

CS316

INTRODUCTION TO AI AND DATA SCIENCE

CHAPTER 2

Python for Data Science

LECTURE 1

Introduction to Python Programming

Prof. Anis Koubaa

SEP 2024

PYTHON

WHY CHOOSE PYTHON

01

INTERPRETED LANGUAGE

Python is a **interpreted, object-oriented, high-level** programming **scripting** language with **dynamic semantics**
Developed in the late 1980s by Guido van Rossum



02

PYTHON IS SIMPLE

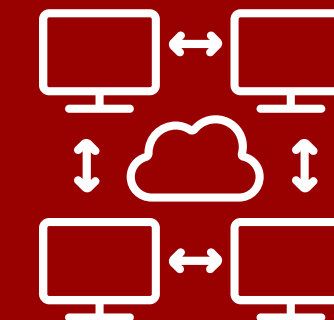
Python is simple and easy to learn syntax
Supports a large variety of standard libraries and packages
Increased productivity for developers – no segmentation fault

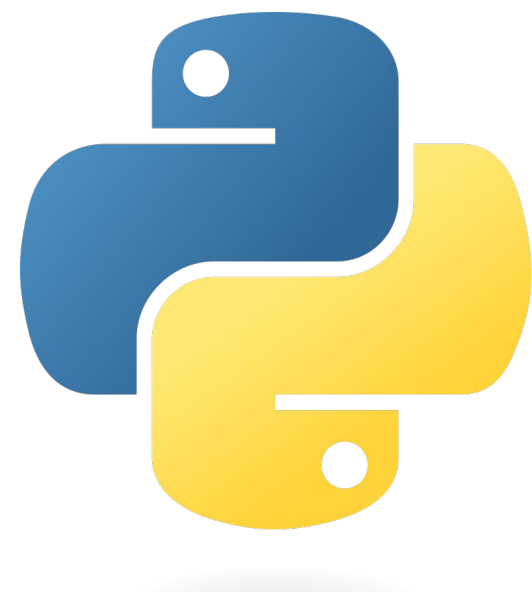


03

MULTI-PURPOSE LANGUAGE

Number 1 language for data science
Web development (back-end side – and recently front-end)
System programming (embedded)





Python Popularity

Prof. Anis Koubaa



SPECIALIZATION

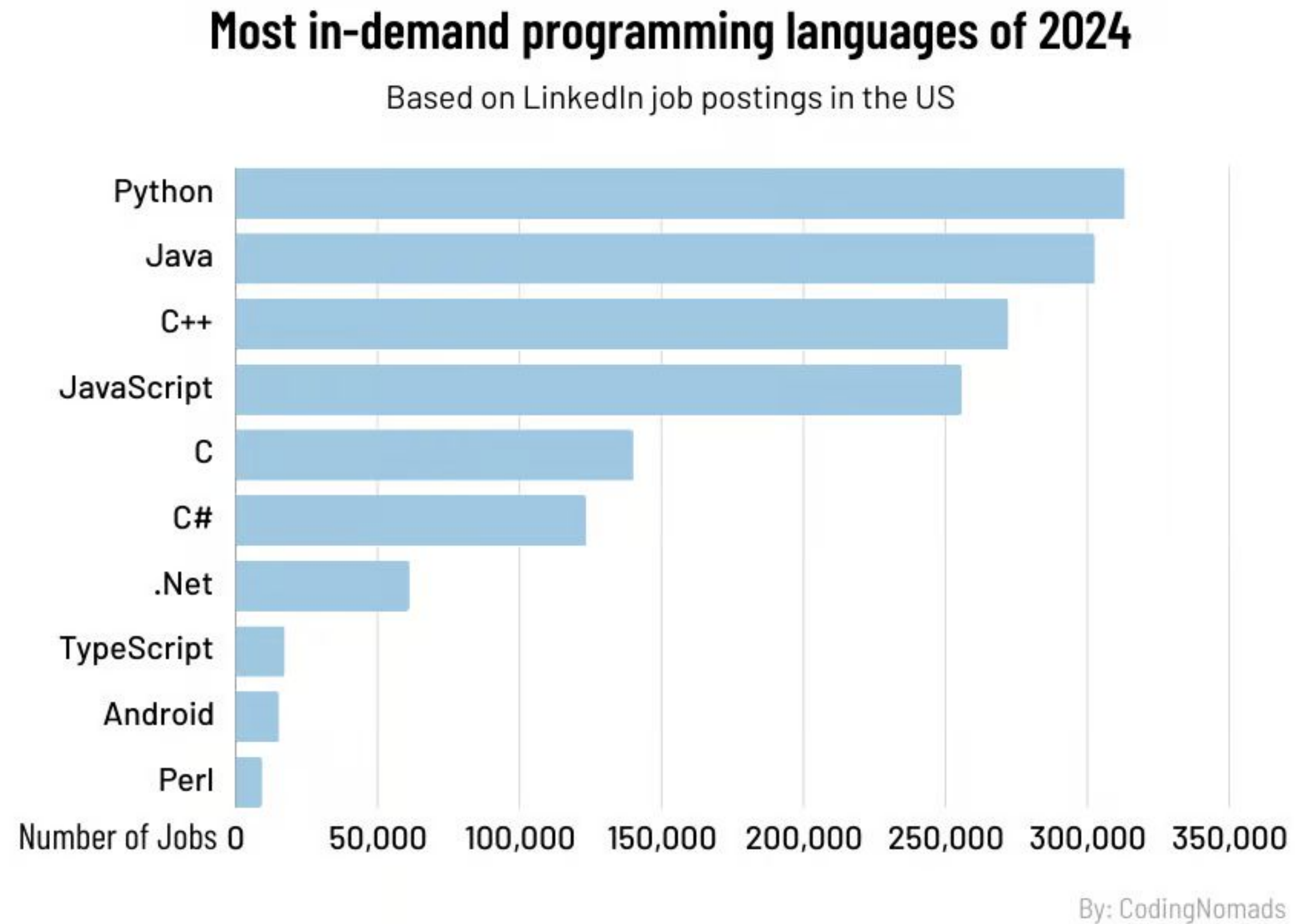
CS316
**INTRODUCTION TO AI AND DATA
SCIENCE**

CHAPTER 2
Python for Data Science

LECTURE 1
Introduction to Python Programming





















PYTHON POPULARITY

IS PYTHON THE MOST POPULAR?

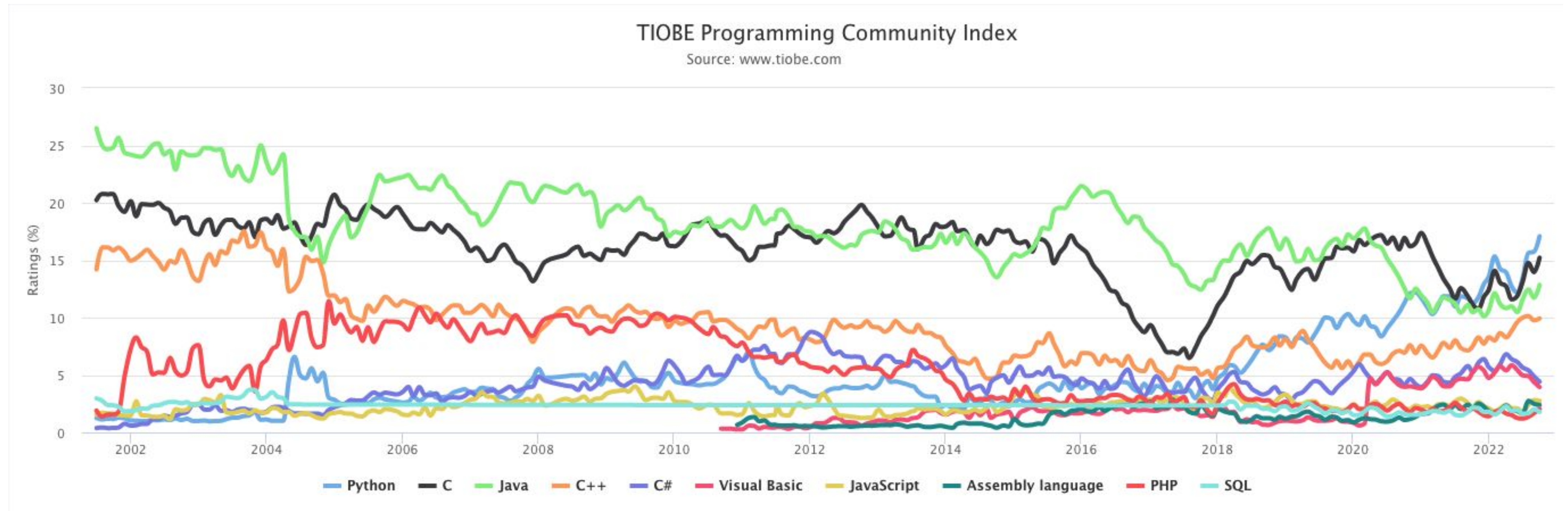


PYTHON POPULARITY

IS PYTHON THE MOST POPULAR?

Oct 2022	Oct 2021	Change	Programming Language		Ratings	Change
1	1			Python	17.08%	+5.81%
2	2			C	15.21%	+4.05%
3	3			Java	12.84%	+2.38%
4	4			C++	9.92%	+2.42%
5	5			C#	4.42%	-0.84%
6	6			Visual Basic	3.95%	-1.29%
7	7			JavaScript	2.74%	+0.55%
8	10	▲		Assembly language	2.39%	+0.33%
9	9			PHP	2.04%	-0.06%
10	8	▼		SQL	1.78%	-0.39%
11	12	▲		Go	1.27%	-0.01%
12	14	▲		R	1.22%	+0.03%
13	29	▲		Objective-C	1.21%	+0.76%
14	13	▼		MATLAB	1.18%	-0.02%
15	17	▲		Swift	1.05%	-0.06%
16	16			Ruby	0.88%	-0.24%
17	11	▼		Classic Visual Basic	0.87%	-0.96%
18	20	▲		Delphi/Object Pascal	0.85%	-0.09%
19	18	▼		Fortran	0.79%	-0.29%
20	26	▲		Rust	0.70%	+0.17%

PROGRAMMING LANGUAGES TRENDS



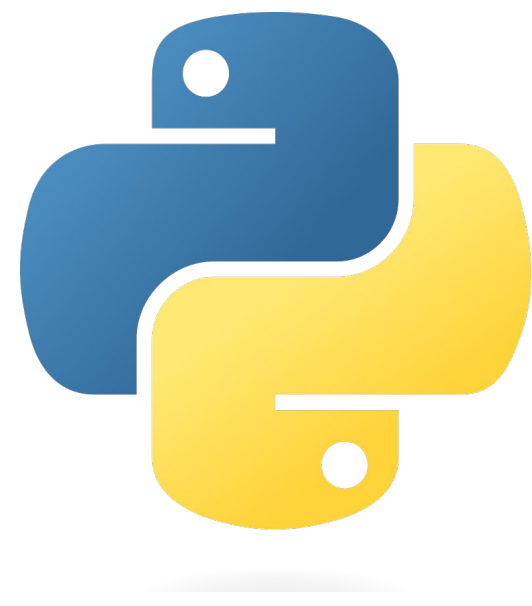
PROGRAMMING LANGUAGES

HISTORICAL RANKS

Programming Language	2022	2017	2012	2007	2002	1997	1992	1987
Python	1	5	8	7	12	28	-	-
C	2	2	1	2	2	1	1	1
Java	3	1	2	1	1	16	-	-
C++	4	3	3	3	3	2	2	6
C#	5	4	5	8	13	-	-	-
Visual Basic	6	15	-	-	-	-	-	-
JavaScript	7	8	10	9	8	23	-	-
Assembly language	8	10	-	-	-	-	-	-
SQL	9	-	-	-	7	-	-	-
PHP	10	7	6	5	6	-	-	-
Prolog	25	32	32	26	16	20	13	3
Lisp	31	31	13	16	14	9	5	2
Pascal	269	117	14	21	99	11	3	5
(Visual) Basic	-	-	7	4	4	3	6	4

BIG TECH APPS USING PYTHON





Install Python

Prof. Anis Koubaa



SPECIALIZATION

CS316
**INTRODUCTION TO AI AND DATA
SCIENCE**

CHAPTER 2
Python for Data Science

LECTURE 1
Introduction to Python Programming

Python Installation Options

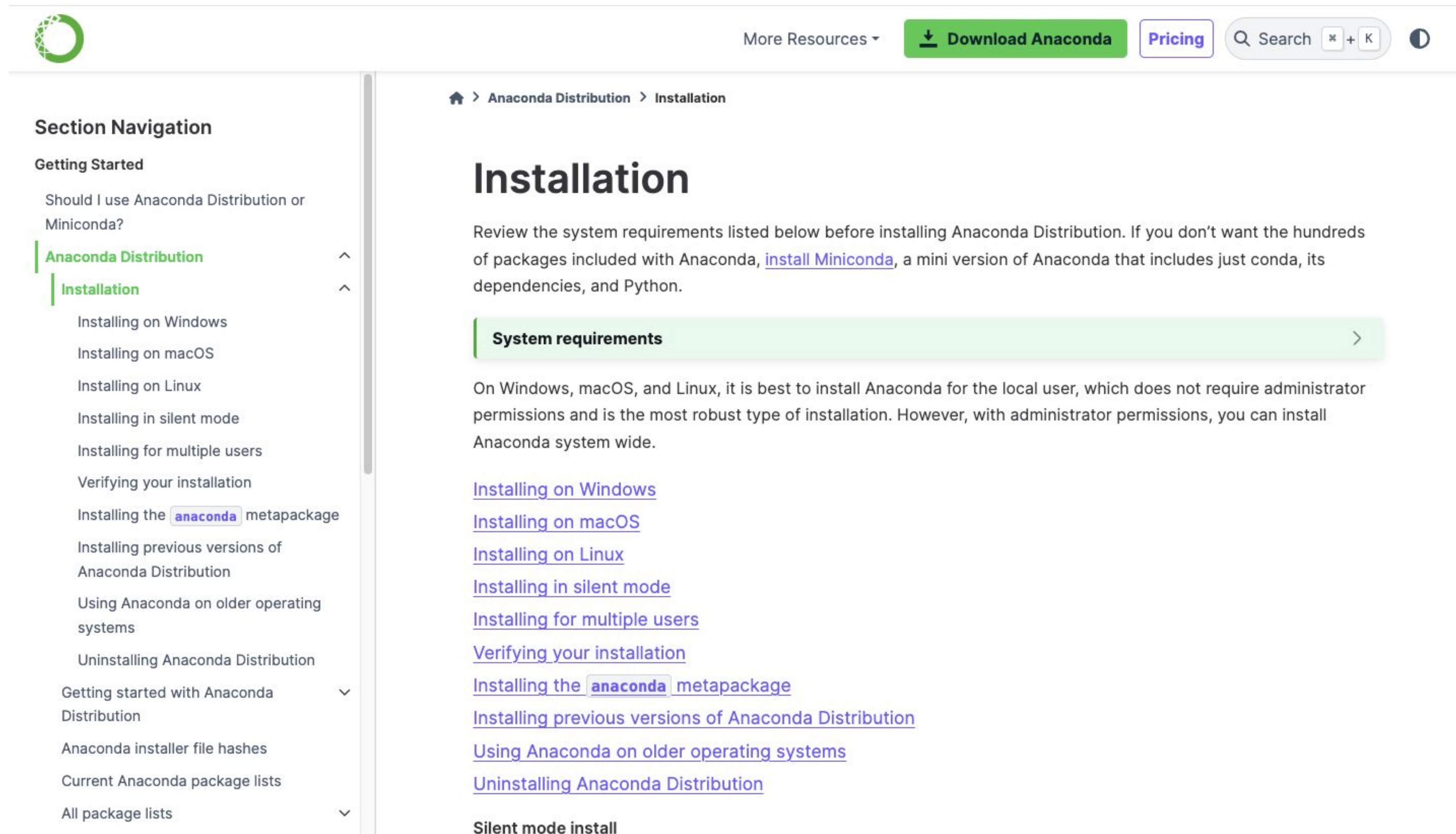
- **Official Python Distribution**

- **Availability:** Windows, Mac, Linux
- **Source:** Download directly from python.org
- **Instructions:** Follow installation steps for your OS, ensuring the addition of Python to the system PATH.

