150 Python Programming Problems

This document provides a structured list of programming problems in Python, categorized by difficulty. Work through them sequentially, or pick and choose based on your learning goals. Each problem is designed to help you solidify your understanding of Python concepts and improve your problem-solving abilities.

Part 1: Beginner Problems (1-50)

These problems focus on fundamental Python syntax, control flow, basic data types, and simple functions. They are ideal for those just starting their Python journey.

1. **Hello World:** Write a program that prints "Hello, World!".
2. **Basic Arithmetic:** Take two numbers as input and print their sum, difference, product, and quotient.
3. **Even or Odd:** Ask the user for an integer and print whether it's even or odd.
4. **Temperature Converter:** Convert a temperature from Celsius to Fahrenheit and vice versa.
5. **Area of a Rectangle:** Take length and width as input, then calculate and print the area.
6. **Find the Maximum:** Take three numbers as input and print the largest among them.
7. **Leap Year Checker:** Determine if a given year is a leap year.
8. **Vowel or Consonant:** Check if a character entered by the user is a vowel or a consonant.
9. **Simple Calculator:** Implement a basic calculator that can add, subtract, multiply, and divide two numbers based on user choice.
10. **Factorial:** Calculate the factorial of a non-negative integer.
11. **Fibonacci Sequence:** Generate the first 'n' terms of the Fibonacci sequence.
12. **Sum of Natural Numbers:** Calculate the sum of the first 'n' natural numbers.
13. **Multiplication Table:** Print the multiplication table for a given number.
14. **Count Vowels:** Count the number of vowels in a given string.
15. **Reverse a String:** Reverse a given string without using built-in [::-1].
16. **Palindrome Checker:** Check if a given string is a palindrome.
17. **Prime Number Checker:** Determine if a given number is prime.
18. **Generate Primes:** Print all prime numbers within a specified range.
19. **Swap Two Variables:** Swap the values of two variables without using a temporary variable.
20. **Check Positive, Negative, or Zero:** Determine if a number is positive, negative, or zero.

21. **Celsius to Fahrenheit Function:** Write a function to convert Celsius to Fahrenheit.
22. **Fahrenheit to Celsius Function:** Write a function to convert Fahrenheit to Celsius.
23. **Greatest Common Divisor (GCD):** Find the GCD of two numbers.
24. **Least Common Multiple (LCM):** Find the LCM of two numbers.
25. **Sum of Digits:** Calculate the sum of the digits of a given number.
26. **Count Digits:** Count the number of digits in a given integer.
27. **Decimal to Binary:** Convert a decimal number to its binary representation.
28. **Binary to Decimal:** Convert a binary string to its decimal representation.
29. **Simple Interest Calculator:** Calculate simple interest given principal, rate, and time.
30. **Compound Interest Calculator:** Calculate compound interest given principal, rate, time, and compounding frequency.
31. **Kilometers to Miles:** Convert kilometers to miles.
32. **Miles to Kilometers:** Convert miles to kilometers.
33. **Solve Quadratic Equation:** Find the roots of a quadratic equation.
34. **Random Number Generator:** Generate a random number within a specified range.
35. **List Sum:** Calculate the sum of all numbers in a list.
36. **List Average:** Calculate the average of numbers in a list.
37. **Find Minimum in List:** Find the smallest number in a list.
38. **Find Maximum in List:** Find the largest number in a list.
39. **Remove Duplicates from List:** Given a list with duplicates, return a new list with unique elements.
40. **Check Element in List:** Check if an element exists in a list.
41. **Count Element Occurrences:** Count how many times a specific element appears in a list.
42. **Sort a List:** Sort a list of numbers in ascending order (without sort() method directly, implement a simple sorting logic).
43. **Merge Two Lists:** Combine two lists into a single list.
44. **Reverse a List:** Reverse the order of elements in a list.
45. **String Length:** Calculate the length of a string without using len().
46. **Concatenate Strings:** Join two strings together.
47. **Check Substring:** Determine if a substring is present in a given string.
48. **Count Words in String:** Count the number of words in a sentence.
49. **Title Case String:** Convert a string to title case (first letter of each word capitalized).

50. **Pyramid Pattern:** Print a simple star pyramid pattern.

