Day 1: Introduction to Python

✅ **Topics Covered:**

- What is Python?
- Why Python? (Use cases: Web, AI, Data Science, Automation, etc.)
- Setting up Python (Install Python, VS Code, Jupyter Notebook)
- Running Python scripts (.py files) vs. Interactive mode (Python shell)

📌 **Tasks:**

1. Install Python and verify the installation using `python --version`.
2. Install VS Code or Jupyter Notebook and open a Python file.
3. Print "Hello, World!" in Python.
4. Run a Python script from the terminal.
5. Open Python shell and execute `print(5 + 3)`.
6. Write a comment in Python explaining what Python is.
7. Research and list 5 companies that use Python in production.
8. Explore the `help()` function in the Python shell.
9. Find out what PEP 8 is and summarize it in 2 lines.
10. Write a script that prints your name, age, and favorite programming language.
11. List different Python use cases and pick one to explore further.
12. Research Python's history and its creator.
13. Identify 5 famous Python libraries and their use cases.
14. Check the Python version installed using `sys.version`.
15. Write a script that prints "Python is fun!" 10 times.

Day 2: Variables & Data Types

✅ **Topics Covered:**

- Variables, Constants, and Comments
- Data types (int, float, string, bool, NoneType)
- Type conversion (int(), float(), str())
- Basic Input/Output (input(), print())

📌 **Tasks:**

1. Declare a variable `name` and assign it your name.
2. Declare `age` as an integer and `height` as a float.

3. Take user input for their favorite color and print it.
4. Convert an integer to a string and print its type.
5. Convert a float to an integer and print the result.
6. Assign `None` to a variable and check its type.
7. Use f-string to print "My name is X and I am Y years old."
8. Write a multi-line comment explaining the difference between `int` and `float`.
9. Swap two variables without using a third variable.
10. Write a script that asks for the user's birth year and calculates their age.
11. Check the memory address of a variable using `id()`.
12. Create a constant variable using `ALL_CAPS` naming convention.
13. Declare a Boolean variable and print its value.
14. Use type casting to combine an integer and a string in a print statement.
15. Assign multiple variables in a single line and print them.

Day 3: Operators & Expressions

✅ **Topics Covered:**

- Arithmetic operators (+ - * / % // **)
- Comparison & Logical operators (==, !=, <, >, and, or, not)
- Assignment & Membership operators

📌 **Tasks:**

1. Calculate the area of a rectangle (length * width).
2. Write an expression that checks if 10 is greater than 5.
3. Find the remainder when 17 is divided by 4.
4. Use `//` to perform floor division between two numbers.
5. Use `**` to calculate 2 raised to the power of 5.
6. Check if 100 is not equal to 200.
7. Use `and` and `or` to evaluate `(5 > 3 and 10 < 20) or (4 == 5)`.
8. Use `in` operator to check if "a" is in "apple".
9. Write an expression that prints `True` if a number is even.
10. Write a Python script that swaps two numbers using XOR (`^`).
11. Create a simple calculator using arithmetic operators.
12. Find the square root of a number using exponentiation.
13. Check if a given number is positive, negative, or zero.
14. Compare two strings using comparison operators.
15. Use bitwise AND, OR, and XOR on two integers.

Day 4: Conditional Statements (if-else)

✅ **Topics Covered:**

- `if`, `elif`, `else` conditions
- Nested if statements
- Short-hand if (`x = 10 if condition else 20`)

📌 **Tasks:**

1. Write a script that asks the user for a number and prints whether it's positive, negative, or zero.
2. Check if a number is even or odd using if-else.
3. Ask the user for their age and print whether they can vote (18+).
4. Find the maximum of three numbers using nested if.
5. Write a script that prints "Weekend" if today is Saturday or Sunday.
6. Use a short-hand if to assign `status = "adult"` if `age >= 18`, else "minor".
7. Write a program that checks if a year is a leap year.
8. Ask the user for their marks and print their grade (A, B, C, Fail).
9. Check if a number is divisible by both 3 and 5.
10. Implement a simple calculator that takes two numbers and an operator (+, -, *, /) from the user and performs the operation.
11. Write a program to check if a character is a vowel or consonant.
12. Use a ternary operator to print whether a number is even or odd.
13. Create a script that categorizes a person's age (child, teenager, adult, senior).
14. Write a Python script to find the greatest of four numbers using if-else.
15. Implement a basic login system that checks if a user enters the correct password.

