بسم الله الرحمن الرحيم

High Performance Multiprocessor Systems

Lecture 3: Python

part1

Outline

- Compiled Language vs. Interpreted Language
- CONDA
- miniCONDA vs. ANACONDA
- Conda vs. Pip
- Integrated Development Environment (IDE)
- Text and source code editor
- Getting started with Python for science

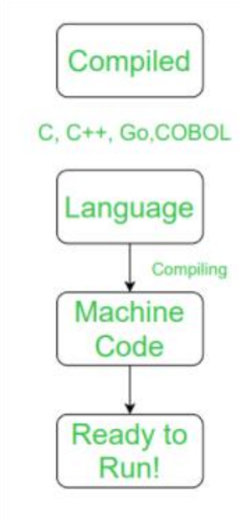
Compiled Language vs. Interpreted Language

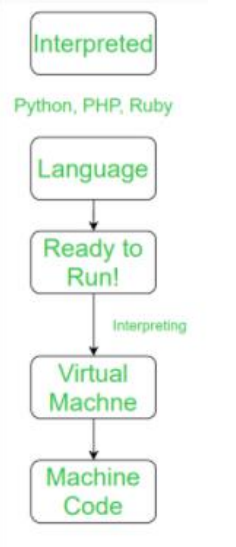
- **Compiled Language:**

Compiled languages are converted directly into machine code that the processor can execute. As a result, they tend to be faster and more efficient to execute than interpreted languages. They also give the developer more control over hardware aspects, like memory management and CPU usage. Types of compiled language – C, C++, C#, etc.

- **Interpreted Language:**

An interpreted language is a programming language which are generally interpreted, without compiling a program into machine instructions. Interpreters run through a program line by line and execute each command. It is one where the instructions are not directly executed by the target machine, but instead read and executed by some other program. Interpreted language ranges – JavaScript, Perl, Python, BASIC, etc.





Compiled Language
vs.
Interpreted Language

What about Python?

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python’s elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

- **Advantages**:
  - Very rich scientific computing libraries. (a bit less than Matlab, though)
  - Well thought out language, allowing to write very readable and well structured code: we “code what we think”.
  - Many libraries for other tasks than scientific computing (web server management, serial port access, etc.)
  - Free and open-source software, widely spread, with a vibrant community.
- **Drawbacks**:
  - Less pleasant development environment than, for example Matlab.
  - Not all the algorithms that can be found in more specialized software or toolboxes.

CONDA

- **Conda** is an open source **package management system** and **environment management system** that runs on Windows, macOS and Linux.
- **Conda** quickly installs, runs and updates packages and their dependencies.
- **Conda** easily creates, saves, loads and switches between environments on your local computer.
- It was created for Python programs, but it can package and distribute software for any language.

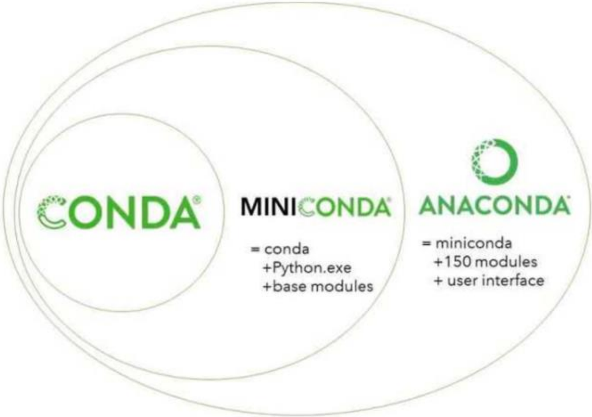
miniCONDA vs. ANACONDA

miniCONDA

ANACONDA

- **There are essentially two main differences:**
  1. **Number of packages: Anaconda** comes with over 150 data science packages, whereas **miniconda** comes with only a handful.
  2. **Interface: Anaconda** has a graphical user interface (GUI) called the Navigator, while **miniconda** has a command-line interface.

- In other words, **miniconda** is a mini version of **Anaconda**. **Miniconda** ships with just the repository management system and a few packages. Whereas, with **Anaconda**, you have the distribution of some 150 built-in packages.

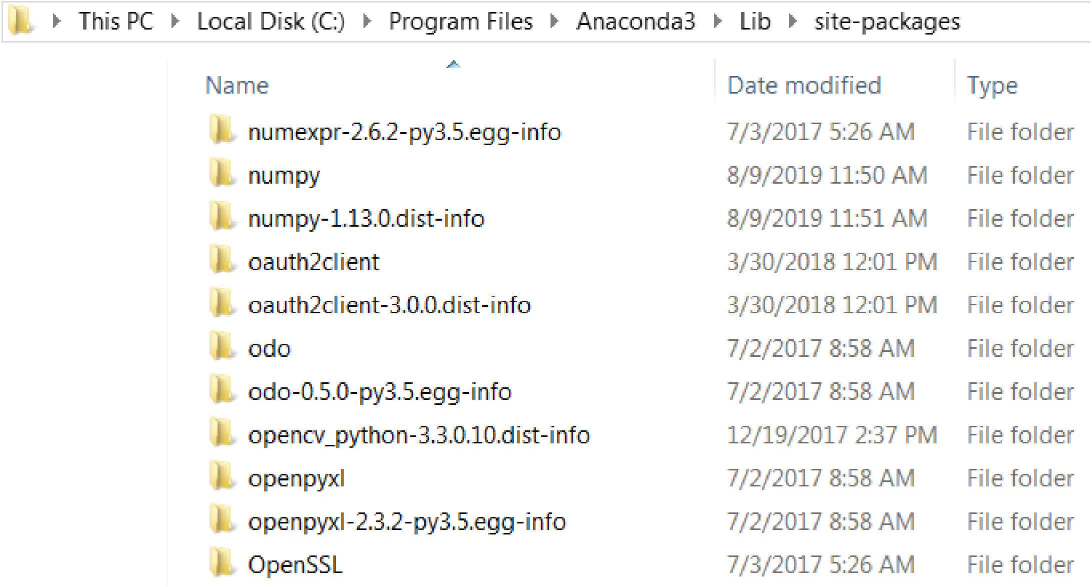


CONDA

MINICONDA
= conda
+Python.exe
+base modules

ANACONDA
= miniconda
+150 modules
+ user interface

ANACONDA



This PC ▸ Local Disk (C:) ▸ Program Files ▸ Anaconda3 ▸ Lib ▸ site-packages

Name	Date modified	Type
numexpr-2.6.2-py3.5.egg-info	7/3/2017 5:26 AM	File folder
numpy	8/9/2019 11:50 AM	File folder
numpy-1.13.0.dist-info	8/9/2019 11:51 AM	File folder
oauth2client	3/30/2018 12:01 PM	File folder
oauth2client-3.0.0.dist-info	3/30/2018 12:01 PM	File folder
odo	7/2/2017 8:58 AM	File folder
odo-0.5.0-py3.5.egg-info	7/2/2017 8:58 AM	File folder
opencv_python-3.3.0.10.dist-info	12/19/2017 2:37 PM	File folder
openpyxl	7/2/2017 8:58 AM	File folder
openpyxl-2.3.2-py3.5.egg-info	7/2/2017 8:58 AM	File folder
OpenSSL	7/3/2017 5:26 AM	File folder

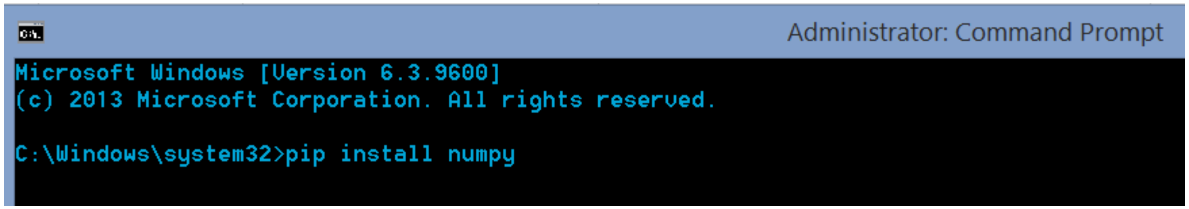
https://docs.python.org/3/tutorial

Conda vs. Pip

- **Conda** and **pip** are often considered as being nearly identical. Although some of the functionality of these two tools overlap, they were designed and should be used for different purposes.

1. **Pip** is the Python Packaging Authority's recommended tool for installing packages from the **Python Package Index, PyPI**. Pip installs Python software packaged as **wheels** or **source distributions**. The latter may require that the system have compatible compilers, and possibly libraries, installed before invoking pip to succeed.
   - **https://pypi.org/**

```
pip install numpy
```



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Windows\system32>pip install numpy
```

Conda vs. Pip

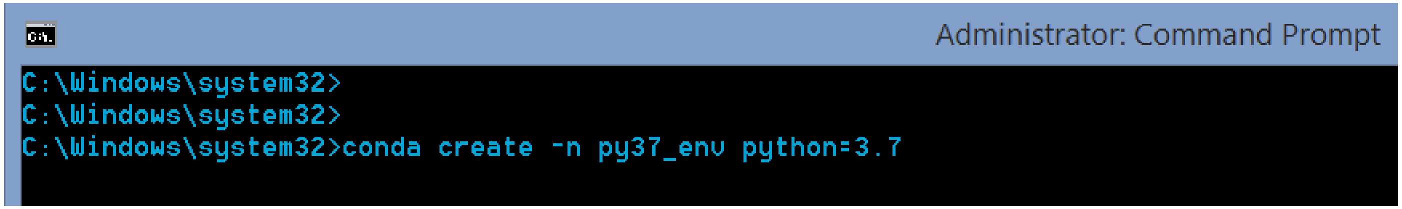
2. **Conda** is a cross platform package and environment manager that installs and manages conda packages from the Anaconda repository as well as from the Anaconda Cloud. Conda packages are binaries. There is never a need to have compilers available to install them. Additionally conda packages are not limited to Python software. They may also contain C or C++ libraries, R packages or any other software.
   - **https://repo.anaconda.com/**

ANACONDA®

Installers & Packages

Creation of virtual environments

Conda create –n py37 python=3.7



```
C:\Windows\system32>
C:\Windows\system32>
C:\Windows\system32>conda create -n py37_env python=3.7
```

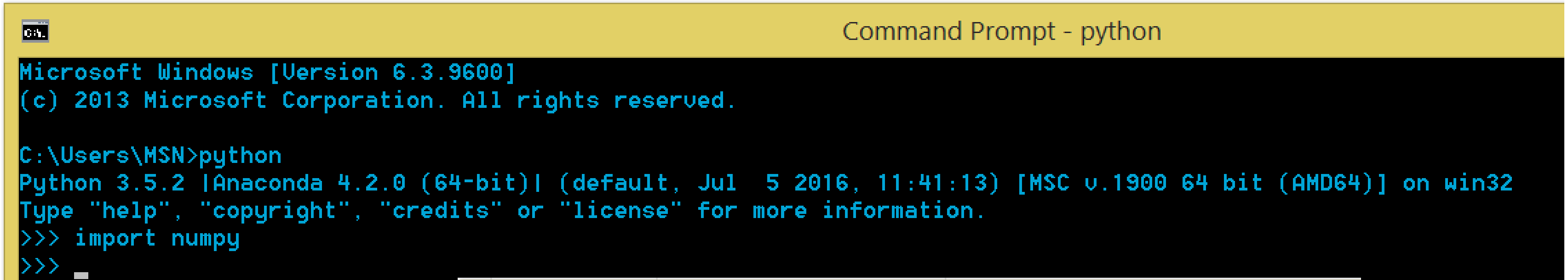
Conda vs. Pip

- One of the key difference between the two tools is that conda has the ability to create isolated environments that can contain different versions of Python and/or the packages installed in them.
- This can be extremely useful when working with data science tools as different tools may contain conflicting requirements which could prevent them all being installed into a single environment.
- Pip has no built in support for environments but rather depends on other tools like virtualenv or venv to create isolated environments. Tools such as pipenv, poetry, and hatch wrap pip and virtualenv to provide a unified method for working with these environments.

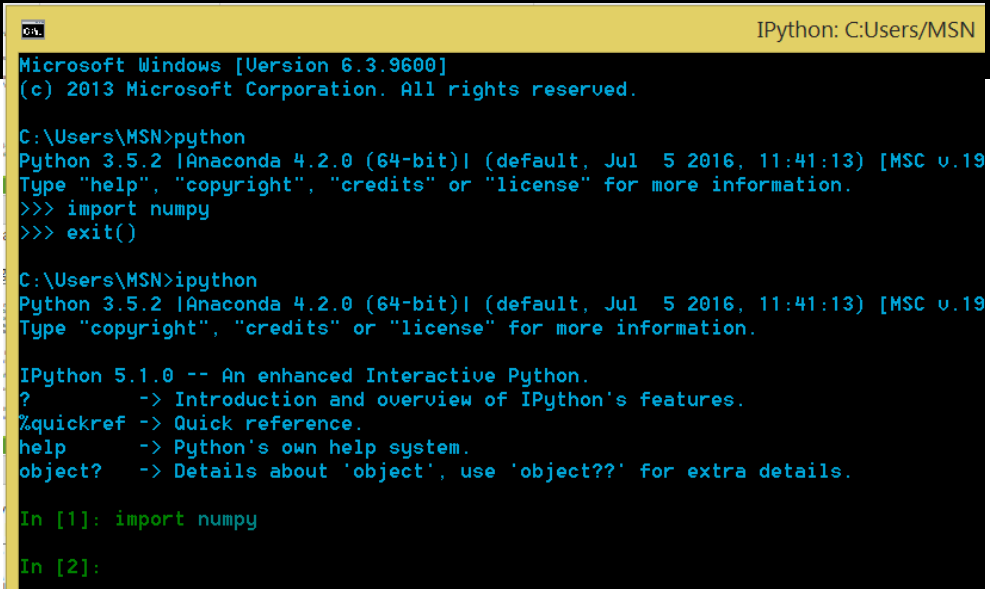
Integrated Development Environment (IDE)

- You need an IDE or a code editor to showcase your coding skills and talent.
- An IDE is a software that consists of common developer tools into a single user-friendly GUI (Graphical User interface)

Integrated Development Environment (IDE)



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\MSN>python
Python 3.5.2 |Anaconda 4.2.0 (64-bit)| (default, Jul  5 2016, 11:41:13) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
>>> 
```



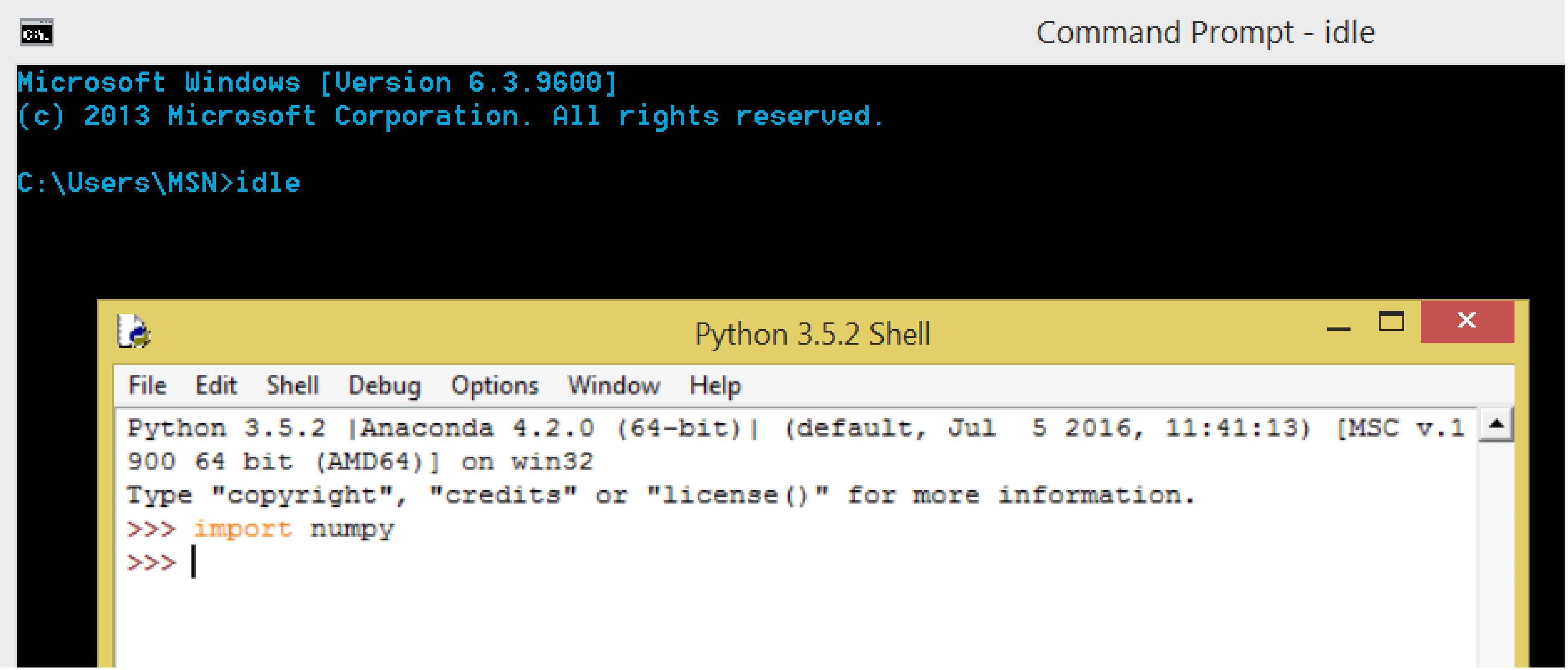
```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\MSN>python
Python 3.5.2 |Anaconda 4.2.0 (64-bit)| (default, Jul  5 2016, 11:41:13) [MSC v.19
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
>>> exit()

C:\Users\MSN>ipython
Python 3.5.2 |Anaconda 4.2.0 (64-bit)| (default, Jul  5 2016, 11:41:13) [MSC v.19
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: import numpy

In [2]:
```

Two interpreter in cmd terminal:

- **Python shell**
- **IPython shell**

Integrated Development Environment (IDE)

Idle



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\MSN>idle
```

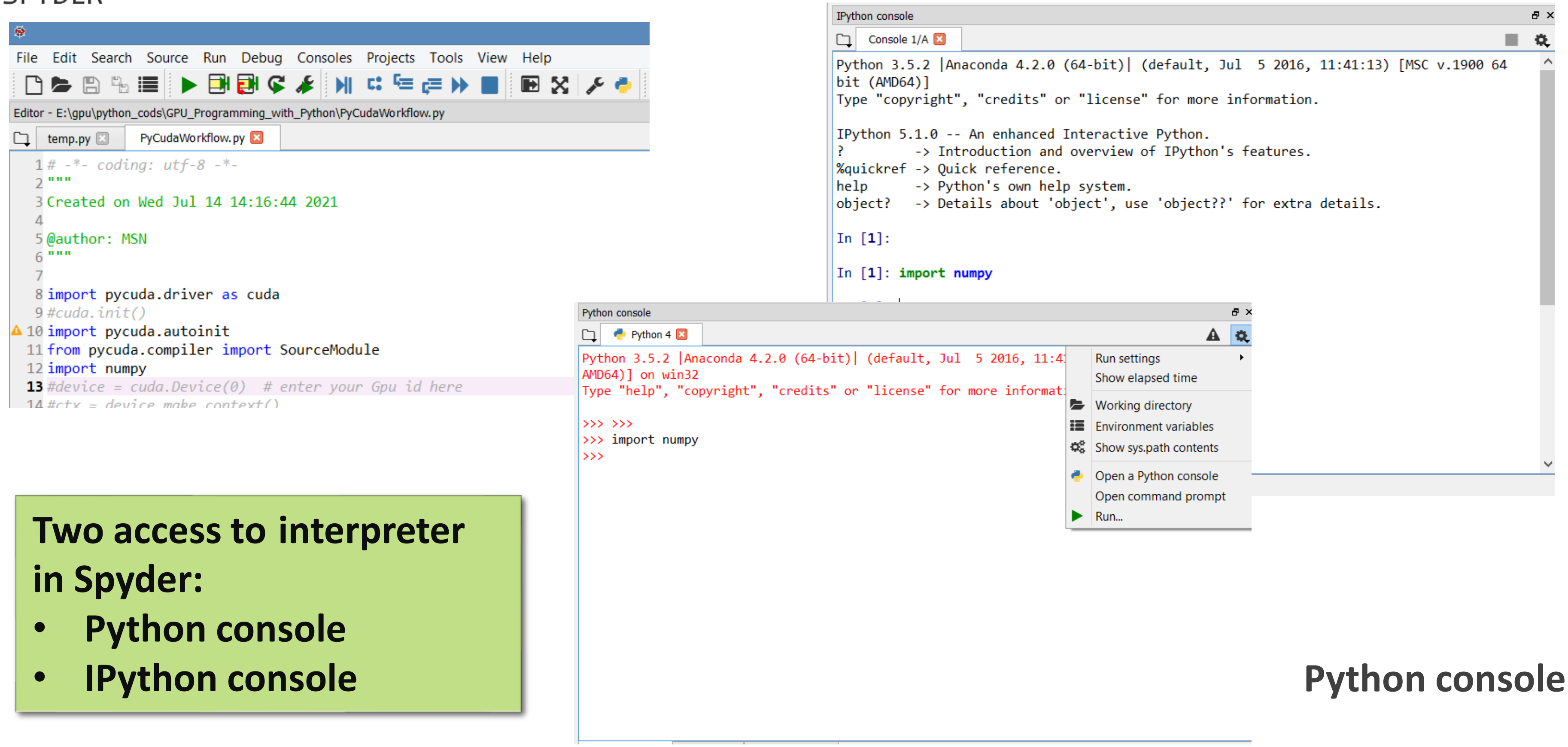
```
Python 3.5.2 |Anaconda 4.2.0 (64-bit)| (default, Jul  5 2016, 11:41:13) [MSC v.1
900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import numpy
>>> 
```

Integrated Development Environment (IDE)

SPYDER

Spyder

IPython console



Editor - E:\gpu\python_cods\GPU_Programming_with_Python\PyCudaWorkflow.py

```python
# -*- coding: utf-8 -*-
"""
Created on Wed Jul 14 14:16:44 2021

@author: MSN
"""

import pycuda.driver as cuda
#cuda.init()
import pycuda.autoinit
from pycuda.compiler import SourceModule
import numpy
#device = cuda.Device(0)  # enter your Gpu id here
#ctx = device.make_context()
```

```
Python 3.5.2 |Anaconda 4.2.0 (64-bit)| (default, Jul  5 2016, 11:41:13) [MSC v.1900 64
bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]:

In [1]: import numpy
```

```
Python 3.5.2 |Anaconda 4.2.0 (64-bit)| (default, Jul  5 2016, 11:4
AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more informat

>>> >>>
>>> import numpy
>>>
```

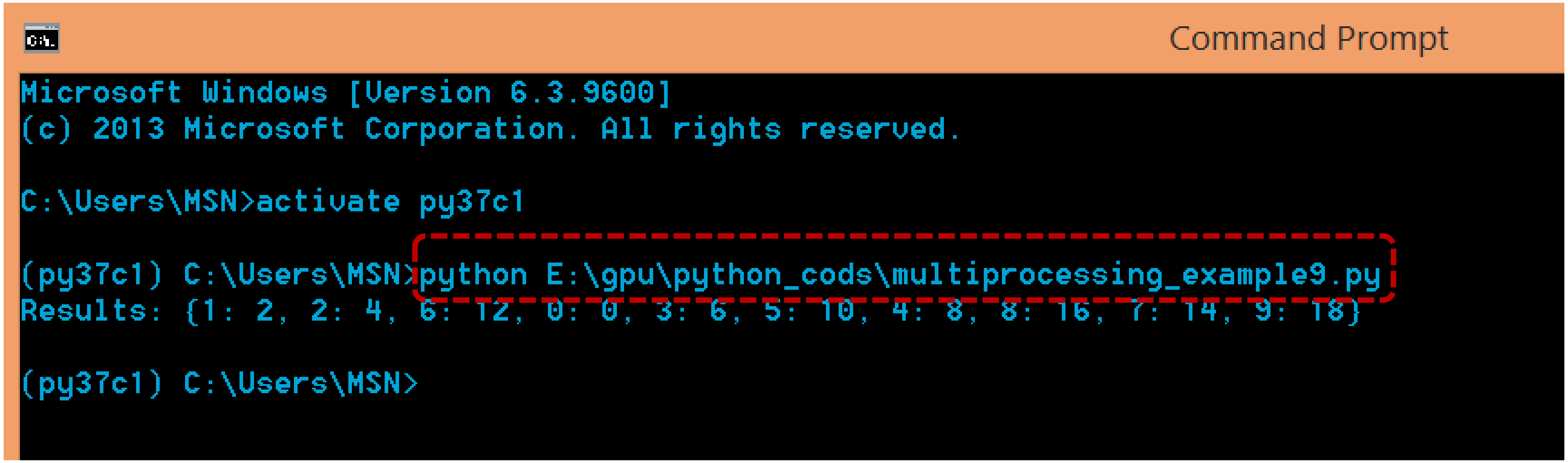
Two access to interpreter in Spyder:

- **Python console**
- **IPython console**

Python console

Integrated Development Environment (IDE)

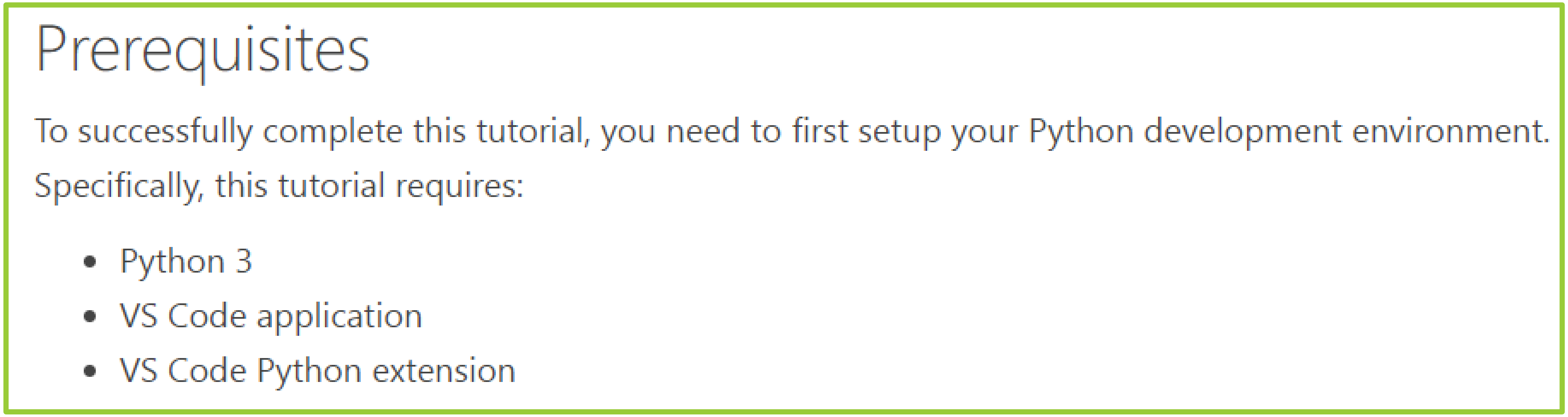
Executing the script inside a shell terminal
(Linux/Mac console or **cmd Windows console)**



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\MSN>activate py37c1

(py37c1) C:\Users\MSN>python E:\gpu\python_cods\multiprocessing_example9.py
Results: {1: 2, 2: 4, 6: 12, 0: 0, 3: 6, 5: 10, 4: 8, 8: 16, 7: 14, 9: 18}

(py37c1) C:\Users\MSN>
```

Integrated Development Environment (IDE)

Python console in VS Code



Prerequisites

To successfully complete this tutorial, you need to first setup your Python development environment. Specifically, this tutorial requires:

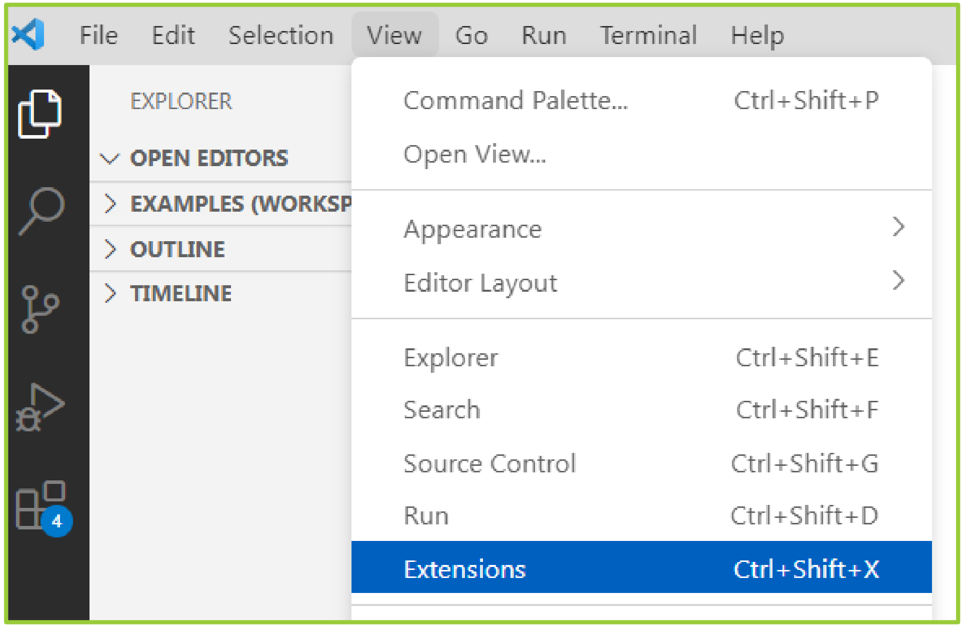
- Python 3
- VS Code application
- VS Code Python extension

https://code.visualstudio.com/docs/python/python-tutorial

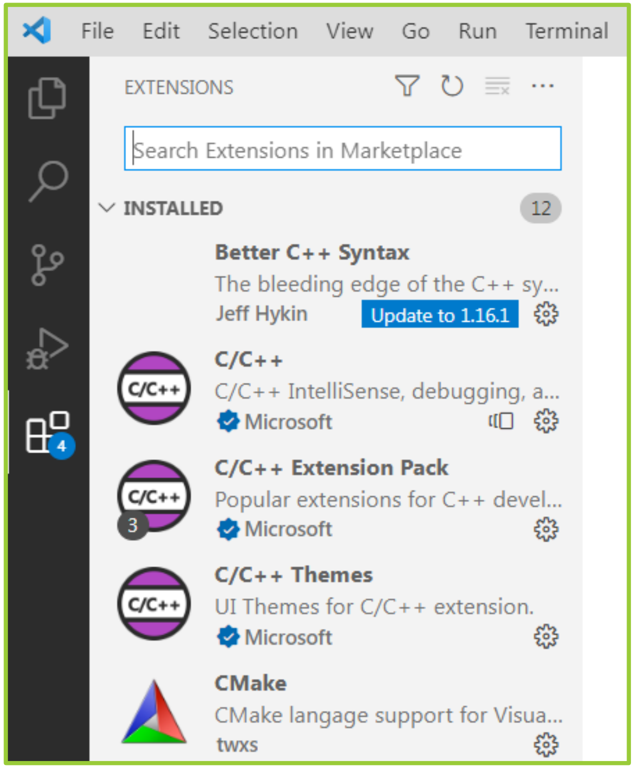
Integrated Development Environment (IDE)

Python console in VS Code

1



2

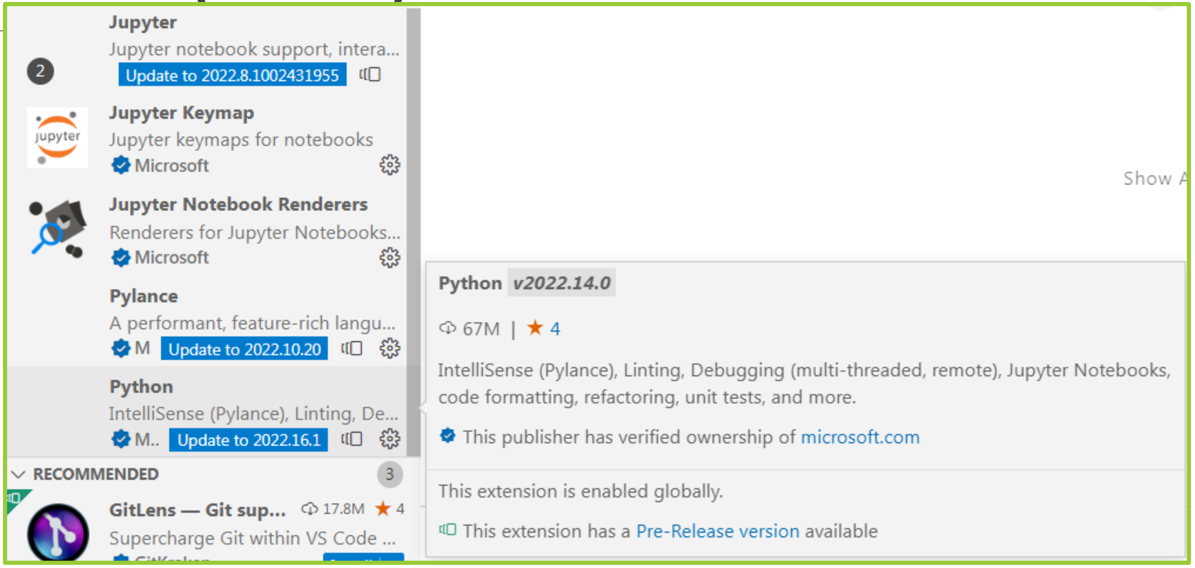

File
Edit
Selection
View
Go
Run
Terminal
EXTENSIONS
Search Extensions in Marketplace
INSTALLED
12
Better C++ Syntax
The bleeding edge of the C++ sy...
Jeff Hykin
Update to 1.16.1
C/C++
C/C++ IntelliSense, debugging, a...
Microsoft
C/C++ Extension Pack
Popular extensions for C++ devel...
Microsoft
3
C/C++ Themes
UI Themes for C/C++ extension.
Microsoft
CMake
CMake langage support for Visua...
twxs
4

https://code.visualstudio.com/docs/python/python-tutorial

Integrated Development Environment (IDE)

Python console in VS Code

1



Jupyter
Jupyter notebook support, intera...
2 Update to 2022.8.1002431955

Jupyter Keymap
Jupyter keymaps for notebooks
Microsoft

Jupyter Notebook Renderers
Renderers for Jupyter Notebooks...
Microsoft

Pylance
A performant, feature-rich langu...
M Update to 2022.10.20

Python
IntelliSense (Pylance), Linting, De...
M.. Update to 2022.16.1

RECOMMENDED 3

GitLens — Git sup... 17.8M ★ 4
Supercharge Git within VS Code ...

Show A

Python v2022.14.0

67M | ★ 4

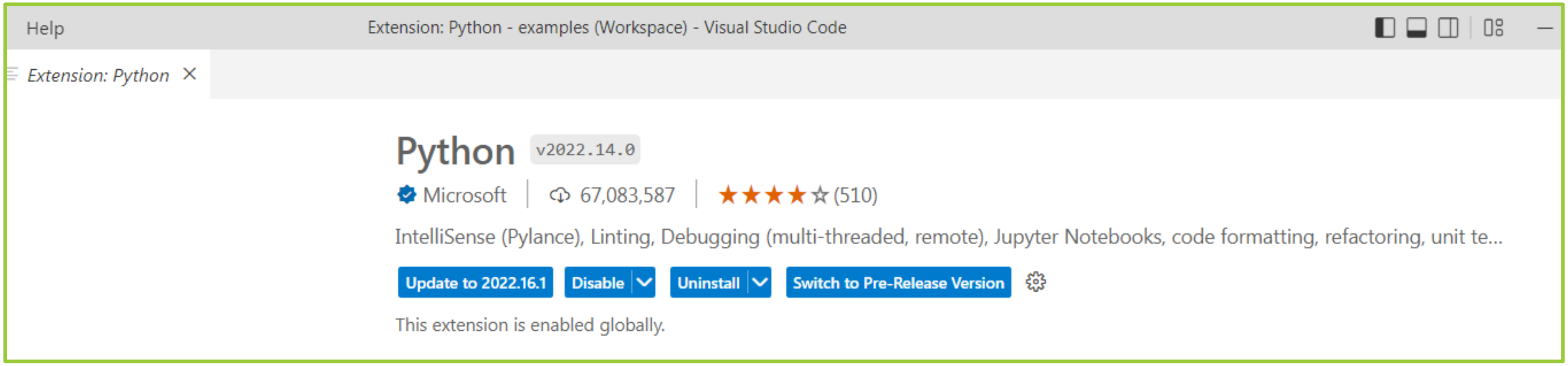
IntelliSense (Pylance), Linting, Debugging (multi-threaded, remote), Jupyter Notebooks, code formatting, refactoring, unit tests, and more.

This publisher has verified ownership of microsoft.com

This extension is enabled globally.

This extension has a Pre-Release version available

2



Help   Extension: Python - examples (Workspace) - Visual Studio Code

Extension: Python ×

Python v2022.14.0

Microsoft | 67,083,587 | ★★★★☆ (510)

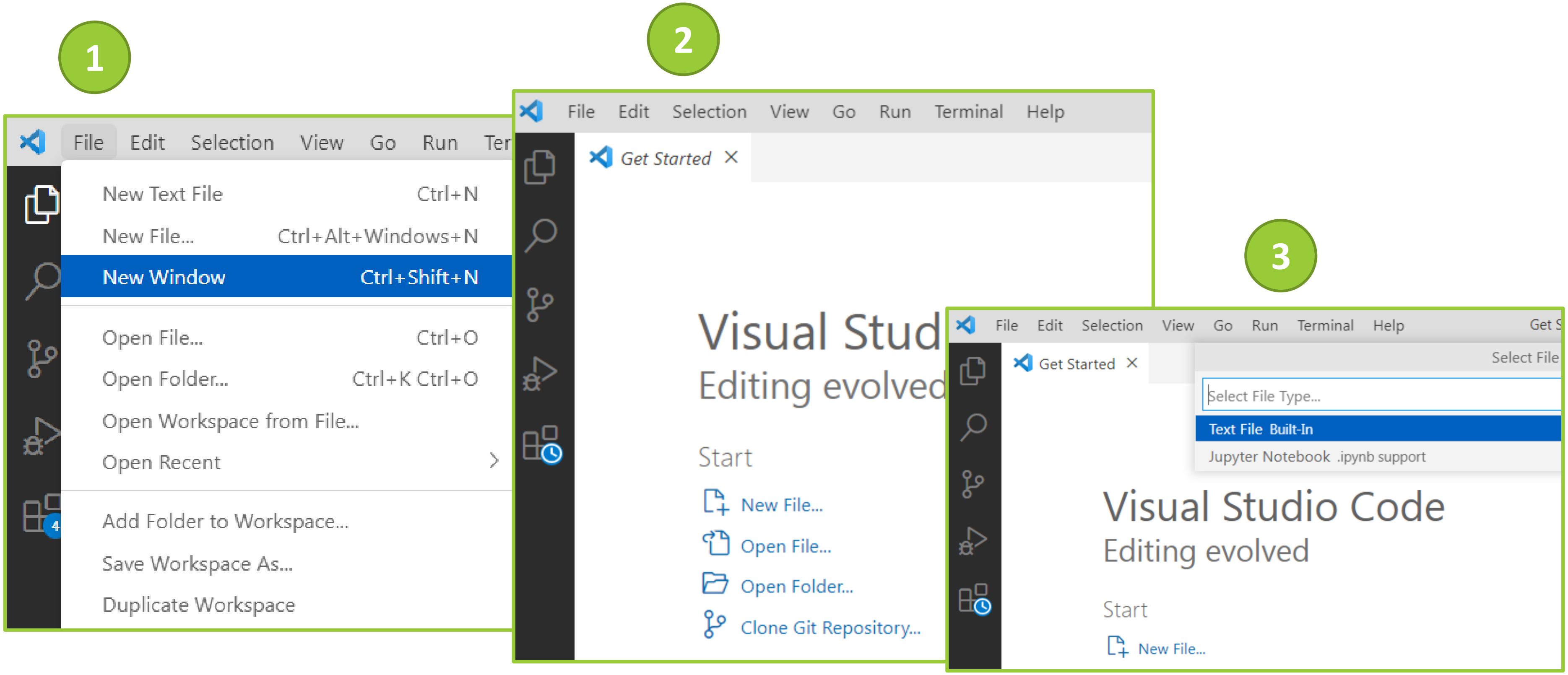
IntelliSense (Pylance), Linting, Debugging (multi-threaded, remote), Jupyter Notebooks, code formatting, refactoring, unit te...

This extension is enabled globally.

https://code.visualstudio.com/docs/python/python-tutorial

Integrated Development Environment (IDE)

Python console in VS Code

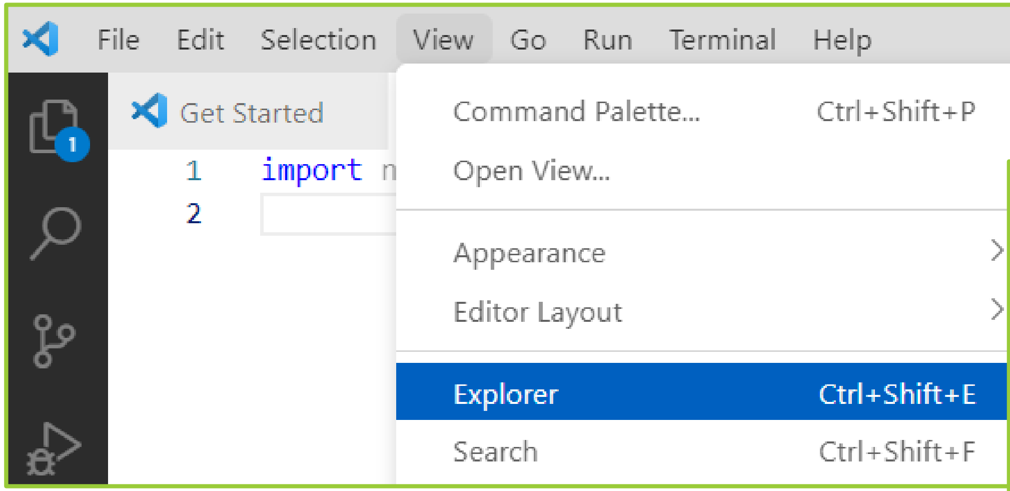


https://code.visualstudio.com/docs/python/python-tutorial

Integrated Development Environment (IDE)

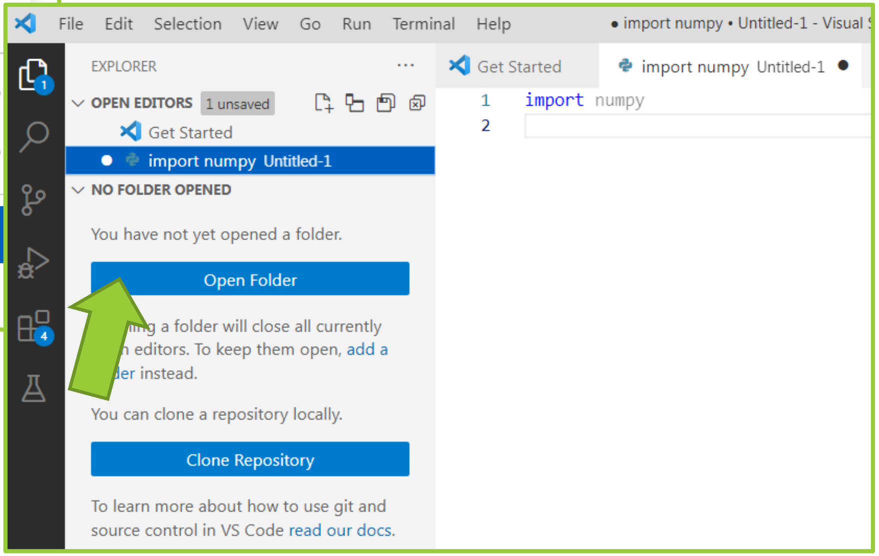
Python console in VS Code

1



```
import n
```

2



```
import numpy
```

You have not yet opened a folder.

