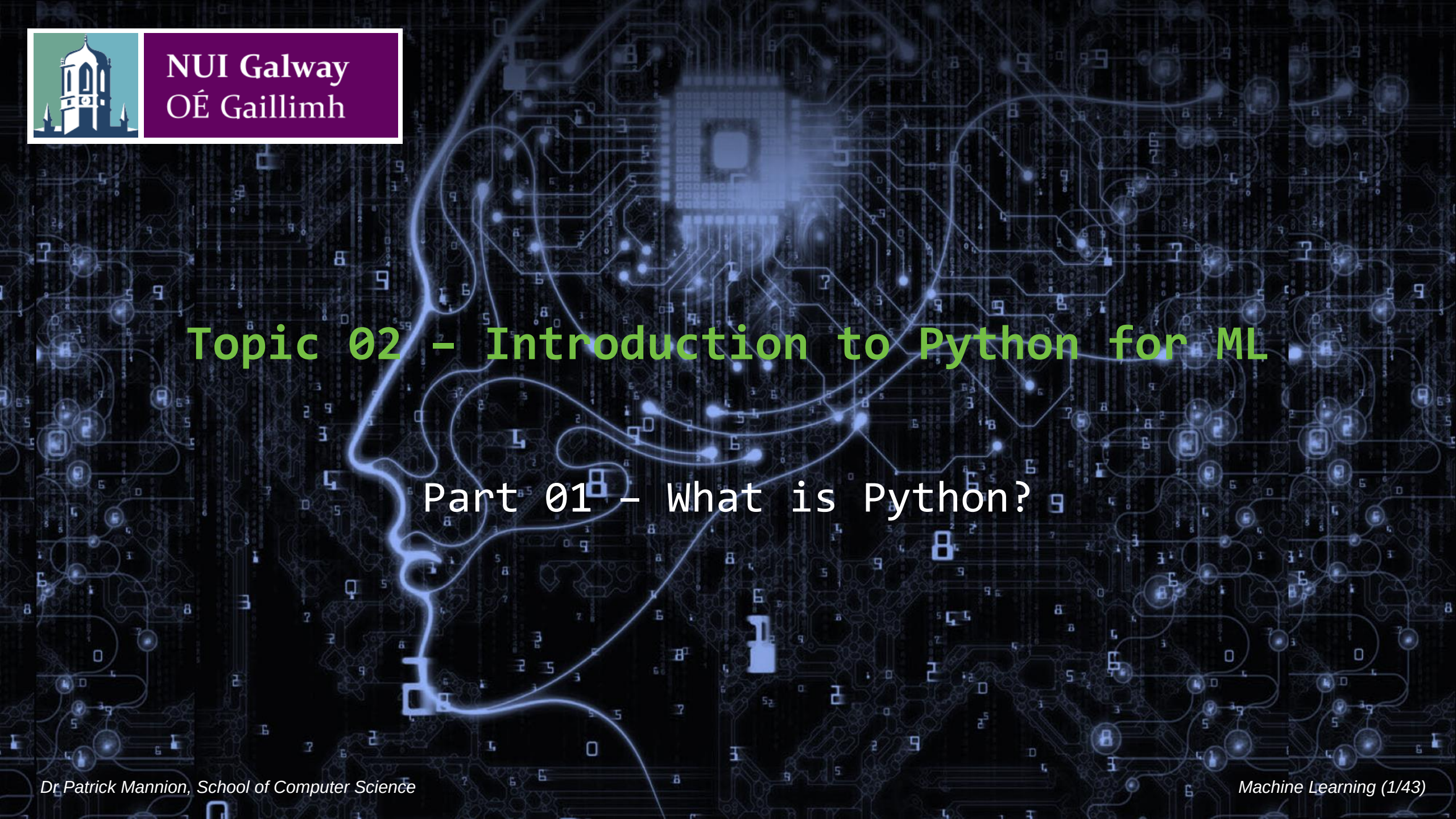


NUI Galway
OÉ Gaillimh

Topic 02 – Introduction to Python for ML

Part 01 – What is Python?

Hardware and software requirements

- You will need Access to a PC/laptop where you can install:
  - Python version 3.8.5 (or newer)
  - A good text/code editor such as Notepad++
  - A suitable Integrated Development Environment (IDE), such as PyCharm Community Edition (or else use Jupyter Lab – more on this later!)

- Details of the software that you need to install are available on the **Module Information** page on Blackboard. You will find links on this page to download the necessary installers.

- Your first exercise for this module: install all the required software listed on the **Module Information** page. Do this as soon as possible, so that you can try out the code samples in the notes for yourself and begin working on the assignments as soon as they are posted.

What is Python?

- Python is general-purpose high-level programming language
- The first version of Python was created by Guido van Rossum and released in 1991
- Python programs are interpreted by an interpreter. E.g., CPython, the reference implementation supported by the Python Software Foundation. CPython is both a compiler and an interpreter as it first compiles Python code into bytecode, before then interpreting it
- Python interpreters are available for a wide variety of operating systems and platforms
- Python supports multiple programming paradigms such as procedural programming, object-oriented programming and functional programming
- Python is **dynamically typed**. By contrast, languages like C, C++ and Java are **statically typed**. Basically, this means that many common error checks are deferred until runtime in Python, whereas in a statically typed language like Java these checks are performed during compilation. We'll discuss the differences between statically typed and dynamically typed languages in more detail later
- Python uses **garbage collection**, meaning that memory management is handled automatically, and there is no need for the programmer to manually allocate and deallocate chunks of memory

What is Python used for?

Python is used for all kinds of computational tasks, including:

- Scientific computing
- Data analytics
- Artificial intelligence & machine learning
- Computer vision
- Web development / web apps
- Mobile apps
- Desktop graphical user interface (GUI) applications
- etc…

Who uses Python?

And countless others …

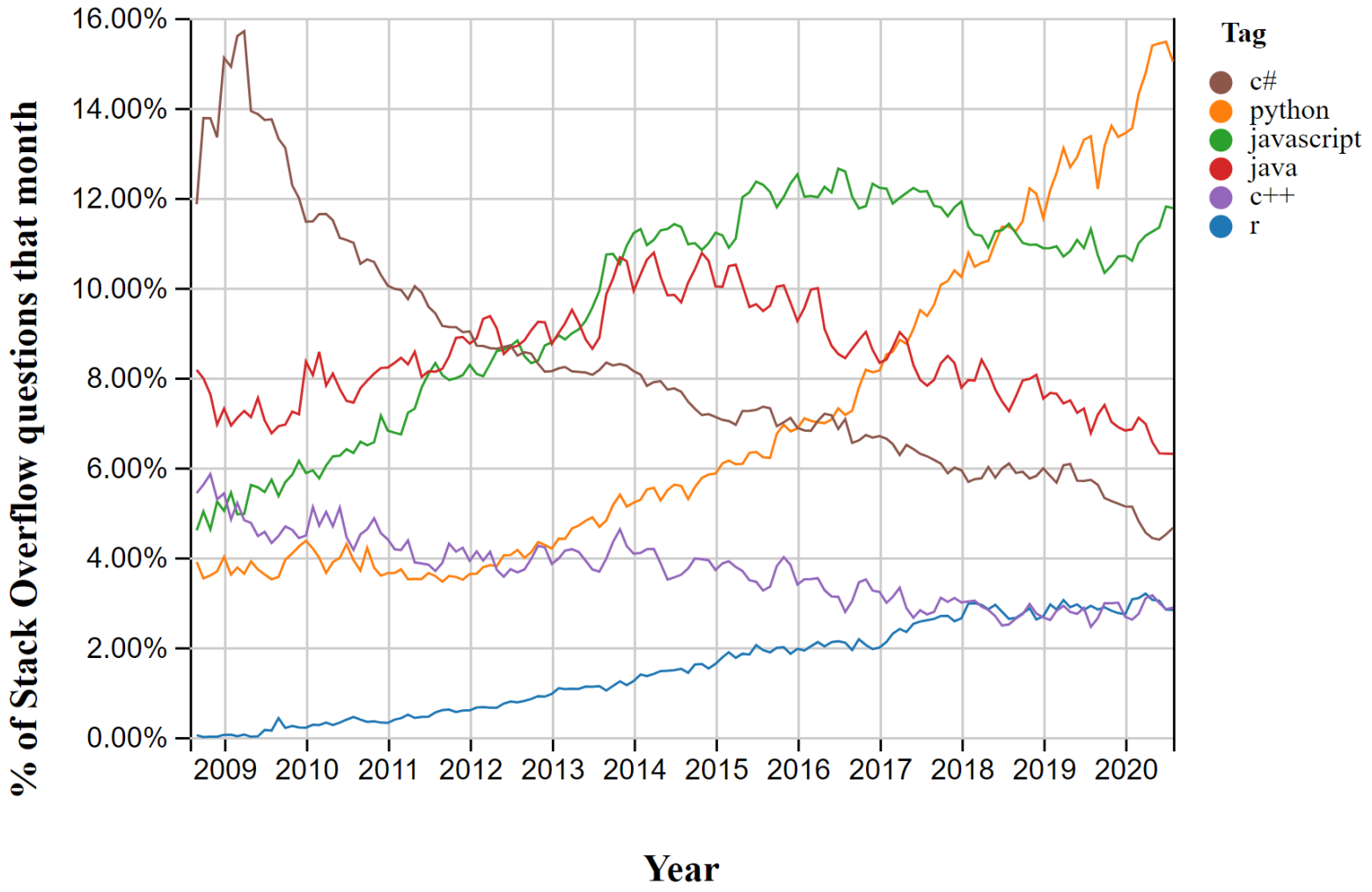
Why use Python? (1/2)

