war/>
- <https://www.theatlantic.com/science/archive/2018/04/the-scientific-paper-is-obsolete/556676/>
- <https://fangohr.github.io/blog/installation-of-python-spyder-numpy-sympy-scipy-pytest-matplotlib-via-anaconda.html>

## Making it work

- Write the code
  - choose a good editor (or integrated development environment - IDE)
    - featuring color coding, syntax checks, ...
  - code is just a text file
- Convert to machine code
  - make sure that you have the right interpreter (or compiler) available
- Run the code
  - run on the command line
  - run in a script mode (Python)
  - run in IDE or in Jupyter notebooks

# Where to start?

- 2 elements needed for programming in Python:
  - writing and editing Python code
  - running that code in an interpreter
- Choose a platform - primary ways to run Python code:
  1. Terminal
    1. Python interpreter
    2. IPython interpreter
    3. Running scripts
  2. IDE
    - Spyder
  3. Jupyter notebook.



# Hello World

## How to run Hello World code?

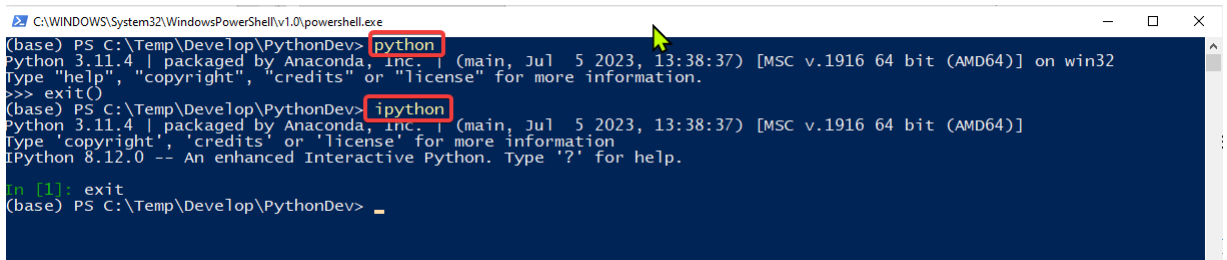
- Interactively: `print('Hello World')` in python interpreter
- `python hello_world.py`
- Run in IDE
  - `%run hello_world.py`
- Run in Jupyter notebook

# Python interpreter

<https://realpython.com/run-python-scripts/>

## Python interpreter

- The interpreter is able to run Python code in two different ways:
  - As a piece of code typed into an interactive session
  - As a script or module



```
C:\WINDOWS\System32\WindowsPowerShell\v1.0\powershell.exe
(base) PS C:\Temp\Develop\PythonDev> python
Python 3.11.4 | packaged by Anaconda, Inc. | (main, Jul 5 2023, 13:38:37) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()
(base) PS C:\Temp\Develop\PythonDev> ipython
Python 3.11.4 | packaged by Anaconda, Inc. | (main, Jul 5 2023, 13:38:37) [MSC v.1916 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.12.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: exit
(base) PS C:\Temp\Develop\PythonDev>
```

# Python interpreter

- The most basic way to execute Python code is line by line within the Python interpreter (interactive session).
- The Python interpreter can be started by typing: `python`
  - Terminal on Mac OS X and Unix/Linux systems,
  - (anaconda)Command Prompt application in Windows
  - `>>>` by default

```
Select C:\WINDOWS\System32\WindowsPowerShell\v1.0\powershell.exe
(base) PS C:\Temp\Develop\PythonDev> python
Python 3.11.4 | packaged by Anaconda, Inc. | (main, Jul 5 2023, 13:38:37) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> a = 1
>>> b = 2.3
>>> c = a / b
>>> print(c)
0.4347826086956522
>>> _
```

