Python 15 Days Hard Challenge

THE HARDEST CHOICES REQUIRES THE STRONGEST WILL

Designed by **Shivraj Anand**

Connect via
linktr.ee/shivrajanand

TABLE OF CONTENTS

DAY 1: INTRODUCTION AND SETUP

- Introduction to Python and its applications
- Setting up Python environment (installing Python, choosing an IDE or text editor)
- Basic understanding of Python syntax and running your first Python program

PRACTICE QUESTIONS

1. What is Python, and what are its main features?
2. Explain the process of setting up a Python environment on your computer.
3. Write a Python script that prints "Hello, World!" when executed.
4. Describe the difference between Python 2.x and Python 3.x.
5. Name at least three popular Integrated Development Environments (IDEs) for Python.
6. What are Python's main data types?
7. How do you declare and initialize variables in Python?
8. Write a Python script that calculates the area of a rectangle given its length and width.
9. What is the purpose of indentation in Python code?
10. Explain the role of the Python interpreter in executing Python programs.

DAY 2: VARIABLES AND DATA TYPES

- Understanding variables and data types in Python (integers, floats, strings, Booleans)
- Variable assignment and basic operations
- Type conversion and built-in functions for data types

PRACTICE QUESTIONS

1. Theory: Explain the concept of variables in Python and how they differ from constants.
Coding: Write a Python script that declares variables for storing a person's name, age, and city, and then prints them.

2. Theory: Discuss the concept of dynamic typing in Python and its advantages.
Coding: Write a Python script that demonstrates dynamic typing by assigning different types of values to a single variable.

3. Theory: Describe the basic data types in Python and provide examples of each.
Coding: Write a Python script that initializes variables with different data types (integers, floats, strings, Booleans) and prints their types.

4. Theory: Explain the difference between mutable and immutable data types in Python.
Coding: Write a Python script that demonstrates the mutability of lists (mutable) and tuples (immutable) by performing operations on them.

5. Theory: Discuss the concept of type conversion in Python and when it's necessary.
Coding: Write a Python script that prompts the user to enter their age as a string, converts it to an integer, and then prints it.

6. Theory: Describe the concept of variable naming conventions in Python.
Coding: Write a Python script that declares variables with meaningful names following the snake case convention.
7. Theory: Explain how variable scope works in Python, including local and global scope.
Coding: Write a Python script that demonstrates local and global variable scope by defining variables inside and outside functions.

8. Theory: Discuss the concept of constants in Python and how they are implemented.
Coding: Write a Python script that defines a constant (e.g., PI) and uses it in a calculation.
9. Theory: Explain the concept of variable reassignment in Python and its implications.
Coding: Write a Python script that demonstrates variable reassignment by assigning different values to the same variable and printing its final value.

10. Theory: Describe the role of comments in Python code and their significance.
Coding: Write a Python script with comments explaining each step of a simple calculation or operation.

11. Theory: Discuss the difference between local and global variables in Python functions.
Coding: Write a Python script that defines a global variable and modifies it within a function.

12. Theory: Explain the concept of data type casting in Python and why it's useful.
Coding: Write a Python script that demonstrates data type casting by converting a float to an integer and vice versa.

13. Theory: Describe the concept of truthiness and falseness in Python.
Coding: Write a Python script that uses truthiness and falseness to control the flow of execution in a conditional statement.

14. Theory: Discuss the importance of choosing meaningful variable names in Python programming.
Coding: Write a Python script that performs a calculation or operation, using descriptive variable names.

15. Theory: Explain the purpose of the `input()` function in Python and how it's used.
Coding: Write a Python script that prompts the user to enter their name and then prints a personalized greeting.

16. Theory: Describe the concept of string interpolation in Python and how it's achieved.
Coding: Write a Python script that uses string interpolation to create a formatted string containing variables.

17. Theory: Discuss the concept of variable assignment and how it differs from variable initialization.
Coding: Write a Python script that demonstrates variable assignment by assigning values to variables and then reassigning them.

18. Theory: Explain the concept of operator precedence in Python and how it affects expressions.
Coding: Write a Python script that uses different operators in expressions and demonstrates their precedence.

19. Theory: Discuss the concept of namespaces in Python and their role in variable scope.
Coding: Write a Python script that defines variables with the same name in different namespaces and prints their values.

