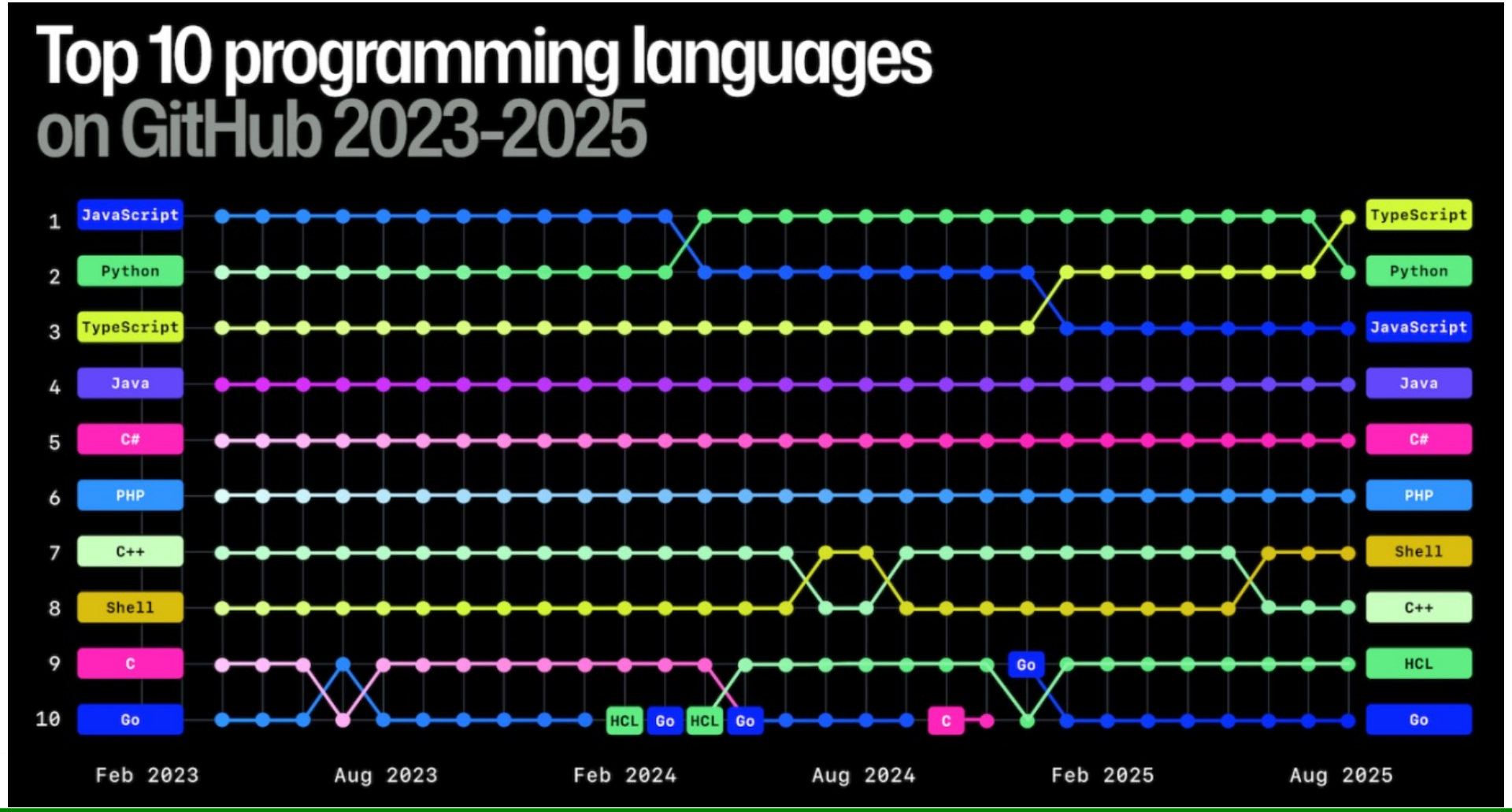
Computer Basics

1405 Instructor: Ruiqing (Sam) Cao

Python: #1-2 Among Global Developers



Source: GitHub Octoverse

Python Job Categories

Career Path	Level 1: Beginner	Level 2: Intermediate	Level 3: Advanced
Python Developer	• Basic syntax • Simple data structures	• Object-oriented programming • APIs	• Advanced data structures • Multithreading