- Python is a high-level programming language, with a relatively simple syntax
- As well as being easy to learn for beginners, it also has very advanced functionality
- One of the most widely used programming languages
- Open source and freely available
- Portability – Python programs will run almost anywhere as installations of the Python interpreter are available for almost any operating system and platform, e.g. x86, ARM, Windows, Linux, Mac etc. By contrast, in languages such as C or C++ separate binaries must be compiled for each specific platform and operating system that is supported by the application.

Why use Python? (2/2)

- Libraries – Python has a vast array of libraries available, most of which are free and open source. In most cases when you're coding in Python and need a specific feature, there is a library available with that feature already implemented.
- Productivity – Python programs are usually much shorter than the equivalent Java or C++ code. This means less code to write leading to faster development times for experienced Python programmers. Brevity means that code is also easier to maintain, debug and refactor as much less reading of source code is required for these tasks. Python code can also be run without the need for ahead of time compilation (as in C or C++), allowing for faster iterations over code versions and faster testing.
- Integration with other software – Python can be easily extended and integrated with software written in many other programming languages.

Python is growing in popularity



- Stack Overflow is one of the largest online discussion forums for programming problems.
- This plot shows the relative frequency of tag use for some of the most popular programming languages since 2008.
- Discussions about Python have been trending upwards since the early 2010s.

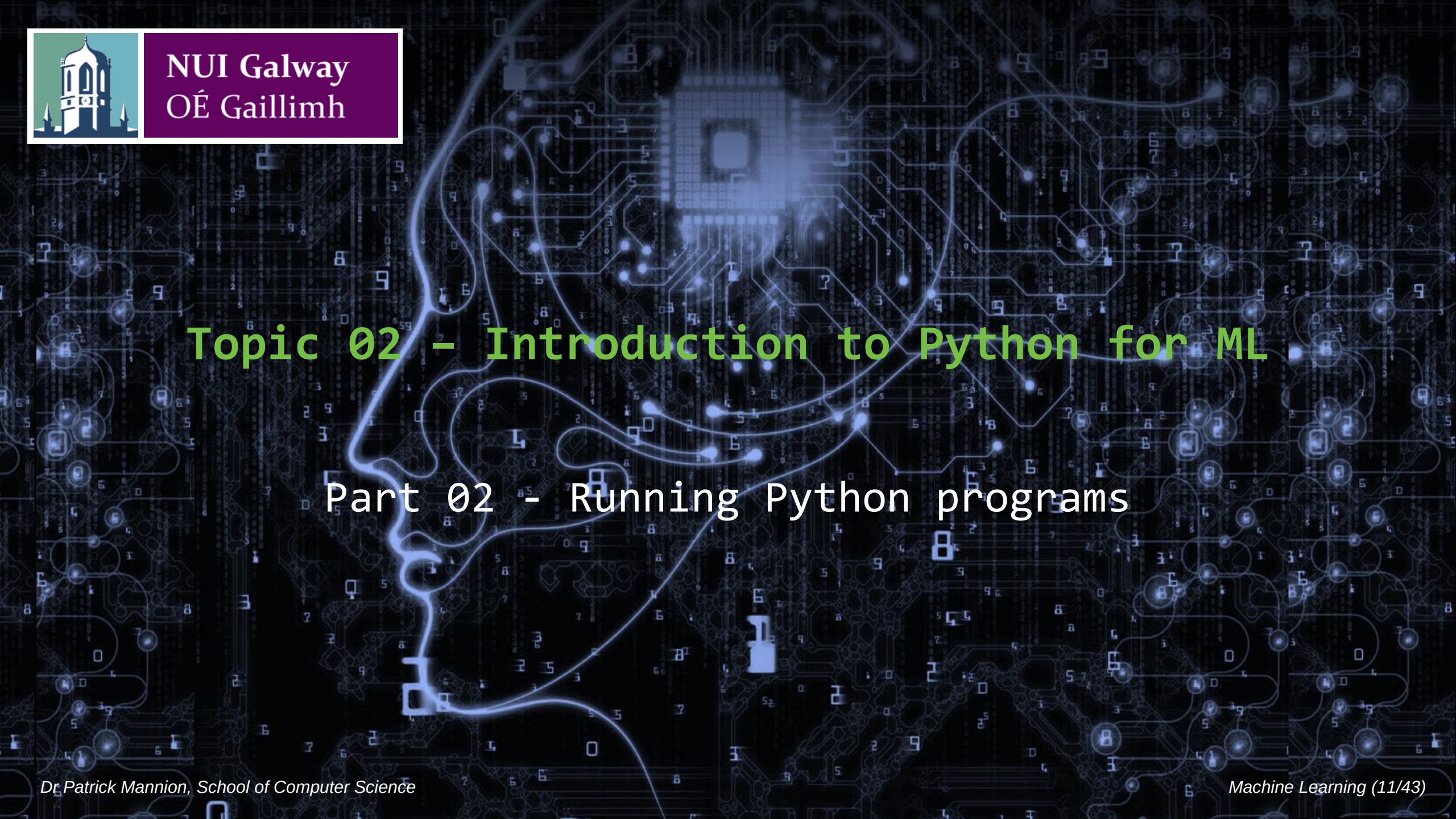
Image source: Stock Overflow Trends, https://insights.stackoverflow.com/trends?tags=r%2Cpython%2Cjavascript%2Cjava%2Cc%2B%2B%2Cc%23

Drawbacks of using Python (1/2)

- Efficiency – Program execution speed in Python is typically a lot slower than more low-level languages such as C or C++. The relative execution speed of Python compared to C or C++ depends a lot on coding practices and the specific application being considered.
- Memory management in Python is also less efficient than well-written C or C++ code
- These efficiency concerns are not usually a big deal, as compute power and memory are now relatively cheap on desktop, laptop and server systems. Python is used in the backend of large web services such as Spotify and Instagram etc. and performs adequately. Exceptions: these efficiency concerns mean Python may be unsuitable for some performance-critical applications, e.g., resource-intensive scientific computing, embedded devices, automotive, etc.
- Faster alternative Python implementations such as PyPy are available – PyPy provides an average four-fold speedup by implementing advanced compilation techniques.
- It's also possible to call code that is implemented in C from within Python to speed up performance-critical sections of your program

Drawbacks of using Python (2/2)

- Dynamic typing can make code more difficult to write and debug, compared to statically typed languages like C, C++ or Java. In statically typed languages, the compiler checks that all variable types match before the code is executed
- Python 2 vs. Python 3 – there are two different major versions of Python in widespread use; these versions are incompatible with each other due to several changes that were made when Python 3 was introduced. This means that some libraries that were originally written in Python 2 have not been ported over to Python 3. Python 2 is now mostly only used in legacy business applications; most new development is in Python 3. Python 2 will no longer be supported or receive updates as of 2020. We will use Python 3 exclusively in this module (specifically Python versions >= 3.8.5).