20. Theory: Describe the importance of choosing appropriate data types for variables in Python programming.
Coding: Write a Python script that demonstrates the use of different data types (e.g., integer, float, string) in a calculation or operation.

DAY 3: CONTROL FLOW AND LOOPS

- Conditional statements (if, elif, else) & Looping structures (for loops, while loops)

PRACTICE QUESTIONS

1. Write a Python script that checks if a given number is prime or not.

2. Write a Python script that calculates the factorial of a given number using a while loop.

3. Write a Python script that generates a multiplication table for a given number up to a specified range.

4. Write a Python script that takes a list of numbers as input and calculates the sum of all the positive numbers.

5. Write a Python script that prompts the user to enter a password. Keep prompting until the correct password "python123" is entered.

6. Write a Python script that takes a string as input and counts the number of vowels (a, e, I, o, u) in it.

7. Write a Python script that calculates the sum of all even numbers between 1 and 100 using a for loop.

8. Write a Python script that prompts the user to enter a number and prints whether it is a perfect square or not.

9. Write a Python script that takes a list of words as input and prints the longest word in the list.

10. Write a Python script that generates the Fibonacci sequence up to a specified number of terms.

ADVANCED QUESTIONS:

1. Write a Python script that prompts the user to enter a string and then prints the reverse of the string without using any built-in reverse functions or slicing.

2. Write a Python script that takes a list of numbers as input and removes all duplicates, preserving the original order of elements.

3. Write a Python script that prompts the user to enter a number and then checks if it's a perfect number. A perfect number is a positive integer that is equal to the sum of its proper divisors (excluding itself).

4. Write a Python script that takes a list of integers as input and rearranges the elements in such a way that all even numbers appear before all odd numbers. The relative ordering of even and odd numbers should remain unchanged.

5. Write a Python script that generates the Pascal's triangle up to a specified number of rows. Pascal's triangle is a triangular array of binomial coefficients, where each number is the sum of the two numbers directly above it in the previous row.

DAY 4: FUNCTIONS

- Defining and calling functions
- Parameters and arguments
- Return statement and function scope

PRACTICE QUESTIONS

1. Define a Python function called `greet` that takes a name as input and prints "Hello, [name]!".

2. Call the `greet` function defined in the previous question with your name as an argument.

3. Define a Python function called `calculate area` that takes the length and width of a rectangle as parameters and returns its area.

4. Call the `calculate area` function with length=5 and width=3, and print the result.

5. Define a Python function called `power` that takes two parameters: a number and its exponent. If no exponent is provided, the function should default to squaring the number.

6. Define a global variable `x` with a value of 10. Inside a function called `add to x`, add 5 to `x`. Print the value of `x` before and after calling the function.

7. Define a Python function called `min max` that takes a list of numbers as input and returns both the minimum and maximum values in the list.

8. Define a Python function called `factorial` that calculates the factorial of a given number using recursion.

9. Define a Python function called `is even` that takes a number as input and returns True if it's even, otherwise returns False. Then define another function called `count even` that takes a list of numbers as input and returns the count of even numbers using the `is even` function.

10. Write a Python script that defines a list of numbers and uses a function to calculate the squares of all numbers in the list.

ADVANCED QUESTIONS

1. Higher-Order Functions:

Define a Python function called `apply operation` that takes three arguments: a function `operation`, and two numbers `x` and `y`. The function should apply the given operation to `x` and `y` and return the result. Then, define functions `add`, `subtract`, `multiply`, and `divide` that perform addition, subtraction, multiplication, and division, respectively. Finally, use the `apply operation` function to perform each operation on two given numbers.

2. Decorators:

Define a Python decorator called `debug` that prints the arguments and return value of a function each time it's called. Apply this decorator to a function and observe its behaviour when called with different arguments.

3. Generator Functions:

Write a Python generator function called `Fibonacci` that yields the Fibonacci sequence indefinitely. Then, write a function called `get_nth_fibonacci` that takes a number `n` as input and returns the nth Fibonacci number. Use the `Fibonacci` generator to implement `get_nth_fibonacci`.

