

AI & ML





Table of Contents

[**Session 1: Python Refresher - Study Guide 2**](#_heading=h.we6bmk7y0tzx)

[1. Python Basics: Variables, Data Types, and Control Structures 2](#_heading=h.8ot7nr4vx7k)

[2. Functions and Modules 5](#_heading=h.90vecjggh0nk)

[3. Introduction to NumPy for Numerical Operations 8](#_heading=h.1ysli4x95ufw)

[4. Introduction to Pandas for Data Manipulation 9](#_heading=h.3pbpnkk9uaxu)

[5. Basic Data Visualization with Matplotlib 11](#_heading=h.hih502yt24yk)

[6. Writing and Using Python Scripts and Modules 12](#_heading=h.5uxu2ooul0qc)

[7. Common Python Pitfalls and Best Practices 14](#_heading=h.2x27bpy67isa)

# 

# 

# Session 1: Python Refresher - Study Guide

## **1. Python Basics: Variables, Data Types, and Control Structures**

**Introduction to Python**

Python is a high-level, interpreted programming language known for its readability and versatility. It is used in various domains such as web development, data science, machine learning, automation, and more.

**Variables and Data Types:**

* **Variables:** Containers for storing data values. In Python, you don’t need to declare a variable’s type explicitly. The interpreter infers the type based on the value assigned to the variable.

**x = 10 # integer**

**y = 15.5 # float**

**name = "Alice" # string**

**is\_active = True # boolean**

**Data Types:**

* **Integers**: Whole numbers, positive or negative.

**a = 5**

**b = -3**

* **Floats**: Numbers with a decimal point.

**pi = 3.14159**

**gravity = 9.8**

* **Strings**: Sequences of characters.

**greeting = "Hello, World!"**

* **Booleans**: `True` or `False`.

**is\_open = True**

* **Complex** **Numbers**: Numbers with a real and imaginary part.

**complex\_num = 2 + 3j**

**Lists**

* **Introduction:** Lists are ordered, mutable collections that can hold items of different data types.

**fruits = ["apple", "banana", "cherry"]**

* **Accessing Elements:** Use indices to access list elements.

**print(fruits[0]) # Output: apple**

* **Modifying** **Elements**: Lists are mutable, so elements can be changed.

**fruits[1] = "blueberry"**

* **List** **Methods**:

**fruits.append("orange") # Adds an item to the end**

**fruits.remove("banana") # Removes an item by value**

**fruits.pop(1) # Removes an item by index**

* **List** **Comprehensions**: Concise way to create lists.

**squares = [x\*\*2 for x in range(10)]**

**Tuples**

* **Introduction:** Tuples are ordered, immutable collections.

**point** = (10, 20)

* **Accessing** **Elements**: Use indices to access tuple elements.

**print(point[0]) # Output: 10**

* **Immutability**: Tuples cannot be changed after creation.

**# point[1] = 30 # This will raise an error**

* **Tuple Packing and Unpacking:**

**coordinates = 1, 2**

**x, y = coordinates**

**Sets**

* **Introduction:** Sets are unordered collections of unique elements.

**fruits = {"apple", "banana", "cherry"}**

* **Adding Elements:** Use `add()` method.

**python**

**fruits.add("orange")**

* **Set Operations:** Union, intersection, difference.

**python**

**set1 = {1, 2, 3}**

**set2 = {3, 4, 5}**

**union\_set = set1.union(set2) # {1, 2, 3, 4, 5}**

**intersection\_set = set1.intersection(set2) # {3}**

**Dictionaries**

* **Introduction:** Dictionaries are collections of key-value pairs.

**python**

**person = {"name": "Alice", "age": 25}**

* **Accessing Elements:** Use keys to access values.

**print(person["name"]) # Output: Alice**

* **Modifying Elements:** Dictionaries are mutable.