# IPython interpreter

- Enhanced Interactive shell
- Enhancements to the basic Python interpreter: `ipython`
- <https://stackoverflow.com/questions/12370457/what-is-the-difference-between-python-and-ipython>

```
IPython: C:\Temp\Develop\PythonDev
(base) PS C:\Temp\Develop\PythonDev> ipython
Python 3.11.4 | packaged by Anaconda, Inc. | (main, Jul 5 2023, 13:38:37) [MSC v.1916 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.12.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: a = 1
In [2]: b = 2.3
In [3]: c = a / b
In [4]: print(c)
0.4347826086956522
In [5]: _
```

# IPython interpreter

- IPython is an enhanced version of python that makes interactive python more productive.
  - Tab autocompletion (on class names, functions, methods, variables)
  - More explicit and color-highlighted error messages
  - Better history management
  - Basic UNIX shell integration (run simple shell commands such as cp, ls, rm, cp, etc. directly from the IPython command line)
  - Nice integration with many common GUI modules (PyQt, PyGTK, and tkinter)
  - <https://www.quora.com/What-is-the-difference-between-IPython-and-Python-Why-would-I-use-IPython-instead-of-just-writing-and-running-scripts>

## Magic commands



- <https://ipython.readthedocs.io/en/stable/interactive/magics.html>
- An enhancement in IPython known as magic commands
- Know more: `magic` command.  
Information of a specific magic function is obtained by `%magicfunction?`
- Designed to solve various common problems in standard data analysis. There are two types of magic commands –
- 2 types
  - Line magics
    - They are similar to command line calls. They start with % character. Rest of the line is its argument passed without parentheses or quotes.
  - Cell magics
    - They have %% character prefix. Unlike line magic functions, they can operate on multiple lines below their call.

# Magic commands



- Access a general description of available magic functions, including some examples:
  - `%magic`
- For a quick and simple list of all available magic functions:
  - `%lsmagic`

# Python scripts

- Programs: save code to file, and execute it all at once.
  - Script: A plain text file containing Python code that is intended to be directly executed by the user
  - By convention, Python scripts are saved in files with a `.py` extension.

```
python_test.py
1 print("Running test.py")
2 a = 1
3 b = 2.3
4 c = a / b
5 print(c)
```

```
Select C:\WINDOWS\System32\WindowsPowerShell\v1.0\powershell.exe
(base) PS C:\Temp\Develo\PythonDev> python .\python_test.py
Running test.py
0.4347826086956522
(base) PS C:\Temp\Develo\PythonDev>
```

# Run Python script

## Linux

- Write script in editor
- Run script using Python interpreter  
`python hello_world.py`
- Make script executable
- `chmod u+x hello_world.py`
- Run script directly  
`./hello_world.py`

## Windows

- Write script in editor
- Run script using Python interpreter  
`python hello_world.py`
- Run script directly  
`hello_world.py`

# Python scripts



## • Linux

`#!/usr/bin/env python`

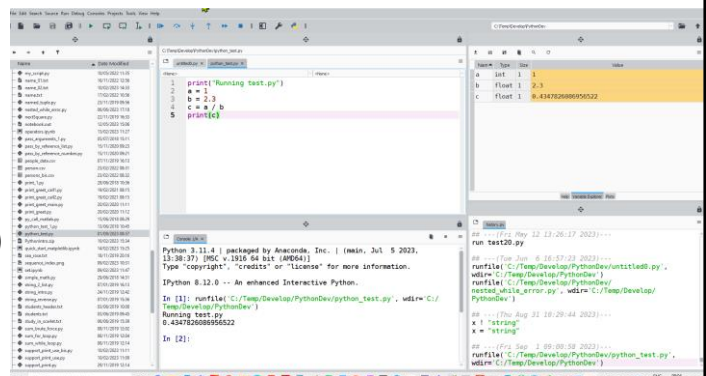
- determines the script's ability to be executed like a standalone executable without typing `python` in the terminal
- double clicking it in a file manager (when configured properly).