DAY 5: STRING MANIPULATION

- String operations and methods
- String formatting and interpolation
- Regular expressions for advanced string manipulation

PRACTICE QUESTIONS

1. Write a Python script that takes a string as input and prints its length.

2. Write a Python script that takes two strings as input and concatenates them to form a single string. Print the concatenated string.

3. Write a Python script that takes a string as input and converts it to uppercase. Print the uppercase string.

4. Write a Python script that takes a string as input and counts the number of occurrences of a specific character in the string.

5. Write a Python script that takes a string as input and checks if it contains only alphabetic characters (letters). Print True if all characters are alphabetic, otherwise print False.

6. Write a Python script that takes a string as input and checks if it starts with a specific substring. Print True if it does, otherwise print False.

7. Write a Python script that takes a string as input and checks if it ends with a specific substring. Print True if it does, otherwise print False.

8. Write a Python script that takes a string as input and reverses it. Print the reversed string.

9. Write a Python script that takes a string as input and checks if it is a palindrome (reads the same forwards and backwards). Print True if it is, otherwise print False.

10. Write a Python script that takes a string as input and splits it into a list of words. Print the list of words.

ADVANCED QUESTIONS:

1. Regular Expressions:

Write a Python script that takes a string as input and extracts all email addresses from it using regular expressions. Print the list of extracted email addresses.

```
Input:
"Contact us at email@example.com or support@example.org for assistance."



Output
["email@example.com", "support@example.org"]
```

2. String Formatting:

Write a Python script that takes a list of tuples containing student names and their corresponding grades, and formats it into a nicely aligned table. Each row should display the student's name followed by their grade.

```
Input Example:
[("Alice", 85), ("Bob", 92), ("Charlie", 78), ("David", 88)]


Output Example:

Name    Grade
-------------
Alice   85
Bob     92
Charlie 78
David   88
```

DAY 6: LISTS AND TUPLES

- Understanding lists and tuples in Python
- List and tuple methods and operations
- List comprehensions for concise code

PRACTICE QUESTIONS

1. Write a Python script that creates an empty list and adds integers from 1 to 10 to it. Print the list.

2. Write a Python script that creates a list of fruits containing "apple", "banana", and "orange". Print the list.

3. Write a Python script that creates a tuple of three elements: your name, age, and city. Print the tuple.

4. Write a Python script that creates a list of integers and then modifies the first and last elements of the list. Print the modified list.

5. Write a Python script that creates a tuple of three elements: a string, a list of numbers, and a Boolean value. Print the tuple.

6. Write a Python script that takes a list of numbers as input and converts it into a tuple. Print the tuple.

7. Write a Python script that creates a list of lists, where each inner list contains a student's name and their corresponding grades in different subjects. Print the list.

8. Write a Python script that creates a list of tuples, where each tuple contains a country name and its capital. Print the list.

9. Write a Python script that creates a list of numbers and calculates the sum of all even numbers in the list.

10. Write a Python script that creates a list of strings and sorts them in alphabetical order. Print the sorted list.

ADVANCED QUESTION:

Write a Python script that takes a list of tuples containing student names and their corresponding scores in three subjects. Calculate the average score for each student and store the results in a new list of tuples. Then, sort the list based on the average scores in descending order and print the sorted list.

Input Example:

```
[("Alice", (85, 90, 88)), ("Bob", (92, 87, 95)), ("Charlie", (78, 82, 80))]
```

Output Example:

```
[("Bob", 91.33), ("Alice", 87.67), ("Charlie", 80)]
```

NOTE:

This question requires manipulation of both lists and tuples, including calculation of averages and sorting based on specific criteria. It tests understanding of data structures and basic arithmetic operations.

DAY 7: DICTIONARIES AND SETS

- Working with dictionaries and sets
- Dictionary and set methods and operations
- Applications of dictionaries and sets in real-world scenarios

PRACTICE QUESTIONS

1. Write a Python script that creates an empty dictionary and adds key-value pairs representing the names and ages of three people. Print the dictionary.

2. Write a Python script that creates a dictionary representing a person's contact information (name, email, phone number). Print the dictionary.

3. Write a Python script that creates a dictionary mapping country name to their capitals. Print the dictionary.

4. Write a Python script that creates a set containing the unique elements from a given list. Print the set.

5. Write a Python script that takes two lists as input and creates a dictionary where elements from the first list are keys and elements from the second list are values. Print the dictionary.

