



Introduction to Python Programming



Overview

- **Introduction to Python**

- ☐ Understanding Python and its applications.

- **Key Features of Python Programming**

- ☐ Exploring Python's simplicity, readability, and versatility.

- **Setting Up the Python Environment**

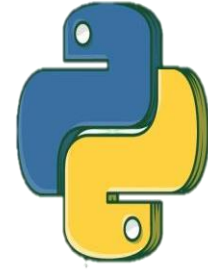
- ☐ Installing Python and setting up your programming environment.

- **Basic Python Syntax**

- ☐ Learning the fundamental syntax and structure of Python code.

- **Writing Your First Python Program**

- ☐ Hands-on experience: Creating and running your first Python script.



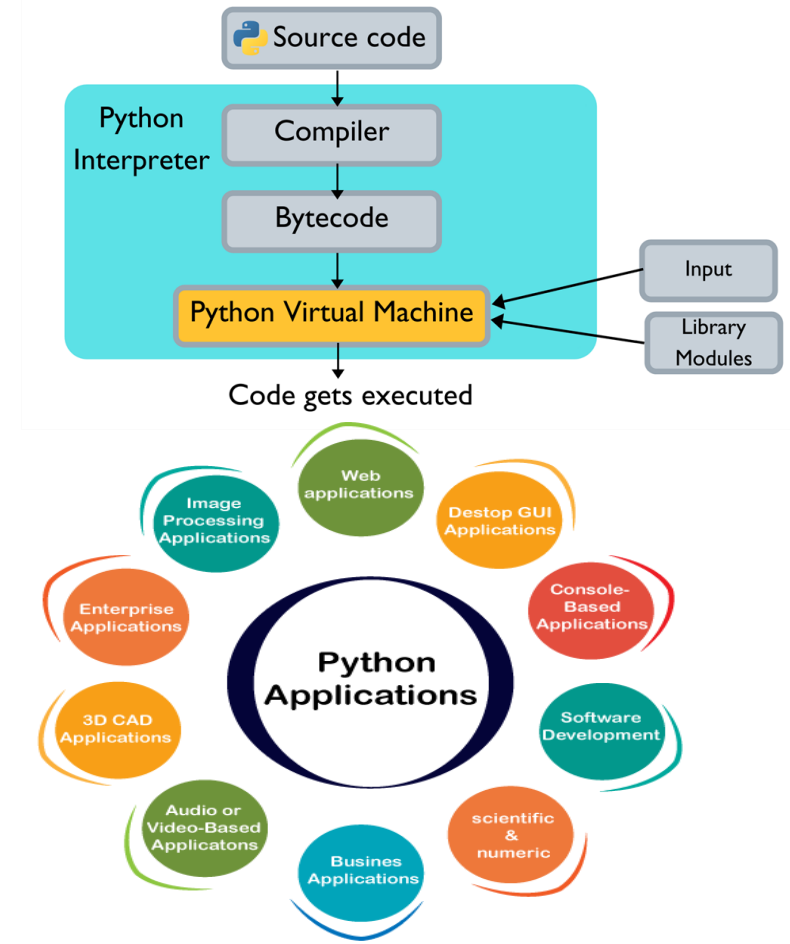
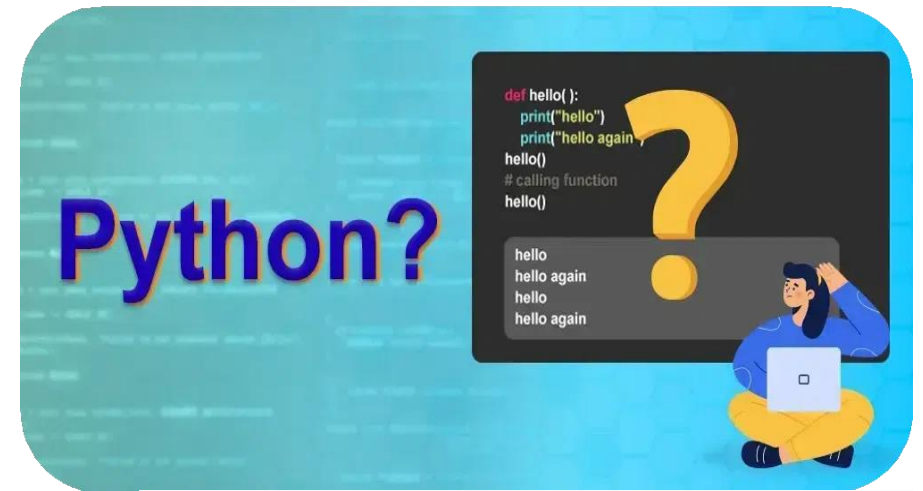
Introduction to Python

• What is Python?

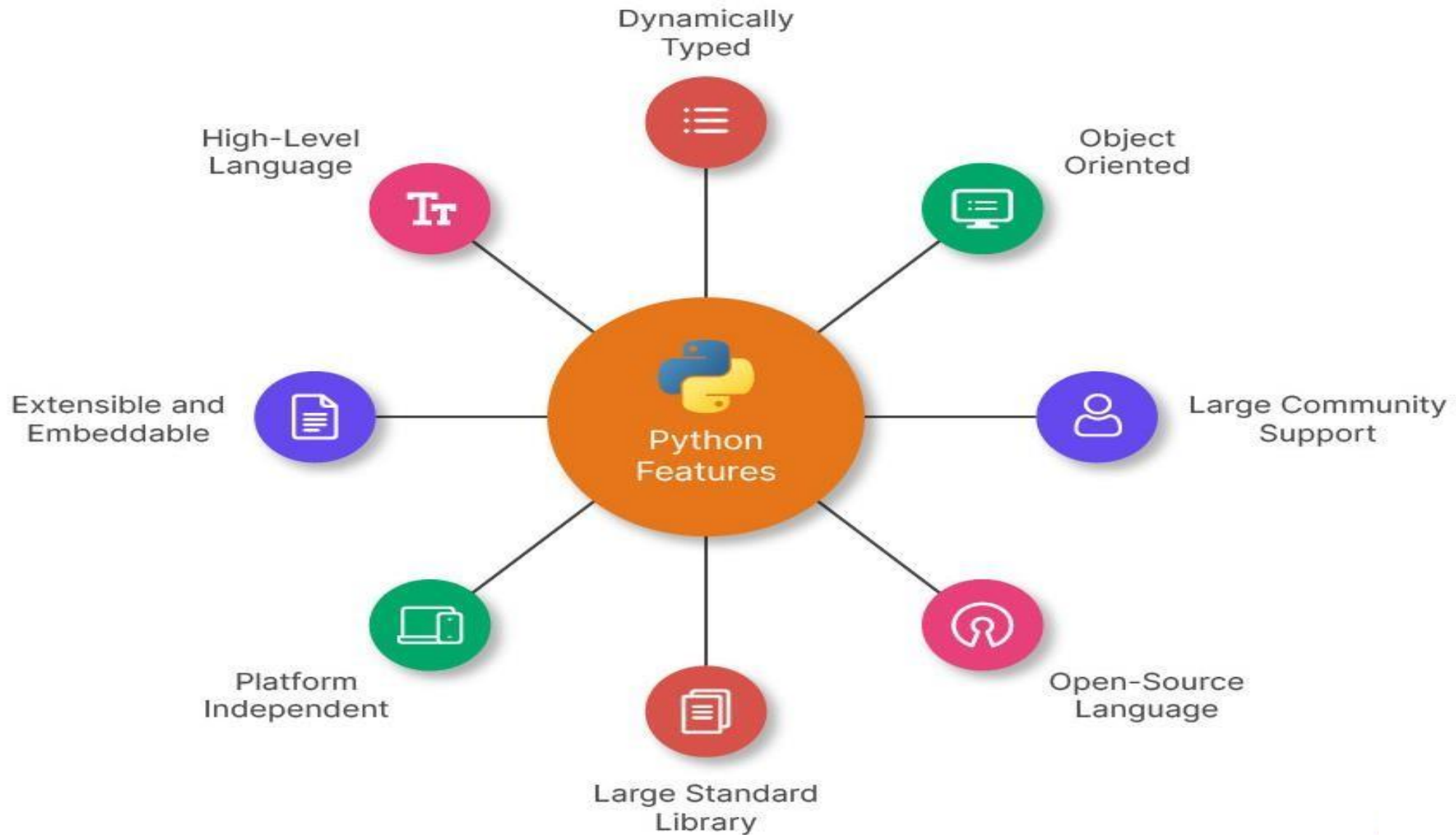
- High-level, interpreted programming language.
- Used in web development, data analysis, automation, etc.