# Spyder

Another IDE

## IDE: Spyder

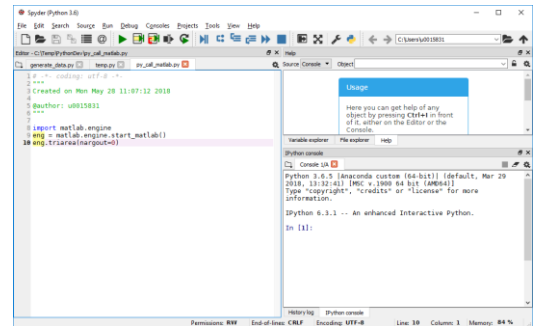
- Integrate different aspects of programming and running code.
- SPyDER: "*Scientific Python Development EnviRonment*"  
<https://www.spyder-ide.org/>
- Several tools in one integrated environment (cfr MATLAB desktop)
  - a code editor
  - IPython interpreter / console
  - variable inspector
  - control icons
- Documentation:  
<https://docs.spyder-ide.org/current/index.html>





# IDE: Spyder

- Spyder for code development.
  - Start from Anaconda Navigator
  - Command window: `spyder`
- Magic commands apply
  - Clear Console:
    - `%cls`
  - Clear all variables from Variable Explorer (reset the namespace):
    - `%reset`
  - With `automagic on`, `%` prefix not needed



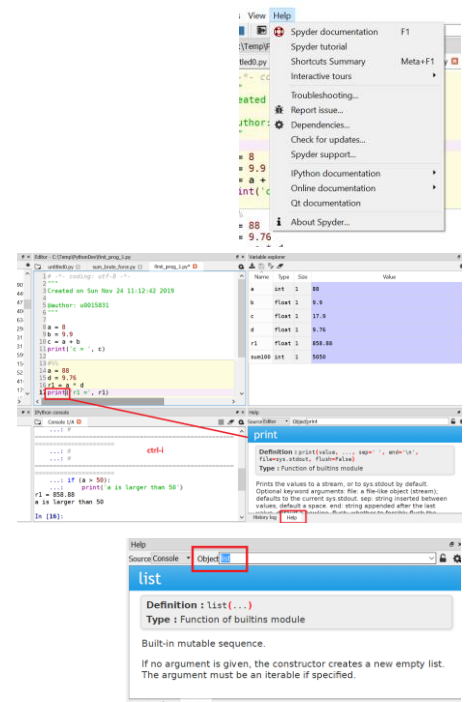
## Spyder Help

- Help on Spyder from Help menu
- Help related to Python
  - Select a command and press `ctrl-I`
    - Information opens in help window
  - Enter object in help window
- `help(command)` in console

```
In [18]: help(print)
Help on built-in function print in module builtins:

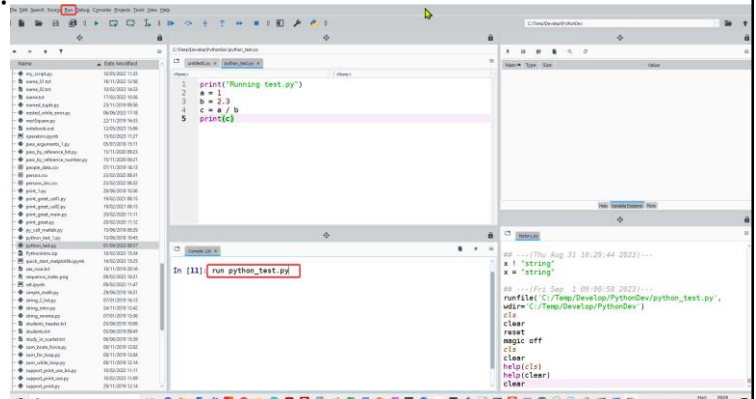
print(...)
    print(value, ..., sep=' ', end='\n',
          file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by
    default.
    Optional keyword arguments:
```



