

Introduction to Python

What is Python?

- ▶ Python is a powerful high-level, interpreted, object-oriented programming language
- ▶ Created by Guido van Rossum and first released in 1991
- ▶ It has a simple and easy-to-use syntax
- ▶ Features a dynamic type system and automatic memory management
- ▶ Has a rich variety of native data structures such as lists, tuples, sets and dictionaries
- ▶ Allows you to create wide range of applications, from Web applications and scientific applications to desktop graphical user interfaces



What is Python?

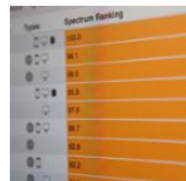
- ▶ Python is ranked as the leading programming language in all recent surveys

31 Jul 2018 | 15:00 GMT

The 2018 Top Programming Languages

Python extends its lead, and Assembly enters the Top Ten























By Stephen Cass



Explore the Interactive Rankings

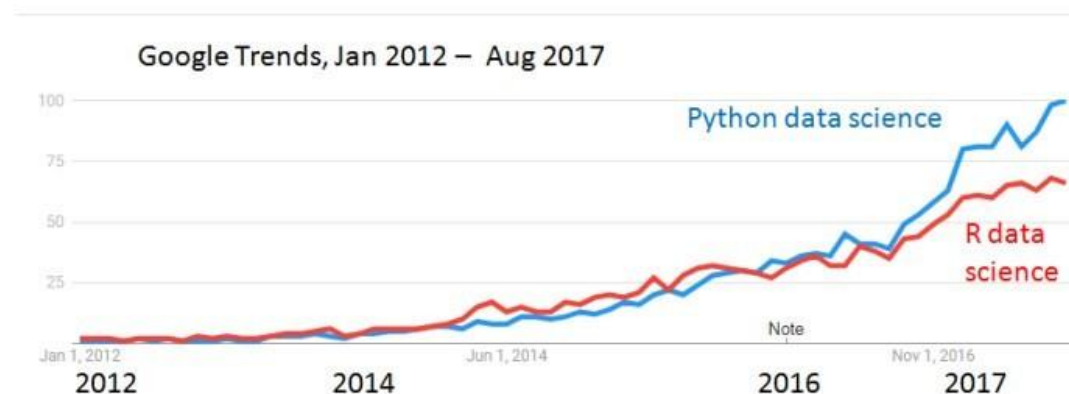
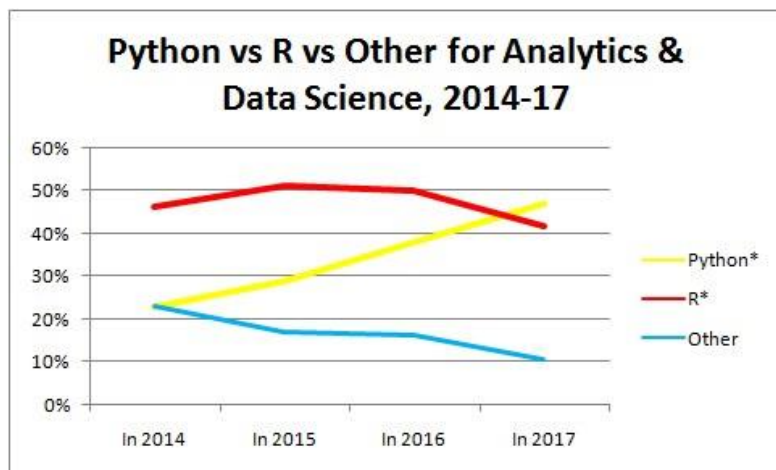
Welcome to *IEEE Spectrum's* [fifth annual interactive ranking of the top programming languages](#). Because no one can peer over the shoulders of every coder out there, anyone attempting to measure the popularity of computer languages must rely on proxy measures of relative popularity. In our case, this means combining metrics from multiple sources to rank 47 languages. But recognizing that different programmers have different needs and domains of interest, we've chosen not to blend all those metrics up into One Ranking to Rule Them All.

Instead, our interactive app lets you choose how these metrics are weighted when they are combined, so you can

Language Rank	Types	Spectrum Ranking
1. Python	  	100.0
2. C++	  	98.4
3. C	  	98.2
4. Java	  	97.5
5. C#	  	89.8
6. PHP		85.4
7. R		83.3
8. JavaScript	 	82.8
9. Go	 	76.7
10. Assembly		74.5

Python vs. R for Data Science

- ▶ Python and R are the two most popular programming languages used by data analysts and data scientists
- ▶ R is a language specialized for handling statistics and big data
- ▶ The existence of high-quality Python libraries for both statistics and machine learning makes Python a more attractive jumping-off point than the more specialized R



Python Vs. C

- ▶ The following program output a list of names on separate lines
- ▶ The same program is written in C and in Python

```
names = ["Isaac Newton", "Marie Curie", "Paul Dirac"]  
for name in names:  
    print(name)
```

```
#include <stdio.h>  
#include <string.h>  
#define MAX_STRING_LENGTH 20  
#define NUMBER_OF_STRINGS 3  
  
int main()  
{  
    char names[NUMBER_OF_STRINGS][MAX_STRING_LENGTH + 1];  
    int i;  
  
    strcpy(names[0], "Isaac Newton");  
    strcpy(names[1], "Marie Curie");  
    strcpy(names[2], "Paul Dirac");  
  
    for (i = 0; i < NUMBER_OF_STRINGS; i++) {  
        printf("%s\n", names[i]);  
    }  
  
    return 0;  
}
```

Python Advantages

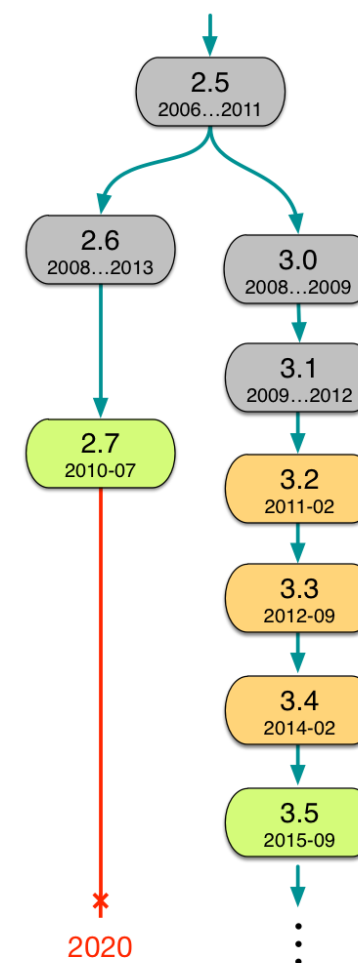
- ▶ Its clean and simple syntax makes writing Python programs fast
 - ▶ Python's syntax aims to ensure that “There should be one – and preferably only one – obvious way to do it”
- ▶ It's free – Python and its associated libraries are free of cost and open source
 - ▶ Unlike commercial offerings such as Matlab or Mathematica
- ▶ Cross-platform support: Python is available for every commonly available operating system, including Windows, Unix, Linux and Mac OS
 - ▶ It is possible to write code that will run on any platform without modification
- ▶ A “multi-paradigm” language that contains the best features from the procedural, object-oriented and functional programming paradigms
- ▶ Has a large library of modules and packages that extend its functionality
 - ▶ Many of these are available as part of the “standard library” provided with Python itself
 - ▶ Others, including NumPy, Pandas and Scikit-learn used in data science can be downloaded separately for no cost

Python Disadvantages

- ▶ The speed of execution of a Python program is not as fast as some other, fully compiled languages such as C and Fortran
 - ▶ For heavily numerical work, the NumPy and SciPy libraries alleviate this to some extent by using compiled-C code “under the hood”, but at the expense of some reduced flexibility
 - ▶ For most application the speed difference is not noticeable
- ▶ It is hard to hide or obfuscate the source code of a Python program to prevent others from copying or modifying it
- ▶ There are some compatibility issues between different Python versions
 - ▶ Especially between Python versions 2 and 3

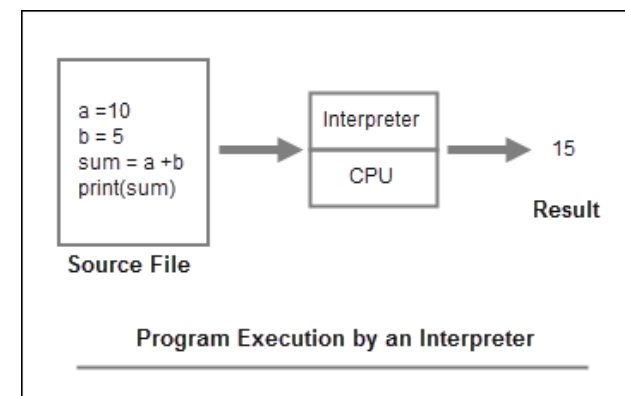
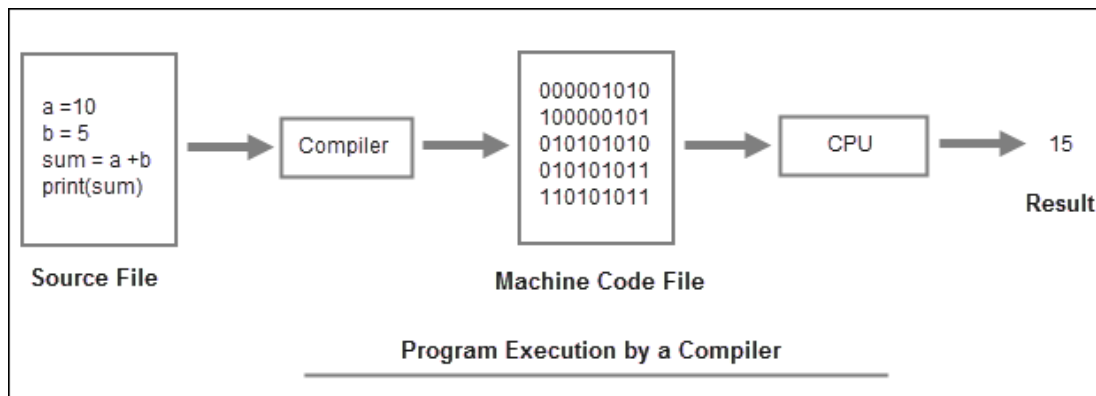
Python Versions

- ▶ Python 2.x is legacy, Python 3.x is the present and future of the language
 - ▶ All recent standard library improvements are only available in Python 3.x
- ▶ Python 3 is not backward-compatible with Python 2
- ▶ Python 3.7.0 is the newest major release of the Python language
 - ▶ Released on June 27, 2018



Compiler vs. Interpreter

- ▶ A program written in a high-level language is called a **source code**
- ▶ We need to convert the source code into machine code (written in binary) before the program can be executed
- ▶ This can be accomplished by compilers and interpreters:
 - ▶ A **compiler** scans the entire program and translates it as a whole into machine code
 - ▶ An **interpreter** translates the program one statement at a time



Compiler vs. Interpreter

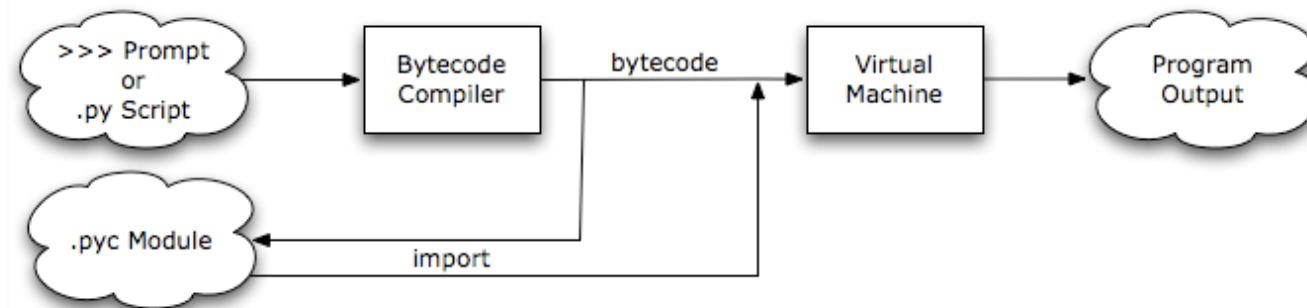
Compiler	Interpreter
Translates the entire program at once	Translates the program line-by-line
Programs are compiled once and run anytime	Programs are interpreted line-by-line every time they are run
Compiled programs execute control statements (like if-else) faster than interpreted programs	Interpreter executes control statements at a slower speed
Compiled programs take more memory because the entire object code needs to reside in memory	Interpreter doesn't generate intermediate object code or machine code, thus is more memory efficient
Errors are reported after the entire program is checked	An error is reported as soon as the first error is encountered, the rest of the program is not checked until the error is fixed
A program is not allowed to run until it is completely error free, which makes it harder to debug	The program runs from the first line and stops execution only if it encounters an error, thus debugging is easier
Examples for programming languages that use compilers: C/C++, Java, C#	Examples for programming languages that use interpreters: Python, PHP, Perl, Ruby, Matlab, Javascript, Lisp

Bytecode and Virtual Machines

- ▶ Python is a byte-code compiled system, in which the source code is translated to an intermediate language known as **bytecode**
- ▶ Bytecode is not a machine code of any particular computer, thus it can run on any CPU architecture without modification, which is a huge benefit over compiled code
- ▶ On the other hand it is lower level than the original source code, thus it runs quicker than interpreted code
- ▶ Bytecode is run on a **Virtual Machine** (VM), which interprets it into machine-code instructions that are understandable by the specific architecture the program is running on
- ▶ Python source code (.py) can be compiled into different byte codes, such as CPython bytecode (.pyc), IronPython (.Net), or Jython (JVM)

The Python Interpreter

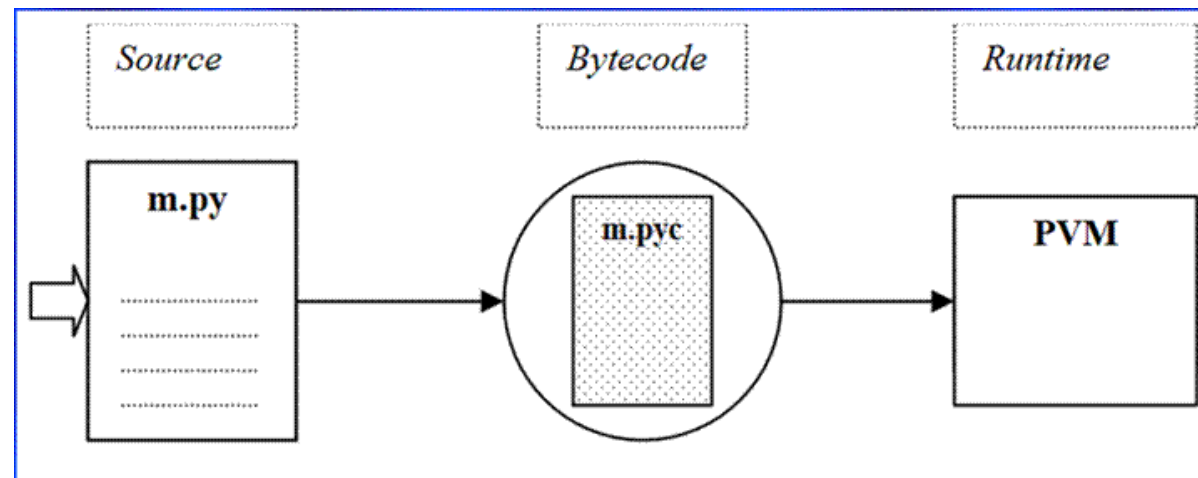
- ▶ The Python interpreter consists of two parts:
 - ▶ A **bytecode compiler** that translates the Python source code into bytecode
 - ▶ A **Python Virtual Machine** (PVM), which executes Python bytecode