Data Analyst	• Data manipulation with Pandas • Matplotlib	• Data cleaning techniques • Seaborn	• Statistical analysis with SciPy • Time series
Software Architect	• System design basics	• Component integration • Middleware understanding	• System security and database management

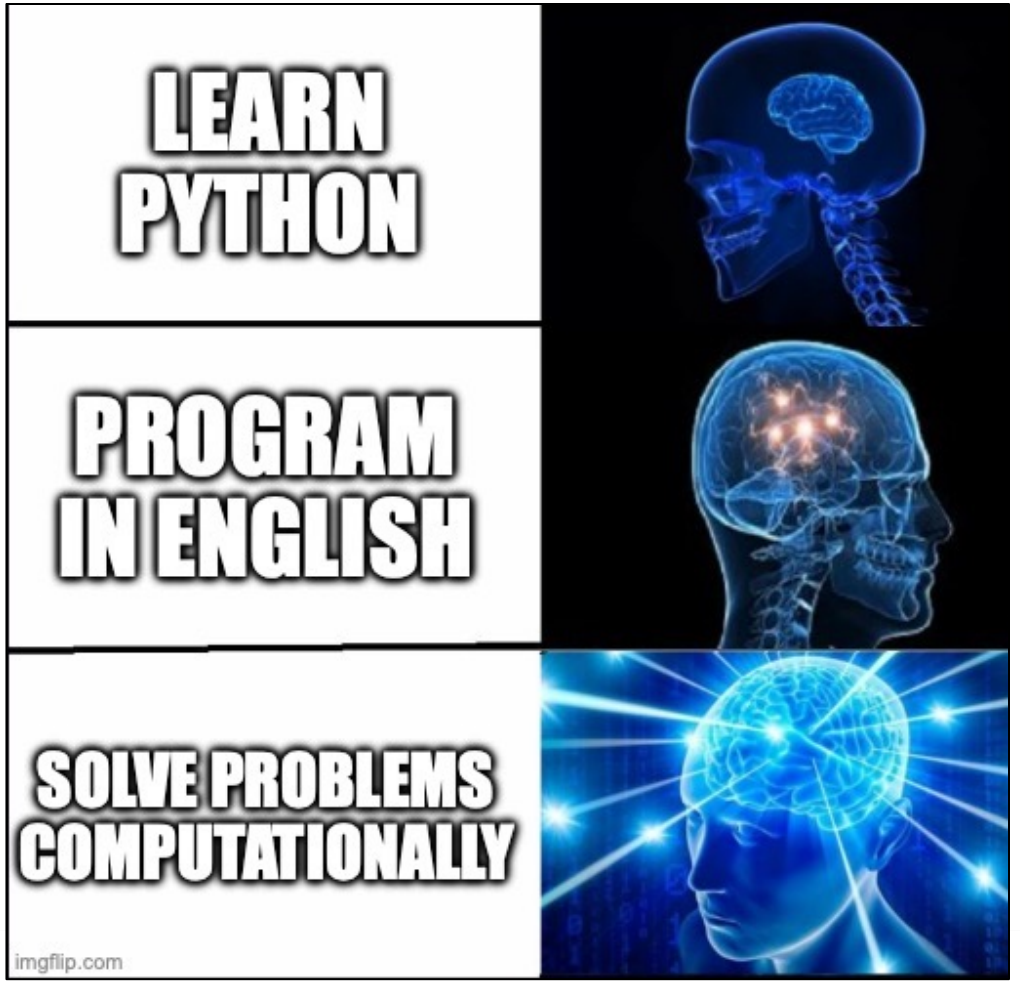
Source (Coursera): https://www.coursera.org/resources/job-leveling-matrix-for-python-career-pathways

