MENG 25510 – A Short Background in Numerical Programming

Spring 2022

Coding Final Project

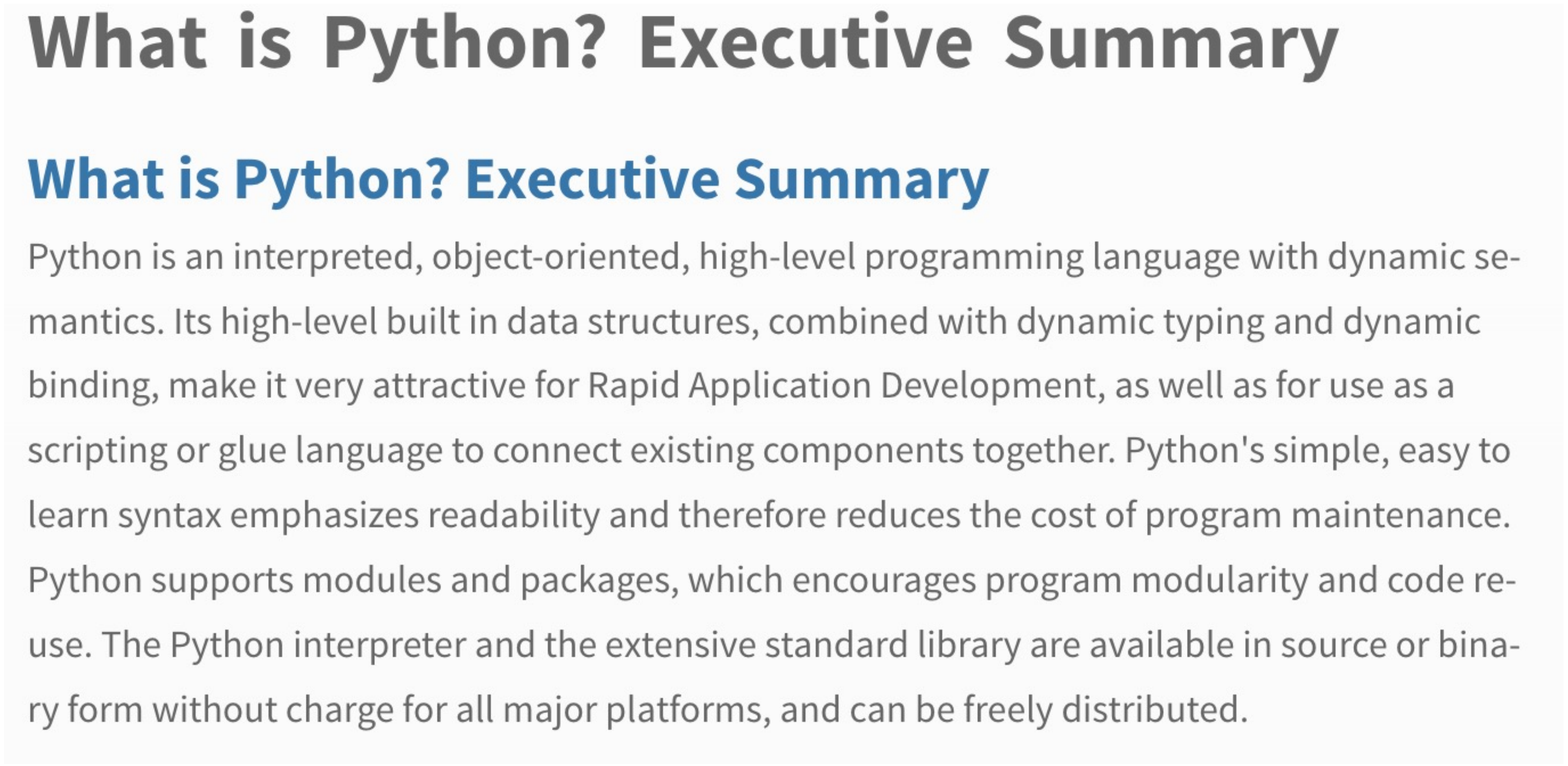
- These slides are for those doing the HF/DFT coding option for the final project who might want a quick background in numerical computing before getting started.
- This is just a summary of useful info (and where to look for more detailed explanations) and not an in-depth guide by any means.

Which language/software to use?

- Will primarily focus on Python and associated tools here.
- Very widely used in scientific computing! (easy to utilize old, fast code from C/C++,Fortan,etc from inside Python)

jupyter

Why Python?



What is Python? Executive Summary

What is Python? Executive Summary

Python is an interpreted, object-oriented, high-level programming language with dynamic se-mantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code re-use. The Python interpreter and the extensive standard library are available in source or bina-ry form without charge for all major platforms, and can be freely distributed.

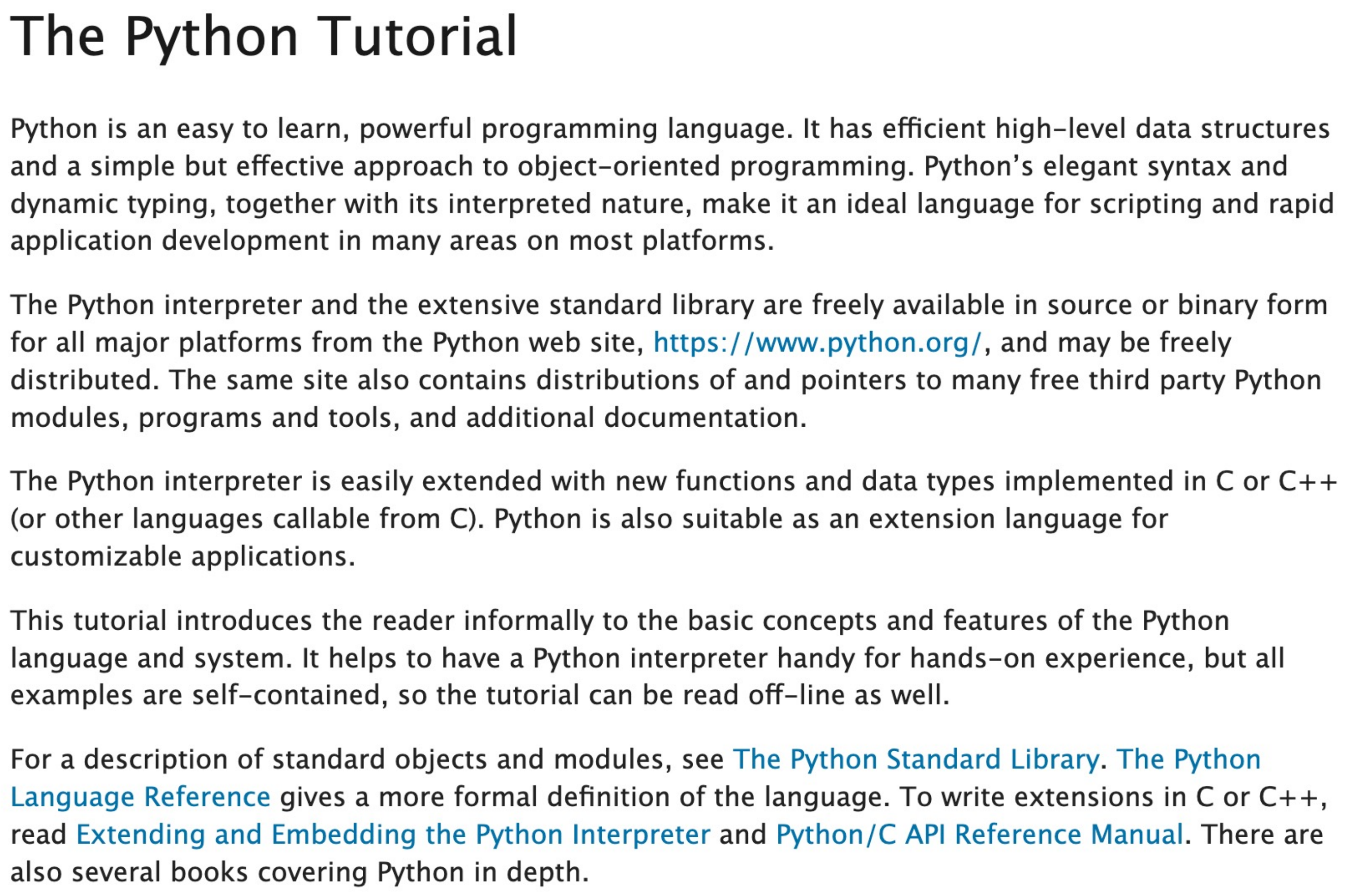
https://www.python.org/doc/essays/blurb/

Core Concepts

The Full Docs

The below website has the official Python documentation that outlines features of the language and how to effectively use it

A Summary of important concepts will follow



The Python Tutorial

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.