- ▶ The Python interpreter can be used in two modes:
 - ▶ Interactive mode – a command line shell which gives immediate feedback for each statement
 - ▶ Script mode – the Python interpreter runs the entire program from the source file

CPython

- ▶ CPython is the standard and most-widely implementation of the Python programming language, written in C
- ▶ When you run a Python script (.py), the CPython interpreter, caches the translated bytecode in **.pyc** files
- ▶ When you next run your code, the Bytecode file can be used instead of your source code, which will result in quicker execution times



Just-In-Time Compilation

- ▶ A **JIT compiler** compiles parts of the bytecodes into machine code, just before they are about to be executed (hence the name “just-in-time”)
 - ▶ The compiled code is then cached and can be reused later, so the program can run faster
- ▶ **PyPy** is an alternative implementation of the Python programming language which often runs faster than CPython
 - ▶ Because it is a just-in-time compiler, while CPython is an interpreter
- ▶ Most Python code runs well on PyPy, except for code that relies on CPython extensions, which either does not work or incurs some overhead when run in PyPy

Python Installation

- ▶ The official website of Python is <http://www.python.org>
- ▶ It contains full and easy-to-follow instructions for downloading Python
- ▶ However, there are several full distributions which include the data science libraries such as NumPy, SciPy, Scikit-learn save you from having to download and install these yourself
- ▶ For Windows, the Anaconda distribution is typically used

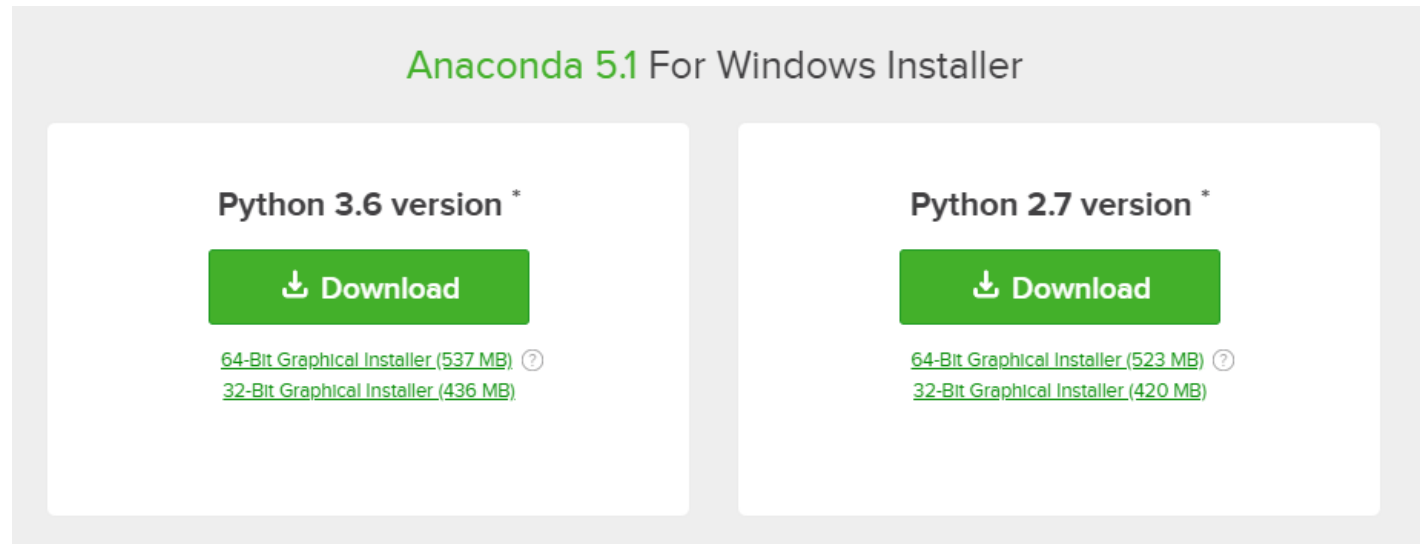
Anaconda Distribution

- ▶ Anaconda is the most popular Python data science platform
- ▶ Anaconda is a distribution of the Python and R programming languages for data science and machine learning related applications
- ▶ It also installs the Jupyter Notebook
- ▶ Includes a collection of over 1,000 open source data science packages
- ▶ Package versions are managed by the package management system conda
- ▶ In contrast to pip, conda can handle library dependencies outside of the Python packages as well as the Python packages themselves



Installing Anaconda

- ▶ Go to <https://www.anaconda.com/download/>
- ▶ Download the Python 3.6 version

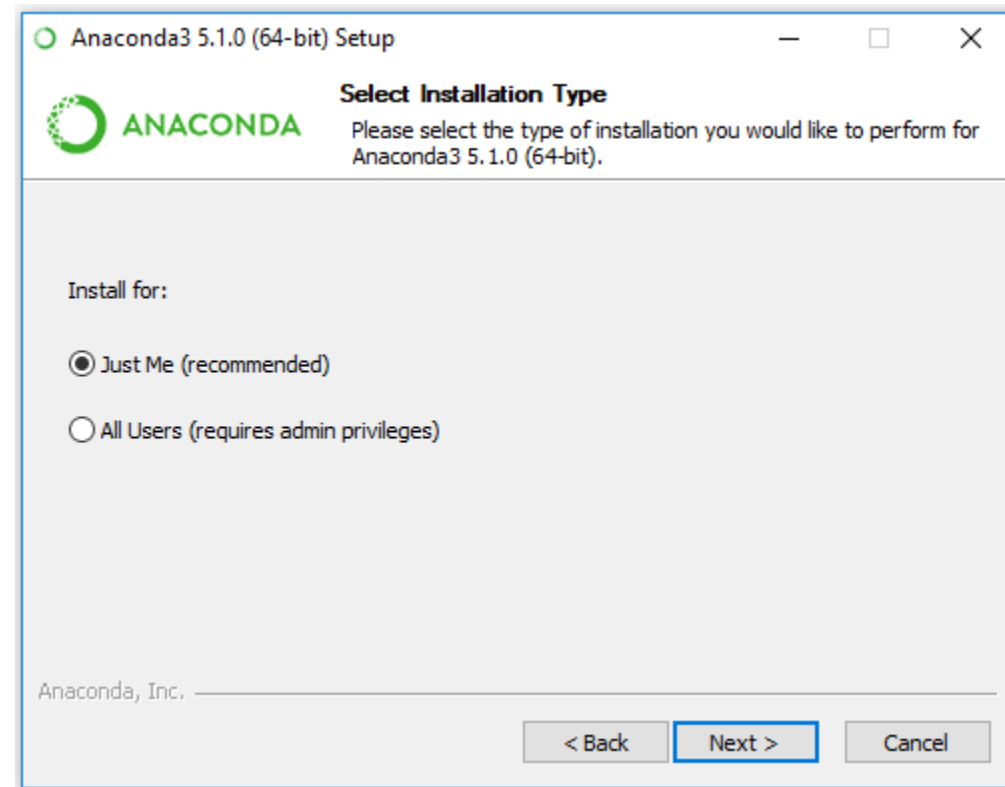


Installing Anaconda

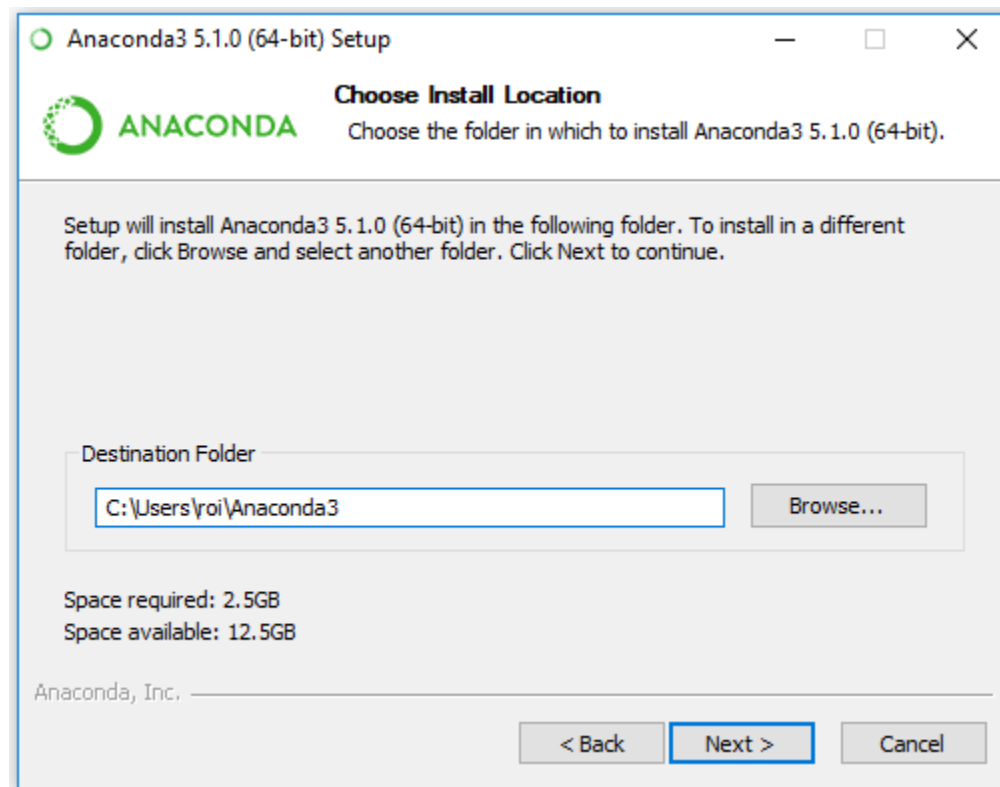
- ▶ Double click the executable file to start the installation



