Python

A quickstart into the very basics

Get to know your Python environment

1

Thank You

- https://github.com/gjbex/training-material/tree/master/Python
- *Whirlwind Tour of Python* by Jake VanderPlas
  https://jakevdp.github.io/WhirlwindTourOfPython/
- https://www.cs.cornell.edu/courses/cs1110/2023fa/materials/python/
- https://fabienmaussion.info/scientific_programming/welcome.html
- https://justinbois.github.io/bootcamp/

2

See also

- https://fangohr.github.io/teaching/python/book.html
- https://patrickwalls.github.io/mathematicalpython/
- https://sites.duke.edu/compsci_101l_001_sp24/
- https://github.com/parrt/msan501
- https://docs.python-guide.org/intro/learning/

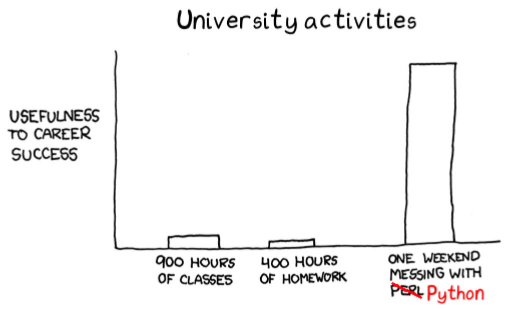
3

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

4



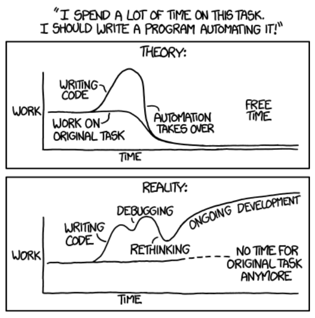
https://fabienmaussion.info/scientific_programming/img/00_messing_python.png

Why Programming?

5

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.



xkcd 1319: Automation

6

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.
- Code can be shared
- Code is much more easy to understand for a non-author than spreadsheets

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

7

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.

8

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?

9

Key concepts in programming

- Check Isaac Computer Science: 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

10

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

11

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

12

Python: setting the scene

get comfortable within the Python universe

13

What is Python?

- From 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/)

16

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

18

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
  - extensive capabilities, documentation, and support
  - Access to advanced math, statistics, and database functions
- Python interfaces well with faster languages
- Python is free
- Cross-platform (Windows, Mac, Linux)

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

19

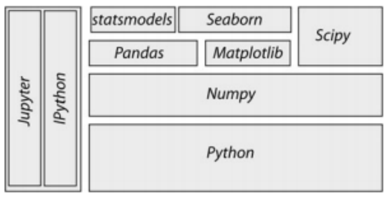
Popular Python?

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

21

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*



22

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*

23

Installing Python

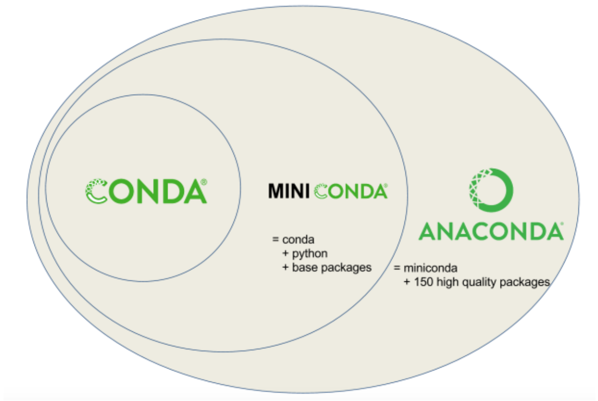
26

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:
  - Conda-forge distribution (https://conda-forge.org/)
    - a minimalistic installer for the conda package manager
  - Anaconda distribution (https://www.anaconda.com/) LICENSE ISSUES!
  - WinPython (https://winpython.github.io/)
    - Windows specific data science distribution

27

Anaconda installation



- Source: https://carpentries-incubator.github.io/introduction-to-conda-for-data-scientists/01-getting-started-with-conda/index.html

30

Note: dependencies

- Python uses external libraries or packages for being able to do almost anything.
- Many packages do not just do everything on their own, they depend on other packages for their functionality.
  - Scipy package is used for numerical routines. The package makes use of other packages, such as numpy (numerical python) and matplotlib (plotting): *dependencies* of Scipy
- Many packages are being developed over time, generating different versions: a function call can change and/or functionalities are added or removed. If one package depends on another, this may create *issues*.

32

Note: Environments

- Idea: solve dependency issues by installing packages in isolated environments.
- Python virtual environments
  - Help decouple and isolate Python installations. This allows end-users to install and manage their own set of packages.
  - 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/

33

Environment: general workflow (CLI)

- Conda is an open source system for managing Python environments.
- A Python environment is a version of Python and some associated Python packages.
- With conda:
  - install the packages you need.
  - let conda do the work of pulling in other packages that a package you want depends upon.
  - maintain different environments for different needs.
  - export an environment file that can be used to recreate the environment on a different system.