Day 5: Loops (for & while)

✅ **Topics Covered:**

- `for` loop
- `while` loop
- `break`, `continue`, `pass`

📌 **Tasks:**

1. Print numbers from 1 to 10 using a `for` loop.
2. Print even numbers between 1 and 20 using a `while` loop.
3. Find the sum of numbers from 1 to `n` using a loop (`n` is user input).
4. Print the first 10 multiples of 3 using a `for` loop.

5. Reverse a given number using a while loop.
6. Print each character of `"Python"` using a `for` loop.
7. Use `break` to stop a loop when a number reaches 7.
8. Use `continue` to skip number 5 while printing 1-10.
9. Use `pass` inside an empty loop block.
10. Print the factorial of a number using a loop.
11. Count the number of digits in a given number.
12. Print the Fibonacci series up to `n` terms.
13. Find the sum of digits of a number using a loop.
14. Print the multiplication table of a given number.
15. Write a program that asks for a password and keeps asking until the correct password is entered.

Day 6: Lists & Tuples

✅ **Topics Covered:**

- Creating and accessing lists
- List methods (`append`, `remove`, `pop`, `sort`, `reverse`)
- Tuples and immutability

📌 **Tasks:**

1. Create a list of 5 fruits and print it.
2. Print the first and last element of a list.
3. Add `"Mango"` to the list using `append()`.
4. Remove `"Apple"` from the list using `remove()`.
5. Sort a list of numbers in ascending order.
6. Reverse a list of numbers.
7. Print the length of a list using `len()`.
8. Create a tuple with 5 elements and print it.
9. Try modifying an element of a tuple (and note the error).
10. Convert a tuple to a list, modify it, and convert it back to a tuple.
11. Use slicing to print the first three elements of a list.
12. Find the index of an element in a list.
13. Count occurrences of an element in a list.
14. Check if an element exists in a list using `in`.
15. Create a nested list and access its elements.

Day 7: Strings & String Manipulation

✅ **Topics Covered:**

- String indexing & slicing
- String methods (`upper`, `lower`, `replace`, `find`, `split`, `join`)
- f-strings & formatting

📌 **Tasks:**

1. Print the length of a string.
2. Convert `"hello"` to uppercase and `"WORLD"` to lowercase.
3. Replace `"Python"` with `"Java"` in `"I love Python"`.
4. Find the index of `"o"` in `"Hello"`.
5. Count occurrences of `"e"` in `"Elephant"`.
6. Check if a string starts with `"A"` and ends with `"Z"`.
7. Extract `"world"` from `"hello world"` using slicing.
8. Reverse a string without using a loop.
9. Split `"apple,banana,grape"` into a list.
10. Join `['a', 'b', 'c']` into `"a-b-c"` using `join()`.
11. Remove spaces from `" hello world "` using `strip()`.
12. Print `"Hello, my name is X and I am Y years old."` using f-string.
13. Check if `"12345"` contains only digits.
14. Check if `"Hello123"` is alphanumeric.
15. Check if a string is a palindrome.

Day 8: Dictionaries & Sets

✅ **Topics Covered:**

- Dictionary basics
- Dictionary methods (`keys()`, `values()`, `items()`, `get()`)
- Sets and set operations (`union`, `intersection`, `difference`)

📌 **Tasks:**

1. Create a dictionary with keys `"name"`, `"age"`, `"city"` and print it.
2. Add a new key `"gender"` to the dictionary.
3. Access the value of `"age"` from the dictionary.
4. Use `get()` to fetch `"city"` safely.
5. Print all keys of the dictionary using `keys()`.

6. Print all values of the dictionary using `values()`.
7. Remove `"age"` from the dictionary using `pop()`.
8. Create a set `{1, 2, 3, 4, 5}` and print it.
9. Add `6` to the set.
10. Remove `3` from the set.
11. Find the union of `{1, 2, 3}` and `{3, 4, 5}`.
12. Find the intersection of `{1, 2, 3}` and `{3, 4, 5}`.
13. Find the difference of `{1, 2, 3}` and `{3, 4, 5}`.
14. Check if `2` is in the set.
15. Convert a list to a set and remove duplicates.