The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python web site, https://www.python.org/, and may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation.

The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications.

This tutorial introduces the reader informally to the basic concepts and features of the Python language and system. It helps to have a Python interpreter handy for hands-on experience, but all examples are self-contained, so the tutorial can be read off-line as well.

For a description of standard objects and modules, see The Python Standard Library. The Python Language Reference gives a more formal definition of the language. To write extensions in C or C++, read Extending and Embedding the Python Interpreter and Python/C API Reference Manual. There are also several books covering Python in depth.

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

Programming Paradigm

- Among other things, Python is designed so you can write procedurally, or step-by-step
- I.e. it is easy to translate the specific actions/operations you would like to perform on your data into line-by-line statements and expressions.
- Example: Change a real number ”x” to be the sum of the square root of “x” and 2, then print the new value of “x”

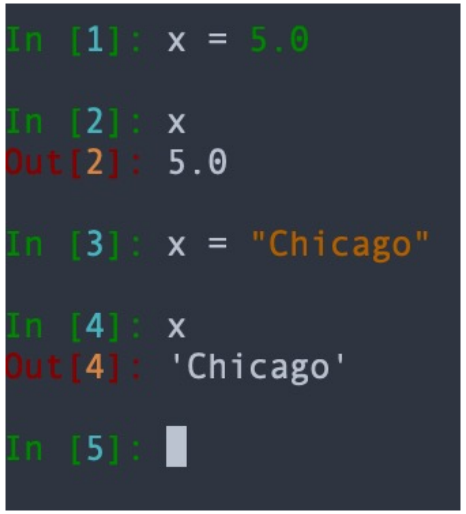
```
x = 2.71
x = np.sqrt(x)+ 2
print(x) #x has changed!
```

Notice that in line 2, RHS is computed *first* and then stored in the LHS.

Variables

- In many other languages, variable types are strictly declared for every variable you use before you can run your code (static typing).
- In Python, this is not the case – variables can be used more freely and you do not have to specify "int", "float", "string", etc before you can use them.

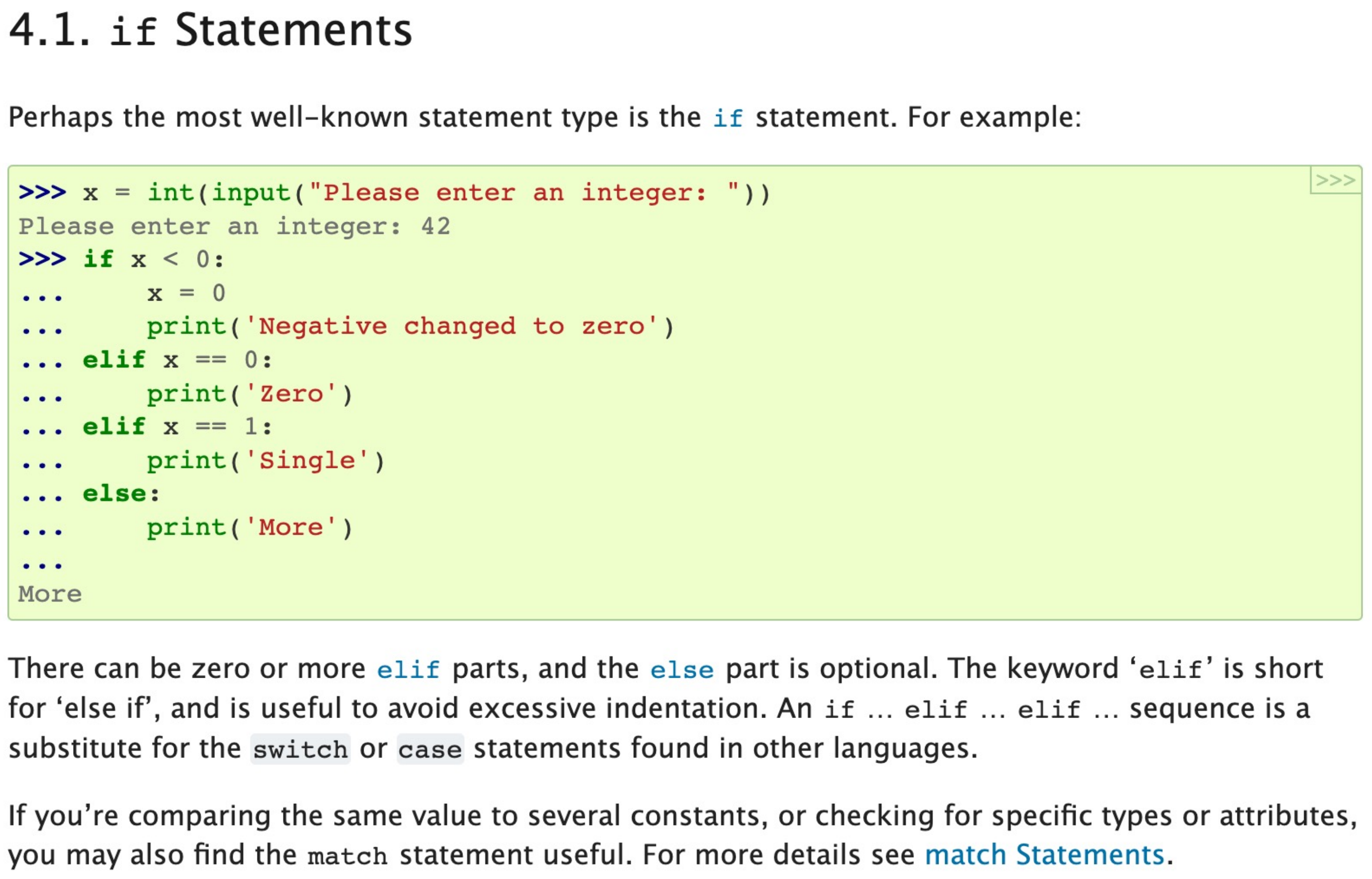
```
x = 5.0 # x is a float
x = "Chicago" # now a string
```



```
In [1]: x = 5.0

In [2]: x
Out[2]: 5.0

In [3]: x = "Chicago"

In [4]: x
Out[4]: 'Chicago'

In [5]: 
```

- Just make sure to keep track of what type you want each variable to have!

Control Statements



4.1. `if` Statements

Perhaps the most well-known statement type is the `if` statement. For example:

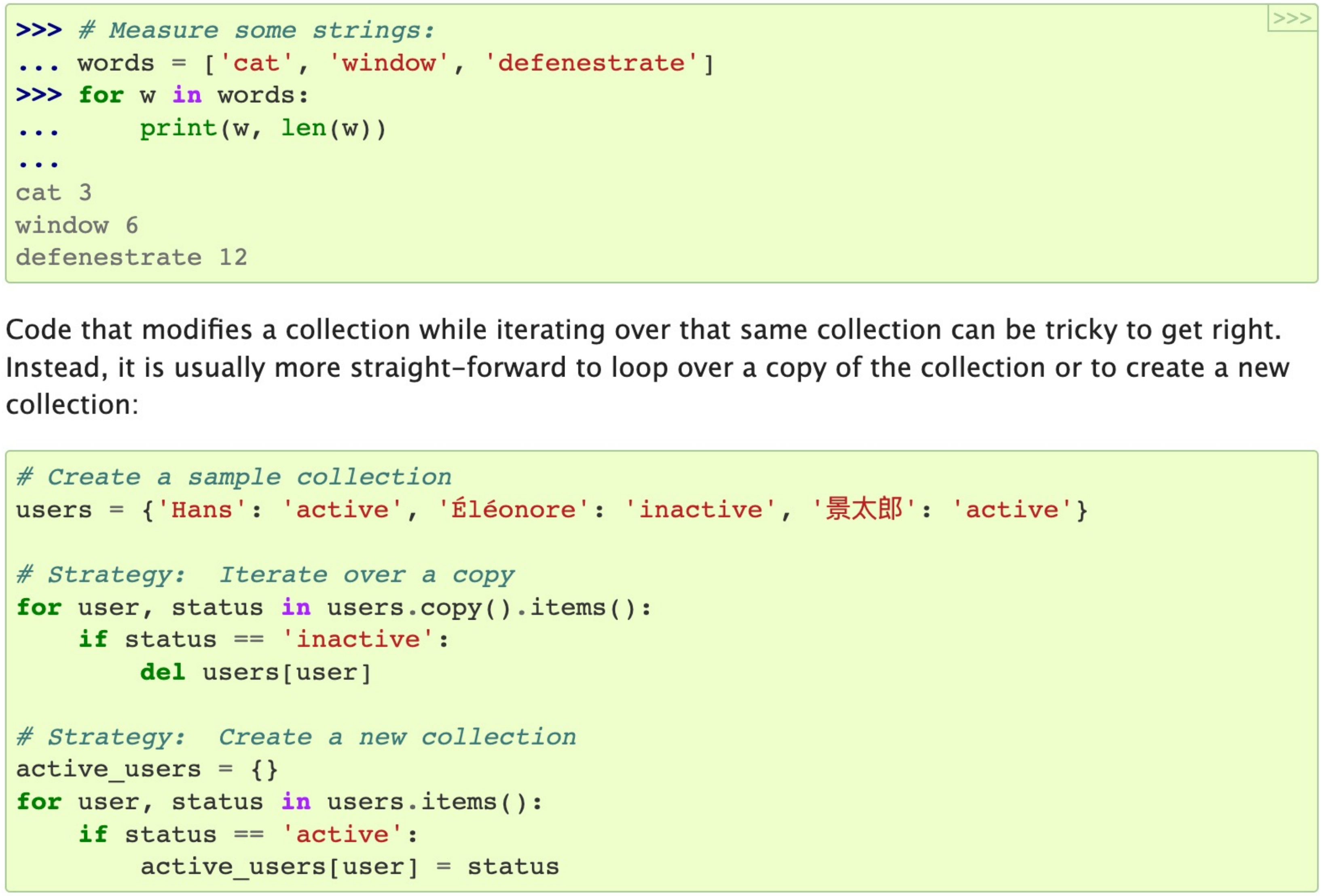
```
>>> x = int(input("Please enter an integer: "))
Please enter an integer: 42
>>> if x < 0:
...     x = 0
...     print('Negative changed to zero')
... elif x == 0:
...     print('Zero')
... elif x == 1:
...     print('Single')
... else:
...     print('More')
...
More
```

There can be zero or more `elif` parts, and the `else` part is optional. The keyword ‘`elif`’ is short for ‘else if’, and is useful to avoid excessive indentation. An `if` … `elif` … `elif` … sequence is a substitute for the `switch` or `case` statements found in other languages.

If you’re comparing the same value to several constants, or checking for specific types or attributes, you may also find the `match` statement useful. For more details see match Statements.

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

Control Statements – for loops



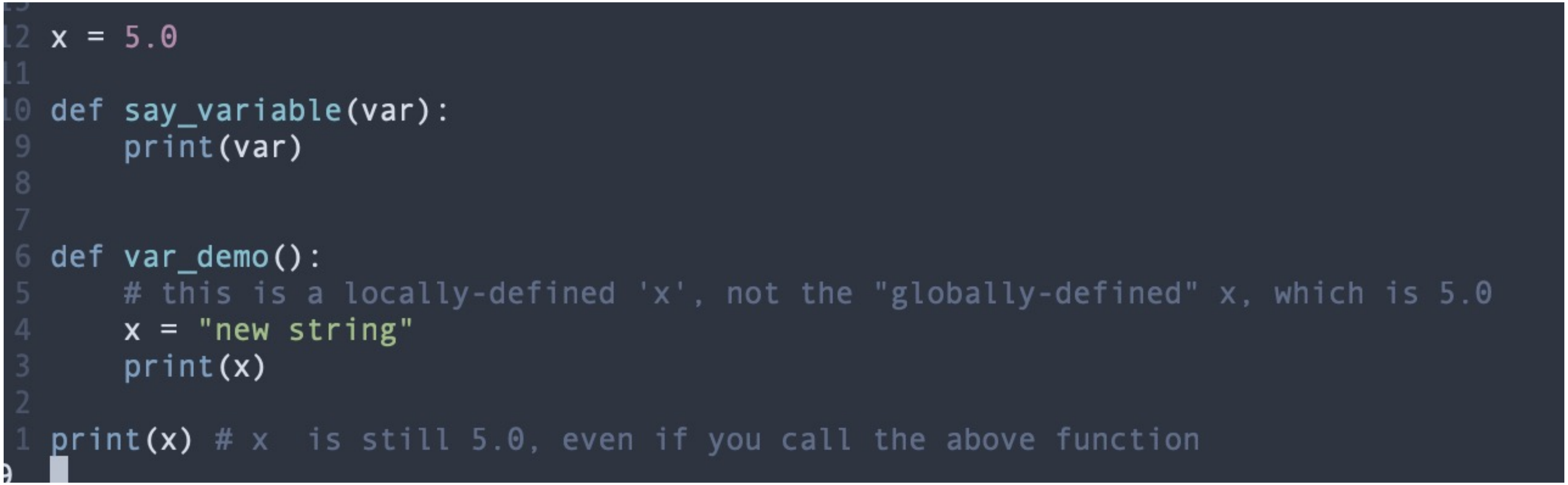
```
>>> # Measure some strings:
... words = ['cat', 'window', 'defenestrate']
>>> for w in words:
...     print(w, len(w))
...
cat 3
window 6
defenestrate 12
```

Code that modifies a collection while iterating over that same collection can be tricky to get right. Instead, it is usually more straight-forward to loop over a copy of the collection or to create a new collection:

```
# Create a sample collection
users = {'Hans': 'active', 'Éléonore': 'inactive', '景太郎': 'active'}

# Strategy:  Iterate over a copy
for user, status in users.copy().items():
    if status == 'inactive':
        del users[user]

# Strategy:  Create a new collection
active_users = {}
for user, status in users.items():
    if status == 'active':
        active_users[user] = status
```

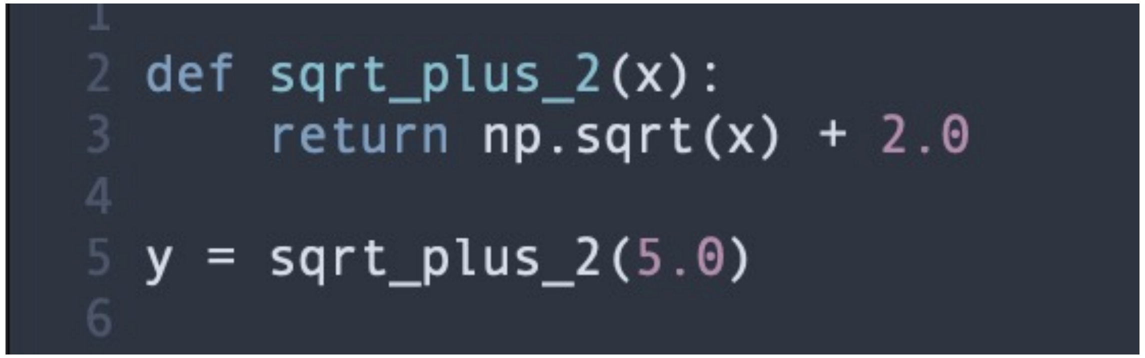
https://docs.python.org/3/tutorial/controlflow.html

Functions and Scope of Variables



```
x = 5.0

def say_variable(var):
    print(var)


def var_demo():
    # this is a locally-defined 'x', not the "globally-defined" x, which is 5.0
    x = "new string"
    print(x)

print(x) # x  is still 5.0, even if you call the above function
```