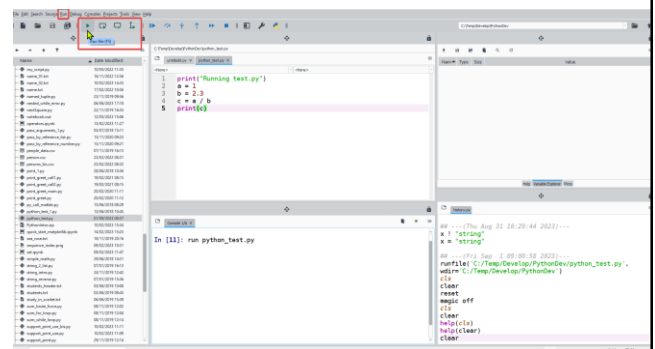
# Running scripts in Spyder console

- Run a .py file from the console
  - `run script.py`
- Tab autocompletion works!



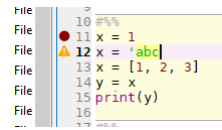
# Running scripts in Spyder

- Run scripts either using the icons or through the Run menu.
- *Run selection or current line* will run a highlighted portion of the script.
- Create **cells** by enclosing chunks of code with lines consisting of `#%%`  
*Run cell/green arrow with a box* runs the cell.
- *File: first\_prog\_1.py*



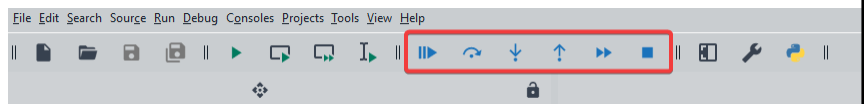
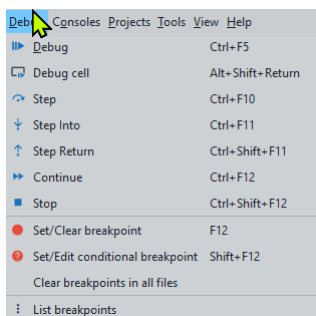
# Running scripts in Spyder

- A yellow triangle beside a line indicates a syntax error or potential problem.
- Tab completion for names familiar to it. It can show a list of members of a package for your selection, and when you have chosen a function it can show you a list of its arguments.



# Debugging in Spyder

- Debugging tool
  - The Debug menu at the top contains a list of all the options for debugging
  - The navigation bar also has the icons associated with those tasks.



# Jupyter notebook

getting\_started\_jupyter.ipynb

## Jupyter notebook

- A nice idea popularized by Mathematica is a “notebook” interface, where you can run and re-run commands
- In the notebook, you can easily mix code with comments, and mix code with the results of that code; including graphics, ...
- <https://realpython.com/jupyter-notebook-introduction/>
- <https://docs.anaconda.com/ae-notebooks/4.2.2/user-guide/basic-tasks/apps/jupyter/>
- <https://towardsdatascience.com/5-reasons-why-jupyter-notebooks-suck-4dc201e27086>

# Jupyter notebook

- Excellent for
  - Explorative programming
  - Data exploration
  - Communication, especially across domains
- Problems
  - What was (re-)executed, what not?
  - Version control?
- [https://github.com/gjbex/training-material/blob/master/Python/python\\_intro.pptx](https://github.com/gjbex/training-material/blob/master/Python/python_intro.pptx)

# Jupyter notebook vs Jupyterlab

- Basically, Jupyterlab is the new generation user interface for executing and editing notebook documents, similar to the Jupyter notebook.
- Jupyterlab is more advanced and offers more features, it gives a more IDE-like experience.
- *Beginner: start with Jupyter notebook*

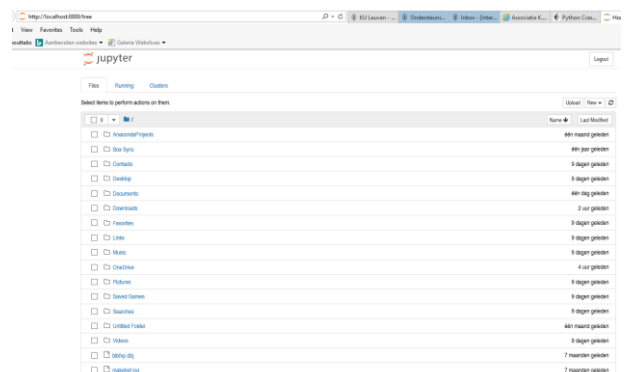
# Jupyter: how to launch?