Day 9: Functions & Lambda

✅ **Topics Covered:**

- Defining and calling functions
- Arguments (`positional`, `keyword`, `default`)
- `return` statement
- `lambda` functions

📌 **Tasks:**

1. Define a function that prints `"Hello, Python!"`.
2. Write a function that takes a number and returns its square.
3. Write a function that checks if a number is even.
4. Create a function that returns the sum of three numbers.
5. Write a function that takes `name` as input and prints `"Hello, {name}"`.
6. Define a function with a default argument (`greet(name, msg="Good morning")`).
7. Write a function that returns the factorial of a number.
8. Write a function that takes a list and returns the maximum element.
9. Use a `lambda` function to find the square of a number.
10. Use a `lambda` function to add two numbers.
11. Use a `lambda` function to return the last character of a string.
12. Write a function that returns the reverse of a string.
13. Write a function that counts vowels in a string.
14. Implement a recursive function to calculate Fibonacci numbers.
15. Write a function that finds the largest element in a list.

Day 10: File Handling

✅ **Topics Covered:**

- Opening and reading files (`open()`, `read()`, `readline()`)
- Writing to files (`write()`, `writelines()`)
- Working with `with open()`

📌 **Tasks:**

1. Create a text file and write `"Hello, world!"` in it.
2. Read the contents of the file and print it.
3. Append `"Welcome to Python!"` to the file.
4. Read the first line of the file.
5. Read all lines and store them in a list.
6. Write a list of fruits into a file.
7. Copy content from one file to another.
8. Count the number of words in a file.
9. Count the occurrences of `"Python"` in a file.
10. Write a program to remove blank lines from a file.
11. Check if a file exists before reading it.
12. Create a file and write user input into it.
13. Find the longest line in a file.
14. Replace `"Hello"` with `"Hi"` in a file.
15. Read a CSV file and print its contents.

Day 11: Exception Handling

✅ **Topics Covered:**

- `try-except` blocks
- Handling multiple exceptions
- `finally` and `else`
- Raising exceptions (`raise`)

📌 **Tasks:**

1. Write a program that handles division by zero.
2. Handle an exception when accessing an invalid list index.
3. Catch a `KeyError` when accessing a non-existing key in a dictionary.
4. Handle multiple exceptions (`ZeroDivisionError`, `TypeError`).
5. Use `finally` to ensure a message prints at the end of the program.

6. Raise an exception manually (`raise ValueError`).
7. Create a function that ensures only positive numbers are passed.
8. Write a program that opens a file and handles `FileNotFoundError`.
9. Catch an exception when trying to convert a string to an integer.
10. Use `assert` to check if a number is positive.
11. Handle `IndexError` when accessing an empty list.
12. Try handling an error inside a function using `try-except`.
13. Handle an exception when trying to divide a number by a string.
14. Use `else` with `try-except` to execute code if no error occurs.
15. Catch an `AttributeError` when accessing an undefined attribute.

Day 12: Object-Oriented Programming (OOP) - Part 1

✅ **Topics Covered:**

- Classes and Objects
- `__init__()` constructor
- Instance and class variables

📌 **Tasks:**

1. Create a class `Car` with attributes `brand` and `model`.
2. Add a method to display car details.
3. Create multiple objects of `Car` and print their details.
4. Use a constructor (`__init__`) to initialize values.
5. Create a class `BankAccount` with `balance` attribute.
6. Add methods `deposit()` and `withdraw()`.
7. Create an object of `BankAccount` and perform operations.
8. Implement a `Person` class with `name` and `age`.
9. Add a method to update the age of a person.
10. Create a class `Student` with attributes `name` and `marks`.
11. Add a method to check if the student has passed (`marks >= 40`).
12. Use instance variables and class variables in a class.
13. Create multiple instances and modify attributes separately.
14. Implement a class with a default parameter in the constructor.
15. Create a class with a method that returns a formatted string.