- **Anaconda Distribution (Recommended for Data Science)**

- **Includes:** Python, Conda (environment manager), and key pre-installed libraries (e.g., NumPy, Pandas, SciPy).
- **Compatibility:** Available for Windows, Mac, and Linux
- **Download:** Obtain from Anaconda.com
- **Advantages:** Simplifies package management and deployment with isolated environments.

Python Installation Options



The screenshot shows the Anaconda Distribution Installation page. The top navigation bar includes the Anaconda logo, a 'More Resources' dropdown, a green 'Download Anaconda' button, a 'Pricing' button, a search bar, and a user profile icon. The left sidebar, titled 'Section Navigation', lists various topics under 'Getting Started', with 'Anaconda Distribution' and 'Installation' highlighted. The main content area, titled 'Installation', provides an overview of the installation process and a list of links for different operating systems and installation methods. A green 'System requirements' section is also visible.

Section Navigation

- Getting Started
 - Should I use Anaconda Distribution or Miniconda?
 - Anaconda Distribution**
 - Installation**
 - Installing on Windows
 - Installing on macOS
 - Installing on Linux
 - Installing in silent mode
 - Installing for multiple users
 - Verifying your installation
 - Installing the `anaconda` metapackage
 - Installing previous versions of Anaconda Distribution
 - Using Anaconda on older operating systems
 - Uninstalling Anaconda Distribution
 - Getting started with Anaconda Distribution
 - Anaconda installer file hashes
 - Current Anaconda package lists
 - All package lists

Installation

Review the system requirements listed below before installing Anaconda Distribution. If you don't want the hundreds of packages included with Anaconda, [install Miniconda](#), a mini version of Anaconda that includes just conda, its dependencies, and Python.

System requirements

On Windows, macOS, and Linux, it is best to install Anaconda for the local user, which does not require administrator permissions and is the most robust type of installation. However, with administrator permissions, you can install Anaconda system wide.

- [Installing on Windows](#)
- [Installing on macOS](#)
- [Installing on Linux](#)
- [Installing in silent mode](#)
- [Installing for multiple users](#)
- [Verifying your installation](#)
- [Installing the `anaconda` metapackage](#)
- [Installing previous versions of Anaconda Distribution](#)
- [Using Anaconda on older operating systems](#)
- [Uninstalling Anaconda Distribution](#)

Silent mode install

<https://docs.anaconda.com/anaconda/install/>

Installing Python from the Terminal

1. Check Existing Python Installation

- **Command:** `python --version` Or `python3 --version`
- **Purpose:** Verify if Python is already installed and check the current version.

Installing Python from the Terminal

1. Installing Python on Different OS

- **Linux:**

- **Update Package List:** `sudo apt-get update`

- **Install Python:**

- **Python 3:** `sudo apt-get install python3`

- **Pip (Package Manager):** `sudo apt-get install python3-pip`

Installing Python from the Terminal

1. Installing Python on Different OS

- **macOS:**

- **Using Homebrew:**

- **Install Homebrew (if not installed):**

- ```
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

- **Install Python:**

- ```
brew install python
```


Installing Python from the Terminal

1. Installing Python on Different OS

- **Windows:**

- **Using Windows Subsystem for Linux (WSL):**

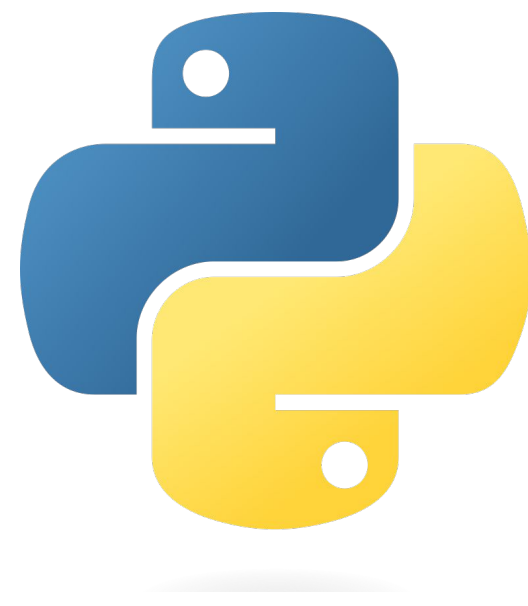
- **Enable WSL:**

- `wsl --install`

- **Install Python within WSL:**

- **Update:** `sudo apt-get update`

- **Install:** `sudo apt-get install python3`



Python IDE

Prof. Anis Koubaa



SPECIALIZATION

CS316
**INTRODUCTION TO AI AND DATA
SCIENCE**

CHAPTER 2
Python for Data Science

LECTURE 1
Introduction to Python Programming

Development Environments

1. IDLE

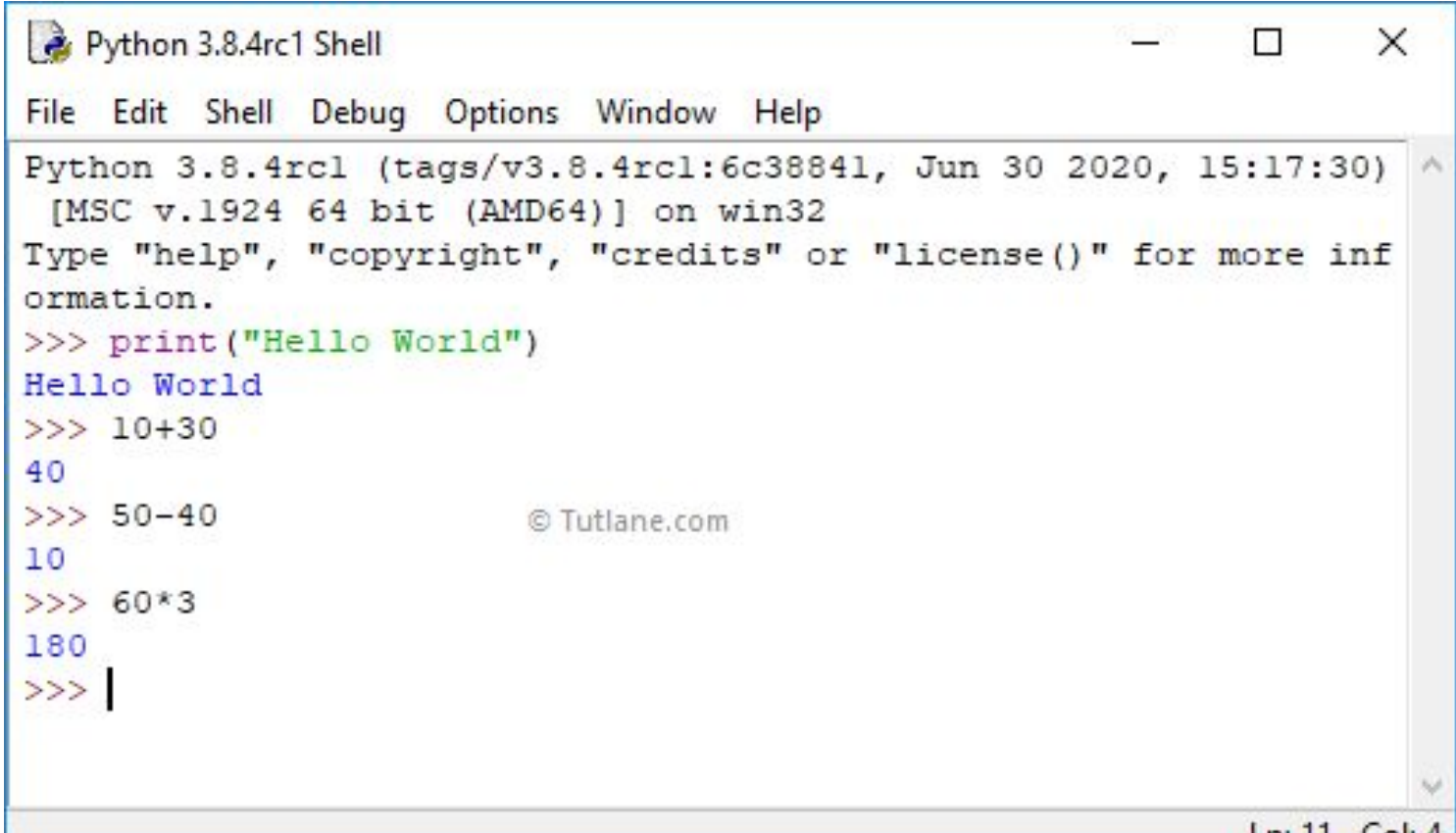
- **Description:** Integrated Development and Learning Environment, comes pre-installed with Python.
- **Use Case:** Suitable for beginners and small-scale projects.

2. PyCharm

- **Versions:** Community Edition (free) and Professional Edition.
- **Features:** Tailored for Python development with advanced debugging, code analysis, and refactoring tools.

3. Visual Studio Code (VS Code)

- **Description:** Free, open source, and highly customizable.
- **Python Support:** Enhanced through extensions like Python and Pylance, ideal for various development tasks.



```
Python 3.8.4rc1 Shell
File Edit Shell Debug Options Window Help
Python 3.8.4rc1 (tags/v3.8.4rc1:6c38841, Jun 30 2020, 15:17:30)
[MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more inf
ormation.
>>> print("Hello World")
Hello World
>>> 10+30
40
>>> 50-40
10
>>> 60*3
180
>>> |
```



Notebook Development Environments

1. Jupyter Notebook


- **Description:** Ideal for interactive Python sessions, especially in data science.
- **Access:** Available via Anaconda distribution or `pip install notebook`.



2. Google Colab

- **Versions:** Free cloud-based Jupyter notebook environment.
- **Advantages:** Provides GPU/TPU support, easy sharing, and no local setup required.



 01-python-crash-course-part-01.ipynb ☆

File Edit View Insert Runtime Tools Help [Last edited on Dec 31, 2019](#)

+ Code + Text

Part 3

- Functions
- Lambda Expressions
- Map and Filters

This review is not intended to be comprehensive, but provides to the main features of the Python programming language, which should good enough to proceed further.

+ Code + Text

▼ 1. Basic Syntax

We start by printing a simple Hello World text message using the method 'print'. Note that in Python 3 is necessary to put the text into paranthesis, where as in Python 2 it was optional

```
[ ] #print hello world
    print ("Hello world, Python3!")
```

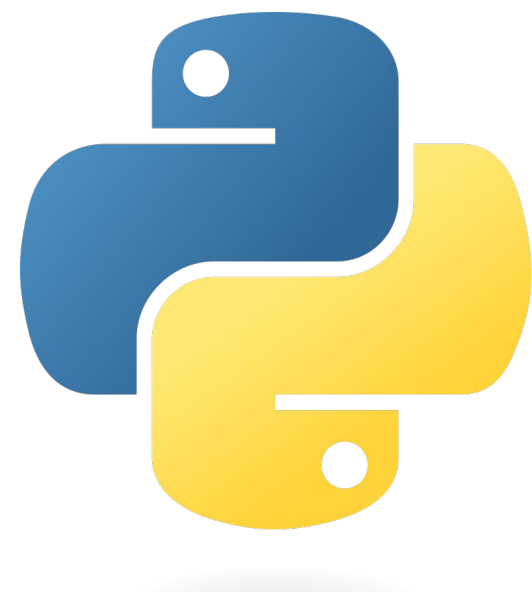
Hello world, Python3!

▼ 1.1 Indentation

Indentation is very important in Python. In termine blocks as no parenthesis are used. This is an example of an if statement