NUI Galway
OÉ Gaillimh

Topic 02 – Introduction to Python for ML

Part 02 - Running Python programs

Options for running Python programs

Python programs can be executed in a variety of different ways:

- Through the Python interactive shell on your computer
- Through remote Python interactive shells that are accessible through web browsers, such as https://www.python.org/shell/ or https://trinket.io/console
- By using the console of your operating system to launch a standalone Python script (.py file)
- By using an integrated development environment (IDE) to launch a .py file
- As graphical user interface (GUI) applications using libraries such as Tkinter or PyQt
- As web applications that provide services to other computers, e.g. by using the Flask framework to create a web server with content that can be accessed using web browsers
- Through Jupyter / JupyterLab notebooks
  - Hosted locally on your own machine
  - Cloud-based Jupyter notebook execution environments such as Google Colab, Microsoft Azure Notebooks, Binder (https://mybinder.org/)

A simple Python program

In the tradition of many programming courses, our first Python program will be a very simple one. This program does only one thing – it writes Hello world! to the screen.

File: helloworld.py

```
print("Hello world!")
```

Later, we will run this simple program using several of the methods that we just discussed. Running this simple program is a good way to test whether you have installed and set up everything correctly.

Analysing hello world

```
print("Hello world!")
```

This simple program demonstrates several important programming concepts:

- The line `print("Hello world!")` is called a **statement**
- `"Hello world!"` is a **string literal**. String is one of the data types used in Python. A string can contain a sequence of characters (e.g., any the letters a-z, numbers 0-9, or other characters). The term literal means that we have **declared** a piece of data; this is not an instruction that Python should try to process.
- `print()` is the name of a **function** that is built into Python. Functions can take **arguments** as inputs, and then process those inputs. In our program, the string literal `"Hello world!")` is the argument to the `print()` function.
- Note that when calling a function, function names are always followed by parentheses `()`. Within the parentheses, any arguments to the function are passed in.

Using the Python interactive shell

Command Prompt - python

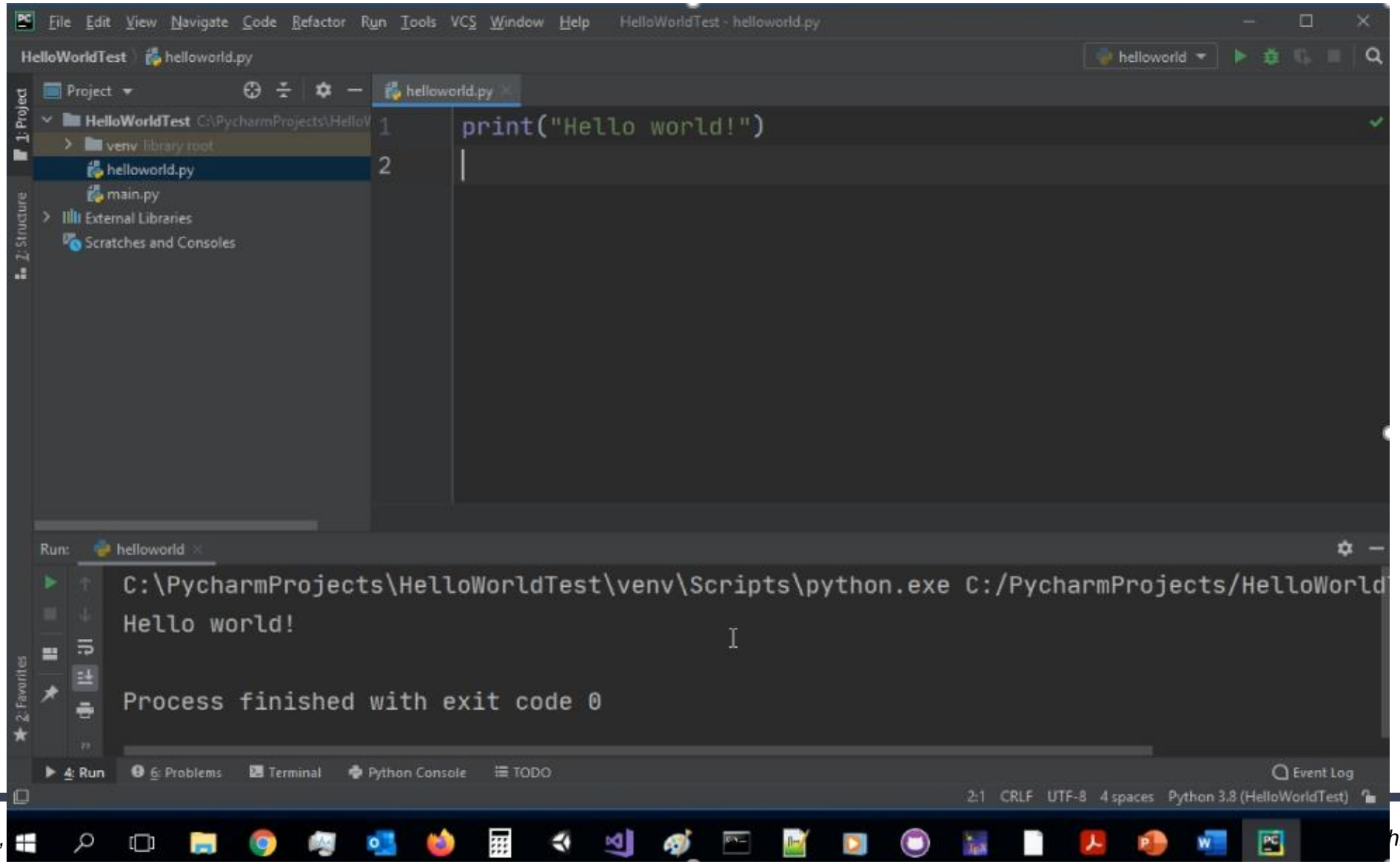
```
Microsoft Windows [Version 10.0.16299.492]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\Tuireann>python
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:57:54) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello world!")
Hello world!
```