Day 13: OOP - Part 2 (Inheritance & Polymorphism)

✅ **Topics Covered:**

- Inheritance (`super()`)
- Method Overriding
- Polymorphism

📌 **Tasks:**

1. Create a `Vehicle` class and `Car` class that inherits it.
2. Add a `speed` method in `Vehicle` and override it in `Car`.
3. Use `super()` to call parent class methods.
4. Implement a `Shape` class and `Rectangle` subclass.
5. Create a `Person` class and `Employee` subclass with an extra attribute `salary`.
6. Add a method in `Employee` to print `salary`.
7. Implement a class hierarchy: `Animal -> Dog`.
8. Override a method in the `Dog` class.
9. Implement a `Bank` class with a method `interest_rate()` and override it in `SavingsAccount`.
10. Demonstrate method overloading by using default arguments.
11. Implement a base class with a method that works differently in multiple subclasses.
12. Create a function that takes different objects and calls a common method.
13. Implement an abstract class in Python.
14. Demonstrate multiple inheritance in Python.
15. Use `isinstance()` to check if an object is of a specific class.

Day 14: Modules & Packages

✅ **Topics Covered:**

- Importing modules (`import`, `from ... import`)
- Creating custom modules
- Installing external packages (`pip install`)

📌 **Tasks:**

1. Import the `math` module and use `sqrt()`.
2. Import `random` and generate a random number.
3. Create a module `mymodule.py` and import it.
4. Use `from datetime import datetime` to print current date/time.
5. Use `dir(math)` to see available functions.
6. Install and use an external package (`pip install numpy`).

7. Create a function in a module and use it in another file.
8. Import a module with an alias (`import math as m`).
9. Create a package with multiple modules.
10. Write a module with a function to return the square of a number.
11. Create a module with constants and use them in another script.
12. Use `sys.path` to check Python module search paths.
13. Import a module inside a function instead of globally.
14. Create a `calculator.py` module with add, subtract, multiply, and divide functions.
15. Import functions selectively using `from module import func`.

Day 15: Working with APIs

✅ **Topics Covered:**

- Sending API requests (`requests` module)
- Parsing JSON responses
- Handling API errors

📌 **Tasks:**

1. Install and import the `requests` module.
2. Make a GET request to a public API (`https://jsonplaceholder.typicode.com/todos/1`).
3. Parse and print JSON response from an API.
4. Extract and print a specific field from the API response.
5. Handle API errors using `try-except`.
6. Make a POST request to an API.
7. Send data in JSON format in a POST request.
8. Use query parameters in an API request.
9. Extract and print a list of users from an API response.
10. Check the response status code before processing the response.
11. Use headers in an API request.
12. Make an API call inside a function.
13. Save API response data into a JSON file.
14. Create a CLI script that fetches data from an API.
15. Fetch weather data from OpenWeatherMap API and display it.

Week 3: Advanced Python Concepts & Problem-Solving

Day 16: File Handling & JSON

✅ **Topics Covered:**

- Reading & writing files (`.txt`, `.csv`)
- Working with JSON (`json` module)
- Handling file exceptions

📌 **Tasks:**

1. Open and read a text file.
2. Write a list of names to a file.
3. Append new content to an existing file.
4. Count the number of lines in a file.
5. Read a CSV file and extract specific columns.
6. Write data to a CSV file.
7. Read and parse a JSON file.
8. Convert a Python dictionary to JSON and save it.
9. Pretty-print JSON data.
10. Handle missing JSON keys without errors.
11. Create a function to find the most frequent word in a file.
12. Find the longest word in a text file.
13. Sort words in a file alphabetically and save the result.
14. Copy contents from one file to another.
15. Delete a file programmatically.

Day 17: Object-Oriented Programming (OOP) in Python

✅ **Topics Covered:**

- Classes and objects
- Constructors (`__init__`)
- Inheritance & polymorphism

📌 **Tasks:**

1. Create a `Car` class with attributes (`brand`, `model`, `year`).
2. Add a method `get_info()` to return car details.
3. Create multiple objects and print their details.
4. Implement a constructor for automatic initialization.
5. Use `@property` to make an attribute read-only.
6. Create a subclass `ElectricCar` that inherits from `Car`.
7. Override a method in the subclass.
8. Use `super()` to call a parent class method.

9. Implement multiple inheritance with a `Vehicle` class.
10. Implement operator overloading (`+` for combining two objects).
11. Define a private attribute and use getter/setter methods.
12. Create a `BankAccount` class with `deposit()` and `withdraw()`.
13. Implement a method to track the total number of objects created.
14. Implement a class method using `@classmethod`.
15. Use `@staticmethod` to define a utility function.