• Why Python?

- Easy to learn with simple syntax.
- Large community and extensive libraries.
- Versatile and widely adopted across industries.



Key Features of Python Programming



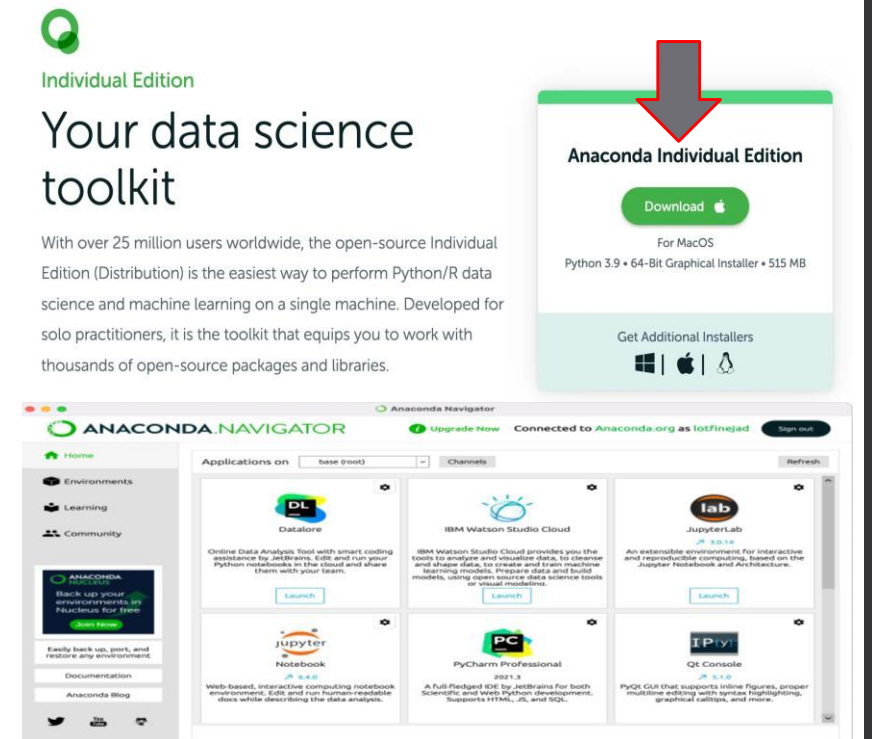
Setting Up the Python Environment

Step 1: Install Python via Anaconda Distribution

- Download the Anaconda distribution from the [official website](#).
- Follow the installation instructions for your operating system.
 - ❑ **Windows/Mac/Linux:** Install the setup file and proceed through the default installation options.
- Anaconda includes Python, Jupyter Notebook, and essential libraries, making it easier for you to start coding right away.

Step 2: Set Up Jupyter Notebook

- **Open Anaconda Navigator:** Once installed, launch the Anaconda Navigator from your system.
- **Launch Jupyter Notebook:**
 - ❑ In the Anaconda Navigator window, find and click on **Jupyter Notebook** to launch it.
- **Create a New Notebook:**
 - ❑ Jupyter Notebook will open in your web browser.
 - ❑ Click on **New** in the top-right corner and select **Python 3**.
 - ❑ A new notebook will open where you can start writing and running Python code.
- **Start Coding:**
 - ❑ In the new notebook, type your Python code in the cells and press **Shift + Enter** to execute.



Using Visual Studio Code for Jupyter Notebook

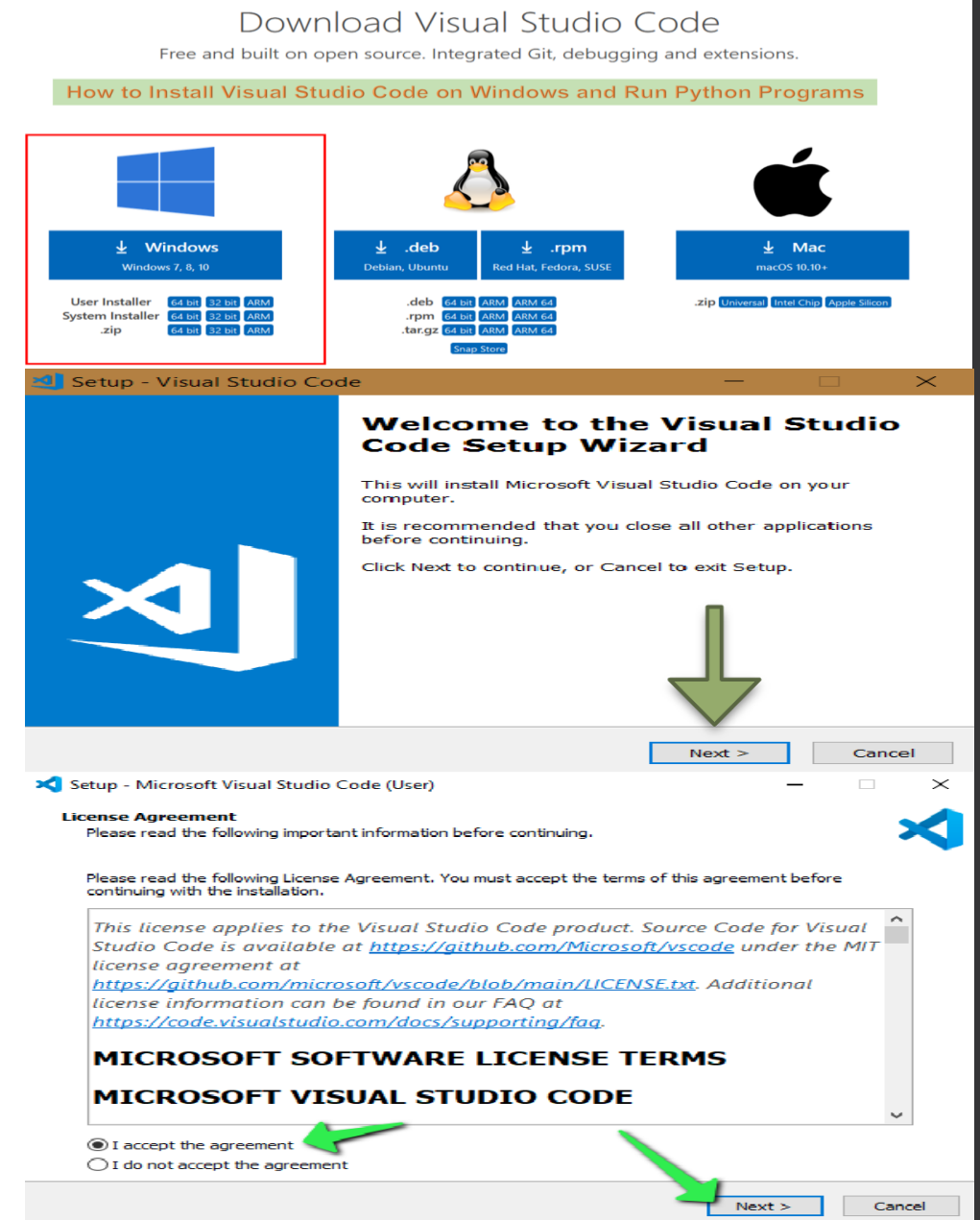
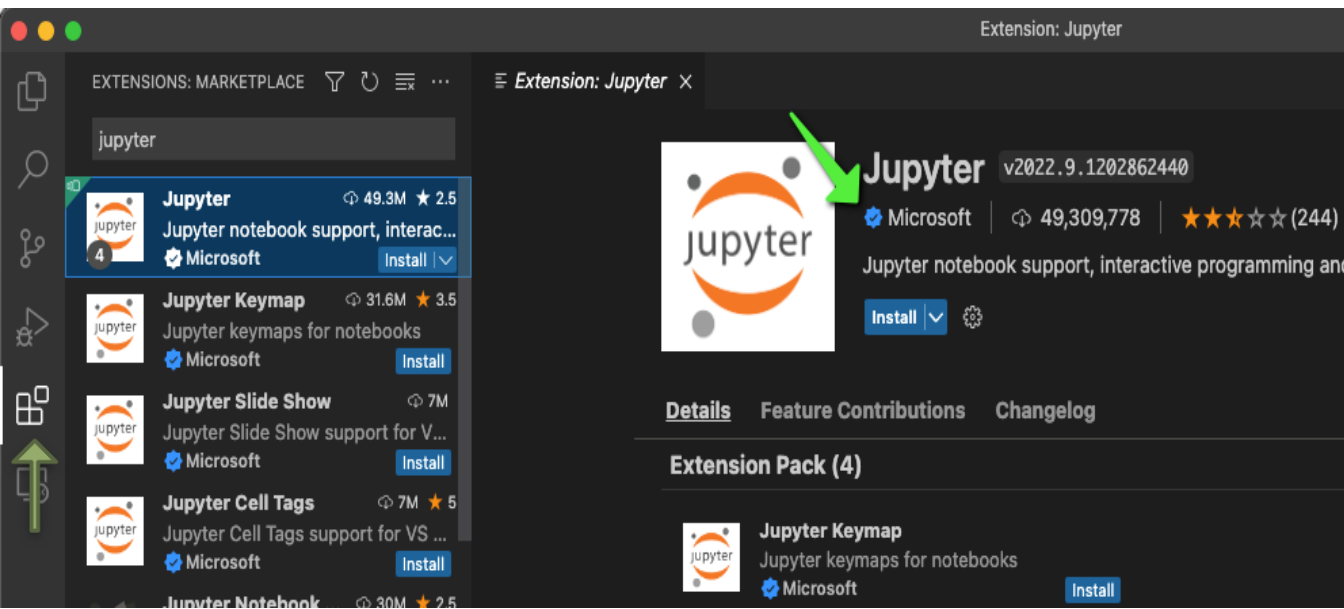
Visit: <https://code.visualstudio.com/download>