...ing a folder will close all currently ...n editors. To keep them open, add a ...der instead.

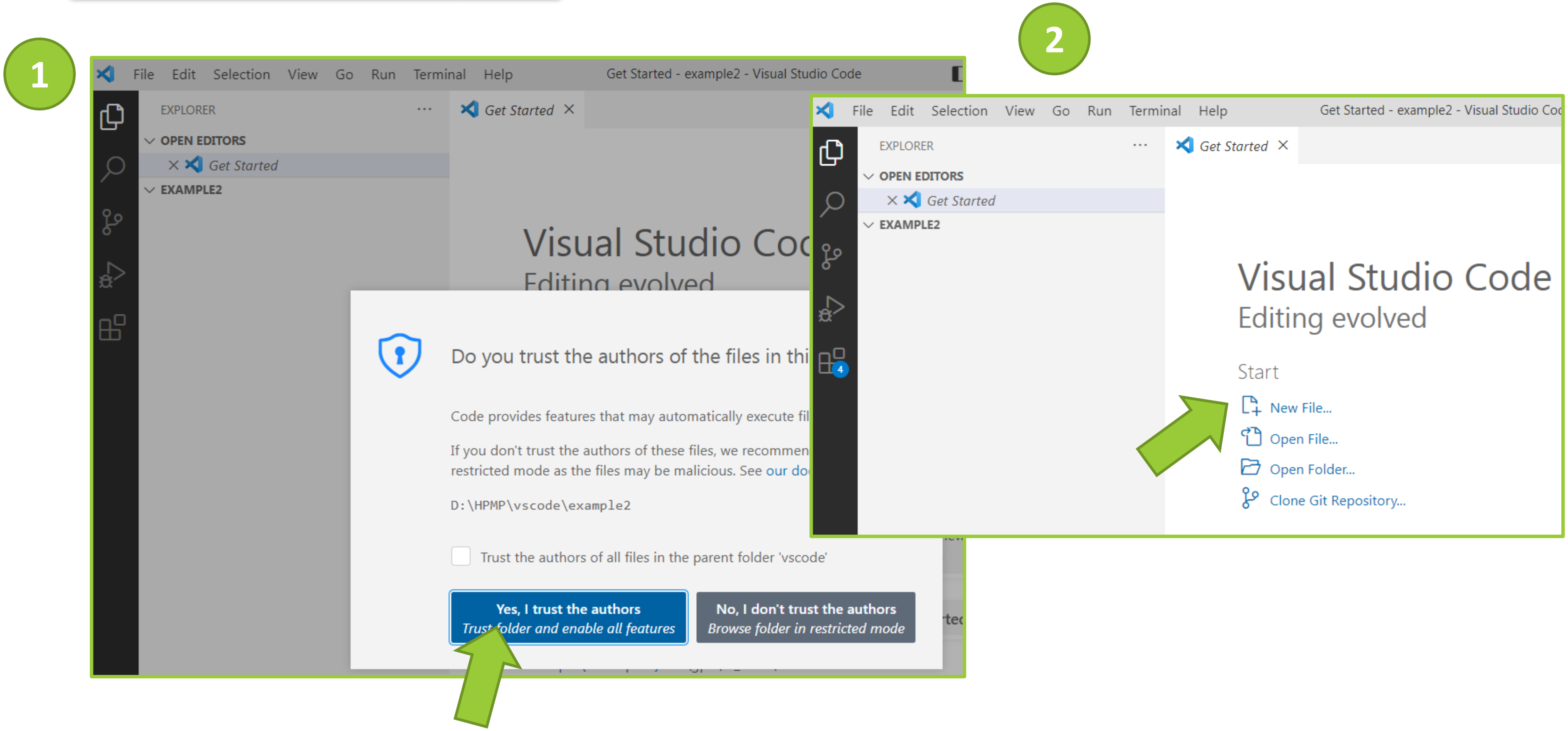
You can clone a repository locally.

To learn more about how to use git and source control in VS Code read our docs.

https://code.visualstudio.com/docs/python/python-tutorial

Integrated Development Environment (IDE)

Python console in VS Code


1
File Edit Selection View Go Run Terminal Help
Get Started - example2 - Visual Studio Code
EXPLORER
OPEN EDITORS
Get Started
EXAMPLE2
Visual Studio Coc
Editing evolved

Do you trust the authors of the files in thi

Code provides features that may automatically execute fil

If you don't trust the authors of these files, we recommen restricted mode as the files may be malicious. See our do

```
D:\HPMP\vscode\example2
```

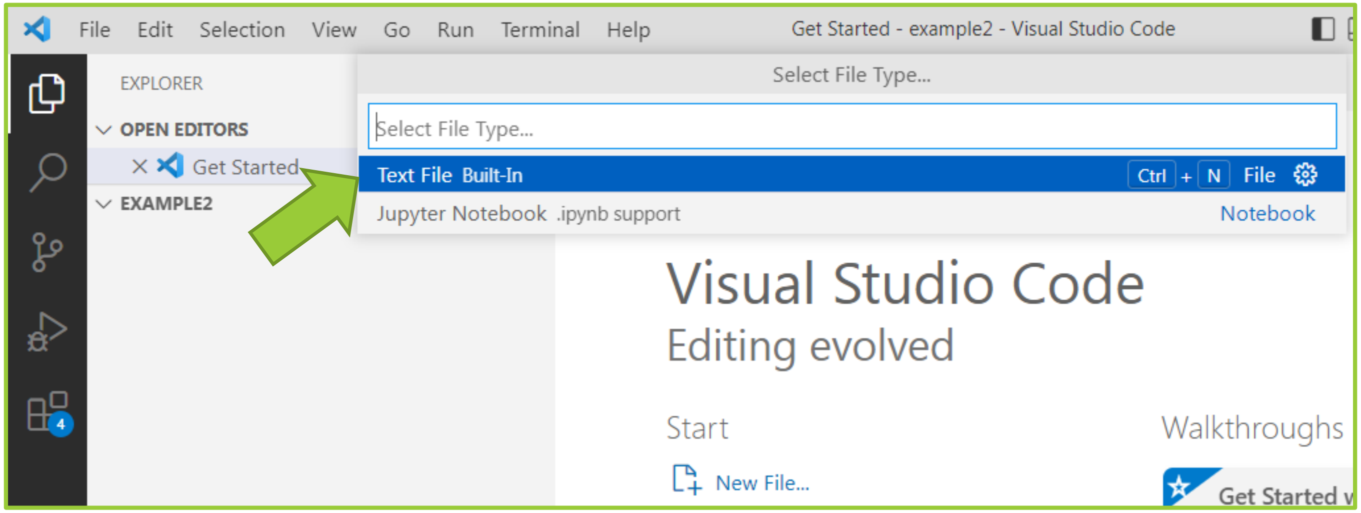
□ Trust the authors of all files in the parent folder 'vscode'

Yes, I trust the authors
Trust folder and enable all features
No, I don't trust the authors
Browse folder in restricted mode
2
File Edit Selection View Go Run Terminal Help
Get Started - example2 - Visual Studio Coc
EXPLORER
OPEN EDITORS
Get Started
EXAMPLE2
Visual Studio Code
Editing evolved
Start
New File...
Open File...
Open Folder...
Clone Git Repository...

https://code.visualstudio.com/docs/python/python-tutorial

Integrated Development Environment (IDE)

Python console in VS Code



Select File Type...

Text File Built-In Ctrl + N File

Jupyter Notebook .ipynb support Notebook

Visual Studio Code

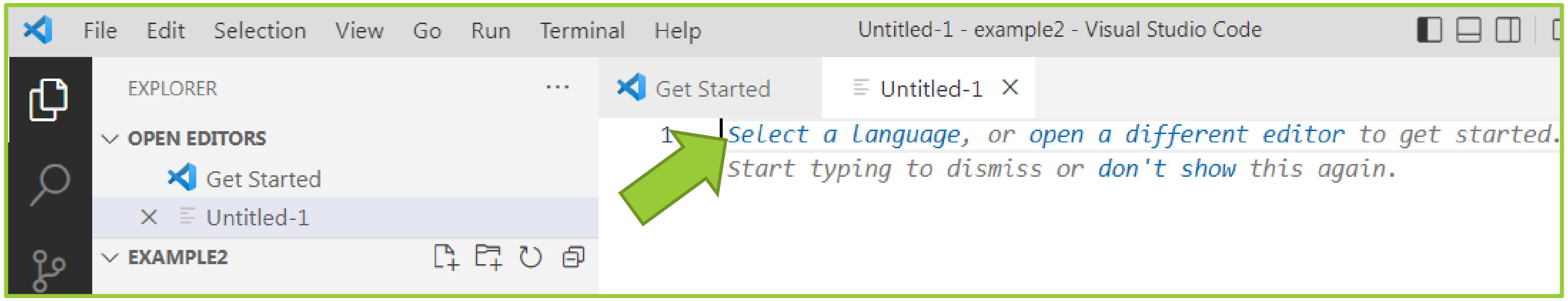
Editing evolved

Start

New File...

Walkthroughs

Get Started w

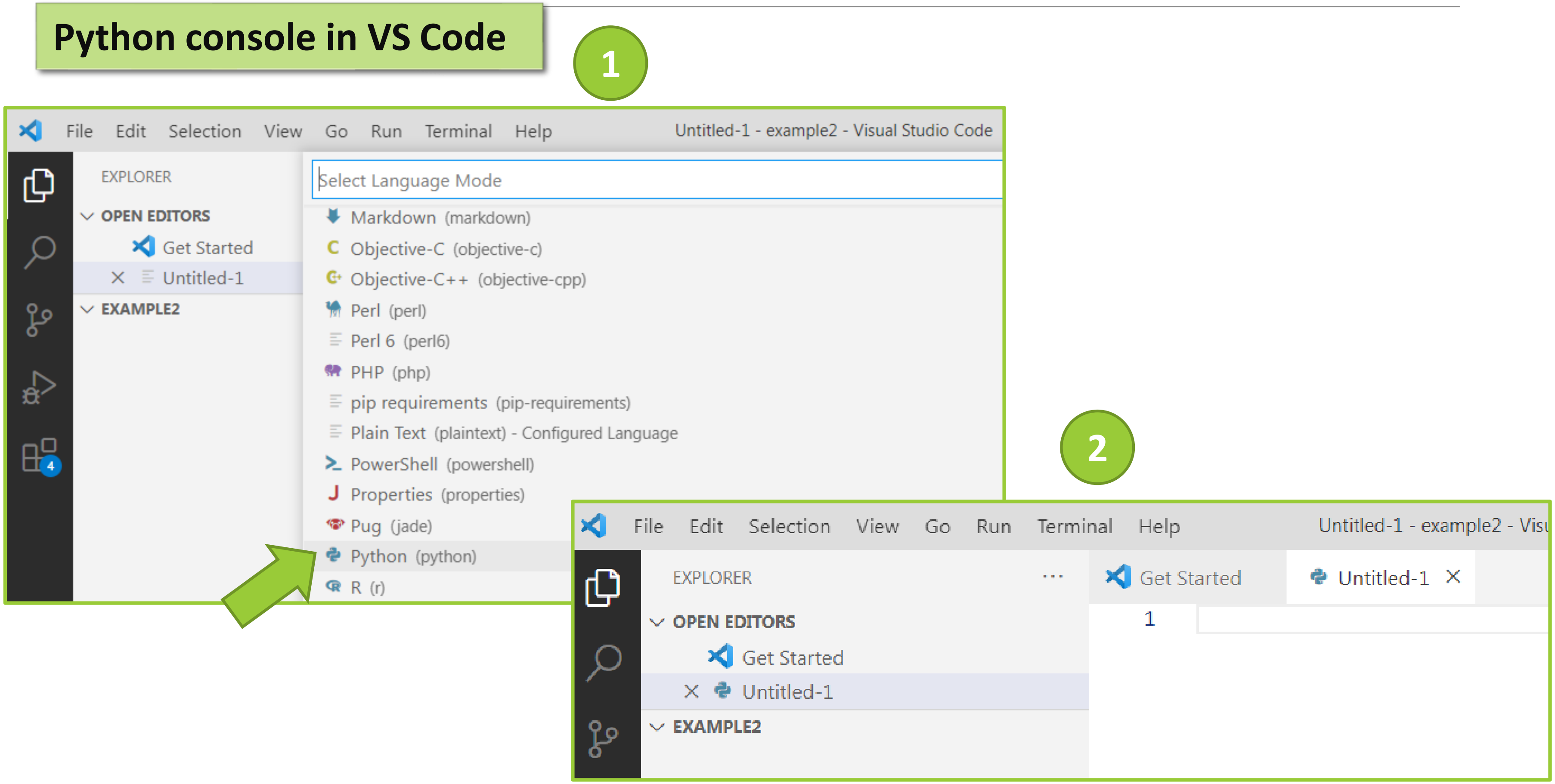


```
Select a language, or open a different editor to get started.
Start typing to dismiss or don't show this again.
```

https://code.visualstudio.com/docs/python/python-tutorial

Integrated Development Environment (IDE)

Python console in VS Code

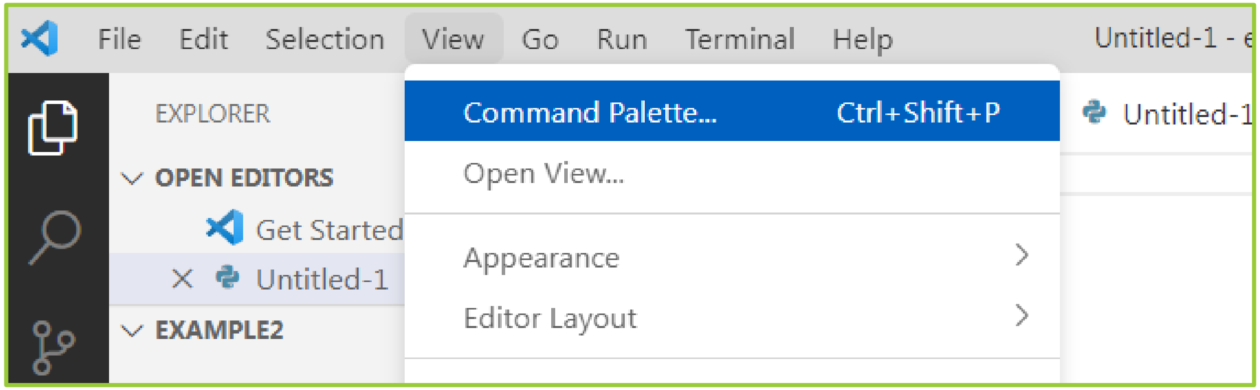

1
File Edit Selection View Go Run Terminal Help
Untitled-1 - example2 - Visual Studio Code
EXPLORER
OPEN EDITORS
Get Started
Untitled-1
EXAMPLE2
Select Language Mode
Markdown (markdown)
Objective-C (objective-c)
Objective-C++ (objective-cpp)
Perl (perl)
Perl 6 (perl6)
PHP (php)
pip requirements (pip-requirements)
Plain Text (plaintext) - Configured Language
PowerShell (powershell)
Properties (properties)
Pug (jade)
Python (python)
R (r)
2
File Edit Selection View Go Run Terminal Help
Untitled-1 - example2 - Visu
EXPLORER
OPEN EDITORS
Get Started
Untitled-1
EXAMPLE2
Get Started
Untitled-1
1

https://code.visualstudio.com/docs/python/python-tutorial

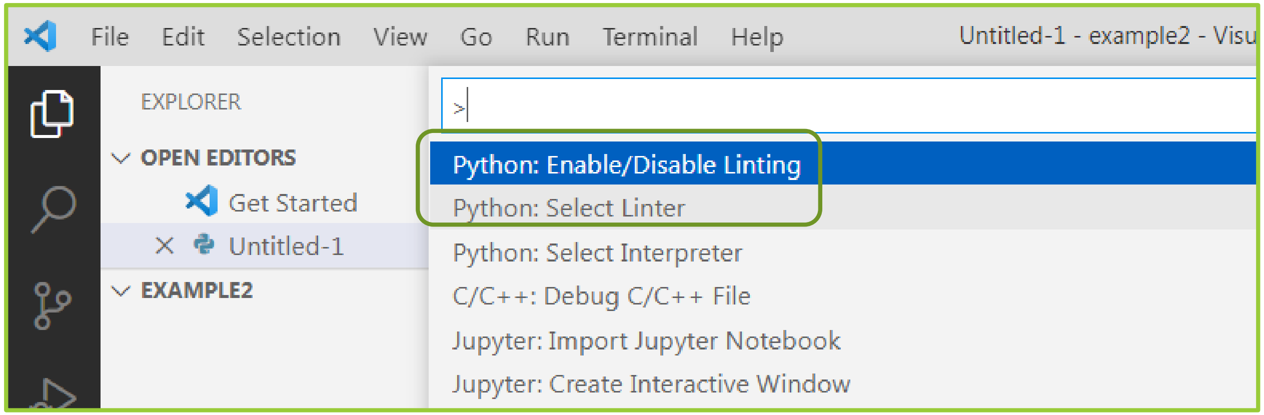
Integrated Development Environment (IDE)

Python console in VS Code

1



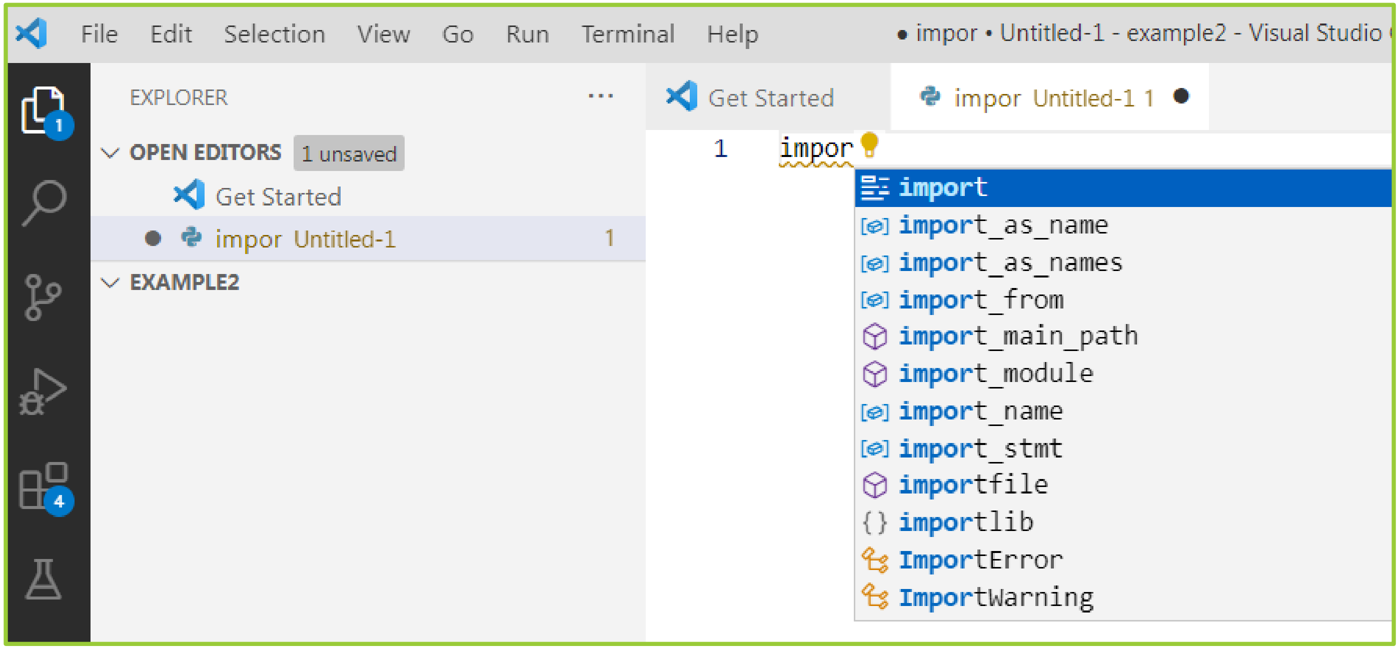
2


File
Edit
Selection
View
Go
Run
Terminal
Help
Untitled-1 - example2 - Visu
EXPLORER
OPEN EDITORS
Get Started
Untitled-1
EXAMPLE2
>
Python: Enable/Disable Linting
Python: Select Linter
Python: Select Interpreter
C/C++: Debug C/C++ File
Jupyter: Import Jupyter Notebook
Jupyter: Create Interactive Window

https://code.visualstudio.com/docs/python/python-tutorial

Integrated Development Environment (IDE)

Python console in VS Code



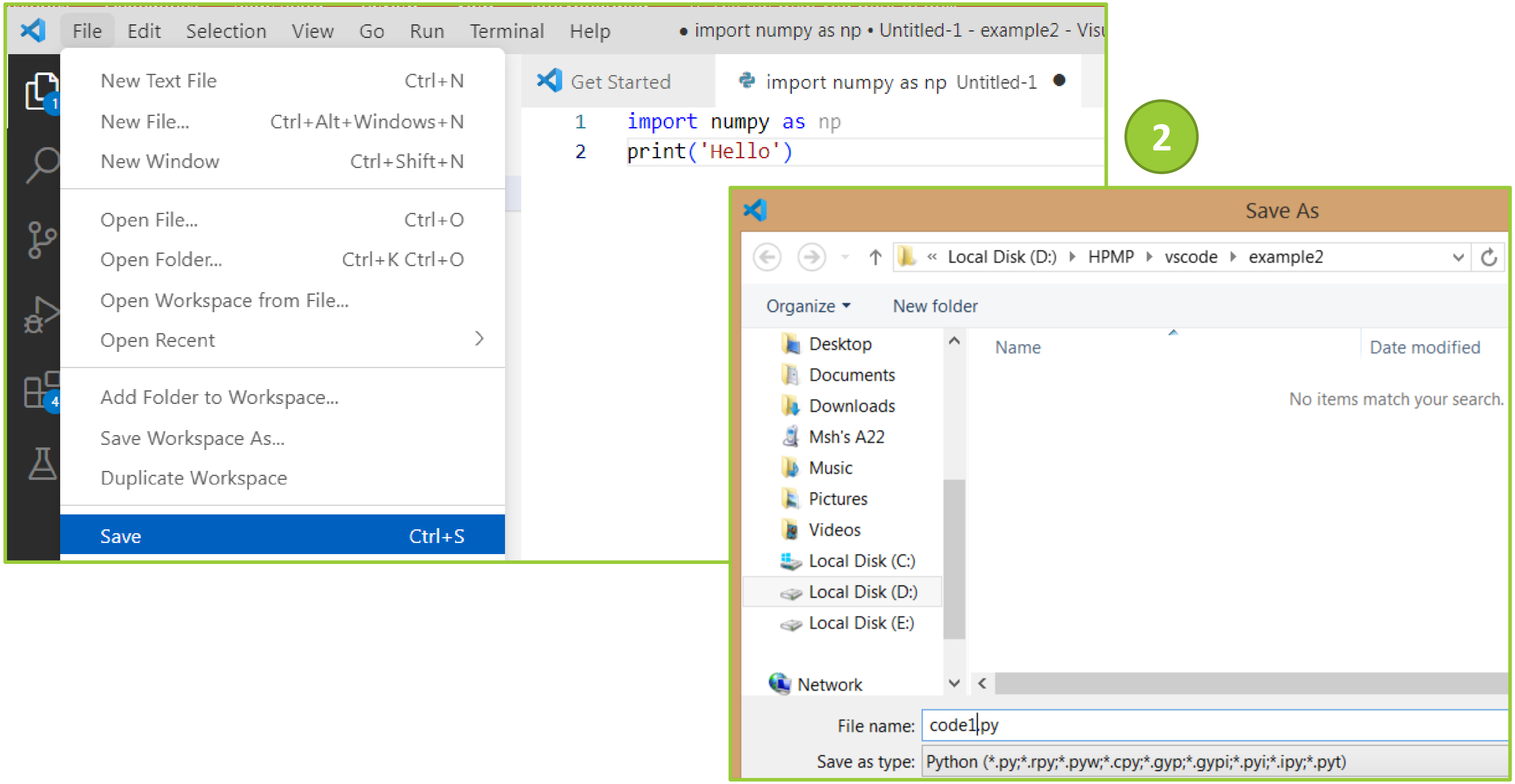
```
impor
import
import_as_name
import_as_names
import_from
import_main_path
import_module
import_name
import_stmt
importfile
importlib
ImportError
ImportWarning
```

https://code.visualstudio.com/docs/python/python-tutorial

Integrated Development Environment (IDE)

Python console in VS Code

1



```python
import numpy as np
print('Hello')
```

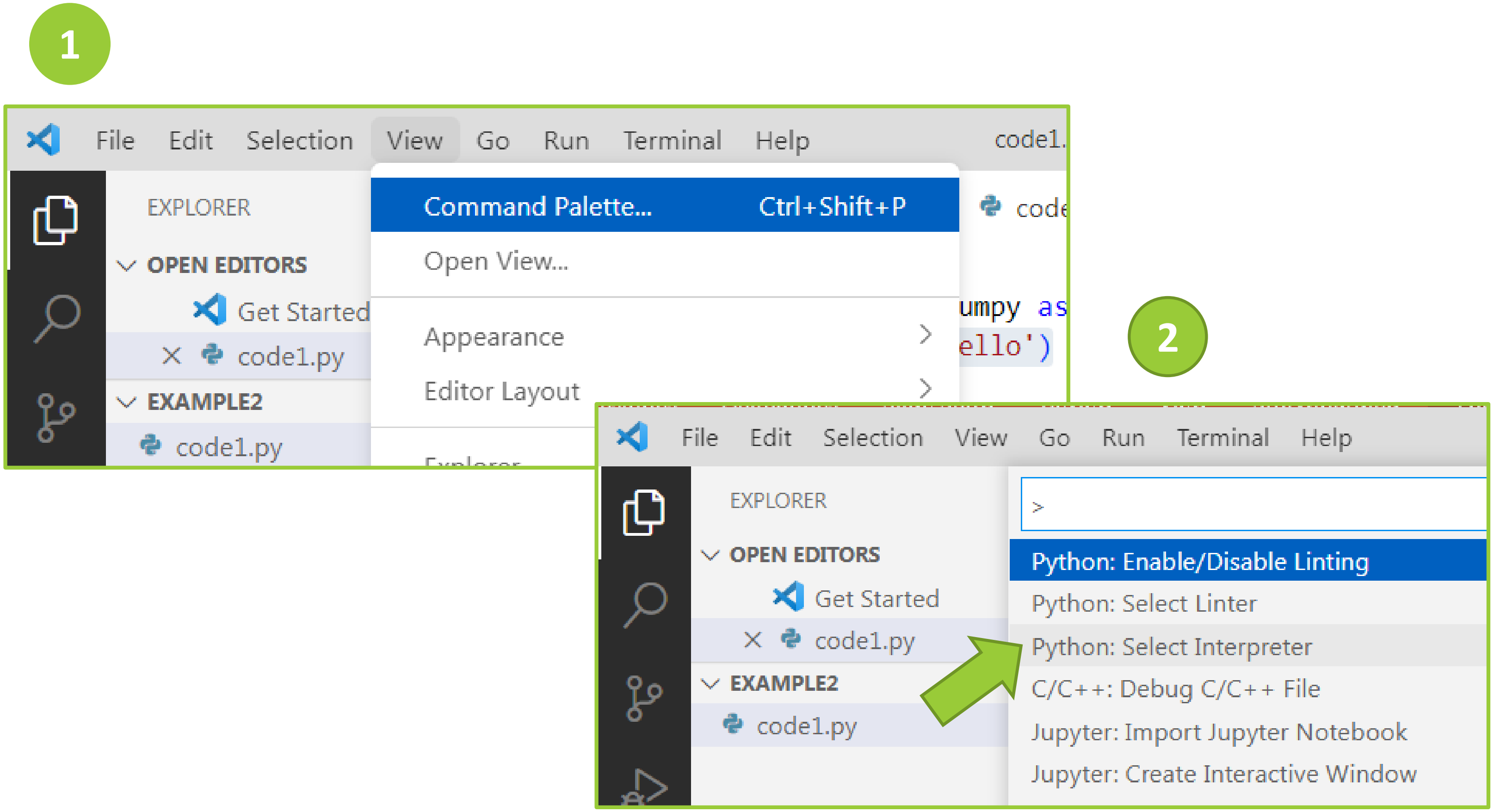
File name: code1.py

Save as type: Python (*.py;*.rpy;*.pyw;*.cpy;*.gyp;*.gypi;*.pyi;*.ipy;*.pyt)

https://code.visualstudio.com/docs/python/python-tutorial

Integrated Development Environment (IDE)

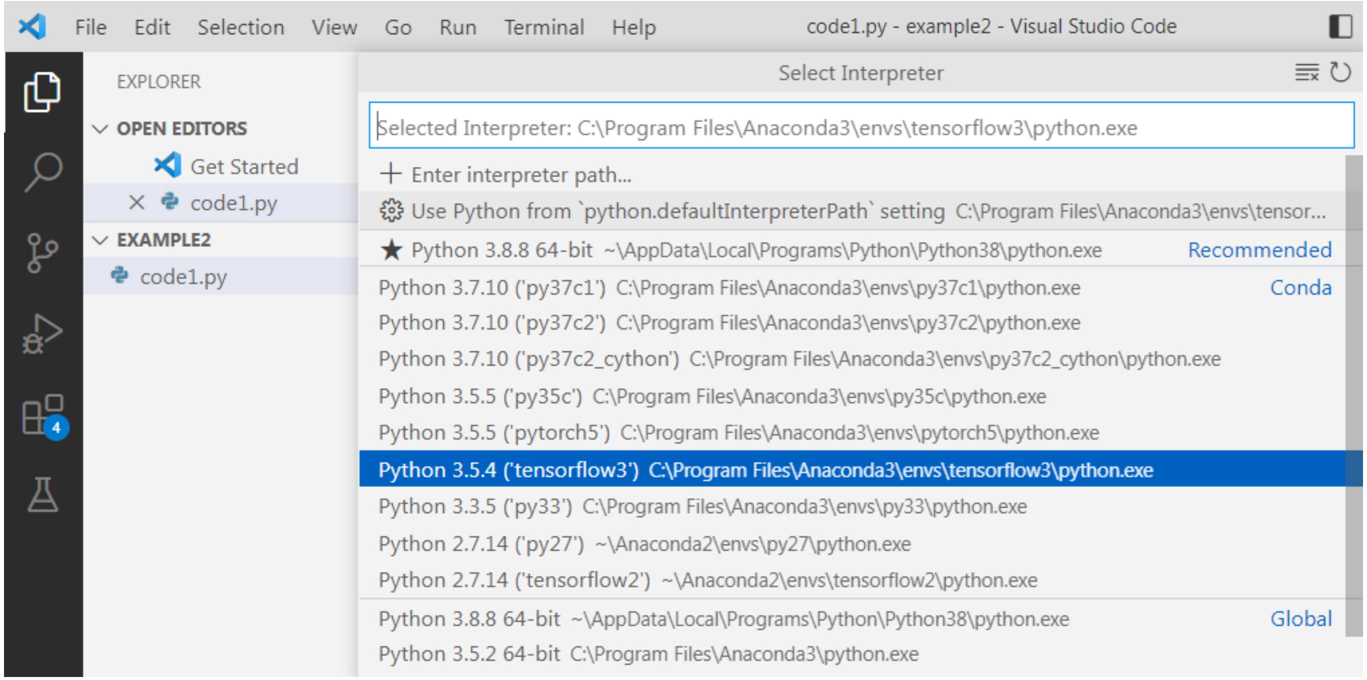
Python console in VS Code


1
File Edit Selection View Go Run Terminal Help code1.
EXPLORER
OPEN EDITORS
Get Started
code1.py
EXAMPLE2
code1.py
Command Palette... Ctrl+Shift+P
Open View...
Appearance
Editor Layout
Explorer
code
umpy as
ello')
2
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
Get Started
code1.py
EXAMPLE2
code1.py
>
Python: Enable/Disable Linting
Python: Select Linter
Python: Select Interpreter
C/C++: Debug C/C++ File
Jupyter: Import Jupyter Notebook
Jupyter: Create Interactive Window

https://code.visualstudio.com/docs/python/python-tutorial

Integrated Development Environment (IDE)

Python console in VS Code



Select Interpreter

Selected Interpreter: C:\Program Files\Anaconda3\envs\tensorflow3\python.exe

+ Enter interpreter path...

Use Python from `python.defaultInterpreterPath` setting C:\Program Files\Anaconda3\envs\tensor...

★ Python 3.8.8 64-bit ~\AppData\Local\Programs\Python\Python38\python.exe Recommended

Python 3.7.10 ('py37c1') C:\Program Files\Anaconda3\envs\py37c1\python.exe Conda

Python 3.7.10 ('py37c2') C:\Program Files\Anaconda3\envs\py37c2\python.exe

Python 3.7.10 ('py37c2_cython') C:\Program Files\Anaconda3\envs\py37c2_cython\python.exe

Python 3.5.5 ('py35c') C:\Program Files\Anaconda3\envs\py35c\python.exe

Python 3.5.5 ('pytorch5') C:\Program Files\Anaconda3\envs\pytorch5\python.exe

Python 3.5.4 ('tensorflow3') C:\Program Files\Anaconda3\envs\tensorflow3\python.exe

Python 3.3.5 ('py33') C:\Program Files\Anaconda3\envs\py33\python.exe

Python 2.7.14 ('py27') ~\Anaconda2\envs\py27\python.exe

Python 2.7.14 ('tensorflow2') ~\Anaconda2\envs\tensorflow2\python.exe

Python 3.8.8 64-bit ~\AppData\Local\Programs\Python\Python38\python.exe Global

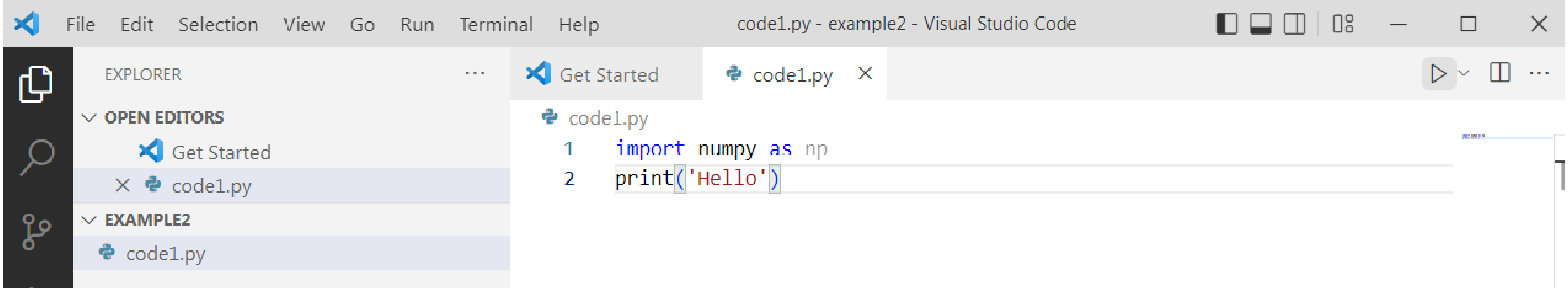
Python 3.5.2 64-bit C:\Program Files\Anaconda3\python.exe

https://code.visualstudio.com/docs/python/python-tutorial

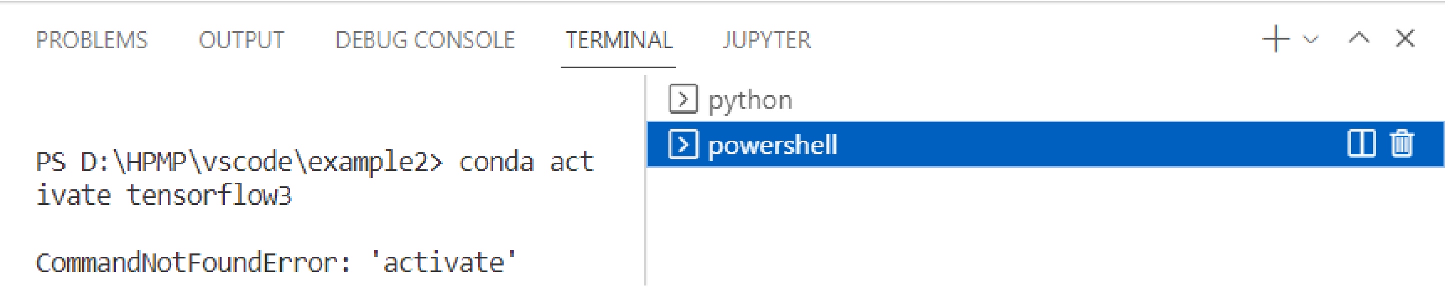
Integrated Development Environment (IDE)

Python console in VS Code

1



```python
import numpy as np
print('Hello')
```



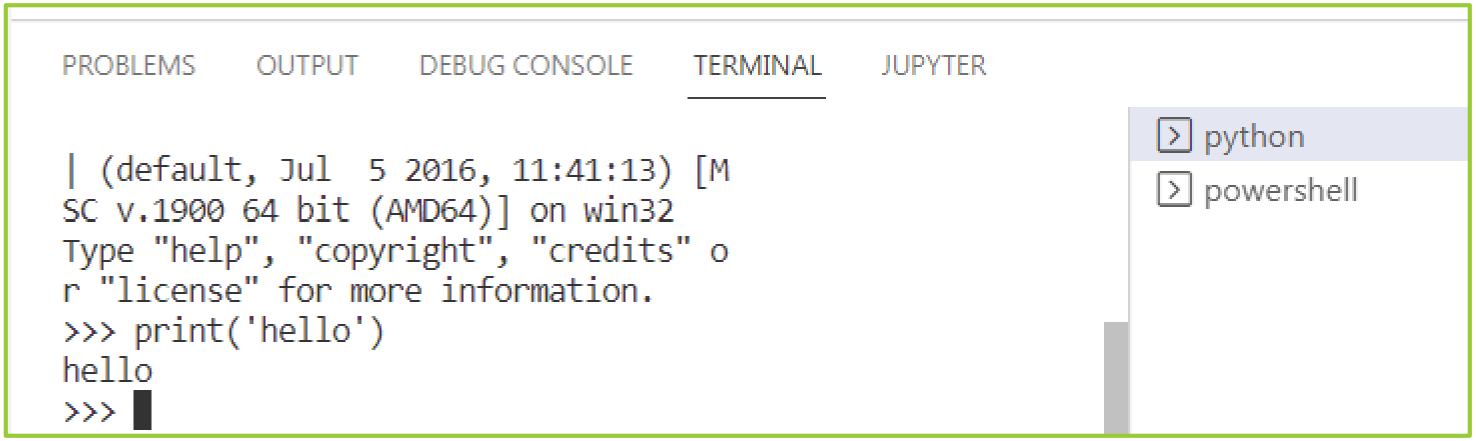
```
PS D:\HPMP\vscode\example2> conda act
ivate tensorflow3

CommandNotFoundError: 'activate'
```

2

https://code.visualstudio.com/docs/python/python-tutorial

Integrated Development Environment (IDE)

Python console in VS Code

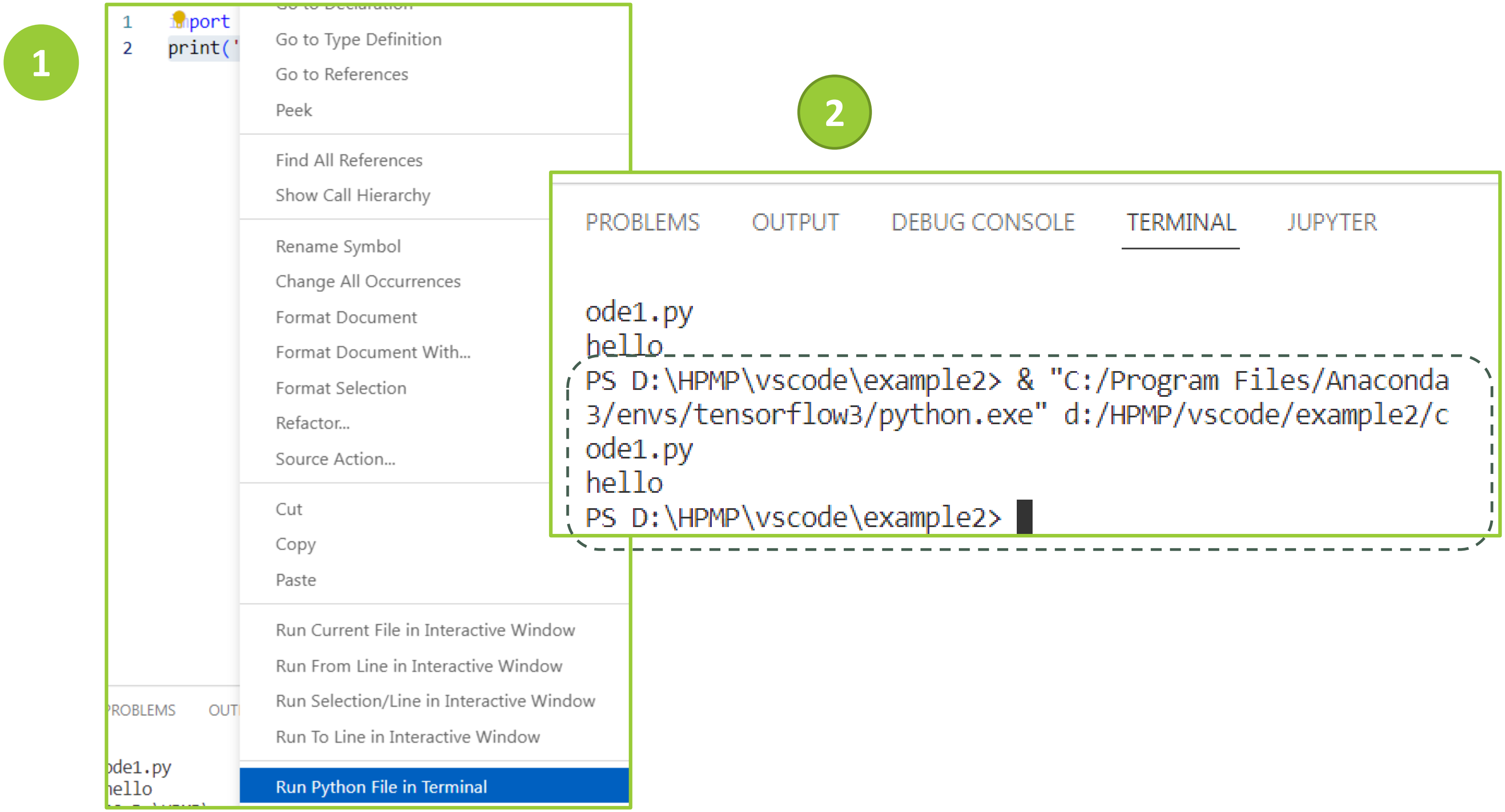


```
| (default, Jul  5 2016, 11:41:13) [M
SC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" o
r "license" for more information.
>>> print('hello')
hello
>>> 
```

https://code.visualstudio.com/docs/python/python-tutorial

Integrated Development Environment (IDE)