```
[ ] i = 0
    if (i==0):
        i=i+1
        print ("i=",i)
    n=1
    print ("n=",n)
```

i= 1
n= 1



Python Libraries

Prof. Anis Koubaa



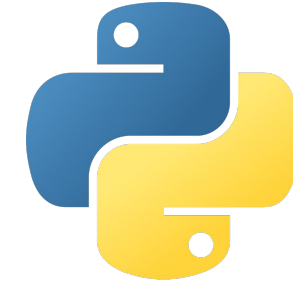
SPECIALIZATION

CS316
**INTRODUCTION TO AI AND DATA
SCIENCE**

CHAPTER 2
Python for Data Science

LECTURE 1
Introduction to Python Programming

PYTHON FOR DATA SCIENCE LIBRARIES/PACKAGES



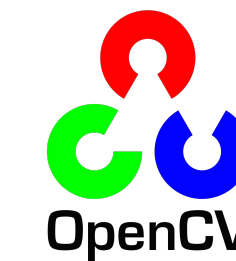
DATA STRUCTURES



VISUALIZATION



SCIENTIFIC COMPUTING



STORAGE AND DATABASES



DEEP LEARNING





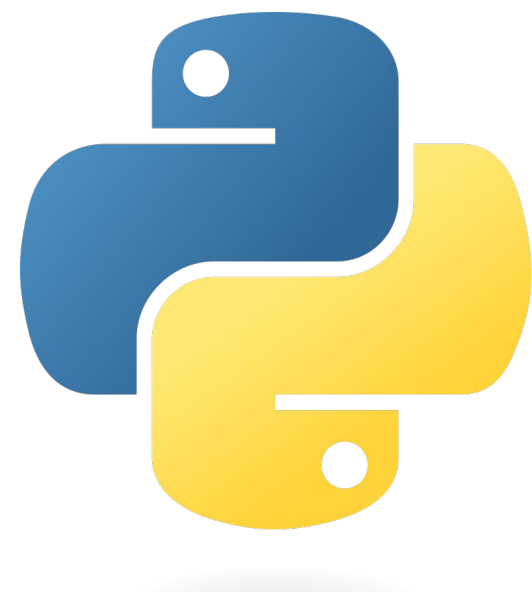
CS316 INTRODUCTION TO AI AND DATA SCIENCE

CHAPTER 2 Python for Data Science

LECTURE 2 *Introduction to Python Programming* *Virtual Envs, Setting-up a Python Package*

Prof. Anis Koubaa

SEP 2024



Python Package Structure

Prof. Anis Koubaa



SPECIALIZATION

CS316
**INTRODUCTION TO AI AND DATA
SCIENCE**

CHAPTER 2
Python for Data Science

LECTURE 1
Introduction to Python Programming

Python Module

1. Definition

- A single file containing Python code, typically with a `.py` extension.

2. Example

- `example.py`

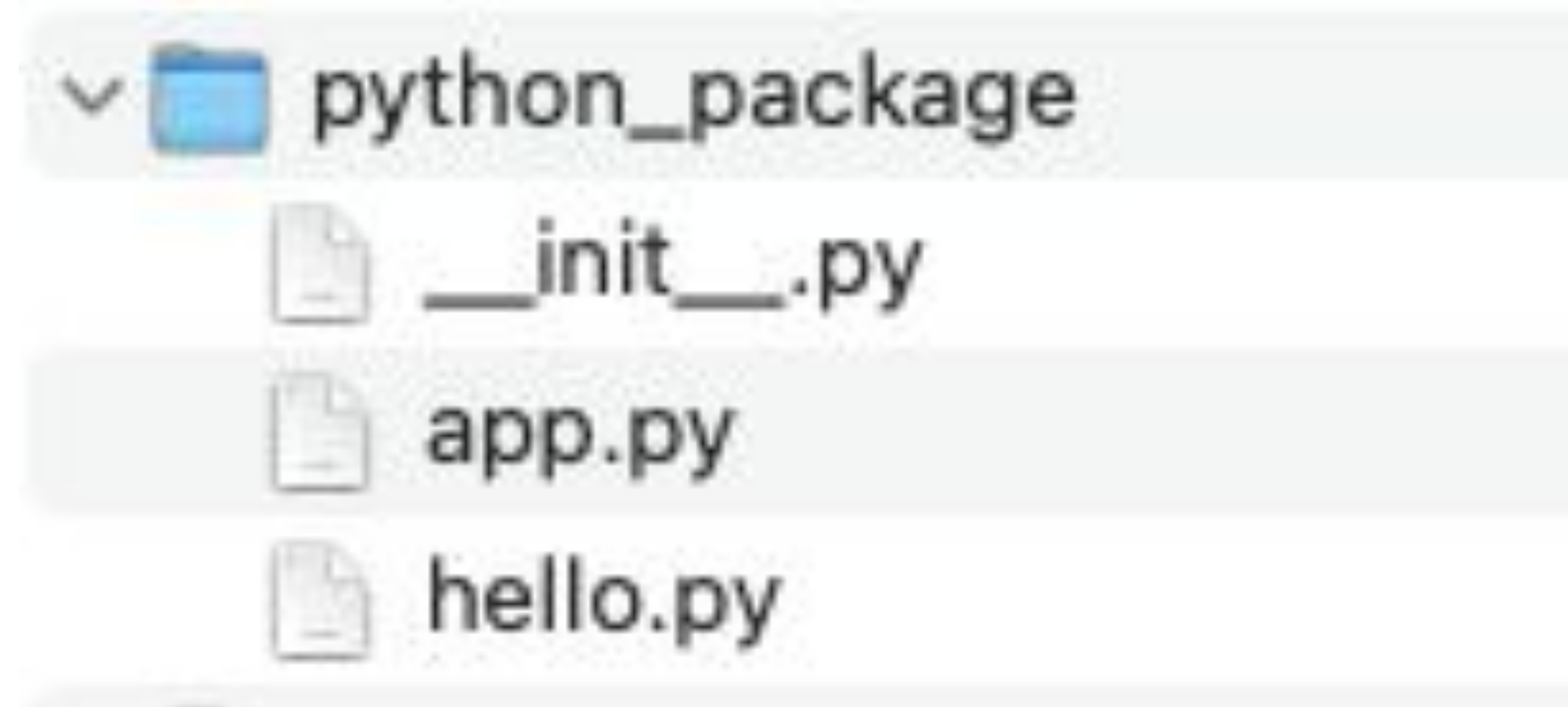
3. Usage

- Functions, classes, and variables defined in the file can be imported into other Python scripts.
- Import Syntax: `import example`
- Example Code:

```
python Copy code

# In example.py
def greet():
    print("Hello, world!")

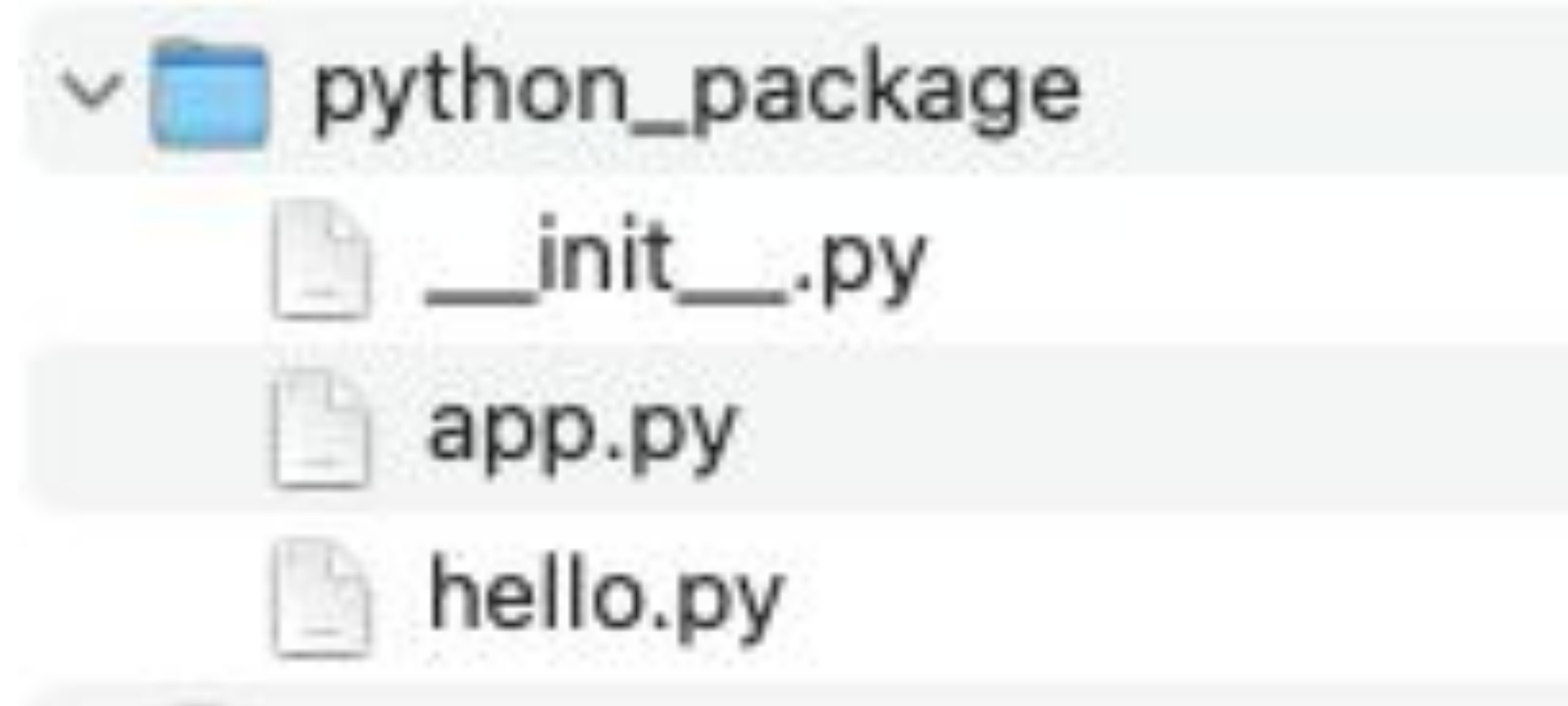
# In another script
import example
example.greet()
```



Python Package

2. Package

- **Definition:** A directory (folder) containing multiple modules, and an `__init__.py` file.
- **Purpose:** Organizes related modules under a single namespace.
- **Example Structure:**
 - ``mypackage/``
 - ``__init__.py`` (can be empty)
 - ``module1.py``
 - ``module2.py``
- **Import Syntax:**
 - ``from mypackage import module1``
 - ``import mypackage.module2``



Python Libraries


3. Standard Library

- **Description:** A rich set of modules and packages included with Python by default.
- **Purpose:** Provides tools for various tasks such as file I/O, system calls, and web development.
- **Examples:** ``os``, ``sys``, ``datetime``, ``json``

4. Third-Party Packages

- **Definition:** External libraries developed by the community, available through package managers like ``pip``.
- **Purpose:** Extend Python's functionality for specific use cases, such as data science or web development.
- **Examples:** ``numpy``, ``requests``, ``pandas``

python

 Copy code

```
# Importing standard library module
import datetime

# Importing third-party library module
import pandas as pd

def main():
    # Using datetime to get today's date
    today = datetime.date.today()

    # Using pandas to create a DataFrame with today's date
    df = pd.DataFrame({'Date': [today]})
    print(df)

if __name__ == "__main__":
    main()
```

PYTHON FILE STRUCTURE

MODULE vs. PACKAGE

PACKAGE

A collection of Python module. It has a `__init__.py` file to distinguish it from a classical director

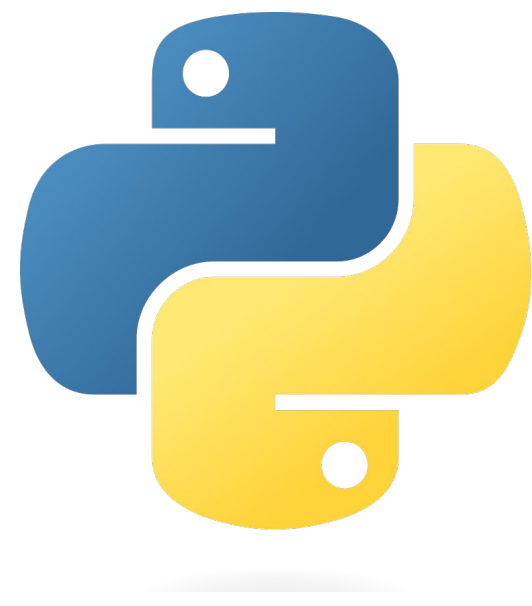
MODULE-1

A module is a file that contains a Python script in runtime for the code specified to the users.

MODULE-2

A module is a file that contains a Python script in runtime for the code specified to the users.





Python Virtual Environment

Prof. Anis Koubaa



SPECIALIZATION

CS316
**INTRODUCTION TO AI AND DATA
SCIENCE**

CHAPTER 2
Python for Data Science

LECTURE 1
Introduction to Python Programming

Creating a Virtual Environment with **pip**

2. Creating a Virtual Environment with `pip`

- Steps:

1. Install `venv` (if not installed):

```
bash Copy code  
  
python -m ensurepip --upgrade
```

2. Create the Environment:

```
bash Copy code  
  
python -m venv myenv
```

3. Activate the Environment:

- Windows:

```
bash Copy code  
  
myenv\Scripts\activate
```

- macOS/Linux:

```
bash Copy code  
  
source myenv/bin/activate
```

4. Deactivate:

```
bash Copy code  
  
deactivate
```

Creating a Virtual Environment with conda

3. Creating a Virtual Environment with `conda`

- Steps:

1. Create the Environment:

```
bash Copy code  
  
conda create --name myenv
```

2. Activate the Environment:

```
bash Copy code  
  
conda activate myenv
```

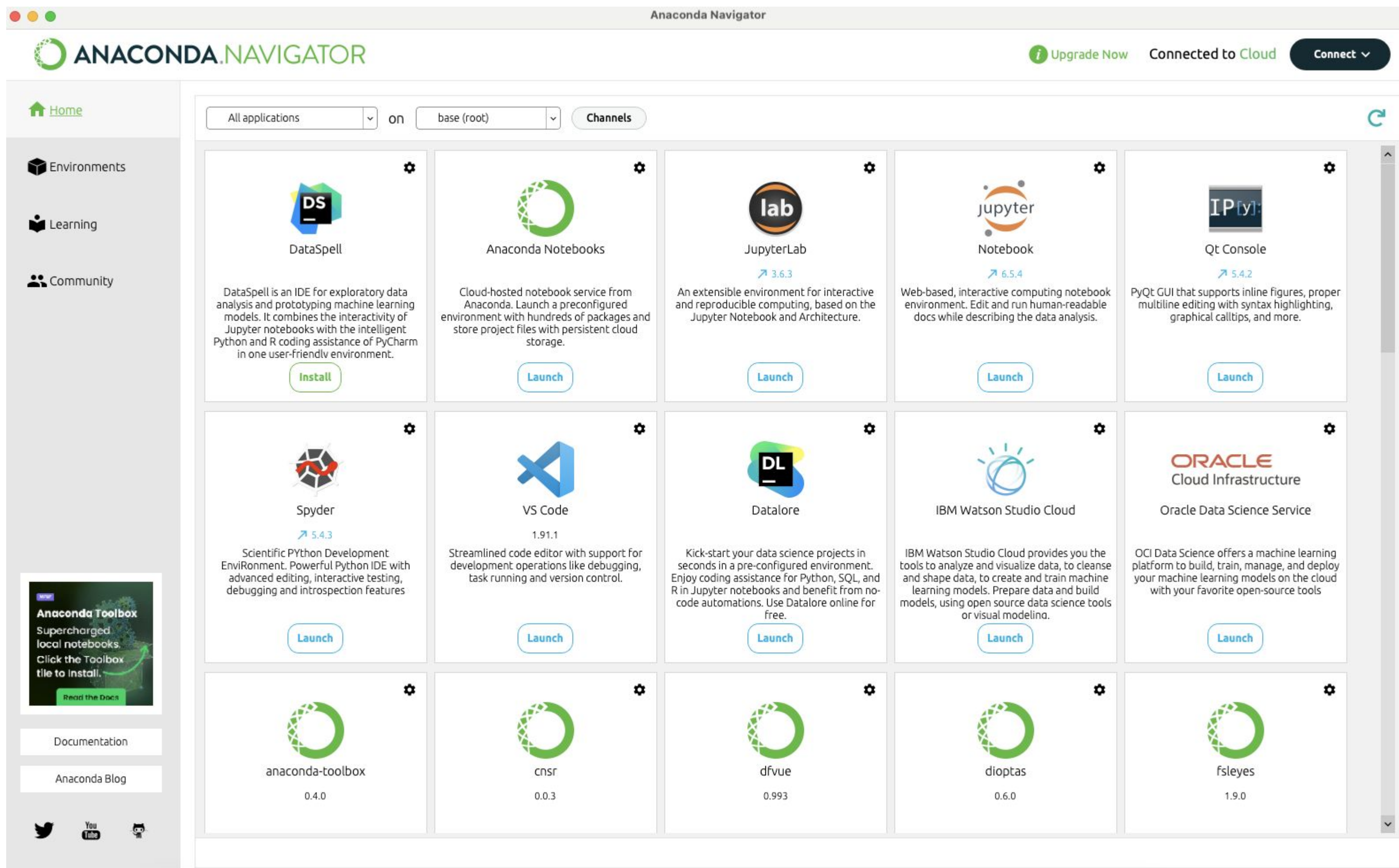
3. Deactivate:

```
bash Copy code  
  
conda deactivate
```

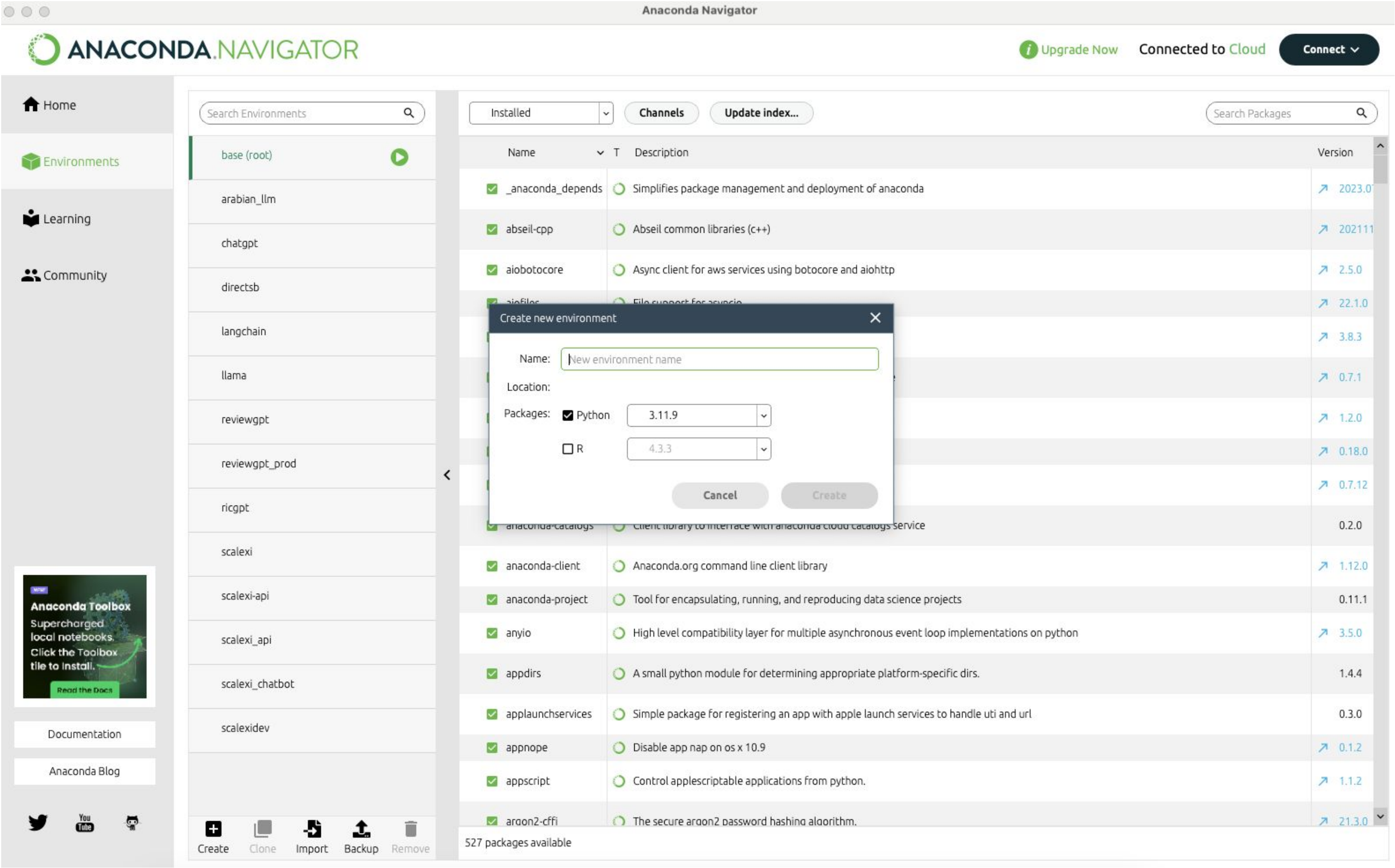
4. Remove the Environment:

```
bash Copy code  
  
conda remove --name myenv --all
```

Creating a Virtual Environment with conda



Creating a Virtual Environment with conda



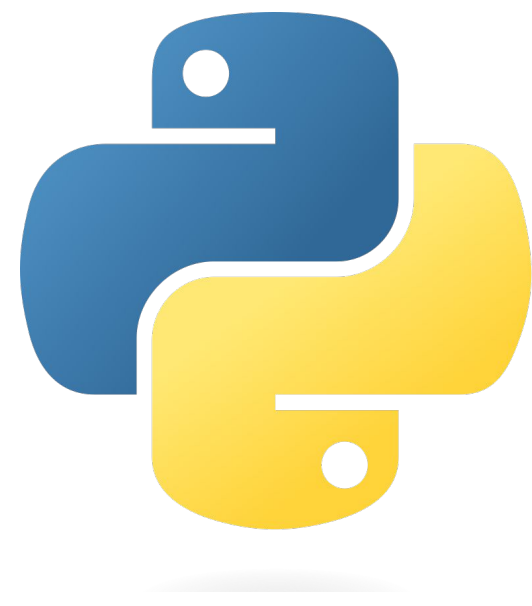
Managing Packages within the Environment

4. Managing Packages within the Environment

- Using ``pip``:
 - Install a package: ``pip install package_name``
 - List installed packages: ``pip list``
- Using ``conda``:
 - Install a package: ``conda install package_name``
 - List installed packages: ``conda list``

5. Considerations

- ``pip``: Works with the standard Python distribution; great for most projects.
- ``conda``: Ideal for data science projects with complex dependencies, especially when working with packages that require compiled binaries (e.g., ``numpy``, ``scipy``).



Setup a Python Package

1. Package Structure

Prof. Anis Koubaa



SPECIALIZATION

CS316
**INTRODUCTION TO AI AND DATA
SCIENCE**

CHAPTER 2
Python for Data Science

LECTURE 1
Introduction to Python Programming

Python Package Structure

1. Package Structure

- Structure Overview:

```
plaintext Copy code

mypackage/
├── mypackage/
│   ├── __init__.py
│   ├── module1.py
│   └── module2.py
├── tests/
│   ├── test_module1.py
│   └── test_module2.py
├── setup.py
├── README.md
└── LICENSE
```

- Key Files:

- `__init__.py`: Marks the directory as a Python package.
- `setup.py`: Script for packaging and distribution.
- `README.md`: Documentation for the package.
- `LICENSE`: License information.

Writing `setup.py`

2. Basic `setup.py` Script Example

```
python Copy code

from setuptools import setup, find_packages

setup(
    name="mypackage",
    version="0.1.0",
    author="Your Name",
    author_email="your.email@example.com",
    description="A short description of the package",
    long_description=open("README.md").read(),
    long_description_content_type="text/markdown",
    url="https://github.com/yourusername/mypackage",
    packages=find_packages(),
    classifiers=[
        "Programming Language :: Python :: 3",
        "License :: OSI Approved :: MIT License",
        "Operating System :: OS Independent",
    ],
    python_requires='>=3.6',
    install_requires=[
        "numpy>=1.21.0", # Numerical computations
        "pandas>=1.3.0", # Data manipulation and analysis
        "requests>=2.26.0", # HTTP library for making requests
        "matplotlib>=3.4.0" # Plotting and visualization
    ],
    entry_points={
        'console_scripts': [
            'mypackage-cli=mypackage.command_line:main',
        ],
    },
)
```

1. Purpose

- `setup.py` is the script used for packaging and distributing your Python project. It contains metadata about the package and specifies dependencies.

3. Explanation

- `name`: The name of the package.
- `version`: The current version of the package.
- `author` and `author_email`: Your details.
- `description`: A brief description of the package.
- `long_description`: Detailed description, often pulled from `README.md`.
- `url`: The URL of the project's repository or website.
- `packages`: Automatically find all packages to include in the distribution.
- `classifiers`: Metadata about the package (e.g., programming language, license).
- `python_requires`: Specifies the Python version compatibility.
- `install_requires`: Lists the dependencies that will be installed with the package.
 - Examples:
 - `numpy`: A library for numerical computations.
 - `pandas`: A library for data manipulation and analysis.
 - `requests`: A library for making HTTP requests.
 - `matplotlib`: A library for plotting and visualization.
- `entry_points`: Defines entry points for creating command-line scripts.

README

MyPackage

Overview

MyPackage is a simple Python package designed for educational purposes. It provides basic utility functions that demonstrate key concepts in Python, such as string manipulation, mathematical operations, and data handling.

Features

- **String Utilities:** Functions to manipulate and analyze strings.
- **Math Utilities:** Basic mathematical operations, including addition, subtraction, multiplication, and division.
- **Data Utilities:** Simple data processing functions.

Installation

You can install **MyPackage** using `pip`:

```
bash
pip install mypackage
```

Alternatively, you can clone the repository and install it manually:

```
bash
git clone https://github.com/yourusername/mypackage.git
cd mypackage
pip install .
```

Usage

Below are some examples of how to use **MyPackage**:

1. String Utilities

```
python
from mypackage import string_utils

# Reverse a string
reversed_string = string_utils.reverse_string("hello")
print(reversed_string) # Output: "olleh"

# Count vowels in a string
vowel_count = string_utils.count_vowels("hello")
print(vowel_count) # Output: 2
```

README

Usage

Below are some examples of how to use MyPackage:

1. String Utilities

```
python
Copy code

from mypackage import string_utils

# Reverse a string
reversed_string = string_utils.reverse_string("hello")
print(reversed_string) # Output: "olleh"

# Count vowels in a string
vowel_count = string_utils.count_vowels("hello")
print(vowel_count) # Output: 2
```

2. Math Utilities

```
python
Copy code

from mypackage import math_utils

# Add two numbers
result = math_utils.add(5, 3)
print(result) # Output: 8

# Divide two numbers
result = math_utils.divide(10, 2)
print(result) # Output: 5.0
```

3. Data Utilities

```
python
Copy code

from mypackage import data_utils

# Sum of a list of numbers
total = data_utils.sum_list([1, 2, 3, 4])
print(total) # Output: 10

# Find maximum in a list of numbers
maximum = data_utils.find_max([1, 2, 3, 4])
print(maximum) # Output: 4
```

Modules

1. string_utils

- ``reverse_string(s: str) -> str``: Reverses the given string ``s``.
- ``count_vowels(s: str) -> int``: Counts the number of vowels in the given string ``s``.

2. math_utils

- ``add(a: int, b: int) -> int``: Returns the sum of ``a`` and ``b``.
- ``subtract(a: int, b: int) -> int``: Returns the difference between ``a`` and ``b``.
- ``multiply(a: int, b: int) -> int``: Returns the product of ``a`` and ``b``.
- ``divide(a: int, b: int) -> float``: Returns the division of ``a`` by ``b``.

3. data_utils

- ``sum_list(numbers: list) -> int``: Returns the sum of all numbers in the list.

README

Contributing

Contributions are welcome! If you have any suggestions or find any bugs, please feel free to open an issue or submit a pull request.

Steps to Contribute:

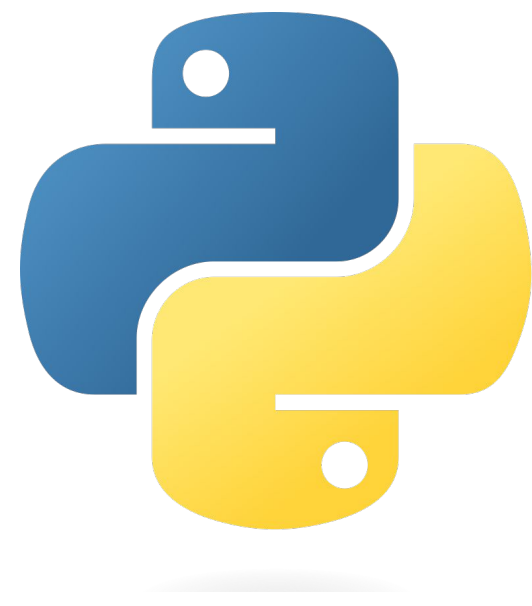
1. Fork the repository.
2. Create a new branch (`git checkout -b feature-branch`).
3. Make your changes.
4. Commit your changes (`git commit -m 'Add new feature'`).
5. Push to the branch (`git push origin feature-branch`).
6. Open a pull request.

License

This project is licensed under the MIT License - see the [LICENSE](#) file for details.

Contact

For any questions or suggestions, feel free to contact me at your.email@example.com.



Setup a Python Package

2. PyPi Repository

Prof. Anis Koubaa



SPECIALIZATION

CS316
**INTRODUCTION TO AI AND DATA
SCIENCE**

CHAPTER 2
Python for Data Science

LECTURE 1
Introduction to Python Programming

Understanding PyPI

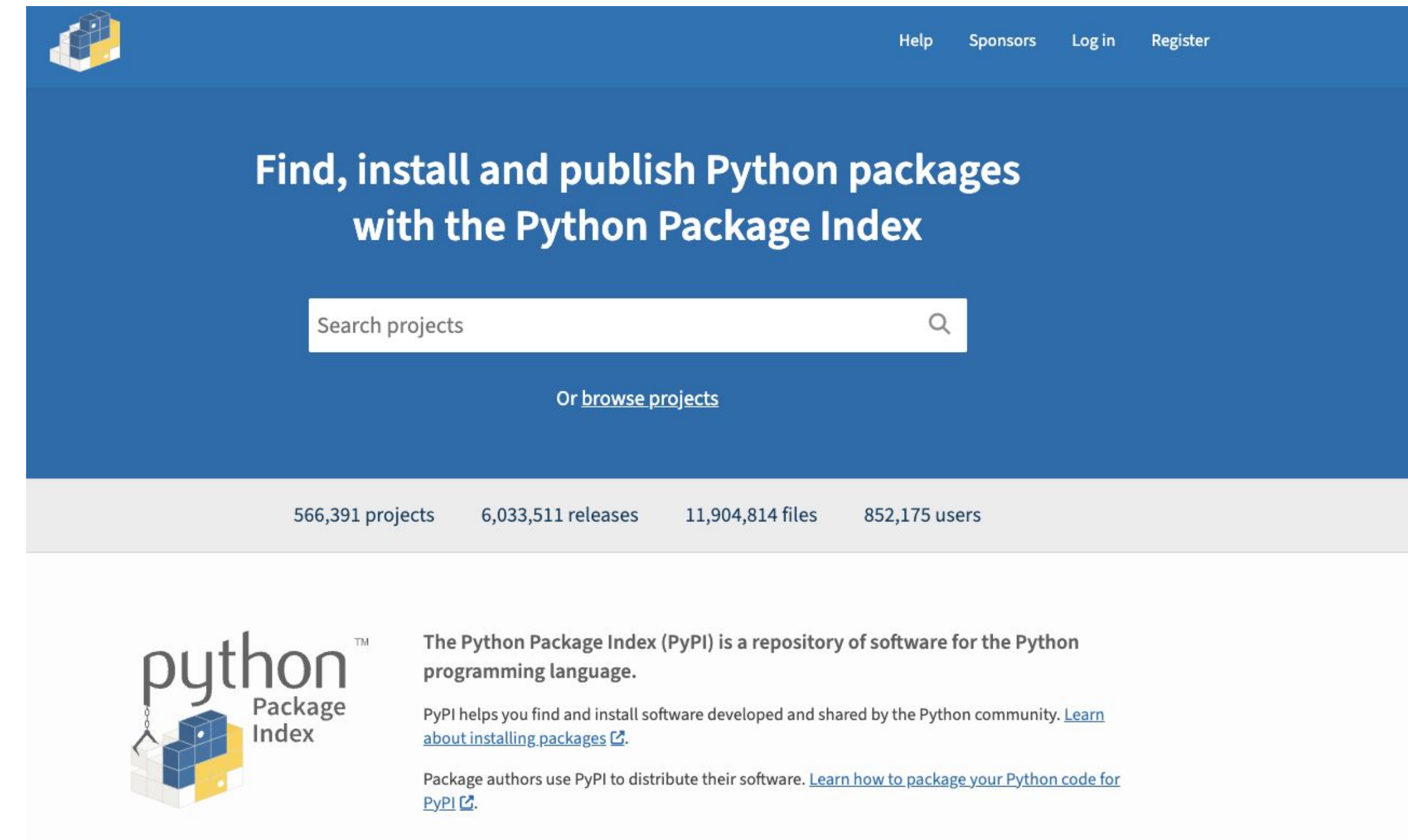
<https://pypi.org/>

1. What is PyPI?


- **Definition:** The Python Package Index (PyPI) is the official third-party software repository for Python.
- **Purpose:** Allows developers to distribute their Python software and libraries to users.

2. Key Features

- **Package Management:** Easy installation and updating of packages using `pip`.
- **Searchable:** Users can search for and discover packages based on functionality and popularity.
- **Community-Driven:** Contributions from developers around the world.



Understanding PyPI



numpy

HelpSponsorsLog inRegister

Filter by [classifier](#)

Framework

Topic

Development Status

License

Programming Language

Operating System

Environment

Intended Audience


Natural Language

Typing

10,000+ projects for "numpy"


Order by

Relevance

 **numpy 2.1.0**


Aug 19, 2024

Fundamental package for array computing in Python

 **numpy1 0.0.1**


Jan 24, 2021

Python

 **numpy4 0.7**


Apr 26, 2024

A package for doing cool stuff

 **numpy5 0.1**


Apr 28, 2024

None

 **numpy6 2.0**

Apr 29, 2024

None

 **gdal2numpy 0.0.336**

Aug 6, 2024

An utils functions package

Example

Your account

Your projects

Your organizations

Account settings

Publishing

Your projects1

scalexi

SOLE OWNER

Last released Aug 7, 2024

The scalexi package is a versatile open-source Python library that focuses on facilitating low-code development and fine-tuning of diverse Large Language Models (LLMs). It extends beyond its initial OpenAI models integration, offering a scalable framework for various LLMs.

Manage


View


CS316: INTRODUCTION TO DATA SCIENCE

INTRODUCTION TO DATA SCIENCE

ANIS KOUBAA | 2024 413

Example



 scalexi ▾


scalexi 0.4.7.16

✓

Latest version

Released: Aug 7, 2024


`pip install scalexi`





The scalexi package is a versatile open-source Python library that focuses on facilitating low-code development and fine-tuning of diverse Large Language Models (LLMs). It extends beyond its initial OpenAI models integration, offering a scalable framework for various LLMs.

Manage project

Navigation

 Project description


 Release history

 Download files

Verified details [\(What is this?\)](#)

These details have been verified by PyPI

Maintainers

 scalexi

Project description

Overview

scalexi is a versatile open-source Python library that focuses on facilitating low-code development and fine-tuning of diverse Large Language Models (LLMs). It extends beyond its initial OpenAI models integration, offering a scalable framework for various LLMs.

Key to scalexi is its low-code approach, significantly reducing the complexity of dataset preparation and manipulation. It features advanced dataset conversion tools, adept at transforming raw contextual data into structured datasets fulfilling LLMs fine-tuning requirements. These tools support multiple question formats, like open-ended, closed-ended, yes-no, and reflective queries, streamlining the creation of customized datasets for LLM fine-tuning.

A standout feature is the library's automated dataset generation, which eases the workload involved in LLM training.

Best Practices for PyPI

1. Versioning

- Use Semantic Versioning: Follow the `MAJOR.MINOR.PATCH` format.
- Example: `1.0.0` for initial stable release.

2. Documentation

- Include a `README.md`: Helps users understand the package's purpose and usage.
- Provide Examples: Demonstrate how to use your package.

3. Testing Before Upload

- Use TestPyPI: Upload to TestPyPI to ensure everything works correctly before the official release.
- Continuous Integration: Implement CI/CD pipelines to automate testing and deployment.

Setting Up PyPI Credentials for Uploading a Wheel

1. Create a PyPI Account

- Steps:
 1. Go to pypi.org.
 2. Click on "Register" and create a new account.
 3. Verify your email address to activate your account.

2. Install `twine`

- **Purpose:** `twine` is the recommended tool for securely uploading your distribution packages to PyPI.
- **Command:**