- Anaconda Navigator:
  - Start menu
  - Launch **Notebook**
- Anaconda prompt
  - open terminal and navigate to the **directory** where you would like to save your notebook
  - `jupyter notebook`
- Start up Jupyter, 2 things will happen:
  - The server component of the Jupyter application will start up in a Windows command line window showing log messages, e.g. that the server is running locally under the address `http://localhost:8888/`.
  - The web-based client application part of Jupyter will open up in your standard web browser showing the Dashboard, the interface for managing your notebooks.



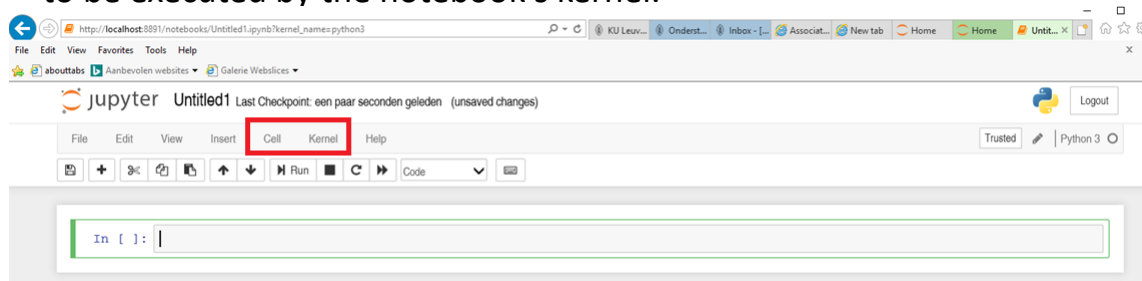
## Jupyter

- Notebook Dashboard, specifically designed for managing your Jupyter Notebooks.
- Use it as the launchpad for exploring, editing and creating your notebooks.



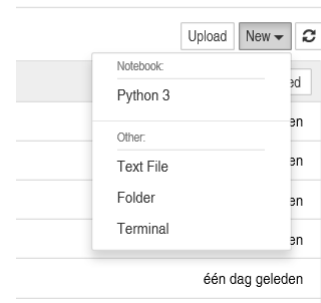
# Jupyter notebook

- Jupyter is essentially an advanced word processor.
- A kernel is a "computational engine" that executes the code contained in a notebook document.
- A cell is a container for text to be displayed in the notebook or code to be executed by the notebook's kernel.



# Jupyter notebook

- Browse to the folder in which you would like to create your first notebook,
- Click the "New" drop-down button in the top-right and
- Select "Python 3" (or the version of your choice).



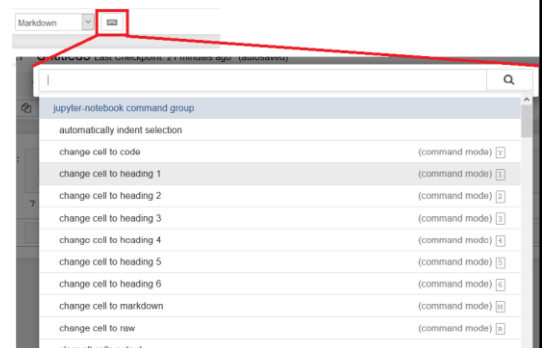
# Jupyter: basics of editing

- Jupyter notebook: sequence of cells
  - **Code**
    - Label “In [ ]” in front of the code
    - a \* will appear when executing
    - replaced by a number that always increases by one with each cell execution. This allows for keeping track of the order in which the cells in the notebook have been executed.
  - **Markdown**
- Important shortcut: ctrl+Enter (execute cell)
- Color code
  - Blue bar on the left: active cell in command mode
  - Click in cell, changes in edit mode – Green bar
- Jupyter will periodically autosave the notebook