Python console in VS Code



```
import
print('
```

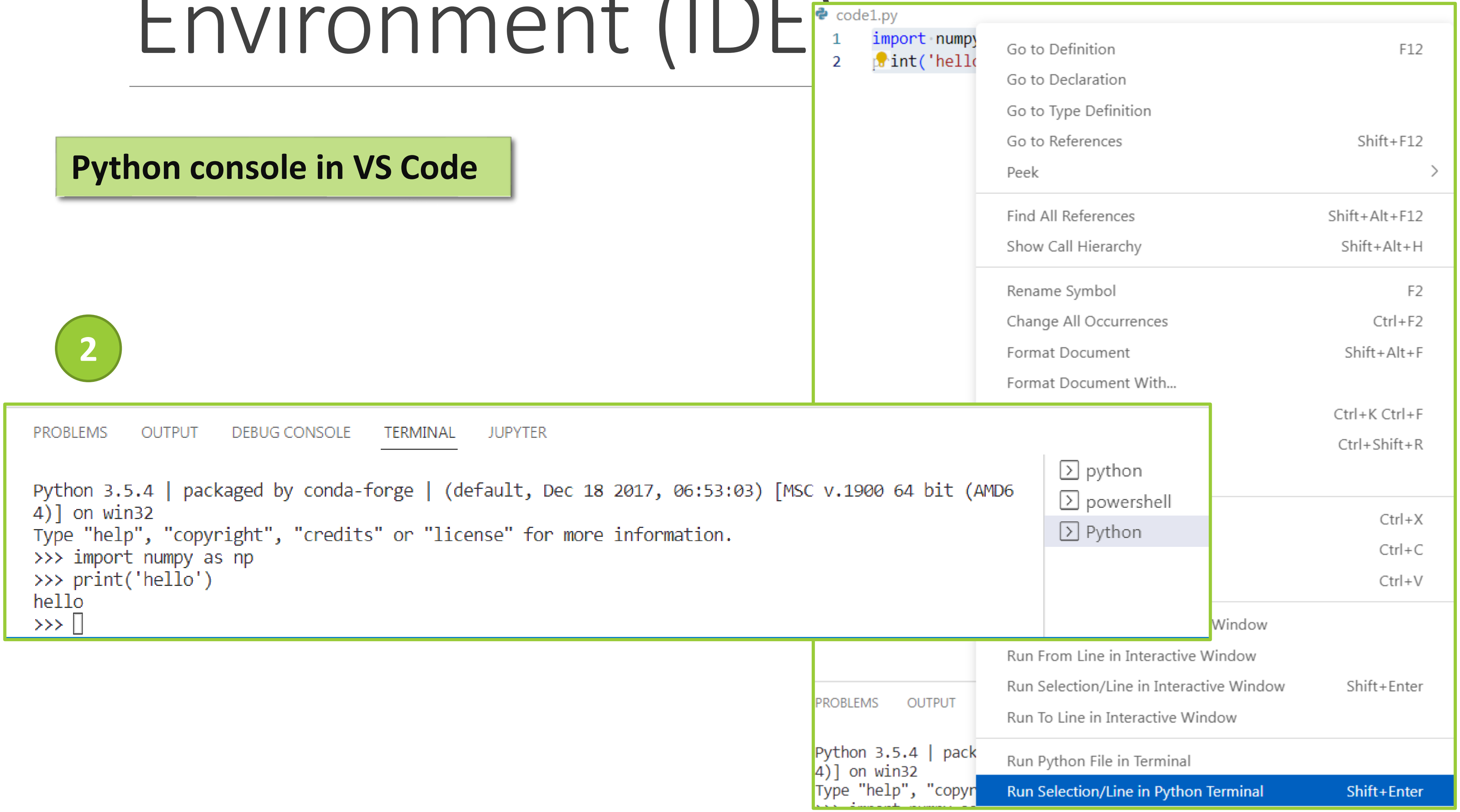
```
ode1.py
hello
```

```
ode1.py
hello
PS D:\HPMP\vscode\example2> & "C:/Program Files/Anaconda
3/envs/tensorflow3/python.exe" d:/HPMP/vscode/example2/c
ode1.py
hello
PS D:\HPMP\vscode\example2>
```

https://code.visualstudio.com/docs/python/python-tutorial

Integrated Development Environment (IDE)

Python console in VS Code


1
code1.py
Go to Definition F12
Go to Declaration
Go to Type Definition
Go to References Shift+F12
Peek
Find All References Shift+Alt+F12
Show Call Hierarchy Shift+Alt+H
Rename Symbol F2
Change All Occurrences Ctrl+F2
Format Document Shift+Alt+F
Format Document With...
Ctrl+K Ctrl+F
Ctrl+Shift+R
Ctrl+X
Ctrl+C
Ctrl+V
Window
Run From Line in Interactive Window
Run Selection/Line in Interactive Window Shift+Enter
Run To Line in Interactive Window
Run Python File in Terminal
Run Selection/Line in Python Terminal Shift+Enter
PROBLEMS
OUTPUT
Python 3.5.4 | pack
4)] on win32
Type "help", "copyr
2
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
JUPYTER
python
powershell
Python

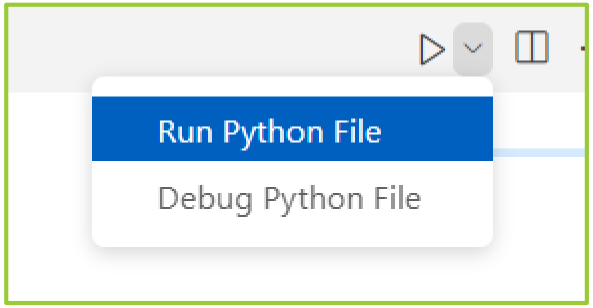
```
import numpy
print('hell
```

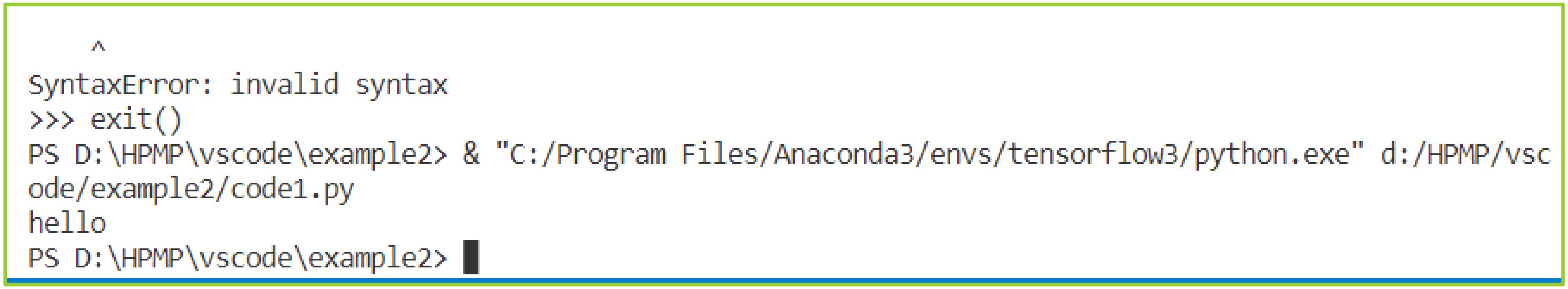
```
Python 3.5.4 | packaged by conda-forge | (default, Dec 18 2017, 06:53:03) [MSC v.1900 64 bit (AMD6
4)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy as np
>>> print('hello')
hello
>>> 
```

https://code.visualstudio.com/docs/python/python-tutorial

Integrated Development Environment (IDE)

Python console in VS Code



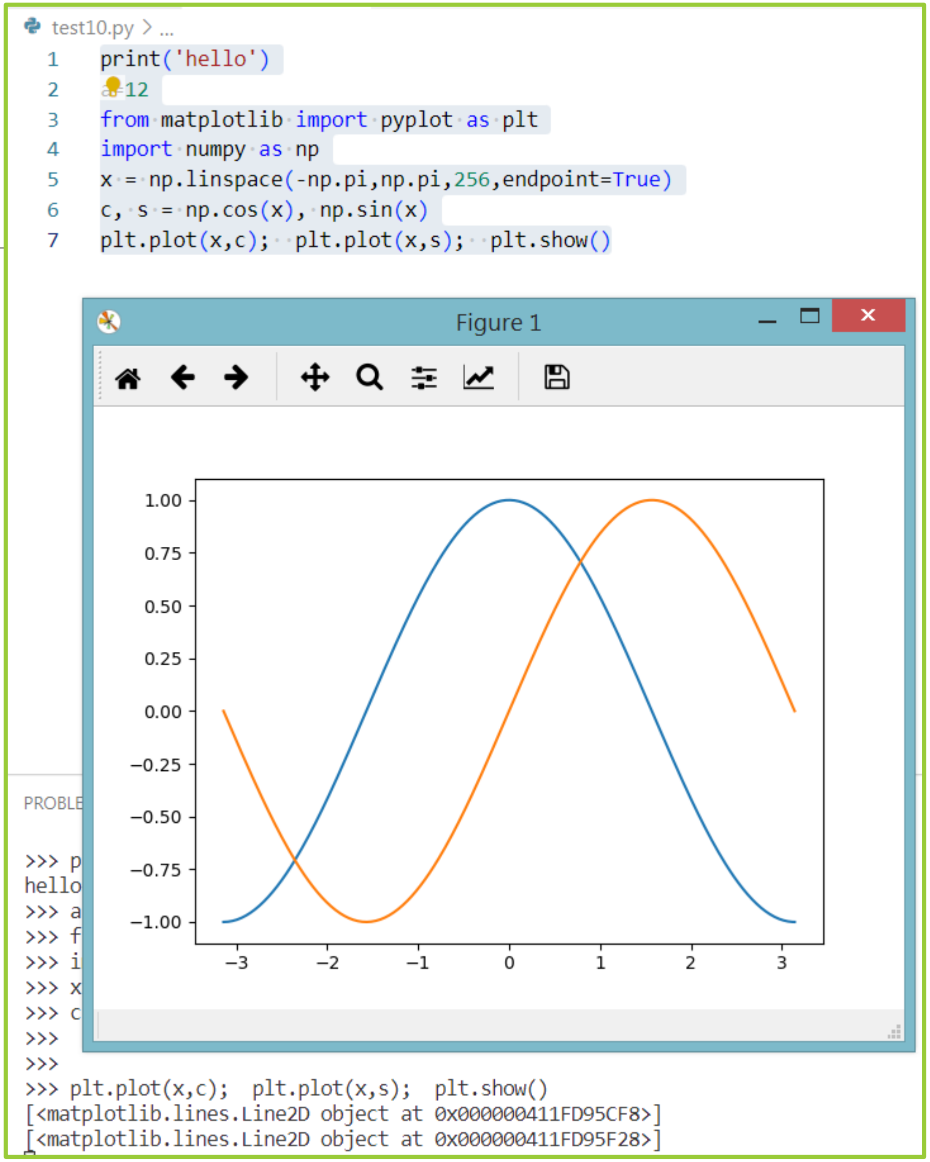


```
    ^
SyntaxError: invalid syntax
>>> exit()
PS D:\HPMP\vscode\example2> & "C:/Program Files/Anaconda3/envs/tensorflow3/python.exe" d:/HPMP/vsc
ode/example2/code1.py
hello
PS D:\HPMP\vscode\example2> █
```

https://code.visualstudio.com/docs/python/python-tutorial

Integrated Development Environment (IDE)

Python console in VS Code

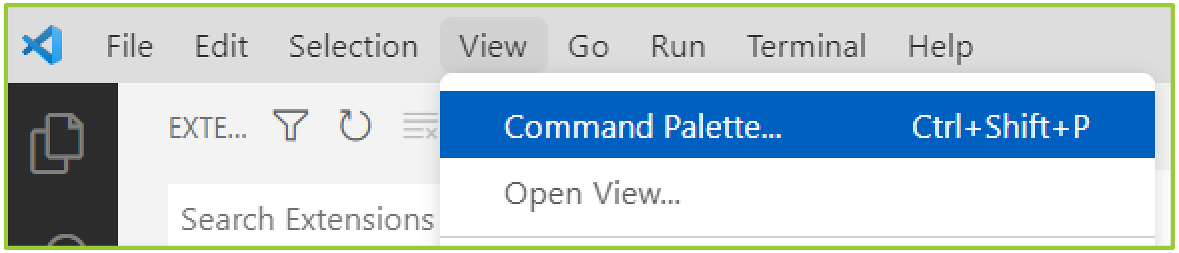


```
print('hello')
a=12
from matplotlib import pyplot as plt
import numpy as np
x = np.linspace(-np.pi,np.pi,256,endpoint=True)
c, s = np.cos(x), np.sin(x)
plt.plot(x,c);  plt.plot(x,s);  plt.show()
```

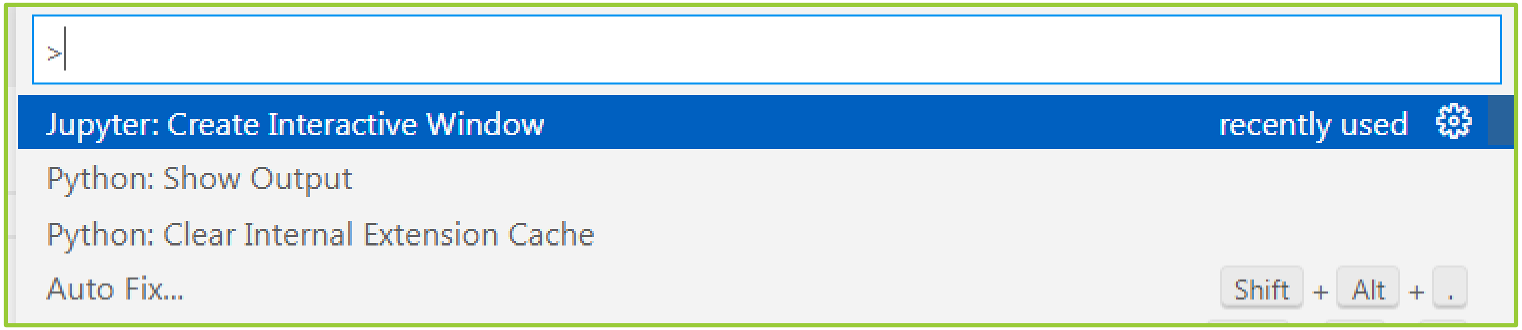
```
>>> p
hello
>>> a
>>> f
>>> i
>>> x
>>> c
>>>
>>>
>>> plt.plot(x,c);  plt.plot(x,s);  plt.show()
[<matplotlib.lines.Line2D object at 0x000000411FD95CF8>]
[<matplotlib.lines.Line2D object at 0x000000411FD95F28>]
```

Integrated Development Environment (IDE)

1

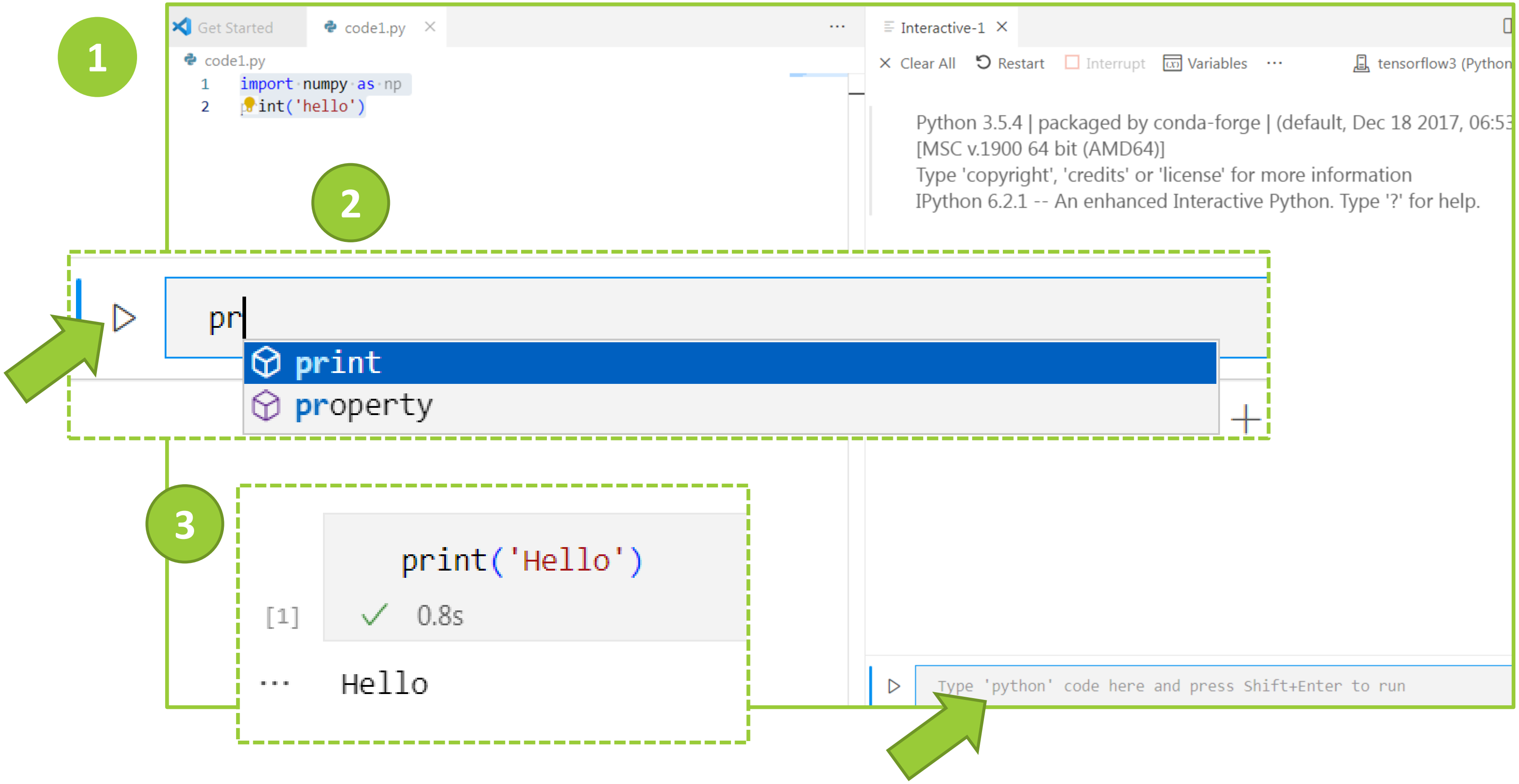


2


>
Jupyter: Create Interactive Window
recently used
Python: Show Output
Python: Clear Internal Extension Cache
Auto Fix...
Shift + Alt + .

https://code.visualstudio.com/docs/python/python-tutorial

Integrated Development Environment (IDE)



```python
import numpy as np
print('hello')
```

```
Python 3.5.4 | packaged by conda-forge | (default, Dec 18 2017, 06:53
[MSC v.1900 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 6.2.1 -- An enhanced Interactive Python. Type '?' for help.
```

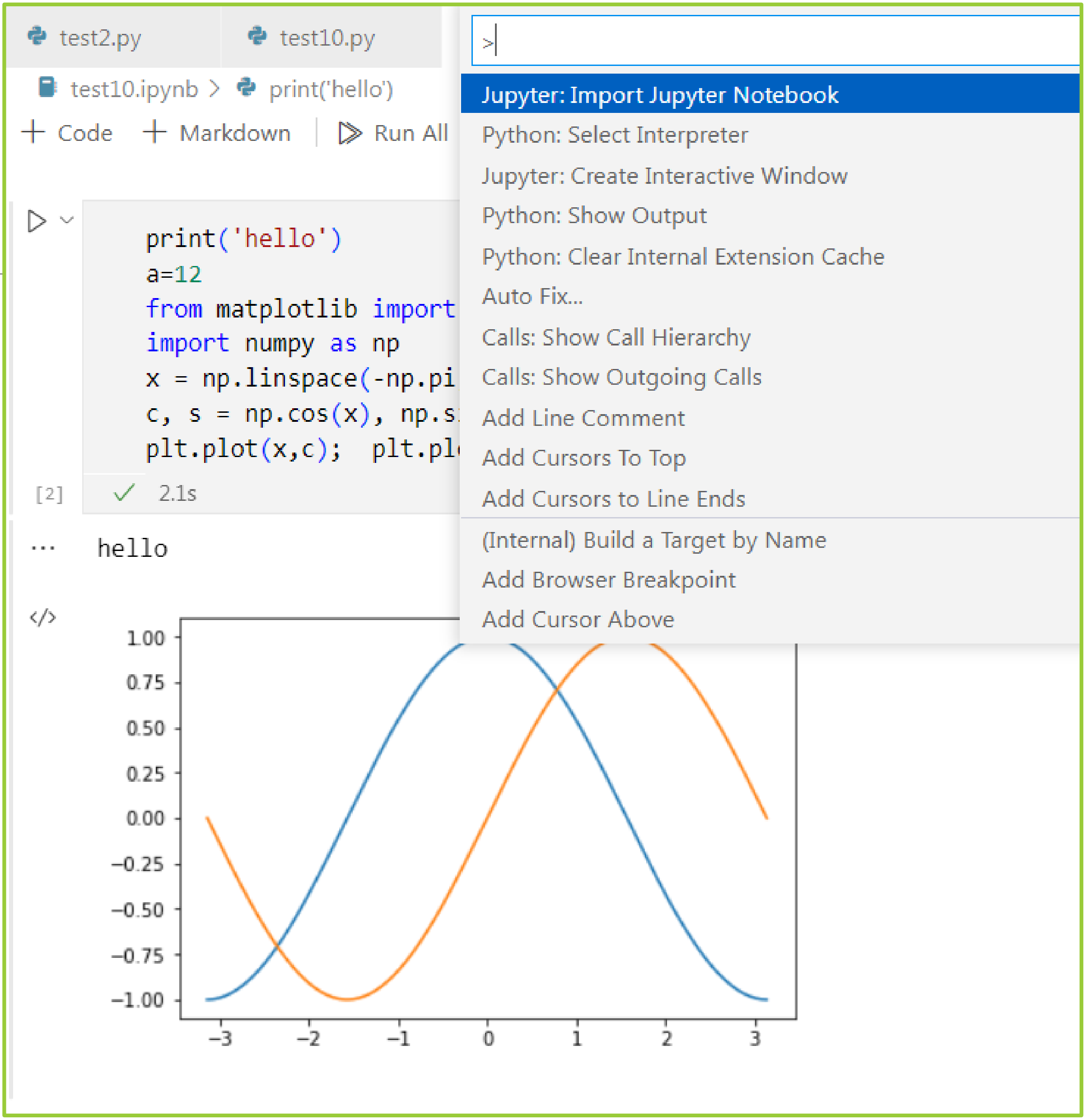
```
pr
```

```python
print('Hello')
```

```
Hello
```

https://code.visualstudio.com/docs/python/python-tutorial

Integrated Development Environment (IDE)



test2.py
test10.py
test10.ipynb > print('hello')
+ Code
+ Markdown
Run All
>
Jupyter: Import Jupyter Notebook
Python: Select Interpreter
Jupyter: Create Interactive Window
Python: Show Output
Python: Clear Internal Extension Cache
Auto Fix...
Calls: Show Call Hierarchy
Calls: Show Outgoing Calls
Add Line Comment
Add Cursors To Top
Add Cursors to Line Ends
(Internal) Build a Target by Name
Add Browser Breakpoint
Add Cursor Above
[2]
✓ 2.1s
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−3
−2
−1
0
1
2
3

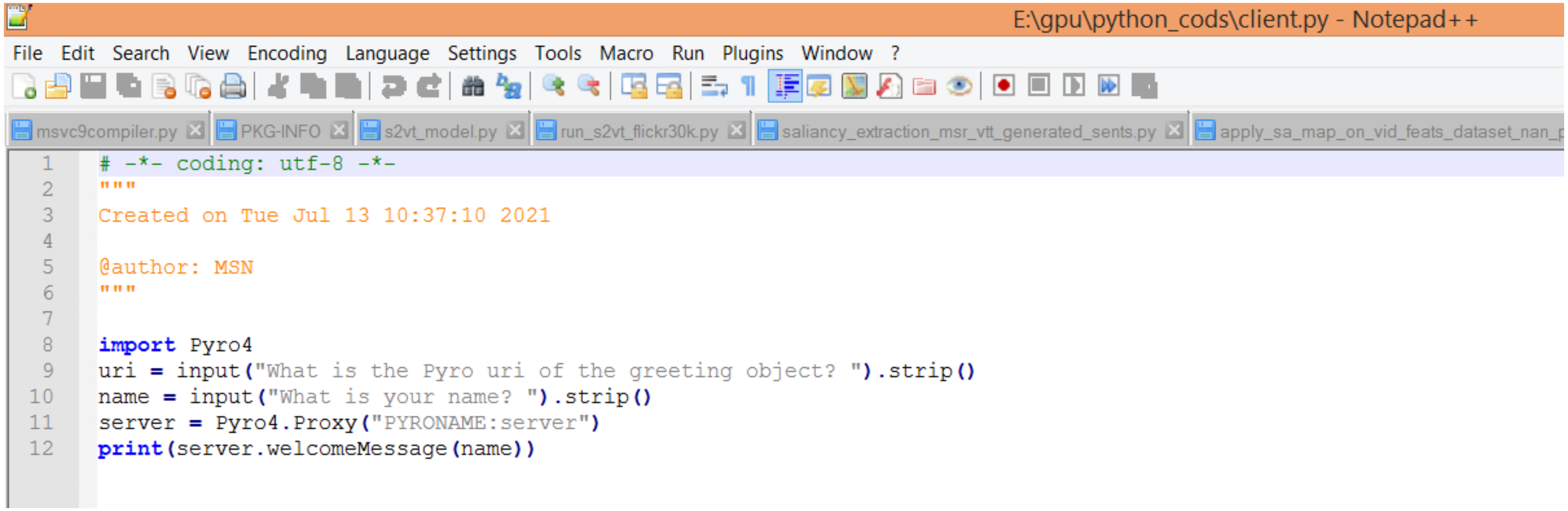
```
print('hello')
a=12
from matplotlib import
import numpy as np
x = np.linspace(-np.pi
c, s = np.cos(x), np.s
plt.plot(x,c);  plt.pl
```

```
hello
```

Text and source code editor

- **Notepad++:**

  Notepad++ is a text and source code editor for use with Microsoft Windows.


E:\gpu\python_cods\client.py - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
msvc9compiler.py
PKG-INFO
s2vt_model.py
run_s2vt_flickr30k.py
saliancy_extraction_msr_vtt_generated_sents.py
apply_sa_map_on_vid_feats_dataset_nan_p

```
# -*- coding: utf-8 -*-
"""
Created on Tue Jul 13 10:37:10 2021

@author: MSN
"""

import Pyro4
uri = input("What is the Pyro uri of the greeting object? ").strip()
name = input("What is your name? ").strip()
server = Pyro4.Proxy("PYRONAME:server")
print(server.welcomeMessage(name))
```

Getting started with Python for science

1. The interactive workflow: IPython and a text editor
    1. Command line interaction
    2. Elaboration of the algorithm in an editor
2. First steps
3. Basic **types**
    1. Numerical types
    2. Containers
    3. Assignment operator
4. **Control** Flow
    1. if/elif/else
    2. for/range
    3. while/break/continue
    4. Conditional Expressions
    5. Advanced iteration
    6. List Comprehensions

SciKits
Numpy
Matplotlib
SciPy
Python
2015
EDITION
Cython
IPython
Scipy
Lecture Notes
www.scipy-lectures.org
Edited by
Gaël Varoquaux
Emmanuelle Gouillart
Olaf Vahtras
creative commons
Gaël Varoquaux • Emmanuelle Gouillart • Olav Vahtras
Valentin Haenel • Nicolas P. Rougier • Ralf Gommers
Fabian Pedregosa • Zbigniew Jędrzejewski-Szmek • Pauli Virtanen
Christophe Combelles • Didrik Pinte • Robert Cimrman
André Espaze • Adrian Chauve • Christopher Burns

Getting started with Python for science

5. Defining **functions**
   1. Function definition
   2. Return statement
   3. Parameters
   4. Passing by value
   5. Global variables
   6. Variable number of parameters

6. Reusing code: **scripts** and **modules**
   1. Scripts
   2. Importing objects from modules
   3. Creating modules
   4. ‘__main__’ and module loading
   5. Scripts or modules? How to organize your code
   6. Packages
   7. Good practices

Getting started with Python for science

7. **Input** and **Output**
    1. Iterating over a file

8. Standard **Library**
    1. os module: operating system functionality
    2. shutil: high-level file operations
    3. glob: Pattern matching on files
    4. sys module: system-specific information
    5. pickle: easy persistence

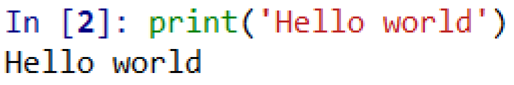
9. Exception handling in Python
    1. Exceptions
    2. Catching exceptions

10. Object-oriented programming (OOP)

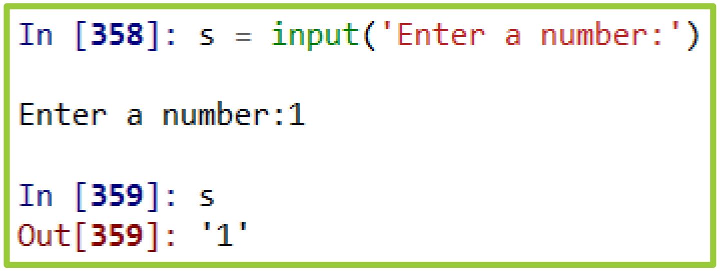
Getting started with Python for science

1. Command line interaction

IPython console in Spyder:



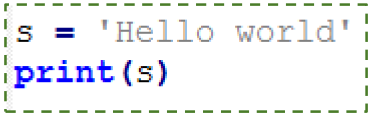
```
In [2]: print('Hello world')
Hello world
```



```
In [358]: s = input('Enter a number:')

Enter a number:1

In [359]: s
Out[359]: '1'
```

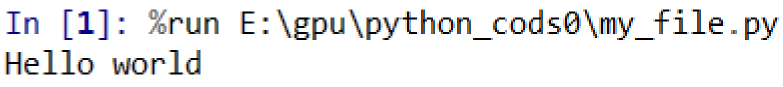
2. Elaboration of the algorithm in an editor

my_file.py →



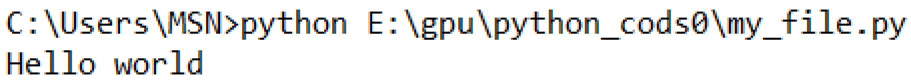
```
s = 'Hello world'
print(s)
```

1 IPython console:



```
In [1]: %run E:\gpu\python_cods0\my_file.py
Hello world
```

2 Command Prompt:

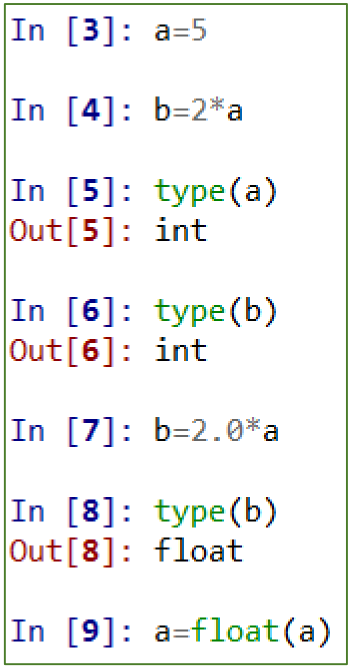


```
C:\Users\MSN>python E:\gpu\python_cods0\my_file.py
Hello world
```

Getting started with Python for science

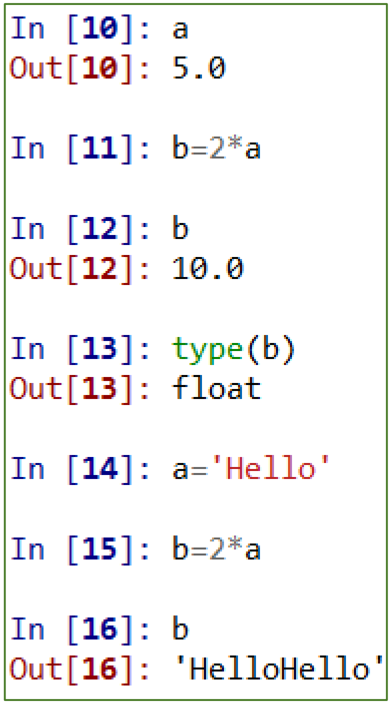
3. First steps

1



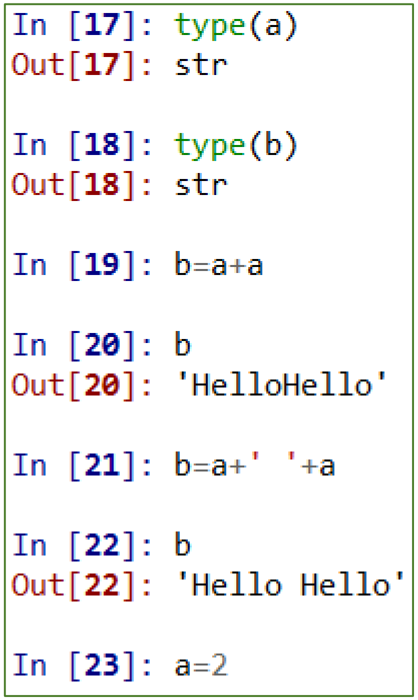
```
In [3]: a=5

In [4]: b=2*a

In [5]: type(a)
Out[5]: int

In [6]: type(b)
Out[6]: int

In [7]: b=2.0*a

In [8]: type(b)
Out[8]: float

In [9]: a=float(a)
```

2



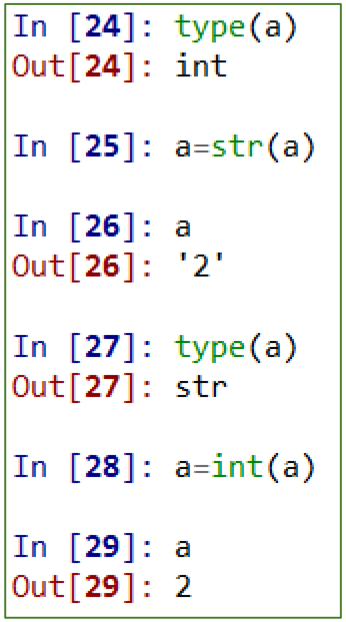
```
In [10]: a
Out[10]: 5.0

In [11]: b=2*a

In [12]: b
Out[12]: 10.0

In [13]: type(b)
Out[13]: float

In [14]: a='Hello'

In [15]: b=2*a

In [16]: b
Out[16]: 'HelloHello'
```

3



```
In [17]: type(a)
Out[17]: str

In [18]: type(b)
Out[18]: str

In [19]: b=a+a

In [20]: b
Out[20]: 'HelloHello'

In [21]: b=a+' '+a

In [22]: b
Out[22]: 'Hello Hello'

In [23]: a=2
```

4

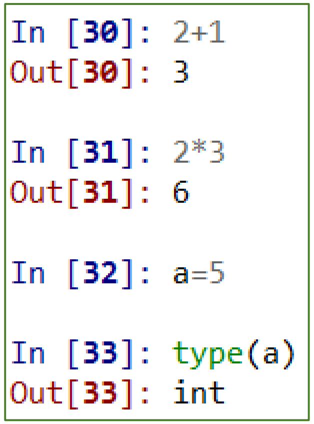


```
In [24]: type(a)
Out[24]: int

In [25]: a=str(a)

In [26]: a
Out[26]: '2'

In [27]: type(a)
Out[27]: str

In [28]: a=int(a)

In [29]: a
Out[29]: 2
```

Getting started with Python for science

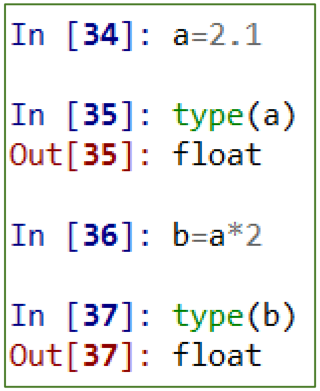
4. Basic types
   1. Numerical types
      - Integer
      - Floats
      - Complex
      - Booleans

Integer



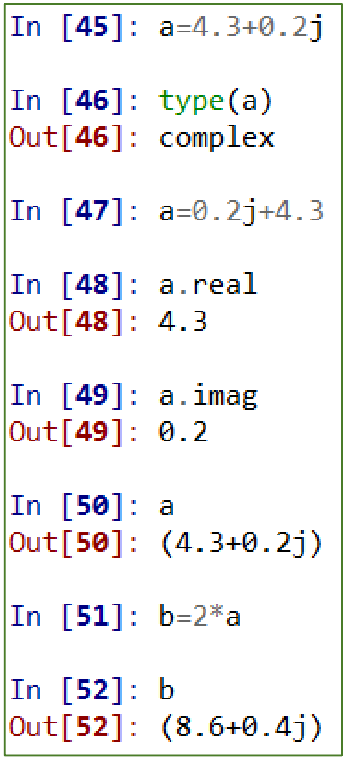
```
In [30]: 2+1
Out[30]: 3

In [31]: 2*3
Out[31]: 6

In [32]: a=5

In [33]: type(a)
Out[33]: int
```

Floats



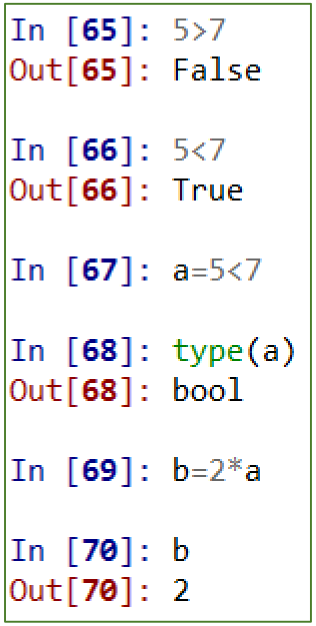
```
In [34]: a=2.1

In [35]: type(a)
Out[35]: float

In [36]: b=a*2

In [37]: type(b)
Out[37]: float
```

Complex

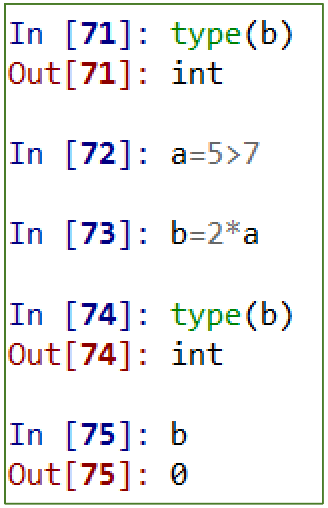


```
In [45]: a=4.3+0.2j

In [46]: type(a)
Out[46]: complex

In [47]: a=0.2j+4.3

In [48]: a.real
Out[48]: 4.3

In [49]: a.imag
Out[49]: 0.2

In [50]: a
Out[50]: (4.3+0.2j)

In [51]: b=2*a

In [52]: b
Out[52]: (8.6+0.4j)
```

Booleans



```
In [65]: 5>7
Out[65]: False

In [66]: 5<7
Out[66]: True

In [67]: a=5<7

In [68]: type(a)
Out[68]: bool

In [69]: b=2*a

In [70]: b
Out[70]: 2
```

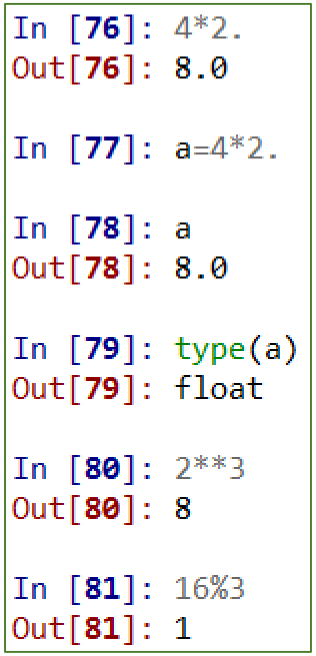


```
In [71]: type(b)
Out[71]: int

In [72]: a=5>7

In [73]: b=2*a

In [74]: type(b)
Out[74]: int

In [75]: b
Out[75]: 0
```

Getting started with Python for science

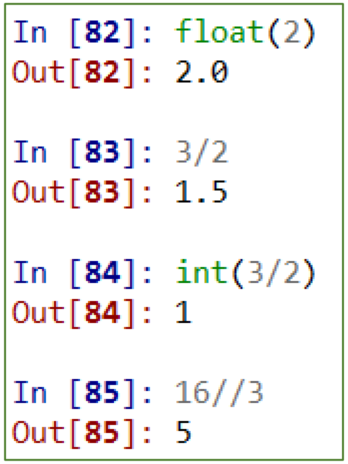
- Basic arithmetic operations +, -, *, /, %

1



```
In [76]: 4*2.
Out[76]: 8.0

In [77]: a=4*2.

In [78]: a
Out[78]: 8.0

In [79]: type(a)
Out[79]: float

In [80]: 2**3
Out[80]: 8

In [81]: 16%3
Out[81]: 1
```

2

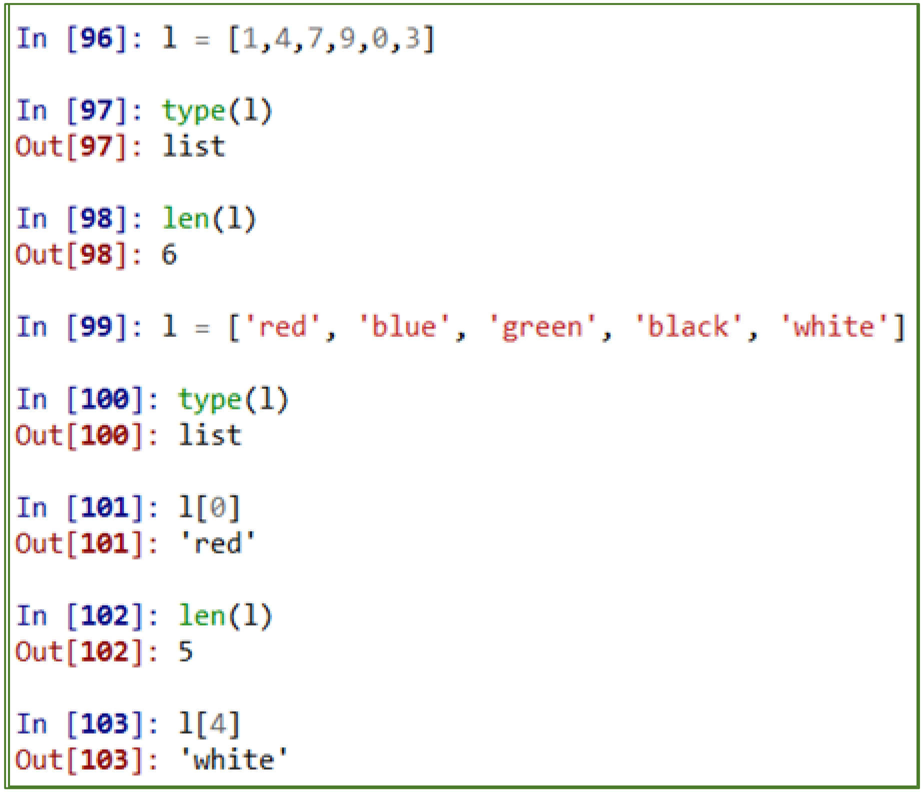


```
In [82]: float(2)
Out[82]: 2.0

In [83]: 3/2
Out[83]: 1.5

In [84]: int(3/2)
Out[84]: 1

In [85]: 16//3
Out[85]: 5
```

Getting started with Python for science

3. Basic types
   2. Containers
      - Lists
      - Strings
      - Dictionaries
      - Tuples
      - Sets

Lists

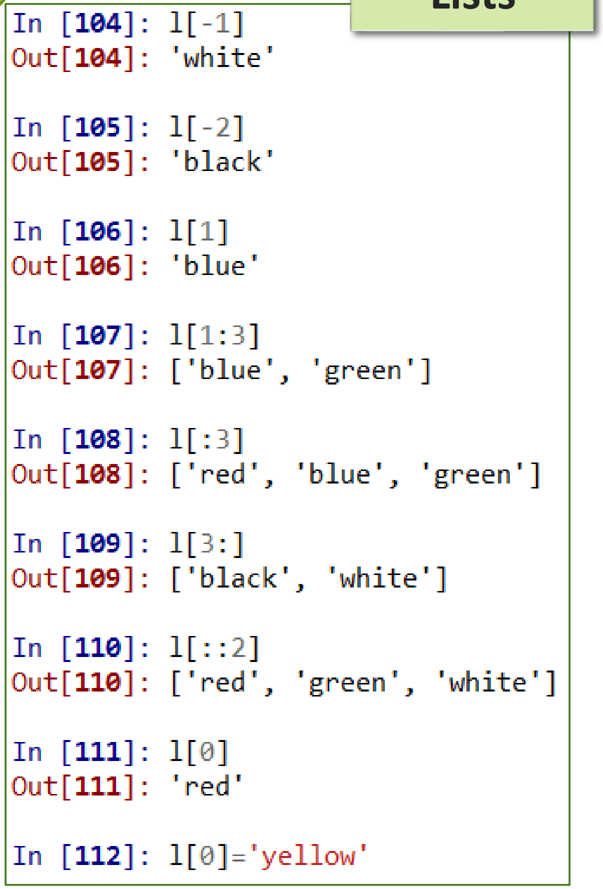


```
In [96]: l = [1,4,7,9,0,3]

In [97]: type(l)
Out[97]: list

In [98]: len(l)
Out[98]: 6

In [99]: l = ['red', 'blue', 'green', 'black', 'white']

In [100]: type(l)
Out[100]: list

In [101]: l[0]
Out[101]: 'red'

In [102]: len(l)
Out[102]: 5

In [103]: l[4]
Out[103]: 'white'
```