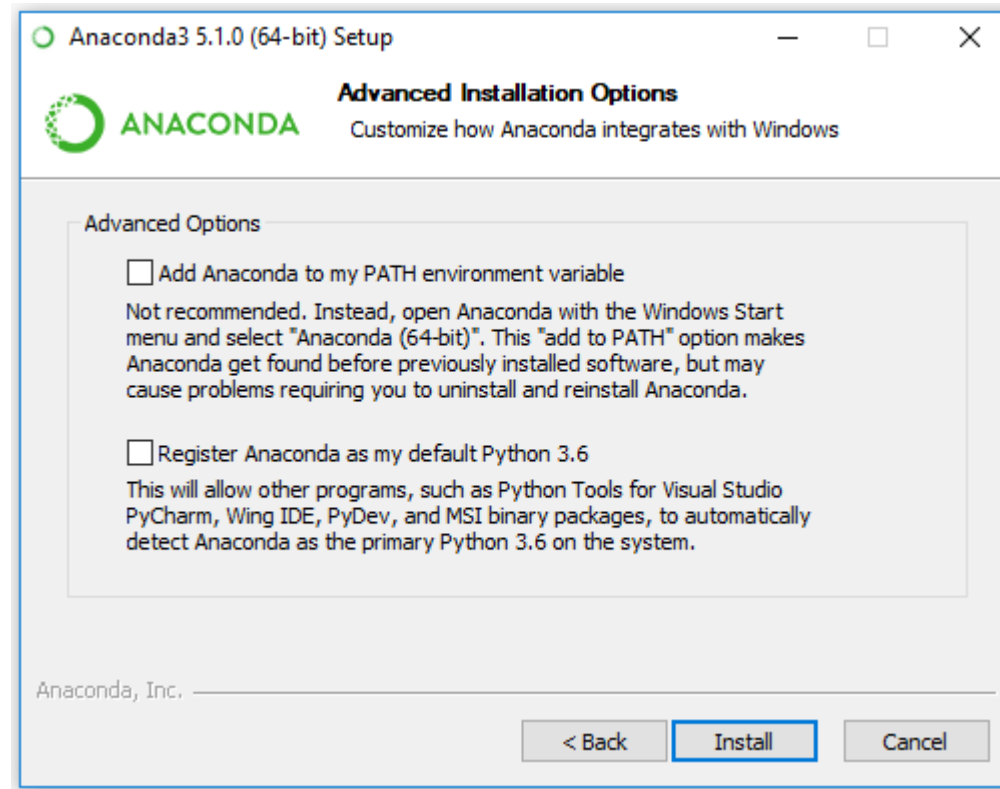
Installing Anaconda



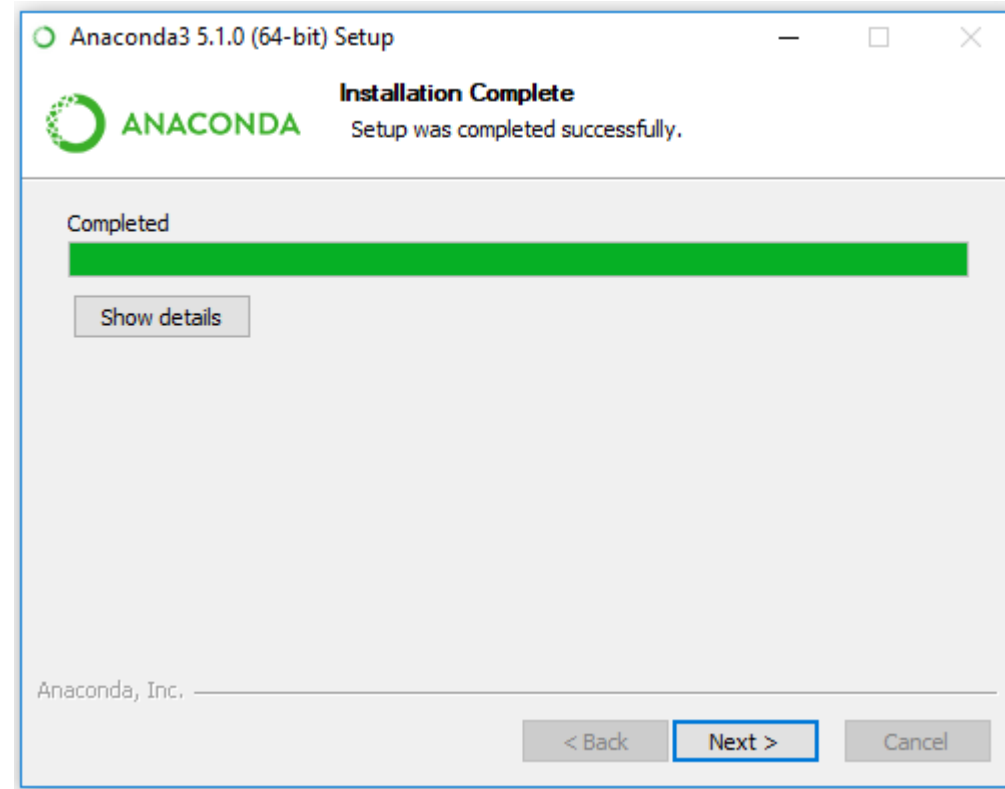
Installing Anaconda



Installing Anaconda

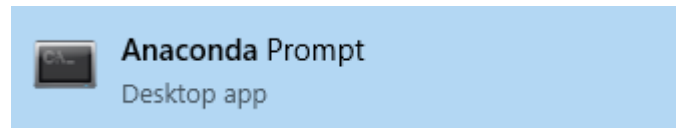


Installing Anaconda

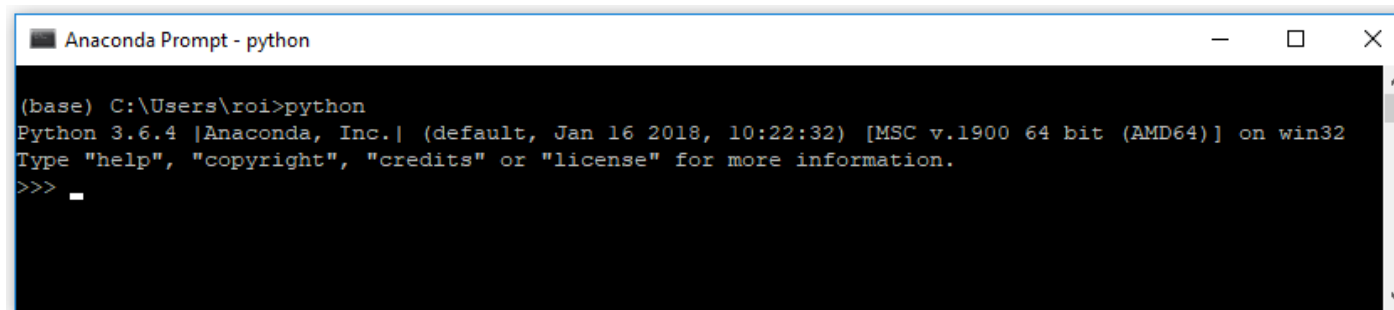


The Python Shell

- ▶ The Python shell is an interactive environment: the user enters Python statements that are executed immediately after the *Enter* key is pressed
- ▶ To start a Python shell from the command line, first open the Anaconda command prompt from the Window start menu:



- ▶ Then type python:

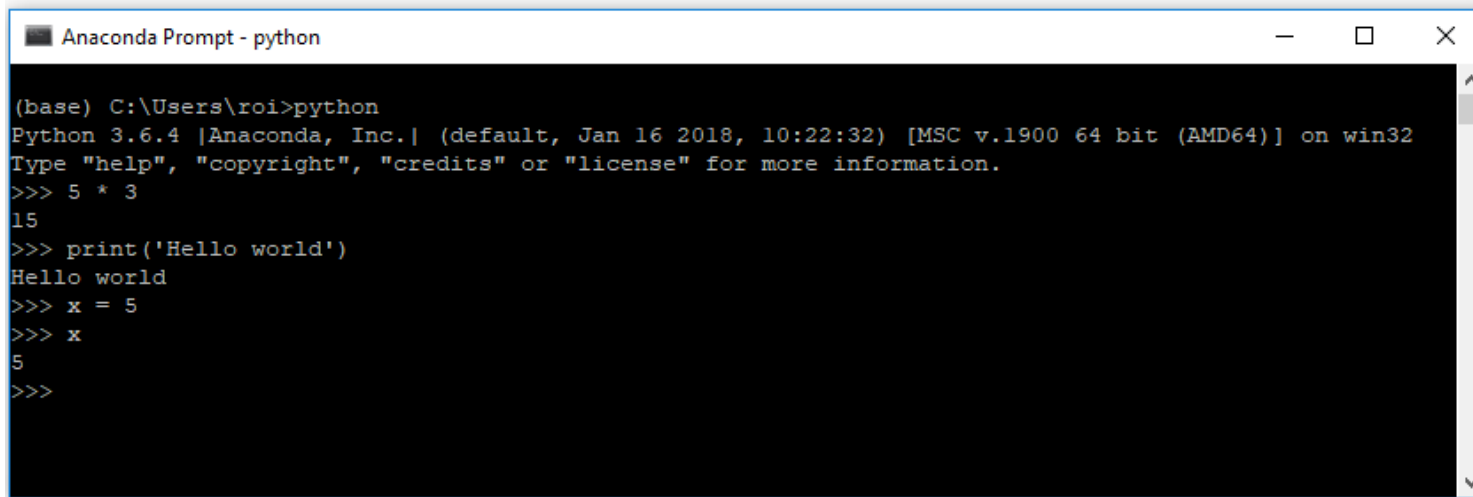


```
(base) C:\Users\roi>python
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 10:22:32) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

- ▶ The three chevrons (>>>) are the **prompt**, which is where you enter your Python commands

The Python Shell

- ▶ Enter the following calculations one by one and hit enter to get the result.



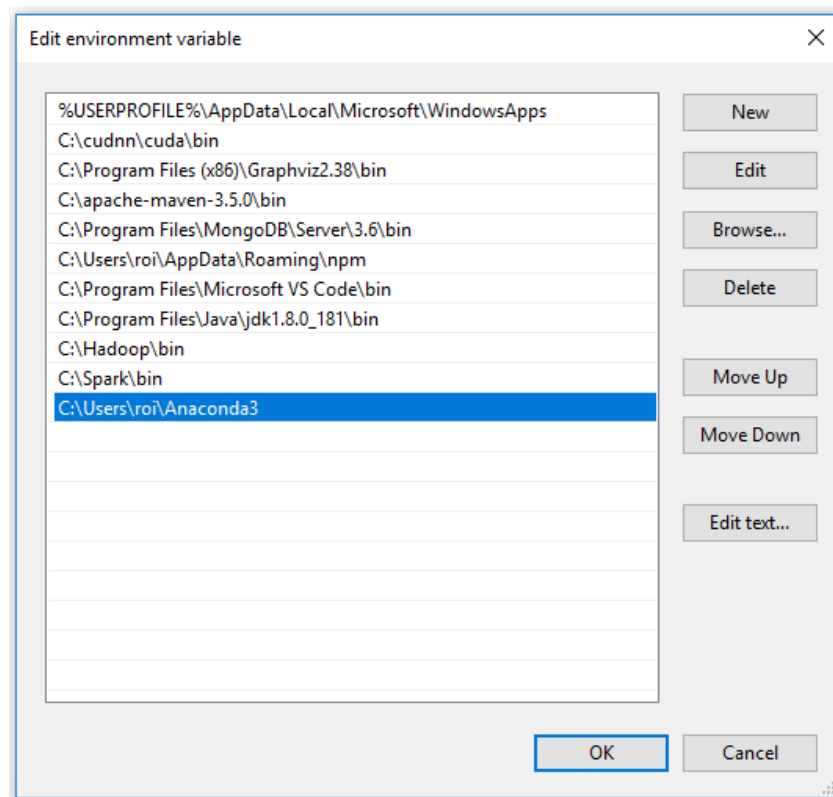
```
Anaconda Prompt - python

(base) C:\Users\roi>python
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 10:22:32) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 5 * 3
15
>>> print('Hello world')
Hello world
>>> x = 5
>>> x
5
>>>
```

- ▶ In Python, we use `print()` function to print something to the screen
- ▶ To exit the Python shell, type **`exit()`** or Ctrl-Z and hit Enter

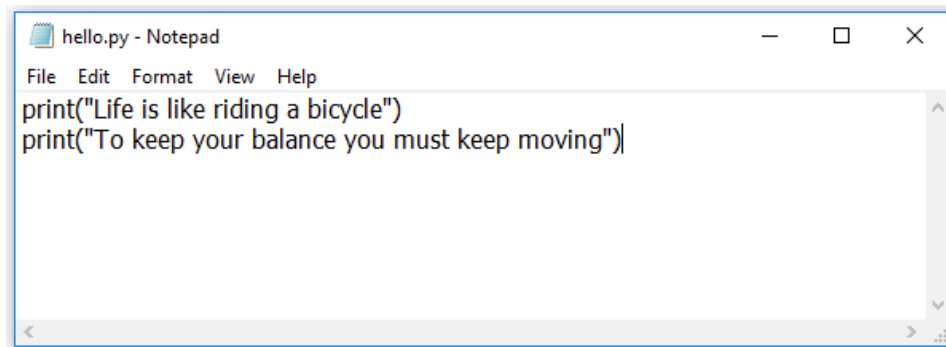
The Python Shell

- ▶ To enable running the Python shell from any location in your computer, you can add python.exe to your PATH environment variable:



Script Mode

- ▶ Python Shell is great for testing small chunks of code, but the statements you enter in the shell are not saved anywhere
- ▶ If you want to execute the same set of statements multiple times you would rather save the entire code in a script file (with extension .py), and then use the Python interpreter in script mode to execute the code in the file
- ▶ To create the script you can use any text editor
- ▶ For example, create a folder C:\Python, and create a new file hello.py in this folder
- ▶ Edit the file in Notepad and add the following code to it:

A screenshot of a Notepad window titled 'hello.py - Notepad'. The window has a menu bar with 'File', 'Edit', 'Format', 'View', and 'Help'. The text area contains two lines of Python code: `print("Life is like riding a bicycle")` and `print("To keep your balance you must keep moving")`. The cursor is at the end of the second line.

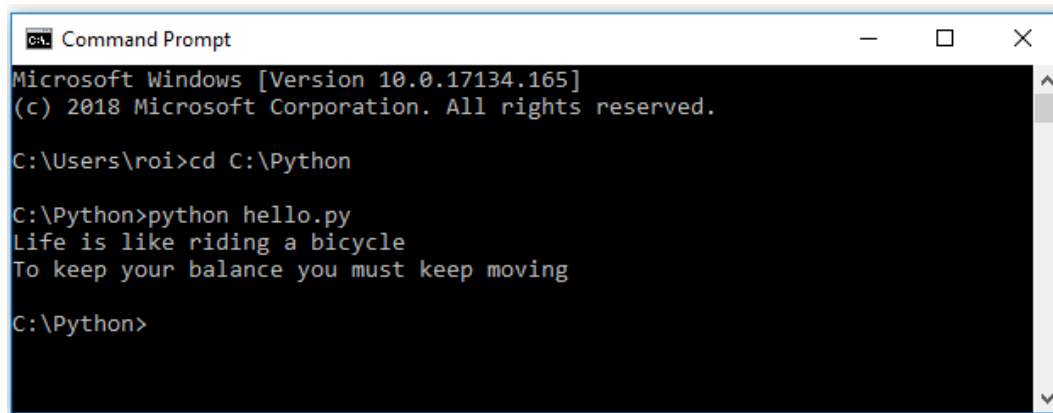
```
hello.py - Notepad
File Edit Format View Help
print("Life is like riding a bicycle")
print("To keep your balance you must keep moving")
```

Script Mode

- ▶ The file hello.py is called source file / script file / module
- ▶ To execute the program, open command prompt, change your current working directory to C:\Python using the cd command, then type the following command:

```
python hello.py
```

- ▶ This instructs the shell to invoke the Python interpreter, sending it the file hello.py as the script to execute
- ▶ Output from the program is then returned to the shell and displayed in your console

A screenshot of a Windows Command Prompt window. The title bar says 'Command Prompt'. The text inside shows the user navigating to 'C:\Python' and running 'python hello.py', which outputs 'Life is like riding a bicycle' and 'To keep your balance you must keep moving'.

```
Microsoft Windows [Version 10.0.17134.165]
(c) 2018 Microsoft Corporation. All rights reserved.

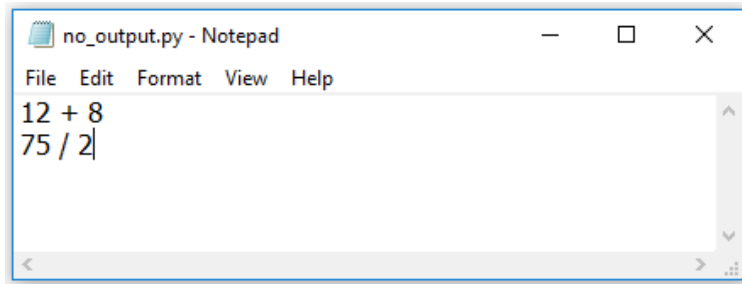
C:\Users\roi>cd C:\Python

C:\Python>python hello.py
Life is like riding a bicycle
To keep your balance you must keep moving

C:\Python>
```

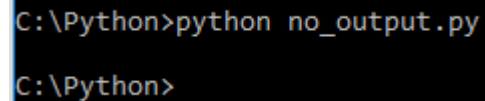
Script Mode

- ▶ To print values from a Python script you must explicitly use the `print()` function
- ▶ For example, create a new file named `no_output.py` with the following code:



```
no_output.py - Notepad
File Edit Format View Help
12 + 8
75 / 2
```

- ▶ To run the file enter the following command:



```
C:\Python>python no_output.py
C:\Python>
```

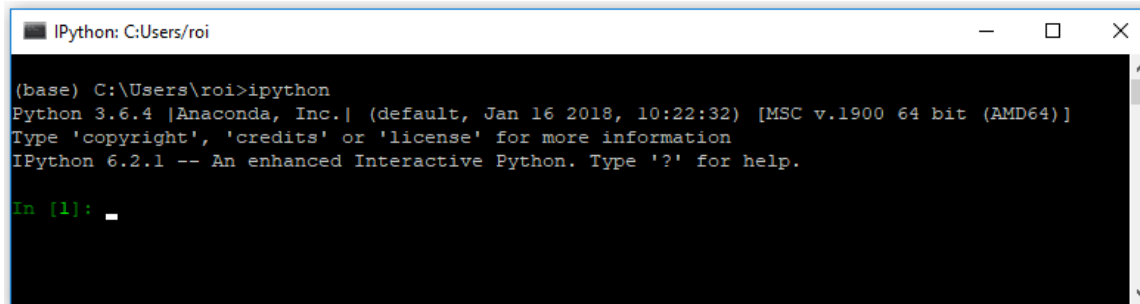
- ▶ As you can see, the program didn't output anything.

Exercise (1)

- ▶ Create a Python script in C:\Python named welcome_to_ds.py
- ▶ In the script print your name and age on two separate lines
- ▶ Run the script from the Python Shell

IPython Shell

- ▶ The IPython shell and the related interactive, browser-based IPython Notebook provide a powerful interface to the Python language
- ▶ IPython has several advantages over the native Python shell, including easy interaction with the operating system, introspection and tab completion
- ▶ IPython is included in the Anaconda distribution
- ▶ To start an interactive IPython session from the command line, simply type `ipython`:



```
IPython: C:\Users\roi
(base) C:\Users\roi>ipython
Python 3.6.4 [Anaconda, Inc.] (default, Jan 16 2018, 10:22:32) [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.