# Jupyter: basics of editing



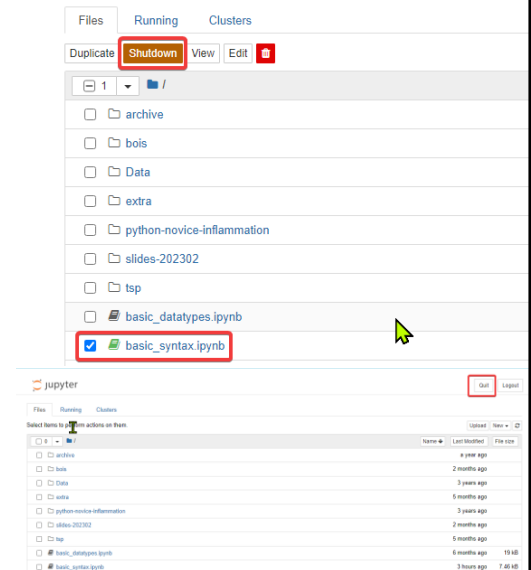
- try to know the basic shortcuts
- **Command mode** shortcuts:
  - Basic navigation: enter, **shift-enter**, up/k, down/j
  - Saving the notebook: s
  - Change Cell types: y, m, 1-6, t
    - m to change the current cell to Markdown,
    - y to change it back to code
  - Cell creation: a, b
    - a to insert a new cell above the current cell,
    - b to insert a new cell below
  - Cell editing: x, c, v, d, z
    - c copy selected cells
    - x cut selected cells
    - v paste copied cells
    - d + d (press the key twice) to delete the current cell
    - z undo cell deletion





# Close and Shutdown Jupyter Notebook

- Close Jupyter Notebook files
  - Close the browser tab displaying the notebook, but you still need Shutdown the notebook from the dashboard.
  - To Shutdown a Jupyter Notebook file (.ipynb), click in the checkbox to left of the filename. An orange button (**Shutdown**) appears in the dashboard menu; click on it to Shutdown any file that is checked in the list.
- Shutdown the Jupyter Notebook Local Server
  - After all of your notebooks are closed and shut down, you can end your Jupyter Notebook session by clicking on the **Quit** button at the top right of the dashboard.
  - Close the terminal by typing the command exit



## Jupyter: some tips



- Run a notebook on the command line with `ipython`
- Jupyter notebook tips  
<https://www.dataquest.io/blog/jupyter-notebook-tips-tricks-shortcuts/>
- <https://www.dataquest.io/blog/jupyter-notebook-tutorial/>
- <https://jupyter4edu.github.io/jupyter-edu-book/>
- <https://reproducible-science-curriculum.github.io/workshop-RR-Jupyter/>
- Change the default startup directory
  - <https://stackoverflow.com/questions/35254852/how-to-change-the-jupyter-start-up-folder>
- Change the default browser
  - <https://support.anaconda.com/customer/en/portal/articles/2925919-change-default-browser-in-jupyter-notebook>

# JupyterLab

- JupyterLab is the next-generation user interface for Project Jupyter.
- It offers all the familiar building blocks of the classic Jupyter Notebook (notebook, terminal, text editor, file browser, rich outputs, etc.) in a flexible and powerful user interface. Eventually, JupyterLab will replace the classic Jupyter Notebook.
  - Interface: JupyterLab offers a more modern, versatile interface, with support for multiple panels and tabs, enabling users to work with multiple notebooks and files simultaneously.
  - File support: JupyterLab provides built-in support for a wider range of file formats and includes integrated terminals and code consoles.
  - Collaboration: JupyterLab supports real-time collaboration, allowing multiple users to work together on the same notebook or document.
  - Extensibility: JupyterLab is designed to be extensible, enabling users to install additional extensions and customize the environment to meet their specific needs.
  - Compatibility: JupyterLab is compatible with existing Jupyter Notebook files and kernels, allowing users to transition smoothly between the two interfaces.
- <https://saturncloud.io/glossary/jupyter-notebook-vs-jupyterlab/>
- Check: <https://www.youtube.com/watch?v=yjjE-MJD5TI> (Cornell CAC JupyterLab tutorial)