40

Environment: general workflow (CLI)

1. Install a Package Manager
   If you haven't already, install a package manager like `conda` (via Miniforge or Anaconda) or `virtualenv`.
2. Create a New Environment:
   `conda create --name myenv`
3. Activate the Environment:
   `conda activate myenv`
4. Install Packages:
   `conda install numpy pandas`
5. Work on Your Project:
   With the environment activated, you can run your Python scripts and work on your project. All installed packages will be isolated to this environment.
6. Deactivate the Environment:
   When done, deactivate the environment:
   `conda deactivate`

- Using environments helps keep your projects organized and ensures that dependencies don't interfere with each other.

41

Environment: extra (CLI)

- Export a current environment to a .yml file:
  - `conda env export --name myenv> myenv.yml`
- 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 the packages and their versions installed in your current conda environment.
  - `conda list`
- To completely delete an enviroment, use the command:
  - `conda remove --name ENV_NAME --all`

42

Running Python Code

47

Some background

- https://docs.anaconda.com/anaconda/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

48

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

49

Where to start?

- 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 / Jupyterlab

50

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

51

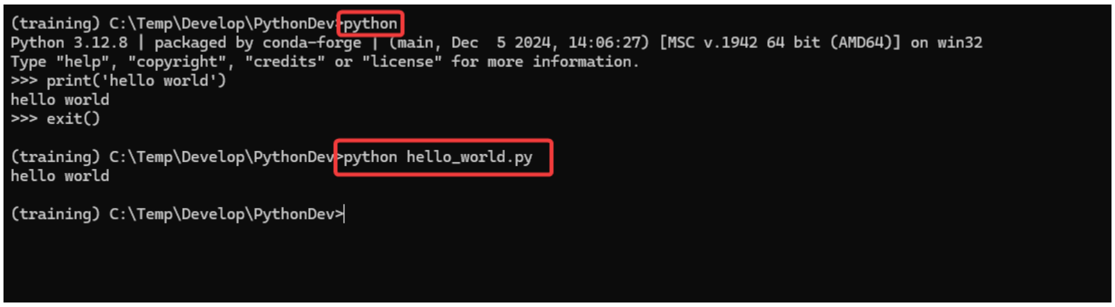
Python interpreter

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

52

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

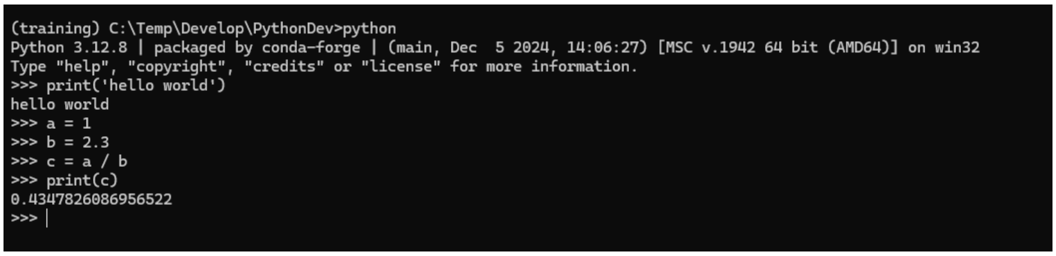


```
(training) C:\Temp\Develop\PythonDev>python
Python 3.12.8 | packaged by conda-forge | (main, Dec  5 2024, 14:06:27) [MSC v.1942 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print('hello world')
hello world
>>> exit()

(training) C:\Temp\Develop\PythonDev>python hello_world.py
hello world

(training) C:\Temp\Develop\PythonDev>
```

53

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,
  - Command Prompt application in Windows
  - >>> by default

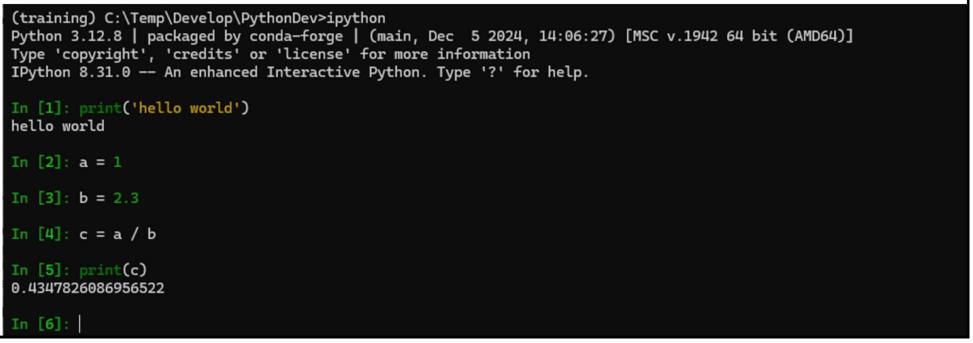


```
(training) C:\Temp\Develop\PythonDev>python
Python 3.12.8 | packaged by conda-forge | (main, Dec  5 2024, 14:06:27) [MSC v.1942 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print('hello world')
hello world
>>> a = 1
>>> b = 2.3
>>> c = a / b
>>> print(c)
0.4347826086956522
>>> |
```