**person["age"] = 26**

* **Dictionary Methods:**

**person.keys() # Returns all keys**

**person.values() # Returns all values**

**person.items() # Returns all key-value pairs**

* **Dictionary Comprehensions**: Concise way to create dictionaries.

**squares = {x: x\*\*2 for x in range(10)}**

**Type Conversion:** Python allows for type conversion using functions like `int()`, `float()`, `str()`, etc.

**x = 5 # integer**

**y = 3.2 # float**

**z = x + y # automatically converted to float**

print**(z) # Output: 8.2**

**# Explicit conversion**

x = 5

y = "10"

z = x + int(y) # converting string to int

print(z)  **# Output: 15**

**Control Structures:** Control structures allow you to control the flow of your program.

* **Conditional Statements:**

**age = 20**

**if age >= 18:**

**print("You are an adult.")**

**elif age < 18 and age >= 13:**

**print("You are a teenager.")**

**else:**

**print("You are a child.")**

**Loops:**

* **For Loop:**

**for i in range(5): # 0 to 4**

**print(i)**

* **While Loop:**

**count = 0**

**while count < 5:**

**print(count)**

**count += 1**

**Loop Control Statements:**

**for i in range(10):**

**if i == 3:**

**continue # skips the rest of the loop for i = 3**

**if i == 8:**

**break # exits the loop when i = 8**

**print(i)**

**List Comprehensions:** A concise way to create lists.

**squares = [x\*\*2 for x in range(10)]**

**print(squares) # Output: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]**

**Exercises:**

1. Create a variable `radius` with a value of 7.5, calculate the area of a circle using this radius, and print the result. (Use the formula: `Area = π \* radius^2`).
2. Write a program that prints all the numbers from 1 to 50 that are divisible by 3.
3. Write a list comprehension that creates a list of the squares of all even numbers between 1 and 20.
4. Create a list of your favorite fruits and print each fruit.
5. Create a tuple with your top three favorite numbers and print the second number.
6. Create a set of your favorite hobbies and add a new hobby to the set.
7. Create a dictionary with keys as subjects and values as your scores in those subjects. Print the score of a particular subject.

## 2. Functions and Modules

**Functions**

Functions are blocks of reusable code that perform a specific task. They help to make your code more organized and modular.

**Defining a Function:**

**def greet(name):**

**return f"Hello, {name}!"**

**print(greet("Alice")) # Output: Hello, Alice!**

**Function Arguments:**

* **Positional Arguments**: Arguments that are passed in a specific order.

**def add(a, b):**

**return a + b**

**print(add(5, 3)) # Output: 8**

* **Keyword Arguments:** Arguments passed by explicitly naming the parameter.

**def introduce(name, age):**

**return f"My name is {name} and I am {age} years old."**

**print(introduce(age=25, name="Bob")) # Output: My name is Bob and I am 25 years old.**

* **Default Arguments:** Arguments that have a default value if not provided.

**def greet(name, message="Hello"):**

**return f"{message}, {name}!"**

**print(greet("Charlie")) # Output: Hello, Charlie!**

**print(greet("Charlie", "Hi")) # Output: Hi, Charlie!**

* **Variable-Length Arguments:** Allows you to pass an arbitrary number of arguments.

**def sum\_all(\*args):**

**return sum(args)**

**print(sum\_all(1, 2, 3, 4)) # Output: 10**

**Returning Values:** Functions can return a value using the `return` statement.

**def multiply(x, y):**

**return x \* y**

**result = multiply(6, 7)**

**print(result) # Output: 42**

**```**

**Anonymous Functions (Lambda Expressions):** Lambda functions are small, unnamed functions defined using the `lambda` keyword.

**add = lambda x, y: x + y**

**print(add(2, 3)) # Output: 5**

**Recursion:** A function that calls itself to solve a problem.

**def factorial(n):**

**if n == 1:**

**return 1**

**else:**

**return n \* factorial(n - 1)**

**print(factorial(5)) # Output: 120**

**Modules**

Modules are files containing Python code (functions, variables, etc.) that can be imported into other Python programs.