Return simple variables or objects with the ‘return’ keyword:



```
def sqrt_plus_2(x):
    return np.sqrt(x) + 2.0

y = sqrt_plus_2(5.0)
```

Lists

- Often, it is helpful to use containers to house lots of data that are related or are members of a sequence
- The “list” object is very useful here, and is quite flexible

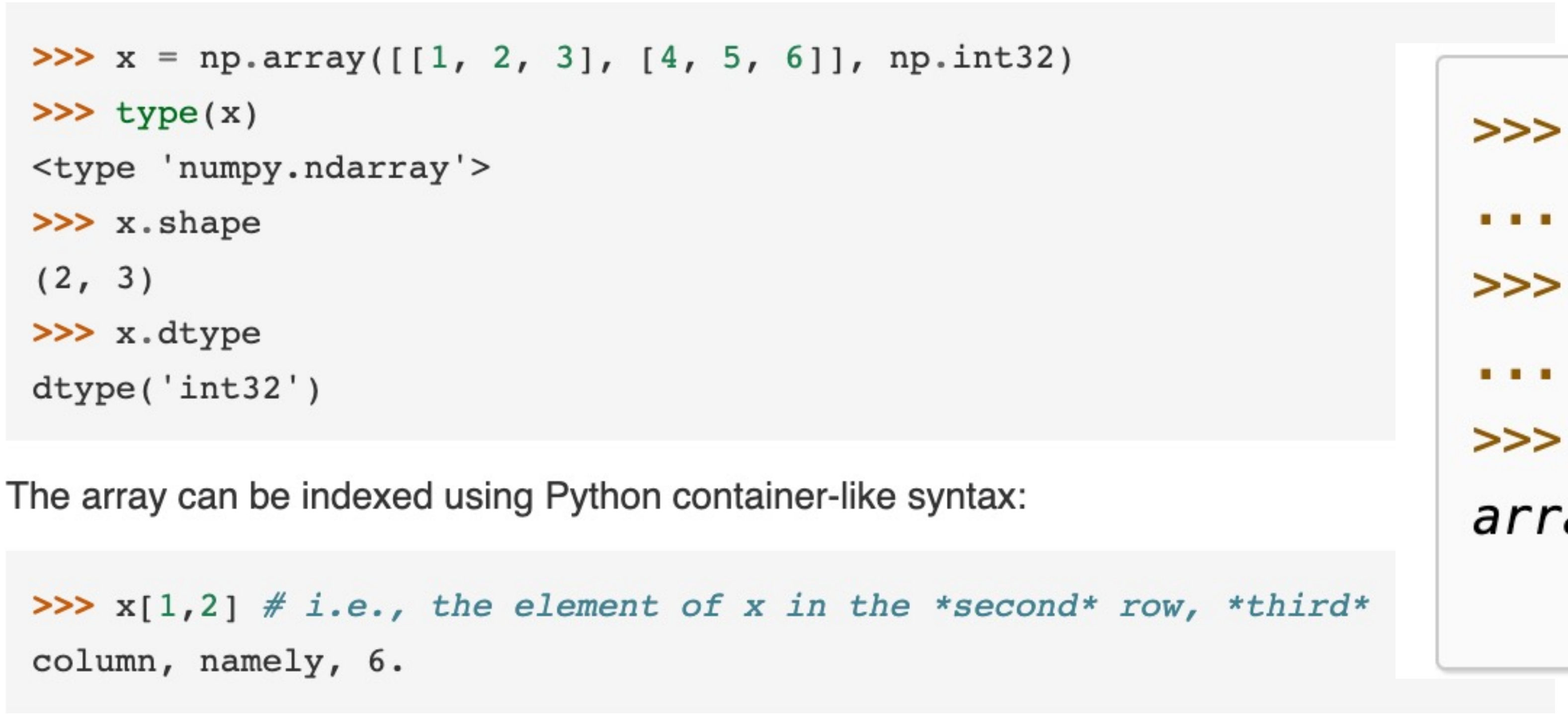
```
x_list = [] # creates empty list
x_list.append(3.14)
for i in range(5):
    x_list.append(i)
print(x) # [3.14, 0, 1, 2, 3, 4]
print(x[0]) # first element: 3.14
print(x[-1]) # last element: 4
```

Useful Libraries:

- **Numpy**: https://numpy.org/ used for computation, has many useful functions that are absent from vanilla Python (sin, cos, exp, sqrt, linear algebra functions)
- **Scipy**: https://scipy.org/ Used for optimization, curve-fitting, numerical integration/differentiation, statistics, linear algebra
- **Pandas**: https://pandas.pydata.org I/O operations, ”data frame” objects for easy use of data, sorting, etc.
- **Matplotlib: https://matplotlib.org/** plotting data, functions
- **Sys, OS:** https://docs.python.org/3/library/sys.html https://docs.python.org/3/library/os.html Useful for scripting and executing shell commands from within python
- **Pickle**: https://docs.python.org/3/library/pickle.html storing heavy objects you have made in python as ”pkl” files for later use.

Numpy Arrays

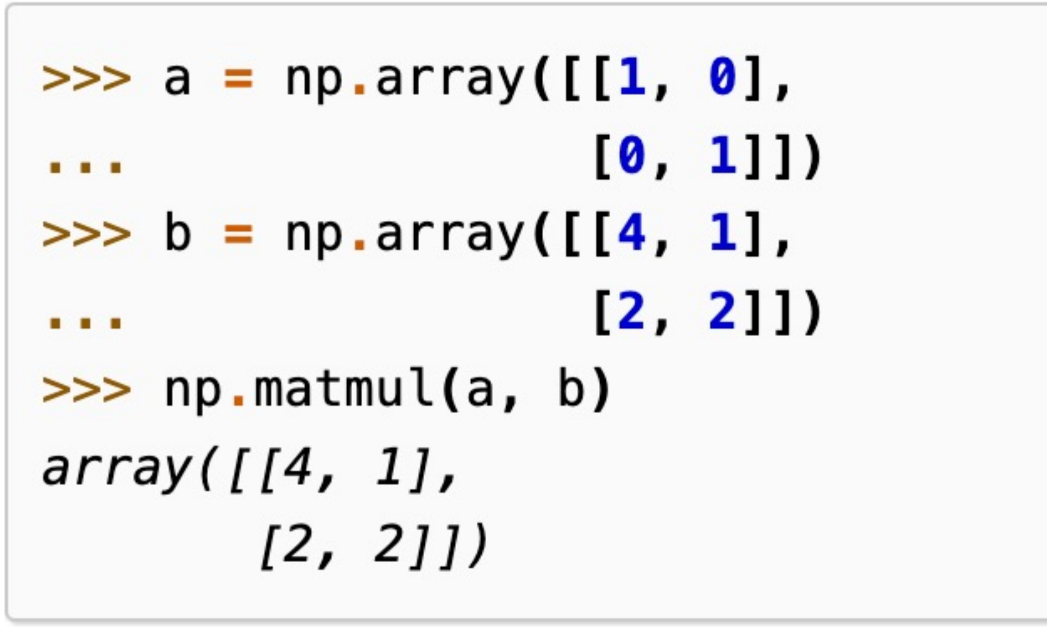
- A more numerically-friendly way to store data than lists, in many cases
- Useful when a fixed-length, fixed data type container is needed.
- Can behave like matrices when needed! (`np.matmul`)



```
>>> x = np.array([[1, 2, 3], [4, 5, 6]], np.int32)
>>> type(x)
<type 'numpy.ndarray'>
>>> x.shape
(2, 3)
>>> x.dtype
dtype('int32')
```

The array can be indexed using Python container-like syntax:

```
>>> x[1,2] # i.e., the element of x in the *second* row, *third*
column, namely, 6.
```