54

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



```
(training) C:\Temp\Develop\PythonDev>ipython
Python 3.12.8 | packaged by conda-forge | (main, Dec  5 2024, 14:06:27) [MSC v.1942 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.31.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: print('hello world')
hello world

In [2]: a = 1

In [3]: b = 2.3

In [4]: c = a / b

In [5]: print(c)
0.4347826086956522

In [6]: |
```

55

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

56

Magic commands

- https://ipython.readthedocs.io/en/stable/interactive/magics.html
- An enhancement in IPython known as magic commands
- Get more information: `%magic`
- Information of a specific magic function is obtained by `%magicfunction?`
- Quick list of all available magic functions: `%lsmagic`

57

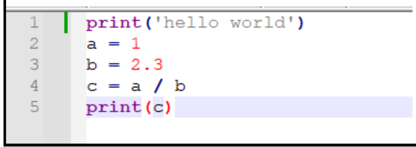
Magic commands

- Designed to solve various common problems in standard data analysis.
- 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.

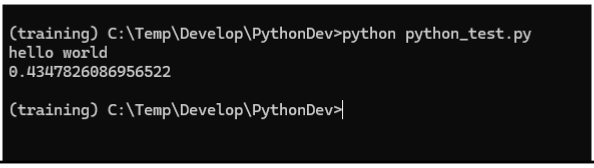
58

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.



```
print('hello world')
a = 1
b = 2.3
c = a / b
print(c)
```



```
(training) C:\Temp\Develop\PythonDev>python python_test.py
hello world
0.43478260869565522

(training) C:\Temp\Develop\PythonDev>
```

59

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`

60

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).

61

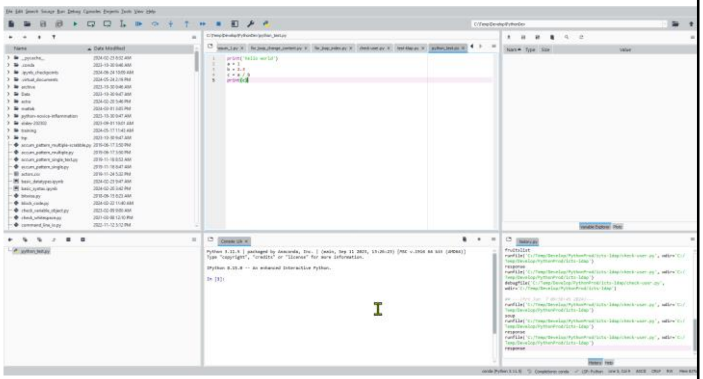
Spyder

Another IDE

63

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



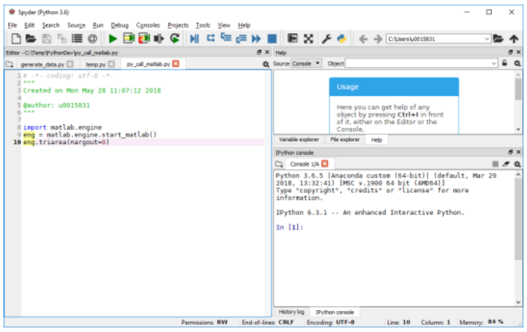
```
print('hello world')
a = 1
b = 2.0
c = a / b
print(c)
```

```
Python 3.11.5 | packaged by Anaconda, Inc. | (main, Sep 11 2023, 13:26:23) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 8.15.0 -- An enhanced Interactive Python.

In [3]:
```

64

IDE: Spyder

- Spyder for code development.
  - 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 (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\u0015831
Editor - C:\Temp\PythonDev\py_call_matlab.py
generate_data.py temp.py py_call_matlab.py
Help
Source Console Object
Usage
Here you can get help of any object by pressing Ctrl+I in front of it, either on the Editor or the Console.
Variable explorer File explorer Help
IPython console
Console 1/A
History log IPython console
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 10 Column: 1 Memory: 84 %

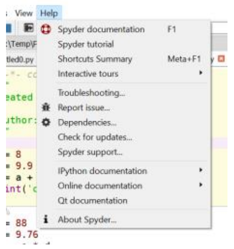
```
# -*- coding: utf-8 -*-
"""
Created on Mon May 28 11:07:12 2018

@author: u0015831
"""

import matlab.engine
eng = matlab.engine.start_matlab()
eng.triarea(nargout=0)
```