6. Write a Python script that creates a dictionary representing the frequency of characters in a given string. Print the dictionary.

7. Write a Python script that takes a dictionary representing inventory items and their quantities, and calculates the total number of items in stock.

8. Write a Python script that takes a dictionary representing student grades in different subjects and calculates the average grade for each student.

9. Write a Python script that takes two dictionaries as input and merges them into a single dictionary. If a key is present in both dictionaries, the value from the second dictionary should overwrite the value from the first dictionary. Print the merged dictionary.

10. Write a Python script that takes a dictionary representing words and their frequencies and returns the word with the highest frequency. If there are multiple words with the same highest frequency, return any one of them.

ADVANCED QUESTION:

1. Write a Python script that takes a list of words as input and creates a dictionary where the keys are the words themselves and the values are sets containing the indices (positions) at which each word appears in the list. Print the dictionary.

Input Example:

```
["apple", "banana", "apple", "orange", "banana", "apple"]
```

Output Example:

```
{
  "apple": {0, 2, 5},
  "banana": {1, 4},
  "orange": {3}
}
```

2. Write a Python script that takes a list of tuples representing student names and their corresponding subjects and scores. The script should calculate the average score for each subject across all students and store the results in a dictionary. Then, it should identify the subject with the highest average score and print it along with its average score.

Input Example:

```
[("Alice", [("Math", 85), ("Physics", 90), ("Chemistry", 88)]),
 ("Bob", [("Math", 92), ("Physics", 87), ("Chemistry", 95)]),
 ("Charlie", [("Math", 78), ("Physics", 82), ("Chemistry", 80)])]
```

Output Example

```
Subject with the highest average score: Physics (average score: 8
```

DAY 8: FILE HANDLING

- Reading from and writing to files
- File modes and methods
- Handling file exceptions and errors

PRACTICE QUESTIONS

1. Write a Python script that creates a new text file named "example.txt" and writes "Hello, world!" to it. Make sure to close the file after writing.

2. Write a Python script that opens the "example.txt" file created in the previous question, reads its contents, and prints them to the console.

3. Write a Python script that appends the string "This is a new line." to the end of the "example.txt" file.

4. Write a Python script that reads the contents of the "example.txt" file line by line and prints each line preceded by its line number.

5. Write a Python script that creates a new CSV file named "data.csv" and writes the following data to it:

```
Name,Age,City
Alice,25,New York
Bob,30,Los Angeles
Charlie,35,Chicago
```

6. Write a Python script that reads the data from the "data.csv" file created in the previous question and prints it to the console.

7. Write a Python script that reads the data from the "data.csv" file created earlier and calculates the average age of the individuals listed in the file.

8. Write a Python script that creates a new JSON file named "data.json" and writes a dictionary containing information about a person (name, age, city) to it.

9. Write a Python script that reads the data from the "data.json" file created in the previous question and prints it to the console.

10. Write a Python script that reads the data from the "data.json" file and updates the age of the person in the file. Then, overwrite the contents of the "data.json" file with the updated data.

DAY 9: OOPS

- Introduction to Object-Oriented Programming (OOP) concepts
- Classes and objects in Python
- Constructors, instance methods, and instance variables

PRACTICE QUESTIONS

1. Create a Python class called `Person` with attributes `name` and `age`. Write a constructor to initialize these attributes and a method `display_info` to print the name and age of the person.

2. Create an object of the `Person` class and call the `display_info` method to print the person's information.

3. Modify the `Person` class to include a method `increase_age` that increases the person's age by a specified amount.

4. Create an object of the modified `Person` class, increase the person's age by 1, and then call the `display_info` method to print the updated information.

5. Create a Python class called `Rectangle` with attributes `length` and `width`. Write methods to calculate the area and perimeter of the rectangle.

6. Create an object of the `Rectangle` class with length 5 and width 3. Call the methods to calculate the area and perimeter, and print the results.

7. Create a Python class called `BankAccount` with attributes `account_number` and `balance`. Write methods to deposit and withdraw money from the account, and also to display the current balance.

8. Create an object of the `BankAccount` class with an initial balance of $1000. Deposit $500 into the account, withdraw $200, and then display the current balance.