In [1]: _
```

- ▶ The prompt `In [1]:` is where you type your Python statements and replaces the native Python `>>>` shell prompt

IPython Shell

- ▶ The counter in square brackets increments with each Python statement or code block:

```
In [1]: 4 + 5
Out[1]: 9

In [2]: x = 10

In [3]: print(x)
10

In [4]: _
```

- ▶ To exit the IPython shell, type quit or exit (no parentheses are required)

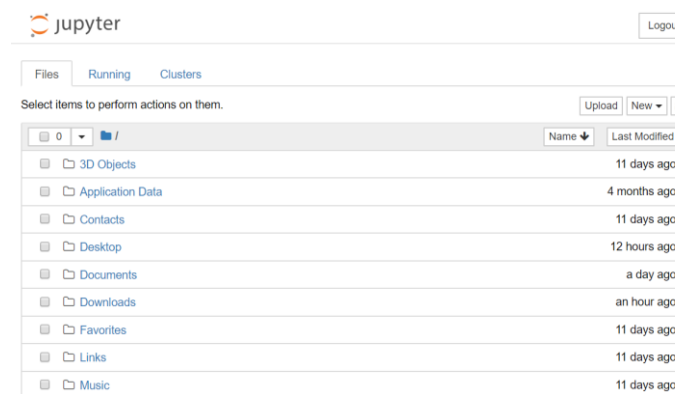
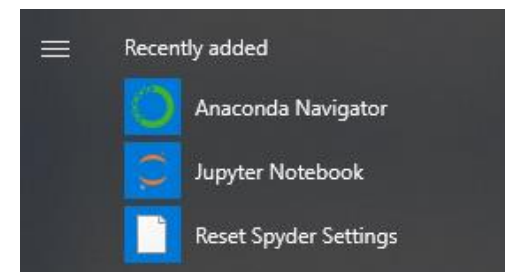
Jupyter Notebook

- ▶ **Jupyter Notebook** (formerly IPython Notebooks) provides an interactive environment for Python programming within a web browser
- ▶ Its main advantage over IPython shell is that Python code can be combined with documentation, rich text, images and even rich media such as embedded videos
- ▶ A **notebook document** is a JSON document, containing an ordered list of input/output cells, that ends with the “.ipynb” extension
 - ▶ Cells can contain code, text, mathematics, plots and rich media
- ▶ A **Jupyter kernel** is a program responsible for executing the code inside a notebook and communicating the results back to the browser.
 - ▶ By default Jupyter Notebook ships with the IPython kernel, but it has bindings for other languages as well (such as Julia and R)
- ▶ If you’ve installed the Anaconda distribution, you already have Jupyter Notebook!

Starting the Jupyter Notebook Server

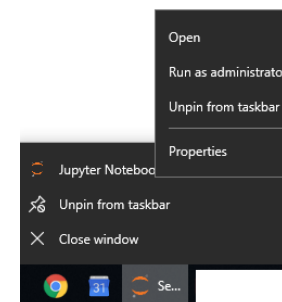
- ▶ Start the Jupyter Notebook server from the Windows-Start menu
- ▶ You can also start it from the command line by typing:

```
ipython notebook
```
- ▶ This will print some information about the notebook server in your terminal, and also open a browser window at the URL of the local IPython notebook application
- ▶ By default this is `http://127.0.0.1:8888`, though it will default to a different port if 8888 is in use



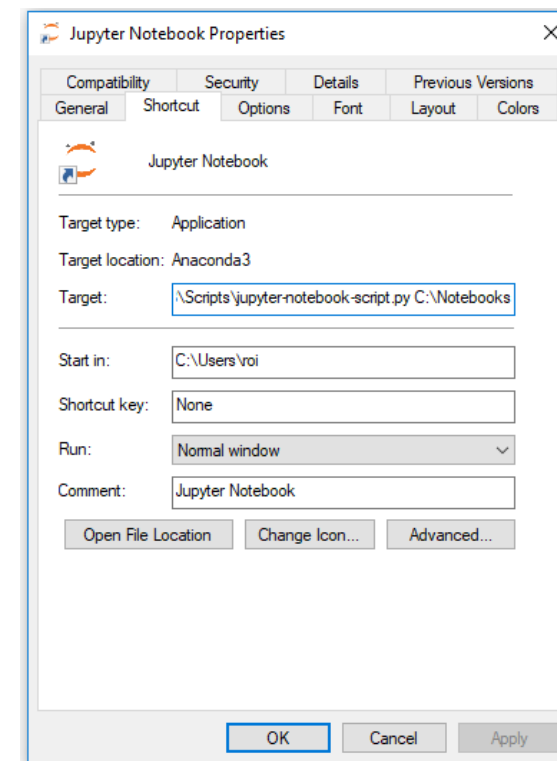
Starting the Jupyter Notebook Server

- ▶ The notebook index page contains a list of the notebooks currently available in the directory from which the notebook server was started
- ▶ This is also the default directory to which notebooks will be saved, so it is a good idea to run the notebook server somewhere convenient in your directory hierarchy for the project you are working on
- ▶ We'll change the startup directory for the notebook server to C:\Notebooks
- ▶ First create the directory C:\Notebooks
- ▶ Now change the shortcut to Jupyter Notebook to start the server from that folder
 - ▶ Right-click the Jupyter Notebook shortcut, and open its properties



Starting the Jupyter Notebook Server

- ▶ In the Target box remove %USERPROFILE% and type instead C:\Notebooks
- ▶ Click OK
- ▶ Close Jupyter Notebook and run it again
- ▶ Now you should see an empty workspace directory



Logout

Files

Running

Clusters

Select items to perform actions on them.

Upload

New ▼



0

/

Name ▼

Last Modified

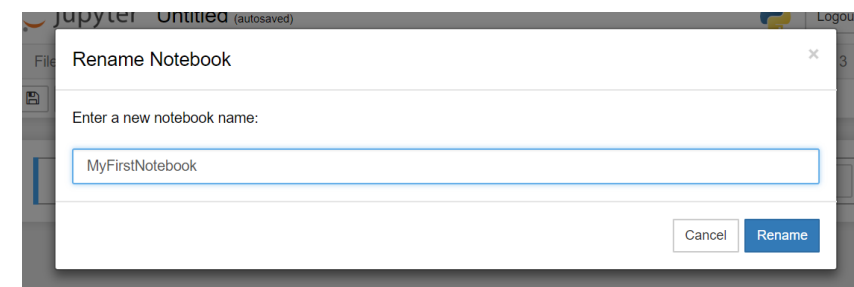
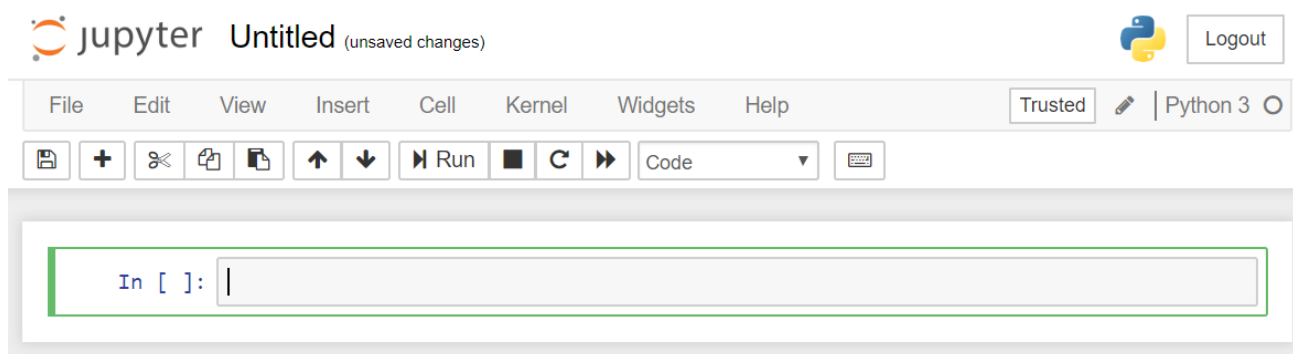
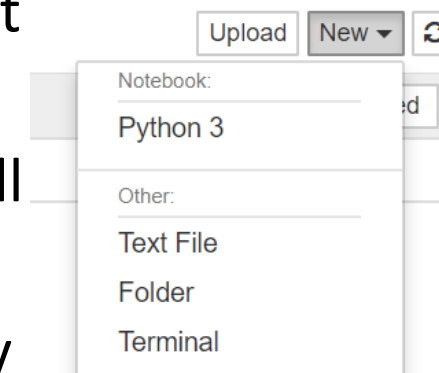
The notebook list is empty.

Notebook Index Page

- ▶ The index page contains three tabs:
 - ▶ **Files** lists the IPython notebooks and subdirectories within the current working directory
 - ▶ **Running** lists those notebooks that are currently active within your session (even if they are not open in a browser window)
 - ▶ **Clusters** provides an interface to IPython's parallel computing engine
- ▶ From the index page, you can start a new notebook (by clicking on “New Notebook”) or open an existing notebook (by clicking on its name)
- ▶ To import an existing notebook into the index page, drag the notebook file into the index listing from elsewhere on your operating system
- ▶ To stop the notebook server, press CTRL-C in the terminal window it was started from

Create a New Notebook

- ▶ To start a new notebook, click the “New Notebook” button and select Python 3
- ▶ This opens a new browser tab containing the interface where you will write your code and connects it to an IPython kernel
- ▶ In the title bar the name of the first notebook you open will probably be “Untitled”
- ▶ Click on it to rename it to something more informative
 - ▶ This will also rename the .ipynb file where the notebook is saved



Cell Types

- ▶ There are four types of input cells where you can write the content for your notebook:
 - ▶ **Code cells:** the default type of cell, this type of cell consists of executable code
 - ▶ **Markdown cells:** this type of cell allows for a rich form of documentation for your code
 - ▶ When executed, the input to a markdown cell is converted into HTML, which can include mathematical equations, font effects, lists, tables, embedded images and videos
 - ▶ **Raw cells:** input into this type of cell is not changed by the notebook – its content and formatting is preserved exactly

Running Cells

- ▶ Each cell can consist of more than one line of input, and the cell is not interpreted until you “run” it
- ▶ This is achieved either by selecting the appropriate option from the menu bar (under the “Cell” drop-down submenu), by clicking the “play” button on the tool bar, or through the following keyboard shortcuts:
 - ▶ **Shift-Enter**: Execute the cell, showing any output, and then *move the cursor* onto the cell below. If there is no cell below, a new, empty one will be created.
 - ▶ **CTRL-Enter**: Execute the cell in place, but *keep the cursor* in the current cell. Useful for quick “disposable” commands to check if a command works.
 - ▶ **Alt-Enter**: Execute the cell, showing any output, and then *insert and move the cursor to a new cell* immediately beneath it.

Code Cells

- ▶ All cells start as “Code” cells by default
- ▶ You can enter anything into a code cell that you can when writing a Python program in an editor or at the regular IPython shell
- ▶ Code in a given cell has access to objects defined in other cells (providing they have been run)
- ▶ For example,

```
In [ ]: x = 10
```

- ▶ Pressing Shift-Enter or clicking Run Cell executes this statement (defining x but producing no output) and opens a new cell underneath the old one:

```
In [1]: x = 10
```

```
In [ ]: |
```

Code Cells

- ▶ Enter the following statement at this new prompt:

```
In [1]: x = 10
```

```
In [ ]: print("x = ", x)
```

- ▶ and executing as before produces output and opens a third empty input cell:

```
In [1]: x = 10
```

```
In [2]: print("x =", x)
```

```
x = 10
```

```
In [ ]: |
```

- ▶ You can edit the value of x in input cell 1 and rerun the entire document to update the output by choosing Kernel -> Restart & Run all

Code Cells

- ▶ It is also possible to set a new value for x *after the* calculation in cell 2:

```
In [3]: x = 5
```

- ▶ Running cell 3 and then cell 2 then changes the output of cell 2 to:

```
In [4]: print("x =", x)
```

```
x = 5
```

- ▶ even though the cell above still defines x to be 10
- ▶ Unless you run the entire document from the beginning, the output doesn't necessarily reflect the output of a script corresponding to the code cells taken in order

Edit Mode

- ▶ There are two modes of operation in a notebook: edit mode and command mode
- ▶ The keyboard does different things depending on which mode the notebook is in
- ▶ **Edit mode** is indicated by a green cell border and a prompt showing in the editor area:

A screenshot of a Jupyter Notebook cell in edit mode. The cell has a green border. Inside, the text 'In [1]:' is in blue, followed by 'x = 10' in green, and a green cursor is positioned at the end of the line.