Using the console of your operating system

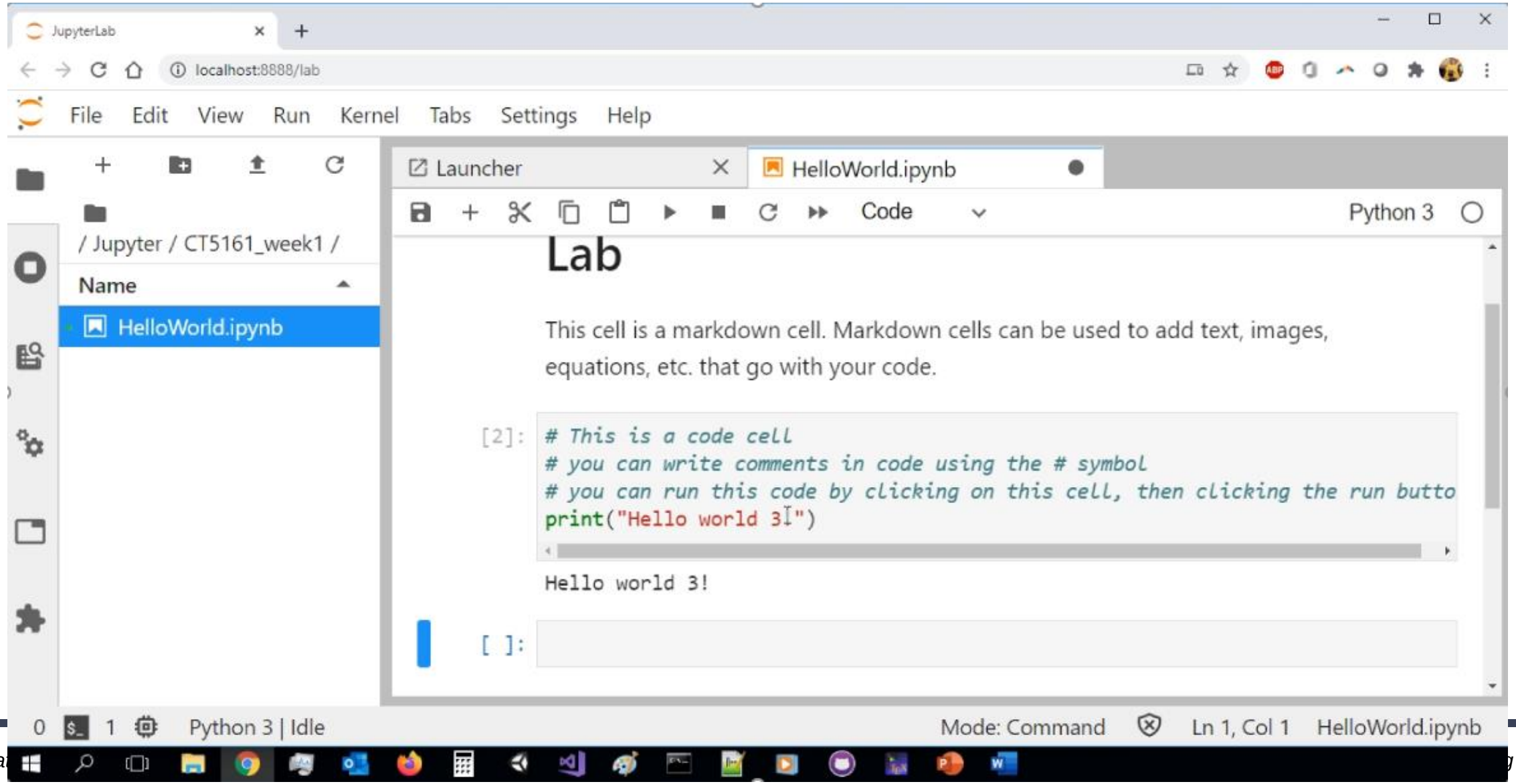
C:\Windows\System32\cmd.exe

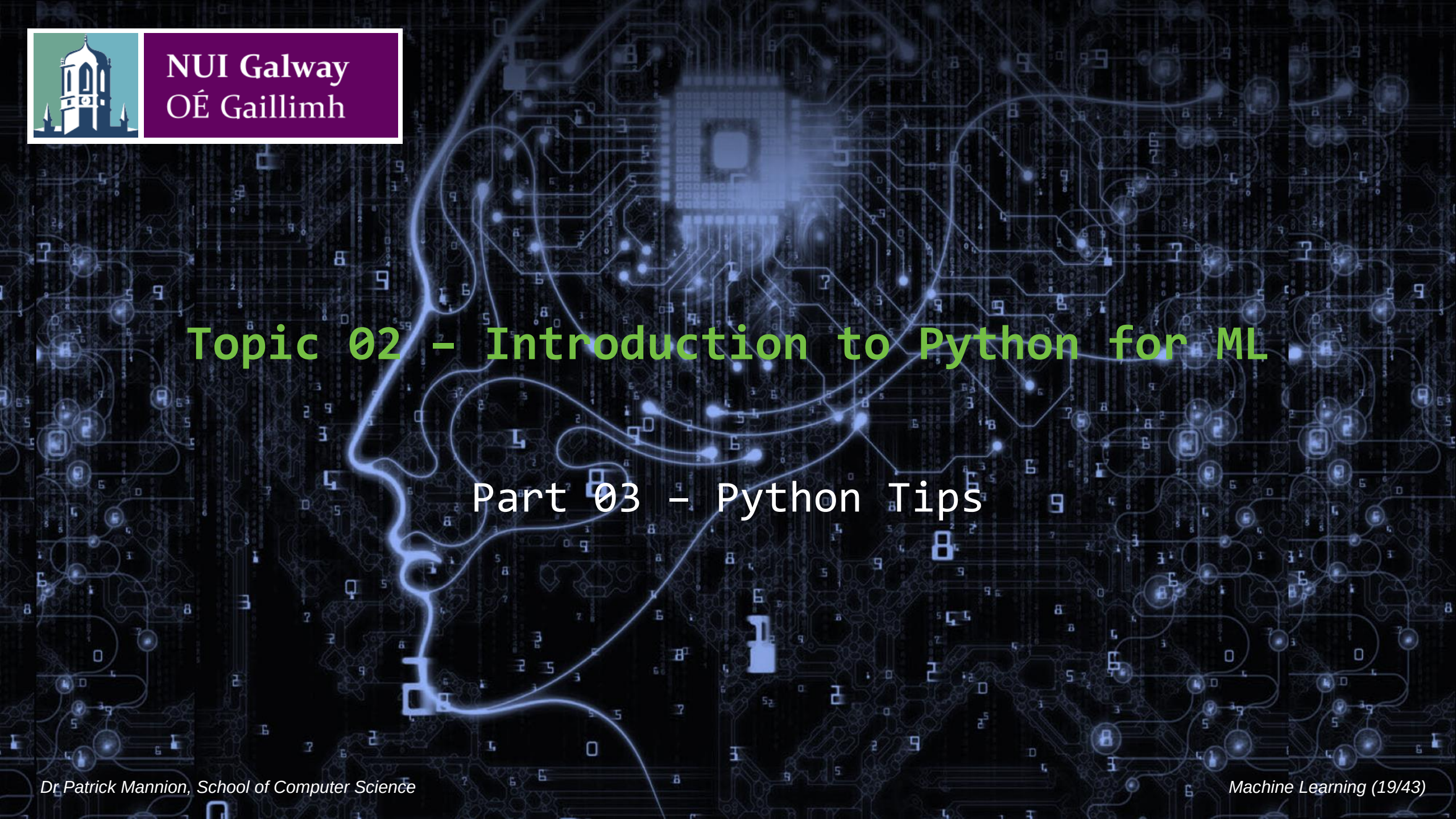
```
Microsoft Windows [Version 10.0.16299.492]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\PythonExamples>python helloworld.py
Hello world!
```

Using an IDE (PyCharm)



Using a JupyterLab notebook





NUI Galway
OÉ Gaillimh

Topic 02 – Introduction to Python for ML

Part 03 – Python Tips

PEP 8 -- Style Guide for Python Code

- PEP 8 “gives coding conventions for the Python code comprising the standard library in the main Python distribution” – see https://www.python.org/dev/peps/pep-0008/
- PEPs (Python Enhancement Proposals) describe and document the way that the Python language evolves over time, e.g., addition of new features, backwards compatibility policy, etc. PEPs can be proposed, then accepted of rejected – full list available at https://www.python.org/dev/peps/
- PEP 8 contains conventions for the user-defined names (e.g., variables, functions, packages), as well as code layout, line length, use of blank lines, style of comments, etc.
- Many professional Python developers and companies adhere to (at least some of) the PEP 8 conventions. It is important to learn to follow these conventions from the start, especially if you want to work with other programmers, as experienced Python developers will often flag violations of the PEP 8 conventions during code reviews. Of course, many companies and open-source software projects have defined their own internal coding style guidelines which take precedence over PEP 8 in the case of conflicts
- Following PEP 8 conventions is relatively easy if you are using a good IDE – e.g., PyCharm automatically finds and alerts you to violations of the PEP 8 conventions

