

---

# Getting Started with Python



# Contents

1. Setting the scene
2. Running Python script code

## Annex

- Creating a virtual environment



Demo folder: 01-GettingStarted

# 1. Setting the Scene

- Hello Python
- What can you do with Python?
- Downloading Python
- Using Python documentation
- Installing Python packages

# Hello Python

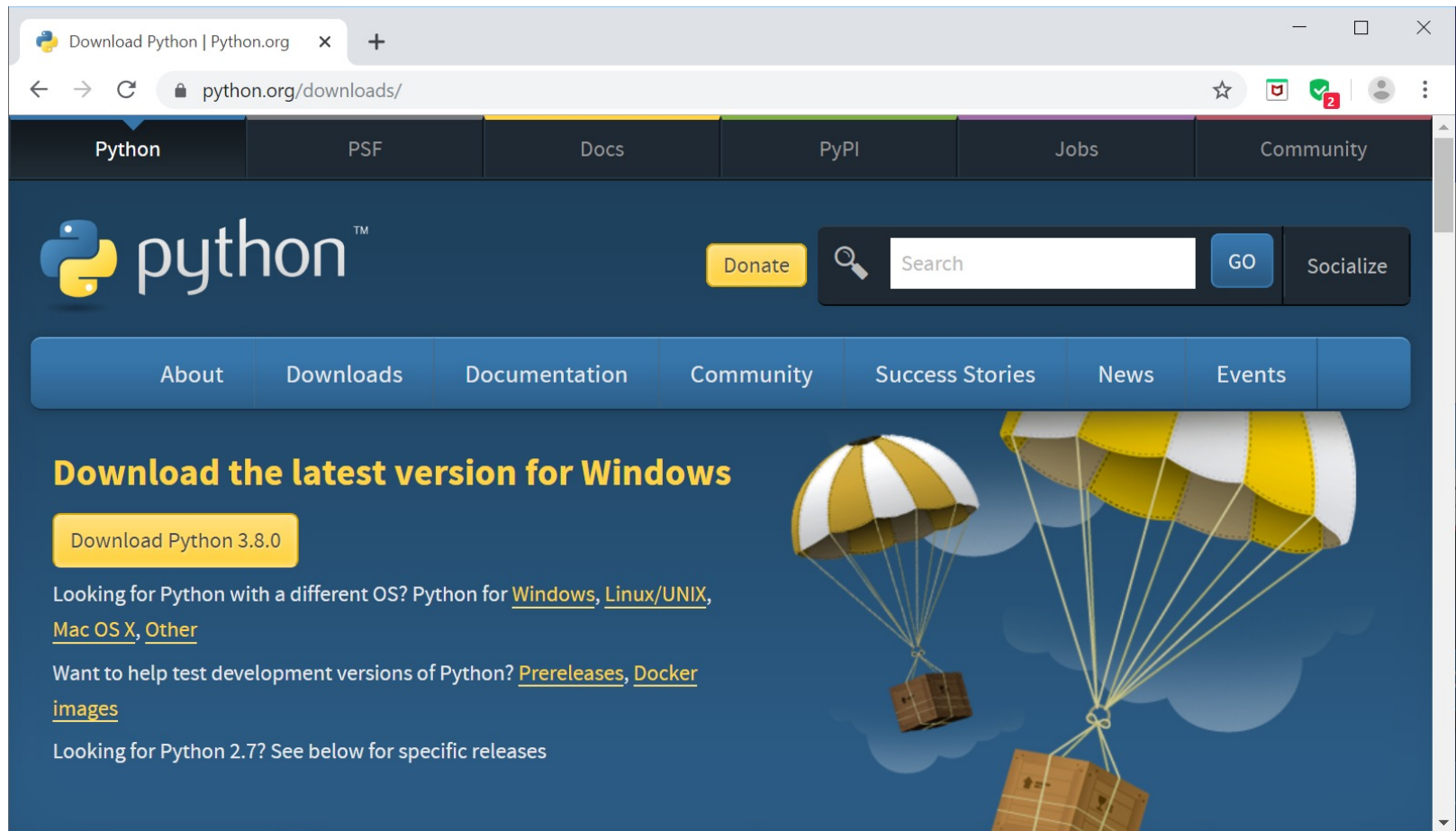
- Python is powerful, expressive programming language
  - Object-oriented
  - Dynamic typing
  - Interpreted
- Available on a wide range of platforms
  - Unix/Linux
  - Windows
  - Mac OS X
  - etc.