Getting started with Python for science

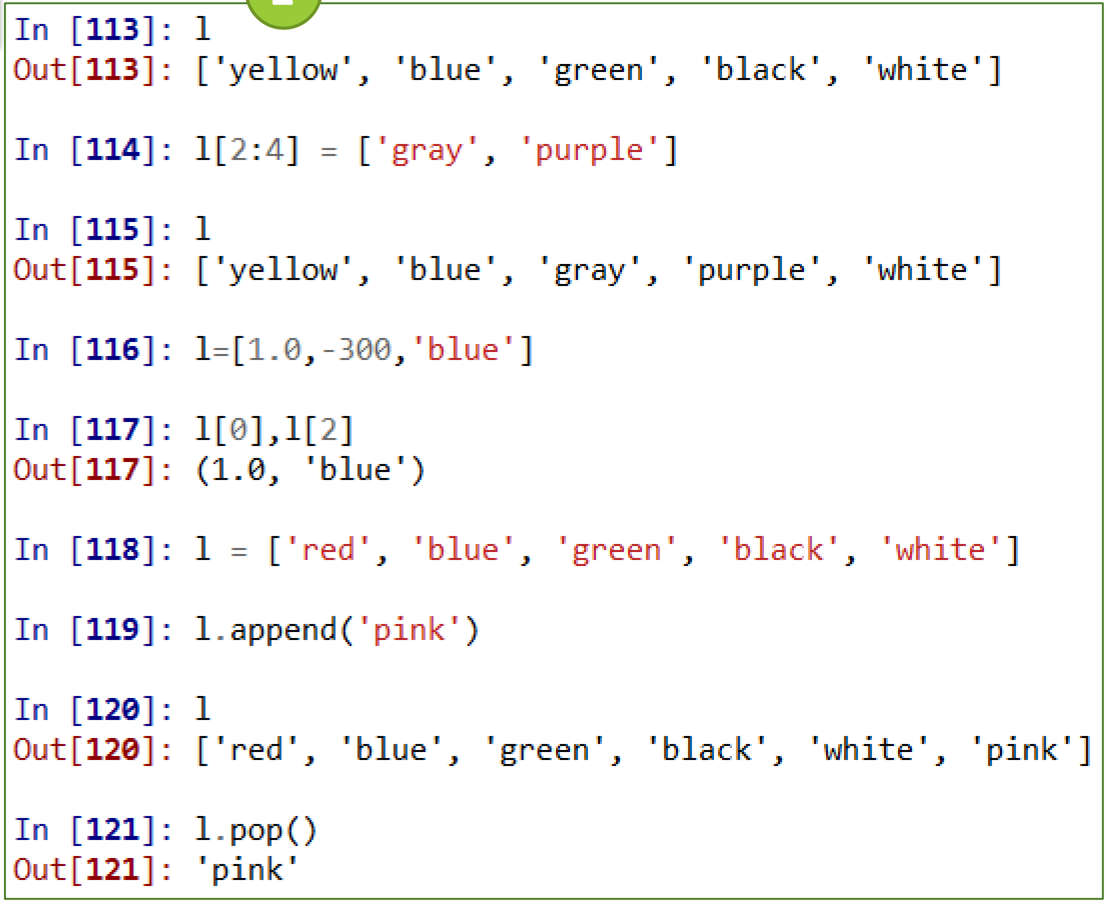
1

Lists



```
In [104]: l[-1]
Out[104]: 'white'

In [105]: l[-2]
Out[105]: 'black'

In [106]: l[1]
Out[106]: 'blue'

In [107]: l[1:3]
Out[107]: ['blue', 'green']

In [108]: l[:3]
Out[108]: ['red', 'blue', 'green']

In [109]: l[3:]
Out[109]: ['black', 'white']

In [110]: l[::2]
Out[110]: ['red', 'green', 'white']

In [111]: l[0]
Out[111]: 'red'

In [112]: l[0]='yellow'
```

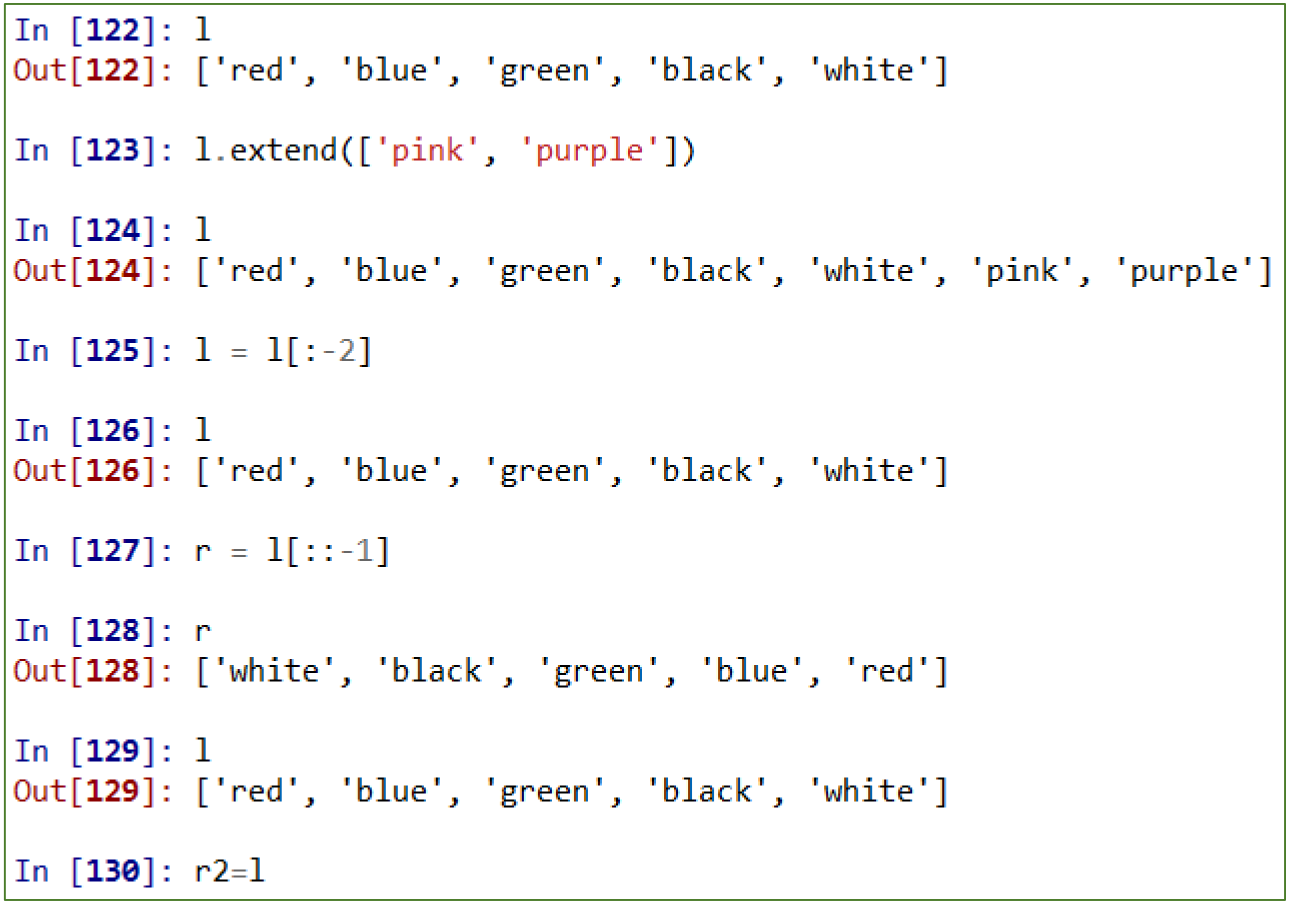
2



```
In [113]: l
Out[113]: ['yellow', 'blue', 'green', 'black', 'white']

In [114]: l[2:4] = ['gray', 'purple']

In [115]: l
Out[115]: ['yellow', 'blue', 'gray', 'purple', 'white']

In [116]: l=[1.0,-300,'blue']

In [117]: l[0],l[2]
Out[117]: (1.0, 'blue')

In [118]: l = ['red', 'blue', 'green', 'black', 'white']

In [119]: l.append('pink')

In [120]: l
Out[120]: ['red', 'blue', 'green', 'black', 'white', 'pink']

In [121]: l.pop()
Out[121]: 'pink'
```

Getting started with Python for science

Lists

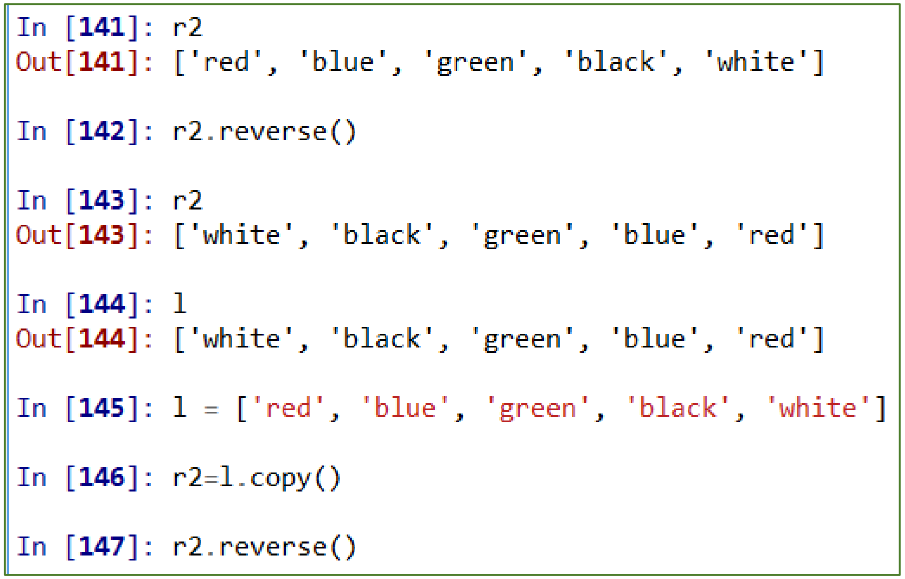


```
In [122]: l
Out[122]: ['red', 'blue', 'green', 'black', 'white']

In [123]: l.extend(['pink', 'purple'])

In [124]: l
Out[124]: ['red', 'blue', 'green', 'black', 'white', 'pink', 'purple']

In [125]: l = l[:-2]

In [126]: l
Out[126]: ['red', 'blue', 'green', 'black', 'white']

In [127]: r = l[::-1]

In [128]: r
Out[128]: ['white', 'black', 'green', 'blue', 'red']

In [129]: l
Out[129]: ['red', 'blue', 'green', 'black', 'white']

In [130]: r2=l
```

Getting started with Python for science

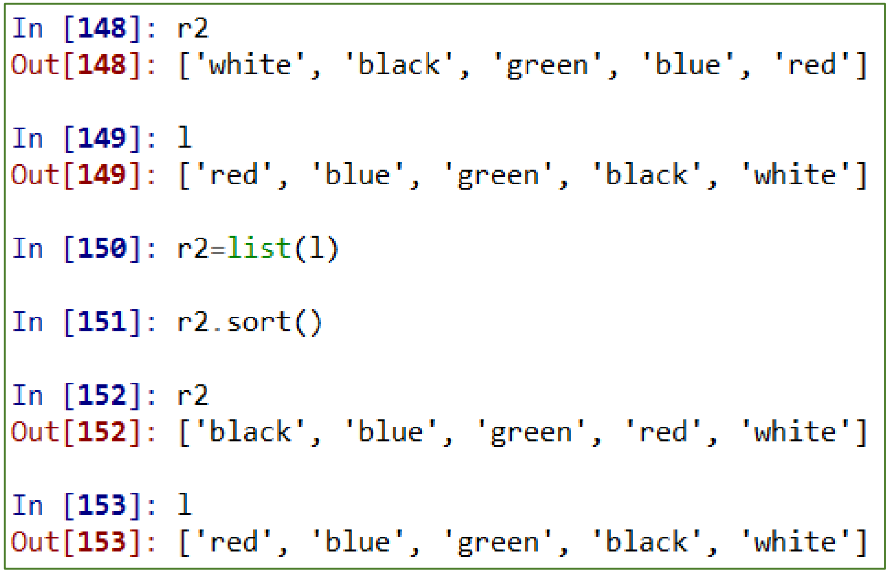
Lists

1



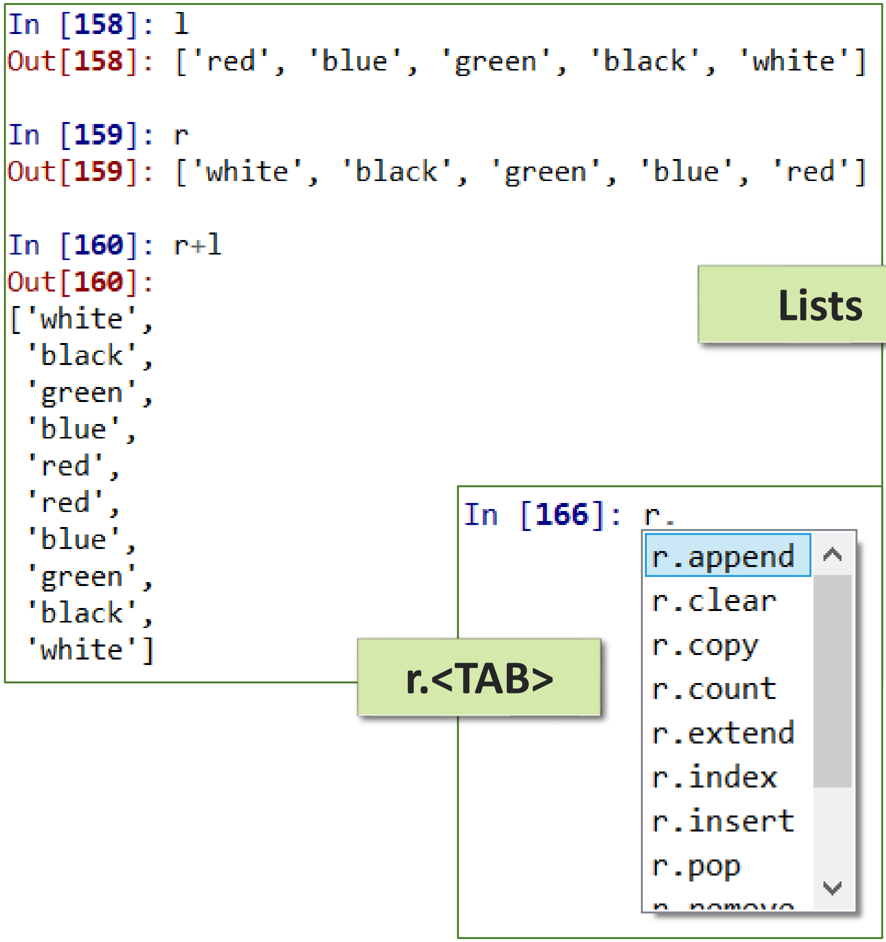
```
In [141]: r2
Out[141]: ['red', 'blue', 'green', 'black', 'white']

In [142]: r2.reverse()

In [143]: r2
Out[143]: ['white', 'black', 'green', 'blue', 'red']

In [144]: l
Out[144]: ['white', 'black', 'green', 'blue', 'red']

In [145]: l = ['red', 'blue', 'green', 'black', 'white']

In [146]: r2=l.copy()

In [147]: r2.reverse()
```

2



```
In [148]: r2
Out[148]: ['white', 'black', 'green', 'blue', 'red']

In [149]: l
Out[149]: ['red', 'blue', 'green', 'black', 'white']

In [150]: r2=list(l)

In [151]: r2.sort()

In [152]: r2
Out[152]: ['black', 'blue', 'green', 'red', 'white']

In [153]: l
Out[153]: ['red', 'blue', 'green', 'black', 'white']
```

Getting started with Python for science

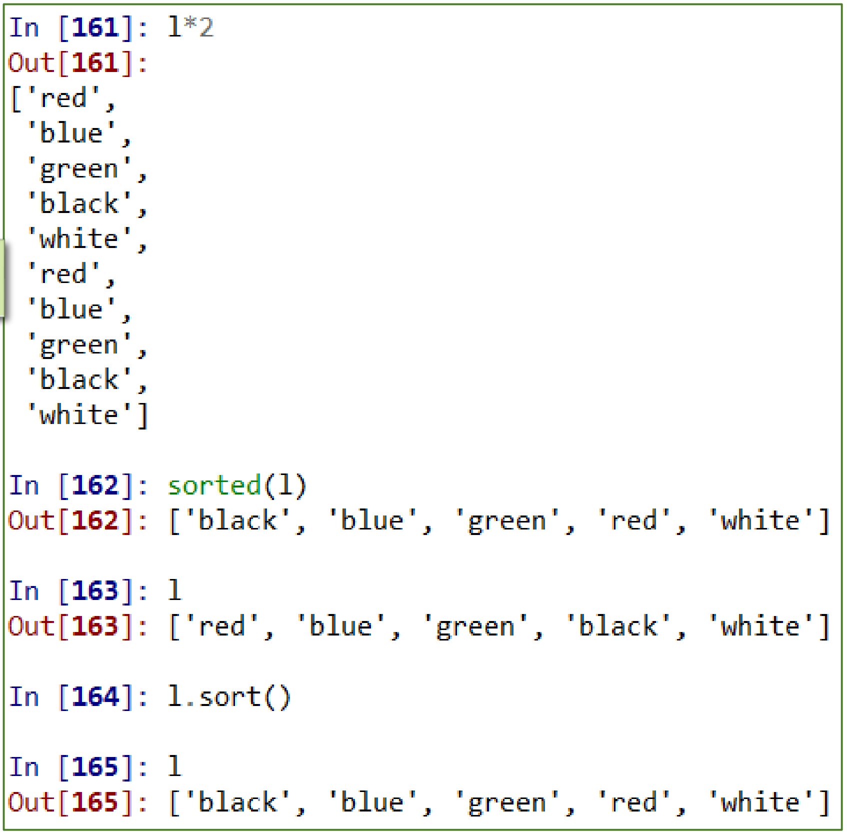


```
In [158]: l
Out[158]: ['red', 'blue', 'green', 'black', 'white']

In [159]: r
Out[159]: ['white', 'black', 'green', 'blue', 'red']

In [160]: r+l
Out[160]:
['white',
 'black',
 'green',
 'blue',
 'red',
 'red',
 'blue',
 'green',
 'black',
 'white']
```

Lists

r.<TAB>

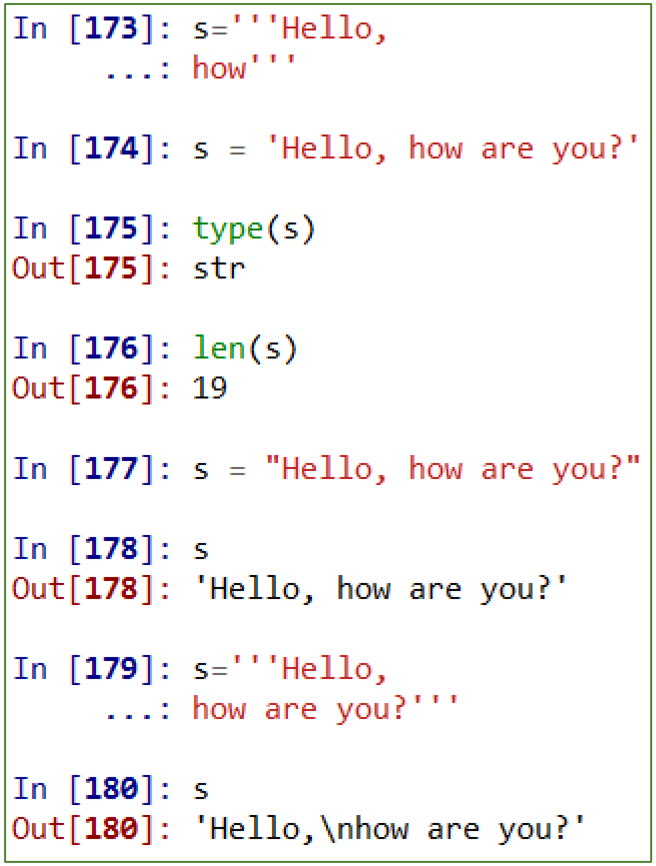
```
In [166]: r.
```



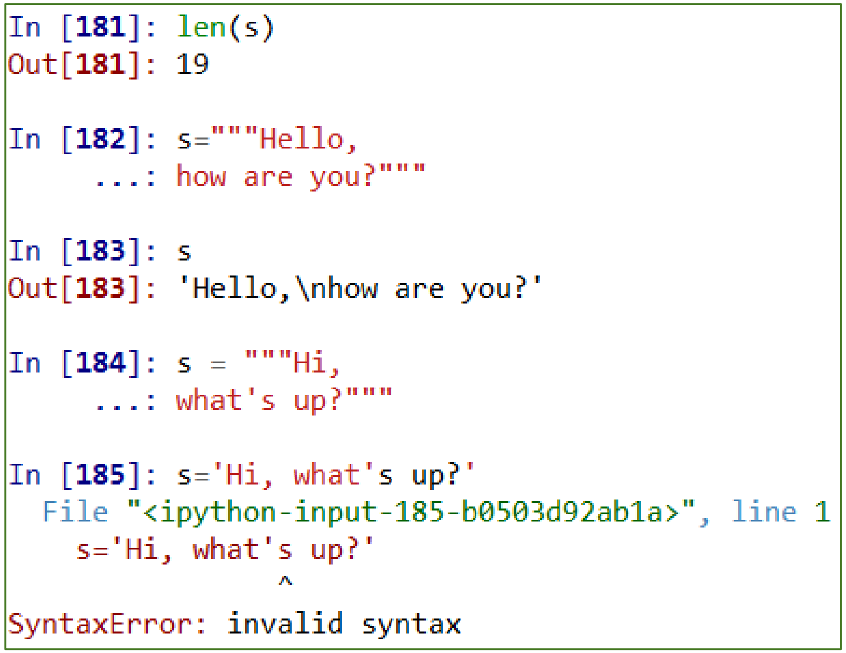
```
In [161]: l*2
Out[161]:
['red',
 'blue',
 'green',
 'black',
 'white',
 'red',
 'blue',
 'green',
 'black',
 'white']

In [162]: sorted(l)
Out[162]: ['black', 'blue', 'green', 'red', 'white']

In [163]: l
Out[163]: ['red', 'blue', 'green', 'black', 'white']

In [164]: l.sort()

In [165]: l
Out[165]: ['black', 'blue', 'green', 'red', 'white']
```

Getting started with Python for science

Strings

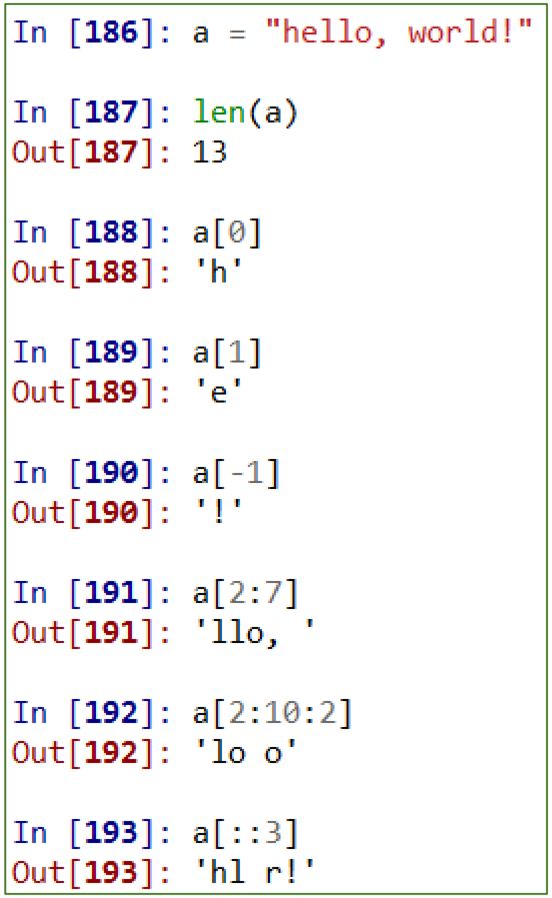


```
In [173]: s='''Hello,
     ...: how'''

In [174]: s = 'Hello, how are you?'

In [175]: type(s)
Out[175]: str

In [176]: len(s)
Out[176]: 19

In [177]: s = "Hello, how are you?"

In [178]: s
Out[178]: 'Hello, how are you?'

In [179]: s='''Hello,
     ...: how are you?'''

In [180]: s
Out[180]: 'Hello,\nhow are you?'
```



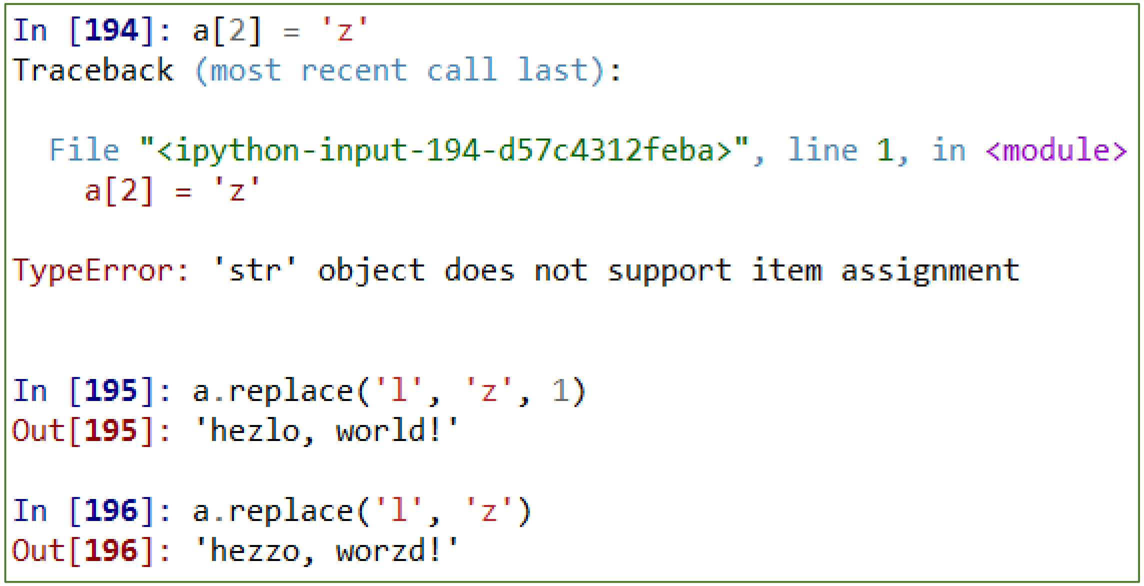
```
In [181]: len(s)
Out[181]: 19

In [182]: s="""Hello,
     ...: how are you?"""

In [183]: s
Out[183]: 'Hello,\nhow are you?'

In [184]: s = """Hi,
     ...: what's up?"""

In [185]: s='Hi, what's up?'
  File "<ipython-input-185-b0503d92ab1a>", line 1
    s='Hi, what's up?'
                ^
SyntaxError: invalid syntax
```

Getting started with Python for science



```
In [186]: a = "hello, world!"

In [187]: len(a)
Out[187]: 13

In [188]: a[0]
Out[188]: 'h'

In [189]: a[1]
Out[189]: 'e'

In [190]: a[-1]
Out[190]: '!'

In [191]: a[2:7]
Out[191]: 'llo, '

In [192]: a[2:10:2]
Out[192]: 'lo o'

In [193]: a[::3]
Out[193]: 'hl r!'
```

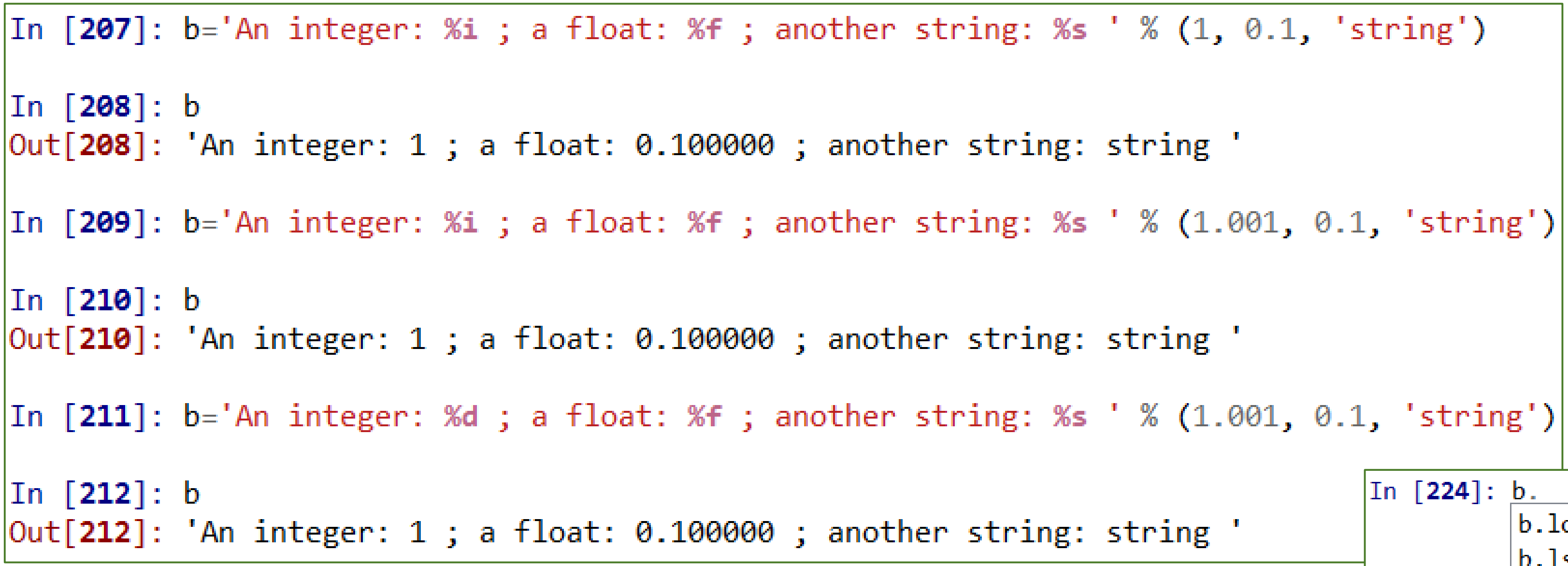
Strings



```
In [194]: a[2] = 'z'
Traceback (most recent call last):

  File "<ipython-input-194-d57c4312feba>", line 1, in <module>
    a[2] = 'z'

TypeError: 'str' object does not support item assignment


In [195]: a.replace('l', 'z', 1)
Out[195]: 'hezlo, world!'

In [196]: a.replace('l', 'z')
Out[196]: 'hezzo, worzd!'
```

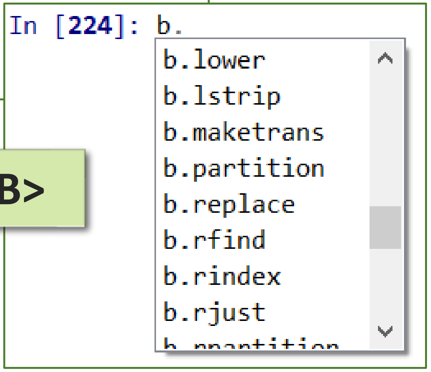
Getting started with Python for science

Strings



```
In [207]: b='An integer: %i ; a float: %f ; another string: %s ' % (1, 0.1, 'string')

In [208]: b
Out[208]: 'An integer: 1 ; a float: 0.100000 ; another string: string '

In [209]: b='An integer: %i ; a float: %f ; another string: %s ' % (1.001, 0.1, 'string')

In [210]: b
Out[210]: 'An integer: 1 ; a float: 0.100000 ; another string: string '

In [211]: b='An integer: %d ; a float: %f ; another string: %s ' % (1.001, 0.1, 'string')

In [212]: b
Out[212]: 'An integer: 1 ; a float: 0.100000 ; another string: string '
```

b.<TAB>



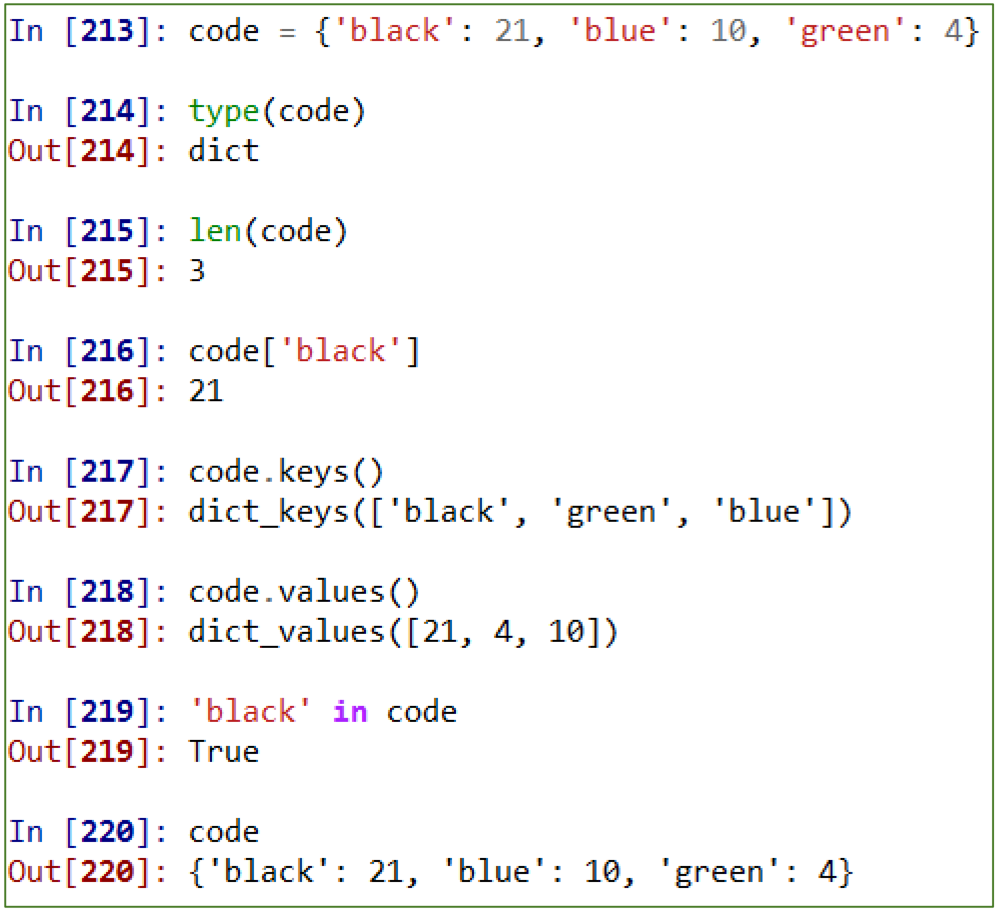
```
In [224]: b.
            b.lower
            b.lstrip
            b.maketrans
            b.partition
            b.replace
            b.rfind
            b.rindex
            b.rjust
            b.rpartition
```

https://zetcode.com/python/fstring/

https://stackoverflow.com/questions/45310254/fixed-digits-after-decimal-with-f-strings

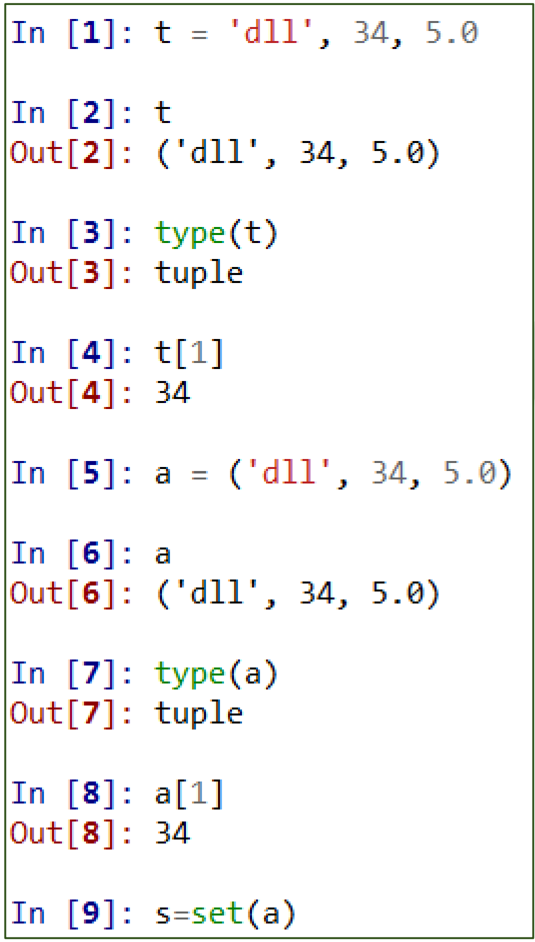
Getting started with Python for science

Dictionaries



```
In [213]: code = {'black': 21, 'blue': 10, 'green': 4}

In [214]: type(code)
Out[214]: dict

In [215]: len(code)
Out[215]: 3

In [216]: code['black']
Out[216]: 21

In [217]: code.keys()
Out[217]: dict_keys(['black', 'green', 'blue'])

In [218]: code.values()
Out[218]: dict_values([21, 4, 10])

In [219]: 'black' in code
Out[219]: True

In [220]: code
Out[220]: {'black': 21, 'blue': 10, 'green': 4}
```

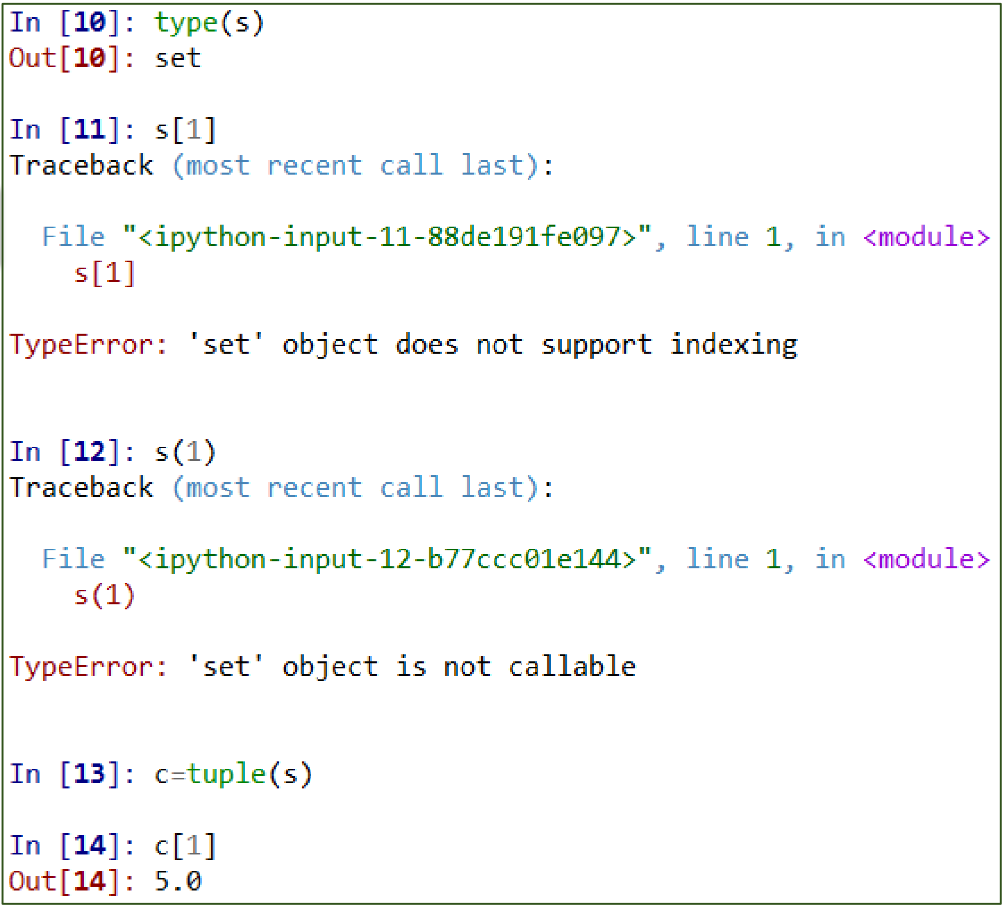
Getting started with Python for science



```
In [1]: t = 'dll', 34, 5.0

In [2]: t
Out[2]: ('dll', 34, 5.0)

In [3]: type(t)
Out[3]: tuple

In [4]: t[1]
Out[4]: 34

In [5]: a = ('dll', 34, 5.0)

In [6]: a
Out[6]: ('dll', 34, 5.0)

In [7]: type(a)
Out[7]: tuple

In [8]: a[1]
Out[8]: 34

In [9]: s=set(a)
```

Tuples

Sets

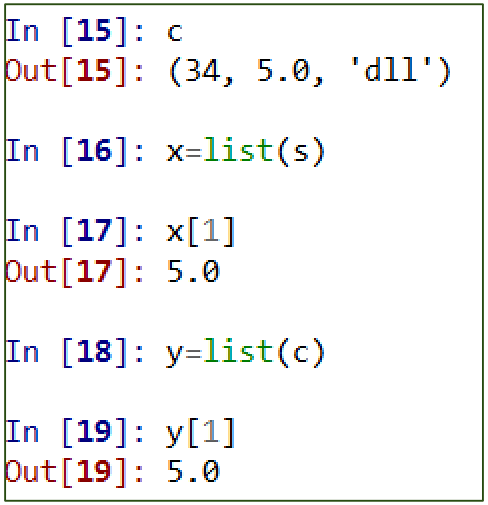


```
In [10]: type(s)
Out[10]: set

In [11]: s[1]
Traceback (most recent call last):

  File "<ipython-input-11-88de191fe097>", line 1, in <module>
    s[1]

TypeError: 'set' object does not support indexing


In [12]: s(1)
Traceback (most recent call last):

  File "<ipython-input-12-b77ccc01e144>", line 1, in <module>
    s(1)

TypeError: 'set' object is not callable


In [13]: c=tuple(s)

In [14]: c[1]
Out[14]: 5.0
```