Learning Python: Not Just the Language

Python is a tool – just like GenAI is a tool – that complements humans' capability to think computationally (and creatively)

Computational Thinking + Python

Computational Thinking + GenAI + Verify and Modify Output



Computer Basics: Storage

Storage

Access and retain data

computer memory, data storage, bits & bytes, RAM, hard disk drive, etc.

→ Enable computers to perform computation and automate tasks at scale

Compute

Process data and execute instructions

Central processing unit (CPU), graphics processing unit (GPU), clock speed, programming languages, etc.

Memory & Data Storage

Binary: two states of a switch (1-*on* 0-*off*)

- In the computer, a variety of **data objects** – numbers (integer and real), characters (and strings), and more complex objects – are encoded as a series of *binary digits* called **bits**

- Computer stores information in **bytes**: 1 byte = 8 bits

- Storage units in computer devices: 1KB=1000 bytes, 1MB=1000KB, 1GB=1000MB, 1TB=1000 GB (for technical measurement, binary base is used for the conversion 2^{10}=1024 instead of 1000)

Memory & Data Storage

Memory: Storage systems that hold data and program instructions, and provide access to them for the processor to execute tasks

- **Random-access memory (RAM)** and Cache memory

  → Temporary: short-term, fast, and small

- **Permanent Storage**: e.g., hard disk drives, USB drives

  → Persistent: long-term, slow, and large

Computer Basics: Compute

Storage

Access and retain data

computer memory, data storage, bits & bytes, RAM, hard disk drive, etc.

→ Enable computers to perform computation and automate tasks at scale

Compute

Process data and execute instructions

Central processing unit (CPU), graphics processing unit (GPU), clock speed, programming languages, etc.

Programming Languages

- Computer programs: instructions for the computer to perform tasks and written in a ***programming language***

- High-level language: Python (also C, C++, Java, FORTRAN, C#, etc.)

→ Close enough to English and relatively easy to learn and write

- As a result of GPT & agentic AI, we now have "*Natural language (e.g., English)* as a programming language"

Programming Languages

"English as a programming language" (GenAI)

High-Level Language *(e.g., Python)*

Low-Level (Assembly) Language

Machine Language

Hardware

Programming Languages

"English as a programming language" (GenAI)