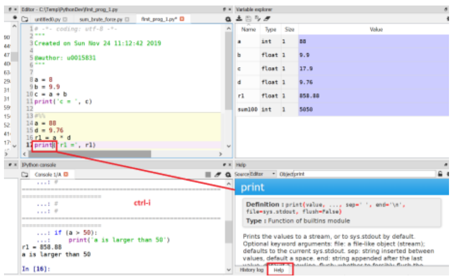
```
Python 3.6.5 |Anaconda custom (64-bit)| (default, Mar 29
2018, 13:32:41) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more
information.

IPython 6.3.1 -- An enhanced Interactive Python.

In [1]:
```

65

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




Editor - C:\Temp\PythonDev\first_prog_1.py
untitled0.py sum_brute_force.py first_prog_1.py*
Variable explorer
IPython console
Console 1/A
ctrl-i
Help
Source Editor Object print
print
History log Help

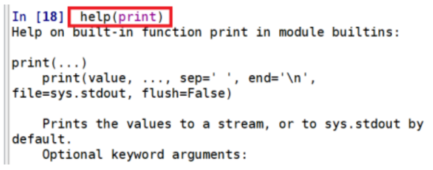
```
# -*- coding: utf-8 -*-
"""
Created on Sun Nov 24 11:12:42 2019

@author: u0015831
"""

a = 8
b = 9.9
c = a + b
print('c = ', c)

#%%
a = 88
d = 9.76
r1 = a * d
print('r1 =', r1)
```

Name	Type	Size	Value
a	int	1	88
b	float	1	9.9
c	float	1	17.9
d	float	1	9.76
r1	float	1	858.88
sum100	int	1	5050

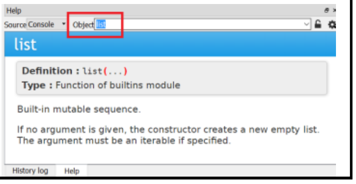
```
    ...: if (a > 50):
    ...:     print('a is larger than 50')
r1 = 858.88
a is larger than 50

In [16]:
```

Definition : `print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)`
Type : Function of builtins module

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



```
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:
```



Definition : `list(...)`
Type : Function of builtins module

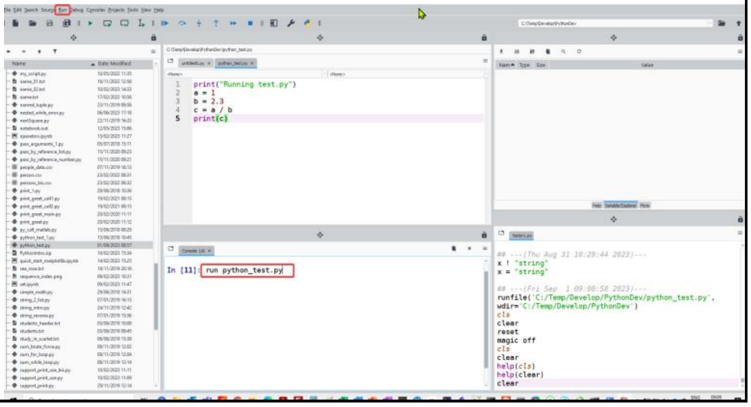
Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list.
The argument must be an iterable if specified.

66

Running scripts in Spyder console

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



```
print("Running test.py")
a = 1
b = 2.3
c = a / b
print(c)
```

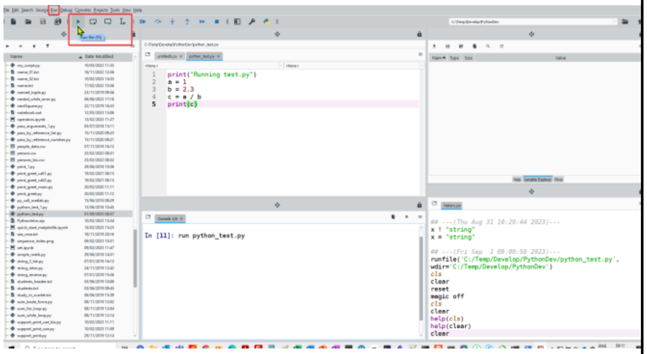
```
In [11]: run python_test.py
```

```
## ---(Thu Aug 31 10:29:44 2023)---
x ! "string"
x = "string"

## ---(Fri Sep  1 09:00:58 2023)---
runfile('C:/Temp/Develop/PythonDev/python_test.py',
wdir='C:/Temp/Develop/PythonDev')
cls
clear
reset
magic off
cls
clear
help(cls)
help(clear)
clear
```

67

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*



```
print("Running test.py")
a = 1
b = 2.3
c = a / b
print(c)
```

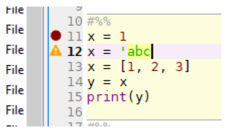
```
In [11]: run python_test.py
```

```
## ---(Thu Aug 31 10:29:44 2023)---
x ! "string"
x = "string"

## ---(Fri Sep  1 09:00:58 2023)---
runfile('C:/Temp/Develop/PythonDev/python_test.py',
wdir='C:/Temp/Develop/PythonDev')
cls
clear
reset
magic off
cls
clear
help(cls)
help(clear)
clear
```

68

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.