Getting started with Python for science

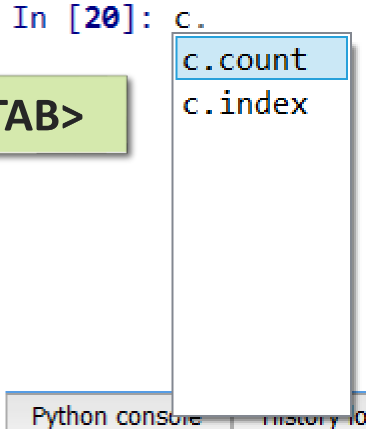
Tuples

Sets



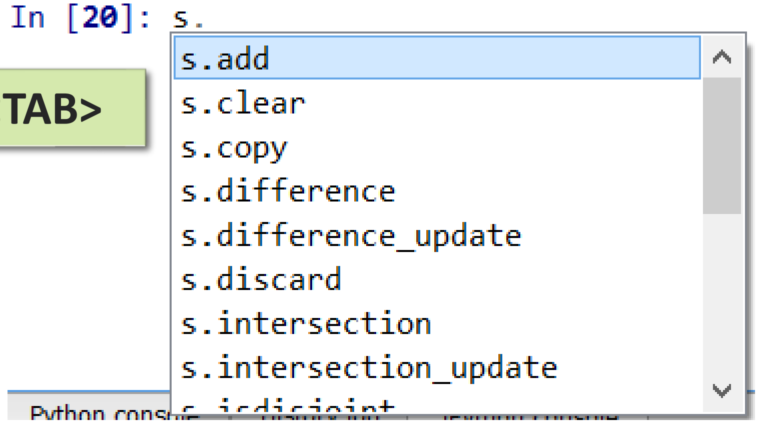
```
In [15]: c
Out[15]: (34, 5.0, 'dll')

In [16]: x=list(s)

In [17]: x[1]
Out[17]: 5.0

In [18]: y=list(c)

In [19]: y[1]
Out[19]: 5.0
```

c.<TAB>



```
In [20]: c.
```

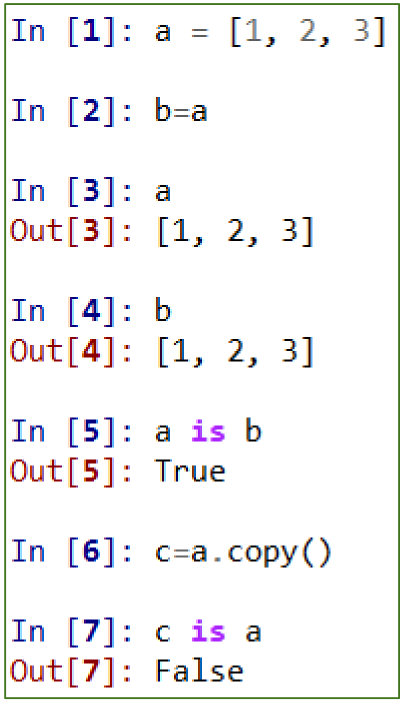
s.<TAB>



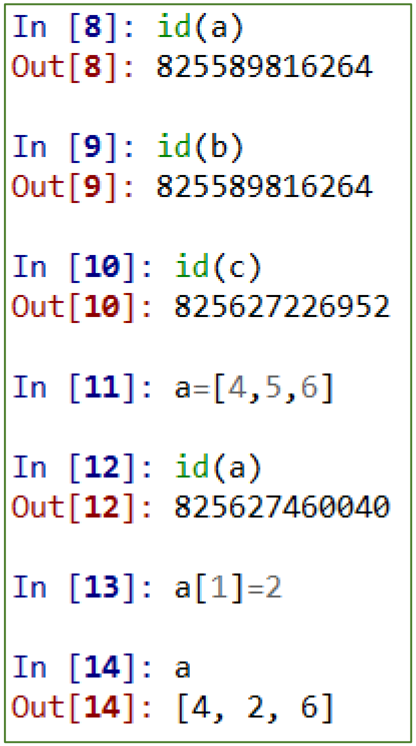
```
In [20]: s.
```

Getting started with Python for science

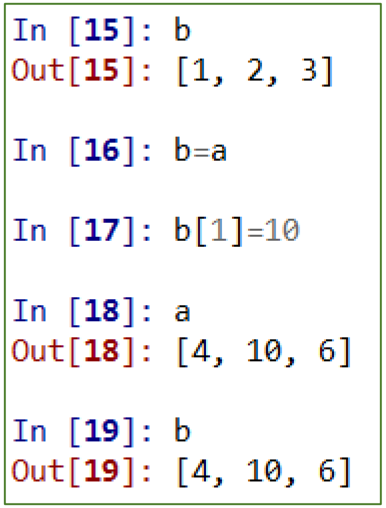
3. Basic types
   3. Assignment operator



```
In [1]: a = [1, 2, 3]

In [2]: b=a

In [3]: a
Out[3]: [1, 2, 3]

In [4]: b
Out[4]: [1, 2, 3]

In [5]: a is b
Out[5]: True

In [6]: c=a.copy()

In [7]: c is a
Out[7]: False
```



```
In [8]: id(a)
Out[8]: 825589816264

In [9]: id(b)
Out[9]: 825589816264

In [10]: id(c)
Out[10]: 825627226952

In [11]: a=[4,5,6]

In [12]: id(a)
Out[12]: 825627460040

In [13]: a[1]=2

In [14]: a
Out[14]: [4, 2, 6]
```



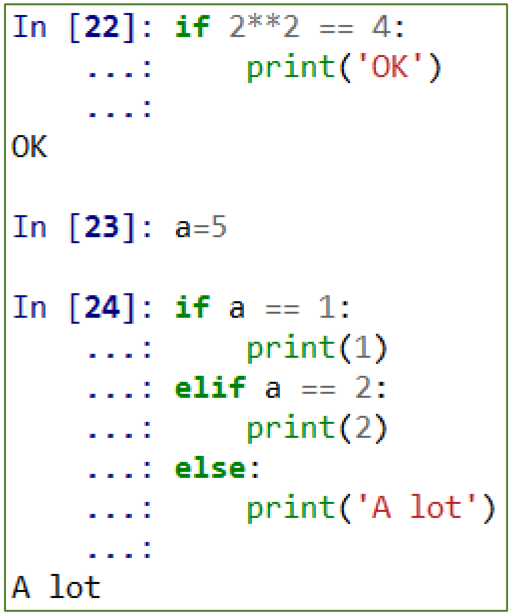
```
In [15]: b
Out[15]: [1, 2, 3]

In [16]: b=a

In [17]: b[1]=10

In [18]: a
Out[18]: [4, 10, 6]

In [19]: b
Out[19]: [4, 10, 6]
```

https://www.w3schools.com/python/ref_func_id.asp

Getting started with Python for science

4. Control Flow
   1. if/elif/else

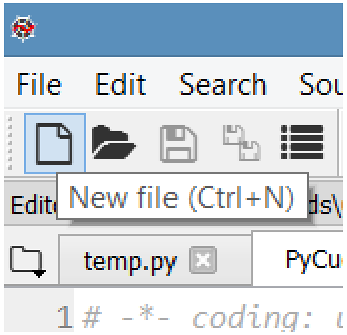
IPython console



```
In [22]: if 2**2 == 4:
    ...:     print('OK')
    ...:
OK

In [23]: a=5

In [24]: if a == 1:
    ...:     print(1)
    ...: elif a == 2:
    ...:     print(2)
    ...: else:
    ...:     print('A lot')
    ...:
A lot
```

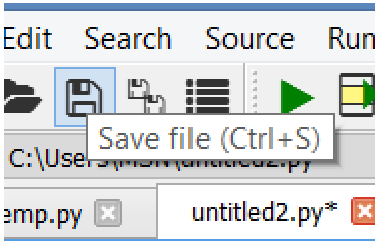
Script

1 **Create a new script in spyder**

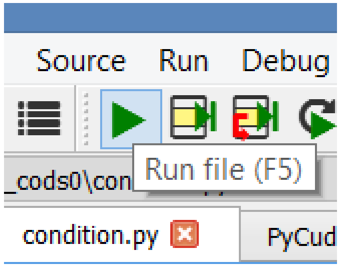


```
1 # -*- coding: 
```

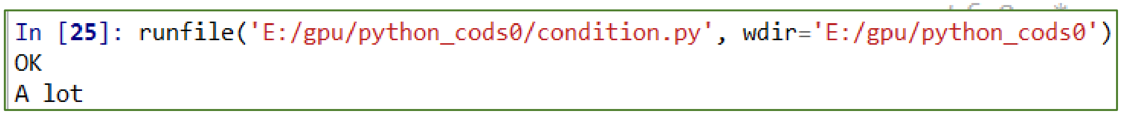
2 **Save as a py file**



3 **Run program**



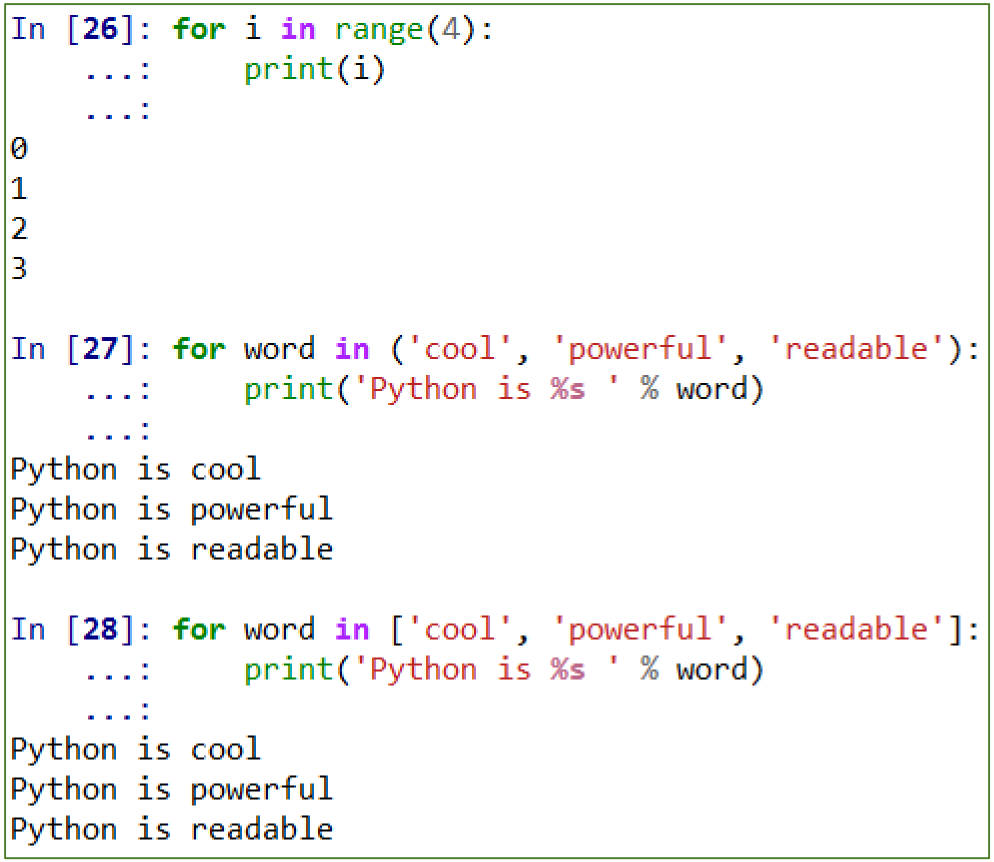
4 **Output in IPython consol**



```
In [25]: runfile('E:/gpu/python_cods0/condition.py', wdir='E:/gpu/python_cods0')
OK
A lot
```

Getting started with Python for science

4. Control Flow
   2. for/range



```
In [26]: for i in range(4):
    ...:     print(i)
    ...:
0
1
2
3

In [27]: for word in ('cool', 'powerful', 'readable'):
    ...:     print('Python is %s ' % word)
    ...:
Python is cool
Python is powerful
Python is readable

In [28]: for word in ['cool', 'powerful', 'readable']:
    ...:     print('Python is %s ' % word)
    ...:
Python is cool
Python is powerful
Python is readable
```

Getting started with Python for science

1. Write a program that takes input from the user involves a string containing ten integer numbers separated with comma, then follow these steps:

   Split numbers

   Convert them to int

   Print them

   Append them to a list

   Extract first and last numbers from the list and print them

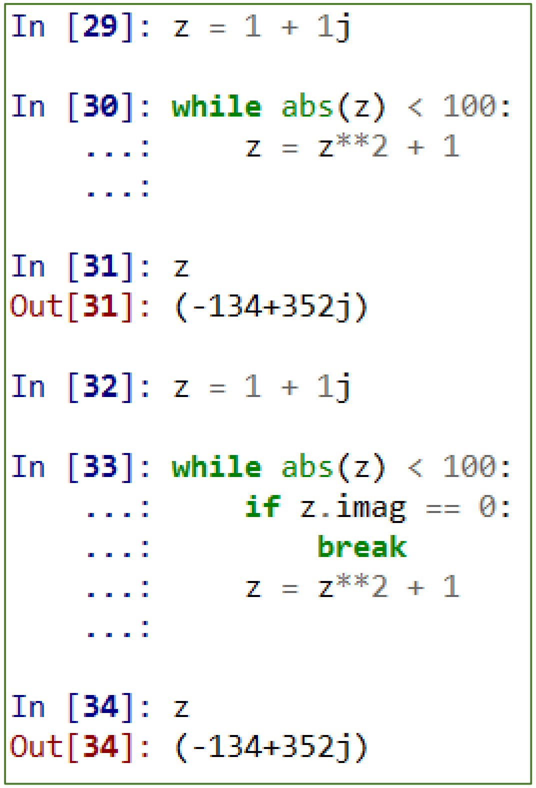
   Apply the average function on them and print the result

Getting started with Python for science

4. Control Flow

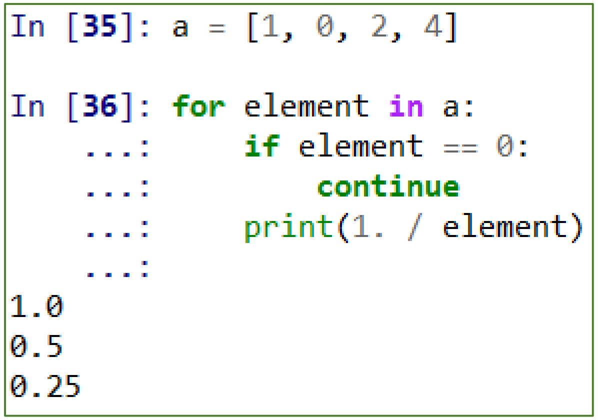
3. while/break/continue

1



```
In [29]: z = 1 + 1j

In [30]: while abs(z) < 100:
    ...:     z = z**2 + 1
    ...:

In [31]: z
Out[31]: (-134+352j)

In [32]: z = 1 + 1j

In [33]: while abs(z) < 100:
    ...:     if z.imag == 0:
    ...:         break
    ...:     z = z**2 + 1
    ...:

In [34]: z
Out[34]: (-134+352j)
```

2



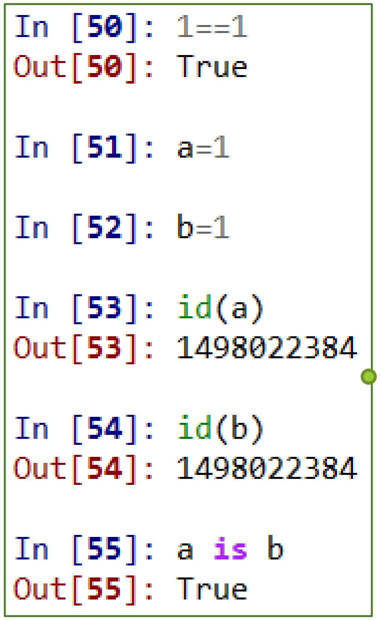
```
In [35]: a = [1, 0, 2, 4]

In [36]: for element in a:
    ...:     if element == 0:
    ...:         continue
    ...:     print(1. / element)
    ...:
1.0
0.5
0.25
```

Getting started with Python for science

4. Control Flow

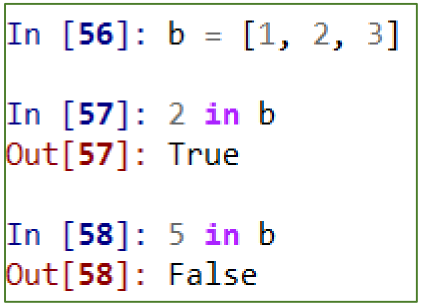
4. Conditional Expressions

1



```
In [50]: 1==1
Out[50]: True

In [51]: a=1

In [52]: b=1

In [53]: id(a)
Out[53]: 1498022384

In [54]: id(b)
Out[54]: 1498022384

In [55]: a is b
Out[55]: True
```

2



```
In [56]: b = [1, 2, 3]

In [57]: 2 in b
Out[57]: True

In [58]: 5 in b
Out[58]: False
```

Getting started with Python for science

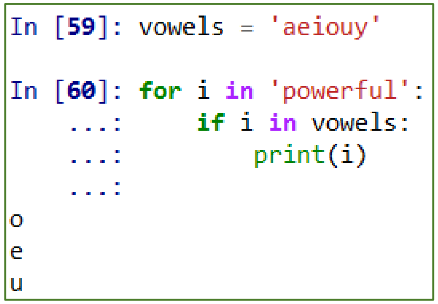
4. Control Flow
    5. Advanced iteration

Iterate over any sequence

You can iterate over any sequence (string, list, keys in a dictionary, lines in a file, ...):

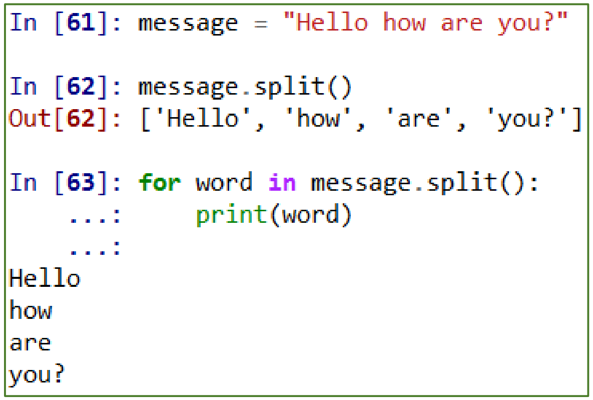
Split:

String



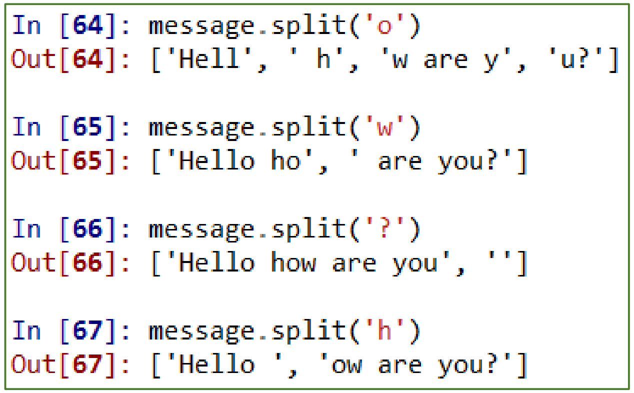
```
In [59]: vowels = 'aeiouy'

In [60]: for i in 'powerful':
    ...:     if i in vowels:
    ...:         print(i)
    ...:
o
e
u
```

1



```
In [61]: message = "Hello how are you?"

In [62]: message.split()
Out[62]: ['Hello', 'how', 'are', 'you?']

In [63]: for word in message.split():
    ...:     print(word)
    ...:
Hello
how
are
you?
```

2



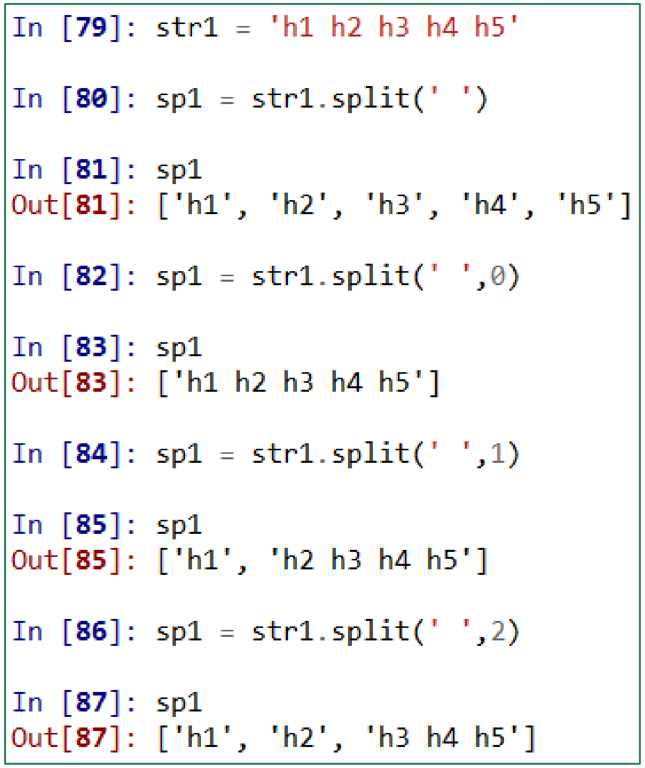
```
In [64]: message.split('o')
Out[64]: ['Hell', ' h', 'w are y', 'u?']

In [65]: message.split('w')
Out[65]: ['Hello ho', ' are you?']

In [66]: message.split('?')
Out[66]: ['Hello how are you', '']

In [67]: message.split('h')
Out[67]: ['Hello ', 'ow are you?']
```

Getting started with Python for science

Split:

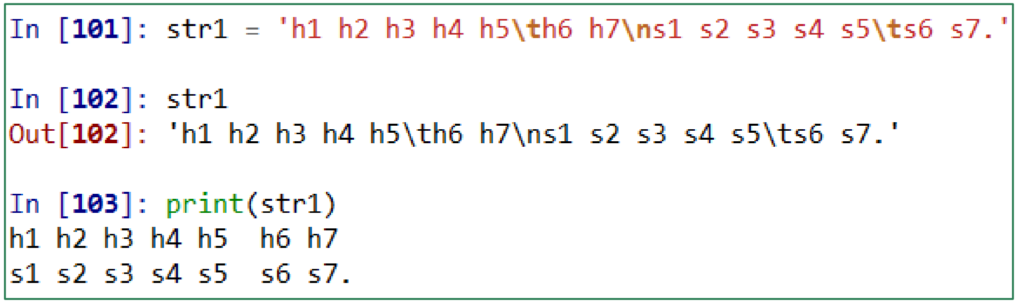
4. Control Flow
    5. Advanced iteration

3



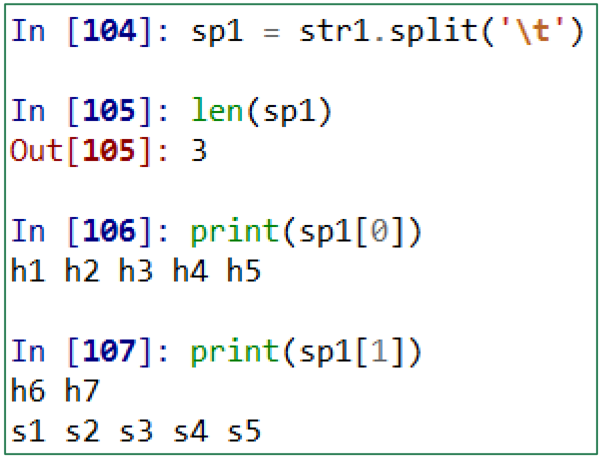
```
In [79]: str1 = 'h1 h2 h3 h4 h5'

In [80]: sp1 = str1.split(' ')

In [81]: sp1
Out[81]: ['h1', 'h2', 'h3', 'h4', 'h5']

In [82]: sp1 = str1.split(' ',0)

In [83]: sp1
Out[83]: ['h1 h2 h3 h4 h5']

In [84]: sp1 = str1.split(' ',1)

In [85]: sp1
Out[85]: ['h1', 'h2 h3 h4 h5']

In [86]: sp1 = str1.split(' ',2)

In [87]: sp1
Out[87]: ['h1', 'h2', 'h3 h4 h5']
```

4



```
In [101]: str1 = 'h1 h2 h3 h4 h5\th6 h7\ns1 s2 s3 s4 s5\ts6 s7.'

In [102]: str1
Out[102]: 'h1 h2 h3 h4 h5\th6 h7\ns1 s2 s3 s4 s5\ts6 s7.'

In [103]: print(str1)
h1 h2 h3 h4 h5  h6 h7
s1 s2 s3 s4 s5  s6 s7.
```

5



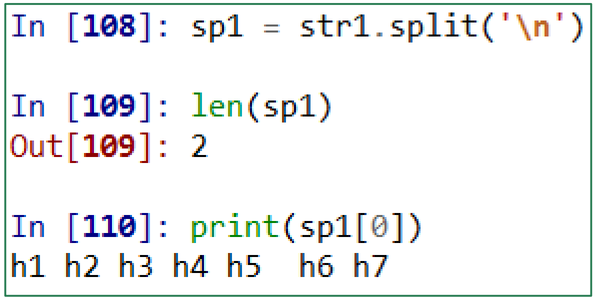
```
In [104]: sp1 = str1.split('\t')

In [105]: len(sp1)
Out[105]: 3

In [106]: print(sp1[0])
h1 h2 h3 h4 h5

In [107]: print(sp1[1])
h6 h7
s1 s2 s3 s4 s5
```

Getting started with Python for science

4. Control Flow
   5. Advanced iteration

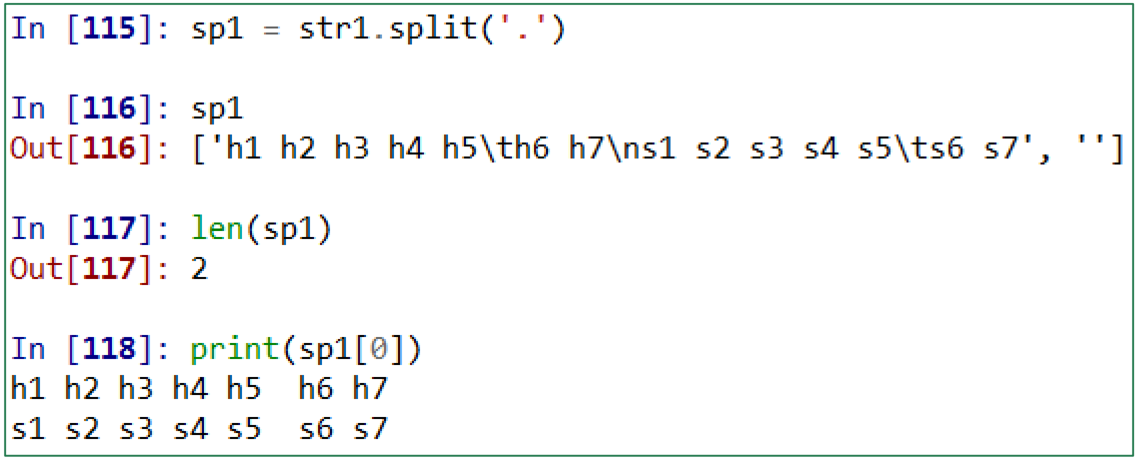
Split:

6



```
In [108]: sp1 = str1.split('\n')

In [109]: len(sp1)
Out[109]: 2

In [110]: print(sp1[0])
h1 h2 h3 h4 h5  h6 h7
```

7



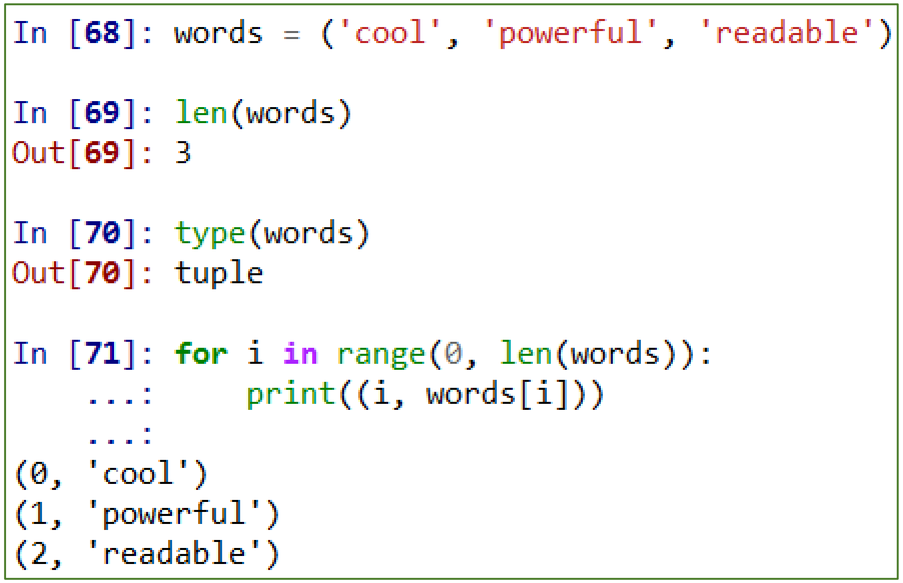
```
In [115]: sp1 = str1.split('.')

In [116]: sp1
Out[116]: ['h1 h2 h3 h4 h5\th6 h7\ns1 s2 s3 s4 s5\ts6 s7', '']

In [117]: len(sp1)
Out[117]: 2

In [118]: print(sp1[0])
h1 h2 h3 h4 h5  h6 h7
s1 s2 s3 s4 s5  s6 s7
```

Getting started with Python for science

4. Control Flow
   5. Advanced iteration

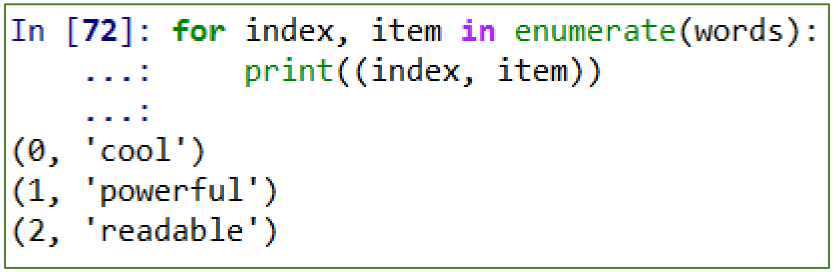
Tuple

1



```
In [68]: words = ('cool', 'powerful', 'readable')

In [69]: len(words)
Out[69]: 3

In [70]: type(words)
Out[70]: tuple

In [71]: for i in range(0, len(words)):
    ...:     print((i, words[i]))
    ...:
(0, 'cool')
(1, 'powerful')
(2, 'readable')
```

2

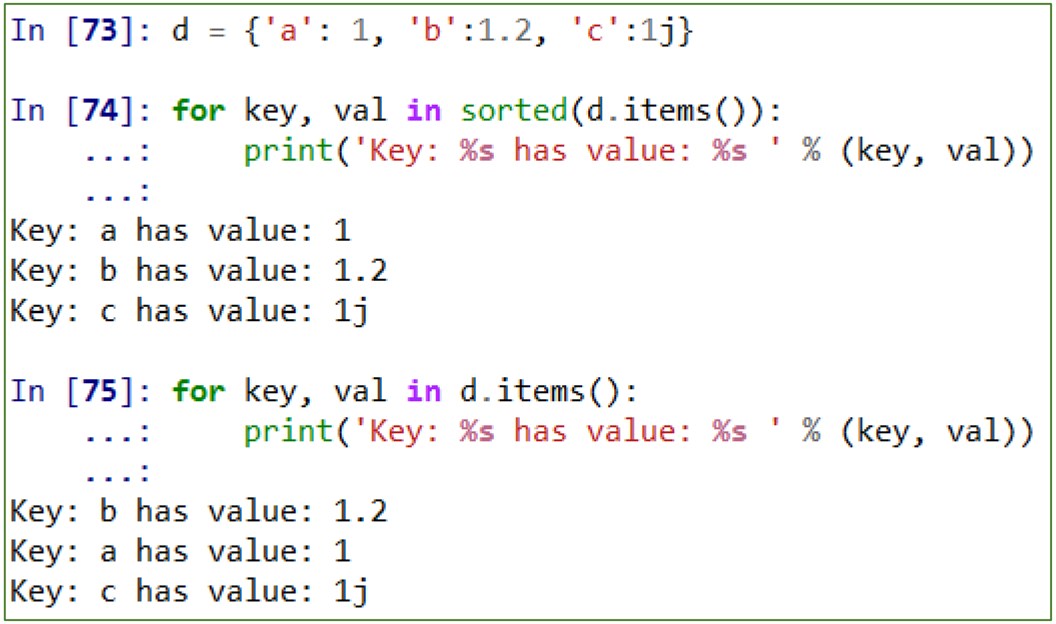


```
In [72]: for index, item in enumerate(words):
    ...:     print((index, item))
    ...:
(0, 'cool')
(1, 'powerful')
(2, 'readable')
```

Getting started with Python for science

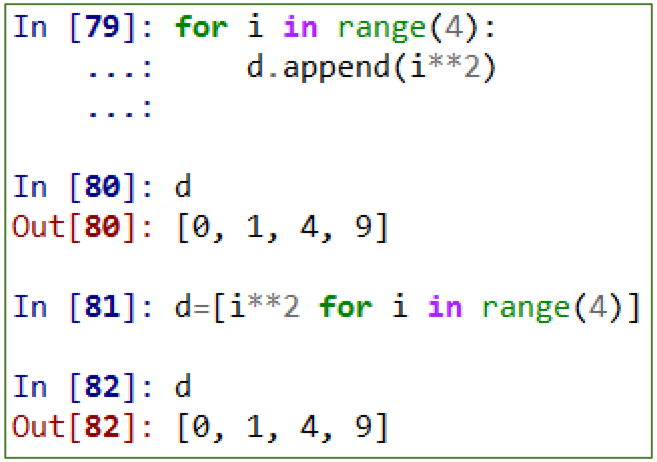
4. Control Flow
    5. Advanced iteration
    6. List Comprehensions

Dictionary



```
In [73]: d = {'a': 1, 'b':1.2, 'c':1j}

In [74]: for key, val in sorted(d.items()):
    ...:     print('Key: %s has value: %s ' % (key, val))
    ...:
Key: a has value: 1
Key: b has value: 1.2
Key: c has value: 1j

In [75]: for key, val in d.items():
    ...:     print('Key: %s has value: %s ' % (key, val))
    ...:
Key: b has value: 1.2
Key: a has value: 1
Key: c has value: 1j
```

List Comprehensions

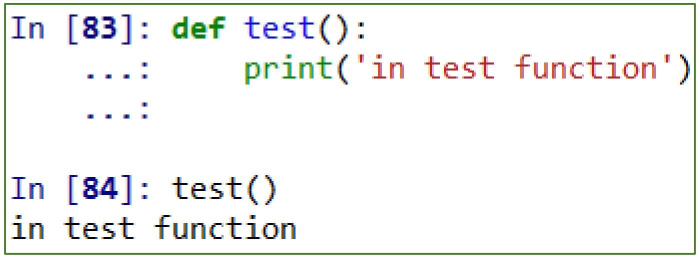


```
In [79]: for i in range(4):
    ...:     d.append(i**2)
    ...:

In [80]: d
Out[80]: [0, 1, 4, 9]

In [81]: d=[i**2 for i in range(4)]

In [82]: d
Out[82]: [0, 1, 4, 9]
```

Getting started with Python for science

5. Defining functions
   1. Function definition
   2. Return statement

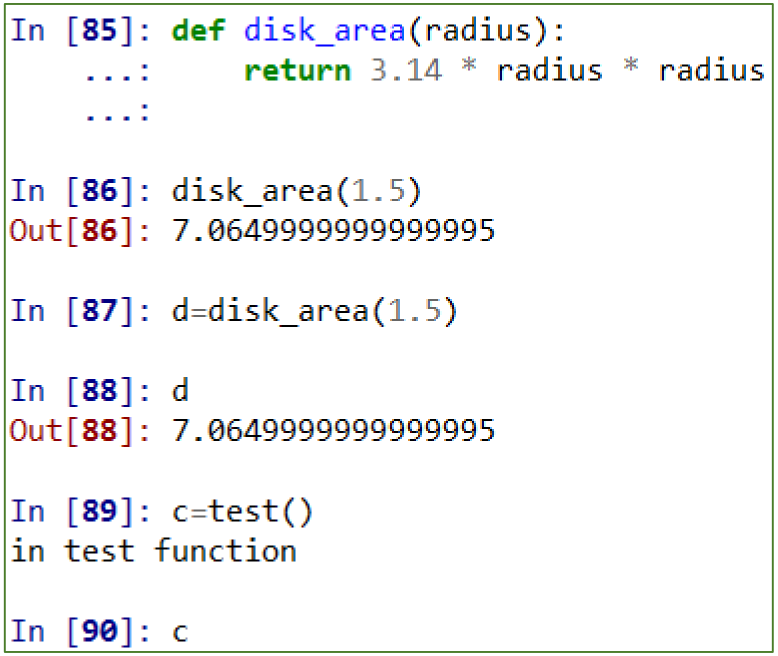
Function definition



```
In [83]: def test():
    ...:     print('in test function')
    ...:

In [84]: test()
in test function
```

By default, functions return None

Return statement

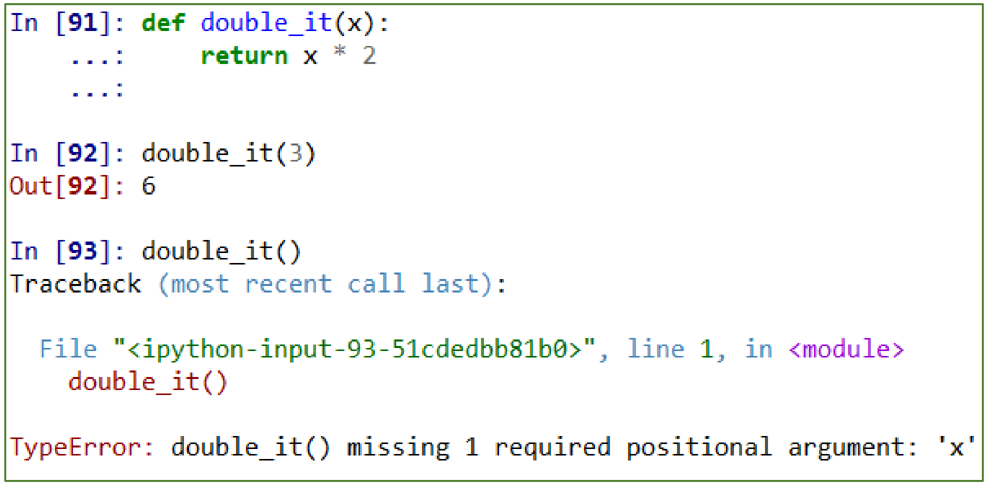


```
In [85]: def disk_area(radius):
    ...:     return 3.14 * radius * radius
    ...:

In [86]: disk_area(1.5)
Out[86]: 7.0649999999999995

In [87]: d=disk_area(1.5)

In [88]: d
Out[88]: 7.0649999999999995

In [89]: c=test()
in test function

In [90]: c
```

Getting started with Python for science

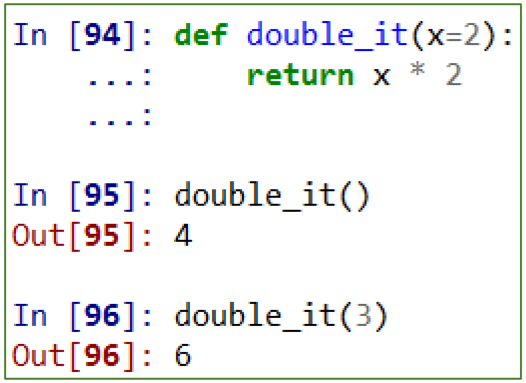
5. Defining functions
   3. Parameters

Mandatory parameters (positional arguments)

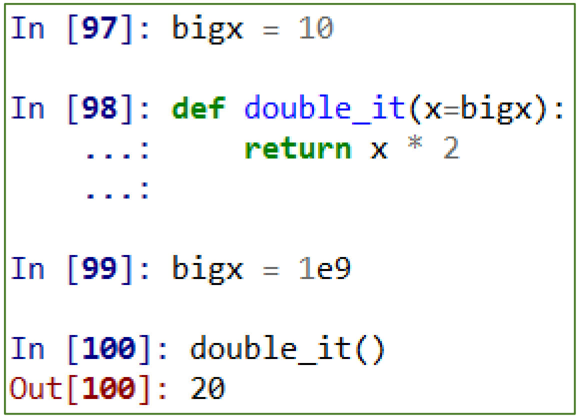


```
In [91]: def double_it(x):
    ...:     return x * 2
    ...:

In [92]: double_it(3)
Out[92]: 6

In [93]: double_it()
Traceback (most recent call last):

  File "<ipython-input-93-51cdedbb81b0>", line 1, in <module>
    double_it()

TypeError: double_it() missing 1 required positional argument: 'x'
```

Optional parameters (keyword or named arguments)



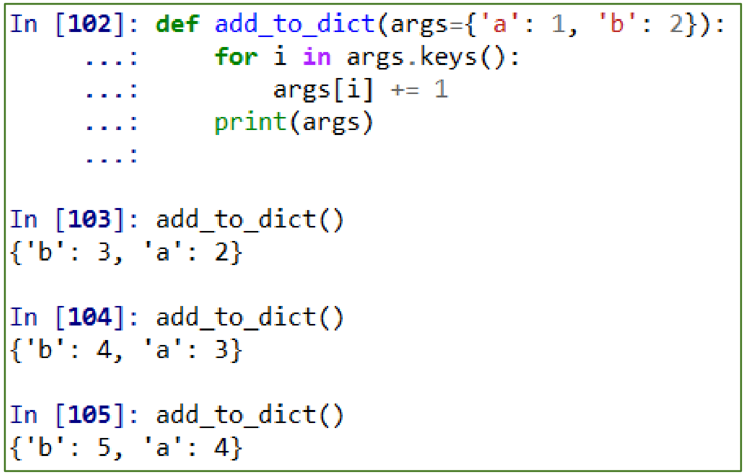
```
In [94]: def double_it(x=2):
    ...:     return x * 2
    ...:

In [95]: double_it()
Out[95]: 4

In [96]: double_it(3)
Out[96]: 6
```



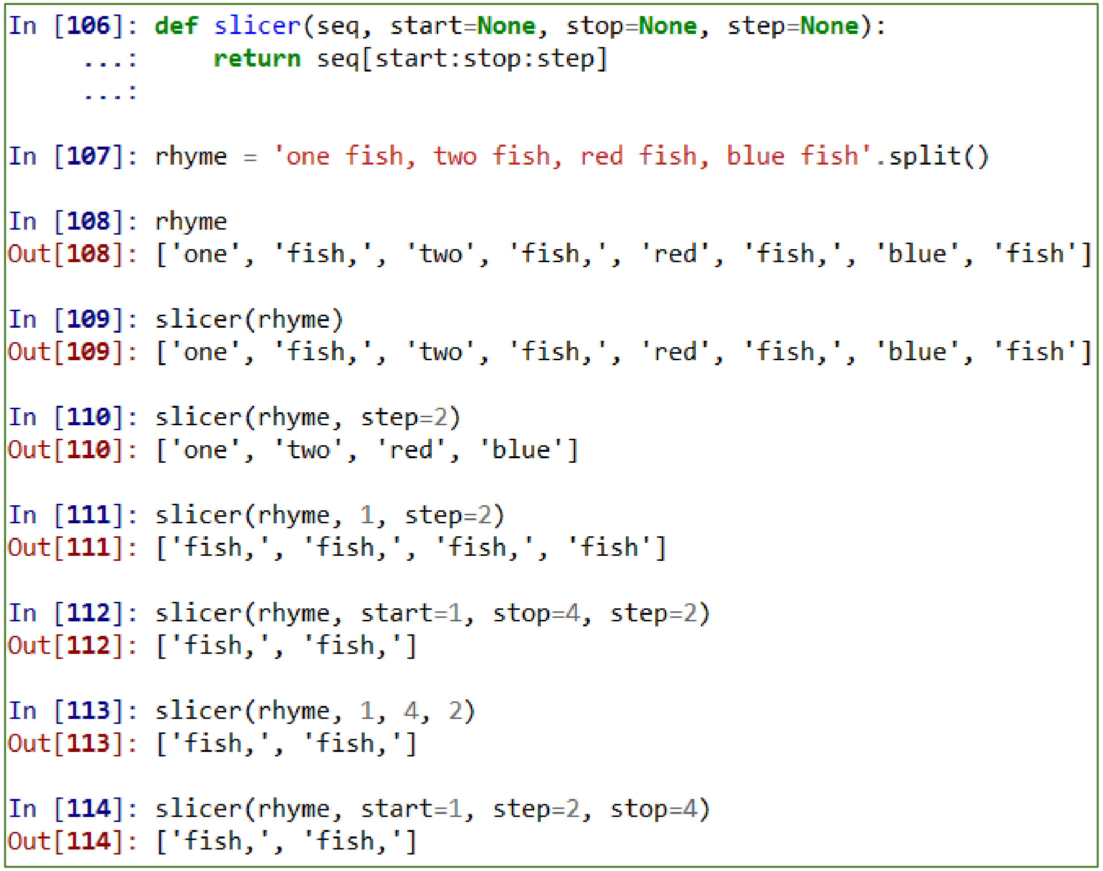
```
In [97]: bigx = 10

In [98]: def double_it(x=bigx):
    ...:     return x * 2
    ...:

In [99]: bigx = 1e9

In [100]: double_it()
Out[100]: 20
```