Install Visual Studio Code (VS Code)

- Download and install VS Code from the official website.

Install Jupyter Extension in VS Code

- Open VS Code.
- Go to the Extensions tab (left sidebar) and search for Jupyter.
- Click Install to add the Jupyter extension.



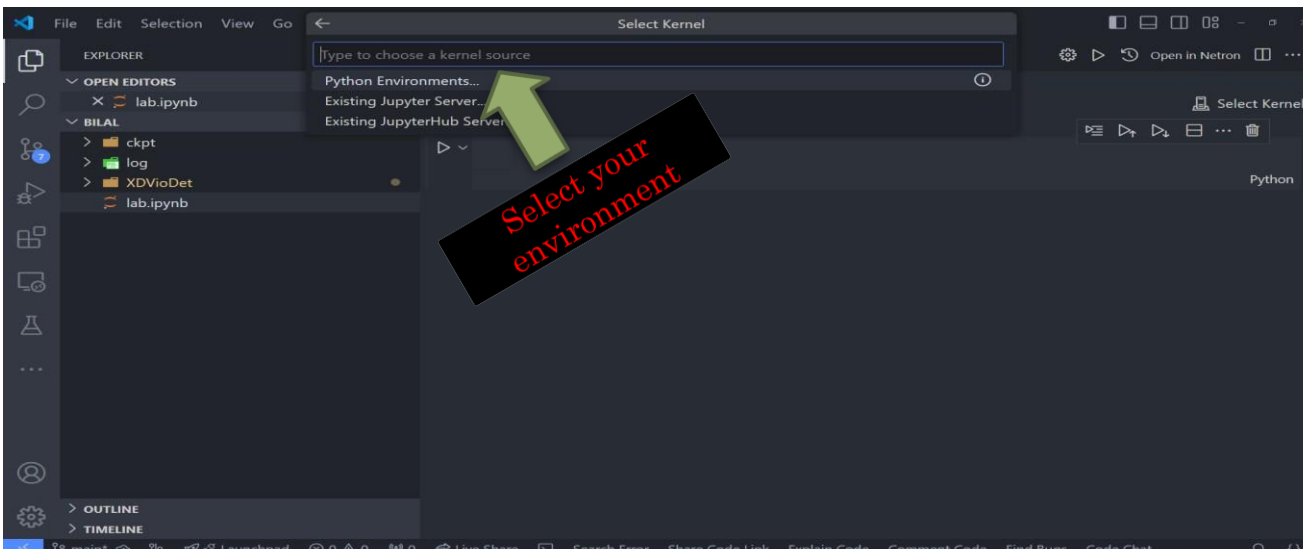
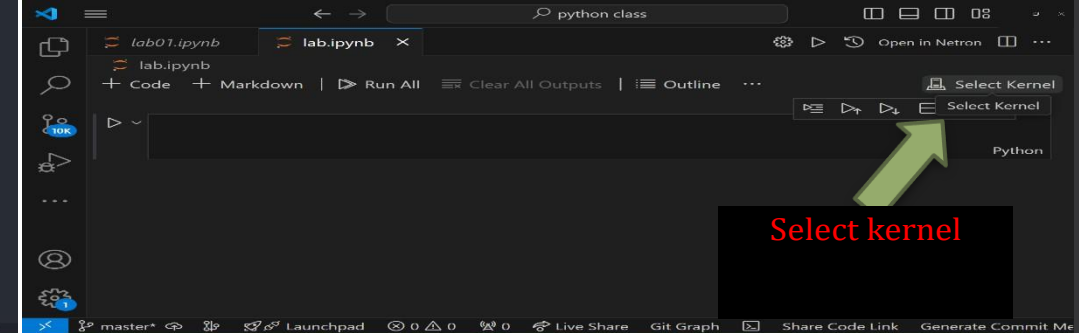
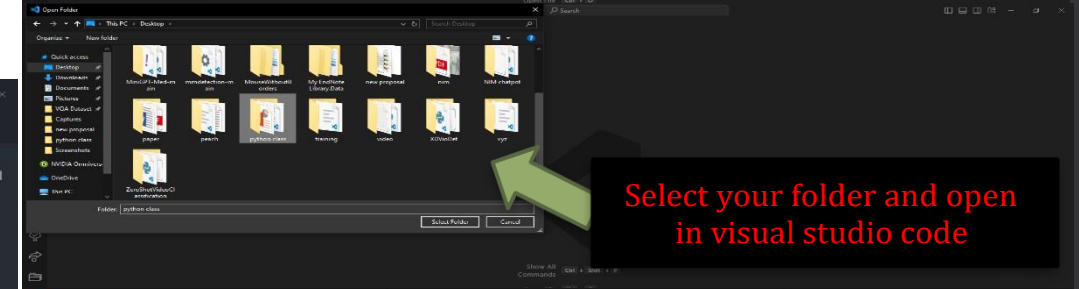
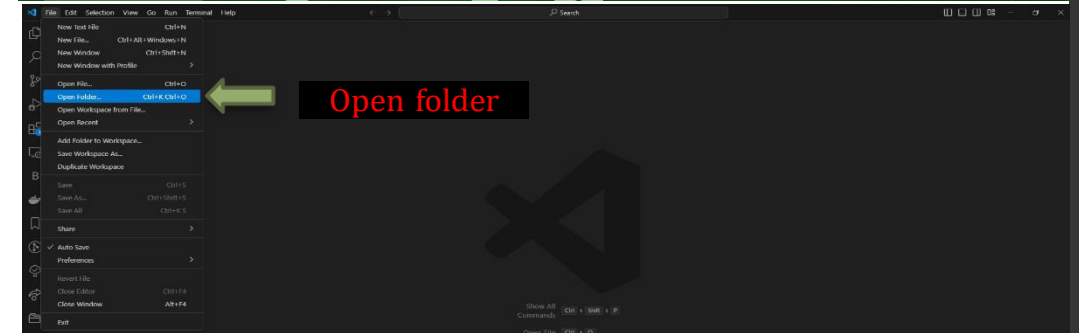
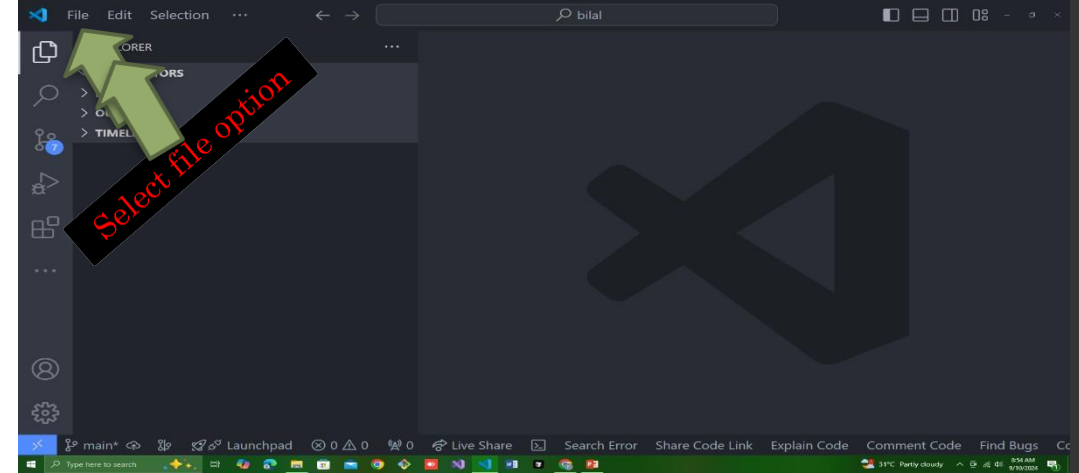
Working with Files, Folders, Kernel, and Terminal in VS Code