# What can you do with Python?

- Scripting
- File I/O
- String handling and regular expressions
- Web applications and REST web services
- Data science

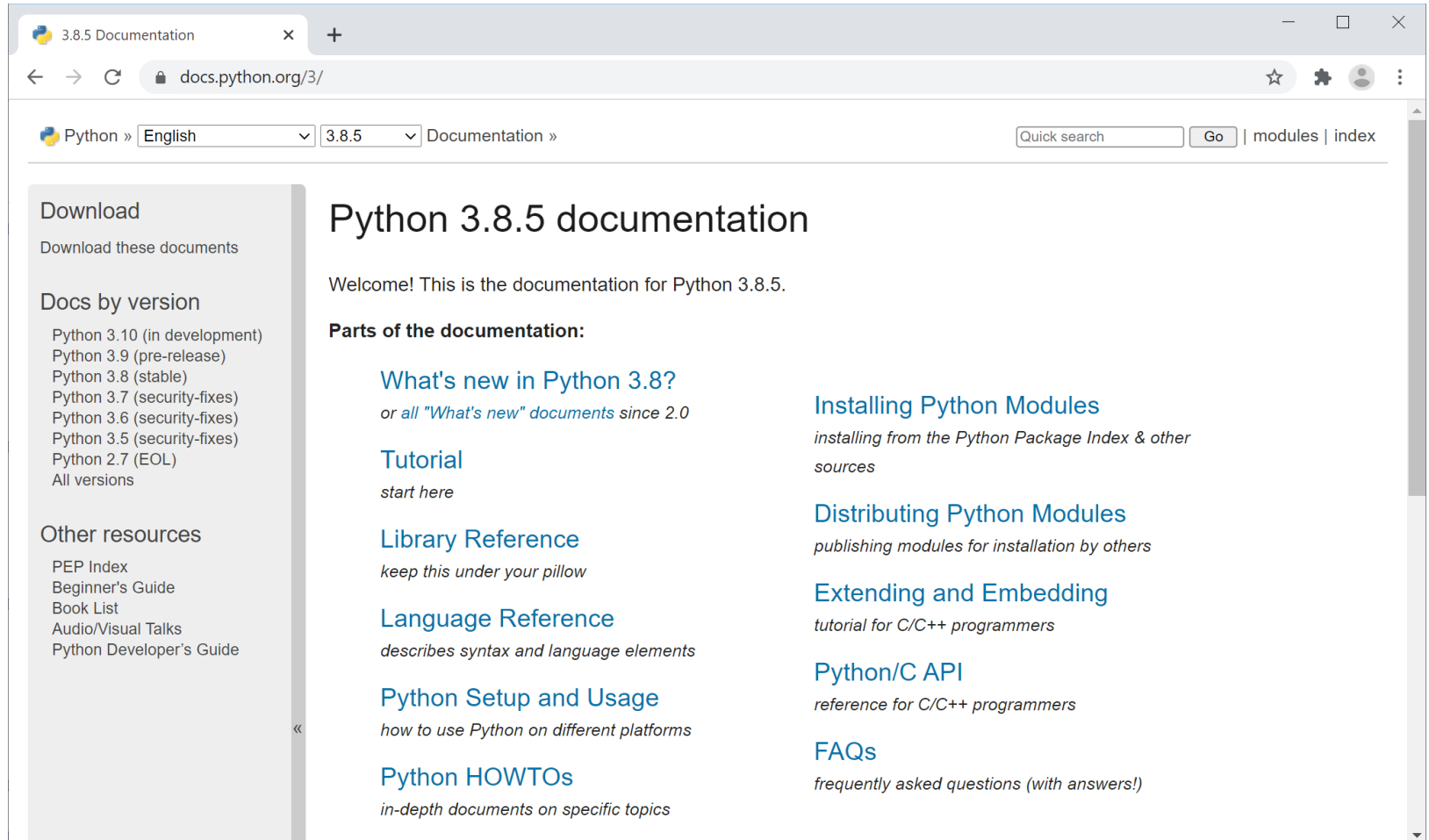
# Downloading Python

- You can download Python for free
  - <https://www.python.org/downloads/>



# Using Python Documentation

- Docs available online at <https://docs.python.org>



The screenshot shows a web browser displaying the Python 3.8.5 documentation page. The browser's address bar shows `docs.python.org/3/`. The page has a navigation bar with "Python »", a language dropdown set to "English", a version dropdown set to "3.8.5", and "Documentation »". There is also a "Quick search" input field and links for "Go", "modules", and "index".