```
In [1]: x = 10|
```

- ▶ When a cell is in edit mode, you can type into the cell, like a normal text editor
- ▶ In edit mode the arrow keys navigate the *contents* of the cell
- ▶ Enter edit mode by pressing Enter or using the mouse to click on a cell's editor area

Command Mode

- ▶ Command mode is indicated by a grey cell border with a blue left margin:

```
In [1]: x = 10
```

- ▶ When you are in command mode, you are able to edit the notebook as a whole, but not type into individual cells
- ▶ In command mode, the keyboard is mapped to a set of shortcuts that let you perform notebook and cell actions efficiently
- ▶ For example, pressing a in command mode will create a new cell above the current cell, and arrow keys navigate *through the cells*
- ▶ Enter command mode by pressing Esc or using the mouse to click *outside* a cell's editor area

Keyboard Shortcuts

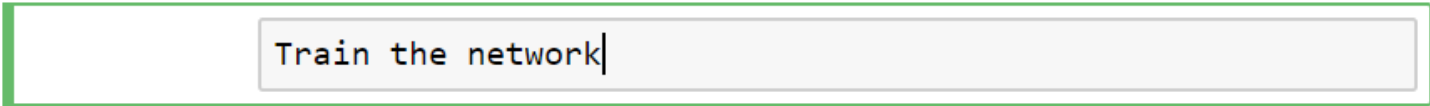
- ▶ The Help->Keyboard Shortcuts dialog lists the available shortcuts
- ▶ Useful commands in command mode:
 - ▶ Basic navigation: shift-enter, ctrl-enter, up/k, down/j
 - ▶ Saving the notebook: s
 - ▶ Change cell type: y, m, 1-6, t
 - ▶ Insert cell above: a
 - ▶ Insert cell below: b
 - ▶ Copy cell: c
 - ▶ Paste cell: v
 - ▶ Delete selected cell: dd

Exercise (2)

- ▶ Create a new notebook in C:\Notebooks named WelcomeToDS.ipynb
- ▶ Print your name and age in two separate cells
- ▶ Insert a new cell between the two existing cells
- ▶ In the new cell print the current date
- ▶ Copy the first two cells to another notebook named SecondNotebook.ipynb
- ▶ Add a third cell that prints your favorite color
- ▶ Delete the second cell
- ▶ Run the second notebook from start to end
- ▶ Delete the second notebook

Markdown Cells

- ▶ Markdown cells are useful to help document your Notebook or to write detailed “readme” descriptions within your notebook
- ▶ You can change the cell type from the menu Cell > Cell Type or by using the key shortcut m (in command mode)



```
Train the network|
```

- ▶ When you “run” the markdown cell, the text in the cell is converted into HTML:

Train the network



```
In [ ]: |
```

- ▶ Markdown is a popular markup language that is a superset of HTML
 - ▶ Its specification can be found here: <https://daringfireball.net/projects/markdown/>

Markdown Basics

- ▶ You can make text italic or bold by using asterisks or underscores:

```
Surrounding text by two asterisks denotes bold style;  
using one asterisk denotes italic text, as does a single  
underscore.
```

Surrounding text by two asterisks denotes **bold style**; using one asterisk denotes *italic text*, as does a single underscore.

- ▶ Inline code examples are created by surrounding the text with backticks (`):

```
Here are some Python keywords: `for`, `while`, and `lambda`.
```

Here are some Python keywords: `for`, `while`, and `lambda`.

- ▶ New paragraphs are started after a blank line

Headings

- ▶ You can add headings by starting a line with one (or multiple) # followed by a space, as in the following example:

```
# Heading 1  
## Heading 1.1  
### Heading 1.1.1
```

Heading 1

Heading 1.1

Heading 1.1.1

- ▶ Six levels of heading are supported

Links

- ▶ Links are created by providing a URL in round brackets after the text to be turned into a link in square brackets

```
Here is a link to the [Python website](http://python.org)
```

Here is a link to the [Python website](http://python.org)

- ▶ If the link is to a file on your local system, give as the URL the path, relative to the notebook directory, prefixed with files/:

```
Here is [the housing data file](files/datasets/housing.csv).
```

Here is [the housing data file](files/datasets/housing.csv).

Lists

- ▶ Itemized lists are created using any of the markers *, + or -, and nested sublists are simply indented:

The inner planets and their satellites:

```
* Mercury
* Venus
* Earth
  * The Moon
* Mars
  * Phobus
  * Deimos
```

The inner planets and their satellites:

- Mercury
- Venus
- Earth
 - The Moon
- Mars
 - Phobus
 - Deimos

- ▶ Ordered (numbered) lists are created by preceding items by a number followed by a full stop (period) and a space:

```
1. Symphony No. 1 in C major, Op. 21
2. Symphony No. 2 in D major, Op. 36
3. Symphony No. 3 in E-flat major ("Eroica"), Op. 55
```

1. Symphony No. 1 in C major, Op. 21
2. Symphony No. 2 in D major, Op. 36
3. Symphony No. 3 in E-flat major ("Eroica"), Op. 55

Mathematical Equations

- ▶ Mathematical equations can be written in LaTeX
- ▶ Inline expressions can be added by surrounding the latex code with \$:

```
Defining the function  $f(x) = \sin(x^2)$ 
```

Defining the function $f(x) = \sin(x^2)$

- ▶ Expressions on their own line are surrounded by \$\$:

```

$$\int_0^\infty e^{-x^2} dx = \frac{\sqrt{\pi}}{2}$$

```

$$\int_0^\infty e^{-x^2} dx = \frac{\sqrt{\pi}}{2}$$

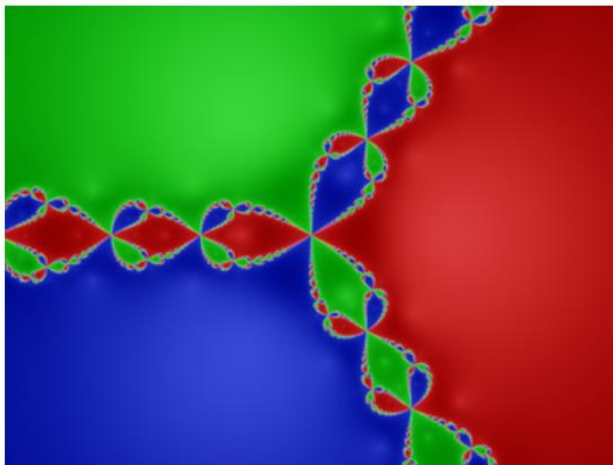
- ▶ Learn LaTeX in 30 minutes:

[https://www.sharelatex.com/learn/Learn LaTeX in 30 minutes](https://www.sharelatex.com/learn/Learn_LaTeX_in_30_minutes)

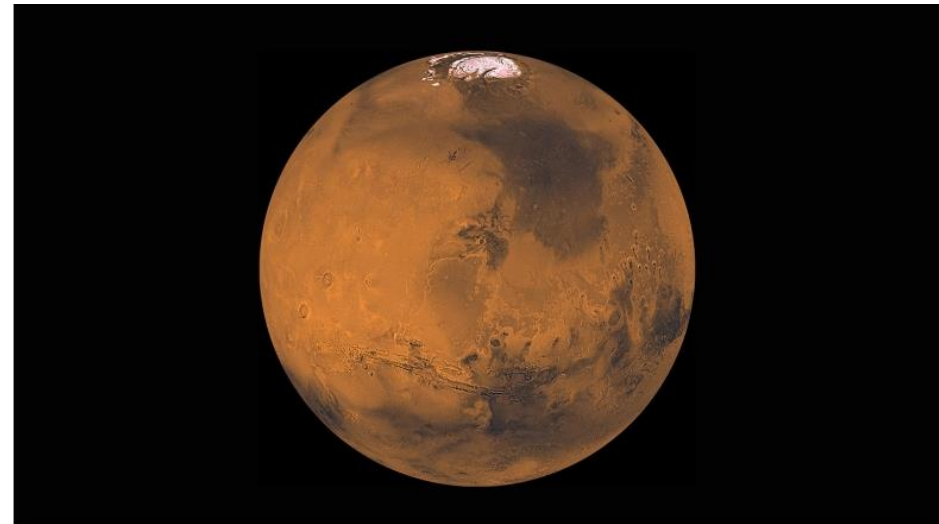
Images

- ▶ Links to image files work in exactly the same way as ordinary links, but are preceded by an exclamation mark, !
- ▶ The text in square brackets between the exclamation mark and the link acts as *alt text* to the image

```
![An interesting plot of the Newton fractal]  
(/files/images/newton_fractal.png)
```



```
![Planet Mars]  
(https://boygeniusreport.files.wordpress.com/2016/05/mars.jpg?  
quality=98&strip=all&w=782)
```



General HTML

- ▶ The markdown used by IPython notebooks encompasses HTML, so valid HTML entities and tags can be used directly, including CSS styles

The following `Punnett table` is `marked up` in HTML.

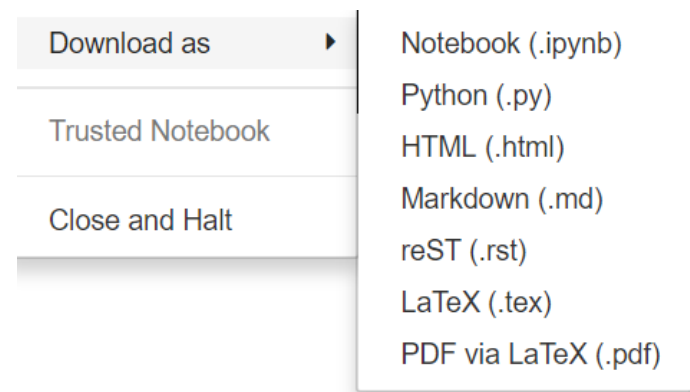
```
<table style="text-align: center;">
<tr>
<th rowspan="2" colspan="2"></th>
<th colspan="2">Male</th>
</tr>
<tr>
<th>A</th>
<th>a</th>
</tr>
<tr>
<th rowspan="2">Female</th>
<th>a</th>
<td style="background: #aaa;">Aa</td>
<td>aa</td>
</tr>
<tr>
<th>a</th>
<td style="background: #aaa;">Aa</td>
<td>aa</td>
</tr>
</table>
```

The following *Punnett table* is marked up in HTML.

		Male	
		A	a
Female	a	Aa	aa
	a	Aa	aa

Converting Notebooks to Other Formats

- ▶ nbconvert is a tool, installed with Jupyter notebook, to convert notebooks from their native .ipynb format to any of several alternative format
- ▶ You can invoke this tool from the File > Download as menu



- ▶ Or run it from the command line as:

```
ipython nbconvert --to <format> <notebook.ipynb>
```

- ▶ where notebook.ipynb is the name of the notebook file to be converted and format is the desired output format
- ▶ The default (if no format is given), is to produce a static HTML file

Converting to Python

- ▶ Choose File > Download as > Python (.py)
- ▶ This will save the notebook as *MyFirstNotebook.py* in your Download folder
- ▶ If any of the notebook's code cells contain IPython magic functions, this script may only be executable from within an IPython session
- ▶ Markdown and other text cells are converted to comments in the generated Python script code
- ▶ You can now run this script from the Python shell

Converting to Python

```
# coding: utf-8

# In[1]:

x = 10

# In[2]:

print("x =", x)

# In[3]:

x = 5

# Train the network

# Surrounding text by two asterisks denotes bold style;
# using one asterisk denotes italic text, as does a single
# underscore_.
```

```
C:\Users\roi>cd c:\Python
c:\Python>python MyFirstNotebook.py
x = 10
c:\Python>
```

Exercise (3)

- ▶ Create the following notebook using markdown cells:

Fruit

- Pears
- Apples
 - Granny Smith
 - Gala
- Oranges
- Berries
 - Strawberries
 - Blueberries
 - Raspberries
- Bananas



$$\sum_{i=1}^n = \frac{n(n+1)}{2}$$

Help Commands

- ▶ There are various helpful commands to obtain information:
 - ▶ Typing a single '?' outputs an overview of the usage of IPython's main features
 - ▶ %quickref provides a brief reference summary of each of the main IPython commands and "magics"
 - ▶ help() or help(object) invokes Python's own help system (interactively or for object if specified)
 - ▶ Typing one question mark after an object name provides information about that object
- ▶ Example for introspecting an object:

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

```
In [5]: a?
```

```
Type:      list
String form: [5, 6]
Length:    2
Docstring:
list() -> new empty list
list(iterable) -> new list initialized from iterable's items
```

Help Commands

- ▶ The ? syntax is particularly useful as a reminder of the arguments that a function or method takes

```
In [5]: print?
```

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

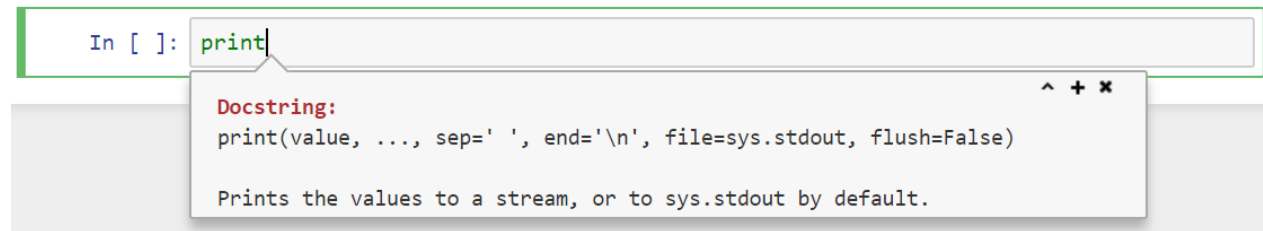
Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file: a file-like object (stream); defaults to the current sys.stdout.
sep: string inserted between values, default a space.
end: string appended after the last value, default a newline.
flush: whether to forcibly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

Type: builtin_function_or_method

- ▶ For some objects, the syntax object?? returns more advanced information, such as the location and details of its source code.

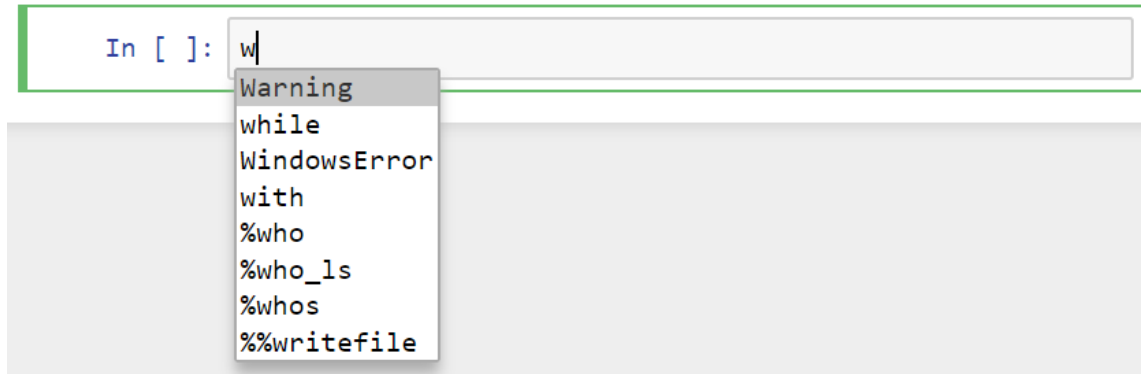
Help Commands

- ▶ Pressing Shift-Tab shows a tooltip containing additional information about the object or function



Tab Completion

- ▶ Just as with many command line shells, IPython supports tab completion: start typing the name of an object or keyword, press the <TAB> key, and it will autocomplete it for you or provide a list of options if more than one possibility exists
- ▶ For example, type the letter w and then press TAB:



- ▶ If you resume typing until the word becomes unambiguous (e.g., add the letters hi) and then press <TAB> again: it will be autocomplete to while.

History

- ▶ IPython stores both the commands you enter and the output they produce in the special variables **In** and **Out**
 - ▶ These are, in fact, a list and a dictionary respectively, and correspond to the prompts at the beginning of each input and output