Select a File or Folder in VS Code

- ❑ Open VS Code.
- ❑ Click on File in the top menu.
- ❑ Select Open Folder to choose your project directory.
- ❑ Alternatively, click the Explorer icon on the left sidebar and select a file.

Select Jupyter Kernel

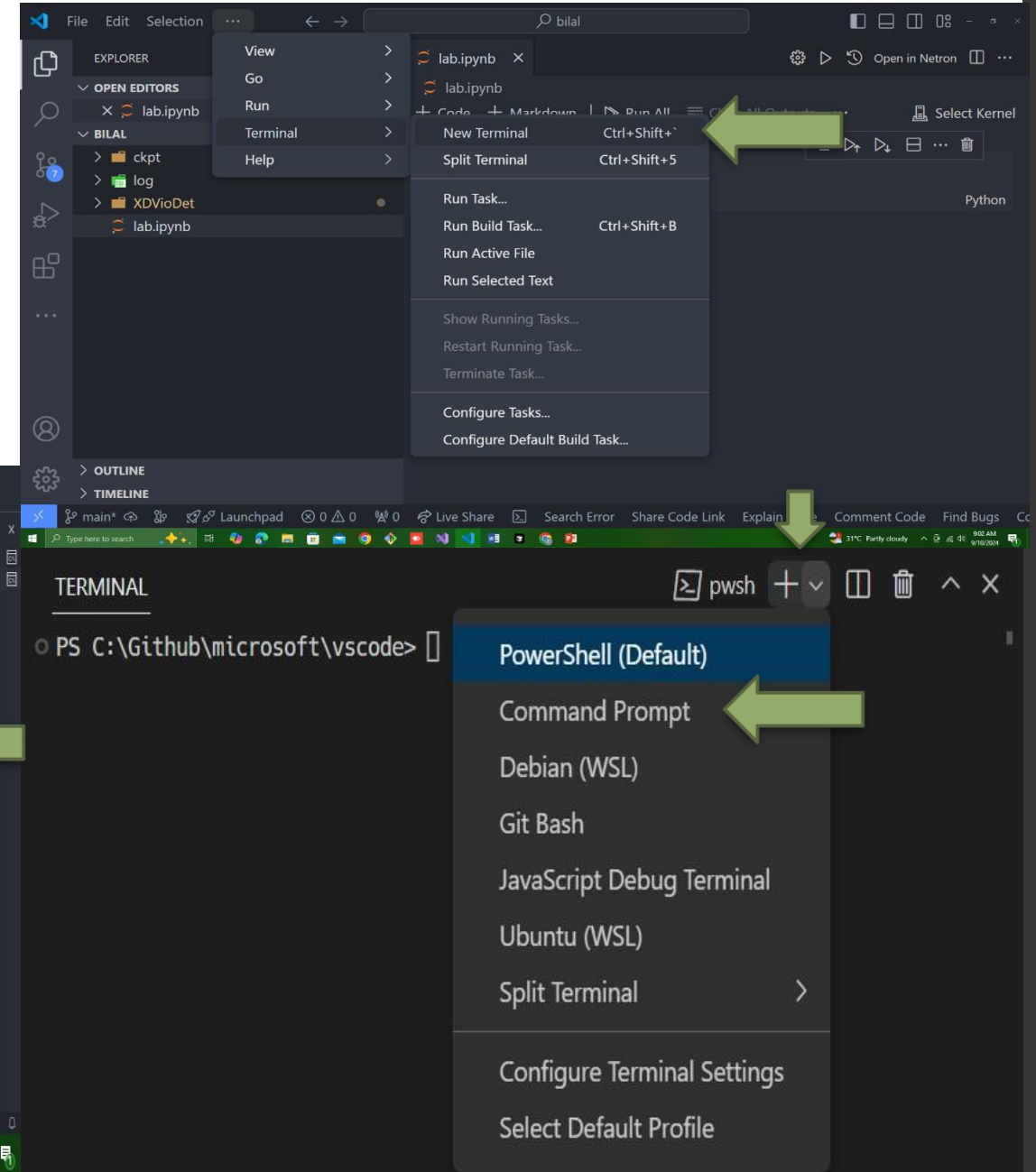
- ❑ After opening a Jupyter Notebook file (.ipynb):
- ❑ Click on the kernel name (top-right of the notebook editor).
- ❑ Choose the Python environment or kernel you want to use for running the notebook.



Working with Files, Folders, Kernel, and Terminal in VS Code

Open Terminal

- In VS Code, press **Ctrl + Shift** to open the integrated terminal.
- Alternatively, go to **View > Terminal** from the top menu to open a new terminal session.