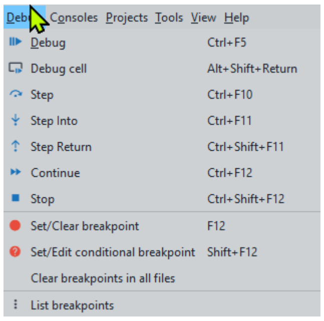


```
#%%
x = 1
x = 'abc
x = [1, 2, 3]
y = x
print(y)
```

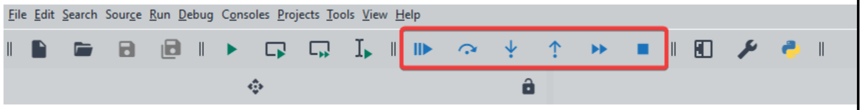
69

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.



Debug Consoles Projects Tools View Help
Debug Ctrl+F5
Debug cell Alt+Shift+Return
Step Ctrl+F10
Step Into Ctrl+F11
Step Return Ctrl+Shift+F11
Continue Ctrl+F12
Stop Ctrl+Shift+F12
Set/Clear breakpoint F12
Set/Edit conditional breakpoint Shift+F12
Clear breakpoints in all files
List breakpoints



File Edit Search Source Run Debug Consoles Projects Tools View Help

70

Jupyterlab / Jupyter notebook

getting_started_jupyter.ipynb

71

(Jupyter) notebook

- A nice idea popularized by Mathematica is a "notebook" interface, where you can run and re-run commands
- 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

72

(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

73

Jupyter notebook vs Jupyterlab

- JupyterLab and Jupyter Notebook serve as interactive computing environments, they differ in their user interface, functionality, and flexibility.
- Jupyter Notebook has a simpler, more lightweight interface.
- JupyterLab offers a more versatile and feature-rich interface , it gives a more IDE-like experience.

74

JupyterLab

- Interactive Development Environment for working with notebooks, code and data.
- 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 inteface.
  Eventually, JupyterLab will replace the classic Jupyter Notebook.
  - File support: JupyterLab provides built-in support for a wider range of file formats and includes integrated terminals and code consoles.
  - 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://realpython.com/using-jupyterlab/
- https://saturncloud.io/glossary/jupyter-notebook-vs-jupyterlab/
- https://www.youtube.com/watch?v=yjjE-MJD5TI (Cornell CAC JupyterLab tutorial)

75

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
  - Note: JupyterLab treats the directory from which it is launched as the top-level directory
  - `jupyter lab`
  - Options possible `jupyter lab --browser=firefox`

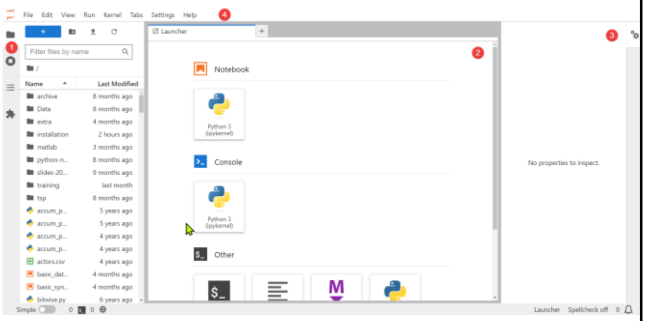
76

Jupyterlab interface

https://jupyterlab.readthedocs.io/en/stable/user/interface.html

Interface can be tweaked

1. Left panel:
    1. File browser
        1. New launcher
        2. New folder
        3. Upload files
    2. Running terminals and kernels
2. Center panel: main work area containing tabs of documents and activities
3. Right panel: shows various properties

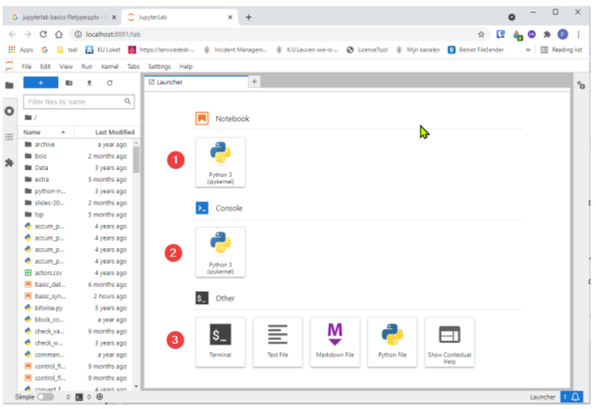


Name	Last Modified
archive	8 months ago
Data	8 months ago
extra	4 months ago
installation	2 hours ago
matlab	3 months ago
python-n...	8 months ago
slides-20...	9 months ago
training	last month
tsp	8 months ago
accum_p...	5 years ago
accum_p...	5 years ago
accum_p...	4 years ago
accum_p...	4 years ago
actors.csv	4 years ago
basic_dat...	4 months ago
basic_syn...	4 months ago
bitwise.py	6 years ago