Getting started with Python for science

5. Defining functions
    3. Parameters



```
In [102]: def add_to_dict(args={'a': 1, 'b': 2}):
     ...:     for i in args.keys():
     ...:         args[i] += 1
     ...:     print(args)
     ...:

In [103]: add_to_dict()
{'b': 3, 'a': 2}

In [104]: add_to_dict()
{'b': 4, 'a': 3}

In [105]: add_to_dict()
{'b': 5, 'a': 4}
```



```
In [106]: def slicer(seq, start=None, stop=None, step=None):
     ...:     return seq[start:stop:step]
     ...:

In [107]: rhyme = 'one fish, two fish, red fish, blue fish'.split()

In [108]: rhyme
Out[108]: ['one', 'fish,', 'two', 'fish,', 'red', 'fish,', 'blue', 'fish']

In [109]: slicer(rhyme)
Out[109]: ['one', 'fish,', 'two', 'fish,', 'red', 'fish,', 'blue', 'fish']

In [110]: slicer(rhyme, step=2)
Out[110]: ['one', 'two', 'red', 'blue']

In [111]: slicer(rhyme, 1, step=2)
Out[111]: ['fish,', 'fish,', 'fish,', 'fish']

In [112]: slicer(rhyme, start=1, stop=4, step=2)
Out[112]: ['fish,', 'fish,']

In [113]: slicer(rhyme, 1, 4, 2)
Out[113]: ['fish,', 'fish,']

In [114]: slicer(rhyme, start=1, step=2, stop=4)
Out[114]: ['fish,', 'fish,']
```

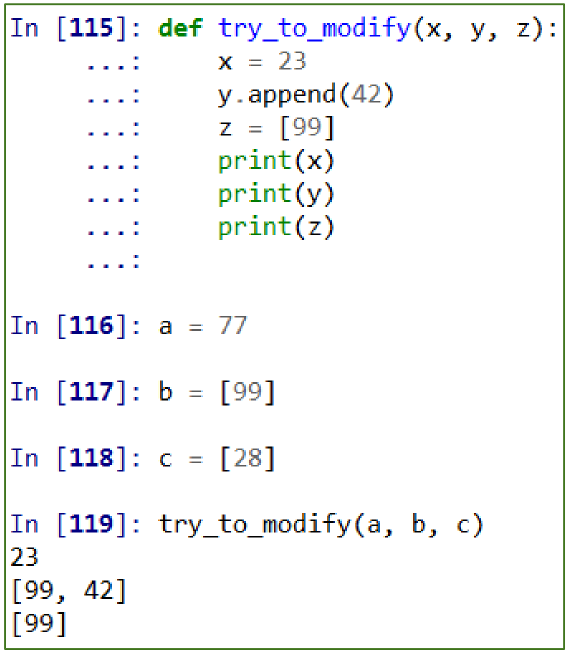
Getting started with Python for science

5. Defining functions

4. Passing by value

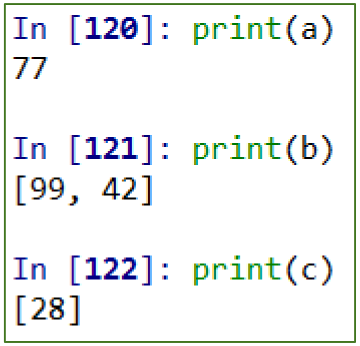
If the value passed in a function is **immutable**, the function does not modify the caller's variable. If the value is **mutable**, the function may modify the caller's variable in-place:

1



```
In [115]: def try_to_modify(x, y, z):
     ...:     x = 23
     ...:     y.append(42)
     ...:     z = [99]
     ...:     print(x)
     ...:     print(y)
     ...:     print(z)
     ...:

In [116]: a = 77

In [117]: b = [99]

In [118]: c = [28]

In [119]: try_to_modify(a, b, c)
23
[99, 42]
[99]
```

2



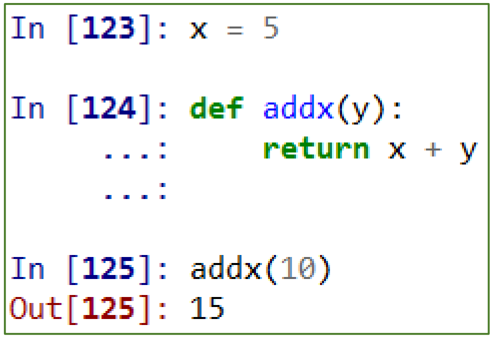
```
In [120]: print(a)
77

In [121]: print(b)
[99, 42]

In [122]: print(c)
[28]
```

Getting started with Python for science

5. Defining functions
   5. Global variables

Variables declared outside the function can be referenced within the function:

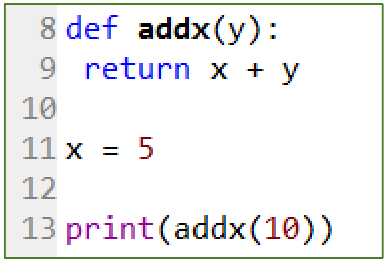
IPython consol



```
In [123]: x = 5

In [124]: def addx(y):
     ...:     return x + y
     ...:

In [125]: addx(10)
Out[125]: 15
```

Script

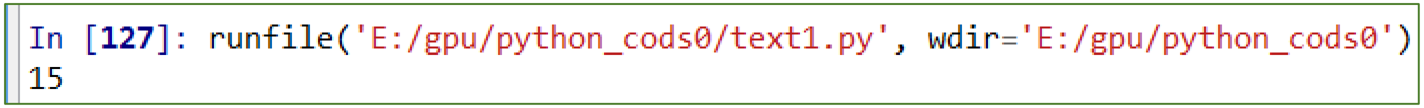
2



```
def addx(y):
 return x + y

x = 5

print(addx(10))
```

1

Output in IPython consol

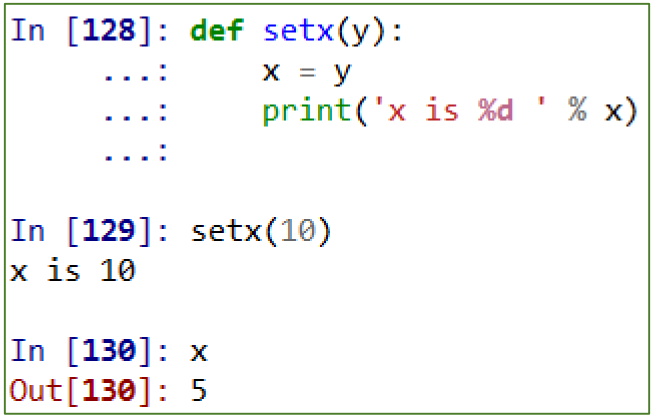


```
In [127]: runfile('E:/gpu/python_cods0/text1.py', wdir='E:/gpu/python_cods0')
15
```

Getting started with Python for science

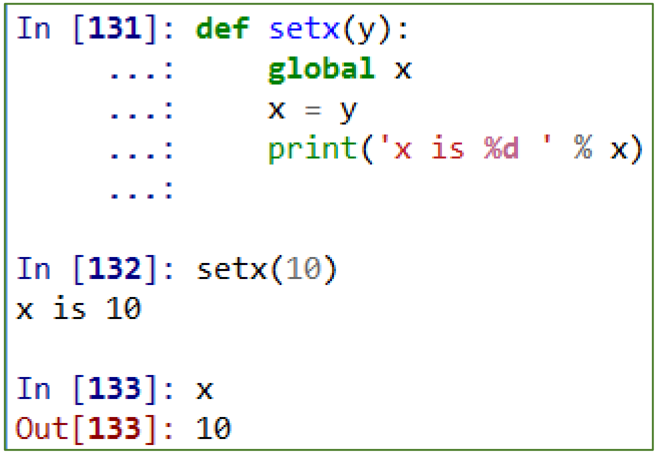
5. Defining functions
   5. Global variables

1



```
In [128]: def setx(y):
     ...:     x = y
     ...:     print('x is %d ' % x)
     ...:

In [129]: setx(10)
x is 10

In [130]: x
Out[130]: 5
```

2



```
In [131]: def setx(y):
     ...:     global x
     ...:     x = y
     ...:     print('x is %d ' % x)
     ...:

In [132]: setx(10)
x is 10

In [133]: x
Out[133]: 10
```

Getting started with Python for science

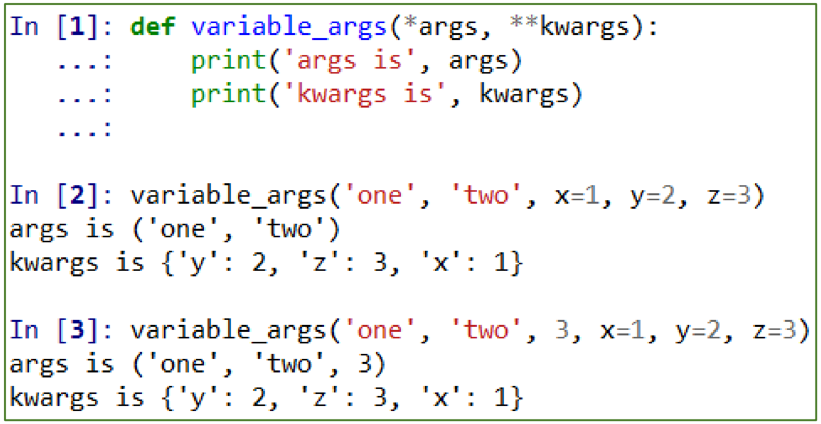
5. Defining functions

7. Variable number of parameters

Special forms of parameters:

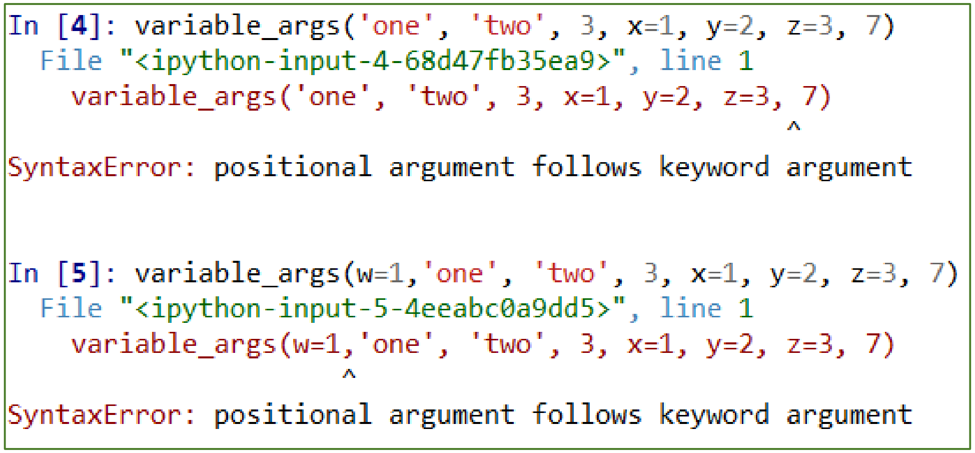
- ***args**: any number of **positional arguments** packed into a tuple
- ****kwargs**: any number of **keyword arguments** packed into a dictionary

1



```
In [1]: def variable_args(*args, **kwargs):
   ...:     print('args is', args)
   ...:     print('kwargs is', kwargs)
   ...:

In [2]: variable_args('one', 'two', x=1, y=2, z=3)
args is ('one', 'two')
kwargs is {'y': 2, 'z': 3, 'x': 1}

In [3]: variable_args('one', 'two', 3, x=1, y=2, z=3)
args is ('one', 'two', 3)
kwargs is {'y': 2, 'z': 3, 'x': 1}
```

2



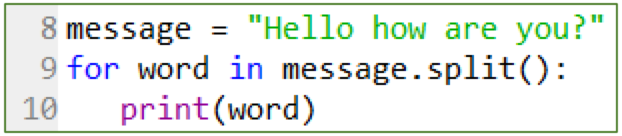
```
In [4]: variable_args('one', 'two', 3, x=1, y=2, z=3, 7)
  File "<ipython-input-4-68d47fb35ea9>", line 1
    variable_args('one', 'two', 3, x=1, y=2, z=3, 7)
                                                 ^
SyntaxError: positional argument follows keyword argument


In [5]: variable_args(w=1,'one', 'two', 3, x=1, y=2, z=3, 7)
  File "<ipython-input-5-4eeabc0a9dd5>", line 1
    variable_args(w=1,'one', 'two', 3, x=1, y=2, z=3, 7)
                     ^
SyntaxError: positional argument follows keyword argument
```

Getting started with Python for science

6. Reusing code: scripts and modules
   1. Scripts

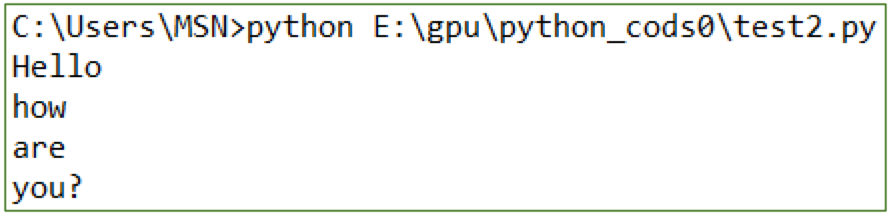
For now, we have typed all instructions in the **interpreter**. For longer sets of instructions we need to change track and write the code in **text files** (using a text editor), that we will call either **scripts** or **modules**. The extension for Python files is .py

Script (test2.py)



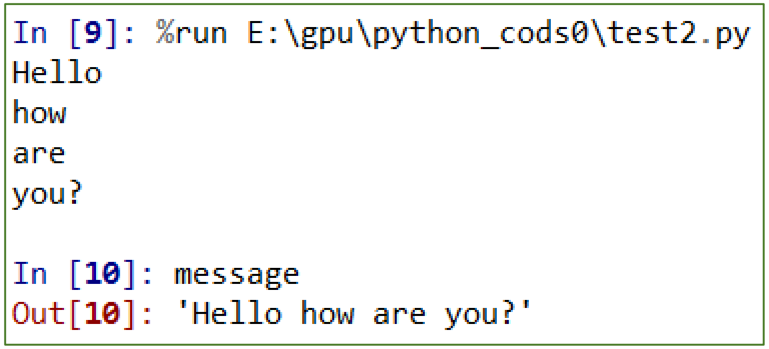
```python
message = "Hello how are you?"
for word in message.split():
    print(word)
```

Run in cmd terminal



```
C:\Users\MSN>python E:\gpu\python_cods0\test2.py
Hello
how
are
you?
```

Run in IPython console



```
In [9]: %run E:\gpu\python_cods0\test2.py
Hello
how
are
you?

In [10]: message
Out[10]: 'Hello how are you?'
```

Getting started with Python for science

2. Write a program that takes information of the five students as follows:

Student identification number (SID):

Name:

Last name:

Age:

Major:

National ID:

A list of elective courses, grades and units in the previous semester:

Getting started with Python for science

- Divide the information into two categories: personal and educational.
- Student information can be searched using the SID.
- It is possible to access information separately.
- Calculate the grade point average (GPA) of the previous semester for each student by defining a function.

Getting started with Python for science

- Define a function that receives student information and prints them as follow:

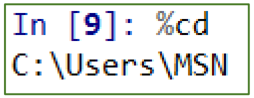
```
SID: 999865021
Personal information:
        Name: Mina
        Last name: Ebrahimi
        Age: 23
        National ID: 0374361221
Educational information:
        Major: physics
        Courses: cs1, cs2, cs3, cs4,...
        GPA: 18.35
```

IPython

https://ipython.readthedocs.io/en/stable/interactive/magics.html

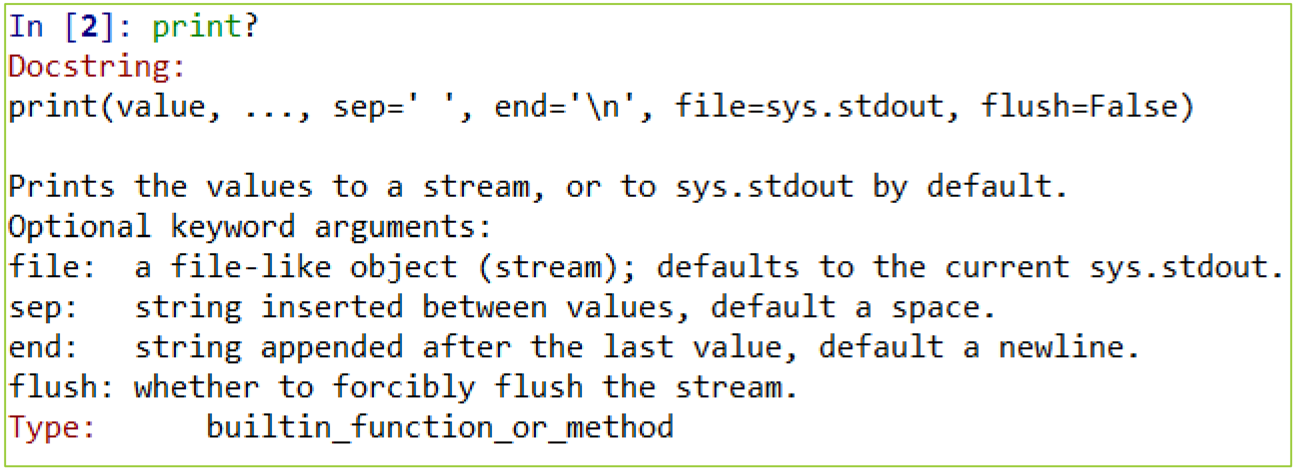
magic functions

%run my_file.py

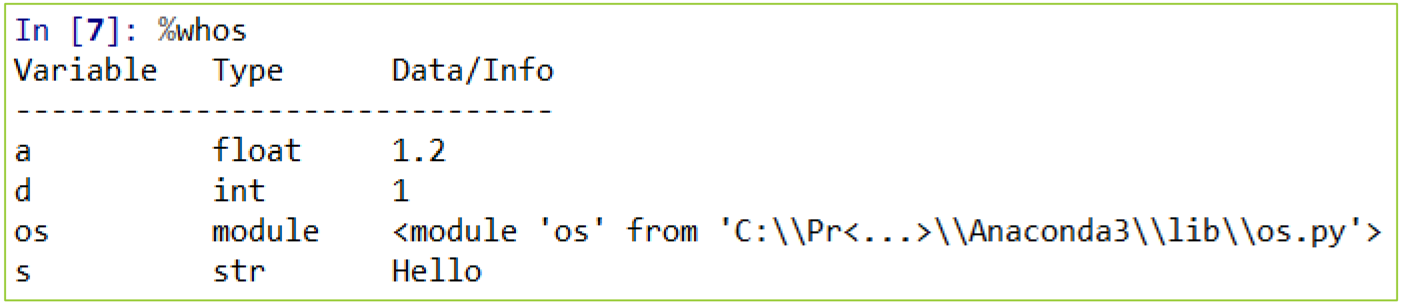


```
In [9]: %cd
C:\Users\MSN
```

Help

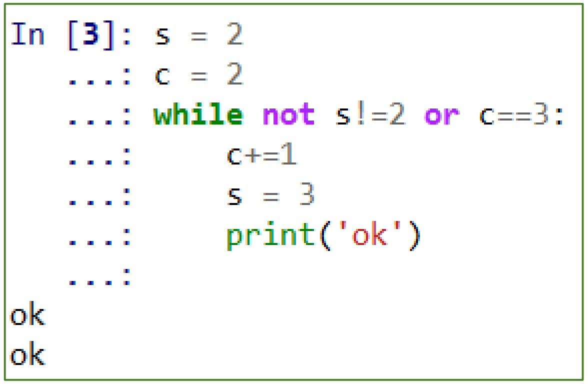


```
In [2]: print?
Docstring:
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

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.
Type:      builtin_function_or_method
```

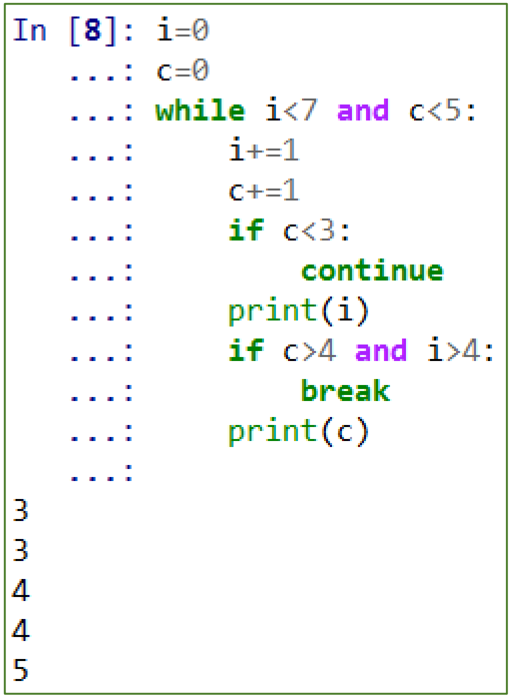


```
In [7]: %whos
Variable   Type      Data/Info
------------------------------
a          float     1.2
d          int       1
os         module    <module 'os' from 'C:\\Pr<...>\\Anaconda3\\lib\\os.py'>
s          str       Hello
```

Examples



```
In [3]: s = 2
   ...: c = 2
   ...: while not s!=2 or c==3:
   ...:     c+=1
   ...:     s = 3
   ...:     print('ok')
   ...:
ok
ok
```

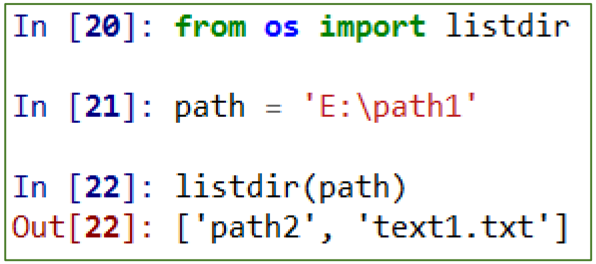


```
In [8]: i=0
   ...: c=0
   ...: while i<7 and c<5:
   ...:     i+=1
   ...:     c+=1
   ...:     if c<3:
   ...:         continue
   ...:     print(i)
   ...:     if c>4 and i>4:
   ...:         break
   ...:     print(c)
   ...:
3
3
4
4
5
```

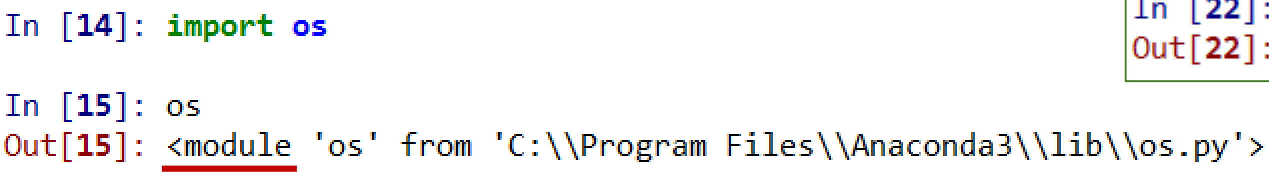
Getting started with Python for science

6. Reusing code: scripts and modules
    2. Importing objects from modules

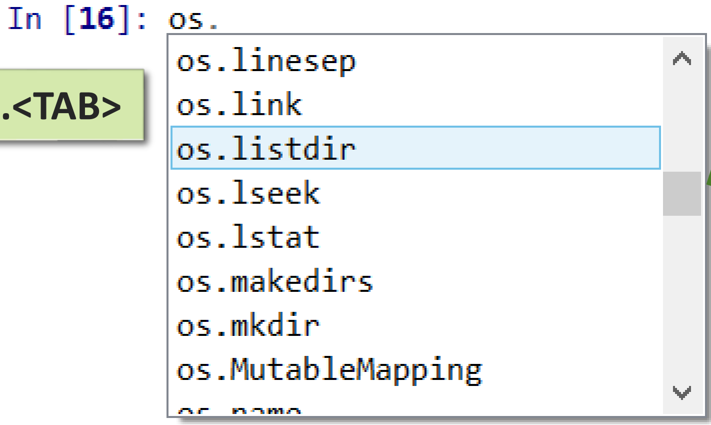
Importing object from module



```
In [20]: from os import listdir

In [21]: path = 'E:\path1'

In [22]: listdir(path)
Out[22]: ['path2', 'text1.txt']
```



```
In [14]: import os

In [15]: os
Out[15]: <module 'os' from 'C:\\Program Files\\Anaconda3\\lib\\os.py'>
```

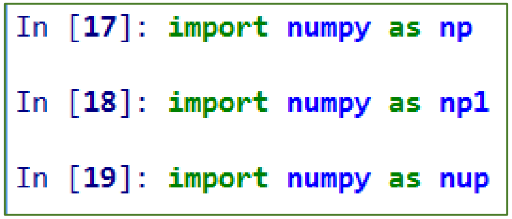


```
In [16]: os.
```

os.<TAB>

Objects of os

Importing shorthands:



```
In [17]: import numpy as np

In [18]: import numpy as np1

In [19]: import numpy as nup
```

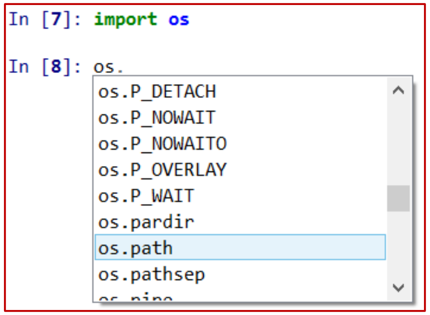
Getting started with Python for science

6. Reusing code: scripts and modules

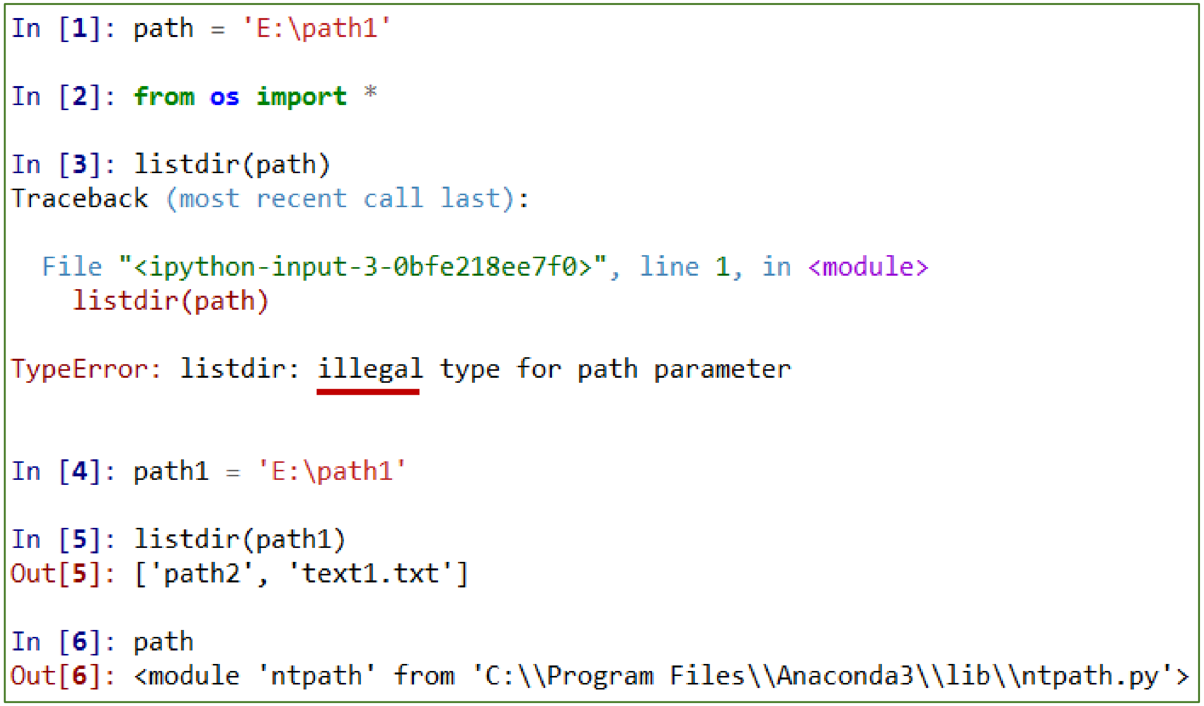
2. Importing objects from modules

Star import

Use it with caution



```
In [7]: import os

In [8]: os.
```

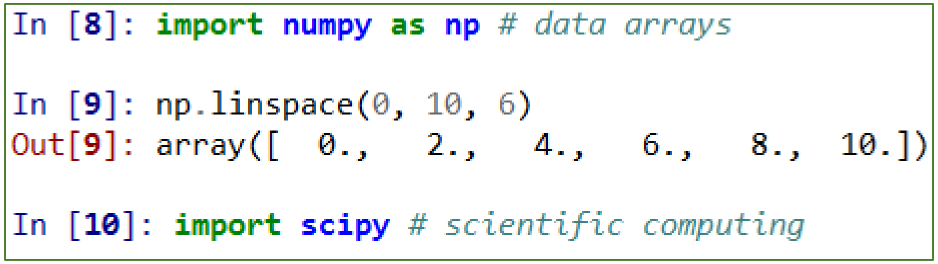


```
In [1]: path = 'E:\path1'

In [2]: from os import *

In [3]: listdir(path)
Traceback (most recent call last):

  File "<ipython-input-3-0bfe218ee7f0>", line 1, in <module>
    listdir(path)

TypeError: listdir: illegal type for path parameter


In [4]: path1 = 'E:\path1'

In [5]: listdir(path1)
Out[5]: ['path2', 'text1.txt']

In [6]: path
Out[6]: <module 'ntpath' from 'C:\\Program Files\\Anaconda3\\lib\\ntpath.py'>
```

Getting started with Python for science

6. Reusing code: scripts and modules
    2. Importing objects from modules

Modules are thus a good way to organize code in a hierarchical way. Actually, all the scientific computing tools we are going to use are modules:



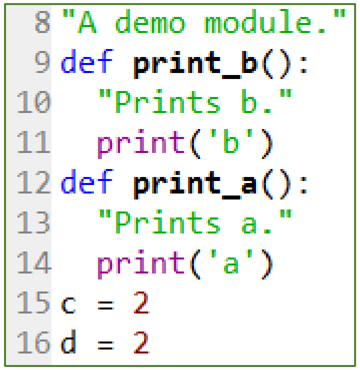
```
In [8]: import numpy as np # data arrays

In [9]: np.linspace(0, 10, 6)
Out[9]: array([  0.,   2.,   4.,   6.,   8.,  10.])

In [10]: import scipy # scientific computing
```

Getting started with Python for science

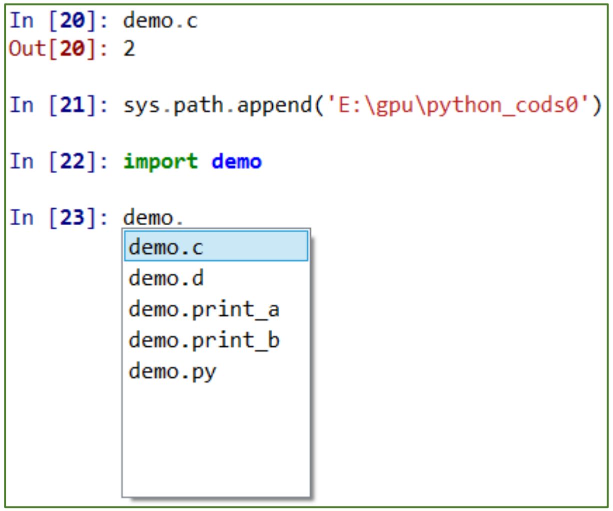
6. Reusing code: scripts and modules
   3. Creating modules

If we want to write larger and better organized programs (compared to simple scripts), where some objects are defined, (variables, functions, classes) and that we want to reuse several times, we have to create our own **modules**.

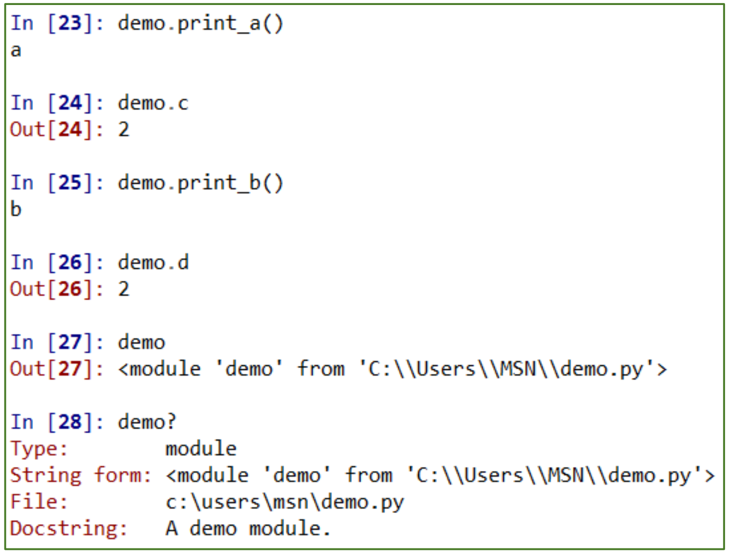
demo.py



```
"A demo module."
def print_b():
  "Prints b."
  print('b')
def print_a():
  "Prints a."
  print('a')
c = 2
d = 2
```



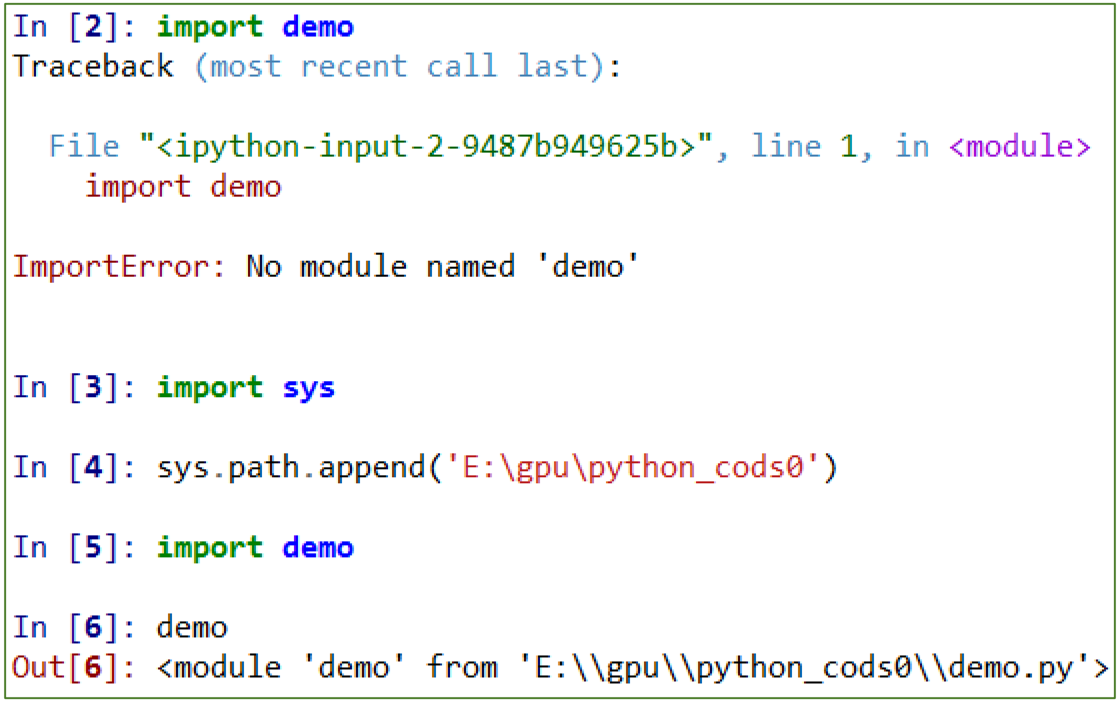
```
In [20]: demo.c
Out[20]: 2

In [21]: sys.path.append('E:\gpu\python_cods0')

In [22]: import demo

In [23]: demo.
         demo.c
         demo.d
         demo.print_a
         demo.print_b
         demo.py
```



```
In [23]: demo.print_a()
a

In [24]: demo.c
Out[24]: 2

In [25]: demo.print_b()
b

In [26]: demo.d
Out[26]: 2

In [27]: demo
Out[27]: <module 'demo' from 'C:\\Users\\MSN\\demo.py'>

In [28]: demo?
Type:        module
String form: <module 'demo' from 'C:\\Users\\MSN\\demo.py'>
File:        c:\users\msn\demo.py
Docstring:   A demo module.
```

Getting started with Python for science

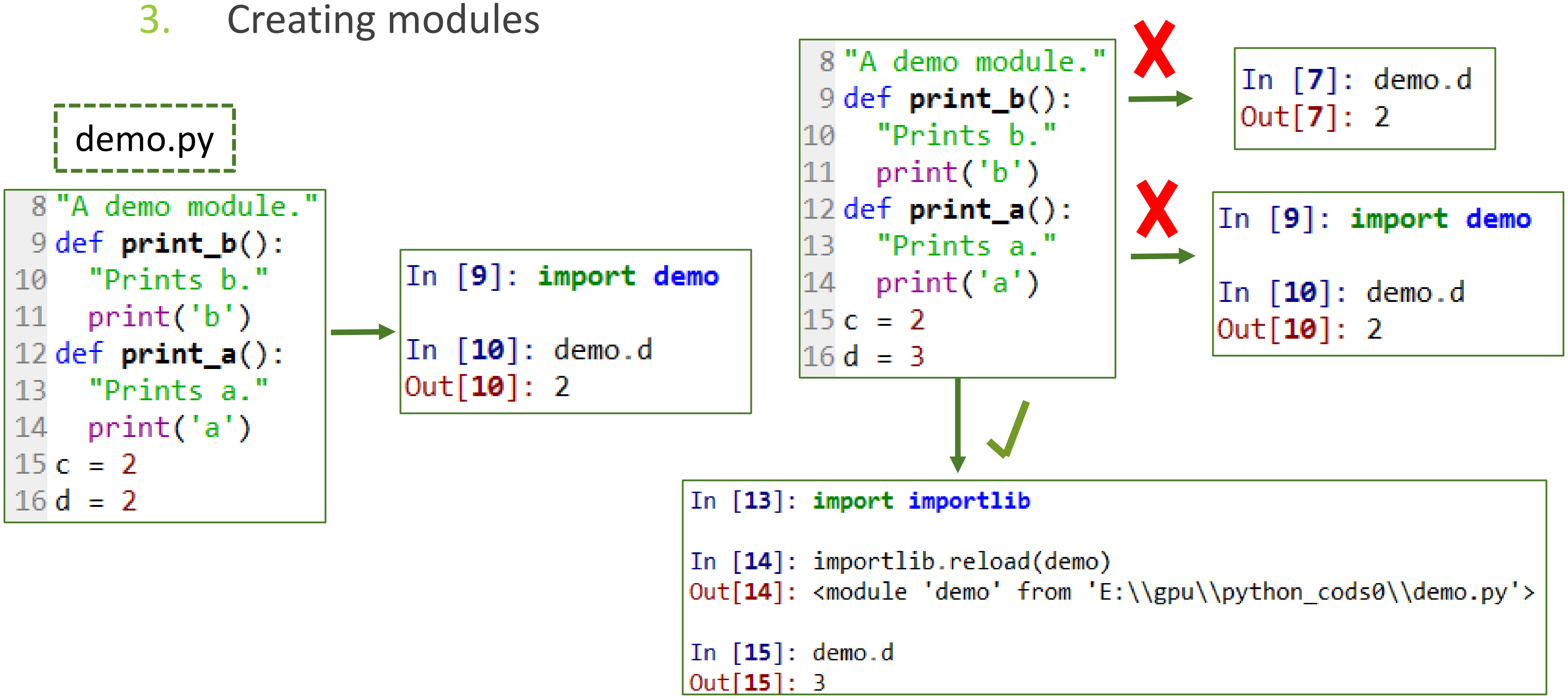
6. Reusing code: scripts and modules
   3. Creating modules



```
In [2]: import demo
Traceback (most recent call last):

  File "<ipython-input-2-9487b949625b>", line 1, in <module>
    import demo

ImportError: No module named 'demo'


In [3]: import sys

In [4]: sys.path.append('E:\gpu\python_cods0')

In [5]: import demo

In [6]: demo
Out[6]: <module 'demo' from 'E:\\gpu\\python_cods0\\demo.py'>
```

Getting started with Python for science

6. Reusing code: scripts and modules
    3. Creating modules



demo.py

```python
"A demo module."
def print_b():
    "Prints b."
    print('b')
def print_a():
    "Prints a."
    print('a')
c = 2
d = 2
```

```
In [9]: import demo

In [10]: demo.d
Out[10]: 2
```

```python
"A demo module."
def print_b():
    "Prints b."
    print('b')
def print_a():
    "Prints a."
    print('a')
c = 2
d = 3
```

✗

```
In [7]: demo.d
Out[7]: 2
```

✗

```
In [9]: import demo

In [10]: demo.d
Out[10]: 2
```

✓

```
In [13]: import importlib

In [14]: importlib.reload(demo)
Out[14]: <module 'demo' from 'E:\\gpu\\python_cods0\\demo.py'>

In [15]: demo.d
Out[15]: 3
```

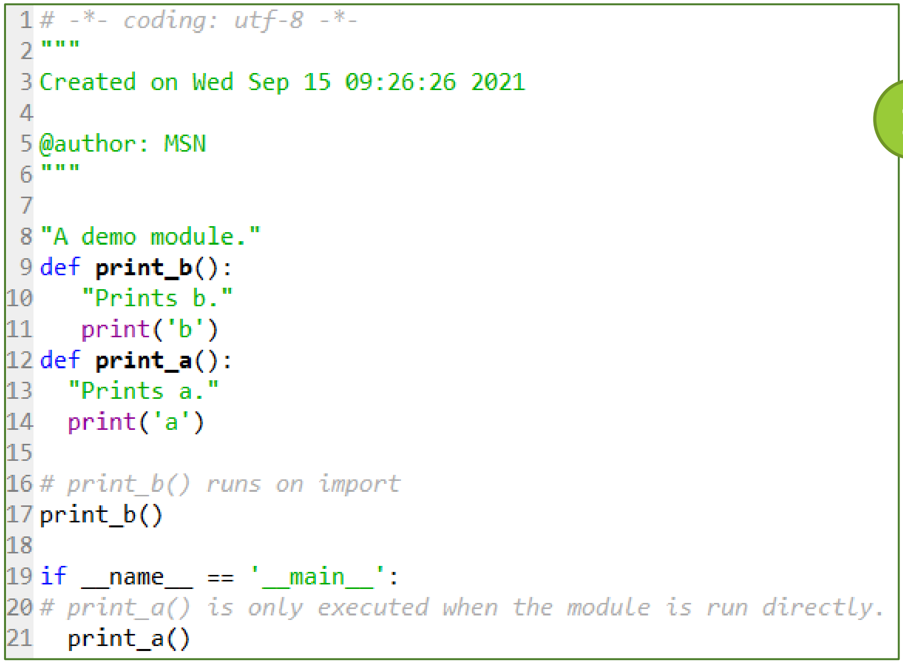
Getting started with Python for science

6. Reusing code: scripts and modules

4. '__main__' and module loading

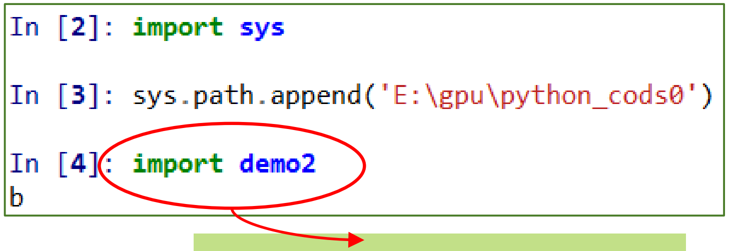
Sometimes we want code to be executed when a module is run directly, but not when it is imported by another module. if **__name__ == '__main__'** allows us to check whether the module is being run directly.

module



```python
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 15 09:26:26 2021

@author: MSN
"""

"A demo module."
def print_b():
    "Prints b."
    print('b')
def print_a():
  "Prints a."
  print('a')

# print_b() runs on import
print_b()

if __name__ == '__main__':
# print_a() is only executed when the module is run directly.
  print_a()
```