Variable naming conventions

- According to PEP 8, variable names “should be lowercase, with words separated by underscores as necessary to improve readability”
- https://www.python.org/dev/peps/pep-0008/#function-and-variable-names
- This is the lower_case_with_underscores style on the previous slide if the variable name is made up of more than one word, e.g. bank_account
- “Never use the characters 'l' (lowercase letter el), 'O' (uppercase letter oh), or 'I' (uppercase letter eye) as single character variable names. In some fonts, these characters are indistinguishable from the numerals one and zero. When tempted to use 'l', use 'L' instead.”
- According to PEP 8 different naming conventions are used for different identifiers, e.g., “Class names should normally use the CapWords convention”. This helps programmers to quickly and easily distinguish which category an identifier name represents

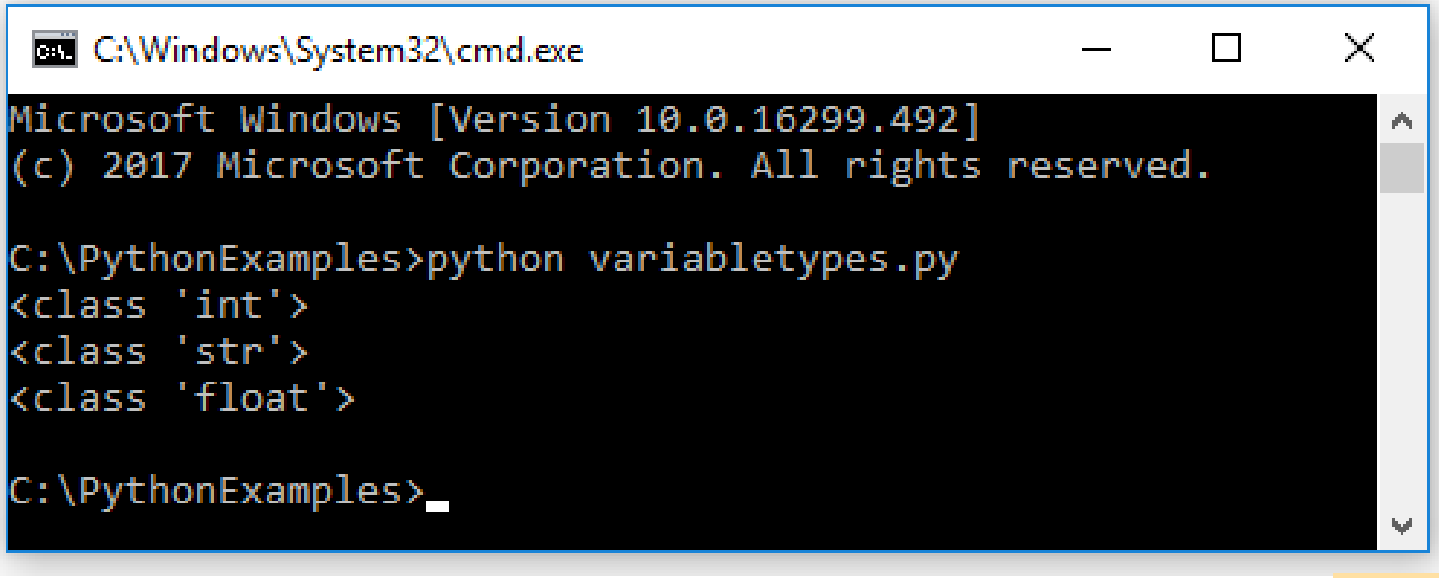
Dynamic type example (1/2)

- In Python, variable names can point to objects of any type
- Built in data types in Python include `str`, `int`, `float`, etc. Each type can hold a different kind of data. As we saw, `str` can hold any combination of characters. `int` can only hold integers (i.e. whole numbers like 4 or 317875). `float` can only hold real numbers (i.e. numbers with a decimal point like 3.14159 or 1.0)
- Because variables in Python are simply pointers to objects, the variable names themselves do not have any attached type information. In Python, types are linked not to the variable names but to the objects themselves. The following code is perfectly legal in Python:

File: variabletypes.py

```
x = 4
print(type(x)) # prints "<class 'int'>" to the console
x = "Hello world!"
print(type(x)) # prints "<class 'str'>" to the console
x = 3.14159
print(type(x)) # prints "<class 'float'>" to the console
```

Dynamic type example (2/2)



Note that `type()` is a built-in Python function that returns the type of any object that is passed to it as an argument. You can use this function when developing and testing your programs if you are ever unsure what type of object is referred to by a variable.

Here the object returned by the `type()` function (a **`type` object**) becomes the input to the `print()` function

```
x = 4
print(type(x)) # prints "<class 'int'>" to the console
x = "Hello world!"
print(type(x)) # prints "<class 'str'>" to the console
x = 3.14159
print(type(x)) # prints "<class 'float'>" to the console
```

Static typing vs. dynamic typing – example 1

- Because the type of object referred to by a variable in Python is not known until runtime, we say that Python is a dynamically typed language. In statically typed languages, we must declare the type of a variable before it is used – the type of every variable in a statically typed language is known before runtime.
- Another important difference between Python and statically typed languages like C, C++ and Java is that we do not need to declare variables before we use them. Assigning a value to a previously undeclared variable name is fine in Python but leads to a compiler error in languages like C, C++ or Java.

File: staticvsdynamic1.py

```
x = 4 # no errors
x = 5 # no errors
x = 3.14159 # no errors
x = "Hello world!" # no errors
y = "abc123" # no errors
```

File: staticvsdynamic1.java

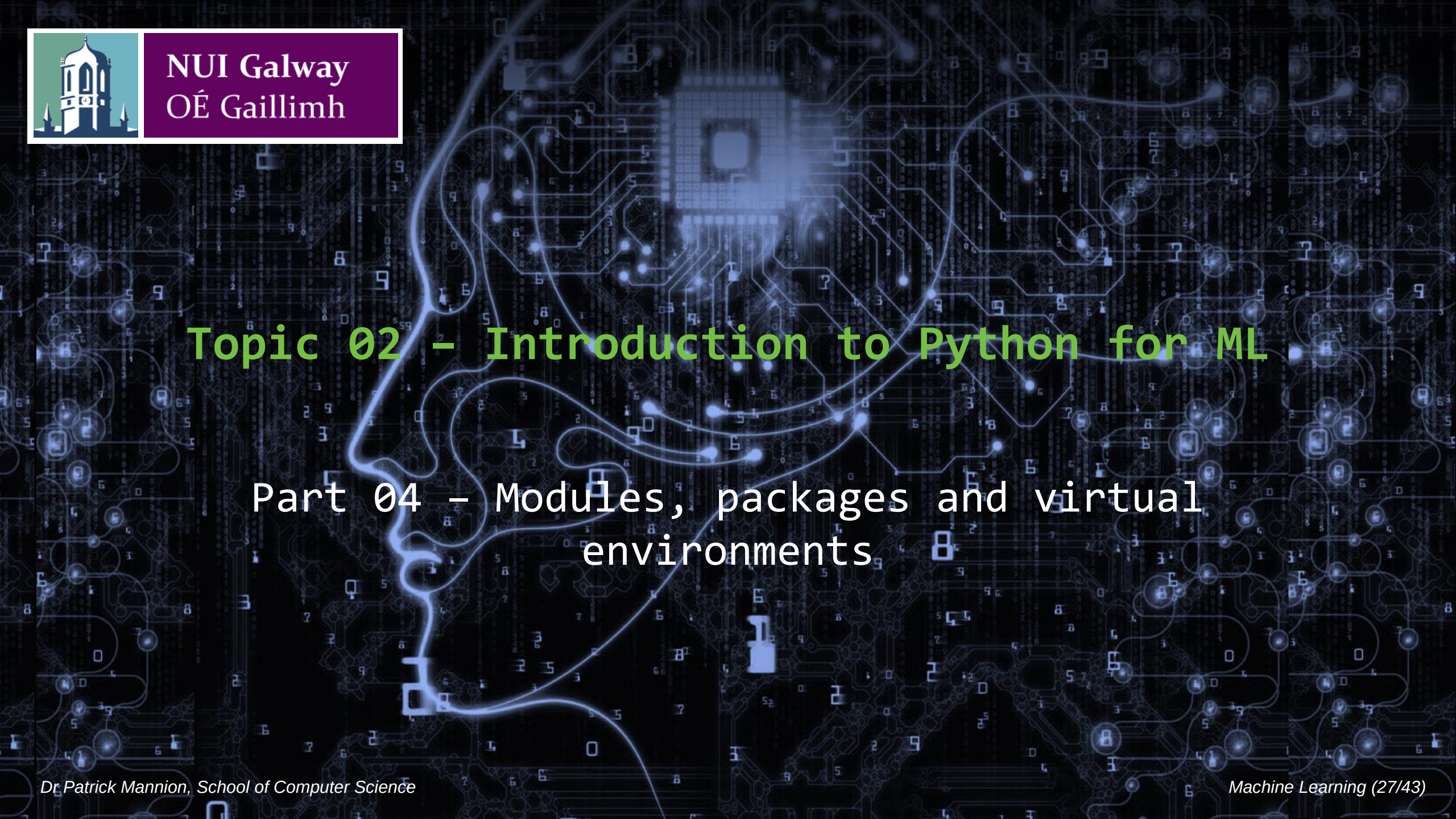
```
public class staticvsdynamic1 {
  public static void main(String[] args) {
    int x = 4; // compiles ok
    x = 5;  // compiles ok
    x = 3.14159; // compiler error
    x = "Hello world!"; // compiler error
    y = "abc123"; // compiler error
  }
}
```