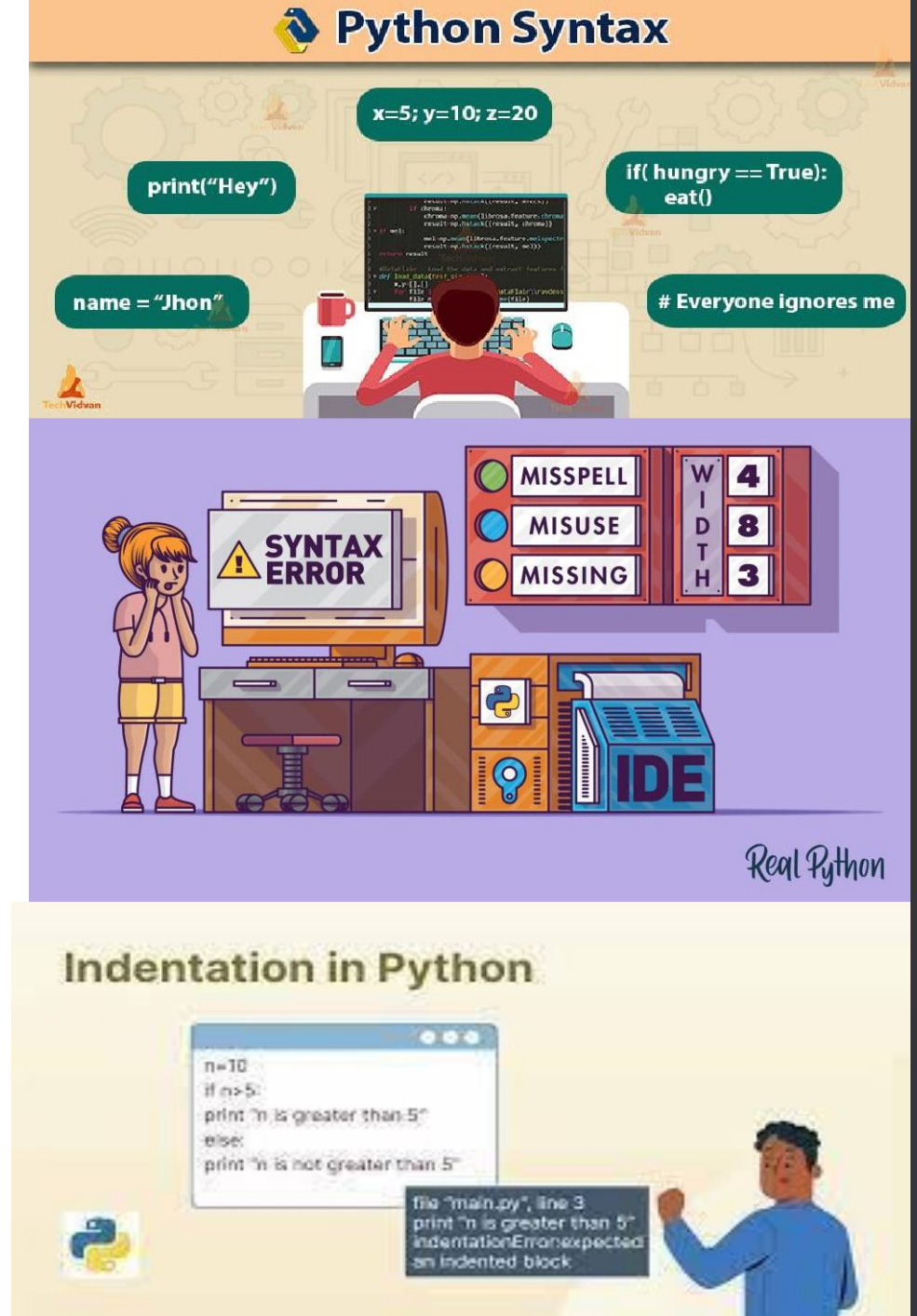
Basic Python Syntax

- **Python Syntax Overview:**

- ❑ Python code is executed line by line (interpreted language).
- ❑ No need for explicit declaration of variables.
- ❑ Indentation is crucial in Python (used for defining blocks of code).
- ❑ Case-sensitive language (e.g., Variable and variable are different).

- **Basic Syntax Elements:**

- ❑ **Variables:** Names assigned to values (e.g., `x = 10`).
- ❑ **Operators:** Symbols for performing operations (e.g., `+`, `-`, `*`, `/`).
- ❑ **Comments:** Notes in the code for clarification (use `#` for single-line).
- ❑ **Functions:** Defined blocks of reusable code (e.g., `def my_function():`).



Writing Your First Python Program

- "Hello, World!" Example:
- Explanation of the **print()** function.
- Writing and executing your first Python program.
- Understanding Python Syntax:
- Code structure in Python.
- Code Cells vs. Markdown Cells in Jupyter Notebook:
- Difference between code and markdown cells.
- Adding notes and explanations in markdown.



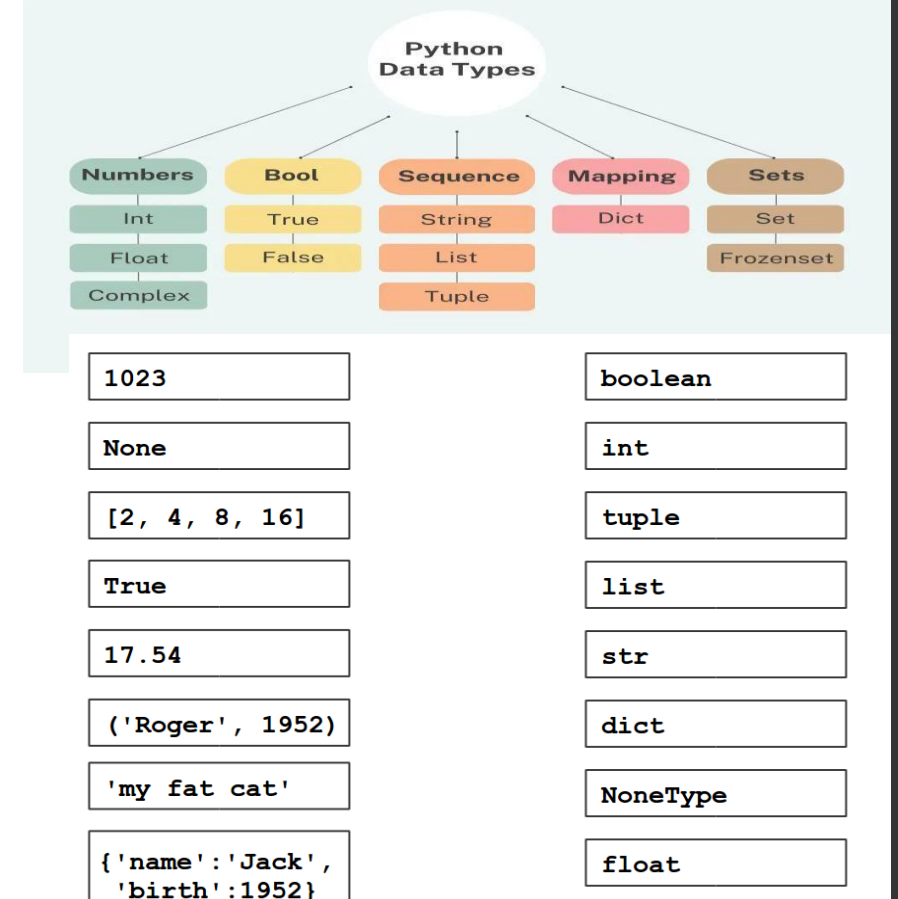
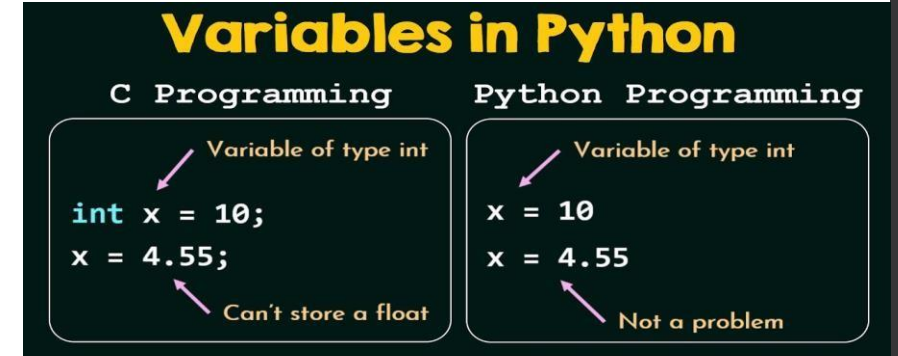
Python Data Types and Variables

• Variables

- ❑ Variables are used to store data values.
- ❑ No need to declare the type; Python determines it automatically.
 - ❑ `x = 10` # Integer
 - ❑ `name = "Alice"` #