Day 18: Error Handling & Logging

✅ **Topics Covered:**

- Try, except, finally
- Custom exceptions
- Logging in Python

📌 **Tasks:**

1. Handle `ZeroDivisionError` in a division function.
2. Use `try-except` to handle a missing file.
3. Handle `KeyError` in a dictionary.
4. Catch multiple exceptions in a single block.
5. Use `finally` to close a file after reading.
6. Raise a `ValueError` if age is negative.
7. Define a custom exception `InvalidAgeError`.
8. Implement a function that logs errors to a file.
9. Create a log file and write different levels of logs (`INFO`, `ERROR`).
10. Use `assert` statements to validate input.
11. Implement a retry mechanism for an unreliable function.
12. Use `logging` to track function execution time.
13. Catch and log all unhandled exceptions globally.
14. Implement nested try-except blocks.
15. Use `with` statement to handle file exceptions.

Day 19: Functional Programming (Lambda, Map, Filter, Reduce)

✅ **Topics Covered:**

- Lambda functions

- `map()`, `filter()`, `reduce()`
- List comprehensions

📌 **Tasks:**

1. Create a lambda function to add two numbers.
2. Use `map()` to square all elements in a list.
3. Use `filter()` to find even numbers from a list.
4. Use `reduce()` to find the product of all numbers.
5. Implement a function to check for prime numbers using `filter()`.
6. Convert a list of temperatures from Celsius to Fahrenheit using `map()`.
7. Use list comprehension to find the squares of numbers from 1 to 10.
8. Generate a dictionary of squares using dictionary comprehension.
9. Flatten a nested list using list comprehension.
10. Find the intersection of two lists using `filter()`.
11. Write a function using `reduce()` to find the largest number in a list.
12. Implement a function that returns a lambda function.
13. Use `map()` with multiple lists.
14. Use `filter()` to find words longer than 5 characters.
15. Implement a simple `cache` function using a dictionary.

Week 4: Advanced Topics & Real-World Applications

Day 20: Multi-threading & Multi-processing

✅ **Topics Covered:**

- `threading` module
- `multiprocessing` module
- Synchronization

📌 **Tasks:**

1. Create a thread to print numbers from 1 to 10.
2. Create multiple threads for different tasks.
3. Use `join()` to wait for a thread to finish.
4. Implement thread synchronization using a lock.
5. Implement a thread that runs in the background.
6. Use `multiprocessing` to run two functions simultaneously.
7. Implement process communication using `Queue()`.

8. Compare performance between multi-threading and multi-processing.
9. Create a thread that prints current time every second.
10. Implement thread-safe increment function.
11. Use `ProcessPoolExecutor` to execute functions in parallel.
12. Implement a worker thread pool for processing tasks.
13. Use `Thread` class with arguments.
14. Use `multiprocessing.Manager()` to share state across processes.
15. Write a function that utilizes multi-processing to speed up computation.

Day 21: Regular Expressions (Regex) in Python

✅ **Topics Covered:**

- `re` module
- Pattern matching
- Search and replace

📌 **Tasks:**

1. Find all email addresses in a given text.
2. Extract all phone numbers from a paragraph.
3. Validate if a string is a valid date format.
4. Replace all occurrences of "Python" with "Java".
5. Extract hashtags from a tweet.
6. Find all words that start with "a".
7. Validate a password (min 8 characters, alphanumeric).
8. Extract all numbers from a string.
9. Find words that end with "ing".
10. Validate an IPv4 address.
11. Match a URL pattern in a string.
12. Extract all capitalized words.
13. Find duplicate words in a sentence.
14. Replace multiple spaces with a single space.
15. Parse an HTML tag using regex.

Day 22 - Day 30: Data Structures & Algorithms in Python

📌 **Topics Covered Each Day:**

- Day 22: **Arrays & Lists**
- Day 23: **Stacks & Queues**

- Day 24: **Linked Lists**
- Day 25: **Hashmaps & Sets**
- Day 26: **Binary Search & Sorting**
- Day 27: **Recursion & Dynamic Programming**
- Day 28: **Graphs & Trees**
- Day 29: **Backtracking & Greedy Algorithms**
- Day 30: **Real-World Problem-Solving**