High-Level Language *(e.g., Python)*

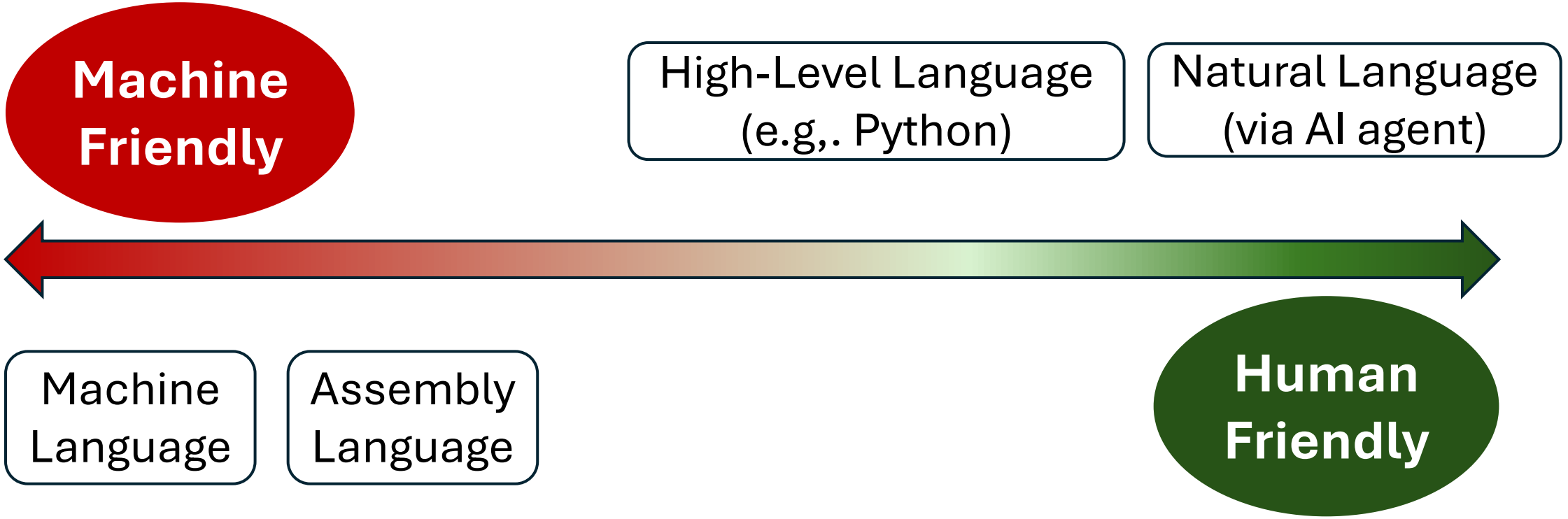
Low-Level (Assembly) Language

Machine Language

Hardware

Driven by more powerful and cheaper compute

Programming Languages



Source Code & Interpreter vs. Compiler

- **Source Code**: computer program written in a high-level language (e.g., Python)

- Computer does not understand source code, and it needs an **interpreter** or a **compiler** to *translate the source code into machine language* to execute the program

- An *interpreter* reads an argument and executes it right away; a *compiler* translates the entire source code, then executes the "compiled" machine-code file at once

Where to Write & Run a Python Program

We use an Integrated Development Environment (IDE) to run Python

- **Jupyter Notebook**: Runs Python code using the IPython kernel as the backend Python interpreter
- **Visual Studio Code** (VS Code): Calls the external system-installed Python interpreter

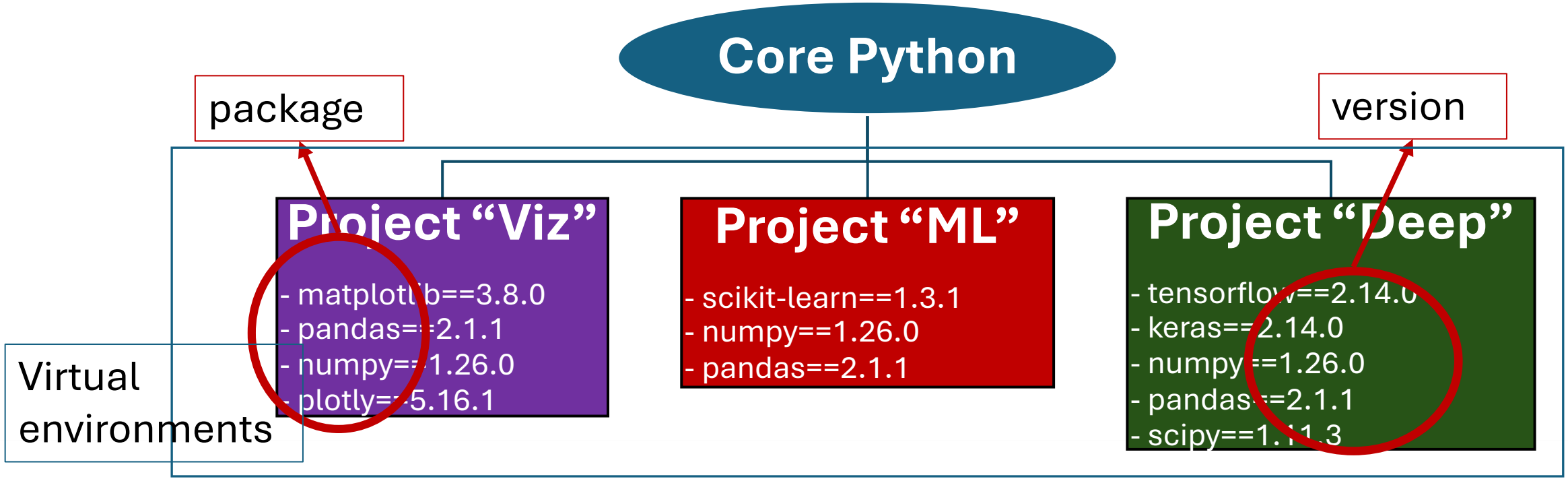
Recommendation: choose what works for you and dare to experiment