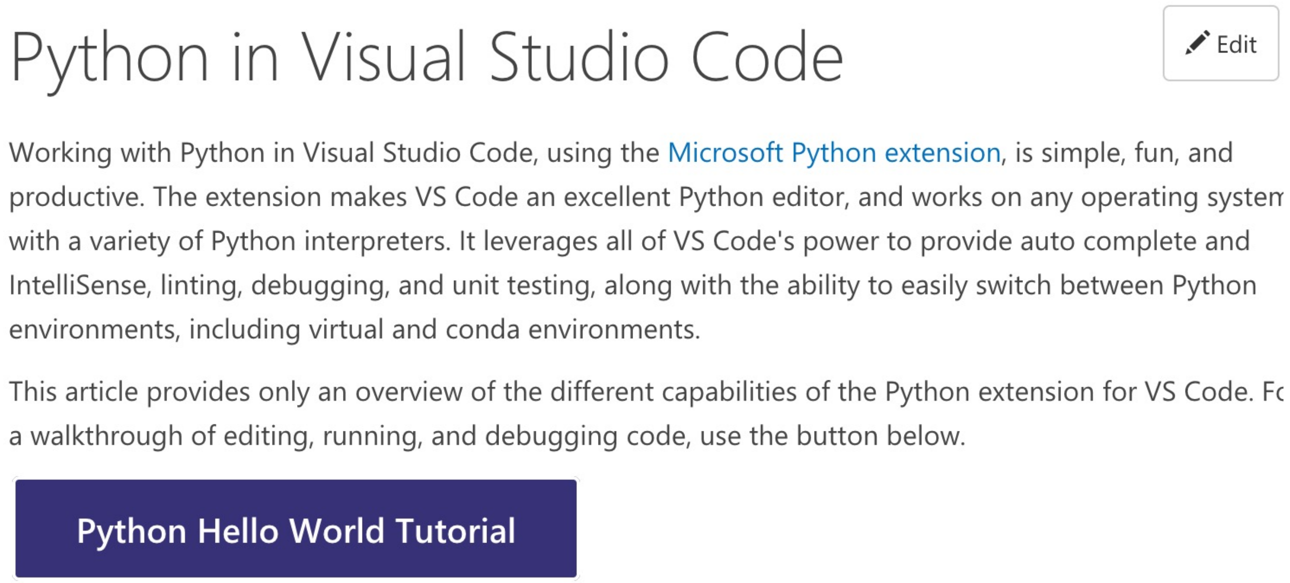


```
>>> a = np.array([[1, 0],
...               [0, 1]])
>>> b = np.array([[4, 1],
...               [2, 2]])
>>> np.matmul(a, b)
array([[4, 1],
       [2, 2]])
```

https://docs.scipy.org/doc/numpy-1.10.1/reference/arrays.ndarray.html

Running your Code

- Can keep code in ".py" files to run via command line or with an IDE

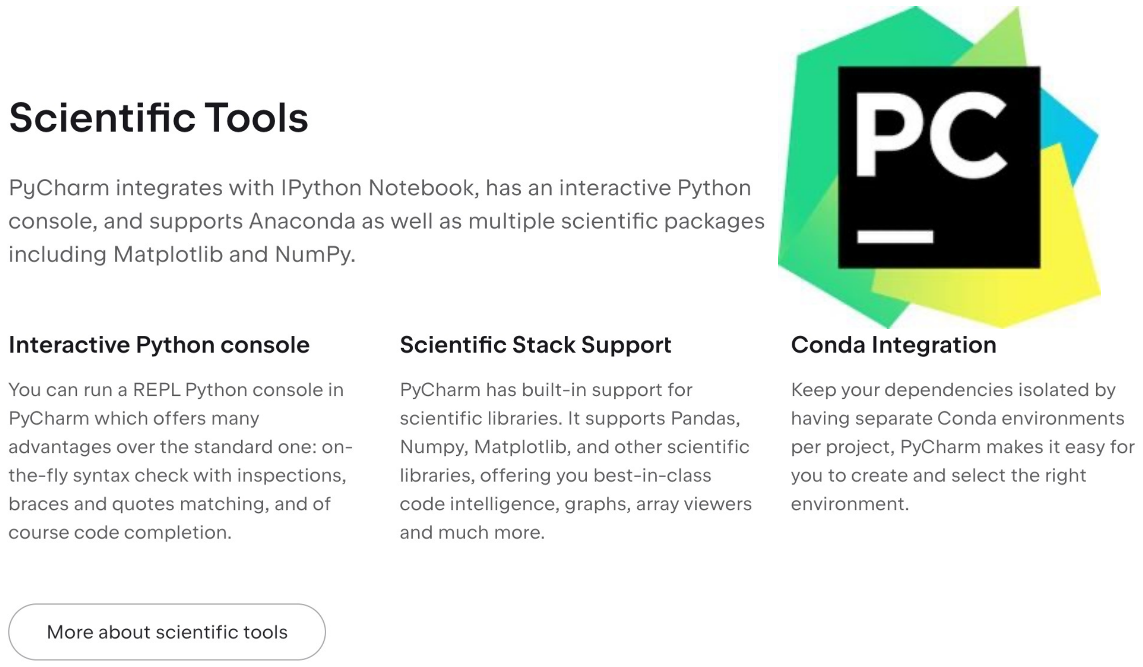


Python in Visual Studio Code

Working with Python in Visual Studio Code, using the Microsoft Python extension, is simple, fun, and productive. The extension makes VS Code an excellent Python editor, and works on any operating system with a variety of Python interpreters. It leverages all of VS Code's power to provide auto complete and IntelliSense, linting, debugging, and unit testing, along with the ability to easily switch between Python environments, including virtual and conda environments.

This article provides only an overview of the different capabilities of the Python extension for VS Code. Fc a walkthrough of editing, running, and debugging code, use the button below.

https://code.visualstudio.com/docs/languages/python



Scientific Tools

PyCharm integrates with IPython Notebook, has an interactive Python console, and supports Anaconda as well as multiple scientific packages including Matplotlib and NumPy.

Interactive Python console

You can run a REPL Python console in PyCharm which offers many advantages over the standard one: on-the-fly syntax check with inspections, braces and quotes matching, and of course code completion.

Scientific Stack Support

PyCharm has built-in support for scientific libraries. It supports Pandas, Numpy, Matplotlib, and other scientific libraries, offering you best-in-class code intelligence, graphs, array viewers and much more.

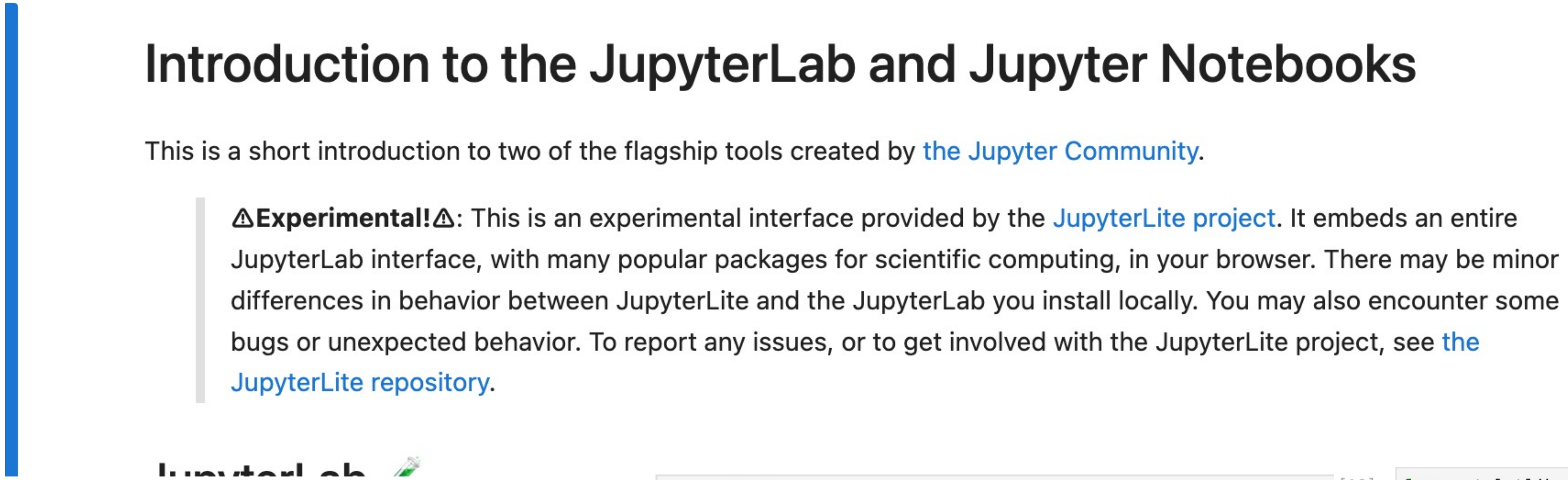
Conda Integration

Keep your dependencies isolated by having separate Conda environments per project, PyCharm makes it easy for you to create and select the right environment.