```
In [1]: name = "John"  
        print(name)
```

John

```
In [2]: age = 25  
        print(age)
```

25

```
In [3]: In[1]
```

```
Out[3]: 'name = "John"\nprint(name)'
```

← In[1] simply holds the string version of the Python statements that were entered at index 1

```
In [4]: exec(In[1])
```

John

← To actually execute the statements, we must send the cell to Python's `exec()` built-in

History

- ▶ The Out[] dictionary is useful if you want to interact with past results
- ▶ For example:

```
In [5]: import math
```

```
In [6]: math.sin(2)
```

```
Out[6]: 0.9092974268256817
```

```
In [7]: math.cos(2)
```

```
Out[7]: -0.4161468365471424
```

```
In [8]: Out[6] ** 2 + Out[7] ** 2
```

```
Out[8]: 1.0
```

- ▶ Note that not all operations have outputs: for example, import statements and print statements don't affect the output

History

- ▶ There are a few more shortcuts:
 - ▶ `_iN` is the same as `In[N]`
 - ▶ `_N` is the same as `Out[N]`
 - ▶ The two most recent outputs are returned by the variables `_` and `__` respectively (skipping any commands with no output)

```
In [8]: Out[6] ** 2 + Out[7] ** 2
```

```
Out[8]: 1.0
```

```
In [9]: _
```

```
Out[9]: 1.0
```

Suppressing the Output

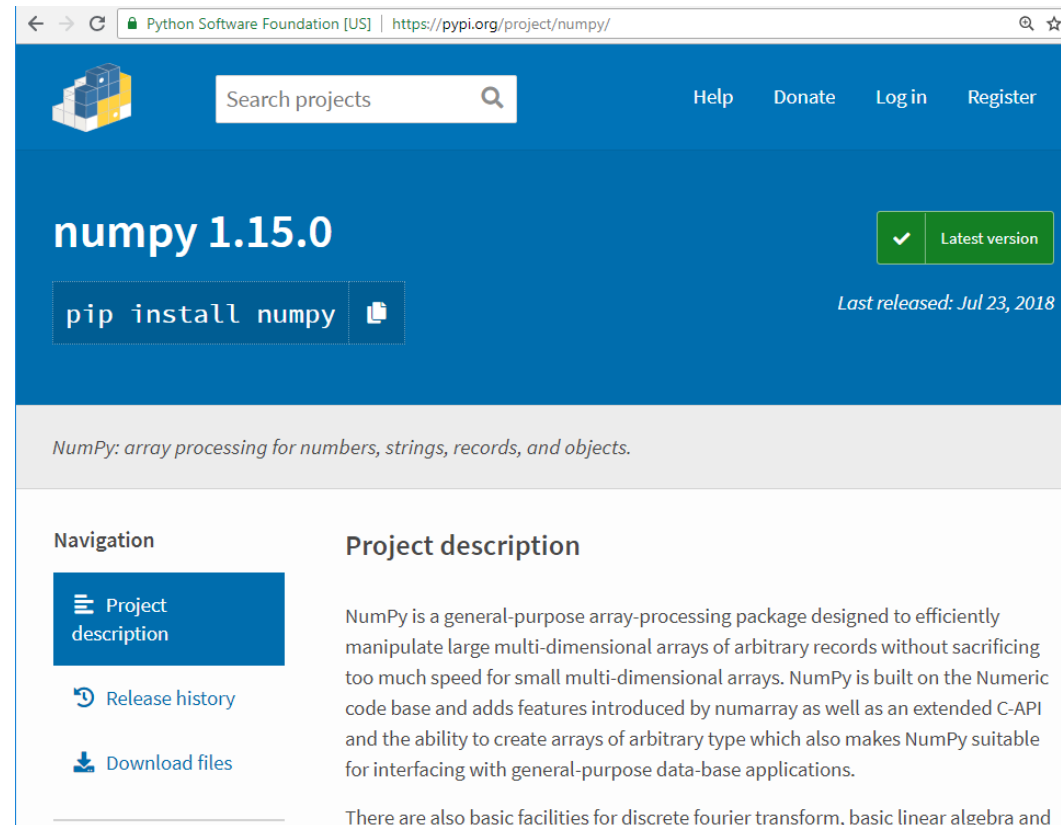
- ▶ Sometimes you might wish to suppress the output of a statement (this is perhaps most common with plotting commands)
- ▶ The easiest way to suppress the output of a command is to add a semicolon to the end of the line:

```
math.sin(2) + math.cos(2);
```

- ▶ Note that the result is computed silently, and the output is neither displayed on the screen or stored in the Out dictionary

PyPI

- ▶ The Python Package Index (PyPI) is a repository of software for the Python programming language
- ▶ <https://pypi.org/>



pip

- ▶ pip is a Python package management system used to install and manage software packages written in Python
- ▶ pip supports installing packages from PyPI, version control, local projects, and directly from distribution files
- ▶ To install the latest version of a package, type from the command-line:

```
pip install some-package
```

- ▶ To install a specific version:

```
pip install some-package==1.4
```

- ▶ To install a version that's "compatible" with a certain version:

```
pip install some-package~=1.4.2
```

- ▶ This means to install any version ">=1.4.*" version that's also ">=1.4.2".

pip

- ▶ To upgrade an already installed SomeProject to the latest from PyPI:

```
pip install --upgrade some-package
```

or:

```
pip install -u some-package
```

- ▶ This command will additionally upgrade all the package requirements to the latest versions available

- ▶ To uninstall a package:

```
pip uninstall some-package
```

- ▶ To list all the packages installed:

```
pip list
```

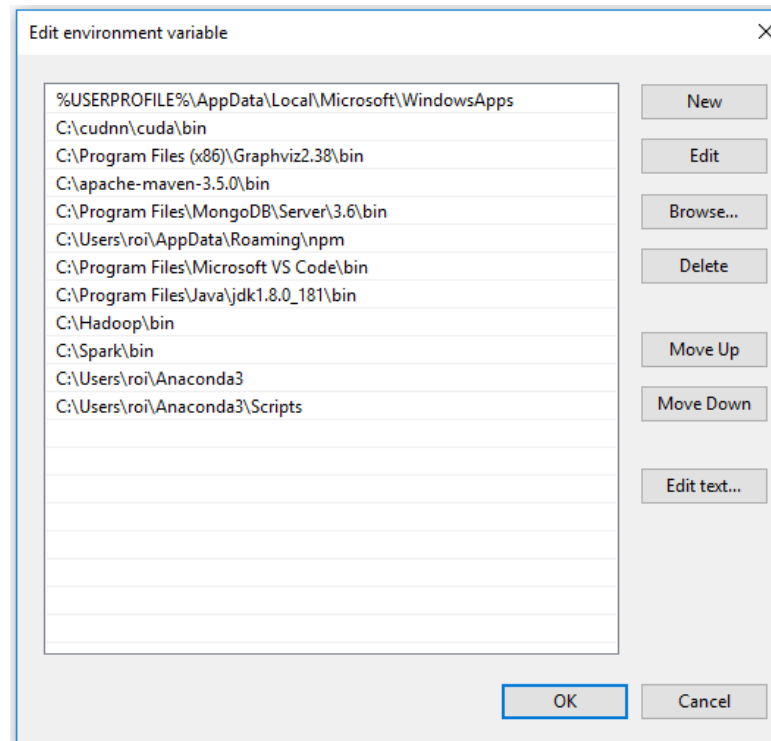
- ▶ To get information on an installed package (its version, dependencies, etc.):

