Practice Problems

This problem set is designed to provide comprehensive practice for Python programming. It includes a progression from basic syntax to advanced topics. Each section starts with fundamental exercises and gradually introduces more complex concepts, ensuring that each problem naturally builds on the previous one.

Section 1: Input, Output, and Basic Data Types

1.1 Basic I/O and Variables

1. Print "Hello, World!".
2. Print an integer constant.
3. Read an integer from the user and print it.
4. Read two integers, display both.
5. Add, subtract, and multiply two integers (without a third variable).
6. Add, subtract, and multiply two integers (using a third variable).
7. Swap two numbers using a third variable.
8. Swap two numbers without using a third variable (Pythonic way).
9. Print the type and ID (`type()`, `id()`) of basic variables (int, float, str, bool).
10. Assign a character to a variable and print it.
11. Read a single character and print it.
12. Read a string from the user and print it (demonstrate reading with and without spaces).

1.2 Arithmetic and Expressions

1. Evaluate and display the result of arithmetic expressions (test integer division vs float division).
2. Find the roots of a quadratic equation given coefficients a, b, and c.
3. Read multiple types of data (int, float, str) and print them with formatted output.
4. Demonstrate explicit and implicit type conversions (casting).

1.3 String Literals and Escape Sequences

1. Use escape sequences (\n, \t, \\) in print statements.
2. Print multi-line strings using triple quotes.

1.4 Constants

1. Use constants: declare a value as a constant by convention (ALL_CAPS) and demonstrate usage.
2. Use enum. Enum for symbolic constants.

Section 2: Conditional Statements and Input Validation

2.1 Basic If, Elif, Else

1. Read an integer and determine if it is positive, negative, or zero.
2. Read two integers and compare them (equality and greater/less).
3. Read three integers and print the smallest and largest.
4. Check if a number is odd or even.
5. Determine if a year is a leap year.
6. Input a character and check if it's an alphabet.
7. Check if the character is a vowel or a consonant.
8. Check if a character is uppercase, lowercase, a digit, or a special character.

2.2 Nested Conditions and Logical Operators

1. Read marks for five subjects and assign a grade based on the percentage.
2. Compute profit or loss given the cost price and the selling price.
3. Input angles and check if they form a valid triangle.
4. Input sides, check for triangle validity, and classify (equilateral, isosceles, scalene).
5. Input the week number (1–7) and print the corresponding weekday name.
6. Input a month number (1–12), display days in the month (handle leap years for February).

2.3 Match-Case (Python 3.10+) & Elif Chains

1. Simple calculator using match-case for operations (+, -, *, /).
2. Print weekday based on user number input using match-case.
3. Input grade character and print remarks using match-case.
4. Handle out-of-range input with a default message.

2.4 Conditional (Ternary) Operator

1. Find the maximum of two numbers using the ternary operator.
2. Determine even/odd using the ternary operator.
3. Assign "Pass"/"Fail" based on marks using the ternary operator.
4. Classify a number as positive/negative/zero using nested ternary operators.

2.5 Input Validation and Error Handling

1. Check for invalid (non-numeric) integer input using exception handling (try/except).
2. Print error messages for inputs outside expected ranges.

Section 3: Loops and Iteration

3.1 While Loops

1. Print numbers from 1 to n.
2. Sum integers from 1 to n.
3. Print the multiplication table of a number.
4. Reverse a number entered by the user.
5. Count and display the number of digits in a number.
6. Calculate factorial using a while loop.

3.2 For Loops

7. Print all even numbers between 1 and 100.
8. Print the first n terms of the Fibonacci sequence.
9. Display all prime numbers from 1 to 100.
10. Calculate the sum of the series 1 + 1/2 + ... + 1/n.
11. Print uppercase ASCII characters and their codes.
12. Create star patterns (square, triangle) of user-given height.

3.3 Control Statements

13. Use break to search for a number in a list and exit on finding.
14. Use continue to skip printing odd numbers.
15. Print only non-negative values from user input, stop on a negative.
16. Find the smallest divisor of a number greater than 1 using break.

3.4 Nested Loops

17. Print multiplication tables for 1 to 10.
18. Print formatted patterns: pyramid, diamond, Pascal's triangle.

3.5 Advanced Looping

19. Calculate the average of positive numbers entered (stop on zero).
20. Check if a number is a palindrome.
21. Count vowels and consonants in a string.
22. Count digit frequency in an integer using loops and list/dictionary.

3.6 Infinite and Sentinel Loops

23. Demonstrate an infinite loop (while True:).
24. Keep reading input until a sentinel value (e.g., -99) is entered.
25. Implement a command processor loop that exits on "quit"/"exit".

3.7 Practical Applications

26. Build a menu-driven program (repeat options until quit).
27. Draw a horizontal/vertical bar chart using user input.
28. Number guessing game with user feedback.

Section 4: Functions and Modular Programming

4.1 Basic Functions

1. Write a function to compute power (base, exponent).
2. Reuse this function to print powers of 2, 3, -3 for exponents 0–9.
3. Function to swap two numbers (by reference via mutable types or tuple return).
4. Functions to find minimum and maximum of three numbers.

4.2 Recursion

5. Factorial function (recursive).
6. Check prime (return 1 or 0), and print all primes up to a given number.
7. Recursive Fibonacci function.
8. Recursive string reversal.