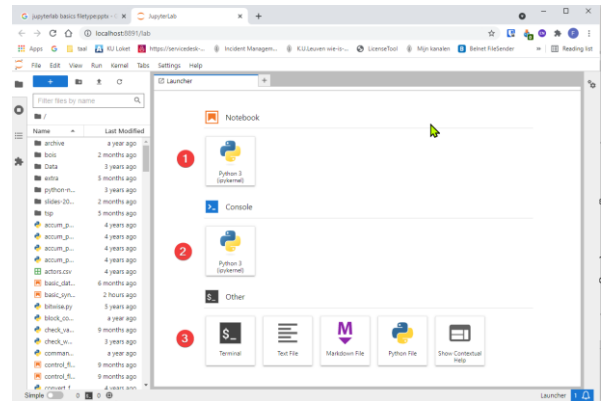
## JupyterLab: how to launch?

- Anaconda Navigator:
  - Start menu
  - Launch **JupyterLab**
- Anaconda prompt
  - open terminal and navigate to the **directory** where you would like to save your notebook
  - `jupyter lab`



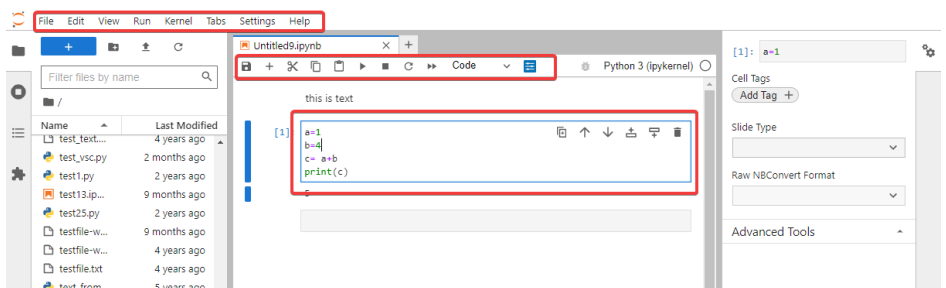
# JupyterLab interface

1. Launch your notebook
2. Launch Python kernel
3. Launch another application (i.e. terminal)



# JupyterLab interface

- Menu bar: different options that may be used to manipulate the way the notebook functions.
- Toolbar: a quick way of performing the most-used operations within the notebook.
- Cell: the notebook cell.



# JupyterLab Notebook

- The notebook consists of a sequence of cells.
  - A cell is a multiline text input field
  - The execution behaviour of a cell is determined by the cell's type.
- 3 types of cells:
  - **Code** cells allow you to edit and write new code, with full syntax highlighting and tab completion. The programming language depends on the kernel chosen.
  - **Markdown** cells allow to alternate descriptive text with code
  - **Raw** cells provide a place in which you can write output directly. Raw cells are not evaluated by the notebook.

## JupyterLab shortcuts

- The essential shortcuts:
- **Shift-Enter**: run cell and move to the next
  - Execute the current cell, show any output, and jump to the next cell below. If Shift-Enter is invoked on the last cell, it creates a new cell below. This is equivalent to clicking the Cell, Run menu item, or the Play button in the toolbar.
- **Ctrl-Enter**: run cell and stay in that cell
  - Execute the current cell, show any output.
- **Esc**: Command mode.
  - In command mode, you can navigate around the notebook using keyboard shortcuts.
- **Enter**: Edit mode.
  - In edit mode, you can edit text in cells

# Some thoughts

## Scripts vs notebooks: script

- Python script:
  - Plain text file ending with the .py extension containing the program
  - Created in editor, IDE
  - Executed from the command line
- Jupyter Notebook:
  - Stored in notebook files, having the .ipynb extension.
  - Multiple cells. Each cell can contain either a block of Python code or plain text.