77

JupyterLab interface

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

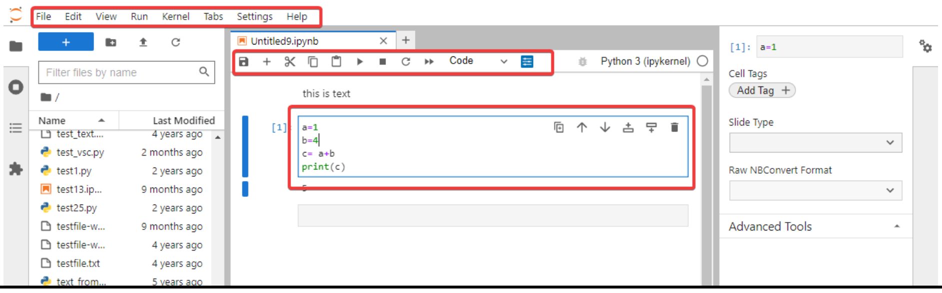


Name	Last Modified
archive	a year ago
bois	2 months ago
Data	3 years ago
extra	5 months ago
python-n...	3 years ago
slides-20...	2 months ago
tsp	5 months ago
accum_p...	4 years ago
accum_p...	4 years ago
accum_p...	4 years ago
accum_p...	4 years ago
actors.csv	4 years ago
basic_dat...	6 months ago
basic_syn...	2 hours ago
bitwise.py	5 years ago
block_co...	a year ago
check_va...	9 months ago
check_w...	3 years ago
comman...	a year ago
control_fl...	9 months ago
control_fl...	9 months ago
convert_f...	4 years ago

78

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.



Name	Last Modified
test_text....	4 years ago
test_vsc.py	2 months ago
test1.py	2 years ago
test13.ip...	9 months ago
test25.py	2 years ago
testfile-w...	9 months ago
testfile-w...	4 years ago
testfile.txt	4 years ago
text_from...	5 years ago

this is text

```
[1]: a=1
b=4
c= a+b
print(c)
```

79

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.

80

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

81

Jupyterlab shutdown

- Make sure everything is saved
- Use File → Shut Down to close the application
- Close the browser (tab)

82

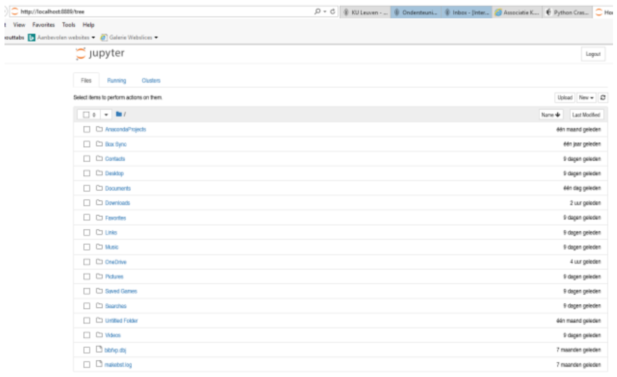
Jupyter: how to launch?

- 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.

83

Jupyter

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



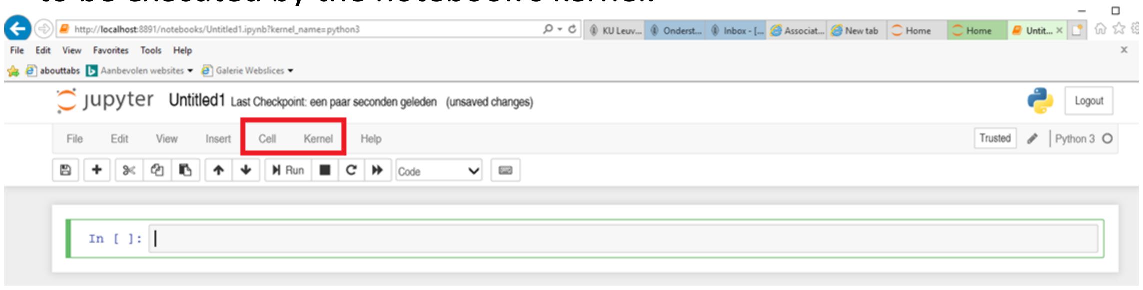
Select items to perform actions on them.