9. Create a Python class called `Car` with attributes `make`, `model`, and `year`. Write a method `get_age` to calculate the age of the car based on the current year and its manufacturing year.

10. Create an object of the `Car` class with make "Toyota", model "Camry", and manufacturing year 2018. Call the `get_age` method to print the age of the car.

DAY 10: INHERITANCE AND ENCAPSULATION

- Inheritance and polymorphism in Python
- Encapsulation and access modifiers
- Implementing inheritance and encapsulation in practical examples

PRACTICE QUESTIONS

1. Create a base class `Vehicle` with attributes `make` and `model`. Write a constructor to initialize these attributes and a method `display_info` to print the vehicle's make and model.
2. Create a derived class `Car` from the `Vehicle` class with an additional attribute `year`. Write a constructor to initialize all attributes and override the `display_info` method to print the vehicle's make, model, and year.

3. Create an object of the `Car` class with make "Toyota", model "Camry", and year 2018. Call the `display_info` method to print the vehicle's information.

4. Create a derived class `Electric Car` from the `Car` class with an additional attribute `battery capacity`. Write a constructor to initialize all attributes and override the `display_info` method to print the vehicle's make, model, year, and battery capacity.

5. Create an object of the `Electric Car` class with make "Tesla", model "Model S", year 2022, and battery capacity 100 kWh. Call the `display_info` method to print the electric car's information.

6. Create a base class `Shape` with methods `area` and `perimeter`. Write methods to calculate the area and perimeter of a shape.
7. Create a derived class `Rectangle` from the `Shape` class with attributes `length` and `width`. Write a constructor to initialize these attributes and override the `area` and `perimeter` methods to calculate the area and perimeter of a rectangle.

8. Create an object of the `Rectangle` class with length 5 and width 3. Call the `area` and `perimeter` methods to calculate and print the area and perimeter of the rectangle.

9. Create a derived class `Circle` from the `Shape` class with attribute `radius`. Write a constructor to initialize this attribute and override the `area` and `perimeter` methods to calculate the area and circumference of a circle.

10. Create an object of the `Circle` class with radius 4. Call the `area` and `perimeter` methods to calculate and print the area and circumference of the circle.

DAY 11: NUMPY

- Introduction to NumPy and its advantages
- Creating NumPy arrays
- Basic array operations and functions

PRACTICE QUESTIONS

1. Create a NumPy array with elements `[1, 2, 3, 4, 5]`.

2. Create a 2D NumPy array with elements `[[1, 2, 3], [4, 5, 6], [7, 8, 9]]`.

3. Create a NumPy array containing all zeros with shape `(3, 4)`.

4. Create a NumPy array containing all ones with shape `(2, 5)`.

5. Create a NumPy array containing random integers between 0 and 9 with shape `(3, 3)`.

6. Create a NumPy array containing evenly spaced numbers between 0 and 10 with a step of 2.

7. Reshape the array `[[1, 2, 3], [4, 5, 6]]` into a 3x2 array.

8. Create a NumPy array containing numbers from 0 to 9, then slice it to get elements from index 2 to 6.

9. Add 5 to each element of the array `[1, 2, 3, 4, 5]`.

10. Multiply each element of the array `[1, 2, 3, 4, 5]` by 2.

11. Calculate the sum of all elements in a NumPy array.

12. Calculate the mean of all elements in a NumPy array.

13. Find the maximum element in a NumPy array.

14. Transpose the array `[[1, 2, 3], [4, 5, 6], [7, 8, 9]]`.

15. Perform element-wise multiplication between two NumPy arrays of the same shape.

DAY 12: PANDAS

- Introduction to Pandas library for data manipulation
- Series and Data Frame objects in Pandas
- Indexing, selection, and manipulation of data using Pandas

PRACTICE QUESTIONS

1. Read the iris flower dataset (iris.csv) into a panda's data frame. You can download the dataset from Kaggle's public datasets repository.
   LINK: https://www.kaggle.com/uciml/iris

2. Display the first 5 rows of the data frame loaded from the iris dataset.

3. Display the last 5 rows of the data frame loaded from the iris dataset.
4. Check the data types of each column in the data frame.