Whitespace in Python

- **Important: whitespace has meaning in Python.**
- This is a key difference between Python and languages like C, C++, C#, Java, etc. This difference often causes difficulties for novice Python programmers who have previously learned a language such as Java

PEP 8 indentation guidelines

- https://www.python.org/dev/peps/pep-0008/#code-lay-out
- PEP 8 style guidelines: “Use 4 spaces per indentation level.” **Not 2 spaces, and not a TAB character**
- This applies to all indented code blocks, e.g. the indented code blocks in `if` statements, `else` statements, `elif` statements, functions, etc.
- When writing code in your IDE (e.g. PyCharm) or in your text editor (e.g. Notepad++) you can use the TAB key as a shortcut to indent by 4 spaces when starting an indented code block. However, you need to make sure that the TAB key is set up to enter 4 spaces when it is pressed, rather than a TAB character. This should already be the case in PyCharm, but you may need to change the default settings in Notepad++
- *Tip: to check your indentation settings are correct in Notepad++, go to Settings -> Preferences -> Language. Under "Tab Settings", highlight “python", then uncheck "Use default value", make sure the tab size is 4, and check "Replace by space".*



NUI Galway
OÉ Gaillimh

Topic 02 – Introduction to Python for ML

Part 04 – Modules, packages and virtual environments

Modules

- https://docs.python.org/3/tutorial/modules.html
  - "A module is a file containing Python definitions and statements. The file name is the module name with the suffix .py appended."
- https://docs.python.org/3/glossary.html#term-module
  - "An object that serves as an organizational unit of Python code. Modules have a **namespace** containing arbitrary Python objects. Modules are loaded into Python by the process of **importing**."
- Modules can be run either as standalone scripts, or they can be **imported** into other modules so that their built-in variables, functions, classes etc. can be used.
- Typically, modules group together statements, functions, classes, etc. with related functionality
- Up to now, our programs have consisted of a single module (.py script file) only. When developing larger programs, it is convenient to split the source code into separate modules.
- As well as creating our own modules to break up our source code into smaller units, we can also **import** built-in modules that come with Python, as well as modules developed by third parties

Built-in modules

- “The Python source distribution has long maintained the philosophy of "batteries included" -- having a rich and versatile standard library which is immediately available, without making the user download separate packages.” (https://www.python.org/dev/peps/pep-0206/#batteries-included-philosophy)
- Python provides a comprehensive set of built-in modules for commonly used functionality, e.g. mathematical functions, date & time, error handling, random number generation, handling command line arguments, parallel processing, networking, sending email messages, etc.
- Examples of modules that are built-in to Python include `math`, `string`, `argparse`, `calendar` etc.
- The `math` module is one of the most commonly used modules in the Python language – we will use some of the functions provided by the `math` module later. **Note:** the functions in the `math` module do not support complex numbers. If you need complex number support, you can use the `cmath` module
- A full list of built-in modules is available at https://docs.python.org/3/py-modindex.html

Packages

- https://docs.python.org/3/tutorial/modules.html#packages
- “Packages are a way of structuring Python’s module namespace by using "dotted module names". For example, the module name A.B designates a submodule named B in a package named A. Just like the use of modules saves the authors of different modules from having to worry about each other’s global variable names, the use of dotted module names saves the authors of multi-module packages like NumPy or Pillow from having to worry about each other’s module names.”
- “Suppose you want to design a collection of modules (a "package") for the uniform handling of sound files and sound data. There are many different sound file formats (… e.g. .wav, .aiff, .au), so you may need to create and maintain a growing collection of modules for the conversion between the various file formats. There are also many different operations you might want to perform on sound data…” On the right is an example package structure, expressed in terms of a hierarchical filesystem
- Individual modules can be imported, e.g. `import sound.effects.echo`

```
sound/
      __init__.py
      formats/
               __init__.py
               wavread.py
               wavwrite.py
               aiffread.py
               aiffwrite.py
               auread.py
               auwrite.py
               ...
      effects/
               __init__.py
               echo.py
               surround.py
               reverse.py
               ...
      filters/
               __init__.py
               equalizer.py
               vocoder.py
               karaoke.py
               ...
```

Package naming conventions

- https://www.python.org/dev/peps/pep-0008/#package-and-module-names
- PEP 8: “Modules should have short, all-lowercase names. Underscores can be used in the module name if it improves readability. Python packages should also have short, all-lowercase names, although the use of underscores is discouraged.”

Virtual environments

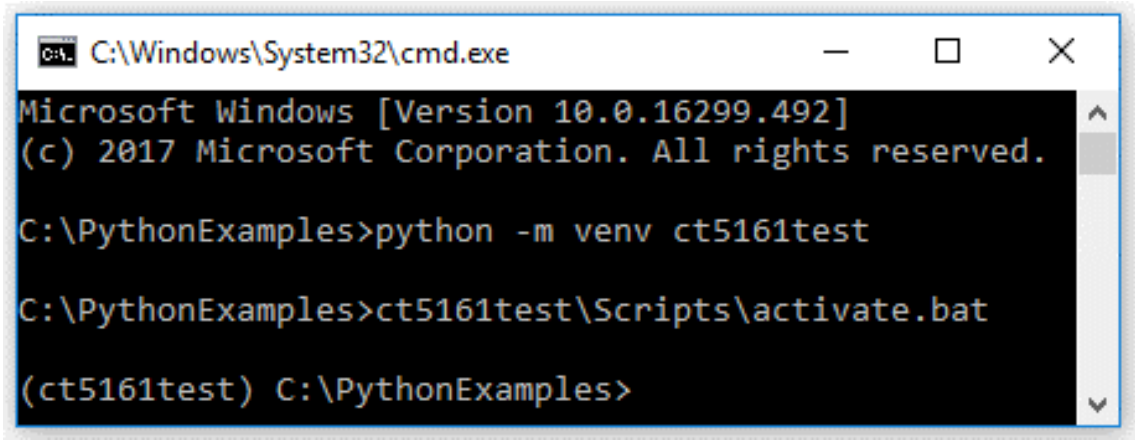
- https://docs.python.org/3/tutorial/venv.html#virtual-environments-and-packages
- “Python applications will often use packages and modules that don’t come as part of the standard library. Applications will sometimes need a specific version of a library, because the application may require that a particular bug has been fixed or the application may be written using an obsolete version of the library’s interface.”
- “This means it may not be possible for one Python installation to meet the requirements of every application. If application A needs version 1.0 of a particular module but application B needs version 2.0, then the requirements are in conflict and installing either version 1.0 or 2.0 will leave one application unable to run.”
- “The solution for this problem is to create a **virtual environment**, a self-contained directory tree that contains a Python installation for a particular version of Python, plus a number of additional packages.”
- “Different applications can then use different virtual environments. To resolve the earlier example of conflicting requirements, application A can have its own virtual environment with version 1.0 installed while application B has another virtual environment with version 2.0. If application B requires a library be upgraded to version 3.0, this will not affect application A’s environment.”