```
bash
```

```
Copy code
```

```
pip install twine
```


Setting Up PyPI Credentials for Uploading a Wheel

3. Set Up PyPI Credentials

- Option 1: Use `.pypirc` File (Recommended)
 1. Create a `.pypirc` file in your home directory (`~/.pypirc` on Linux/macOS or `%USERPROFILE%\pypirc` on Windows).
 2. Add the following configuration:

```
ini Copy code

[distutils]
index-servers =
    pypi

[pypi]
username = your-username
password = your-password
```

3. Explanation:

- The `username` and `password` are your PyPI credentials.
- This file allows you to avoid entering your credentials every time you upload a package.

Setting Up PyPI Credentials for Uploading a Wheel

3. Set Up PyPI Credentials

- Option 1: Use `.pypirc` File (Recommended)

1. Create a `.pypirc` file in your home directory (`~/.pypirc` on Linux/macOS or `%USERPROFILE%\pypirc` on Windows).
2. Add the following configuration:

```
ini Copy code

[distutils]
index-servers =
    pypi

[pypi]
username = your-username
password = your-password
```

3. Explanation:

- The `username` and `password` are your PyPI credentials.
- This file allows you to avoid entering your credentials every time you upload a package.

- Option 2: Manual Entry

- When using `twine` without a `.pypirc` file:

```
bash Copy code

twine upload dist/*
```

- Prompt: `twine` will prompt you for your PyPI username and password during the upload process.

4. Upload the Wheel to PyPI

- Build the Wheel:

```
bash Copy code

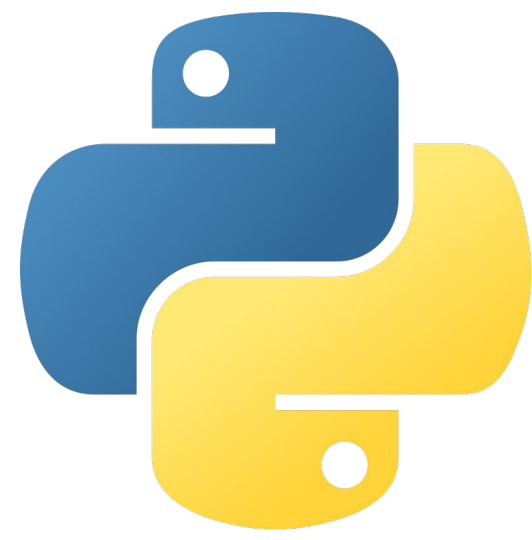
python setup.py bdist_wheel
```

- Upload the Package:

```
bash Copy code

twine upload dist/*
```

- Outcome: Your package will be uploaded to PyPI and can be installed by others using `pip install your-package-name`.



Setup a Python Package

3. Create a Wheel (Python Distribution)

Prof. Anis Koubaa



SPECIALIZATION

CS316
**INTRODUCTION TO AI AND DATA
SCIENCE**

CHAPTER 2
Python for Data Science

LECTURE 1
Introduction to Python Programming

Python Wheel

1. What is a Wheel?

- **Definition:** A **Wheel** is a built distribution format for Python packages. It is the standard format for distributing Python libraries and applications and is designed to be faster and simpler to install compared to source distributions (`.tar.gz`).
- **File Extension:** `.whl`

2. Advantages of Using Wheels

- **Faster Installation:** Wheels are pre-compiled, which speeds up the installation process because they don't require building or compiling code on the target machine.
- **Compatibility:** Wheels ensure compatibility across different environments, as they bundle all the necessary files and metadata.
- **Ease of Use:** Installing a wheel is as simple as running `pip install package_name.whl`.

Python Package Structure for Wheel Distribution

1. Package Structure Overview

plaintext

Copy code

```
mypackage/
├── mypackage/
│   ├── __init__.py      # Initializes the package
│   ├── module1.py       # Example module 1
│   └── module2.py       # Example module 2
├── tests/
│   ├── test_module1.py  # Unit tests for module 1
│   └── test_module2.py  # Unit tests for module 2
├── dist/                # Distribution files, including .whl
│   └── mypackage-0.1.0-py3-none-any.whl # The Wheel file
├── build/               # Build artifacts created by setup.py
├── mypackage.egg-info/  # Metadata about the package
├── setup.py            # Script for packaging and distribution
├── README.md           # Package description and usage
└── LICENSE             # License information
```

Key Components

- `mypackage/`` Directory: Contains the actual Python package with modules.
- `__init__.py``: Initializes the package and can define what is exposed to users.
- `tests/`` Directory: Contains unit tests for the package.
- `dist/`` Directory: Stores the distribution files, including the `.whl`` file after building.
- `build/`` Directory: Contains temporary build artifacts created during the package build process.
- `mypackage.egg-info/`` Directory: Contains metadata about the package, including version, dependencies, and more.
- `setup.py``: Script that defines the package details and is used to build the distribution files.
- `README.md``: Provides a detailed description of the package, including installation and usage instructions.
- `LICENSE``: Specifies the licensing terms under which the package is distributed.

Building the Package and Creating a Wheel

1. Building the Package

- Command:

```
bash
python setup.py sdist bdist_wheel
```

[Copy code](#)

- Explanation:

- ``sdist``: Creates a source distribution.
- ``bdist_wheel``: Creates a binary distribution (wheel).

2. Resulting Files

- The command generates the following files in the ``dist/`` directory:
 - ``mypackage-0.1.0.tar.gz``: Source distribution.
 - ``mypackage-0.1.0-py3-none-any.whl``: Wheel file.

1. Package Structure Overview

```
plaintext
mypackage/
├── mypackage/
│   ├── __init__.py    # Initializes the package
│   ├── module1.py     # Example module 1
│   └── module2.py     # Example module 2
├── tests/
│   ├── test_module1.py # Unit tests for module 1
│   └── test_module2.py # Unit tests for module 2
├── dist/              # Distribution files, including .whl
│   └── mypackage-0.1.0-py3-none-any.whl # The Wheel file
├── build/             # Build artifacts created by setup.py
├── mypackage.egg-info/ # Metadata about the package
├── setup.py           # Script for packaging and distribution
├── README.md          # Package description and usage
└── LICENSE             # License information
```

[Copy code](#)

Python Package Distribution Files

1. Generated Files After Building the Package

After running the build command (`python setup.py sdist bdist_wheel`), two key files are generated in the `dist/` directory:

- `mypackage-0.1.0.tar.gz`
 - **Type:** Source Distribution
 - **Description:** This is a compressed archive containing the source code of the package. It includes the Python files, `setup.py`, and any other resources needed to build the package.
 - **Purpose:** Useful for users who want to build the package from source, possibly in environments where pre-built wheels aren't available or applicable.
- `mypackage-0.1.0-py3-none-any.whl`
 - **Type:** Wheel File
 - **Description:** This is a built distribution format, commonly referred to as a "wheel". It is a binary package format that contains everything needed to install the package quickly.
 - **Purpose:** Provides a fast and efficient installation method because the package is already built and does not require compiling on the user's system. The `py3` indicates compatibility with Python 3, and `none-any` signifies that this wheel is universal, meaning it can be installed on any platform without modification.

Using Wheels to Install Python Package Locally

2. Usage

- Installing the Source Distribution:

```
bash
```

[Copy code](#)

```
pip install mypackage-0.1.0.tar.gz
```

- Installing the Wheel File:

```
bash
```

[Copy code](#)

```
pip install mypackage-0.1.0-py3-none-any.whl
```

- Installing Directly from the Source Directory:

```
bash
```

[Copy code](#)

```
pip install .
```

- **Explanation:** The command `pip install .` installs the package directly from the current directory where `setup.py` is located. This is useful during development or when testing a package before distribution.

3. Best Practices

- **Distribute Both:** Offer both the source distribution (`.tar.gz`) and the wheel file (`.whl`) to give users flexibility based on their environment.
- **Use Wheels When Possible:** Recommend the wheel file for faster installations, especially for packages with complex dependencies.

Python Package Structure for Wheel Distribution

1. Package Structure Overview

```
plaintext Copy code

mypackage/
├── mypackage/
│   ├── __init__.py      # Initializes the package
│   ├── module1.py       # Example module 1
│   └── module2.py       # Example module 2
├── tests/
│   ├── test_module1.py  # Unit tests for module 1
│   └── test_module2.py  # Unit tests for module 2
├── dist/                # Distribution files, including .whl
│   └── mypackage-0.1.0-py3-none-any.whl # The Wheel file
├── build/               # Build artifacts created by setup.py
├── mypackage.egg-info/  # Metadata about the package
├── setup.py             # Script for packaging and distribution
├── README.md            # Package description and usage
└── LICENSE               # License information
```

2. Key Components

- `mypackage/`` Directory: Contains the actual Python package with modules.
- `__init__.py``: Initializes the package and can define what is exposed to users.
- `tests/`` Directory: Contains unit tests for the package.
- `dist/`` Directory: Stores the distribution files, including the `.whl`` file after building.
- `build/`` Directory: Contains temporary build artifacts created during the package build process.
- `mypackage.egg-info/`` Directory: Contains metadata about the package, including version, dependencies, and more.
- `setup.py``: Script that defines the package details and is used to build the distribution files.
- `README.md``: Provides a detailed description of the package, including installation and usage instructions.
- `LICENSE``: Specifies the licensing terms under which the package is distributed.

Python Wheel

5. Uploading a Wheel to PyPI

- Steps:


1. Build the Wheel:

```
bash  
  
python setup.py bdist_wheel
```

 Copy code

2. Upload to PyPI using Twine:

```
bash  
  
twine upload dist/*.whl
```

 Copy code

- Explanation:

- This process uploads the wheel to PyPI, making it available for others to install.

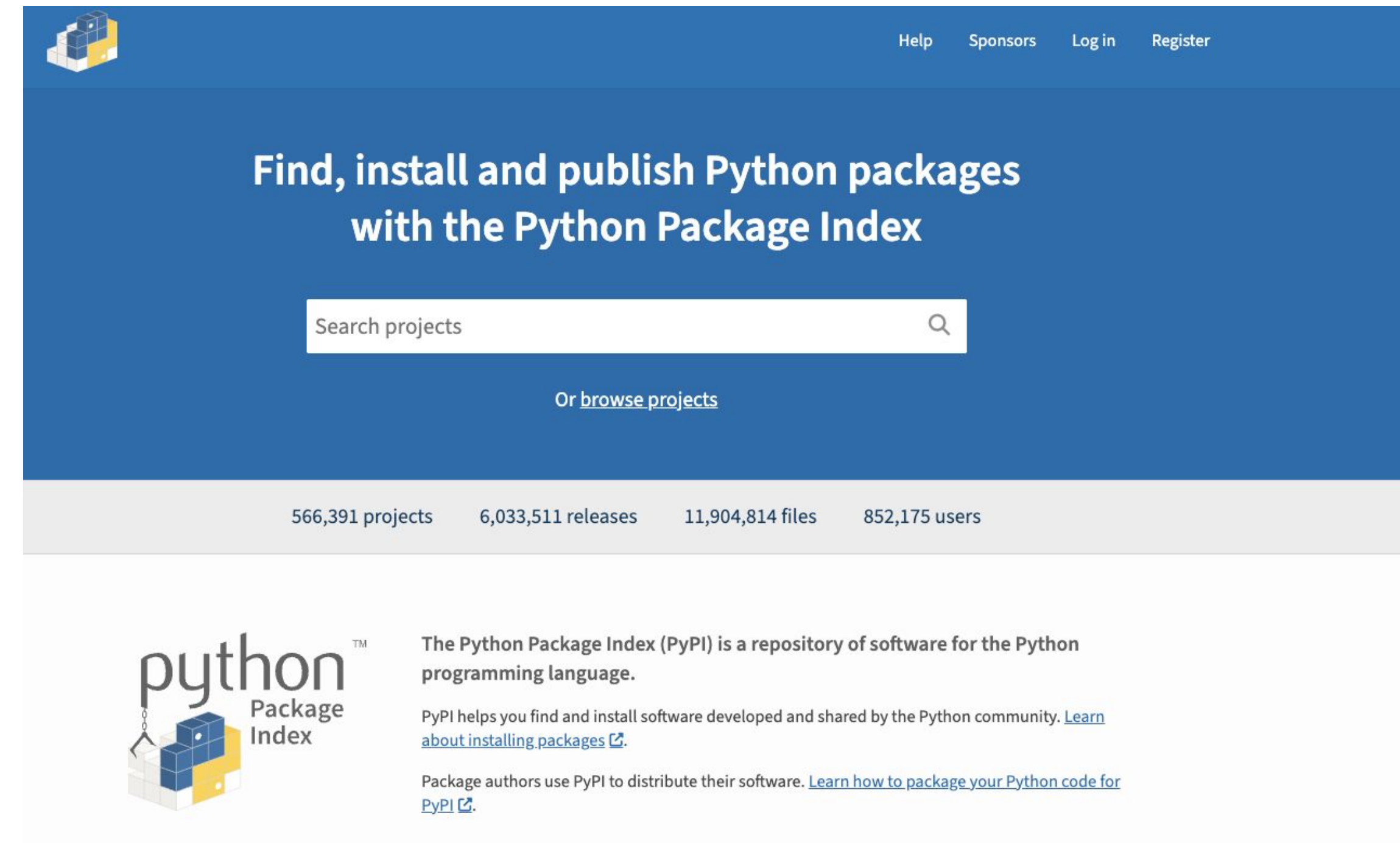
Relationship Between Distributions and PyPI

1. What is PyPI?

- **Definition:** The Python Package Index (PyPI) is the official repository where Python packages are uploaded and shared with the Python community.
- **Purpose:** Enables easy distribution, installation, and management of Python packages via `pip`.

2. Role of Distributions on PyPI

- **Source Distribution (`.tar.gz`)**
 - **Purpose:** Allows users to download and build the package from source.
 - **Usage on PyPI:** Uploaded to PyPI for environments that may require or prefer building from source, such as when specific platform optimizations or custom build steps are needed.
- **Wheel File (`.whl`)**
 - **Purpose:** Provides a pre-built, binary distribution of the package.
 - **Usage on PyPI:** Uploaded to PyPI for easy and fast installation across various environments without the need for compilation, making it the preferred distribution format for most users.



Uploading the Package to PyPI

1. Install `twine`

- Command:

```
bash
pip install twine
```

Copy code

2. Upload to PyPI

- Command:

```
bash
twine upload dist/*
```

Copy code

- Explanation:

- Uploads the package files from the `dist/` directory to PyPI.
- You'll need your PyPI credentials.

3. Test on TestPyPI (Optional)

- Command:

```
bash
twine upload --repository-url https://test.pypi.org/legacy/ dist/*
```

Copy code

- Purpose: Test the upload process before uploading to the official PyPI.

Good Practice Define a Shell Script

```
upload_package() {
    echo "Removing old distribution..."
    rm -rf dist/*
    sleep 2 # Waits for 2 seconds

    echo "Building the distribution..."
    python setup.py sdist bdist_wheel
    sleep 2 # Waits for 2 seconds

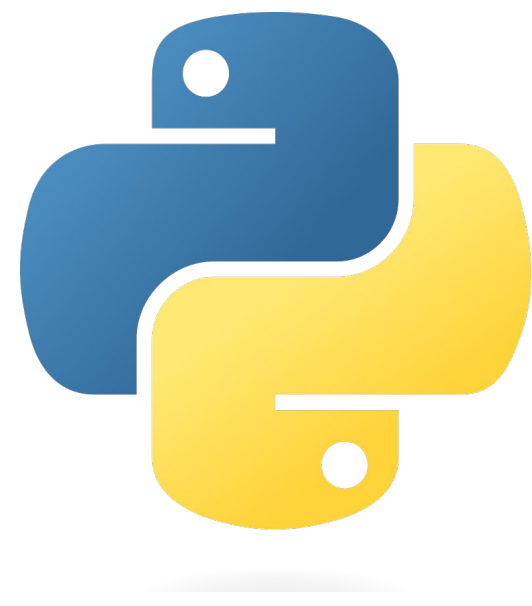
    echo "Uploading the distribution..."
    pip install twine
    sleep 1
    twine upload dist/*.whl dist/*.tar.gz --verbose
}
```

- Availability: Once uploaded, the package becomes available to anyone using `pip`:

```
bash
pip install package_name
```

Copy code

- Example: Users can install your package directly from PyPI with a simple command.



Setup a Python Package

2. GitHub Repository

Prof. Anis Koubaa



SPECIALIZATION

CS316
**INTRODUCTION TO AI AND DATA
SCIENCE**

CHAPTER 2
Python for Data Science

LECTURE 1
Introduction to Python Programming

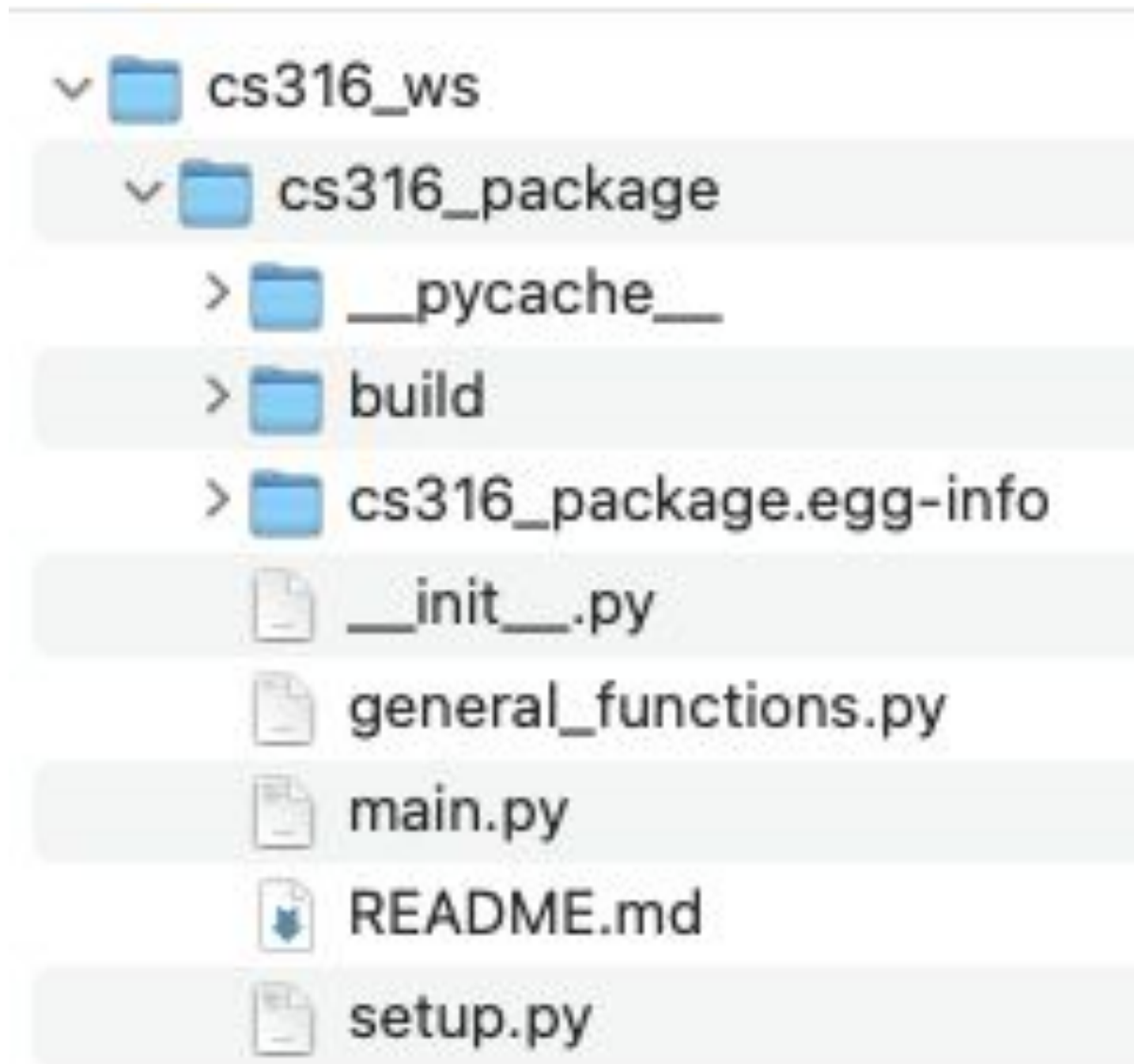
Setting Up a GitHub Account and Uploading a Python Package

1. Creating a GitHub Account

- Visit github.com.
- Click on "Sign up" and fill in your details.
- Verify your email to activate the account.
- Set up a new repository after logging in.

2. Setting Up a Python Package

- Directory Structure:
 - Root directory: ``cs316_ws``
 - Package directory: ``cs316_package/``
 - Essential files:
 - ``__init__.py``: Marks the directory as a Python package.
 - ``general_functions.py``: Contains general utility functions.
 - ``main.py``: Main execution file.
 - ``setup.py``: Used for package installation.
 - ``README.md``: Project description.



Setting Up a GitHub Account and Uploading a Python Package

3. Uploading Package to GitHub

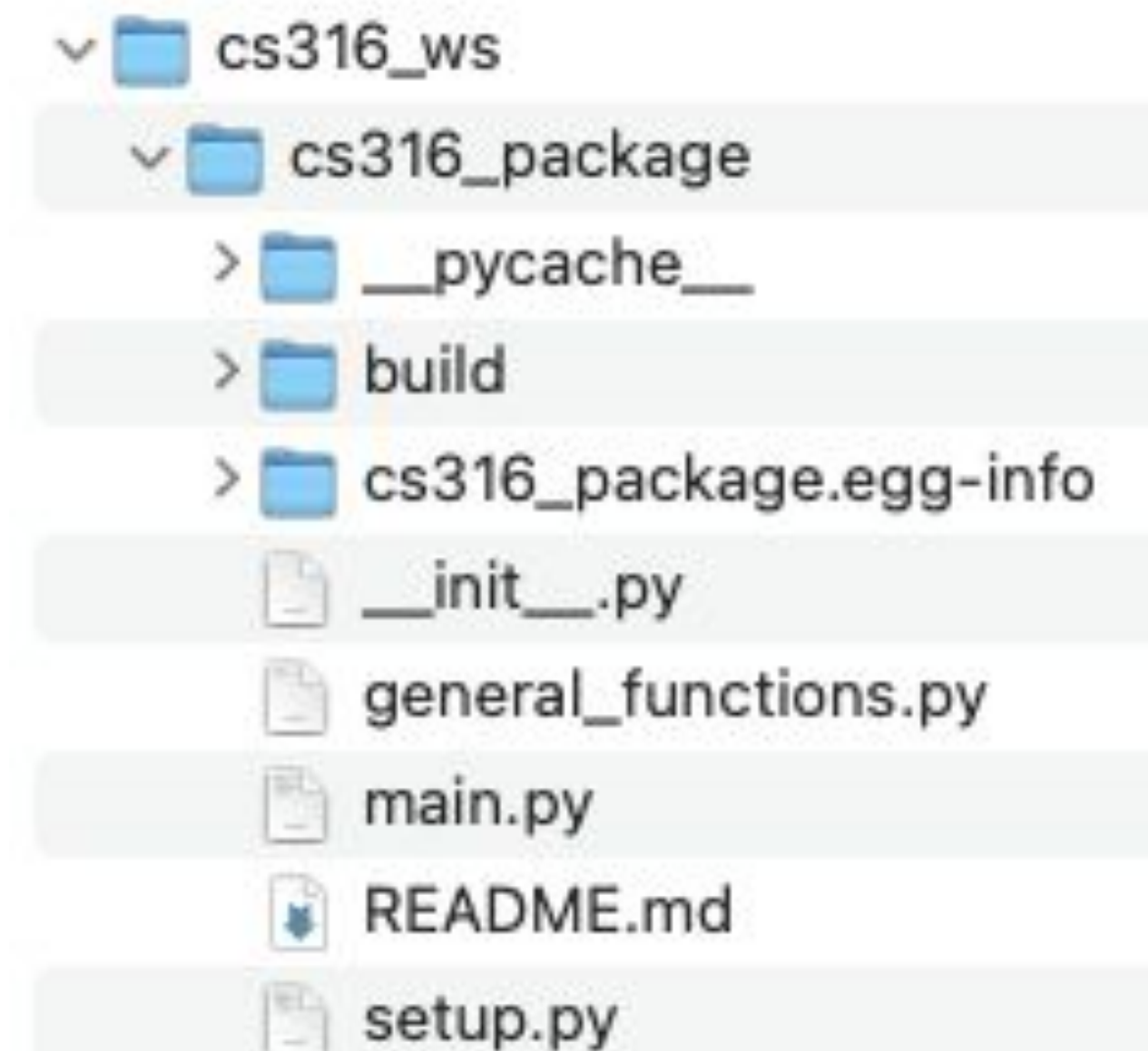
- Initialize Local Repository:
 - ``git init`` inside ``cs316_ws/``.
- Add Files:
 - ``git add .`` to stage all files.
- Commit Changes:
 - ``git commit -m "Initial commit"``
- Push to GitHub:

- Link repository:

```
bash Copy code  
  
git remote add origin https://github.com/yourusername/cs316_package.git
```

- Push changes:

```
bash Copy code  
  
git push -u origin master
```



Setting Up a GitHub Account and Uploading a Python Package

3. Uploading Package to GitHub

- Initialize Local Repository:
 - ``git init`` inside ``cs316_ws/``.
- Add Files:
 - ``git add .`` to stage all files.
- Commit Changes:
 - ``git commit -m "Initial commit"``
- Push to GitHub:
 - Link repository:

bash

Copy code

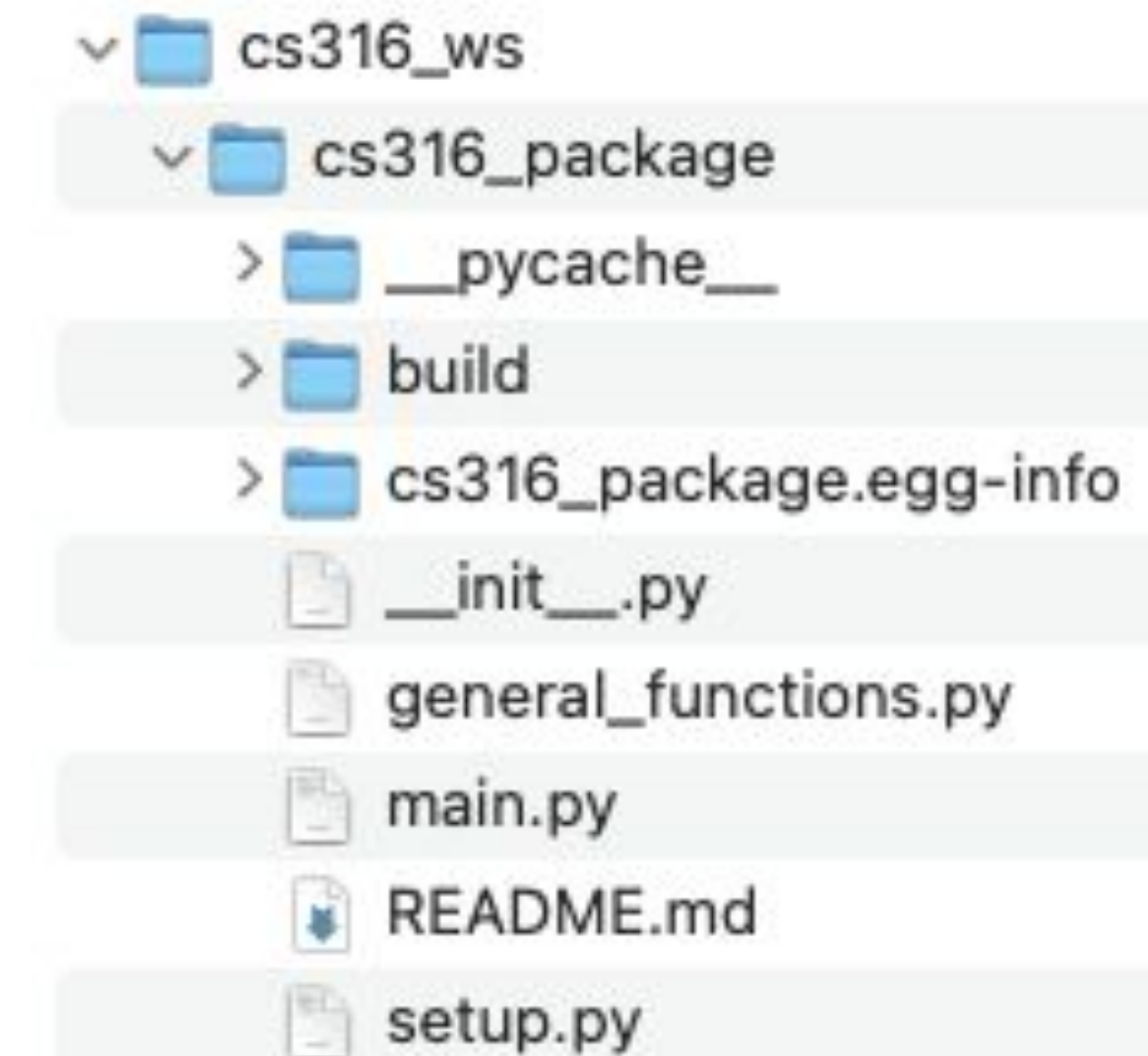
```
git remote add origin https://github.com/yourusername/cs316_package.git
```

- Push changes:

bash

Copy code

```
git push -u origin master
```



4. Verifying on GitHub

- Check the repository on GitHub to ensure all files are uploaded.
- Update ``README.md`` with project details.

CS316
INTRODUCTION TO AI AND DATA SCIENCE

CHAPTER 2
Python for Data Science

LECTURE 3
Python Programming Language
Basic Syntax

Prof. Anis Koubaa

SEP 2024

CS316

INTRODUCTION TO AI AND DATA SCIENCE

CHAPTER 2

Python for Data Science

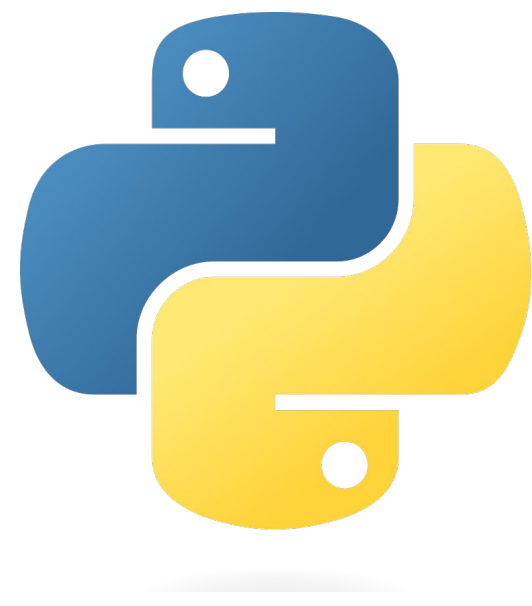
LECTURE 3 – Demo

Python Programming Language

Basic Syntax

Prof. Anis Koubaa

SEP 2024



Python Basic Syntax

Prof. Anis Koubaa



SPECIALIZATION

CS316
**INTRODUCTION TO AI AND DATA
SCIENCE**

CHAPTER 2
Python for Data Science

LECTURE 1
Introduction to Python Programming

Python Basic Syntax

1. Introduction to Python Syntax

- **Overview:** Python syntax is designed to be simple and easy to read, making code more intuitive and accessible.
- **Key Features:**
 - **Whitespace:** Indentation is used to define code blocks (e.g., loops, functions).
 - **Minimal Syntax:** No need for semicolons or curly braces.
 - **Readability:** Emphasizes clear, human-readable code.

2. Code Example

```
python Copy code  
  
# Define a function  
def greet(name):  
    print(f"Hello, {name}!")  
  
# Call the function  
greet("Alice")
```

- **Explanation:**
 - **Indentation:** The function body is indented to indicate it belongs to the `greet` function.
 - **Function Call:** The function is called with an argument (`"Alice"`), and the output is `Hello, Alice!`.

Basic Syntax Elements

1. Indentation

- **Description:** Python uses indentation to define blocks of code, unlike other languages that use braces `{}`.
- **Standard Practice:** Use four spaces per indentation level.
- **Example:**

```
python Copy code  
  
if x > 0:  
    print("Positive number")
```

- **Explanation:** The indented line belongs to the `if` statement block.

Basic Syntax Elements

2. Comments

- **Description:** Use the hash `#` symbol to write comments. Python ignores anything after `#` on a line.
- **Usage:** Comments are used for notes, documentation, and explanations within the code.
- **Example:**

```
python Copy code  
  
# This is a comment  
print("Hello, world!") # This prints a message
```

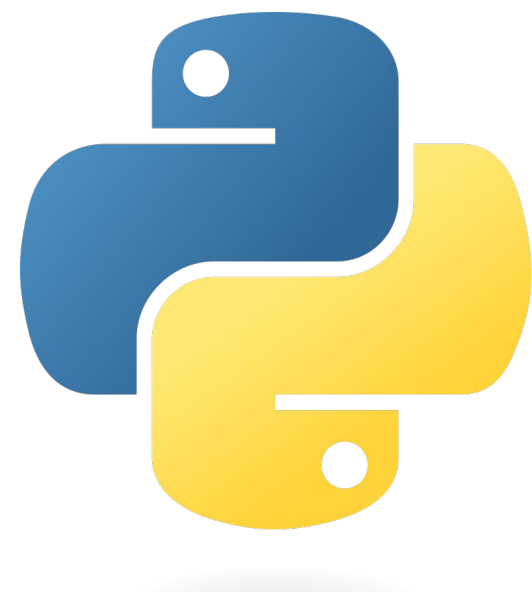
Basic Syntax Elements

3. Case Sensitivity

- **Description:** Python is case-sensitive, meaning that variables like ``myVariable``, ``MyVariable``, and ``myvariable`` are distinct.
- **Example:**

```
python Copy code  
  
myVariable = 10  
MyVariable = 20  
print(myVariable) # Outputs: 10  
print(MyVariable) # Outputs: 20
```

- **Explanation:** Different capitalization creates different variables.



Python Variables and Data Types

Prof. Anis Koubaa



SPECIALIZATION

CS316
**INTRODUCTION TO AI AND DATA
SCIENCE**

CHAPTER 2
Python for Data Science


LECTURE 1
Introduction to Python Programming

Variables and Data Types

1. Declaring Variables

- **Description:** Python automatically allocates memory when you assign a value to a variable—no need for explicit declaration.
- **Example:**

python

 Copy code

```
x = 10      # Integer
y = 3.14    # Float
name = "Alice" # String
is_valid = True # Boolean
```

- **Explanation:** The variables `x`, `y`, `name`, and `is_valid` are automatically created with their respective data types based on the assigned values.

Variables and Data Types

2. Basic Data Types

- **Integers:** Whole numbers, positive or negative, without decimals.
 - Example: `x = 10`
- **Floats:** Decimal numbers, used for more precise calculations.
 - Example: `y = 3.14`
- **Strings:** Sequences of characters, enclosed in single or double quotes.
 - Example: `name = "Alice"`
- **Booleans:** Represent logical values, either `True` or `False`.
 - Example: `is_valid = True`

Non-Basic Python Data Types

Data Type	Description	Example	Characteristics
List	Ordered, mutable collection of items	<code>`my_list = [1, 2, 3]`</code>	Dynamic size, supports mixed data types, accessed by index
Tuple	Ordered, immutable collection of items	<code>`my_tuple = (1, 2, 3)`</code>	Fixed size, supports mixed data types, accessed by index
Set	Unordered collection of unique items	<code>`my_set = {1, 2, 3}`</code>	No duplicates, unordered, supports set operations
Dictionary	Unordered collection of key-value pairs	<code>`my_dict = {'a': 1, 'b': 2}`</code>	Keys must be unique, values accessed by keys
Complex	Numbers with real and imaginary parts	<code>`z = 3 + 4j`</code>	Precision depends on float representation for real/imaginary parts
Bytes	Immutable sequence of bytes	<code>`my_bytes = b'hello'`</code>	Used for binary data, immutable
Bytearray	Mutable sequence of bytes	<code>`my_bytearray = bytearray(b'hello')`</code>	Mutable version of <code>`bytes`</code> , can be modified in place
NoneType	Represents the absence of a value	<code>`value = None`</code>	Singleton, only one instance (<code>`None`</code>) exists

Finding the Type of a Variable with `type()`

1. Using `type()` Function

- Purpose: `type()` is a built-in Python function used to determine the type of a variable.

2. Example Code

```
python Copy code

# Define variables of different types
x = 10          # Integer
y = 3.14        # Float
name = "Alice"  # String
is_valid = True # Boolean

# Use type() to find the type of each variable
print(type(x))    # Output: <class 'int'>
print(type(y))    # Output: <class 'float'>
print(type(name)) # Output: <class 'str'>
print(type(is_valid)) # Output: <class 'bool'>
```

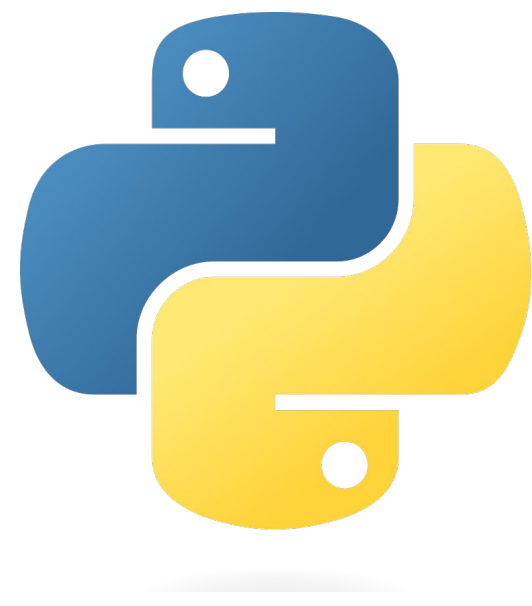
3. Explanation

- The `type()` function returns the type of the variable passed as an argument.
- The output shows the class of the variable (`'int'`, `'float'`, `'str'`, `'bool'`), which corresponds to its data type.

Finding the Type of a Variable with type()

```
1 #String variable
2 print ("--- STRING VARIABLES --- ")
3 name = "Mohamed"
4 print ("student name: ", name)
5 print ("the type of the variable name is: ", type(name))
6
7
8 #Boolean variable
9 print ("--- BOOLEAN VARIABLES --- ")
10 flag = True
11 print ("Flag: ", flag)
12 print ("the type of the variable flag is: ", type(flag))
13
14 #Hexadecimal Number
15 print ("--- HEXADECEMAL VARIABLES --- ")
16 w = 0xfa2
17 v=hex(547)
18 print ("w: ", w)
19 print ("v: ", v)
20 print ("the type of the variable w is: ", type(w))
21 print ("the type of the variable v is: ", type(v))
```

```
--- STRING VARIABLES ---
student name: Mohamed
the type of the variable name is: <class 'str'>
--- BOOLEAN VARIABLES ---
Flag: True
the type of the variable flag is: <class 'bool'>
--- HEXADECEMAL VARIABLES ---
w: 4002
v: 0x223
the type of the variable w is: <class 'int'>
the type of the variable v is: <class 'str'>
```

Type Conversion in Python

Prof. Anis Koubaa



SPECIALIZATION

CS316
**INTRODUCTION TO AI AND DATA
SCIENCE**

CHAPTER 2
Python for Data Science


LECTURE 1
Introduction to Python Programming

Implicit Type Conversion

1. Implicit Type Conversion

- **Description:** Python automatically converts one data type to another when necessary.
- **Example:**

python

 Copy code

```
x = 5      # Integer
y = 3.2    # Float
z = x + y  # Implicitly converts x to float
print(z)   # Output: 8.2 (float)
```

- **Explanation:** In the above example, the integer `x` is implicitly converted to a float during the addition.

Explicit Type Conversion

2. Explicit Type Conversion (Type Casting)

- **Description:** Converting one data type to another manually using built-in functions.
- **Common Functions:**
 - `int(x)`: Converts `x` to an integer.
 - `float(x)`: Converts `x` to a float.
 - `str(x)`: Converts `x` to a string.
 - `bool(x)`: Converts `x` to a boolean.

4. Considerations

- **Loss of Data:** Converting from a float to an integer truncates the decimal part.
- **Errors:** Converting incompatible types (e.g., `int("abc")`) will raise a `ValueError`.

3. Examples of Explicit Type Conversion

- Integer to String:

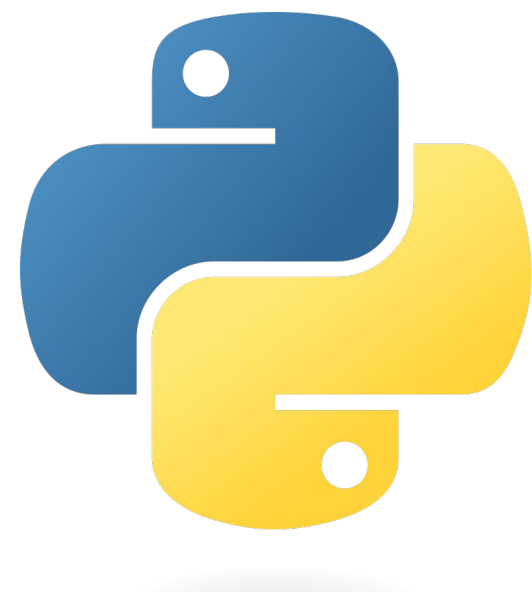
```
python
num = 10
num_str = str(num)
print(num_str) # Output: "10" (string)
```

- String to Integer:

```
python
s = "123"
num = int(s)
print(num) # Output: 123 (integer)
```

- Float to Integer:

```
python
pi = 3.14
int_pi = int(pi)
print(int_pi) # Output: 3 (integer, truncated)
```



Python Basic Operators

Prof. Anis Koubaa



SPECIALIZATION

CS316
**INTRODUCTION TO AI AND DATA
SCIENCE**

CHAPTER 2
Python for Data Science

LECTURE 1
Introduction to Python Programming

Basic Operators in Python

Operator Type	Operator	Description	Example	Result
Arithmetic Operators	<code>`+`</code>	Addition	<code>`5 + 3`</code>	<code>`8`</code>
	<code>`-`</code>	Subtraction	<code>`5 - 3`</code>	<code>`2`</code>
	<code>`*`</code>	Multiplication	<code>`5 * 3`</code>	<code>`15`</code>
	<code>`/`</code>	Division (returns a float)	<code>`5 / 2`</code>	<code>`2.5`</code>
	<code>`%`</code>	Modulus (remainder of division)	<code>`5 % 2`</code>	<code>`1`</code>
	<code>`**`</code>	Exponentiation (power)	<code>`5 ** 3`</code>	<code>`125`</code>
Comparison Operators	<code>`==`</code>	Equal to	<code>`5 == 3`</code>	<code>`False`</code>
	<code>`!=`</code>	Not equal to	<code>`5 != 3`</code>	<code>`True`</code>
	<code>`>`</code>	Greater than	<code>`5 > 3`</code>	<code>`True`</code>
	<code>`<`</code>	Less than	<code>`5 < 3`</code>	<code>`False`</code>
	<code>`>=`</code>	Greater than or equal to	<code>`5 >= 3`</code>	<code>`True`</code>
	<code>`<=`</code>	Less than or equal to	<code>`5 <= 3`</code>	<code>`False`</code>

Basic Operators in Python

```
1  a = 2
2  b = 3
3
4  add = a+b
5  sub = a-b
6  mult = a*b
7  div = a/b
8
9  mod = a%b
10 e = a ** b
11
12 print ("a = ",a)
13 print ("b = ",b)
14 print ("add = a+b = ",add)
15 print ("sub = a-b = ",sub)
16 print ("mult = a*b = ",mult)
17 print ("div = a/b = ",div)
18 print ("mod = a%b = ",mod)
19 print ("e = a**b = ",e)
20
21
```

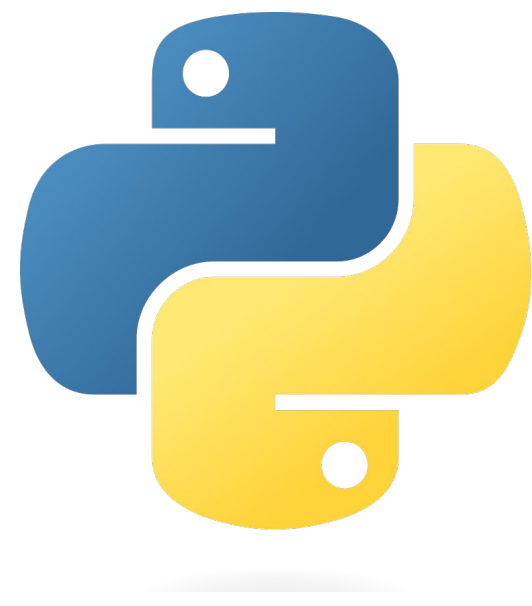
```
⇒ a = 2
   b = 3
   add = a+b = 5
   sub = a-b = -1
   mult = a*b = 6
   div = a/b = 0.6666666666666666
   mod = a%b = 2
   e = a**b = 8
```

Typing Constraints

- **Type Compatibility:** Python operators generally require operands to be of compatible types.

For example:

- **Arithmetic Operators:** Operands must typically be of numeric types (`int`, `float`), though some operators like `+` can also concatenate strings.
 - Example: `"Hello" + " World"` results in `"Hello World"`
- **Comparison Operators:** Can compare numbers, strings, and other comparable types, but comparing incompatible types (e.g., `int` and `str`) will raise a `TypeError`.
 - Example: `5 > "3"` will result in a `TypeError`
- **Implicit Type Conversion:** Python may perform implicit type conversion (e.g., `int` to `float`) when necessary, but explicit conversion using functions like `int()`, `float()`, or `str()` is often required to avoid errors.



Simple Input/Output in Python

Prof. Anis Koubaa



SPECIALIZATION

CS316
**INTRODUCTION TO AI AND DATA
SCIENCE**

CHAPTER 2
Python for Data Science


LECTURE 1
Introduction to Python Programming

Simple Input/Output in Python

1. Input: `input()` Function

- **Purpose:** Used to get user input from the keyboard.
- **Syntax:** `variable = input("Prompt message")`
- **Example:**

python

 Copy code

```
name = input("Enter your name: ")
```

- **Explanation:** The input provided by the user is stored as a string in the variable `name`.

Simple Input/Output in Python

2. Output: `print()` Function

- Purpose: Used to display output to the screen.
- Syntax: `print(value1, value2, ..., sep=' ', end='\n')`
- Example:


```
python Copy code  
  
print("Hello, World!")  
print("Welcome,", name)
```

- Explanation:
 - The first `print()` displays the message `"Hello, World!"`.
 - The second `print()` uses the variable `name` to display a personalized welcome message.

Simple Input/Output in Python

3. Combined Example

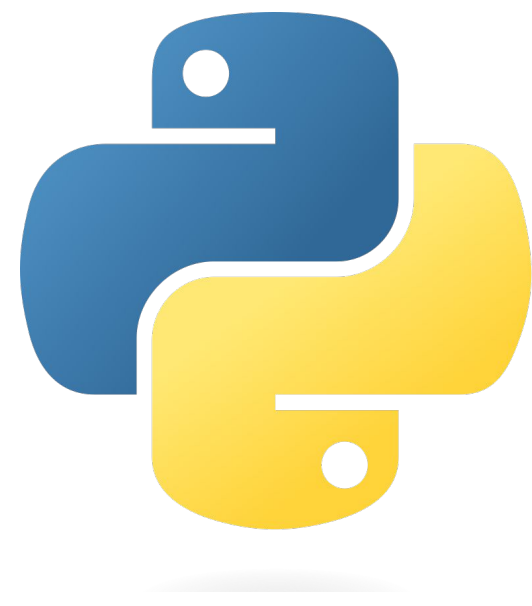
python

 Copy code

```
# Getting user input
age = input("Enter your age: ")

# Displaying the input
print("You are", age, "years old.")
```

- **Explanation:** This code prompts the user for their age and then displays it back to them in a complete sentence.



Formatted Output in Python

Prof. Anis Koubaa



SPECIALIZATION

CS316
**INTRODUCTION TO AI AND DATA
SCIENCE**

CHAPTER 2
Python for Data Science

LECTURE 1
Introduction to Python Programming


Formatted Output in Python with print()

1. Basic `print()`

- **Description:** Displays one or more values separated by a space (default).
- **Example:**

”

python

 Copy code

```
print("Hello", "World!")
```

- **Output:**
`Hello World`

2. Using `sep` Parameter

- **Description:** Customizes the separator between multiple values.
- **Example:**

python

 Copy code

```
print("Hello", "World", sep="-")
```


- **Output:**
`Hello-World`

Formatted Output in Python with print()

3. Using `end` Parameter

- **Description:** Customizes what is printed at the end of the output (default is a newline ``\n``).
- **Example:**

python

 Copy code

```
print("Hello", end=", ")  
print("World!")
```


- **Output:**
`Hello, World!`

Formatted Output in Python with print()

4. String Formatting with `%` Operator

- Description: Uses placeholders (`%s`, `%d`, `%f`) to embed values in a string.
- Example:

python

 Copy code

```
name = "Alice"  
age = 30  
print("My name is %s and I am %d years old." % (name, age))
```


- Output:
`My name is Alice and I am 30 years old.`

Formatted Output in Python with print()

5. `str.format()` Method

- **Description:** A more versatile way to format strings, using curly braces `{}` as placeholders.
- **Example:**

python

 Copy code

```
name = "Bob"  
score = 95.5  
print("Name: {}, Score: {:.1f}".format(name, score))
```

- **Output:**
``Name: Bob, Score: 95.5``

Formatted Output in Python with print()

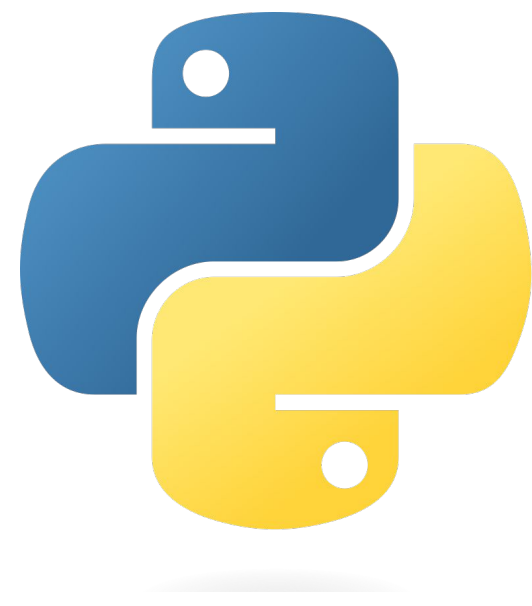
6. f-Strings (Python 3.6+)

- **Description:** An easy and efficient way to embed expressions inside string literals using `{}` with an `f` prefix.

- **Example:**

```
python Copy code  
  
name = "Charlie"  
height = 1.75  
print(f"{name} is {height} meters tall.")
```

- **Output:**
`Charlie is 1.75 meters tall.`



Math Functions in Python

Prof. Anis Koubaa



SPECIALIZATION

CS316
**INTRODUCTION TO AI AND DATA
SCIENCE**

CHAPTER 2
Python for Data Science

LECTURE 1
Introduction to Python Programming

Common Mathematical Functions in Python

Function	Description	Example	Result
<code>`abs(x)`</code>	Returns the absolute value of <code>`x`</code>	<code>`abs(-5)`</code>	<code>`5`</code>
<code>`round(x, n)`</code>	Rounds <code>`x`</code> to <code>`n`</code> decimal places	<code>`round(3.14159, 2)`</code>	<code>`3.14`</code>
<code>`max(x1, x2, ...)`</code>	Returns the largest value among arguments	<code>`max(3, 7, 1)`</code>	<code>`7`</code>
<code>`min(x1, x2, ...)`</code>	Returns the smallest value among arguments	<code>`min(3, 7, 1)`</code>	<code>`1`</code>
<code>`pow(x, y)`</code>	Returns <code>`x`</code> raised to the power of <code>`y`</code>	<code>`pow(2, 3)`</code>	<code>`8`</code>
<code>`sum(iterable)`</code>	Returns the sum of all items in an iterable	<code>`sum([1, 2, 3, 4])`</code>	<code>`10`</code>
<code>`math.sqrt(x)`</code>	Returns the square root of <code>`x`</code>	<code>`math.sqrt(16)`</code>	<code>`4.0`</code>
<code>`math.exp(x)`</code>	Returns <code>`e`</code> raised to the power of <code>`x`</code>	<code>`math.exp(1)`</code>	<code>`2.718281828459045`</code>
<code>`math.log(x, base)`</code>	Returns the logarithm of <code>`x`</code> to the specified base (default is <code>`e`</code>)	<code>`math.log(8, 2)`</code>	<code>`3.0`</code>
<code>`math.sin(x)`</code>	Returns the sine of <code>`x`</code> (in radians)	<code>`math.sin(math.pi / 2)`</code>	<code>`1.0`</code>
<code>`math.cos(x)`</code>	Returns the cosine of <code>`x`</code> (in radians)	<code>`math.cos(0)`</code>	<code>`1.0`</code>

Basic Operators in Python

Operator Type	Operator	Description	Example	Result
Arithmetic Operators	<code>`+`</code>	Addition	<code>`5 + 3`</code>	<code>`8`</code>
	<code>`-`</code>	Subtraction	<code>`5 - 3`</code>	<code>`2`</code>
	<code>`*`</code>	Multiplication	<code>`5 * 3`</code>	<code>`15`</code>
	<code>`/`</code>	Division (returns a float)	<code>`5 / 2`</code>	<code>`2.5`</code>
	<code>`%`</code>	Modulus (remainder of division)	<code>`5 % 2`</code>	<code>`1`</code>
	<code>`**`</code>	Exponentiation (power)	<code>`5 ** 3`</code>	<code>`125`</code>
Comparison Operators	<code>`==`</code>	Equal to	<code>`5 == 3`</code>	<code>`False`</code>
	<code>`!=`</code>	Not equal to	<code>`5 != 3`</code>	<code>`True`</code>
	<code>`>`</code>	Greater than	<code>`5 > 3`</code>	<code>`True`</code>
	<code>`<`</code>	Less than	<code>`5 < 3`</code>	<code>`False`</code>
	<code>`>=`</code>	Greater than or equal to	<code>`5 >= 3`</code>	<code>`True`</code>
	<code>`<=`</code>	Less than or equal to	<code>`5 <= 3`</code>	<code>`False`</code>

Common Mathematical Functions in Python

```
1 x = -2.56
2 y = 3.54
3 n = 3
4
5 from math import *
6
7
8 print ('the absolute value of x is : ', abs(x)) #we can use abs or fabs, math.fabs
9 print ('-----')
10 print ('the ceiling of x is : ', ceil(x))
11 print ('the floor of x is : ', floor(x))
12 print ('the ceiling of y is : ', ceil(y))
13 print ('the floor of y is : ', floor(y))
14 print ('-----')
15 print ('the log of y is : ', log(y)) #log only applies for positive values
16 print ('the log10 of x is : ', log10(y)) #log10 only applies for positive values
17 print ('the exponential of y is : ', exp(y))
18 print ('the square root of y is : ', sqrt(y))
19 print ('the power of x by y is : ', pow(x,n))
```

➡ the absolute value of x is : 2.56

the ceiling of x is : -2
the floor of x is : -3
the ceiling of y is : 4
the floor of y is : 3

the log of y is : 1.264126727145683
the log10 of x is : 0.5490032620257879
the exponential of y is : 34.46691919085739
the square root of y is : 1.8814887722226779
the power of x by y is : -16.777216000000003

END OF LECTURE