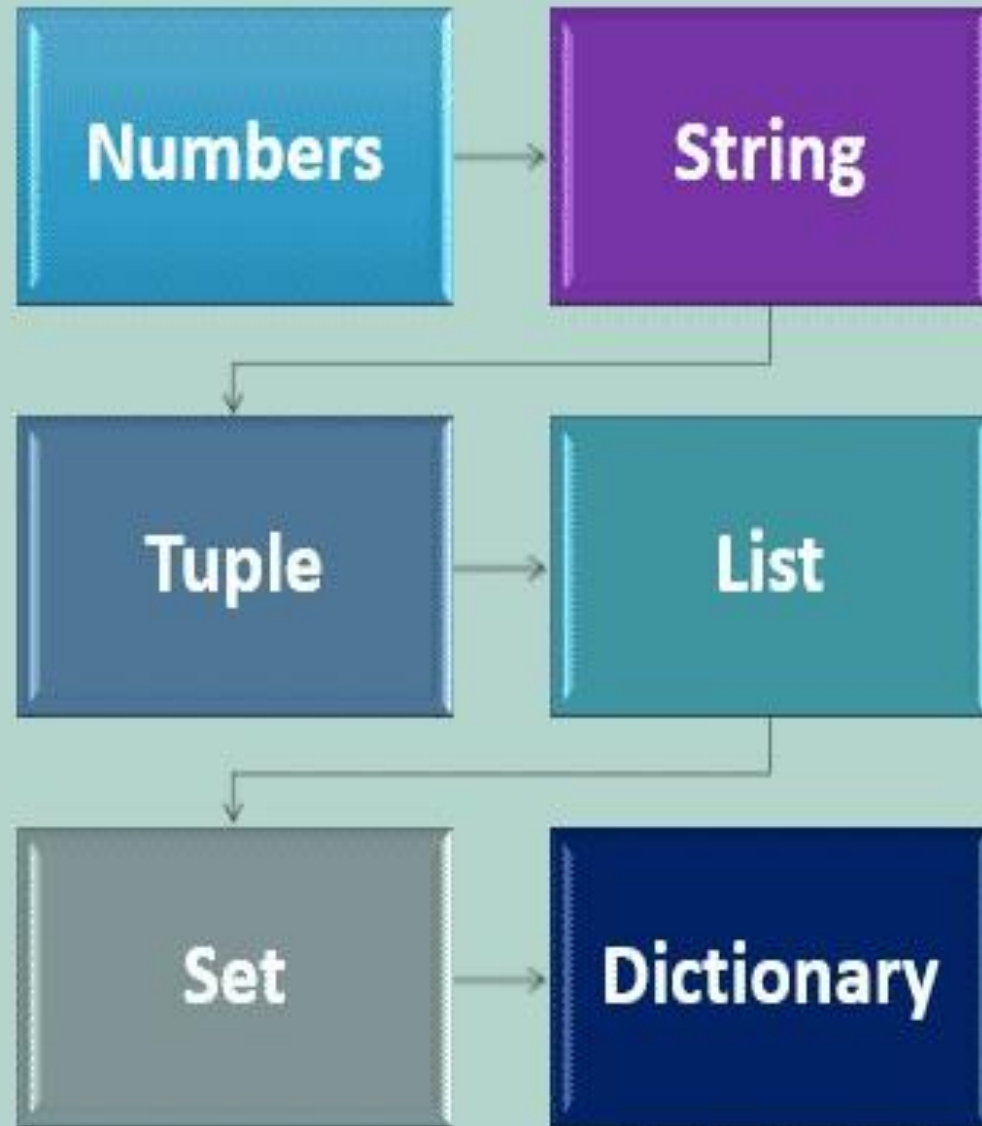
• Common Data Types

- ❑ **int**: Whole numbers (e.g., 5, 100)
- ❑ **float**: Decimal numbers (e.g., 3.14, 0.99)
- ❑ **str**: Text or string (e.g., "Hello", "Python")
- ❑ **bool**: True or False values (e.g., True, False)
- ❑ **list**: Ordered, mutable collection (e.g., [1, 2, 3])
- ❑ **tuple**: Ordered, immutable collection (e.g., (1, 2, 3))
- ❑ **dict**: Key-value pairs (e.g., {"key": "value"})