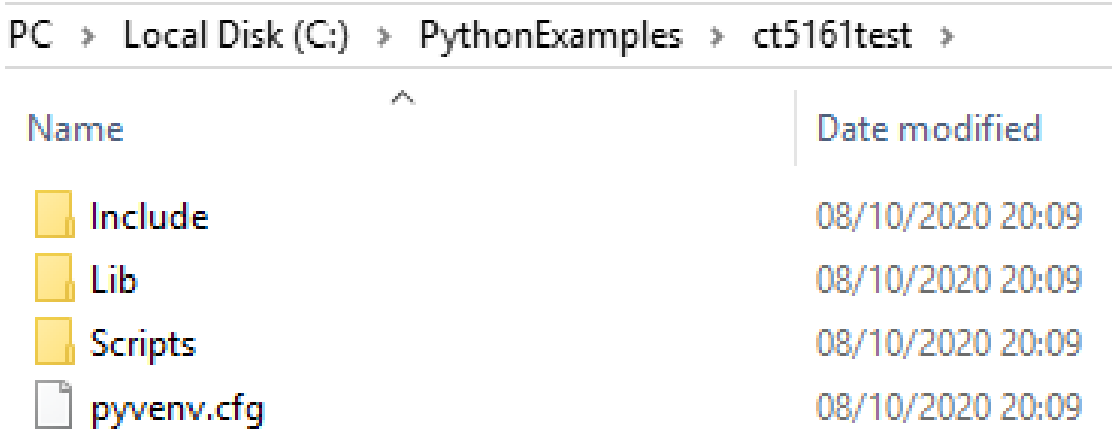
Creating and using virtual environments

- By default, most IDEs (e.g. PyCharm) will create a new virtual environment for each new project that you create. It is also possible to set up a project to run on a specific preconfigured virtual environment
- The built-in module venv can also be used to create and manage virtual environments through the console

Using the venv module

- First decide where you want the virtual environment to be created, then open a command prompt in that location and type in the command `python –m venv environmentname` to create a virtual environment with the specified name. You should then see the directory containing the virtual environment appear on the file system, with a similar structure to the example.
- On Windows, you can activate a virtual environment from its parent directory using the command
  - `environmentname\Scripts\activate.bat`
- On Unix or Mac, use
  - `source environmentname/bin/activate`
- Note that the command prompt window shows which Python virtual environment you are using, e.g. `(ct5161test)`



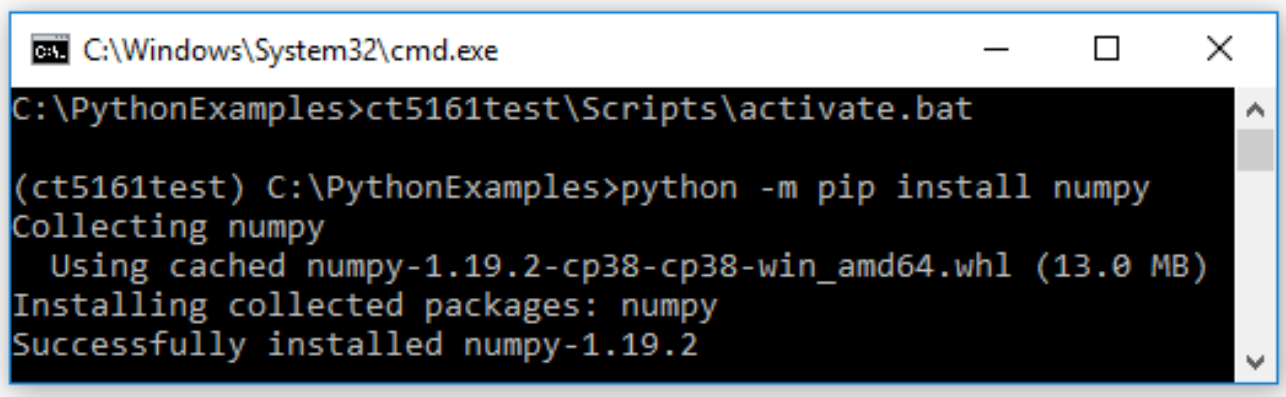


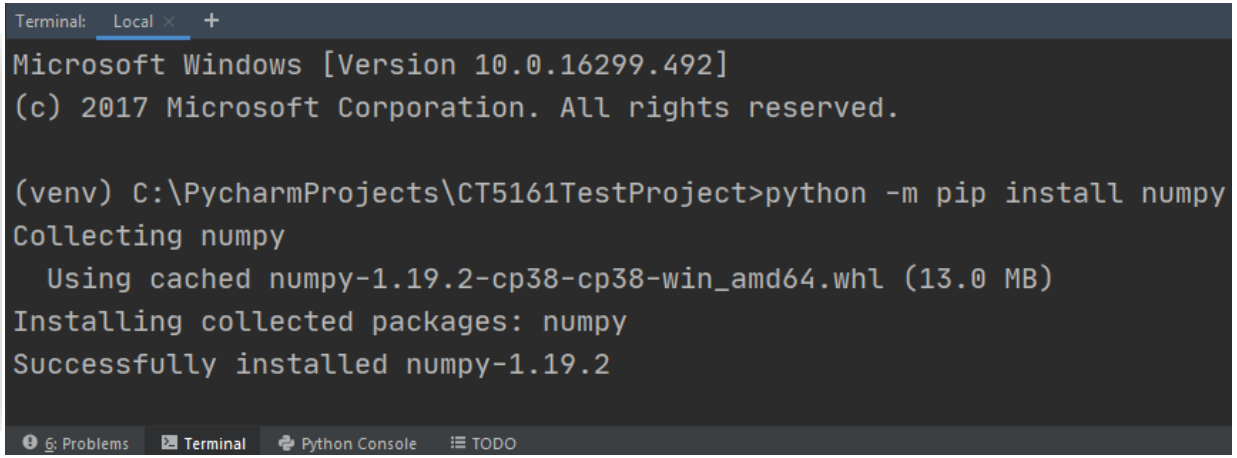
Managing packages with pip

- https://docs.python.org/3/tutorial/venv.html#managing-packages-with-pip
- "You can install, upgrade, and remove packages using a program called `pip`. By default pip will install packages from the Python Package Index (PyPI), https://pypi.org. You can browse the Python Package Index by going to it in your web browser." `pip` is supplied by default with your Python installation.
- You can use the command `python -m pip install projectname` from the console to install packages from PyPI (see https://packaging.python.org/tutorials/installing-packages/#installing-from-pypi)
- **Note:** in most cases, you should not install packages directly to your main Python installation. Instead, create a virtual environment and install packages within that virtual environment. This keeps your main Python environment clean, and helps to avoid issues with different package versions, certain Python applications requiring different versions of the same package, etc.
- You can upgrade a package to the latest version, e.g. `python -m pip install --upgrade pip` (this command will upgrade `pip`, other packages may be upgraded in a similar manner)