➢Jupyter Notebook is a safe choice for beginners, but VS Code is better for more serious projects and production-grade code

➢Jupyter Notebook is more interactive and user-friendly, but VS Code has version control integration (GitHub) and now *free* AI with Copilot

Virtual Environment (venv)

Virtual environment: A separate environment with its own Python installation and packages to run a project in a clean, isolated setup



Virtual Environment (venv)

Advantages of Using Virtual Environments for Projects

- **Avoid Dependency Conflicts**: Prevent issues caused by incompatible package versions or changes in one project affecting others

- **Simplify Collaboration**: Use a requirements.txt file to list and share necessary packages, making it easier for others to reproduce your results

Exercise: Choose an IDE to open

Jupyter Notebook

PC or *MacBook*:

- Open command line or terminal and type "`Jupyter Notebook`", then press Enter
- The Jupyter Notebook interface will open in your web browser automatically
- If it doesn't open, copy the URL shown in the terminal (e.g., http://localhost:8888) and paste it into your browser

Visual Studio Code

MacBook:

- Open Visual Studio Code from Applications or press Command + Space, type "Visual Studio Code," and select it from Spotlight Search

PC:

- Search for "Visual Studio Code" in the Start menu or double-click the desktop shortcut to open it.

Exercise: Anatomy of a Code Snippet

- Open a new file (choose .ipynb) in either Jupyter Notebook or VS Code

- Type the code below, click the Run button “▷”, and review the output displayed beneath the code

```
# Print "Hello world!" on the screen
st = "Hello world!"
print(st)
```

Variable & The Value Stored In It

- A **variable** (`st`) is a named container storing a value that can be referenced by calling its name

- A **value** (`"Hello world!"`) is the data stored in a variable and can be of different types, e.g., number, string, other objects

```
# Print "Hello world!" on the screen
st = "Hello world!"
print(st)
```

Variable & Value Stored in it

The glass (variable) holds **orange juice** (value)

The same glass (variable) holds **black coffee** (value)

```
glass = "orange juice"
print(glass)
```

Output: `orange juice`

```
glass = "black coffee"
print(glass)
```

Output: `black coffee`

The same `print(glass)` prints different outputs because the value stored in `glass` has changed !

Comments

- The highlighted **comment** explains what the next few lines of code aims to do: *print "Hello world!" on the screen*

- After a pound sign (#), anything on the same line is considered a comment and will be ignored by the interpreter (hence not run)

```
# Print "Hello world!" on the screen
st = "Hello world!"
print(st)
```

Operator, Expression, & Statement

- An operator is a special symbol that acts on variables and values to perform a particular operation
- A statement is a sequence of actions to perform a particular task
- An expression is part of a statement and produces a value

```
# Print "Hello world!" on the screen
st = "Hello world!"
print(st)
```

Function & Method

- Round brackets ( ) are used to call a *function* or a *method*, and the content inside ( ) are *arguments* passed to the function or method

```
# Print "Hello world!" on the screen
st = "Hello world!”
print(st)
```