4.3 Call by Value, Reference, and Mutability

9. Demonstrate modifying a list within a function.
10. Attempt to modify an integer parameter, observe no change.
11. Show mutable vs immutable types in argument passing.

4.4 Variable Scope

12. Demonstrate the effect of local and global variables.
13. Use global keyword to modify a global variable from inside a function.
14. Use nonlocal inside nested functions.

4.5 Modularization

15. Split code into functions and modules.
16. Demonstrate importing user-defined modules.
17. Use docstrings and comments appropriately.

4.6 Lambda, Map, Filter, Reduce

18. Use a lambda to square elements of a list.
19. Use map() to double numbers in a list.
20. Use filter() to keep only even numbers.
21. Use reduce() (from functools) to compute the product of a list.

Section 5: Lists, Tuples, Sets, and Dictionaries

5.1 Lists

1. Create a list, print elements.
2. Add/Remove elements, sort, reverse.
3. Slice lists, copy lists.
4. Find min/max/sum/average of a list of numbers.
5. Use list comprehensions for squares, filtering, etc.

5.2 Tuples

6. Create a tuple of numbers and strings.
7. Demonstrate immutability.

8. Unpack tuples (including nested).
9. Use tuple swapping.

5.3 Sets

10. Create a set, add/remove elements.
11. Perform union, intersection, difference, symmetric difference.
12. Remove duplicates from a list using a set.

5.4 Dictionaries

13. Create a dictionary for student names and marks.
14. Add, remove, update entries.
15. Iterate keys, values, items.
16. Count letter frequency in a string using a dictionary.

5.5 Collections and Advanced Mapping

17. Use collections. Counter to count word frequency.
18. Use defaultdict for grouping data.
19. Practice with namedtuple.

Section 6: Strings and String Processing

1. Read and print strings (single/multi-line).
2. String concatenation, repetition.
3. String formatting using %, format(), and f-strings.
4. Slice and index strings.
5. Strip, replace, split, and join strings.
6. Check palindromes, anagrams.
7. Count the vowels/consonants, words in a string.

Section 7: File Handling

8. Open a file, read its contents, close it.
9. Write data to a new file.
10. Copy contents from one file to another.
11. Count characters, lines, and words in a file.
12. Append data to an existing file.
13. Handle file not found and other IO errors.
14. Use context manager (with statement) for files.

Section 8: Exception Handling

15. Basic try-except for division by zero.
16. Handle multiple exceptions (KeyError, ValueError, IOError).
17. Custom exception raising and handling.
18. Use else and finally with try-except.

Section 9: Object-Oriented Programming

9.1 Classes and Objects

1. Define a class with attributes and methods.
2. Create and use objects, demonstrate __init__.
3. Add methods to modify object state.
4. Use __str__ and __repr__.

9.2 Inheritance and Polymorphism

5. Create a base class and subclass; override methods.
6. Demonstrate super(), method resolution order.
7. Use polymorphism via method overriding.

9.3 Encapsulation, Class and Static Methods

8. Use private/protected attributes by convention.
9. Add class variables and methods (@classmethod).
10. Use static methods (@staticmethod).

9.4 Dunder Methods and Operator Overloading

11. Overload arithmetic operators for a custom class.
12. Implement comparison methods (__eq__, __lt__, etc.).

Section 10: Iterators, Generators, and Comprehensions

1. Create a custom iterator class.
2. Write generator functions using yield.
3. Use generator expressions.
4. Demonstrate list, set, and dictionary comprehensions.

Section 11: Advanced Topics

1. Use datetime for current date/time and formatting.
2. Use random for number generation, shuffling.
3. Introduction to regular expressions: match, search, replace text.
4. Command-line argument parsing (sys.argv, argparse).
5. JSON serialization and deserialization.
6. Read environment variables using os.environ.

Section 12: Data Structures & Algorithms (Pythonic)

1. Stack and queue implementations using lists and collections.deque.
2. Singly linked list: class-based implementation.
3. Binary tree: insert, search, traversals (inorder, preorder, postorder).
4. Graph representation using adjacency lists/dictionaries.
5. BFS and DFS traversals (recursive and iterative).
6. Sorting algorithms: bubble sort, insertion sort (with list methods and manual implementation).

Section 13: Functional Programming & Decorators

1. First-class functions and closures.
2. Write and use decorators for logging and timing functions.
3. Chaining multiple decorators.
4. Use functools.lru_cache for memoization.

Section 14: Testing and Best Practices

1. Use assert statements for basic testing.
2. Write unit tests with the unittest framework.
3. Demonstrate basic use of pytest.
4. Apply type hints and use mypy for static type checking.
5. Document functions with docstrings.

Section 15: Mini-Projects and Integrative Exercises

1. Build a mini contact book (CRUD operations stored in a file).
2. Word/line/character count tool for a user-selected file.
3. Number guessing game with difficulty levels.
4. Simple calculator (GUI with tkinter or CLI-based).
5. To-do list manager (with save/load to file).

6. Basic text analyzer: count frequency of words and letters, print top N frequent words.
7. Implement a simple address book using classes and file storage.
8. Write a script that renames multiple files in a directory based on a given pattern.

Miscellaneous :

1. Create and run Python code in Jupyter Notebook cells.
2. Install and use packages with pip.
3. Practice code organization with packages and modules.
4. Create and activate a virtual environment.