## Scripts vs notebooks: script

- Pros:
  - Scripts are reliable and the most common way to write Python code.
  - Top-down execution makes it less confusing to debug and reason through the code.
  - Scripts support modularity. Variables and functions inside a Python script can be imported.
  - Can be placed in version control
  - Minimal setup is required (you only need a text editor).
  - There are many text editors and IDEs with tons of features to choose from.
- Cons:
  - Scripts are plain text files. Formatted text or figures cannot be added to them.
  - No output is saved anywhere. The script must be executed to see messages, outputs, and results.

## Scripts vs notebooks: notebook

- Pros:
  - Code blocks can be surrounded by helpful notes, figures, and links.
  - Notebooks provide nonlinear execution. Code cells can be run independently from one another.
  - Output (messages, plots, etc.) appear automatically under each cell
  - Exploratory computing, prototyping
  - Sharing results
- Cons:
  - Nonlinear execution can make debugging confusing, especially if you lose track of which cells were executed or not.
  - Version control can be a problem
  - Require installing the jupyter-notebook package
  - Notebooks must be served and accessed through a web browser, making them slightly harder to use than scripts.

## What to use?

- Compromise between Python scripts and Jupyter Notebooks
- Start out with a Notebook: explore ideas and get a clearer picture of what is needed.
- As the ideas grow clearer:
  - Put the code in a Python script
  - Put effort in using functions, modules

## Getting Help

# Getting Help

- Passing an Object to help()
  - help(print)
  - help(str.upper)
- Passing a string to help()
  - pass a string as an argument, the string will be treated as the name of a function, module, keyword, method, class, or a documentation topic and the corresponding help page will be printed.
- When needing help about a function from a certain Python library:
  - First import the library.
  - Ask to get the documentation for the function defined in the Python library.

```
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file: a file-like object (stream); defaults to the current sys.stdout.
    sep: string inserted between values, default a space.
    end: string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.

>>> help(str.upper)
Help on method_descriptor:

upper(self, /)
    Return a copy of the string converted to uppercase.

>>>
```

```
>>> help(log)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'log' is not defined
>>> import math
>>> help(log)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'log' is not defined
>>> help(math.log)
Help on built-in function log in module math:

log(...)
    log(x, [base=math.e])
    Return the logarithm of x to the given base.

    If the base not specified, returns the natural logarithm (base e) of x.
```

# Getting Help

- Python comes with a built-in help system. This means that you don't have to seek help outside of Python itself.
- help() : running the function without an argument, the interactive Python's help utility will be started
  - q to quit
  - Type the command to get the help information

```
Anaconda PowerShell Prompt

>>> help()
Welcome to Python 3.10's help utility!

If this is your first time using Python, you should definitely check out
the tutorial on the internet at https://docs.python.org/3.10/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules. To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics". Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".

help> ls
No Python documentation found for 'ls'.
Use help() to get the interactive help utility.
Use help(str) for help on the str class.

help> dir
Help on built-in function dir in module builtins:

dir(...)
    dir([object]) -> list of strings

    If called without an argument, return the names in the current scope.
    Else, return an alphabetized list of names comprising (some of) the attributes
    of the given object, and of attributes reachable from it.
    If the object supplies a method named __dir__, it will be used; otherwise
    the default dir() logic is used and returned:
    for a module object: the module's attributes;
    for a class object: its attributes, and recursively the attributes
    of its bases;
    for any other object: its attributes, its class's attributes, and
    recursively the attributes of its class's base classes.

help> print
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file: a file-like object (stream); defaults to the current sys.stdout.
    sep: string inserted between values, default a space.
    end: string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```



## Getting Help

- Python website provides in depth online documentation: <https://docs.python.org/3/index.html>
- Python website provides a comprehensive tutorial that has many examples: <https://docs.python.org/3/tutorial/index.html>