DSECLPFDS/AIMLCPFDS – Introduction to Python

Exercises for Practice (M1 to M3)

Introduction

Welcome to your Python coding assignment! This set of 30 exercises is designed to help you practice and reinforce the fundamental Python programming concepts covered in our curriculum, including basic syntax, data structures, and control flow.

The exercises are organized by curriculum topic and categorized into three difficulty levels: Easy, Moderate, and Difficult. Work through them sequentially or focus on areas where you feel you need more practice.

Important Instruction: For the core logic of these problems, aim to implement the solution yourself using fundamental Python constructs (loops, conditionals, basic data structures, functions). Avoid relying on built-in functions or libraries that would perform the entire task for you, unless the question specifically allows it. The goal is to understand and implement the underlying logic.

Exercises

1. Python Basics

These questions cover fundamental concepts like variable types, basic input/output, arithmetic operations, and using simple functions.

Easy:

1. Variable Types and Output: Declare four variables, each holding a value of a different data type: integer, float, string, and boolean. Assign values that make sense for each type (e.g., an age, a price, a name, a True/False state). Print the value of each variable followed by its data type obtained using the type() function.
2. Basic Input and Arithmetic: Get two numbers from the user as input. One should be an integer and the other a float. Calculate and print their sum, difference,

product, and the result of dividing the float by the integer. Format the results of calculations to two decimal places using f-strings or the .format() method.

3. Formatted Output (f-strings): Ask the user for their favorite programming language and their experience level (e.g., "Beginner", "Intermediate"). Use an f-string to create and print a sentence summarizing this information, such as "You are a [Experience Level] in [Programming Language]."
4. Simple Type Casting: Prompt the user to enter a number as text (e.g., "105"). Convert this input string to an integer and then to a float using type casting functions like int() and float(). Print the original input and the results of both conversions along with their types.

Moderate:

5. Function Introduction: Write a Python function called greet(name) that takes one string argument name and prints a simple greeting message like "Hello, [name]! Welcome to Python programming." Call this function with a name provided by the user input.
6. Handling Multiple Inputs on One Line: Ask the user to enter three numbers on a single line, separated by spaces (e.g., "10 20 30"). Read this input, split the string into individual number strings, convert them to integers, and calculate and print their average.
7. Basic Error Handling for Input: Write a program that asks the user to enter an integer. Use a try-except block to specifically handle a potential ValueError if the user enters input that cannot be converted to an integer (e.g., text). Print a success message if conversion is successful, and a helpful error message otherwise.

Difficult:

8. Combined Basics and Error Handling: Create a program that calculates a person's Body Mass Index (BMI). Prompt the user for their height in meters and weight in kilograms. Implement robust error handling using a try-except block to catch ValueError for non-numeric input (in both height and weight) and ZeroDivisionError if the height entered is zero. Print the calculated BMI value or an appropriate error message indicating the specific issue.

---

2. Python Data Structures

These questions focus on using and manipulating Python's built-in data structures: strings, tuples, lists, sets, and dictionaries.

Easy:

9. List Indexing and Slicing: Create a list containing the names of five different animal names. Print the animal at index 0, the animal at index 3, and a slice containing the last two animals in the list.
10. Tuple Creation and Access: Create a tuple containing the names of three different countries. Print the country at the first index and the country at the third index from the tuple using indexing.
11. String Indexing and Slicing: Create a string variable city = "Metropolis". Print the first character, the character at index 5, and a slice containing characters from index 2 up to (but not including) index 6.
12. Tuple and List Conversion: Given a tuple (10, 20, 30), convert it into a list. Append the number 40 to the list. Then, convert the modified list back into a tuple. Print the final tuple.

Moderate:

13. Palindrome Check (Strings): Given a string, write a Python function to determine if it is a palindrome. A palindrome reads the same forwards and backwards, ignoring case and spaces (e.g., "Level", "A man, a plan, a canal: Panama").
14. Removing Duplicates (Lists): Write a Python program that takes a list which may contain duplicate elements (e.g., [1, 2, 2, 3, 4, 3, 5]) and returns a new list with all duplicates removed, while preserving the original order of the first occurrence of each element.
15. List Intersection: Given two lists of numbers, write a Python function to find and return a new list containing only the elements that are common to both input lists (their intersection).
16. Element Frequency (Lists to Dictionary): Write a Python program that takes a list of elements (e.g., ['apple', 'banana', 'apple', 'orange', 'banana', 'apple']) and returns a dictionary. The keys of the dictionary should be the unique elements from the list, and the values should be their corresponding frequencies (how many times each element appears).
17. Dictionary Operations: Create a dictionary representing a book with keys 'title', 'author', and 'year_published'. Add a new key 'genre' with a string value. Update the value associated with the 'year_published' key. Print the final dictionary.

Difficult:

18. Processing List of Dictionaries: Given a list of dictionaries, where each dictionary represents a product with keys 'name' (string) and 'price' (float), calculate and print the total price of all products in the list.

---

3. Python Programming Constructs

These questions involve using conditional statements (if, elif, else), loops (while, for), flow control (break, continue), and structured error handling (try-except).

Easy:

19. Leap Year Check: Write a Python program that takes a year as input from the user and determines if it is a leap year. Print "Leap year" or "Not a leap year". (A year is a leap year if it is divisible by 4, except for years divisible by 100 but not by 400).
20. Even or Odd: Ask the user to enter an integer. Use a simple if-else statement to check if the number is even or odd and print the result.
21. Simple Boolean Logic: Ask the user to enter an integer. Use and and comparison operators to check if the number is greater than 10 AND less than 20. Print a message indicating whether the condition is True or False.

Moderate:

22. Sum of Digits: Given a positive integer input from the user, write a program using a loop (while or for) to calculate and print the sum of its digits.
23. Prime Factors: Given an integer greater than 1 as input from the user, write a Python function to find and print all of its prime factors. For example, if the input is 12, the output should show the prime factors 2, 2, and 3.
24. Factorial using Recursion: Write a Python function to calculate the factorial of a non-negative integer using recursion. The factorial of a number n is the product of all positive integers[1] less than or equal to n.
25. Binomial Coefficient (nCr): Write a Python function to calculate the binomial coefficient nCr=fracnr(n−r) given non-negative integers n and r as input, where nger. You can use a helper function (recursive or iterative) to calculate factorials.
26. Linear and Binary Search: Given a sorted list of numbers and a target value, write a function to perform a linear search to find the index of the target value. Write another function to perform a binary search for the same purpose. Return -1 if the target is not found in either function.
27. For Loop with Range: Write a program using a for loop and the range() function to calculate and print the sum of all integers from 1 to 50 (inclusive).

28. Exception Handling (Division): Write a program that asks the user for two numbers. Perform division of the first number by the second. Use a try-except block to specifically handle the ZeroDivisionError if the second number is zero, and print an appropriate message.

Difficult:

29. Armstrong Number Check: Given a positive integer and its order (number of digits) as input from the user, write a program to check if it is an Armstrong number. An Armstrong number of order n is a number that is equal to the sum of its own digits each raised to the power of n. Example: 153=13+53+33.
30. Word Frequency from File: Write a program that takes a text file path as input from the user. Read the content of the file, count the frequency of each word (treating words case-insensitively and ignoring common punctuation like periods, commas, exclamation marks, and question marks). Print the frequency count for each unique word and then identify and print the top 3 most frequent words in the file. (Requires file I/O, string manipulation, dictionary usage, and sorting/finding max).