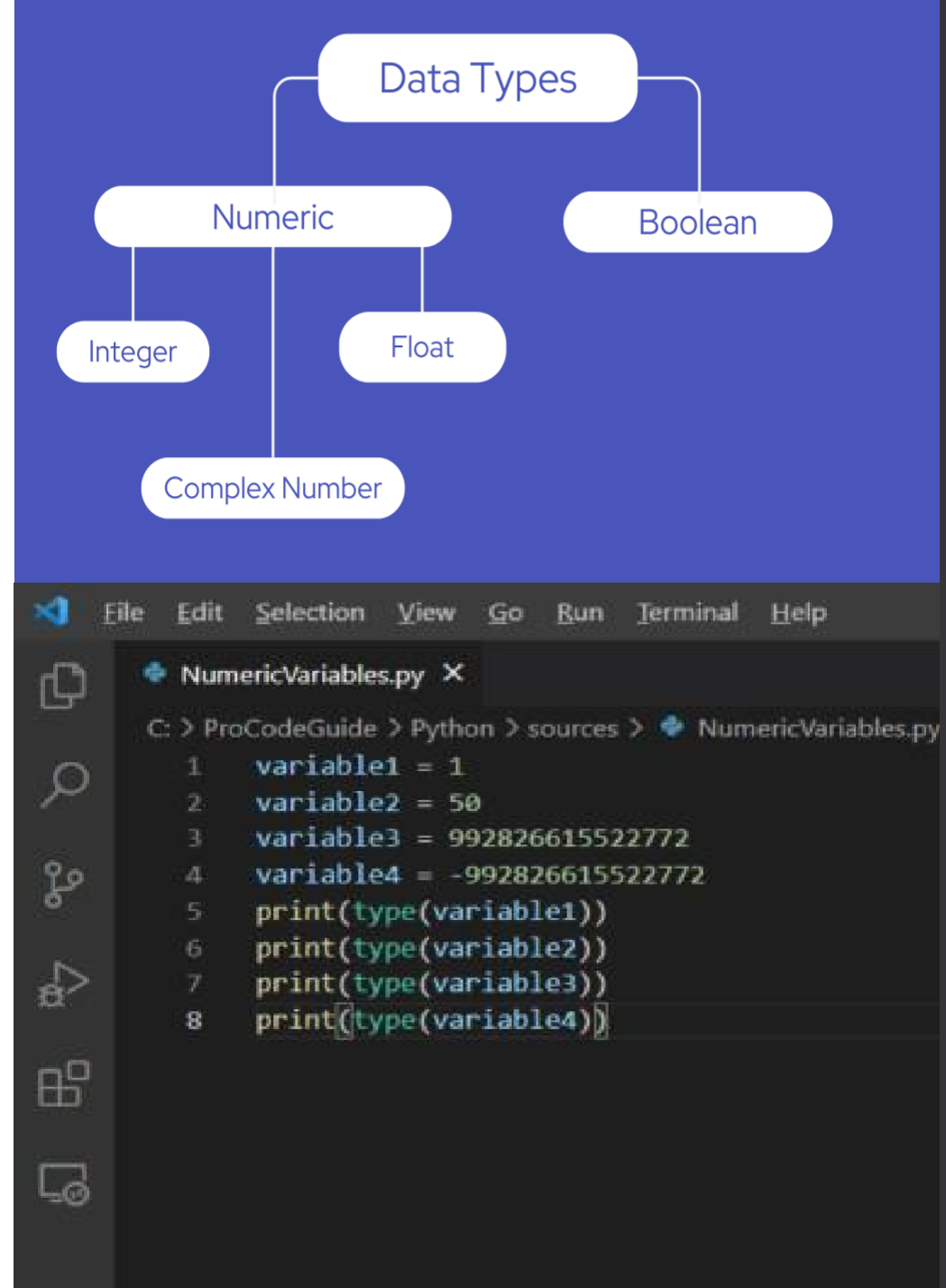
Python

Data Types



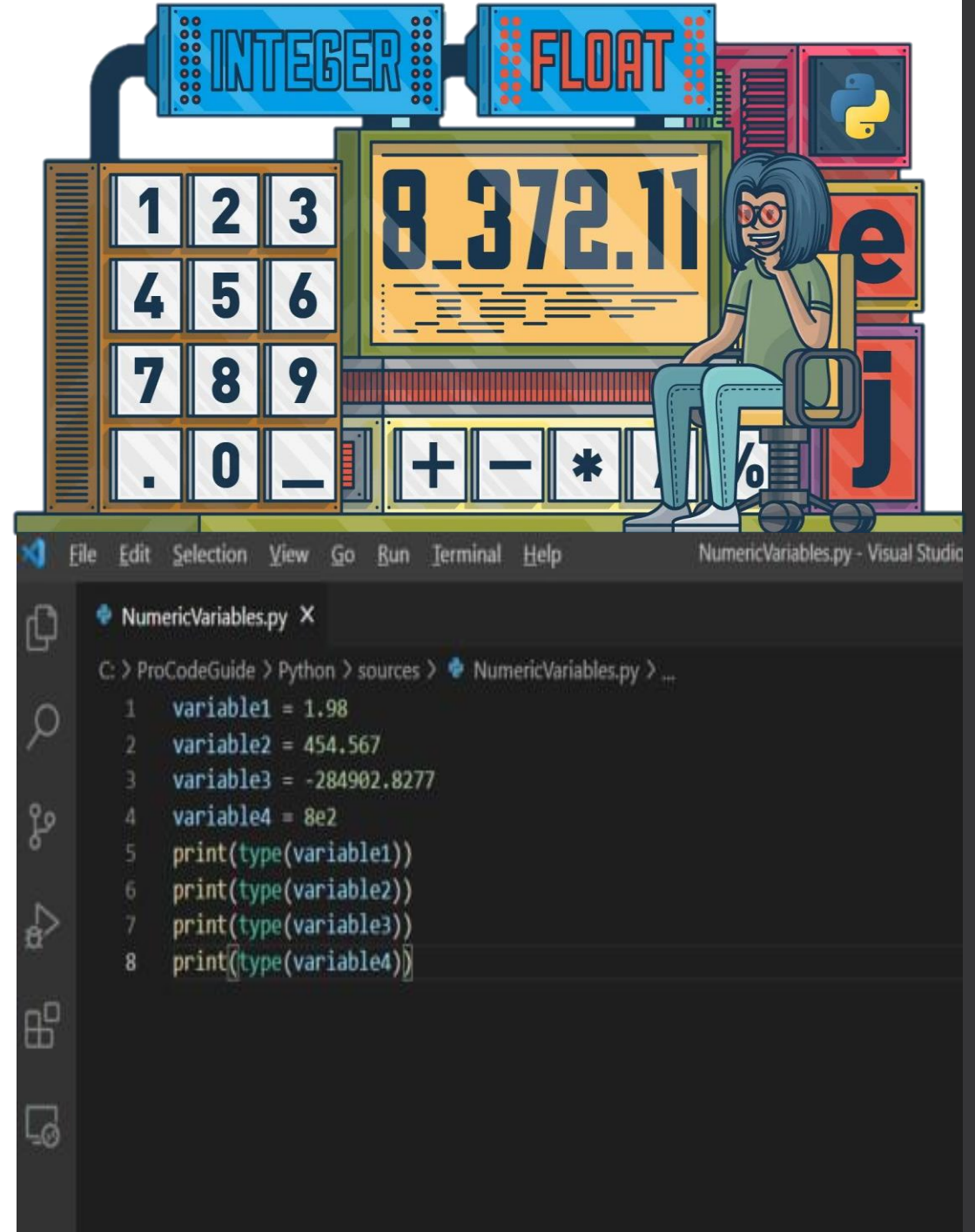
Integer (int)

- Integers are whole numbers, either positive, negative, or zero.
- Used for counting, indexing, and arithmetic operations
- Cannot have decimal or fractional parts
- Commonly used in loops, indexing arrays, and mathematical computations
- Examples of integers: -10, 0, 15



Float (float)

- Floats are numbers with decimal points, representing fractional values.
- Useful for precise calculations (e.g., financial transactions, measurements)
- Can represent large or small numbers using scientific notation (e.g., $1.23e4$)
- Can be positive or negative
- Floats take up more memory than integers because of the decimal precision
- **Example:**
 - `price = 19.99` Represents the price of an item



String (str)

- Strings are sequences of characters used to store and manipulate text.
- Can contain letters, numbers, symbols, and spaces
- Enclosed in either single (') or double (") quotes
- Strings are immutable: once created, their content cannot be changed
- Commonly used for names, messages, filenames, etc.
- Supports various operations like concatenation (+), slicing ([]), and formatting (f-strings)
- Example:
 - `greeting = "Hello, world!"` # A basic greeting message

```
strings.py

Student1 = "George"
Student2 = "George"
Student3 = "Geordie"

print(id(Student1)) # 2873542725936
print(id(Student2)) # 2873542725936
print(id(Student3)) # 8653454375582
```

```
# Defining strings
var1 = "Hello "
var2 = "Geek"

# + Operator is used to combine strings
var3 = var1 + var2
print(var3)

# output: Hello Geek
```

```
year = 2022

print(f'Hello {year}')
print(f'Hello {year=}')
print(f'Hello {year:0<4}')
print(f'Hello {year:.1f}')
print(f'Hello {year:;,}')
```

Boolean (bool)

- Booleans represent one of two possible values: True or False.
- Used in conditional statements (e.g., if, while) to control program flow
- Result of logical operations (e.g., $5 > 3$ is True)
- Can be converted from other data types: `bool(0)` is False, `bool(1)` is True
- Often used to represent states, such as on/off, open/closed
- Example:
 - ❑ `is_logged_in = True` # Indicates if a user is logged in

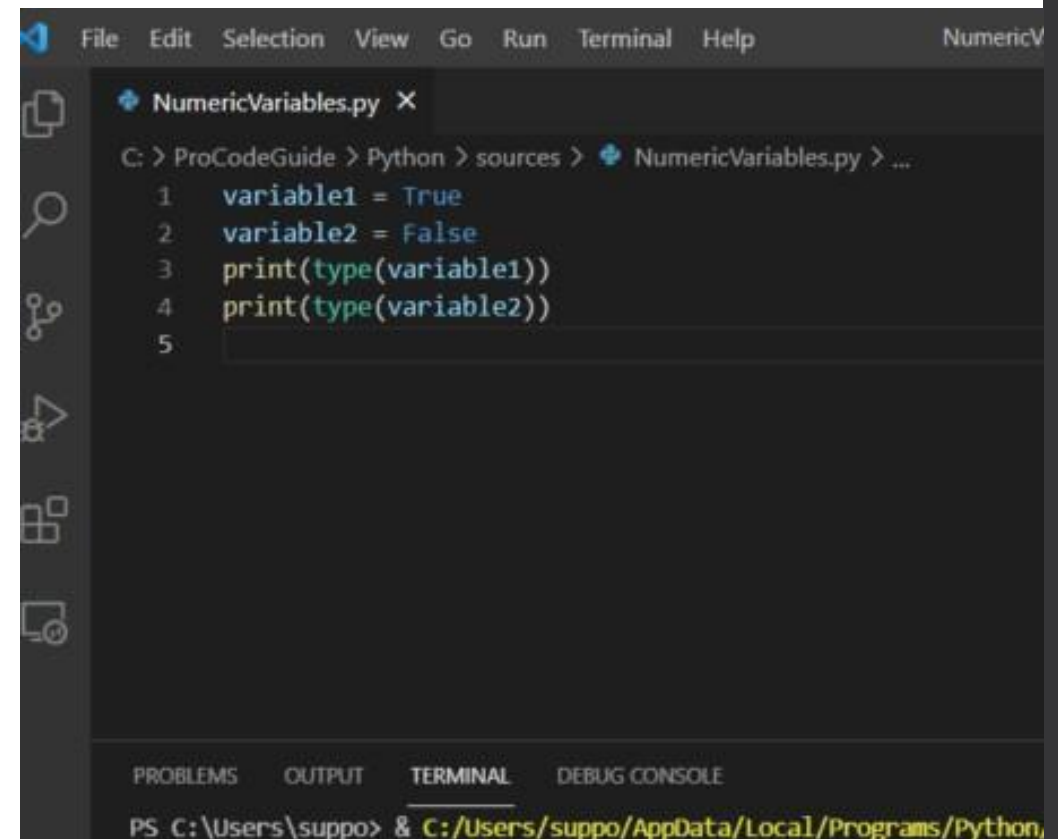


True

False



Working with boolean data type in python

A screenshot of a code editor window titled 'NumericVariables.py'. The code inside is:

```
1 variable1 = True
2 variable2 = False
3 print(type(variable1))
4 print(type(variable2))
5
```

The editor has a dark theme and a sidebar on the left with icons for Explorer, Search, Source Control, and Run and Debug. The bottom of the window shows a terminal with the command prompt 'PS C:\Users\suppo>' and the path 'C:/Users/suppo/AppData/Local/Programs/Python,'.