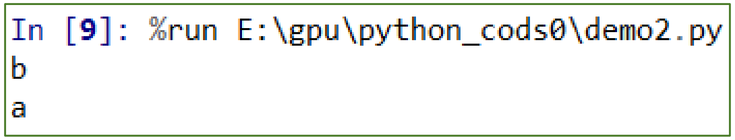
1



```
In [2]: import sys

In [3]: sys.path.append('E:\gpu\python_cods0')

In [4]: import demo2
b
```

imported by another module

2 **run directly**



```
In [9]: %run E:\gpu\python_cods0\demo2.py
b
a
```

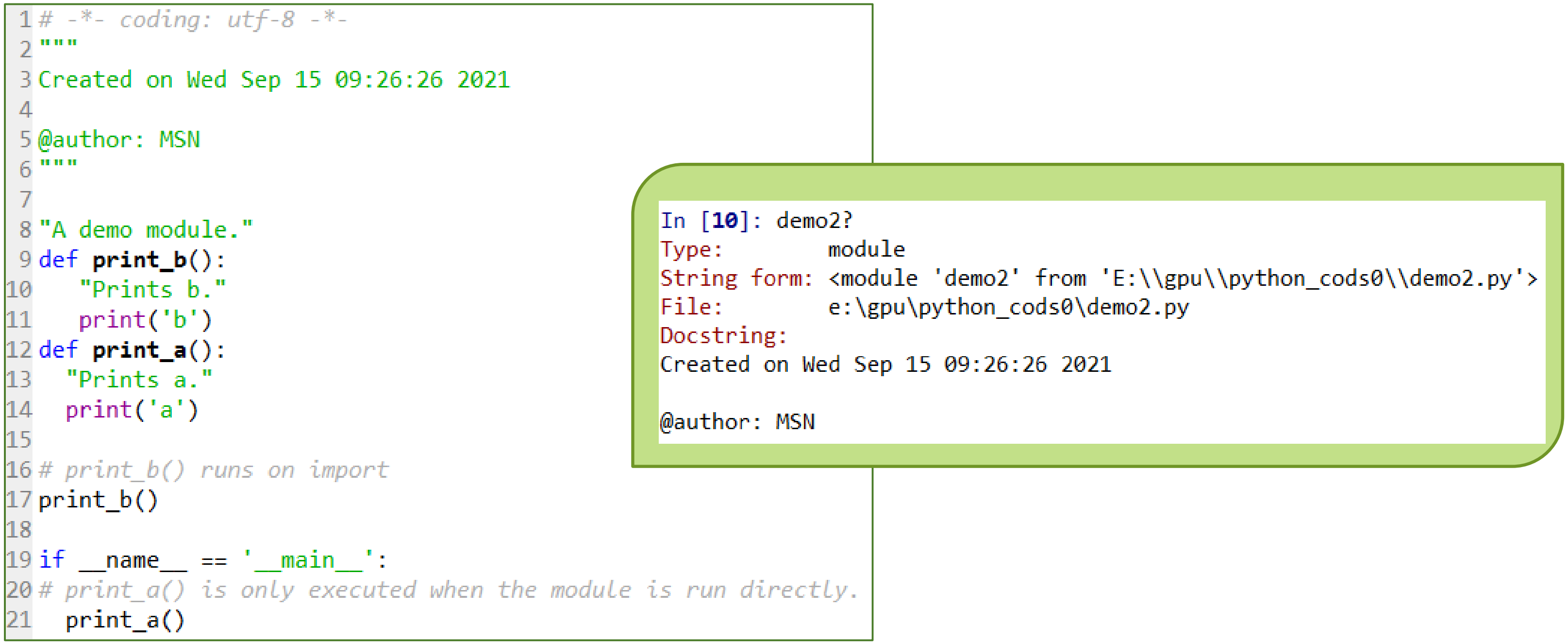
Getting started with Python for science

6. Reusing code: scripts and modules

4. '__main__' and module loading

Sometimes we want code to be executed when a module is run directly, but not when it is imported by another module. if **__name__ == '__main__'** allows us to check whether the module is being run directly.

module



```python
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 15 09:26:26 2021

@author: MSN
"""

"A demo module."
def print_b():
    "Prints b."
    print('b')
def print_a():
  "Prints a."
  print('a')

# print_b() runs on import
print_b()

if __name__ == '__main__':
# print_a() is only executed when the module is run directly.
  print_a()
```

```
In [10]: demo2?
Type:        module
String form: <module 'demo2' from 'E:\\gpu\\python_cods0\\demo2.py'>
File:        e:\gpu\python_cods0\demo2.py
Docstring:
Created on Wed Sep 15 09:26:26 2021

@author: MSN
```

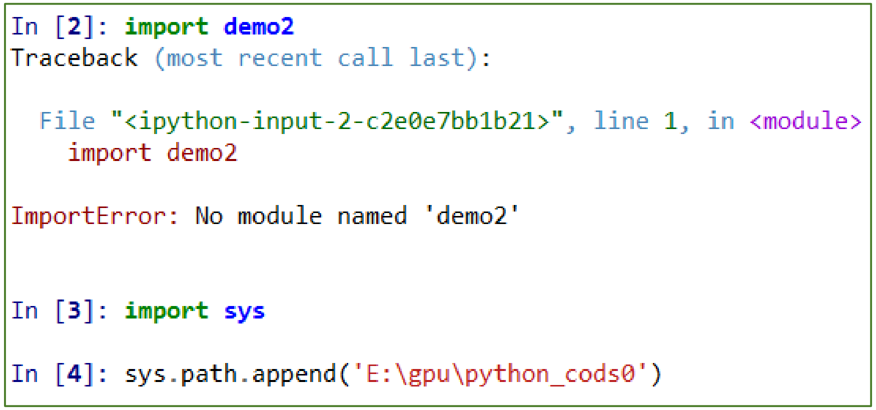
Getting started with Python for science

6. Reusing code: scripts and modules

4. ‘__main__’ and module loading

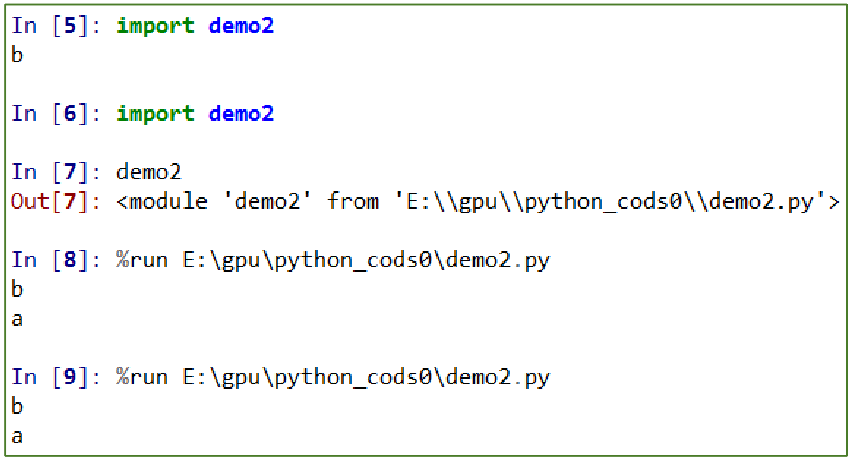
Sometimes we want code to be executed when a module is run directly, but not when it is imported by another module. if **__name__ == ’__main__’** allows us to check whether the module is being run directly.

1



```
In [2]: import demo2
Traceback (most recent call last):

  File "<ipython-input-2-c2e0e7bb1b21>", line 1, in <module>
    import demo2

ImportError: No module named 'demo2'


In [3]: import sys

In [4]: sys.path.append('E:\gpu\python_cods0')
```

2



```
In [5]: import demo2
b

In [6]: import demo2

In [7]: demo2
Out[7]: <module 'demo2' from 'E:\\gpu\\python_cods0\\demo2.py'>

In [8]: %run E:\gpu\python_cods0\demo2.py
b
a

In [9]: %run E:\gpu\python_cods0\demo2.py
b
a
```

Getting started with Python for science

6. Reusing code: scripts and modules
   5. Scripts or modules? How to organize your code

Rule of thumb

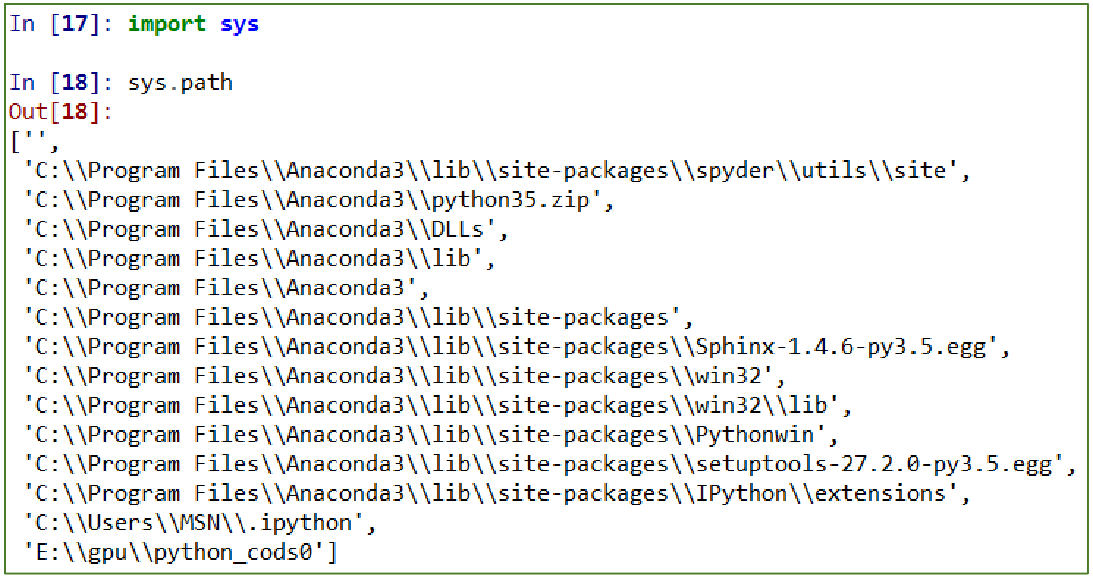
- Sets of instructions that are called several times should be written inside **functions** for better code reusability.
- Functions (or other bits of code) that are called from several scripts should be written inside a **module**, so that only the module is imported in the different scripts (do not copy-and-paste your functions in the different scripts!).

How modules are found and imported

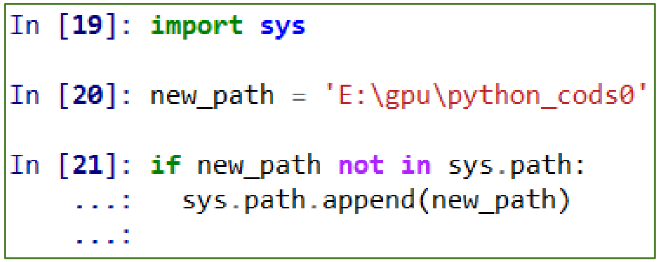
When the import mymodule statement is executed, the module mymodule is searched in a given list of directories.

Getting started with Python for science

6. Reusing code: scripts and modules
    5. Scripts or modules? How to organize your code



```
In [17]: import sys

In [18]: sys.path
Out[18]:
['',
 'C:\\Program Files\\Anaconda3\\lib\\site-packages\\spyder\\utils\\site',
 'C:\\Program Files\\Anaconda3\\python35.zip',
 'C:\\Program Files\\Anaconda3\\DLLs',
 'C:\\Program Files\\Anaconda3\\lib',
 'C:\\Program Files\\Anaconda3',
 'C:\\Program Files\\Anaconda3\\lib\\site-packages',
 'C:\\Program Files\\Anaconda3\\lib\\site-packages\\Sphinx-1.4.6-py3.5.egg',
 'C:\\Program Files\\Anaconda3\\lib\\site-packages\\win32',
 'C:\\Program Files\\Anaconda3\\lib\\site-packages\\win32\\lib',
 'C:\\Program Files\\Anaconda3\\lib\\site-packages\\Pythonwin',
 'C:\\Program Files\\Anaconda3\\lib\\site-packages\\setuptools-27.2.0-py3.5.egg',
 'C:\\Program Files\\Anaconda3\\lib\\site-packages\\IPython\\extensions',
 'C:\\Users\\MSN\\.ipython',
 'E:\\gpu\\python_cods0']
```



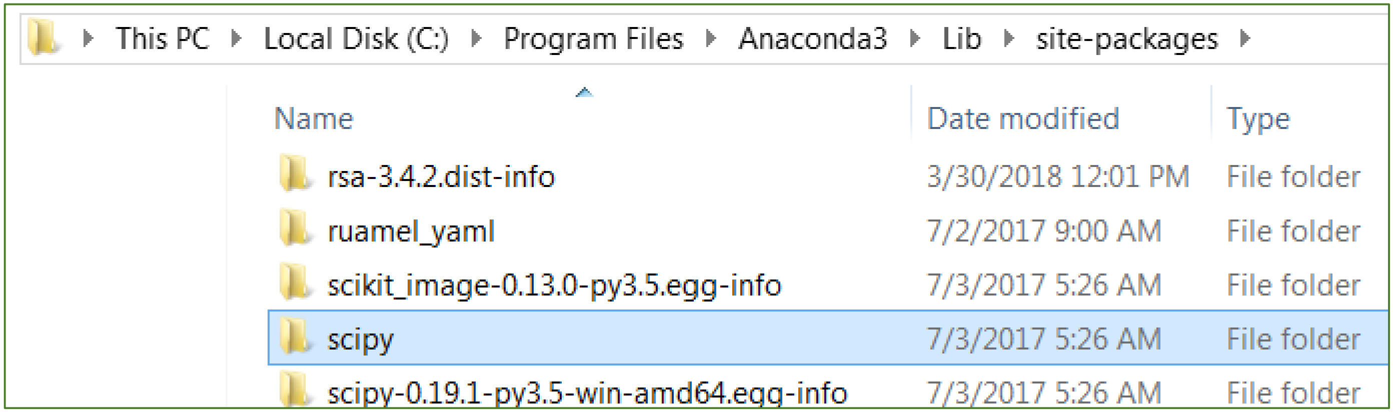
```
In [19]: import sys

In [20]: new_path = 'E:\gpu\python_cods0'

In [21]: if new_path not in sys.path:
    ...:     sys.path.append(new_path)
    ...:
```

Getting started with Python for science

6. Reusing code: scripts and modules

    6. Packages

A directory that contains many modules is called a **package**. A package is a module with submodules (which can have submodules themselves, etc.). A special file called **__init__.py** (which may be empty) tells Python that the directory is a Python package, from which modules can be imported.



This PC ▸ Local Disk (C:) ▸ Program Files ▸ Anaconda3 ▸ Lib ▸ site-packages ▸

Name	Date modified	Type
rsa-3.4.2.dist-info	3/30/2018 12:01 PM	File folder
ruamel_yaml	7/2/2017 9:00 AM	File folder
scikit_image-0.13.0-py3.5.egg-info	7/3/2017 5:26 AM	File folder
scipy	7/3/2017 5:26 AM	File folder
scipy-0.19.1-py3.5-win-amd64.egg-info	7/3/2017 5:26 AM	File folder

Getting started with Python for science

- A **script** is a Python file that’s intended to be run directly. When you run it, it should do something. This means that scripts will often contain code written outside the scope of any classes or functions.

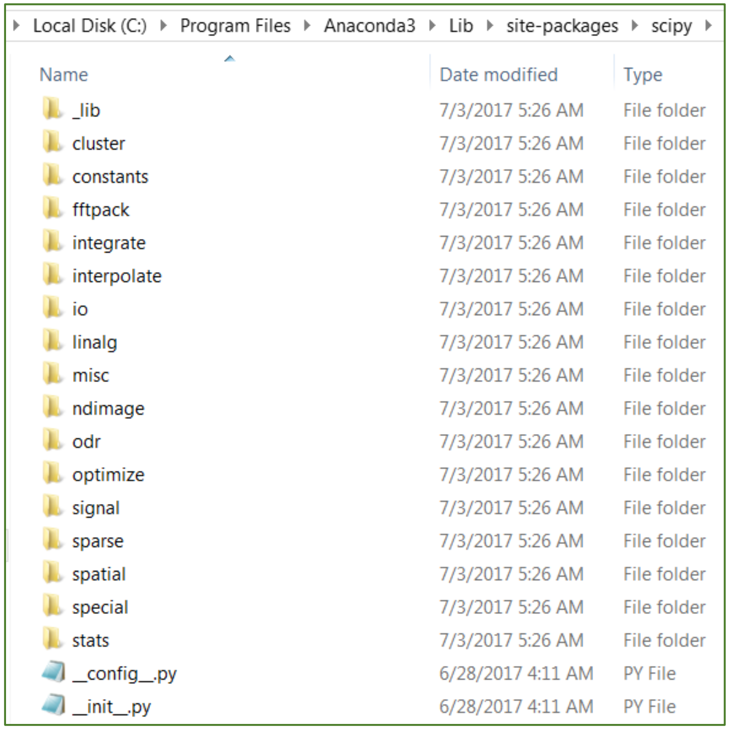
- A **module** is a Python file that’s intended to be imported into scripts or other modules. It often defines members like classes, functions, and variables intended to be used in other files that import it.

- A **package** is a collection of related modules that work together to provide certain functionality. These modules are contained within a folder and can be imported just like any other modules. This folder will often contain a special __init__ file that tells Python it’s a package, potentially containing more modules nested within subfolders

Getting started with Python for science

6. Reusing code: scripts and modules
    6. Packages

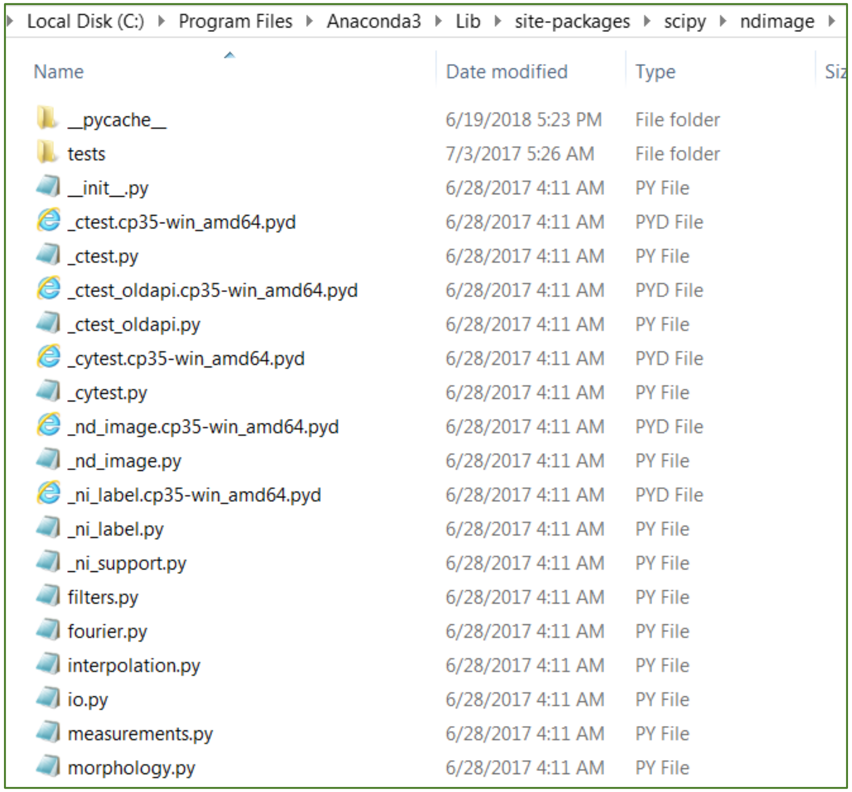
1



Local Disk (C:) ▸ Program Files ▸ Anaconda3 ▸ Lib ▸ site-packages ▸ scipy ▸

Name	Date modified	Type
_lib	7/3/2017 5:26 AM	File folder
cluster	7/3/2017 5:26 AM	File folder
constants	7/3/2017 5:26 AM	File folder
fftpack	7/3/2017 5:26 AM	File folder
integrate	7/3/2017 5:26 AM	File folder
interpolate	7/3/2017 5:26 AM	File folder
io	7/3/2017 5:26 AM	File folder
linalg	7/3/2017 5:26 AM	File folder
misc	7/3/2017 5:26 AM	File folder
ndimage	7/3/2017 5:26 AM	File folder
odr	7/3/2017 5:26 AM	File folder
optimize	7/3/2017 5:26 AM	File folder
signal	7/3/2017 5:26 AM	File folder
sparse	7/3/2017 5:26 AM	File folder
spatial	7/3/2017 5:26 AM	File folder
special	7/3/2017 5:26 AM	File folder
stats	7/3/2017 5:26 AM	File folder
__config__.py	6/28/2017 4:11 AM	PY File
__init__.py	6/28/2017 4:11 AM	PY File

2

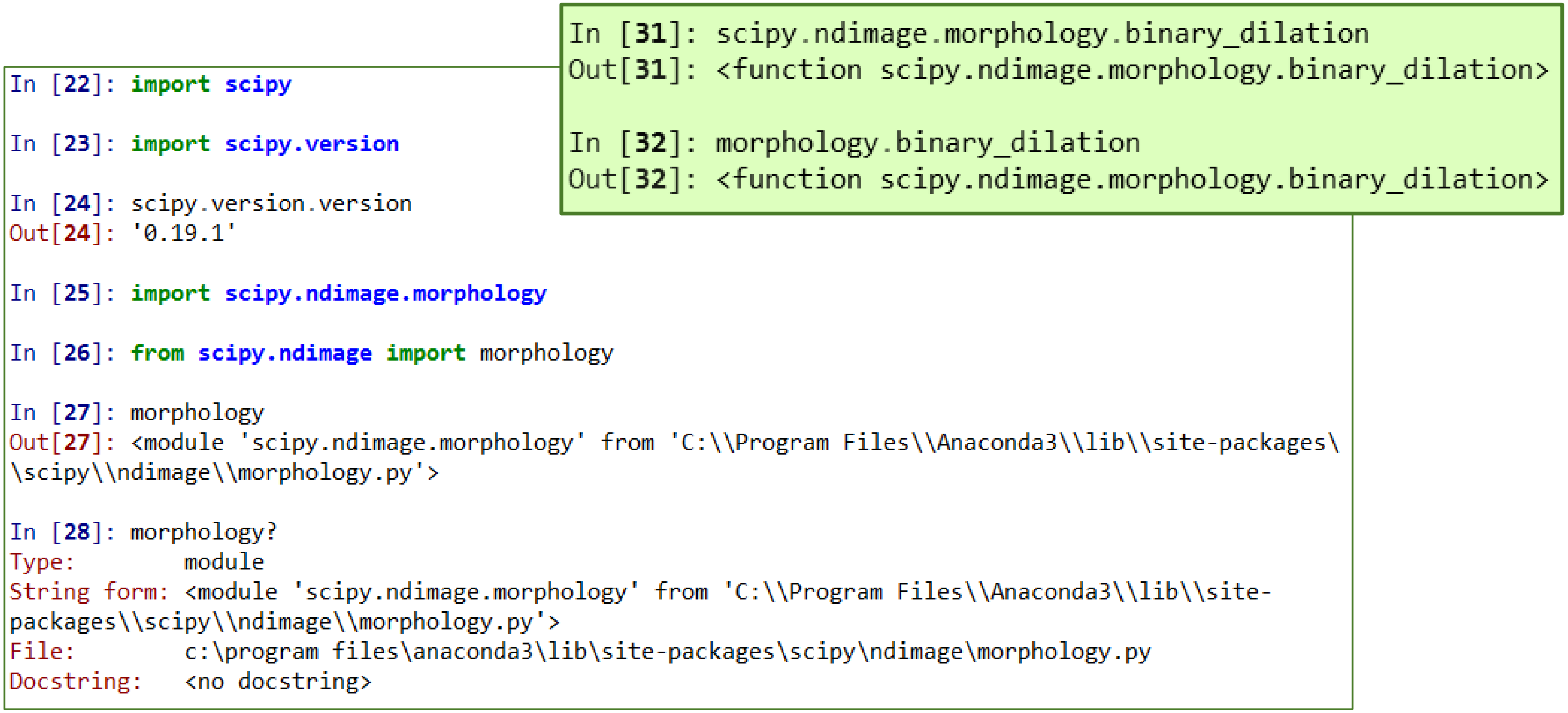


Local Disk (C:) ▸ Program Files ▸ Anaconda3 ▸ Lib ▸ site-packages ▸ scipy ▸ ndimage ▸

Name	Date modified	Type	Siz
__pycache__	6/19/2018 5:23 PM	File folder	
tests	7/3/2017 5:26 AM	File folder	
__init__.py	6/28/2017 4:11 AM	PY File	
_ctest.cp35-win_amd64.pyd	6/28/2017 4:11 AM	PYD File	
_ctest.py	6/28/2017 4:11 AM	PY File	
_ctest_oldapi.cp35-win_amd64.pyd	6/28/2017 4:11 AM	PYD File	
_ctest_oldapi.py	6/28/2017 4:11 AM	PY File	
_cytest.cp35-win_amd64.pyd	6/28/2017 4:11 AM	PYD File	
_cytest.py	6/28/2017 4:11 AM	PY File	
_nd_image.cp35-win_amd64.pyd	6/28/2017 4:11 AM	PYD File	
_nd_image.py	6/28/2017 4:11 AM	PY File	
_ni_label.cp35-win_amd64.pyd	6/28/2017 4:11 AM	PYD File	
_ni_label.py	6/28/2017 4:11 AM	PY File	
_ni_support.py	6/28/2017 4:11 AM	PY File	
filters.py	6/28/2017 4:11 AM	PY File	
fourier.py	6/28/2017 4:11 AM	PY File	
interpolation.py	6/28/2017 4:11 AM	PY File	
io.py	6/28/2017 4:11 AM	PY File	
measurements.py	6/28/2017 4:11 AM	PY File	
morphology.py	6/28/2017 4:11 AM	PY File	

Getting started with Python for science

6. Reusing code: scripts and modules
    6. Packages



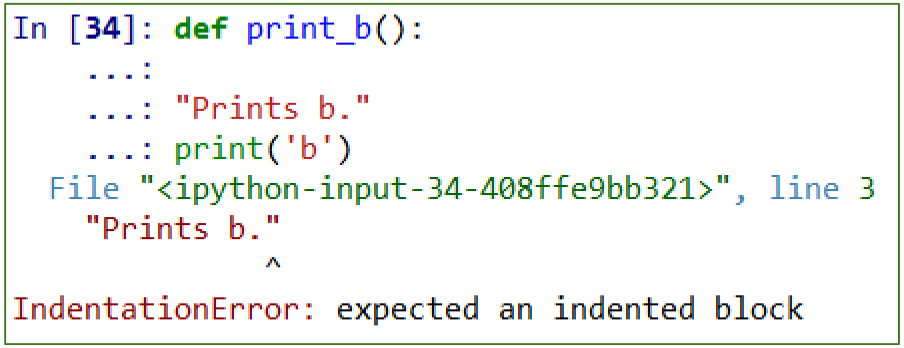
```
In [22]: import scipy

In [23]: import scipy.version

In [24]: scipy.version.version
Out[24]: '0.19.1'

In [25]: import scipy.ndimage.morphology

In [26]: from scipy.ndimage import morphology

In [27]: morphology
Out[27]: <module 'scipy.ndimage.morphology' from 'C:\\Program Files\\Anaconda3\\lib\\site-packages\
\scipy\\ndimage\\morphology.py'>

In [28]: morphology?
Type:        module
String form: <module 'scipy.ndimage.morphology' from 'C:\\Program Files\\Anaconda3\\lib\\site-
packages\\scipy\\ndimage\\morphology.py'>
File:        c:\program files\anaconda3\lib\site-packages\scipy\ndimage\morphology.py
Docstring:   <no docstring>
```

```
In [31]: scipy.ndimage.morphology.binary_dilation
Out[31]: <function scipy.ndimage.morphology.binary_dilation>

In [32]: morphology.binary_dilation
Out[32]: <function scipy.ndimage.morphology.binary_dilation>
```

Getting started with Python for science

6. Reusing code: scripts and modules
    7. Good practices

- Use meaningful object names
- **Indentation**: no choice!



```
In [34]: def print_b():
    ...:
    ...: "Prints b."
    ...: print('b')
  File "<ipython-input-34-408ffe9bb321>", line 3
    "Prints b."
              ^
IndentationError: expected an indented block
```

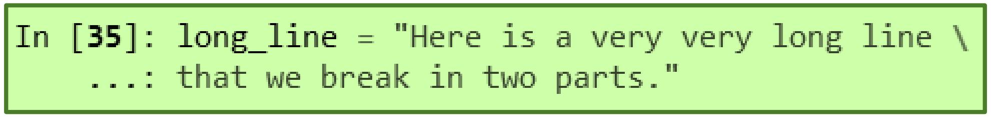
- **Indentation depth**: Inside your text editor, you may choose to indent with any positive number of spaces (1, 2, 3, 4, ...). However, it is considered good practice to indent with 4 spaces. You may configure your editor to map the Tab key to a 4-space indentation.

Getting started with Python for science

6. Reusing code: scripts and modules
   7. Good practices

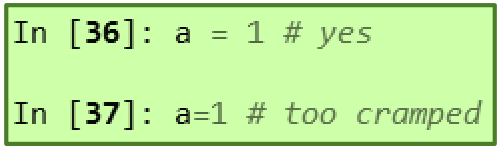
Style guidelines

- **Long lines:** you should not write very long lines that span over more than (e.g.) 80 characters. Long lines can be broken with the \ character



```
In [35]: long_line = "Here is a very very long line \
    ...: that we break in two parts."
```

- **Spaces:** Write well-spaced code: put whitespaces after commas, around arithmetic operators, etc.:



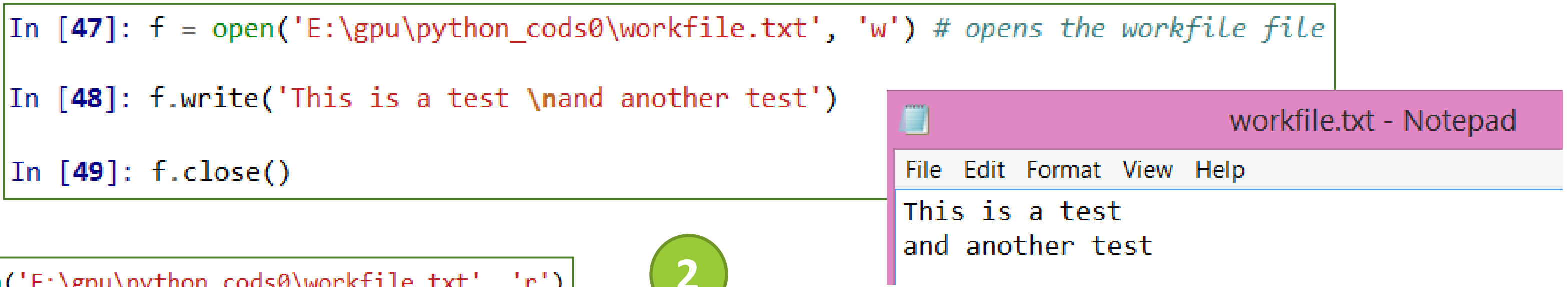
```
In [36]: a = 1 # yes

In [37]: a=1 # too cramped
```

Getting started with Python for science

7. Input and Output

We write or read **strings** to/from files (other types must be converted to strings). To write in a file:

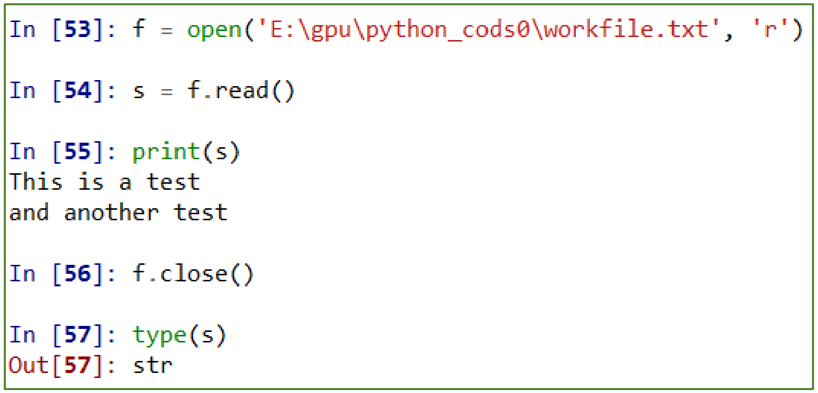
Write



```
In [47]: f = open('E:\gpu\python_cods0\workfile.txt', 'w') # opens the workfile file

In [48]: f.write('This is a test \nand another test')

In [49]: f.close()
```

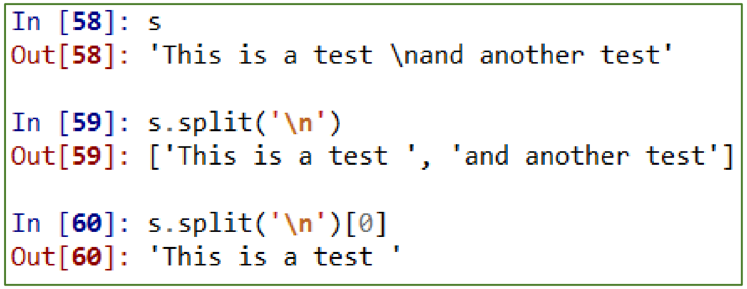
```
This is a test
and another test
```

1 **Read**



```
In [53]: f = open('E:\gpu\python_cods0\workfile.txt', 'r')

In [54]: s = f.read()

In [55]: print(s)
This is a test
and another test

In [56]: f.close()

In [57]: type(s)
Out[57]: str
```

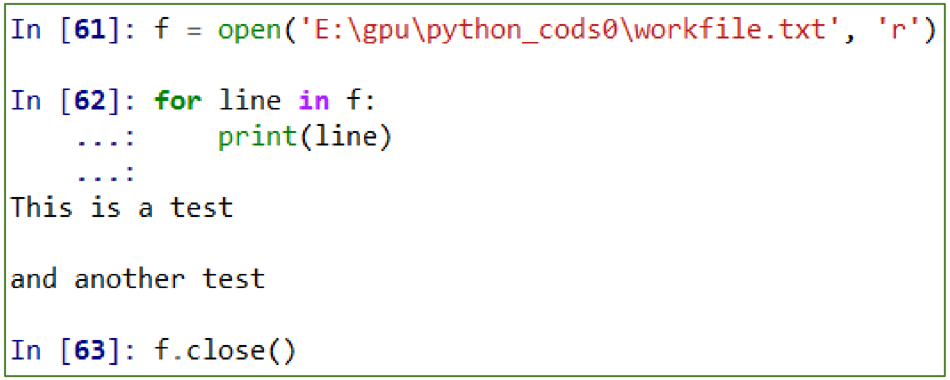
2



```
In [58]: s
Out[58]: 'This is a test \nand another test'

In [59]: s.split('\n')
Out[59]: ['This is a test ', 'and another test']

In [60]: s.split('\n')[0]
Out[60]: 'This is a test '
```

Getting started with Python for science

7. Input and Output
    1. Iterating over a file



```
In [61]: f = open('E:\gpu\python_cods0\workfile.txt', 'r')

In [62]: for line in f:
    ...:     print(line)
    ...:
This is a test

and another test

In [63]: f.close()
```

File modes

- Read-only: r
- Write-only: w
  - Note: Create a new file or overwrite existing file.
- Append a file: a
- Read and Write: r+
- Binary mode: b
  - Note: Use for binary files, especially on Windows.

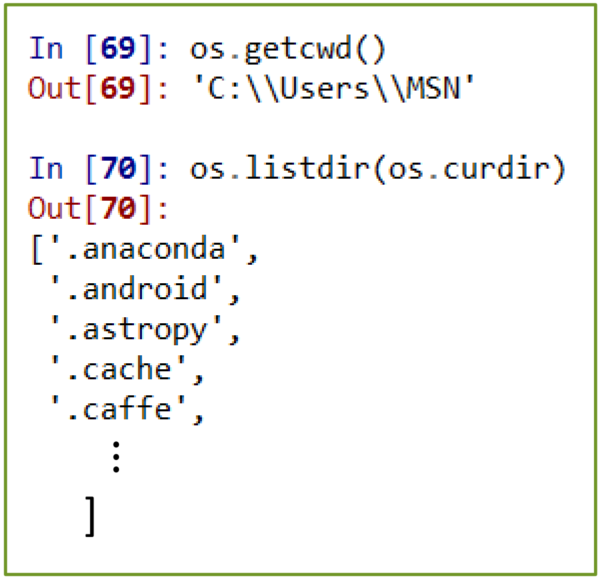
Getting started with Python for science

8. Standard Library

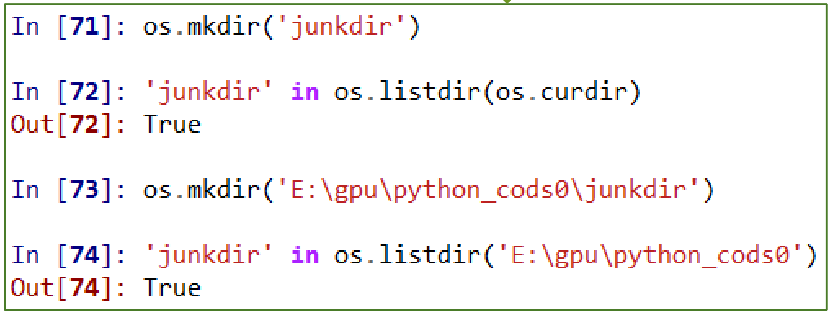
   1. os module: operating system functionality

Directory and file manipulation

- Current directory
- List a directory
- Make a directory



```
In [69]: os.getcwd()
Out[69]: 'C:\\Users\\MSN'

In [70]: os.listdir(os.curdir)
Out[70]:
['.anaconda',
 '.android',
 '.astropy',
 '.cache',
 '.caffe',
      ⋮
   ]
```



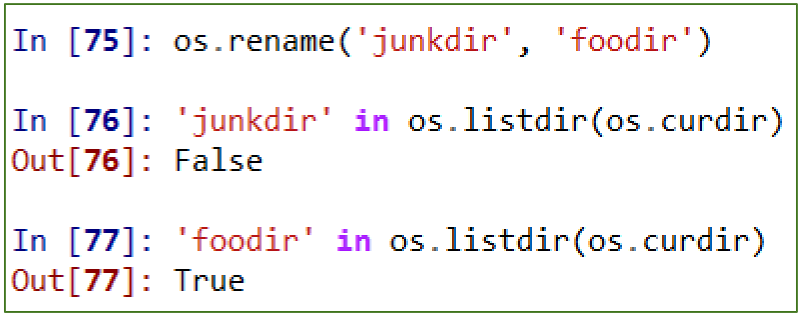
```
In [71]: os.mkdir('junkdir')

In [72]: 'junkdir' in os.listdir(os.curdir)
Out[72]: True

In [73]: os.mkdir('E:\gpu\python_cods0\junkdir')

In [74]: 'junkdir' in os.listdir('E:\gpu\python_cods0')
Out[74]: True
```

Getting started with Python for science

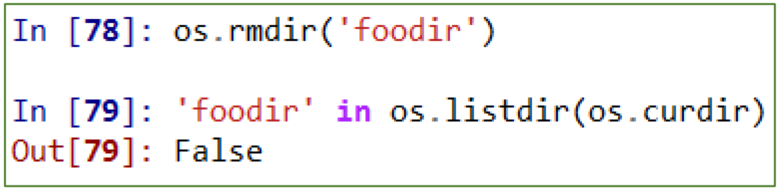
8. Standard Library
   1. os module: operating system functionality

Directory and file manipulation

- Rename the directory
- Remove directory



```
In [75]: os.rename('junkdir', 'foodir')

In [76]: 'junkdir' in os.listdir(os.curdir)
Out[76]: False

In [77]: 'foodir' in os.listdir(os.curdir)
Out[77]: True
```



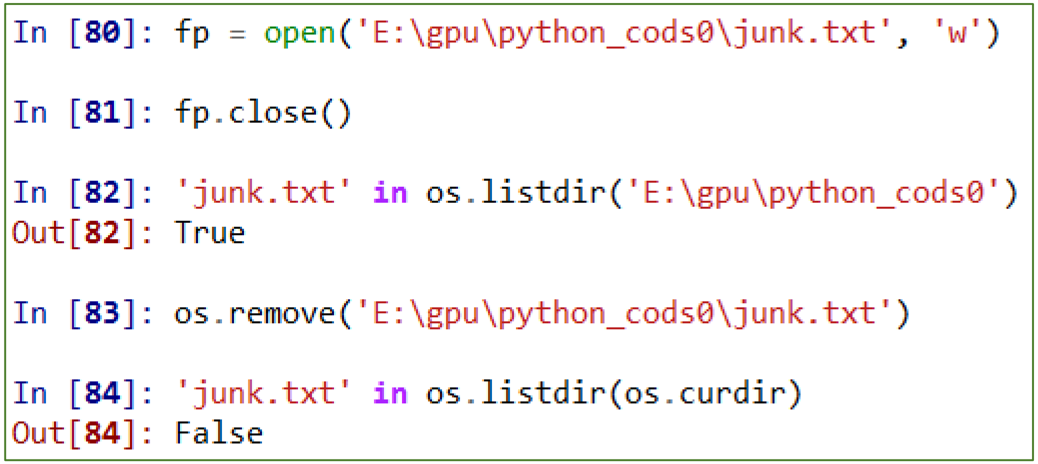
```
In [78]: os.rmdir('foodir')

In [79]: 'foodir' in os.listdir(os.curdir)
Out[79]: False
```

Getting started with Python for science

8. Standard Library
   1. os module: operating system functionality

Directory and file manipulation

- Delete a file

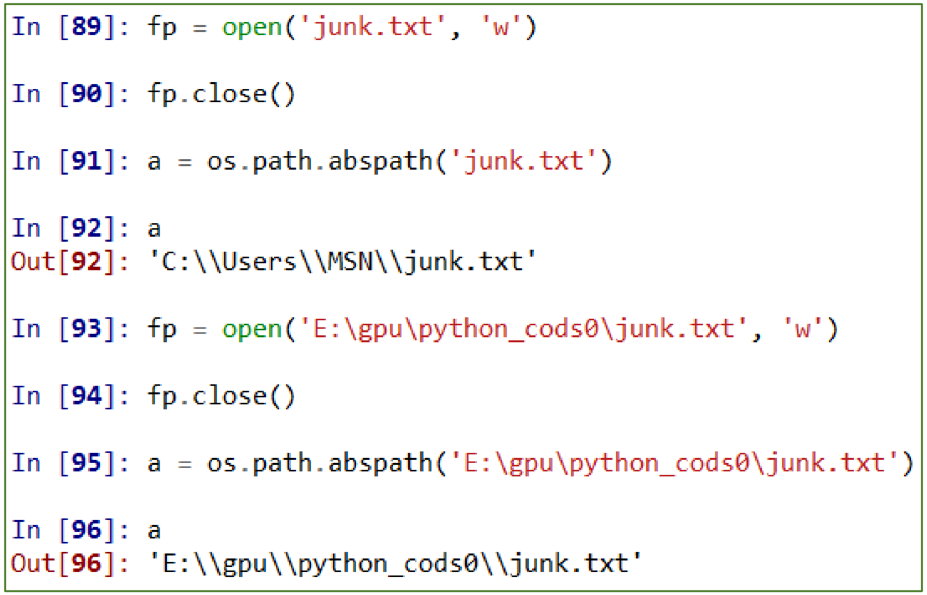


```
In [80]: fp = open('E:\gpu\python_cods0\junk.txt', 'w')

In [81]: fp.close()

In [82]: 'junk.txt' in os.listdir('E:\gpu\python_cods0')
Out[82]: True

In [83]: os.remove('E:\gpu\python_cods0\junk.txt')

In [84]: 'junk.txt' in os.listdir(os.curdir)
Out[84]: False
```