The main content area is titled "Python 3.8.5 documentation" and includes a welcome message: "Welcome! This is the documentation for Python 3.8.5." Below this, a section titled "Parts of the documentation:" lists several links with descriptions:

- [What's new in Python 3.8?](#)  
*or all "What's new" documents since 2.0*
- [Tutorial](#)  
*start here*
- [Library Reference](#)  
*keep this under your pillow*
- [Language Reference](#)  
*describes syntax and language elements*
- [Python Setup and Usage](#)  
*how to use Python on different platforms*
- [Python HOWTOs](#)  
*in-depth documents on specific topics*
- [Installing Python Modules](#)  
*installing from the Python Package Index & other sources*
- [Distributing Python Modules](#)  
*publishing modules for installation by others*
- [Extending and Embedding](#)  
*tutorial for C/C++ programmers*
- [Python/C API](#)  
*reference for C/C++ programmers*
- [FAQs](#)  
*frequently asked questions (with answers!)*

A left sidebar contains additional navigation options:

- Download**  
Download these documents
- Docs by version**
  - Python 3.10 (in development)
  - Python 3.9 (pre-release)
  - Python 3.8 (stable)
  - Python 3.7 (security-fixes)
  - Python 3.6 (security-fixes)
  - Python 3.5 (security-fixes)
  - Python 2.7 (EOL)
  - All versions
- Other resources**
  - PEP Index
  - Beginner's Guide
  - Book List
  - Audio/Visual Talks
  - Python Developer's Guide

# Installing Python Packages

- There are many Python packages available
  - E.g. NumPy, Matplotlib, etc.
  - See <https://pypi.python.org/pypi> for details
- You can use the `pip` package manager to install Python packages
  - For example, to install the NumPy package:

```
pip install numpy
```

- To find where `pip` installed a package:

```
pip show numpy
```

- Note: These are already installed in Anaconda



## 2. Running Python Script

- Running Python script interactively
- Creating variables
- Line continuation
- Blocks
- Creating and running Python modules
- Python keywords

# Running Python Script Interactively (1 of 2)

- First, ensure Python is on the path

```
set path=C:\python38-32;%path%
```

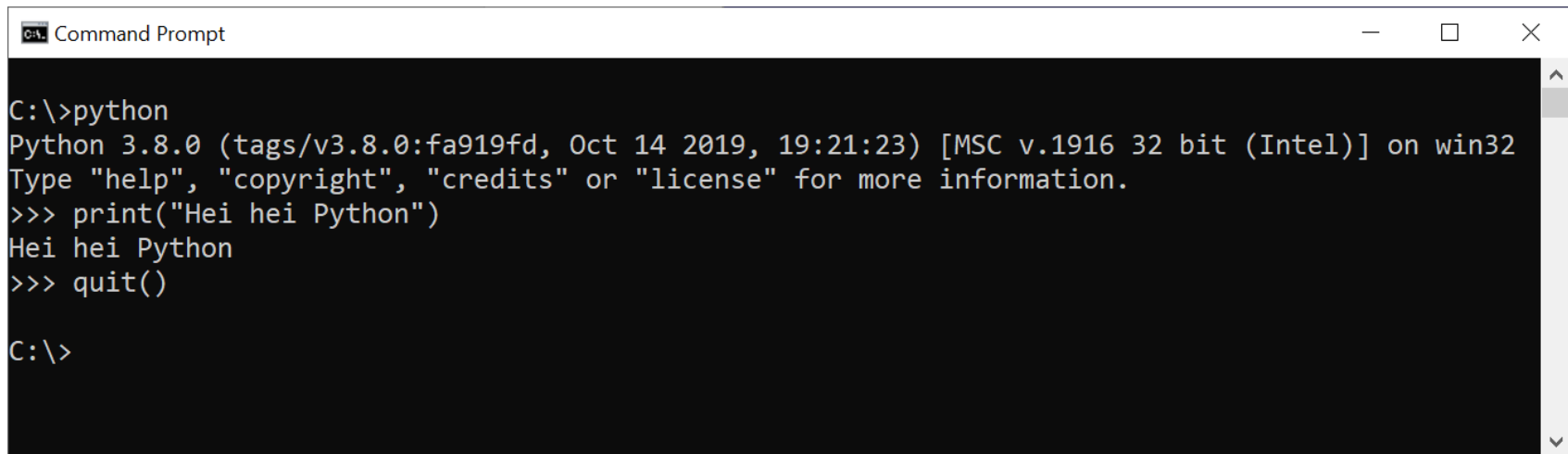
- Then run the Python interpreter in interactive mode, and execute Python code

```
python
```

```
print("Hello Python!!!")
```