5. Display basic statistics (mean, median, min, max, etc.) For numeric columns in the data frame.

6. Add a new column named "species" to the data frame and assign it a value of "iris" for all rows.

7. Drop the "species" column from the data frame.

8. Sort the data frame by the "sepal length(cm)" column in ascending order.

9. Select rows from the data frame where the "petal width(cm)" column is greater than 1.5.

10. Select rows from the data frame where the "species" column contains the substring "versicolor".

11. Group the data frame by the "species" column and calculate the mean values for all numeric columns for each species.

12. Replace all occurrences of "iris" in the "species" column with "flower".

13. Write the modified data frame back to a new csv file named "modified_iris.csv".

14. Create a new data frame containing only the rows where the "petal length(cm)" is greater than the mean petal length(cm) across all species.

15. Create a new data frame containing only the rows where the "sepal width(cm)" is less than the mean sepal width(cm) across all species.

DAY 13: DATA VISUALIZATION USING MATPLOTLIB AND SEABORN

- Introduction to Matplotlib and Seaborn libraries for data visualization
- Basic plotting techniques (line plots, scatter plots, bar plots)

PRACTICE QUESTIONS

1. Plot a line graph showing the trend of a numeric variable over time.

2. Create a scatter plot showing the relationship between two numeric variables.

3. Generate a histogram to visualize the distribution of a numeric variable.

4. Create a bar plot to compare the counts of different categories.

5. Plot a pie chart to show the distribution of categories as proportions of the whole.

6. Generate a box plot to visualize the distribution of a numeric variable across different categories.

7. Create a violin plot to compare the distributions of a numeric variable across different categories.

8. Plot a heatmap to visualize the correlation matrix of numeric variables.

9. Generate a pair plot to visualize the relationships between multiple numeric variables.

10. Create a joint plot to show the relationship between two numeric variables along with their individual distributions.

11. Plot a swarm plot to visualize the distribution of a numeric variable within each category.

12. Generate a bar plot with error bars to show the mean value of a numeric variable across different categories along with their confidence intervals.

13. Create a box plot with grouped data to compare the distributions of a numeric variable across multiple groups.

14. Plot a stacked bar chart to show the composition of a total value broken down by different categories.

15. Generate a line plot with shaded error bands to show the trend of a numeric variable along with its uncertainty.

DAY 14 & 15: FINAL PROJECT

SAMPLE PROJECT IDEAS

1. **To-do List Application:**
Create a simple too list application where users can add, delete, and mark tasks as completed.

2. **Weather App:**
Build a weather application that retrieves weather data from an API and displays it to the user based on their location or input city.

3. **Calculator:**
Develop a basic calculator application that performs arithmetic operations like addition, subtraction, multiplication, and division.

4. **Text-based Adventure Game:**
Design a text-based adventure game where players make choices that affect the outcome of the story.

5. **Quiz App:**
Create a quiz application with multiple-choice questions on various topics. Users can answer questions and receive feedback on their performance.

6. **Budget Tracker:**
Build a simple budget tracker that allows users to input their expenses and income, categorize them, and view summaries and visualizations of their finances.

7. **Password Generator:**
Develop a password generator application that generates strong, random passwords based on user-defined criteria like length and character types.

8. **Hangman Game:**
Implement a hangman game where players try to guess a word by inputting letters. Include features such as displaying the current state of the word and keeping track of incorrect guesses.

9. **Recipe Finder:**
Create a recipe finder application that allows users to search for recipes based on ingredients they have or dietary preferences. Retrieve recipe data from an API.

10. **Simple Chatbot:**
Build a simple chatbot that responds to user input with pre-defined messages or answers questions on specific topics.

BIBLIOGRAPHY

1. CHATGPT for content and question generation
2. Python Programming by Reema Thareja
3. Python official documentation

SUGGESTIONS AND RULES:

1. Before attempting questions learn the topics from any book or online sources
2. Attempt questions without any help
3. Keep all the practice codes in separate files or create a day wise Jupyter notebook.
4. It will be better if you don't use AI autocompletion for programming in IDEs like Visual Studio Code.
5. 15 days hard challenge is not necessary to complete in exactly 15 days. If you find topic of the day too hard then take at most 2 or 3 days to complete it.
6. You can take more than 2 days for Project building as it is no child's play.
7. Best of Luck !