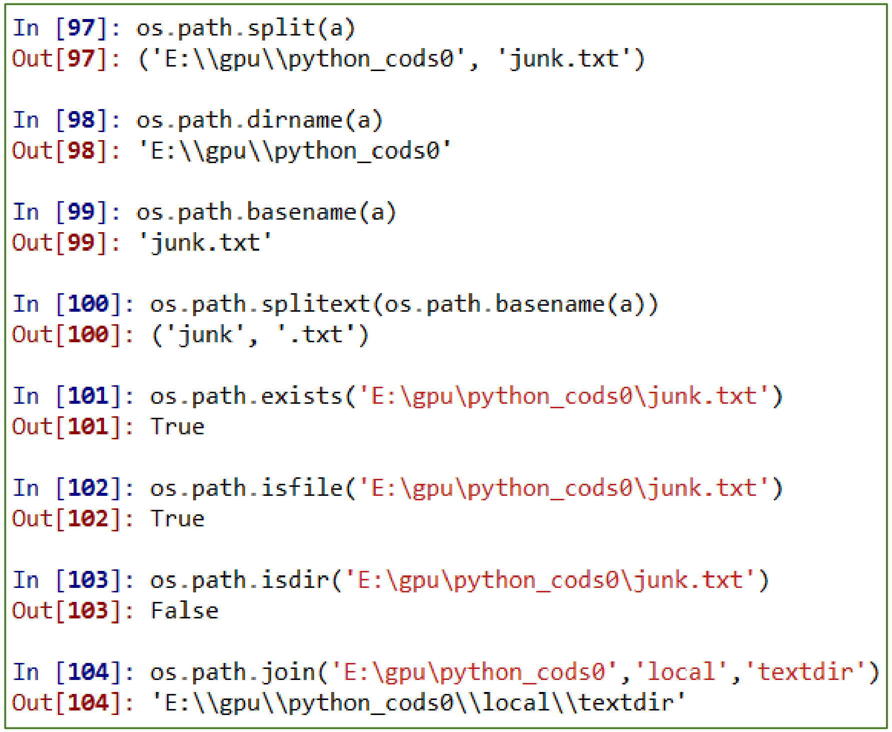
Getting started with Python for science

8. Standard Library
   1. os module: operating system functionality

os.path: path manipulations



```
In [89]: fp = open('junk.txt', 'w')

In [90]: fp.close()

In [91]: a = os.path.abspath('junk.txt')

In [92]: a
Out[92]: 'C:\\Users\\MSN\\junk.txt'

In [93]: fp = open('E:\gpu\python_cods0\junk.txt', 'w')

In [94]: fp.close()

In [95]: a = os.path.abspath('E:\gpu\python_cods0\junk.txt')

In [96]: a
Out[96]: 'E:\\gpu\\python_cods0\\junk.txt'
```

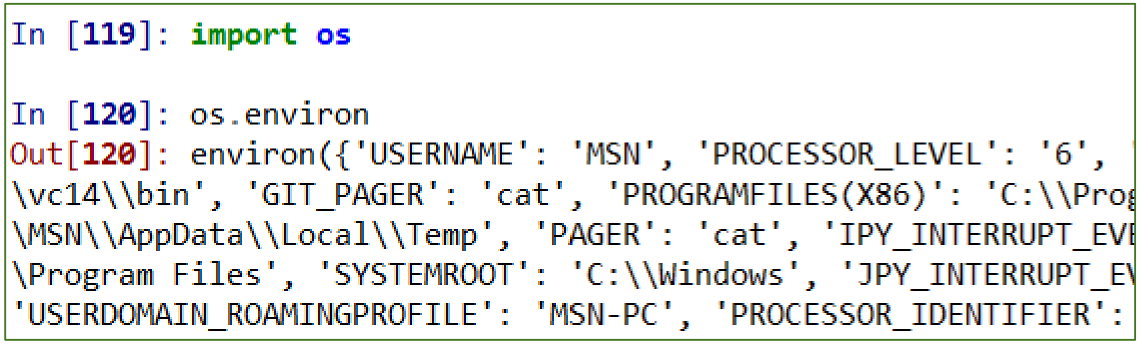


```
In [97]: os.path.split(a)
Out[97]: ('E:\\gpu\\python_cods0', 'junk.txt')

In [98]: os.path.dirname(a)
Out[98]: 'E:\\gpu\\python_cods0'

In [99]: os.path.basename(a)
Out[99]: 'junk.txt'

In [100]: os.path.splitext(os.path.basename(a))
Out[100]: ('junk', '.txt')

In [101]: os.path.exists('E:\gpu\python_cods0\junk.txt')
Out[101]: True

In [102]: os.path.isfile('E:\gpu\python_cods0\junk.txt')
Out[102]: True

In [103]: os.path.isdir('E:\gpu\python_cods0\junk.txt')
Out[103]: False

In [104]: os.path.join('E:\gpu\python_cods0','local','textdir')
Out[104]: 'E:\\gpu\\python_cods0\\local\\textdir'
```

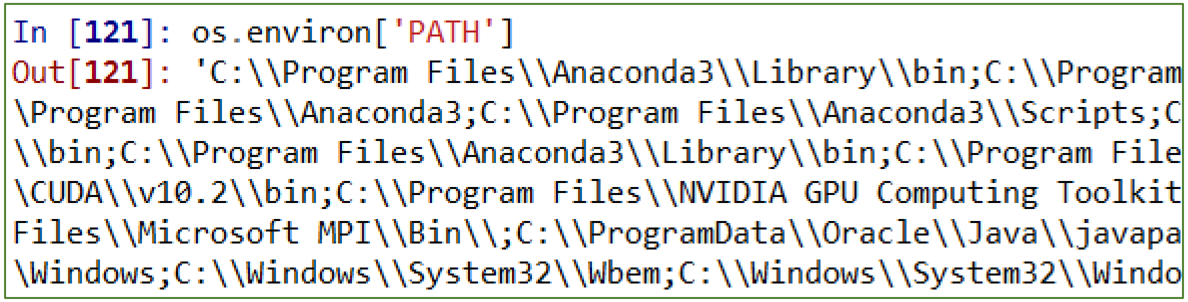
Getting started with Python for science

8. Standard Library
   1. os module: operating system functionality

Environment variables



```
In [119]: import os

In [120]: os.environ
Out[120]: environ({'USERNAME': 'MSN', 'PROCESSOR_LEVEL': '6', '
\vc14\\bin', 'GIT_PAGER': 'cat', 'PROGRAMFILES(X86)': 'C:\\Prog
\MSN\\AppData\\Local\\Temp', 'PAGER': 'cat', 'IPY_INTERRUPT_EVE
\Program Files', 'SYSTEMROOT': 'C:\\Windows', 'JPY_INTERRUPT_EV
'USERDOMAIN_ROAMINGPROFILE': 'MSN-PC', 'PROCESSOR_IDENTIFIER':
```



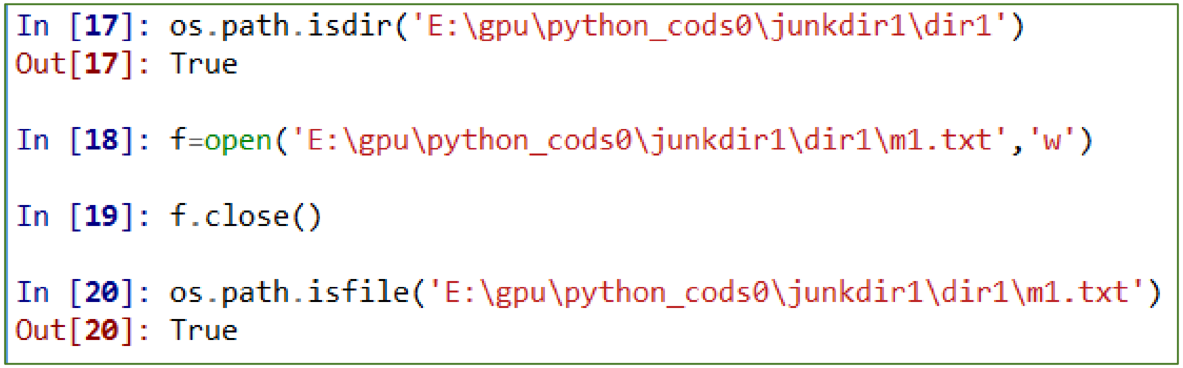
```
In [121]: os.environ['PATH']
Out[121]: 'C:\\Program Files\\Anaconda3\\Library\\bin;C:\\Program
\Program Files\\Anaconda3;C:\\Program Files\\Anaconda3\\Scripts;C
\\bin;C:\\Program Files\\Anaconda3\\Library\\bin;C:\\Program File
\CUDA\\v10.2\\bin;C:\\Program Files\\NVIDIA GPU Computing Toolkit
Files\\Microsoft MPI\\Bin\\;C:\\ProgramData\\Oracle\\Java\\javapa
\Windows;C:\\Windows\\System32\\Wbem;C:\\Windows\\System32\\Windo
```

Getting started with Python for science

8. Standard Library
   2. shutil: high-level file operations

The shutil provides useful file operations:

- shutil.rmtree: Recursively delete a directory tree.
- shutil.move: Recursively move a file or directory to another location.
- shutil.copy: Copy files or directories.



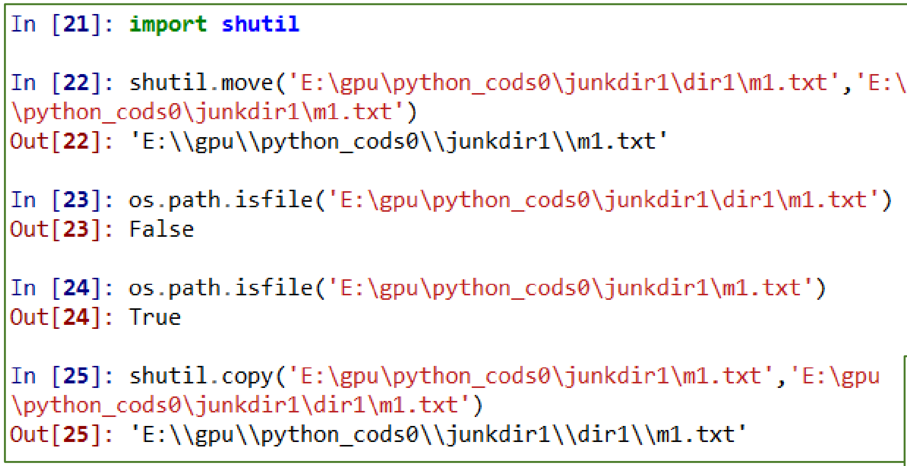
```
In [17]: os.path.isdir('E:\gpu\python_cods0\junkdir1\dir1')
Out[17]: True

In [18]: f=open('E:\gpu\python_cods0\junkdir1\dir1\m1.txt','w')

In [19]: f.close()

In [20]: os.path.isfile('E:\gpu\python_cods0\junkdir1\dir1\m1.txt')
Out[20]: True
```

Getting started with Python for science

8. Standard Library

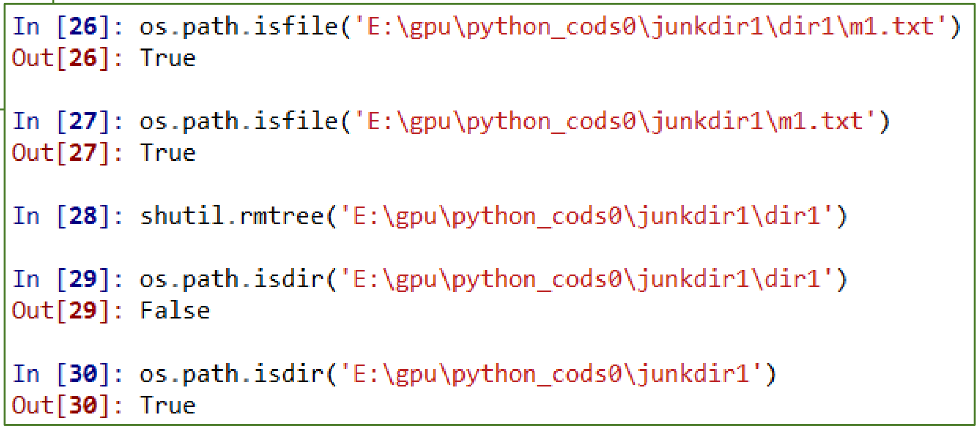
   2. shutil: high-level file operations

1



```
In [21]: import shutil

In [22]: shutil.move('E:\gpu\python_cods0\junkdir1\dir1\m1.txt','E:\gpu
\python_cods0\junkdir1\m1.txt')
Out[22]: 'E:\\gpu\\python_cods0\\junkdir1\\m1.txt'

In [23]: os.path.isfile('E:\gpu\python_cods0\junkdir1\dir1\m1.txt')
Out[23]: False

In [24]: os.path.isfile('E:\gpu\python_cods0\junkdir1\m1.txt')
Out[24]: True

In [25]: shutil.copy('E:\gpu\python_cods0\junkdir1\m1.txt','E:\gpu
\python_cods0\junkdir1\dir1\m1.txt')
Out[25]: 'E:\\gpu\\python_cods0\\junkdir1\\dir1\\m1.txt'
```

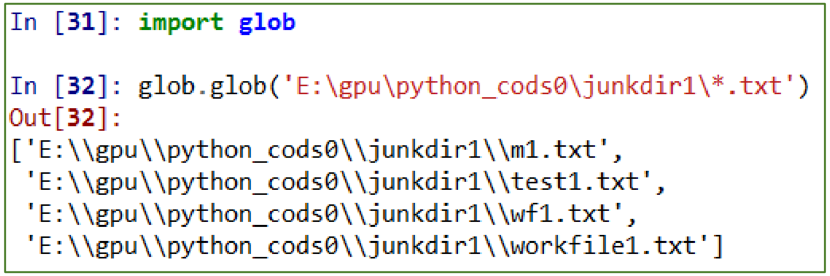
2



```
In [26]: os.path.isfile('E:\gpu\python_cods0\junkdir1\dir1\m1.txt')
Out[26]: True

In [27]: os.path.isfile('E:\gpu\python_cods0\junkdir1\m1.txt')
Out[27]: True

In [28]: shutil.rmtree('E:\gpu\python_cods0\junkdir1\dir1')

In [29]: os.path.isdir('E:\gpu\python_cods0\junkdir1\dir1')
Out[29]: False

In [30]: os.path.isdir('E:\gpu\python_cods0\junkdir1')
Out[30]: True
```

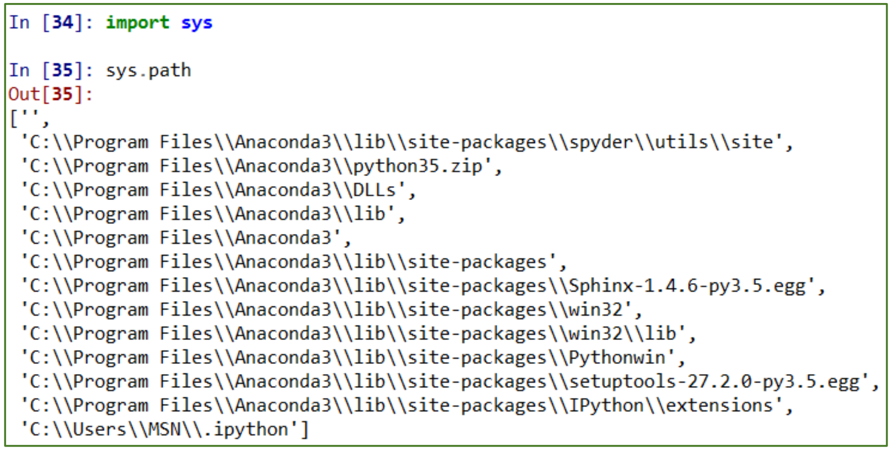
Getting started with Python for science

8. Standard Library
   3. glob: Pattern matching on files



```
In [31]: import glob

In [32]: glob.glob('E:\gpu\python_cods0\junkdir1\*.txt')
Out[32]:
['E:\\gpu\\python_cods0\\junkdir1\\m1.txt',
 'E:\\gpu\\python_cods0\\junkdir1\\test1.txt',
 'E:\\gpu\\python_cods0\\junkdir1\\wf1.txt',
 'E:\\gpu\\python_cods0\\junkdir1\\workfile1.txt']
```

   4. sys module: system-specific information

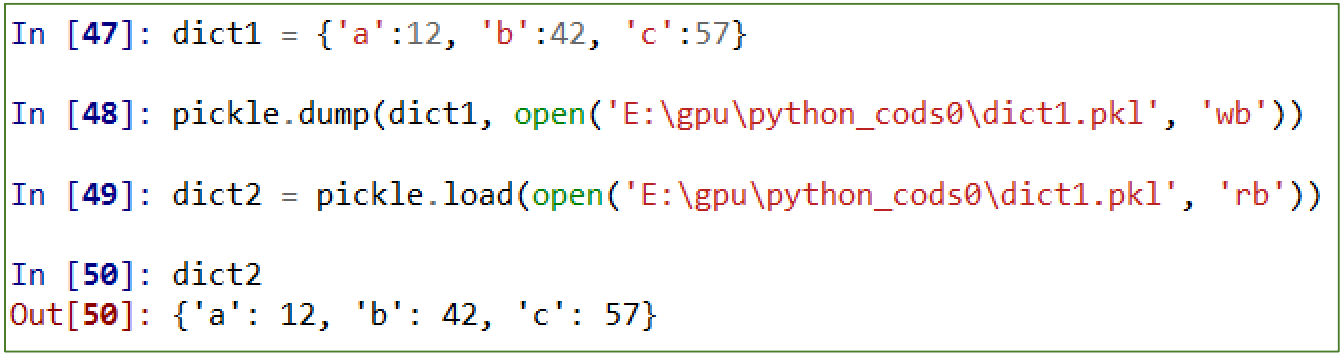


```
In [34]: import sys

In [35]: sys.path
Out[35]:
['',
 'C:\\Program Files\\Anaconda3\\lib\\site-packages\\spyder\\utils\\site',
 'C:\\Program Files\\Anaconda3\\python35.zip',
 'C:\\Program Files\\Anaconda3\\DLLs',
 'C:\\Program Files\\Anaconda3\\lib',
 'C:\\Program Files\\Anaconda3',
 'C:\\Program Files\\Anaconda3\\lib\\site-packages',
 'C:\\Program Files\\Anaconda3\\lib\\site-packages\\Sphinx-1.4.6-py3.5.egg',
 'C:\\Program Files\\Anaconda3\\lib\\site-packages\\win32',
 'C:\\Program Files\\Anaconda3\\lib\\site-packages\\win32\\lib',
 'C:\\Program Files\\Anaconda3\\lib\\site-packages\\Pythonwin',
 'C:\\Program Files\\Anaconda3\\lib\\site-packages\\setuptools-27.2.0-py3.5.egg',
 'C:\\Program Files\\Anaconda3\\lib\\site-packages\\IPython\\extensions',
 'C:\\Users\\MSN\\.ipython']
```

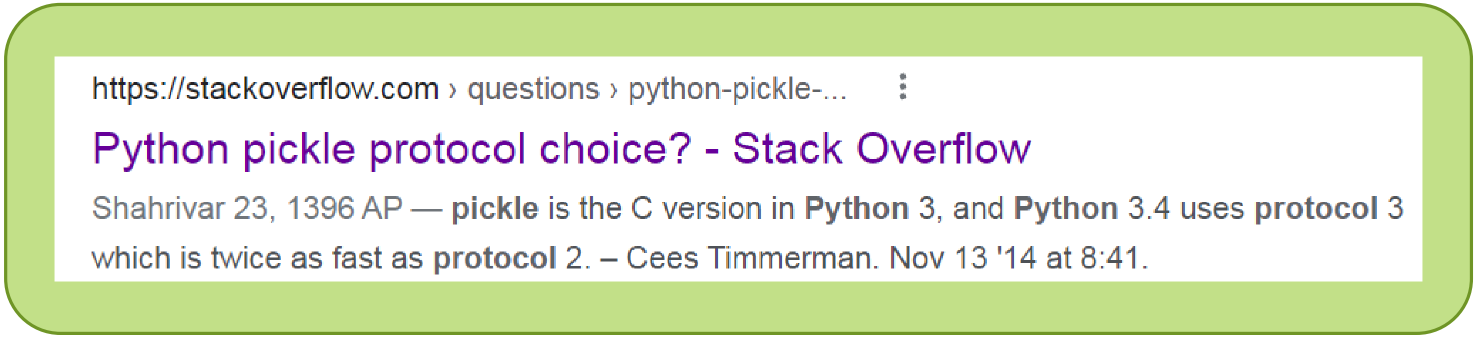
sys.path is a list of strings that specifies the search path for modules. Initialized from PYTHONPATH:

Getting started with Python for science

8. Standard Library
   5. pickle: easy persistence



```
In [47]: dict1 = {'a':12, 'b':42, 'c':57}

In [48]: pickle.dump(dict1, open('E:\gpu\python_cods0\dict1.pkl', 'wb'))

In [49]: dict2 = pickle.load(open('E:\gpu\python_cods0\dict1.pkl', 'rb'))

In [50]: dict2
Out[50]: {'a': 12, 'b': 42, 'c': 57}
```



https://stackoverflow.com › questions › python-pickle-...

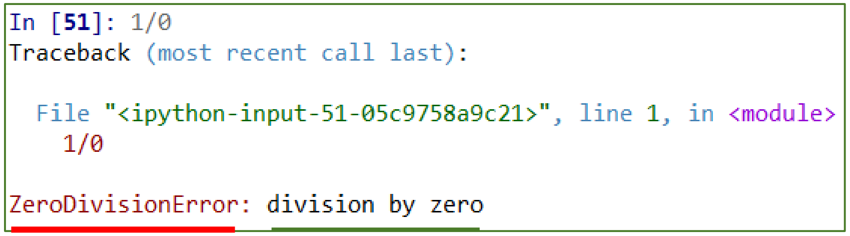
Python pickle protocol choice? - Stack Overflow

Shahrivar 23, 1396 AP — **pickle** is the C version in **Python** 3, and **Python** 3.4 uses **protocol** 3 which is twice as fast as **protocol** 2. – Cees Timmerman. Nov 13 '14 at 8:41.

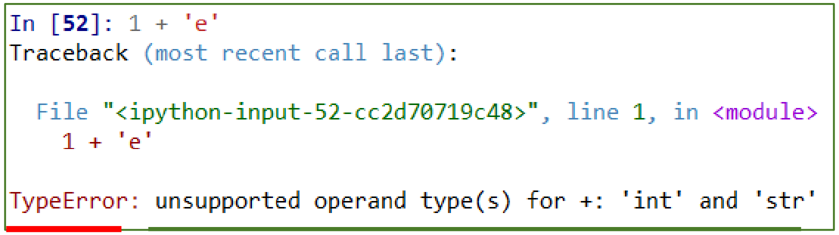
Getting started with Python for science

9. Exception handling in Python
   1. Exceptions

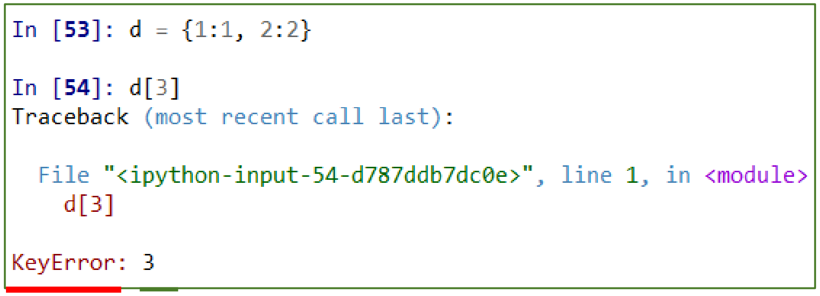
Exceptions are raised by errors in Python.



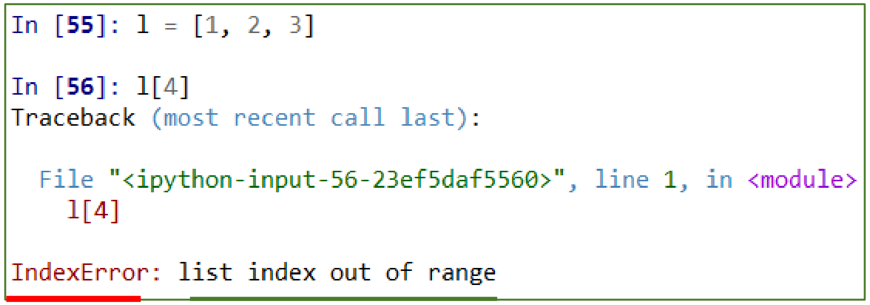
```
In [51]: 1/0
Traceback (most recent call last):

  File "<ipython-input-51-05c9758a9c21>", line 1, in <module>
    1/0

ZeroDivisionError: division by zero
```



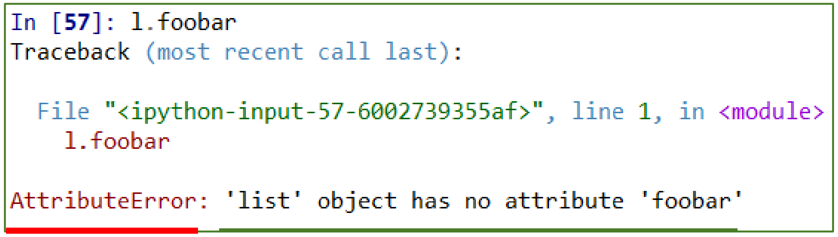
```
In [52]: 1 + 'e'
Traceback (most recent call last):

  File "<ipython-input-52-cc2d70719c48>", line 1, in <module>
    1 + 'e'

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```



```
In [53]: d = {1:1, 2:2}

In [54]: d[3]
Traceback (most recent call last):

  File "<ipython-input-54-d787ddb7dc0e>", line 1, in <module>
    d[3]

KeyError: 3
```



```
In [55]: l = [1, 2, 3]

In [56]: l[4]
Traceback (most recent call last):

  File "<ipython-input-56-23ef5daf5560>", line 1, in <module>
    l[4]

IndexError: list index out of range
```

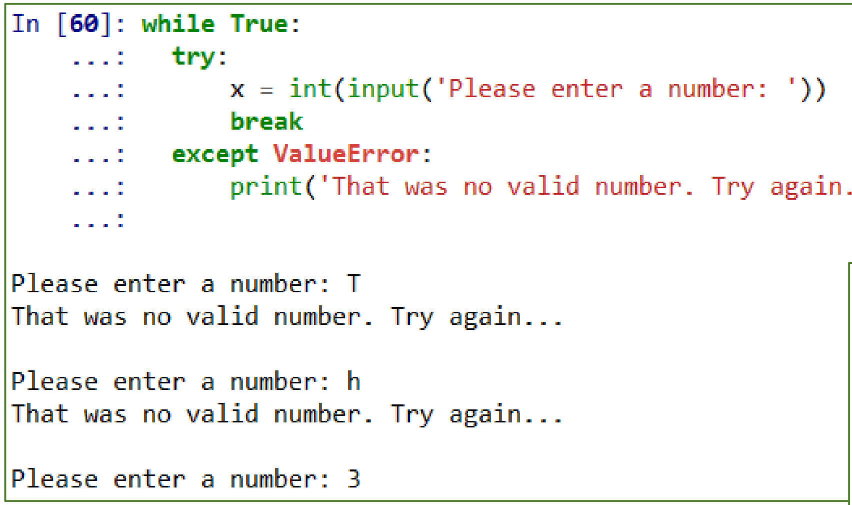


```
In [57]: l.foobar
Traceback (most recent call last):

  File "<ipython-input-57-6002739355af>", line 1, in <module>
    l.foobar

AttributeError: 'list' object has no attribute 'foobar'
```

There are different types of **exceptions** for different **errors**.

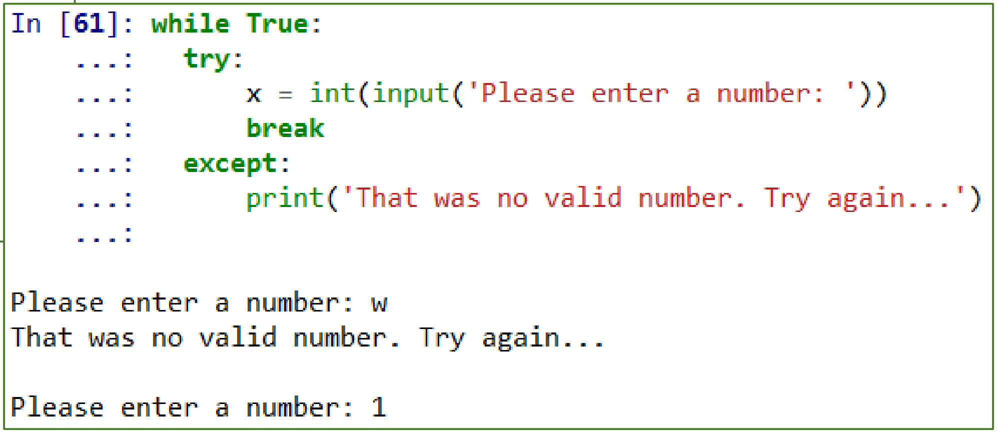
Getting started with Python for science

9. Exception handling in Python
   2. Catching exceptions



```
In [60]: while True:
    ...:     try:
    ...:         x = int(input('Please enter a number: '))
    ...:         break
    ...:     except ValueError:
    ...:         print('That was no valid number. Try again...')
    ...:

Please enter a number: T
That was no valid number. Try again...

Please enter a number: h
That was no valid number. Try again...

Please enter a number: 3
```

try/except

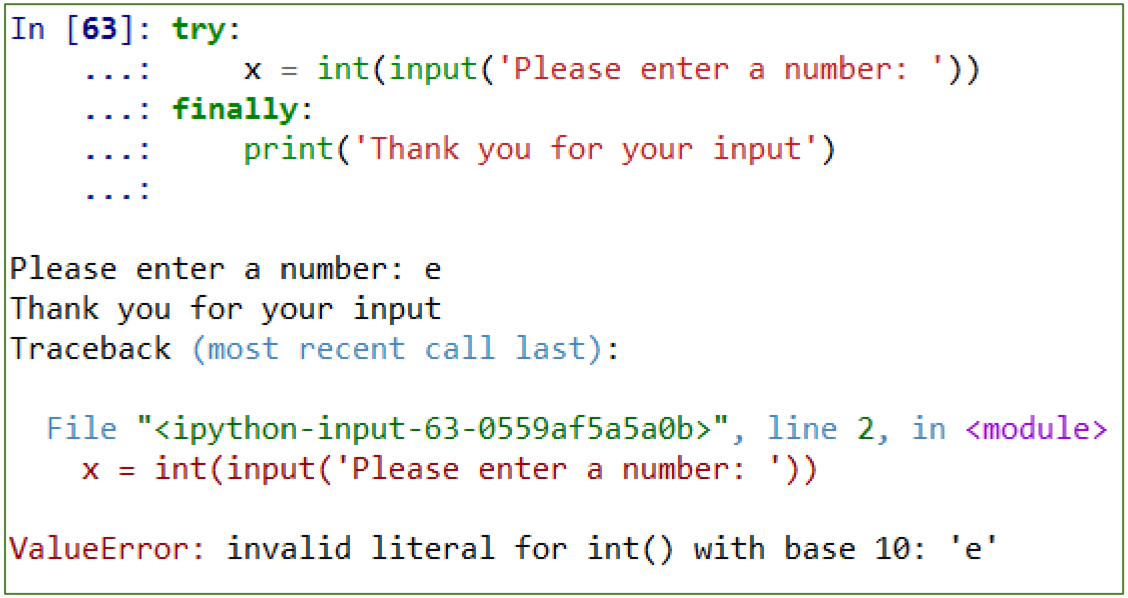


```
In [61]: while True:
    ...:     try:
    ...:         x = int(input('Please enter a number: '))
    ...:         break
    ...:     except:
    ...:         print('That was no valid number. Try again...')
    ...:

Please enter a number: w
That was no valid number. Try again...

Please enter a number: 1
```

Getting started with Python for science

9. Exception handling in Python
   2. Catching exceptions

try/finally



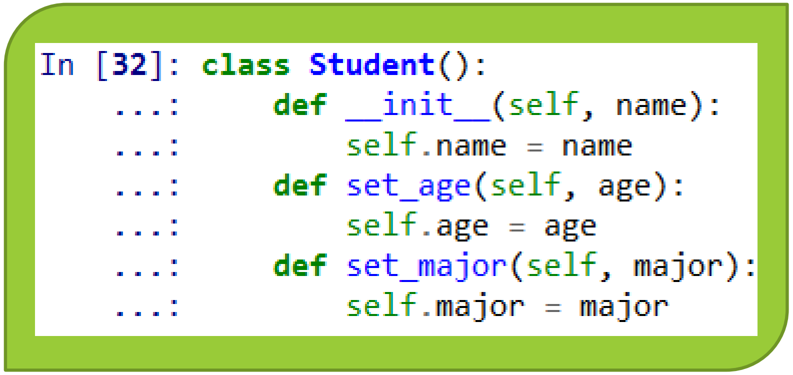
```
In [63]: try:
    ...:     x = int(input('Please enter a number: '))
    ...: finally:
    ...:     print('Thank you for your input')
    ...:

Please enter a number: e
Thank you for your input
Traceback (most recent call last):

  File "<ipython-input-63-0559af5a5a0b>", line 2, in <module>
    x = int(input('Please enter a number: '))

ValueError: invalid literal for int() with base 10: 'e'
```

Getting started with Python for science

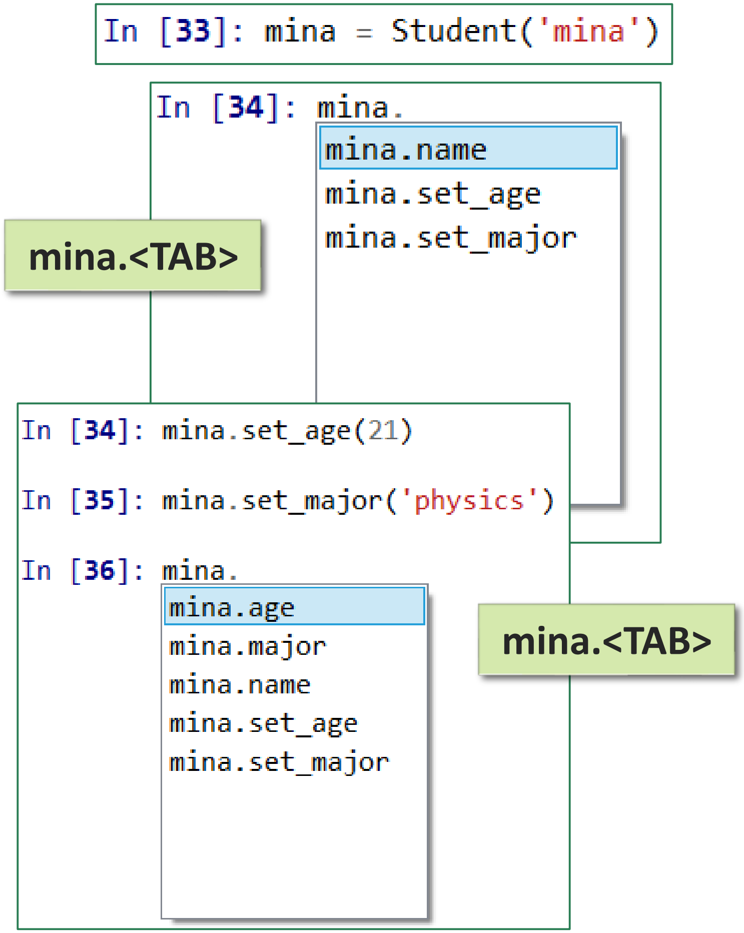
10. Object-oriented programming (OOP)

The goals of OOP are:

- to organize the code, and
- to re-use code in similar contexts.



```
In [32]: class Student():
    ...:     def __init__(self, name):
    ...:         self.name = name
    ...:     def set_age(self, age):
    ...:         self.age = age
    ...:     def set_major(self, major):
    ...:         self.major = major
```



```
In [33]: mina = Student('mina')
```

```
In [34]: mina.
          mina.name
          mina.set_age
          mina.set_major
```

mina.<TAB>

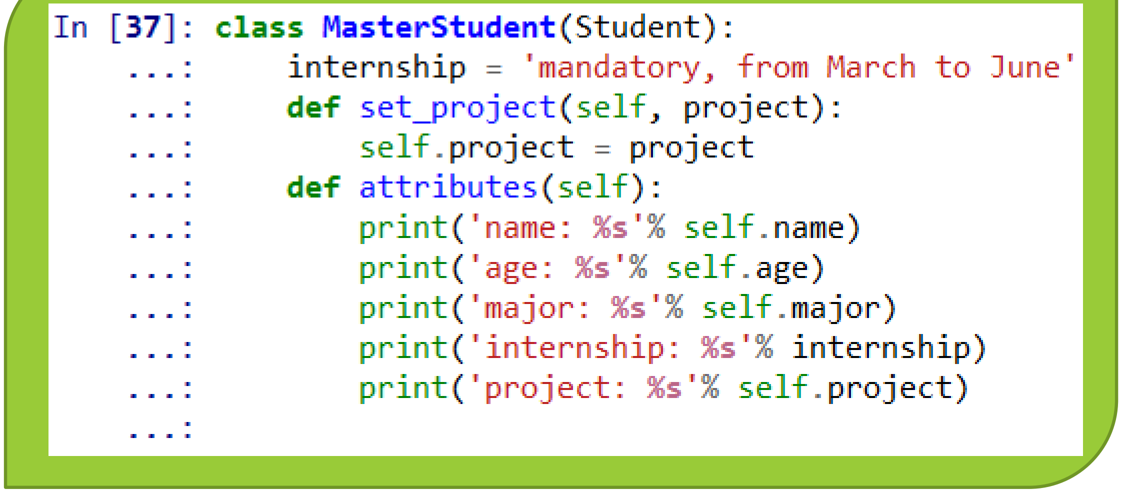
```
In [34]: mina.set_age(21)

In [35]: mina.set_major('physics')

In [36]: mina.
          mina.age
          mina.major
          mina.name
          mina.set_age
          mina.set_major
```

mina.<TAB>

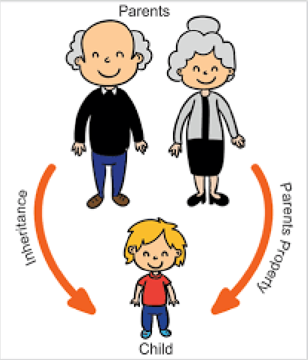
Getting started with Python for science

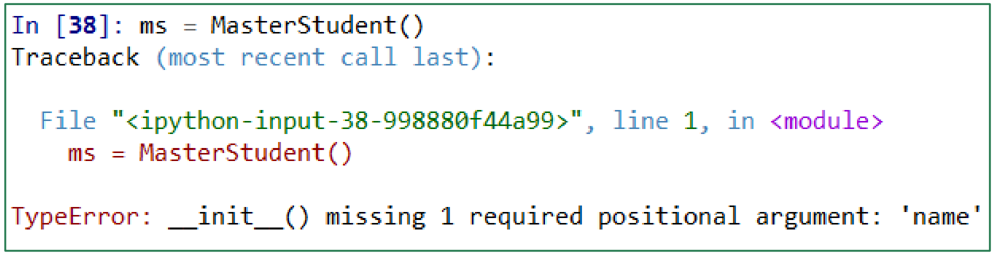
10. Object-oriented programming (OOP)

Inheritance

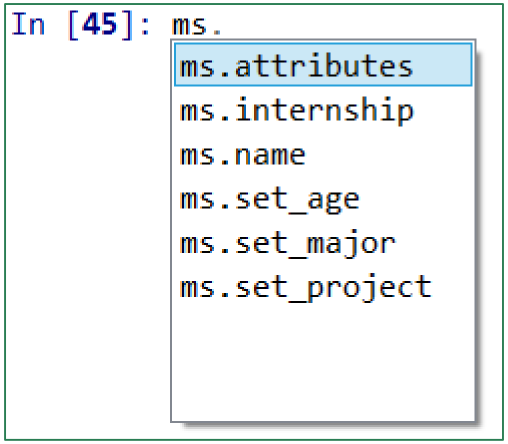


```
In [37]: class MasterStudent(Student):
    ...:     internship = 'mandatory, from March to June'
    ...:     def set_project(self, project):
    ...:         self.project = project
    ...:     def attributes(self):
    ...:         print('name: %s'% self.name)
    ...:         print('age: %s'% self.age)
    ...:         print('major: %s'% self.major)
    ...:         print('internship: %s'% internship)
    ...:         print('project: %s'% self.project)
    ...:
```





```
In [38]: ms = MasterStudent()
Traceback (most recent call last):

  File "<ipython-input-38-998880f44a99>", line 1, in <module>
    ms = MasterStudent()

TypeError: __init__() missing 1 required positional argument: 'name'
```

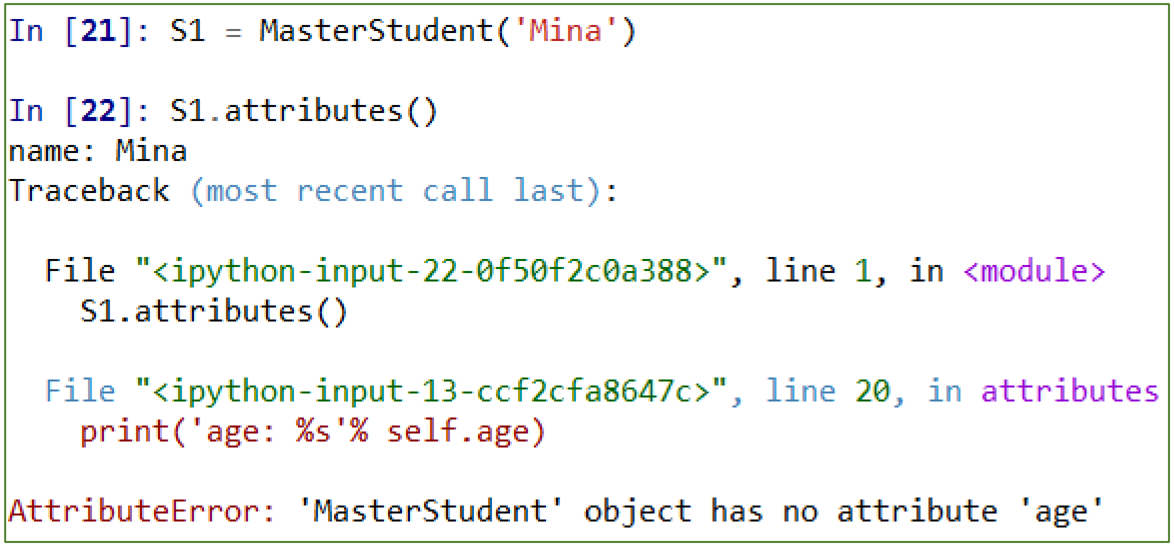


```
In [45]: ms.
         ms.attributes
         ms.internship
         ms.name
         ms.set_age
         ms.set_major
         ms.set_project
```

Getting started with Python for science

10. Object-oriented programming (OOP)

3. Solve the problem



```
In [21]: S1 = MasterStudent('Mina')

In [22]: S1.attributes()
name: Mina
Traceback (most recent call last):

  File "<ipython-input-22-0f50f2c0a388>", line 1, in <module>
    S1.attributes()

  File "<ipython-input-13-ccf2cfa8647c>", line 20, in attributes
    print('age: %s'% self.age)

AttributeError: 'MasterStudent' object has no attribute 'age'
```

Getting started with Python for science

4. Write a program that takes information of the ten students from the user as follows:

Degree Level:

Student identification number (SID):

Name:

Last name:

Age:

Major:

National ID:

A list of elective courses, grades and units in the previous semester:

Getting started with Python for science

- In case of receiving wrong data, the appropriate message should be given by the system.
- Create three classes for three groups of students (Bachelor, Master, Doctoral) using inheritance law in the form of a module so that the upper class has more attributes (at least two) than the lower class.
- Calculate the grade point average (GPA) of the previous semester for each student by defining a function in the base class.
- Define two functions in the base class to find the minimum and maximum score of the previous semester with the name of the course.
- Arrange the received information in the form of objects using the defined classes.

Getting started with Python for science

- Arrange the three categories of objects using lists and a dictionary, and finally, save the dictionary as a pickle file.
- Create 'student_info\BSc‘, 'student_info\MSc‘, 'student_info\PhD‘ directories.
- For each student, create a text file with the name of SID in the appropriate path and save him/her information in it.
- Using the glob module, retrieve all information in the text files and rearrange them in the form of lists and a dictionary.
- Remove all created files and directories.
- The various steps should be written regularly in the form of functions and a script file. Functions are applicable if the program runs directly.