# Running Python Script Interactively (2 of 2)

## ■ Example



```
Command Prompt
C:\>python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hei hei Python")
Hei hei Python
>>> quit()

C:\>
```

# Creating Variables

- In Python, you don't need to declare a variable
  - Just assign it a value, and Python will create it for you dynamically
- Rules for identifiers in Python
  - Can contain uppercase or lowercase letters, digits, and underscore
  - But can't start with a digit

```
firstname = "Homer"  
lastname = "Simpson"  
fullname = firstname + " " + lastname  
print(fullname)
```

# Line Continuation

- If a statement spans multiple lines...
  - You can use \ to continue from one line to the next

```
firstname = "Homer"  
lastname = "Simpson"  
fullname = firstname + \  
" " + \  
lastname  
print(fullname)
```

# Blocks

- Python uses indentation to denote blocks
  - Don't use {}
  - Use : to indicate the start of an indented block

```
age = 21
if age >= 18 and age <= 30:
    print("You are eligible for an 18-30s holiday!")
print("That's all folks")
```

# Creating and Running Python Modules

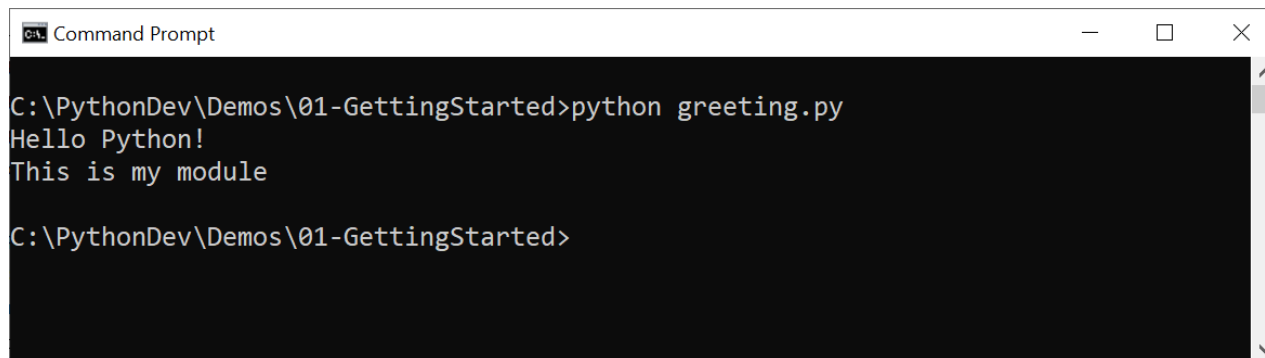
- You can put Python code into modules
  - A module is just a script file containing Python code
  - Typically starts with a lowercase letter, and ends in .py

```
print("Hello Python!")  
print("This is my module")
```

greeting.py

- You can run the module via the Python interpreter

```
python greeting.py
```



A screenshot of a Windows Command Prompt window. The title bar reads "C:\ Command Prompt". The command prompt shows the following text:

```
C:\PythonDev\Demos\01-GettingStarted>python greeting.py  
Hello Python!  
This is my module  
  
C:\PythonDev\Demos\01-GettingStarted>
```

# Python Keywords

- Here is a full list of all the keywords in Python 3.8
  - False, None, True
  - if, elif, else, assert, is
  - and, or, not
  - for, in, from, while, break, continue, pass
  - def, return, global, nonlocal, lambda
  - import, from
  - class, del
  - raise, try, except, as, finally
  - with, as
  - yield
  - await, async
- You can ask Python to tell you about all its keywords:

```
import keyword  
keyword.kwlist
```