Installing a package in a virtual environment

- To test out `pip`, let's install `numpy` (https://numpy.org/) in the virtual environment that we created earlier
- `numpy` is a very commonly used Python package, that supports computations using arrays and matrices, along with various linear algebra operations
- To install `numpy` from the command prompt, first activate the virtual environment that you plan to install it to, then enter the command `python -m pip install numpy`
- To install `numpy` to a PyCharm project's virtual environment, first open the terminal by clicking the button at the bottom of the IDE window, the enter the command above





Testing out the installed numpy package

- This simple program uses `numpy` to calculate the Euclidean distance between two points: $(X_1, Y_1) = (1,2)$ and $(X_2, Y_2) = (3,4)$
- Note the use of the `as` keyword in the `import` statement, binding the `numpy` package to the identifier `np` within the module **testnumpy.py**
- The call to `numpy.array()` can be used to create arrays (in this case 1 dimensional arrays). We will cover the `numpy` package in detail later.
- Euclidean distance is also referred to as the L^2 norm, hence the call to the `.norm()` function

2D Distance Calculator

(X_1, Y_1) = 1,2

(X_2, Y_2) = 3,4

Clear | Calculate

Answer:

d = 2.828427

For:
$(X_1, Y_1) = (1, 2)$
$(X_2, Y_2) = (3, 4)$

Distance Equation Solution:

$$d = \sqrt{(3-1)^2 + (4-2)^2}$$

$$d = \sqrt{(2)^2 + (2)^2}$$

$$d = \sqrt{4+4}$$

$$d = \sqrt{8}$$

$$d = 2.828427$$

File: testnumpy.py

```
import numpy as np
a = np.array((1, 2))
b = np.array((3, 4))
dist = np.linalg.norm(a-b)
print("The distance is:", dist)
```

```
C:\Windows\System32\cmd.exe
C:\PythonExamples>ct5161test\Scripts\activate.bat

(ct5161test) C:\PythonExamples>python testnumpy.py
The distance is: 2.8284271247461903
```

https://www.calculatorsoup.com/calculators/geometry-plane/distance-two-points.php?a=1%2C2&b=3%2C4&action=solve

Adding your virtual environment to JupyterLab

- If you have installed your packages (e.g., numpy, pandas) to a virtual environment, you will need to make that virtual environment available to Jupyter Lab so that your .ipynb flies can be executed on the correct environment
- You can use the package `ipykernel` to do this
  - install this package to your virtual environment using the command `python -m pip install ipykernel`

```
(ct4101) C:\PythonExamples>python -m ipykernel install --user --name=ct4101
Installed kernelspec ct4101 in C:\Users\        \AppData\Roaming\jupyter\kernels\ct4101
```

Putting all of this to use …

- Example on Blackboard in archive **ct4101_02_code_samples.zip**
  - Jupyter Lab notebook **linear_regression_svd.ipynb**
  - file **external_data.csv**
- Try to download and run this notebook yourself later. You will need to have Python + Jupyter Lab installed correctly.
- You will also need to have numpy, pandas, matplotlib, scikit-learn and ipykernel installed in a virtual environment
- You could also try running this notebook on a cloud-based service such as Google Colab

Suggested installation steps for Windows 10 (1/2)

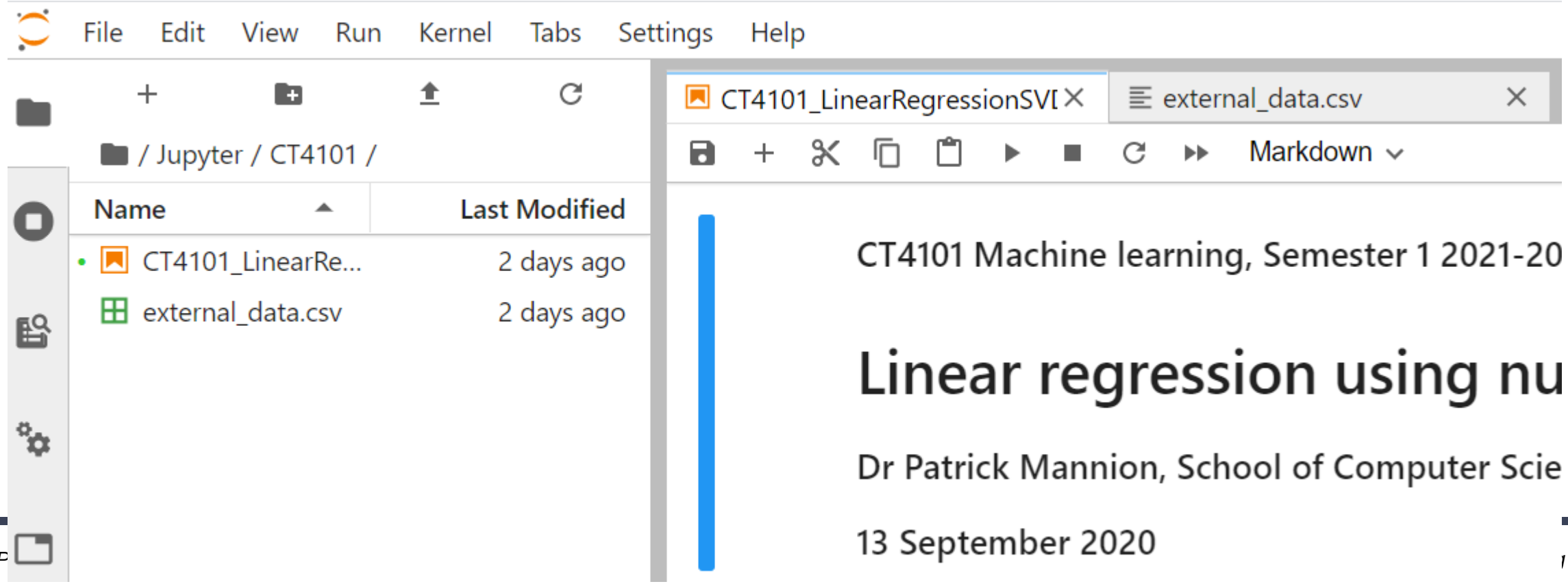
1. Install Python 3.9.6 from https://www.python.org/downloads/
2. Open a command prompt (on windows type 'cmd' into the search bar)
3. Type the command `python –m pip install jupyterlab` into the command prompt and press return
4. Create a folder to hold virtual environments (e.g., C:\PythonEnvs)
5. Return to the command prompt and navigate to the directory you just created using the command `cd C:\PythonExamples`
6. Create a virtual environment for CT4101 using the following command `python –m venv ct4101`
7. Activate the virtual environment you just created by entering the command `ct4101\scripts\activate.bat`

Suggested installation steps for Windows 10 (2/2)

8. **With the virtual environment still active**, install numpy in the ct4101 virtual environment with command `python -m pip install numpy`
9. Install pandas in the ct4101 virtual environment with command `python -m pip install pandas`
10. Install matplotlib in the ct4101 virtual environment with command `python -m pip install matplotlib`
11. Install scikit-learn in the ct4101 virtual environment with command `python -m pip install sklearn`
12. Install ipykernel in the ct4101 virtual environment with command `python -m pip install ipykernel`
13. Make the ct4101 virtual environment available to Jupyter Lab with command `python -m ipykernel install --user --name=ct4101`

Running the example .ipynb in Jupyter Lab

- Once you have Jupyter Lab installed, open a command prompt, type in the command `jupyter lab` and press return. This will launch the Jupyter Lab interface in your web browser. You can then use the file browser pane on the left of the Jupyter Lab interface to choose the .ipynb file that you would like to open.



Switching execution environments in Jupyter Lab

- Make sure that you are using the correct execution environment – it should be the ct4101 virtual environment that you just created (if not you will get errors stating that required packages are not found)
- There is a button on the top right of the Jupyter Lab web browser interface that allows you to switch execution environments (it is labelled ‘Switch kernel’)