https://www.jetbrains.com/pycharm/features/

- Alternatively, code can be kept in Jupyter notebooks for interactive use next to markdown-style text boxes (next slide)

Jupyter - https://jupyter.org/try-jupyter



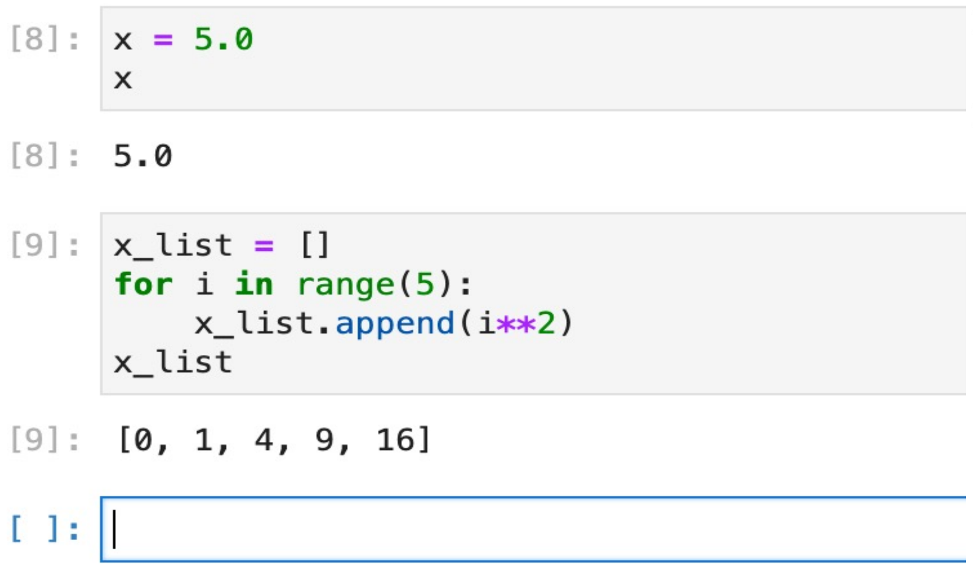
Introduction to the JupyterLab and Jupyter Notebooks

This is a short introduction to two of the flagship tools created by the Jupyter Community.

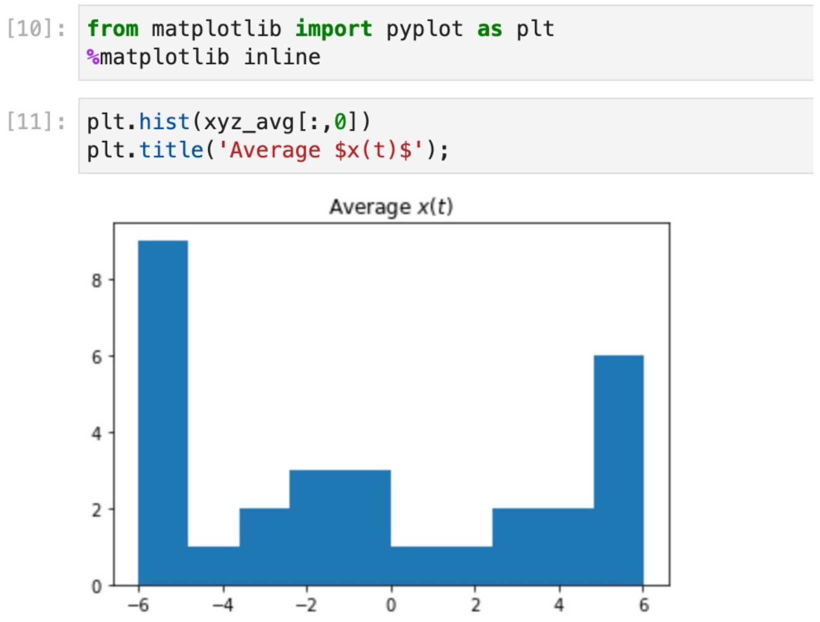
> ⚠**Experimental!**⚠: This is an experimental interface provided by the JupyterLite project. It embeds an entire JupyterLab interface, with many popular packages for scientific computing, in your browser. There may be minor differences in behavior between JupyterLite and the JupyterLab you install locally. You may also encounter some bugs or unexpected behavior. To report any issues, or to get involved with the JupyterLite project, see the JupyterLite repository.

JupyterLab

Lets you run code in cells and see output and results on the fly. Can also be used for keeping all of your plots in one place.



```
[8]: x = 5.0
     x

[8]: 5.0

[9]: x_list = []
     for i in range(5):
         x_list.append(i**2)
     x_list

[9]: [0, 1, 4, 9, 16]

[ ]:
```

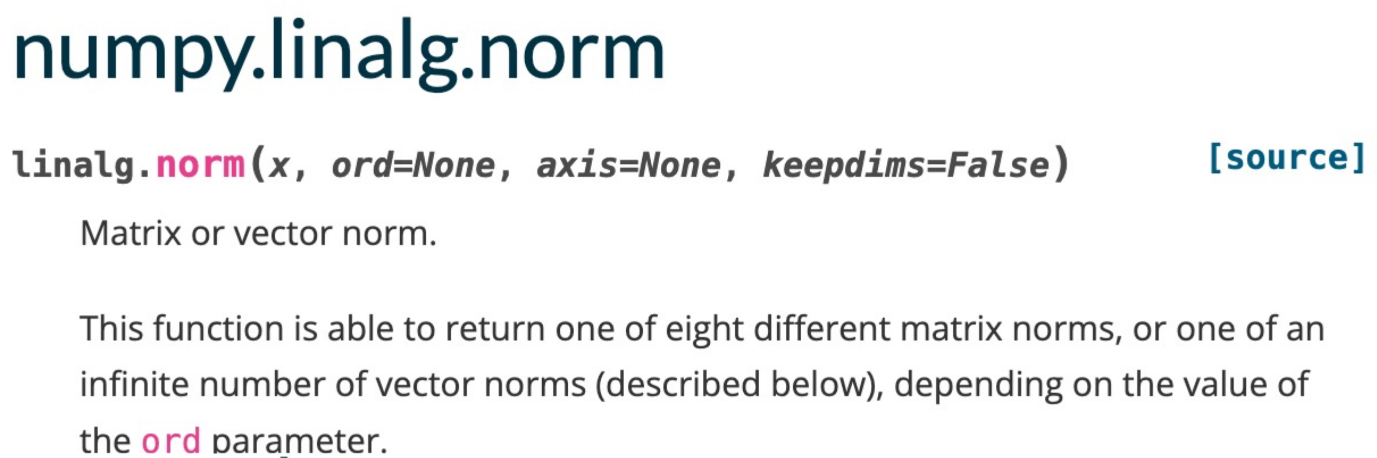


```
[10]: from matplotlib import pyplot as plt
      %matplotlib inline

[11]: plt.hist(xyz_avg[:,0])
      plt.title('Average $x(t)$');
```

Speeding Up Python

(because it can be slow…)

Prefer library functions to hand-made functions for fast computation!