```
pip show some-package
```

```
(base) C:\Users\roi>pip show numpy
Name: numpy
Version: 1.14.0
Summary: NumPy: array processing for numbers, strings, records, and objects.
Home-page: http://www.numpy.org
Author: NumPy Developers
Author-email: numpy-discussion@python.org
License: BSD
Location: c:\users\roi\anaconda3\lib\site-packages
Requires:
Required-by: tensorflow-gpu, tensorboard, tables, seaborn, PyWavelets, patsy,
pandas, odo, numexpr, numba, matplotlib, h5py, datashape, Bottleneck, bokeh, b
kcharts, astropy
```

pip

- ▶ pip.exe is typically located in the folder Scripts under your Anaconda installation folder
- ▶ To run pip from anywhere on your computer, you can add its path to the PATH environment variable:



PyCharm

- ▶ PyCharm is an IDE for the Python language developed by JetBrains, a team responsible for one of the most famous Java IDE, the IntelliJ IDEA
- ▶ PyCharm is cross-platform, with Windows, macOS and Linux versions
- ▶ PyCharm integrates with IPython Notebook, has an interactive Python console, and supports Anaconda as well as multiple scientific packages including matplotlib and NumPy
- ▶ It provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems like git, and supports web development
- ▶ It provides a free Community Edition



Installing PyCharm

- ▶ Go to <https://www.jetbrains.com/pycharm/download>
- ▶ Choose your operating system
- ▶ Download the Community edition



Version: 2018.2

Build: 182.3684.100

Released: July 25, 2018

[System requirements](#)

[Installation Instructions](#)

[Previous versions](#)

Download PyCharm

Windows

macOS

Linux

Professional

Full-featured IDE
for Python & Web
development

DOWNLOAD

Free trial

Community

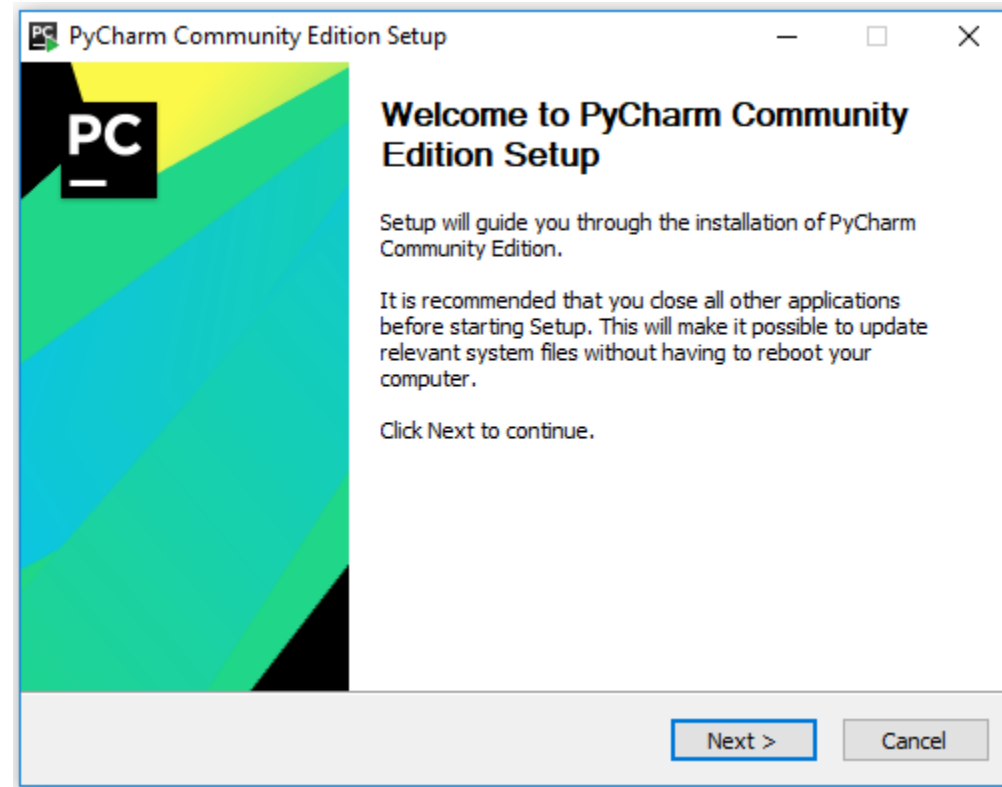
Lightweight IDE
for Python & Scientific
development

DOWNLOAD

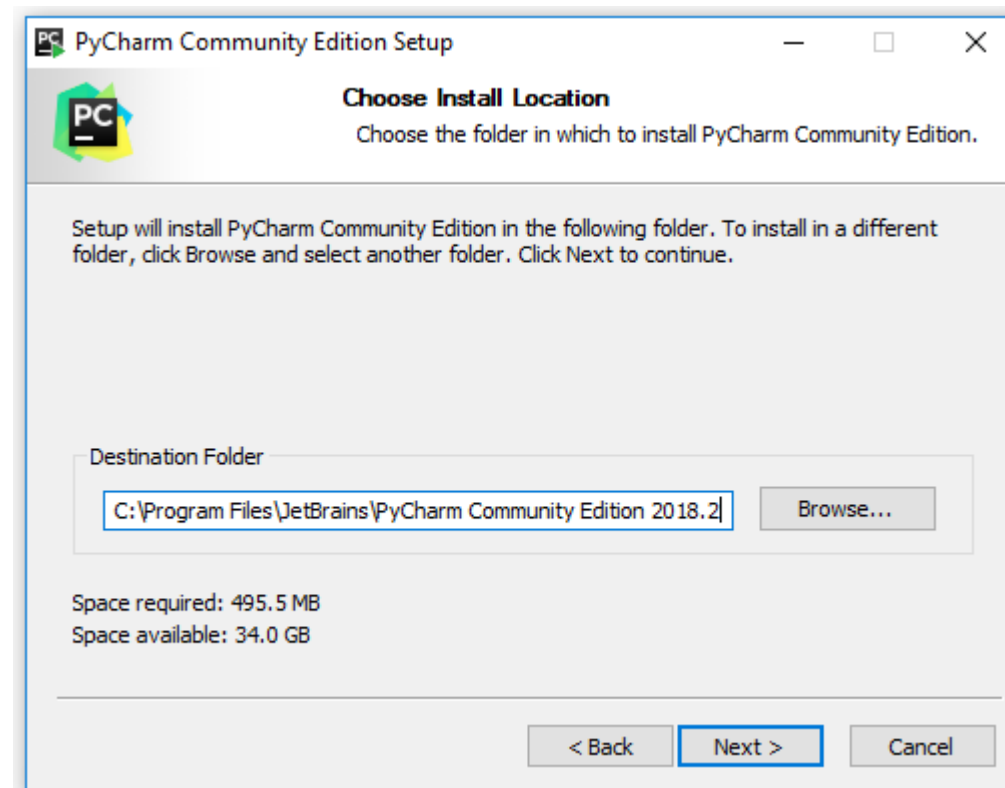
Free, open-source

Installing PyCharm

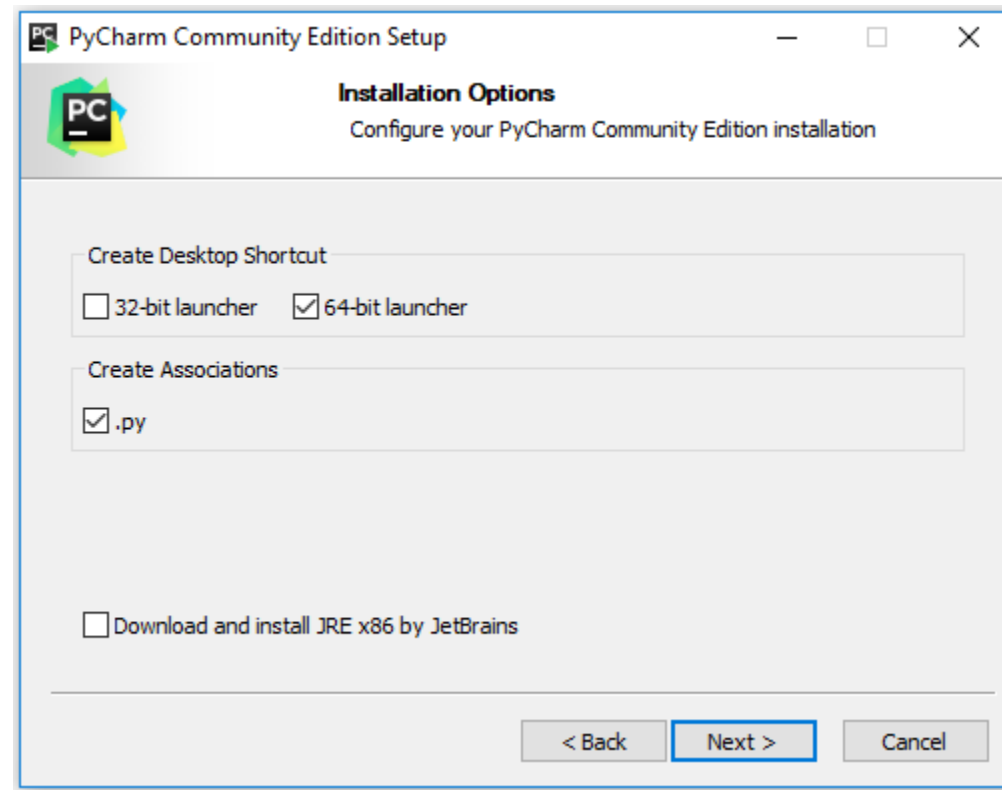
- ▶ Click Next to install



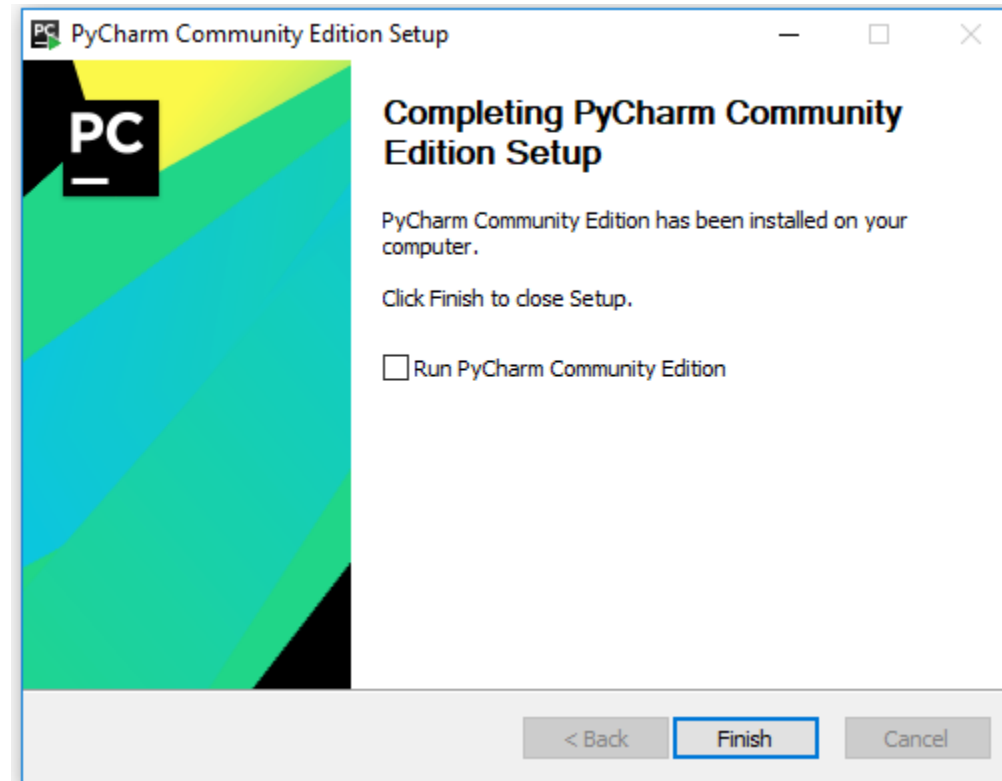
Installing PyCharm



Installing PyCharm

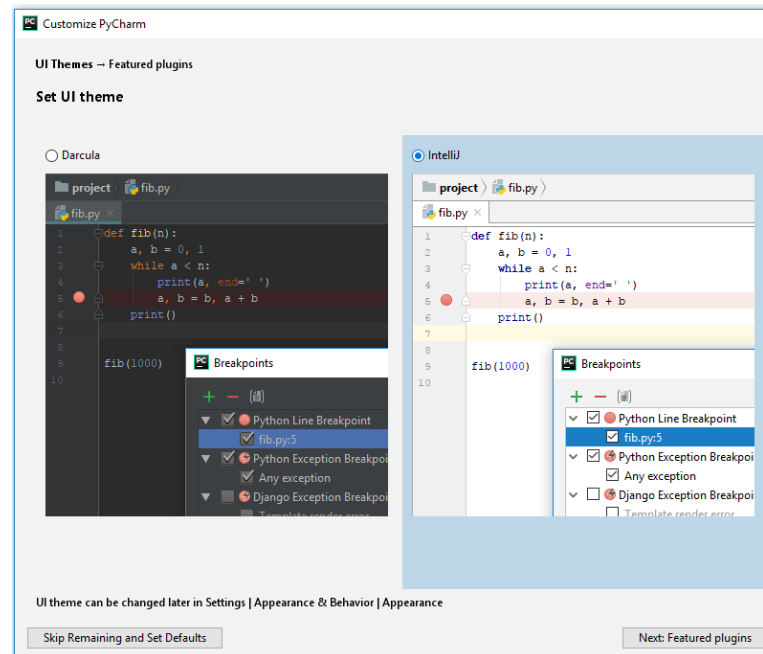


Installing PyCharm



Installing PyCharm

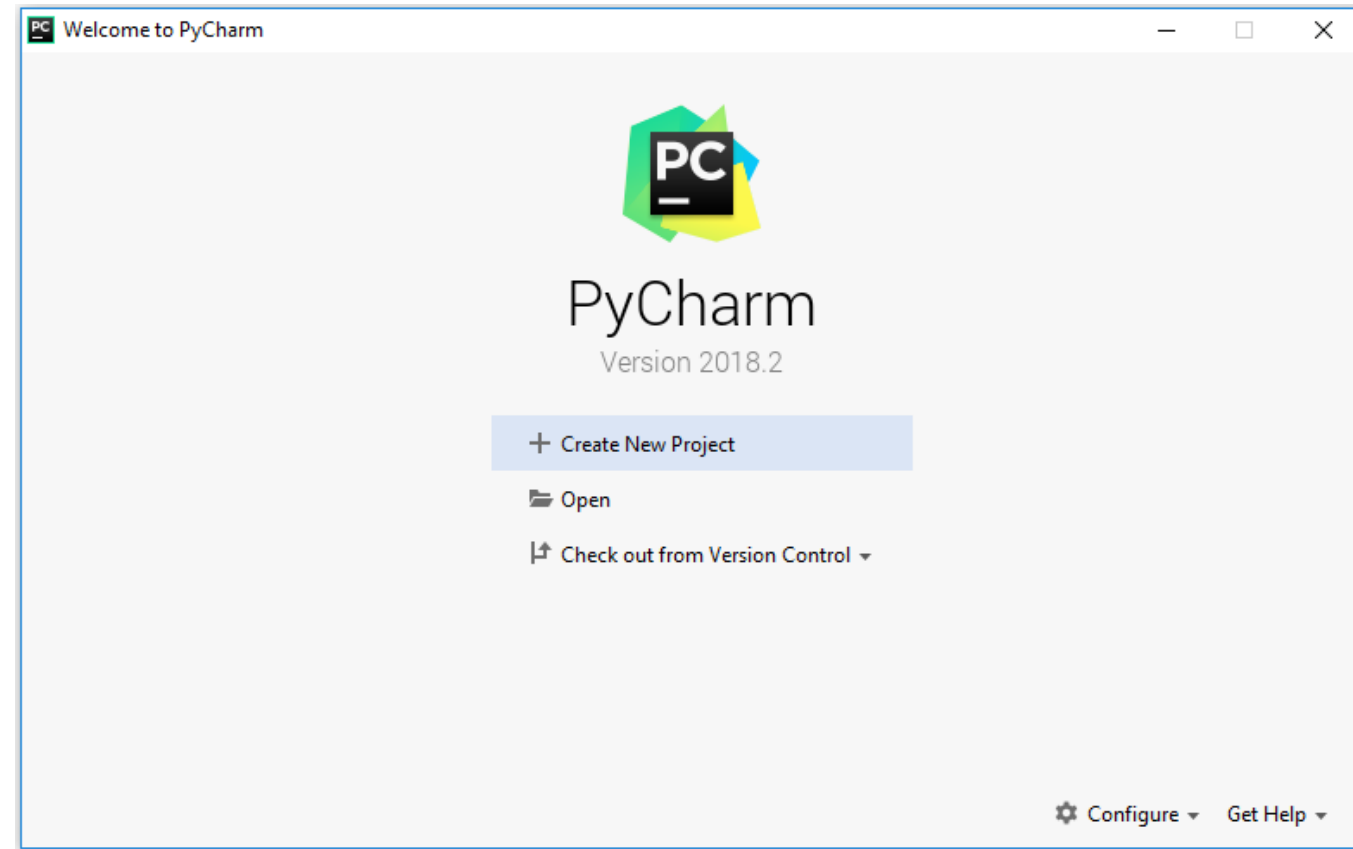
- ▶ Start PyCharm
- ▶ Choose your favorite UI theme