**Importing Modules:**

**import math**

**print(math.sqrt(16)) # Output: 4.0**

**Custom Modules:** You can create your own module by saving a `.py` file and importing it into another script.

**# In my\_module.py**

**def greet(name):**

**return f"Hello, {name}!"**

**# In another script**

**import my\_module**

**print(my\_module.greet("Diana")) # Output: Hello, Diana!**

**Exercises:**

1. Write a function `is\_even()` that checks if a number is even. Use this function to filter all even numbers from a list of numbers.
2. Create a custom module `calculator.py` with functions for addition, subtraction, multiplication, and division. Import this module into another script and use its functions.

## 3. Introduction to NumPy for Numerical Operations

**Introduction to NumPy**

NumPy is a powerful library for numerical computing in Python. It provides support for arrays, matrices, and a wide range of mathematical functions.

**Creating Arrays:**

**import numpy as np**

**arr = np.array([1, 2, 3, 4, 5])**

**print(arr) # Output: [1 2 3 4 5]**

**matrix = np.array([[1, 2, 3], [4, 5, 6]])**

**print(matrix)**

**# Output:**

**# [[1 2 3]**

**# [4 5 6]]**

**Array Operations:** NumPy allows for element-wise operations on arrays.

**arr1 = np.array([1, 2, 3])**

**arr2 = np.array([4, 5, 6])**

**print(arr1 + arr2) # Output: [5 7 9]**

**print(arr1 \* arr2) # Output: [ 4 10 18]**

**Broadcasting:** Broadcasting allows you to perform operations on arrays of different shapes.

**arr = np.array([1, 2, 3])**

**print(arr + 5) # Output: [6 7 8]**

**Array Reshaping:** Reshaping allows you to change the shape of an array without changing its data.

**arr = np.array([1, 2, 3, 4, 5,**

**6])**

**reshaped\_arr = arr.reshape((2, 3))**

**print(reshaped\_arr)**

**# Output:**

**# [[1 2 3]**

**# [4 5 6]]**

**Statistical Operations:**

**data = np.array([1, 2, 3, 4, 5])**

**print(np.mean(data)) # Output: 3.0**

**print(np.std(data)) # Output: 1.4142135623730951**

**Exercises:**

1. Create a NumPy array containing numbers from 1 to 50. Reshape this array into a 5x10 matrix.
2. Use NumPy to create an array of random numbers and compute the mean, median, and standard deviation.

## 4. Introduction to Pandas for Data Manipulation

**Introduction to Pandas**

Pandas is a powerful data analysis library that provides data structures like Series and DataFrames, which are essential for handling and analyzing structured data.

**Creating DataFrames:**

**import pandas as pd**

**data = {**

**'Name': ['Alice', 'Bob', 'Charlie', 'David'],**

**'Age': [24, 27, 22, 32],**

**'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']**

**}**

**df = pd.DataFrame(data)**

**print(df)**

**DataFrame Operations:** Accessing data, modifying data, and performing operations.

**print(df['Name']) # Accessing a column**

**print(df.iloc[1]) # Accessing a row by index**

**print(df.loc[df['Age'] > 25]) # Filtering data based on a condition**

**df['Age'] += 1 # Modifying data**

**print(df)**

**Data Cleaning:** Handling missing values, removing duplicates, and transforming data.

**df = pd.DataFrame({**

**'A': [1, 2, np.nan, 4],**

**'B': [5, np.nan, np.nan, 8],**

**'C': [10, 11, 12, 13]**

**})**

**df.fillna(0, inplace=True) # Replace NaN with 0**

**df.dropna(inplace=True) # Drop rows with NaN**

**print(df)**

**Grouping and Aggregation:** Grouping data and performing aggregate functions.

**df = pd.DataFrame({**

**'Department': ['HR', 'Engineering', 'HR', 'Engineering'],**

**'Employee': ['Alice', 'Bob', 'Charlie', 'David'],**

**'Salary': [50000, 60000, 55000, 65000]**

**})**

**grouped = df.groupby('Department').mean()**

**print(grouped)**

**Exercises:**

1. Create a Pandas DataFrame from a CSV file. Filter the data to show only rows where a certain condition is met.
2. Group data in a DataFrame by a categorical column and calculate the mean for another column.