/	
AnacondaProjects	één maand geleden
Box Sync	één jaar geleden
Contacts	9 dagen geleden
Desktop	9 dagen geleden
Documents	één dag geleden
Downloads	2 uur geleden
Favorites	9 dagen geleden
Links	9 dagen geleden
Music	9 dagen geleden
OneDrive	4 uur geleden
Pictures	9 dagen geleden
Saved Games	9 dagen geleden
Searches	9 dagen geleden
Untitled Folder	één maand geleden
Videos	9 dagen geleden
bibfvp.dbj	7 maanden geleden
makebst.log	7 maanden geleden

84

Jupyter notebook

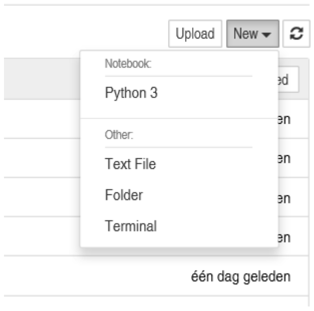
- Jupyter is essentially an advanced word processor.
- A kernel is a "computational engine" that executes the code contained in a notebook document.
- A cell is a container for text to be displayed in the notebook or code to be executed by the notebook's kernel.


http://localhost:8891/notebooks/Untitled1.ipynb?kernel_name=python3
jupyter Untitled1 Last Checkpoint: een paar seconden geleden (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Help
Trusted
Python 3
Run
Code
In [ ]:

85

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).



86

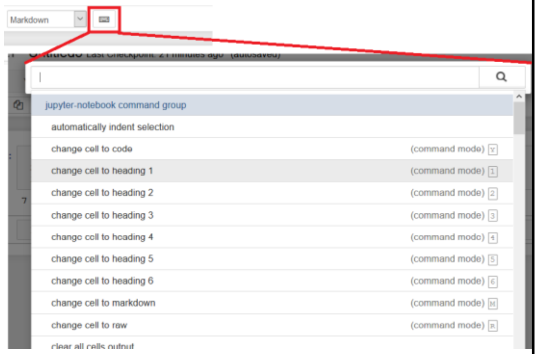
Jupyter: basics of editing

- 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

87

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

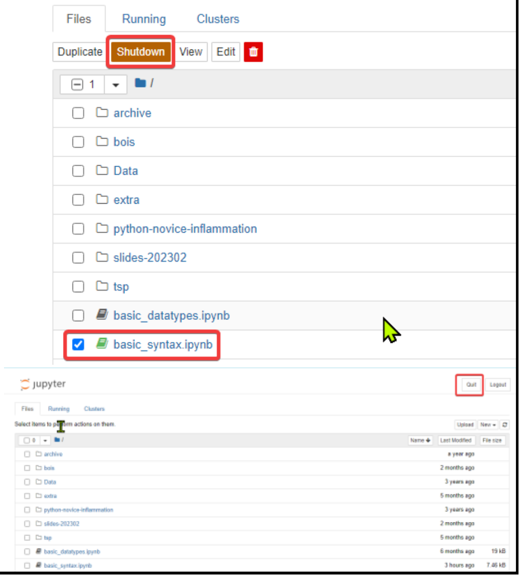


jupyter-notebook command group
automatically indent selection
change cell to code (command mode) Y
change cell to heading 1 (command mode) 1
change cell to heading 2 (command mode) 2
change cell to heading 3 (command mode) 3
change cell to heading 4 (command mode) 4
change cell to heading 5 (command mode) 5
change cell to heading 6 (command mode) 6
change cell to markdown (command mode) M
change cell to raw (command mode) R
clear all cells output

88

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



90

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

91

Some thoughts

94

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.

95

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.

96

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.