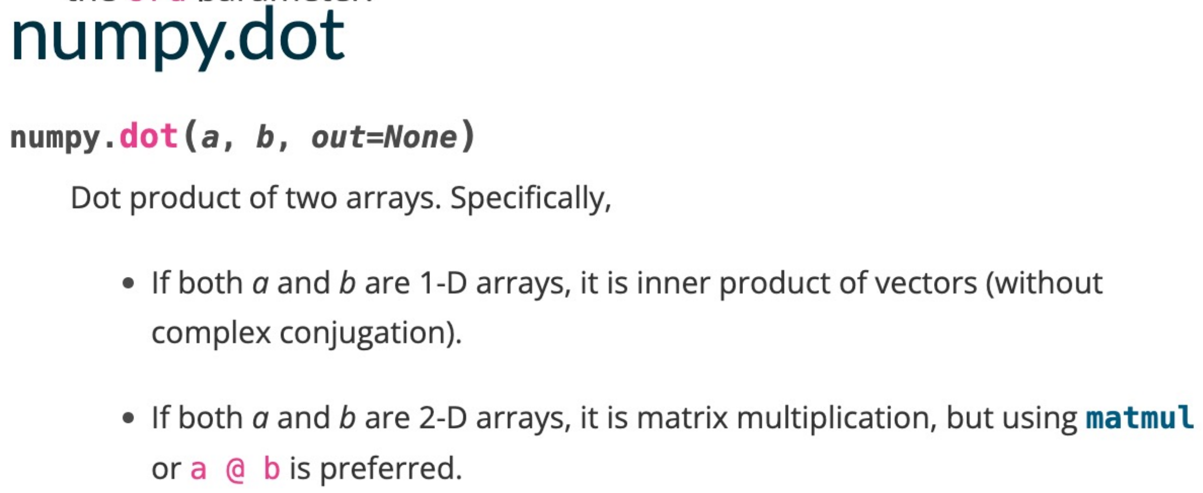


numpy.linalg.norm

`linalg.norm(x, ord=None, axis=None, keepdims=False)` [source]

Matrix or vector norm.

This function is able to return one of eight different matrix norms, or one of an infinite number of vector norms (described below), depending on the value of the `ord` parameter.



numpy.dot

`numpy.dot(a, b, out=None)`

Dot product of two arrays. Specifically,

- If both *a* and *b* are 1-D arrays, it is inner product of vectors (without complex conjugation).
- If both *a* and *b* are 2-D arrays, it is matrix multiplication, but using **`matmul`** or `a @ b` is preferred.

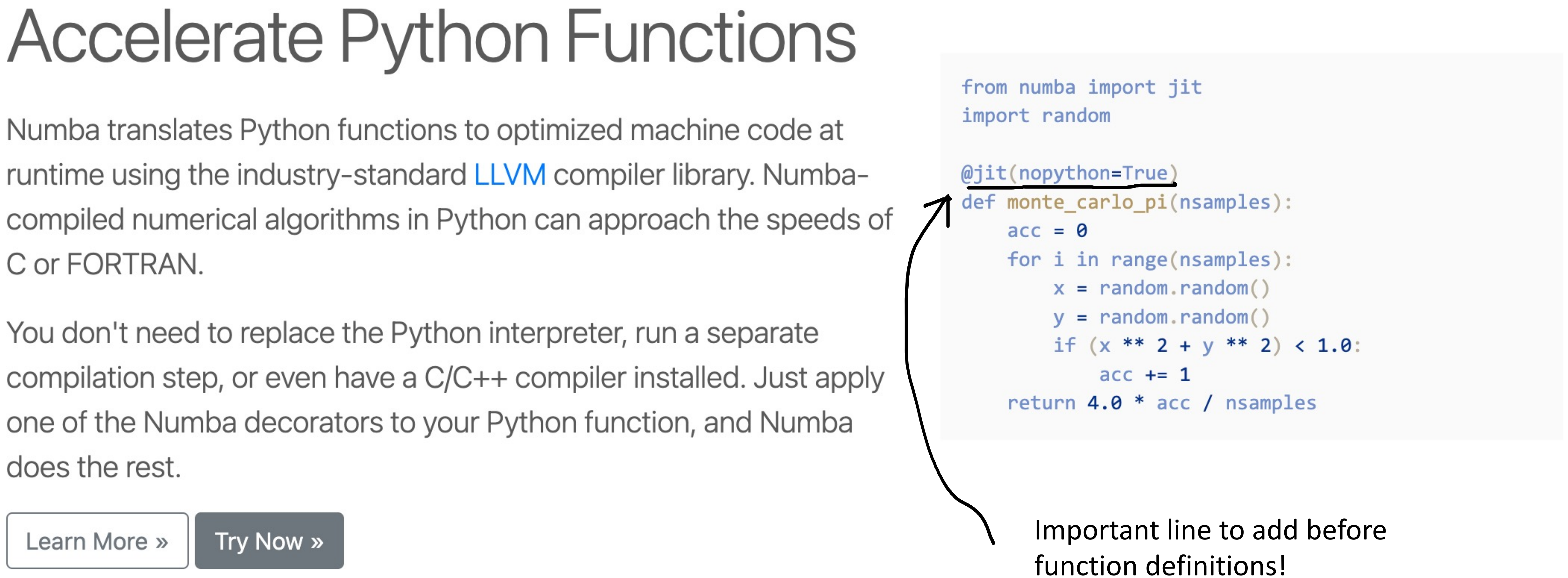
These are almost always faster than hand-made functions to do the same thing!

Look up if there are already functions made for what you need!

https://numpy.org/doc/stable/reference/generated/numpy.linalg.norm.html

https://numpy.org/doc/stable/reference/generated/numpy.dot.html

Numba



Accelerate Python Functions

Numba translates Python functions to optimized machine code at runtime using the industry-standard LLVM compiler library. Numba-compiled numerical algorithms in Python can approach the speeds of C or FORTRAN.

You don't need to replace the Python interpreter, run a separate compilation step, or even have a C/C++ compiler installed. Just apply one of the Numba decorators to your Python function, and Numba does the rest.

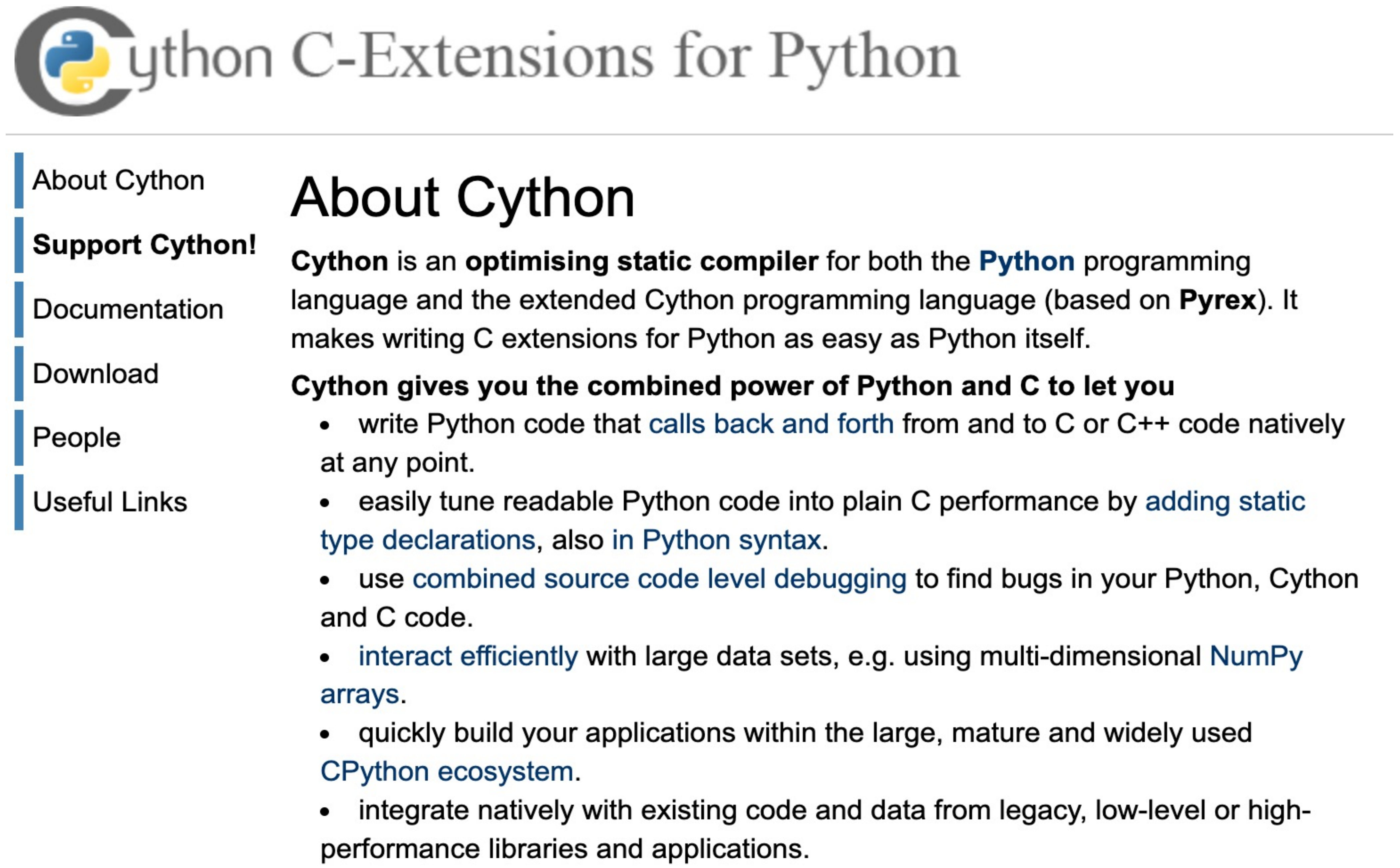
Learn More » Try Now »

```python
from numba import jit
import random

@jit(nopython=True)
def monte_carlo_pi(nsamples):
    acc = 0
    for i in range(nsamples):
        x = random.random()
        y = random.random()
        if (x ** 2 + y ** 2) < 1.0:
            acc += 1
    return 4.0 * acc / nsamples
```