- ▶ Click Skip Remaining and Set Defaults

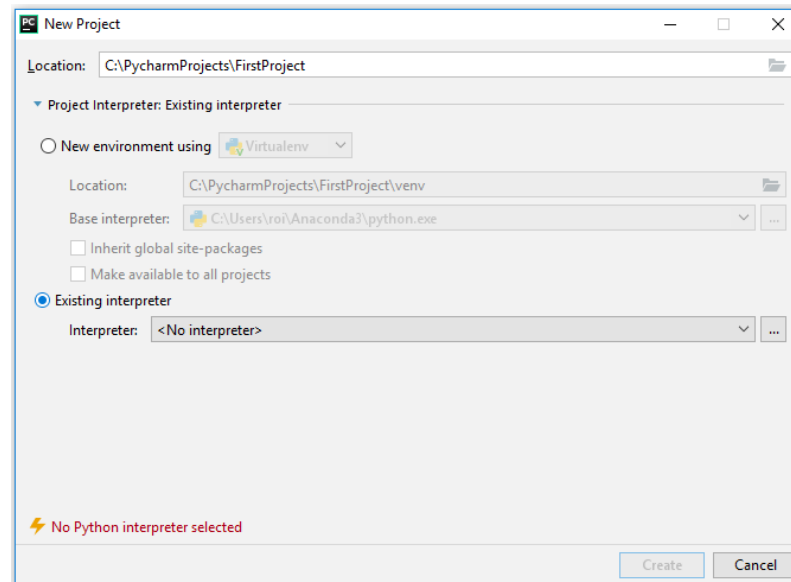
Creating New Project

- ▶ Click Create New Project



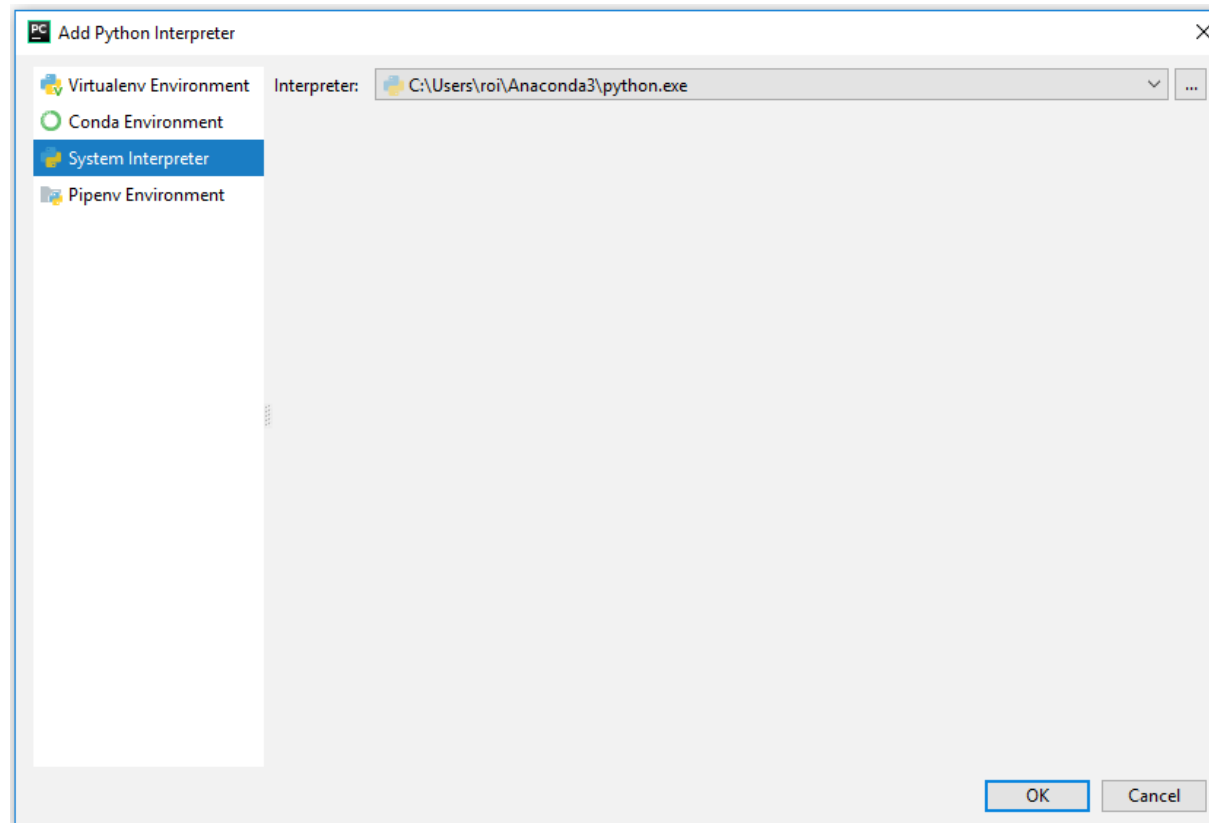
Creating New Project

- ▶ Choose a folder for your project
- ▶ Choose Existing interpreter
 - ▶ **virtualenv** is a tool to create isolated Python environments, which enables multiple side-by-side installations of Python, one for each project
- ▶ Double the ellipsis button next to the interpreter list box



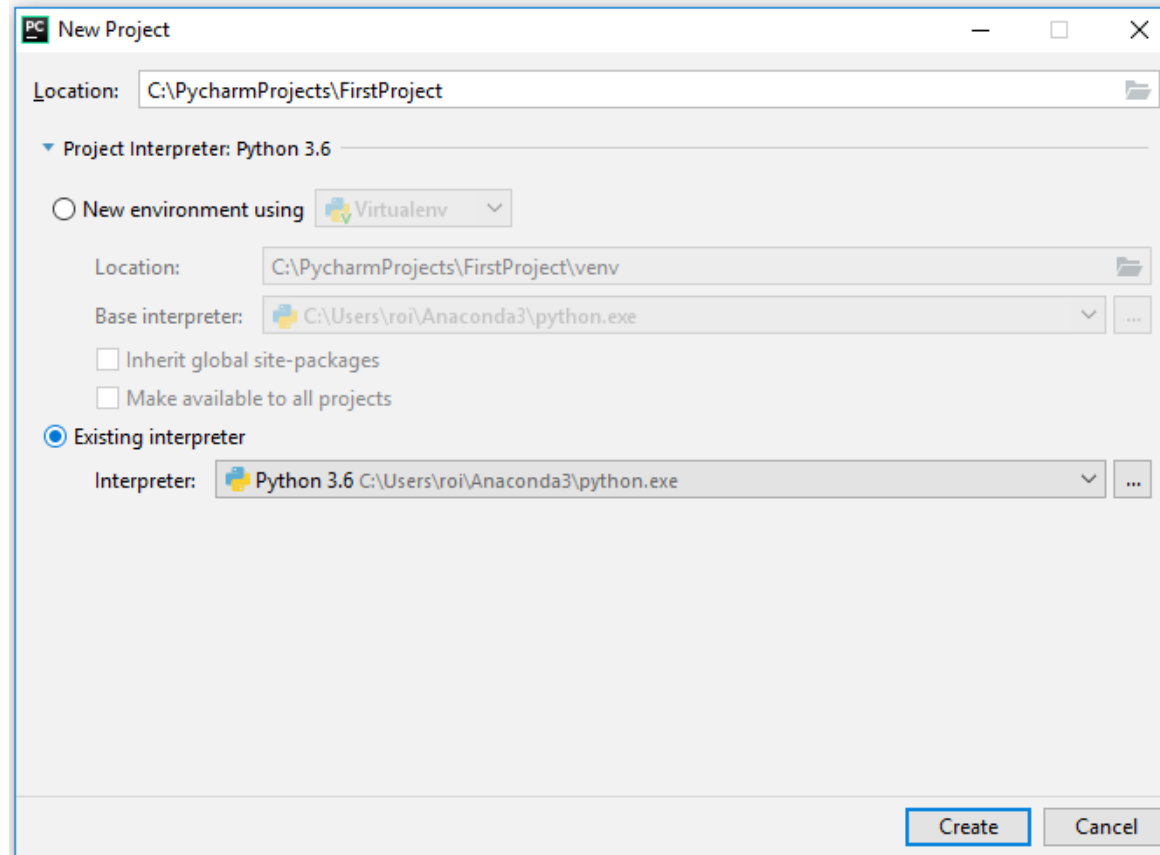
Creating New Project

- ▶ Go to System interpreter
- ▶ Choose the Anaconda python interpreter

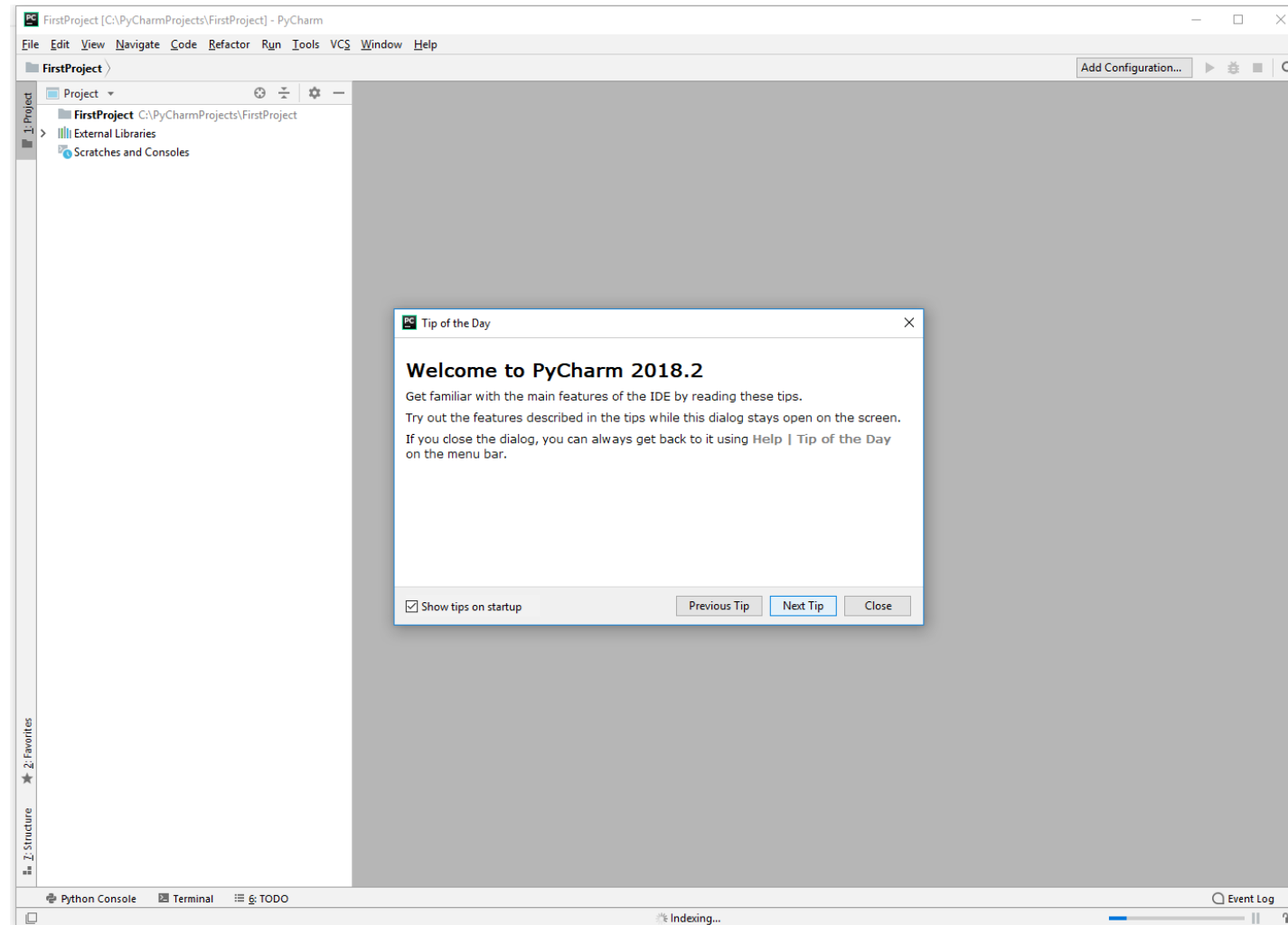


Creating New Project

► Click Create

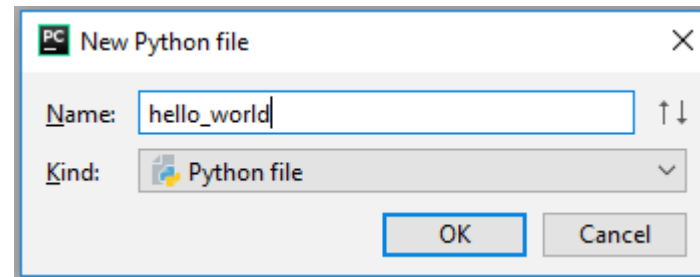


Creating New Project

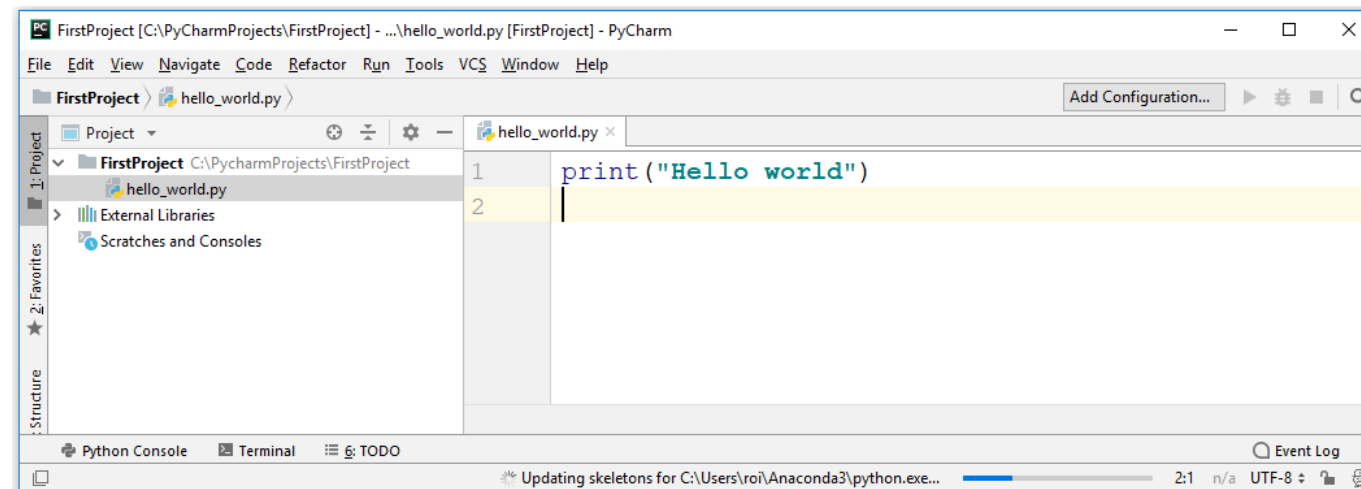


Adding New Python File

- ▶ To add a new Python file right-click the project name and choose New -> Python file

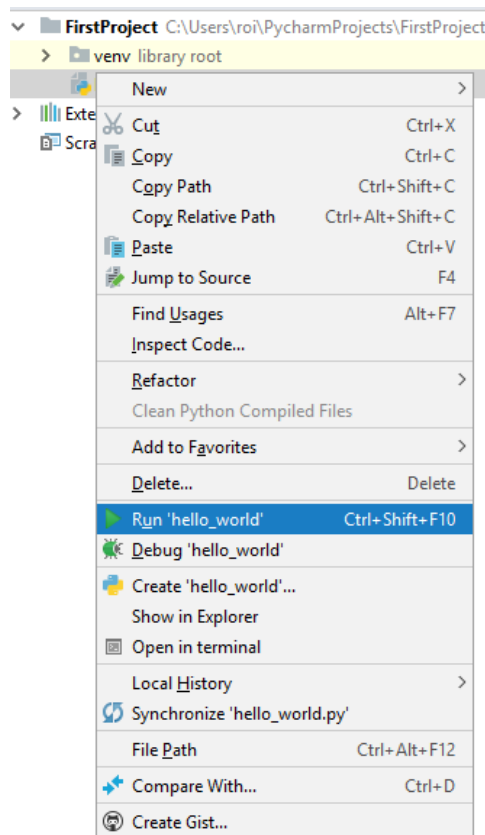


- ▶ Double-click hello_world.py and enter the following code:



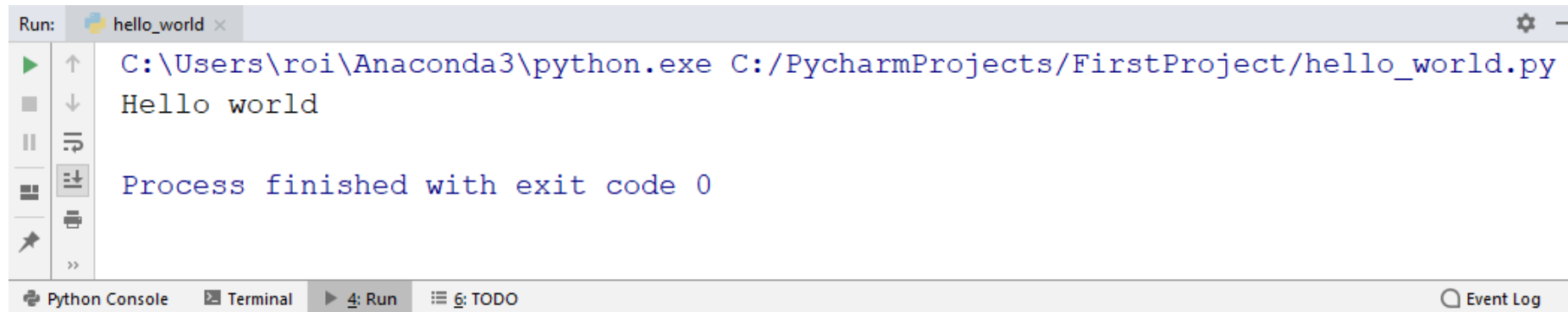
Running the Script

- ▶ Right-click the python file and choose Run



Running the Script

- ▶ The output of your script appears at the bottom of the screen:



The screenshot shows the 'Run' console in PyCharm. The top bar indicates the file 'hello_world.py' is being executed. The console output shows the command 'C:\Users\roi\Anaconda3\python.exe C:/PycharmProjects/FirstProject/hello_world.py', the output 'Hello world', and the status 'Process finished with exit code 0'. On the left, there is a vertical toolbar with icons for running, debugging, and other actions. At the bottom, there are tabs for 'Python Console', 'Terminal', 'Run', and 'TODO', along with an 'Event Log' button.

```
Run: hello_world x
C:\Users\roi\Anaconda3\python.exe C:/PycharmProjects/FirstProject/hello_world.py
Hello world
Process finished with exit code 0
```

- ▶ If you need to run the script again, you can click the Play button on the left pane

Exercise (4)

- ▶ Create a new Pycharm project in C:\PycharmProjects named WelcomeToDS
- ▶ Add a script my_quote.py to this project
- ▶ In the script print your favorite quote to the console
- ▶ Run the script