# Any Questions?



# Annex: Creating a Virtual Environment

- Overview
- Installing virtualenv
- Creating a virtual environment for a project
- Activating a virtual environment
- Using virtualenv
- Deactivating a virtual environment

# Overview

- In your life as a Python developer, you'll likely create many applications that use diverse Python packages
- Ideally you would like the applications to be independent of each other
  - The Python packages you download for one application shouldn't interfere with the Python packages for other applications
- To help you keep Python application environments isolated from each other, you can use the `virtualenv` tool

# Installing virtualenv

- You install `virtualenv` via `pip` as a one-off exercise, as follows:

```
pip install virtualenv
```

- You can test your installation as follows:

```
virtualenv --version
```

# Creating a Virtual Environment for a Project

- To create a virtual environment for a particular project:

```
virtualenv MyProject
```

- This command creates a folder named MyProject that contains:
  - Python executable files
  - A copy of the pip library, which you can use to install other packages (locally for this virtual environment)

# Activating a Virtual Environment

- To begin using a virtual environment, you must activate it

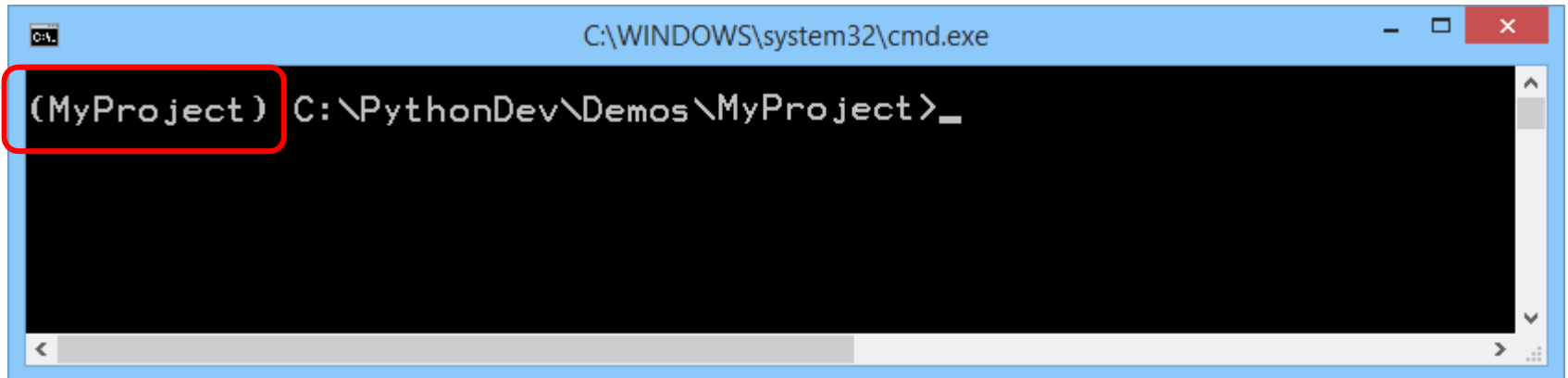
In Unix:

```
source MyProject/bin/activate
```

In Windows:

```
MyProject\Scripts\activate
```

- After you've activated a virtual environment, its name will appear in the command prompt
  - E.g. in Windows:



A screenshot of a Windows command prompt window. The title bar shows 'C:\WINDOWS\system32\cmd.exe'. The command prompt displays '(MyProject) C:\PythonDev\Demos\MyProject>'. The text '(MyProject)' is highlighted with a red rectangular box, indicating that the virtual environment has been successfully activated.

# Using a Virtual Environment

- You can now use pip to install packages into your virtual environment
  - E.g. to install the "requests" package:
  - Installs the package into the Lib\site-packages folder

```
pip install requests
```

- You can write a Python script that uses the package

```
import requests
response = requests.get('https://httpbin.org/ip')
print('Your IP is {0}'.format(response.json()['origin']))
```

main.py

- Run the Python script as normal

```
python main.py
```

# Deactivating a Virtual Environment

- You can deactivate a virtual environment as follows:

```
deactivate
```

- This tears down your virtual environment
  - You don't see the packages in that virtual environment any more
  - You can reactivate it whenever you need to (see 2 slides previous)