Important line to add before function definitions!

(this method will not always work, but when it does it is useful)

https://numba.pydata.org/



Cython C-Extensions for Python

- About Cython
- **Support Cython!**
- Documentation
- Download
- People
- Useful Links

About Cython

Cython is an **optimising static compiler** for both the **Python** programming language and the extended Cython programming language (based on **Pyrex**). It makes writing C extensions for Python as easy as Python itself.

Cython gives you the combined power of Python and C to let you

- write Python code that calls back and forth from and to C or C++ code natively at any point.
- easily tune readable Python code into plain C performance by adding static type declarations, also in Python syntax.
- use combined source code level debugging to find bugs in your Python, Cython and C code.
- interact efficiently with large data sets, e.g. using multi-dimensional NumPy arrays.
- quickly build your applications within the large, mature and widely used CPython ecosystem.
- integrate natively with existing code and data from legacy, low-level or high-performance libraries and applications.

https://cython.org/#about

Installing Libraries and Dependencies

Conda

What is a conda package?

A conda package is a compressed tarball file (.tar.bz2) or .conda file that contains:

- system-level libraries.
- Python or other modules.
- executable programs and other components.
- metadata under the `info/` directory.
- a collection of files that are installed directly into an `install` prefix.

Conda keeps track of the dependencies between packages and platforms. The conda package format is identical across platforms and operating systems.

Only files, including symbolic links, are part of a conda package. Directories are not included. Directories are created and removed as needed, but you cannot create an empty directory from the tar archive directly.

Package, dependency and environment management for any language—Python, R, Ruby, Lua, Scala, Java, JavaScript, C/ C++, Fortran, and more.

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.

Conda as a package manager helps you find and install packages. If you need a package that requires a different version of Python, you do not need to switch to a different environment manager, because conda is also an environment manager. With just a few commands, you can set up a totally separate environment to run that different version of Python, while continuing to run your usual version of Python in your normal environment.

In its default configuration, conda can install and manage the thousand packages at repo.anaconda.com that are built, reviewed and maintained by Anaconda®.

https://docs.conda.io/en/latest/

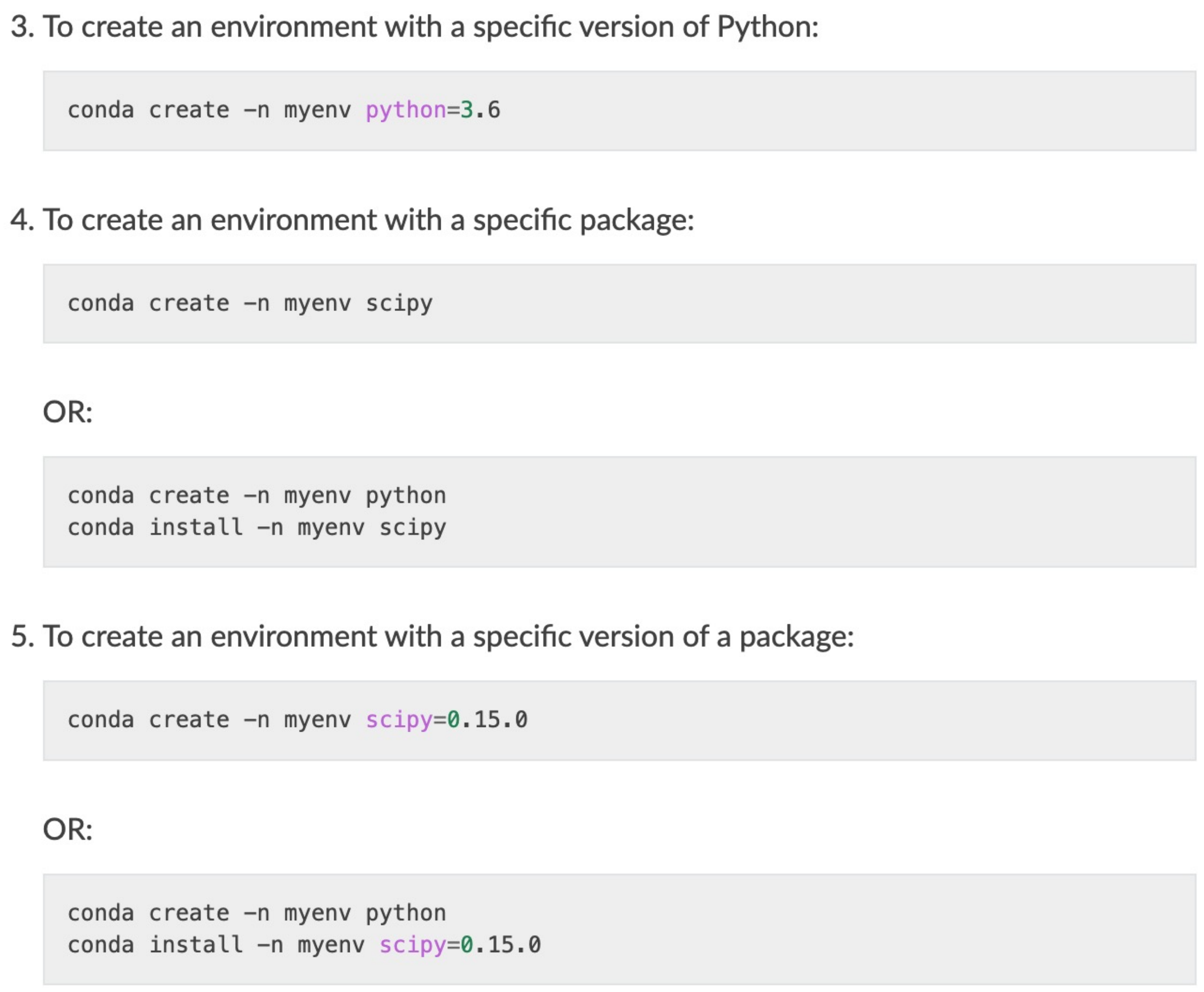
Conda Environments

Often a specific library (or version of said library) is needed

Conda environments are used to give the user control over what libraries their code and use and when

Launch an existing environment with ``conda activate <name_of_environment>”

For the HF/DFT code project, it might be helpful to make an environment with everything that you want to have access to. (New packages can always be added as you go along).



3. To create an environment with a specific version of Python:

```
conda create -n myenv python=3.6
```

4. To create an environment with a specific package:

```
conda create -n myenv scipy
```

OR:

```
conda create -n myenv python
conda install -n myenv scipy
```

5. To create an environment with a specific version of a package:

```
conda create -n myenv scipy=0.15.0
```

OR:

```
conda create -n myenv python
conda install -n myenv scipy=0.15.0
```

https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html