## 5. Basic Data Visualization with Matplotlib

**Introduction to Matplotlib**

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

**Creating Basic Plots:**

**import matplotlib.pyplot as plt**

**# Line Plot**

**plt.plot([1, 2, 3, 4], [10, 20, 25, 30])**

**plt.title("Simple Line Plot")**

**plt.xlabel("X-axis")**

**plt.ylabel("Y-axis")**

**plt.show()**

**# Scatter Plot**

**plt.scatter([1, 2, 3, 4], [10, 20, 25, 30])**

**plt.title("Simple Scatter Plot")**

**plt.show()**

**Customizing Plots:** Adding titles, labels, legends, and customizing the appearance.

**plt.plot([1, 2, 3, 4], [10, 20, 25, 30], marker='o', linestyle='--', color='r')**

**plt.title("Customized Line Plot")**

**plt.xlabel("X-axis")**

**plt.ylabel("Y-axis")**

**plt.grid(True)**

**plt.show()**

**Saving Plots:**

**plt.plot([1, 2, 3, 4], [10, 20, 25, 30])**

**plt.savefig("plot.png") # Save as PNG**

**plt.savefig("plot.pdf") # Save as PDF**

**Exercises:**

1. Create a bar plot that shows the number of students in different classes.
2. Create a scatter plot of two variables from a dataset and customize the plot with labels, title, and a grid.

## 6. Writing and Using Python Scripts and Modules

**Writing Python Scripts**

Scripts are files containing Python code that can be executed from the command line.

**Basic Script:**

**# script.py**

**def greet(name):**

**return f"Hello, {name}!"**

**if \_\_name\_\_ == "\_\_main\_\_":**

**print(greet("World"))**

**Run this script from the command line:**

**python script.py**

**Using Python Modules and Packages**

**Creating a Custom Module:** Save a Python file (e.g., `my\_module.py`) with some functions or classes, and import it in another script.

**# my\_module.py**

**def add(a, b):**

**return a + b**

**# main.py**

**import my\_module**

**print(my\_module.add(5, 3)) # Output: 8**

**Virtual Environments:** Virtual environments are isolated Python environments that allow you to manage dependencies for different projects.

**Creating and Activating a Virtual Environment:**

**python -m venv myenv**

**source myenv/bin/activate # On Windows: myenv\Scripts\activate**

**Installing Packages in a Virtual Environment:**

**pip install numpy pandas matplotlib**

**Exercises:**

1. Create a Python script that reads a CSV file, processes the data, and saves the result to a new file.
2. Set up a virtual environment for a project, install the necessary packages, and create a simple script that uses those packages.

## 7. Common Python Pitfalls and Best Practices

**Common Pitfalls**

**Mutable vs Immutable Data Types:** Understand how mutable (e.g., lists, dictionaries) and immutable (e.g., tuples, strings) types behave in Python.

**# Mutable example**

**list\_a = [1, 2, 3]**

list\_b **= list\_a**

**list\_b.append(4)**

**print(list\_a) # Output: [1, 2, 3, 4]**

**# Immutable example**

**str\_a = "Hello"**

**str\_b = str\_a**

**str\_b += " World"**

**print(str\_a) # Output: "Hello"**

**Best Practices:**

**Writing Clean Code:**

* Follow PEP 8 guidelines for code style.
* Use meaningful variable names.
* Comment your code where necessary.

**Exercises:**

1. Write a Python script that includes both mutable and immutable data types. Modify the variables and observe the differences.