List (list)

- Lists are ordered collections of items that can store different data types.
- **Mutable:** You can add, remove, or modify elements in a list
- Lists are indexed, starting from 0
- Supports various operations such as appending (.append()), removing (.remove()), and sorting (.sort())
- Can store different data types: integers, strings, even other lists
- Commonly used to store sequences of items like names, numbers, or objects
- Example:
 - fruits = ["apple", "banana", "cherry"] # A list of fruits

```
script.py
row_1 = ['Facebook', 0.0, 'USD', 2974676, 3.5]
        0         1         2         3         4
        └──────────┘
                Index numbers

print(row_1[0])
print(row_1[1])
print(row_1[2])
print(row_1[3])
print(row_1[4])
```

Output

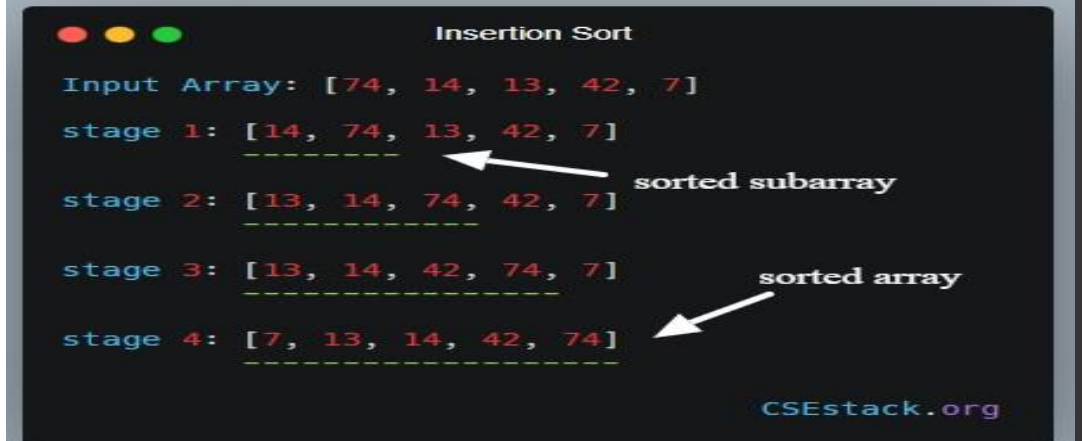
```
Facebook
0.0
USD
2974676
3.5
```

```
script.py
row_1 = ['Facebook', 0.0, 'USD', 2974676, 3.5]

print(row_1[-1])
print(row_1[4])
```

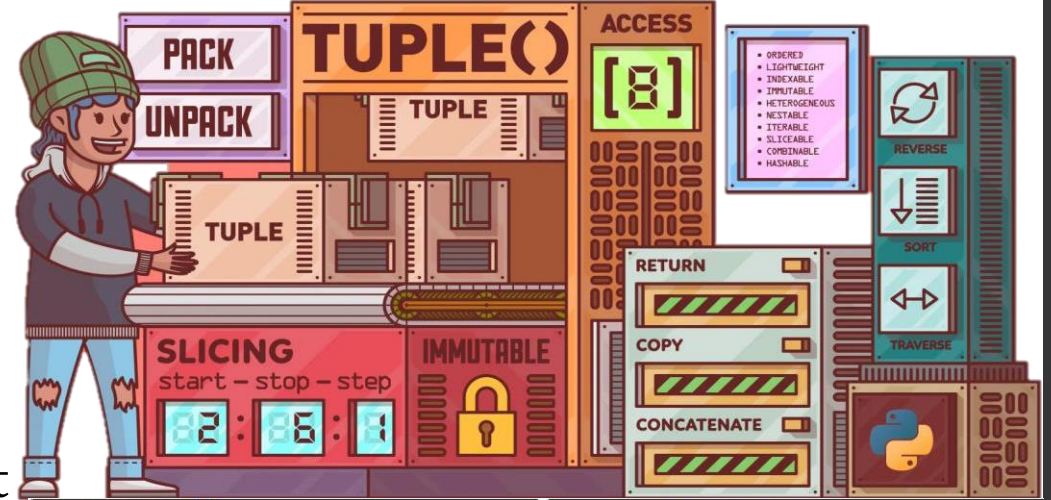
Output

```
3.5
3.5
```



Tuple (tuple)

- Tuples are ordered, immutable collections of items, meaning their values cannot be changed once assigned.
- Tuples are useful for fixed data sets that should not be altered
- Like lists, tuples are indexed starting from 0
- Can contain mixed data types, including integers, floats, strings, or other tuples
- Tuples can be unpacked into separate variables: x, y = (10, 20)
- Commonly used for storing related pieces of data, like coordinates or color values
- Example:
 - coordinates = (34.0522, -118.2437) # Latitude and longitude of a location



```
1 T1=() // tuple display construct
2
3 T2= tuple() // Empty Tuple
4
5 T3= (1)
6 T4= 3, // Single Element
7 T5= (4,) Tuple
8
9 T6= (1,2,3,4,5,6,7,8,9,10) // Long Tuple
10
11 T7= (11,12,(13,14)) // Nested Tuple
12
13 T8= tuple('hello') // Tuple from existing
14 // sequence
15 L= ['a','b','c','d','e']
16 T9= tuple(L)
17
18 T10= tuple(input('Enter tuple Elements:'))
19 // Taking input from Keyboard
```

```
● ● ●
my_tuple = ("dog", 4.5, True, 7, "apple")
print(my_tuple)
# ('dog', 4.5, True, 7, 'apple')

my_second_tuple = tuple(["dog", 4.5, True, 7, "apple"])
print(my_second_tuple)
# ('dog', 4.5, True, 7, 'apple')
```


Dictionary (dict)

- Dictionaries store data in key-value pairs, allowing fast lookups by unique keys.
- Keys must be unique and immutable (e.g., strings, numbers, tuples)
- Values can be of any data type and can be changed (mutable)
- Commonly used for structured data like user profiles, settings, or JSON data
- Supports operations like adding new key-value pairs (dict['key'] = value) and removing keys (del dict['key'])
- Keys are accessed quickly compared to lists due to the dictionary's hashing mechanism
- Example:
 - ❑ student = {"name": "Alice", "age": 22} # A dictionary representing a student

```
a = {'one': 1, 'two': 2}
print(a, type(a))
# output: {'one': 1, 'two': 2} <class 'dict'>

a.update({'three': 3}) # equivalent to a['three'] = 3
print(a)
# output: {'one': 1, 'two': 2, 'three': 3}

a['two'] = 2.1
print(a['two'])
# output: 2.1
```

```
# UPDATE A PYTHON DICTIONARY
>>> dict1 = {'topic': 'Update Python dict'}
>>> dict2 = {'is_helpful': True}
>>> dict1.update(dict2)
>>> dict1
{'topic': 'Update Python dict',
 'is_helpful': True}
```

```
# Add two pairs to the dictionary, using the update method
>>> dictionary.update({'ran': 'run in the past tense',
                       'shoes': 'shoe plural'})

>>> dictionary
{'marathon': 'runners race about 26 miles',
 'person': 'human',
 'ran': 'run in the past tense',
 'run': 'move with speed',
 'shoe': 'shoe type, covering the leg no higher than the ankle',
 'shoes': 'shoe plural'}
```