97

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

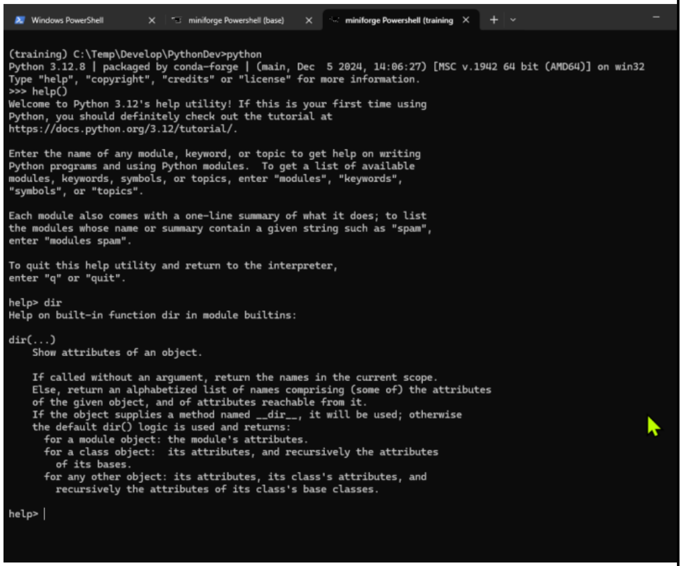
98

Getting Help

99

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

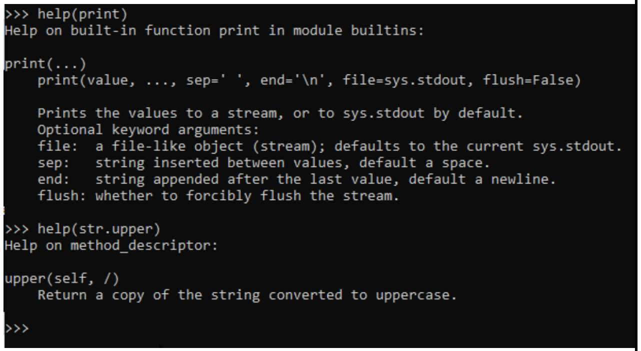


```
(training) C:\Temp\Develop\PythonDev>python
Python 3.12.8 | packaged by conda-forge | (main, Dec  5 2024, 14:06:27) [MSC v.1942 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> help()
Welcome to Python 3.12's help utility! If this is your first time using
Python, you should definitely check out the tutorial at
https://docs.python.org/3.12/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To get a list of available
modules, keywords, symbols, or topics, enter "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",
enter "modules spam".

To quit this help utility and return to the interpreter,
enter "q" or "quit".

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

dir(...)
    Show attributes of an object.

    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 returns:
      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> |
```

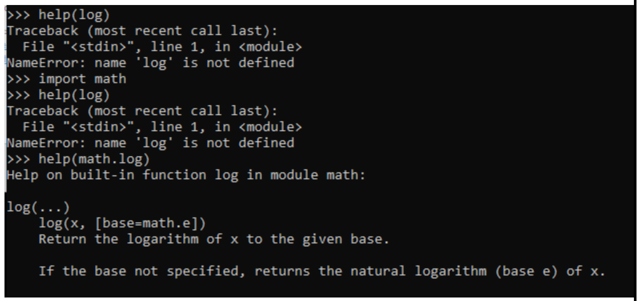
100

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.

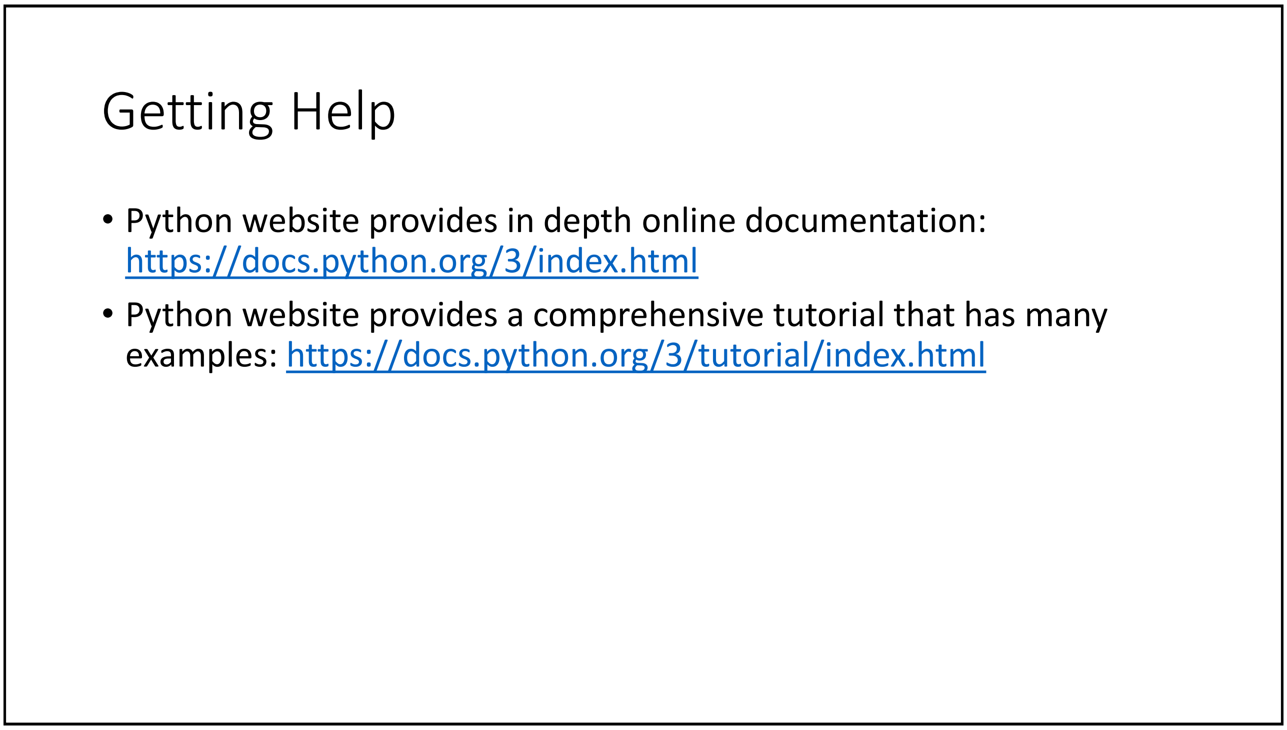


```
>>> help(print)
Help on built-in function print in module builtins:

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.
```

101



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

102