Part 2: Intermediate Problems (51-100)

These problems delve into more complex data structures (dictionaries, sets), file handling, basic object-oriented programming, and common algorithms.

51. **Dictionary Basics:** Create a dictionary with names as keys and ages as values. Add a new entry, update an entry, and delete an entry.
52. **Iterate Dictionary:** Iterate through a dictionary and print all key-value pairs.
53. **Count Word Frequency:** Given a sentence, count the frequency of each word and store it in a dictionary.
54. **Merge Two Dictionaries:** Merge two dictionaries into one.
55. **List of Dictionaries:** Create a list of dictionaries, where each dictionary represents a person with 'name' and 'age'. Find the oldest person.
56. **Sets: Union and Intersection:** Given two sets, find their union and intersection.
57. **Sets: Difference:** Find the difference between two sets.
58. **File I/O: Read File:** Read the content of a text file and print it to the console.
59. **File I/O: Write to File:** Write a string to a new text file.
60. **File I/O: Append to File:** Append a string to an existing text file.
61. **File I/O: Line Counter:** Count the number of lines in a text file.
62. **File I/O: Word Counter:** Count the total number of words in a text file.
63. **Error Handling: Division by Zero:** Write a function that performs division and handles ZeroDivisionError.
64. **Error Handling: Invalid Input:** Prompt the user for an integer and handle ValueError if non-integer input is given.
65. **Custom Exception:** Create a custom exception for a specific scenario (e.g., NegativeNumberError).
66. **Basic Class: Dog:** Create a Dog class with attributes like name and age, and a method bark().
67. **Class with Constructor:** Enhance the Dog class with an __init__ method to initialize name and age.
68. **Class Method:** Add a class method to the Dog class that returns the number of dogs created.
69. **Inheritance: Animal Class:** Create an Animal base class and a Dog subclass that inherits from Animal.
70. **Polymorphism:** Demonstrate polymorphism with different animal classes having a make_sound() method.
71. **Simple Guessing Game:** A game where the computer picks a random number and the user tries to guess it. Provide hints (higher/lower).

72. **Rock-Paper-Scissors:** Implement a simple Rock-Paper-Scissors game against the computer.
73. **Tic-Tac-Toe (Text-based):** A basic text-based Tic-Tac-Toe game for two players.
74. **List Comprehension: Squares:** Generate a list of squares of numbers from 1 to 10 using list comprehension.
75. **List Comprehension: Filter Evens:** Filter out even numbers from a list using list comprehension.
76. **Dictionary Comprehension:** Create a dictionary where keys are numbers from 1 to 5 and values are their squares.
77. **Map Function:** Use map() to square all numbers in a list.
78. **Filter Function:** Use filter() to get only even numbers from a list.
79. **Lambda Functions:** Use a lambda function with map or filter.
80. **Date and Time:** Get the current date and time. Format it in a specific way.
81. **Time Difference:** Calculate the difference between two dates.
82. **Sleep Function:** Pause program execution for a few seconds.
83. **Module: Math:** Use functions from the math module (e.g., sqrt, ceil, floor).
84. **Module: Random:** Generate random numbers and shuffle a list using the random module.
85. **Recursion: Sum of List:** Calculate the sum of elements in a list recursively.
86. **Recursion: Factorial (Recursive):** Implement factorial using recursion.
87. **Binary Search:** Implement binary search on a sorted list.
88. **Bubble Sort:** Implement the bubble sort algorithm to sort a list.
89. **Selection Sort:** Implement the selection sort algorithm.
90. **Insertion Sort:** Implement the insertion sort algorithm.
91. **Matrix Addition:** Add two matrices (represented as lists of lists).
92. **Matrix Multiplication:** Multiply two matrices.
93. **Transpose Matrix:** Transpose a given matrix.
94. **Check Anagrams:** Determine if two strings are anagrams of each other.
95. **First Non-Repeating Character:** Find the first non-repeating character in a string.
96. **Remove Whitespace:** Remove all whitespace from a string.
97. **Check Pangram:** Determine if a string is a pangram (contains every letter of the alphabet at least once).
98. **Number to Words:** Convert a single-digit number (0-9) to its word representation (e.g., 5 -> "Five").
99. **Caesar Cipher:** Implement a simple Caesar cipher (encryption and decryption).
100. **Find Missing Number:** Given a list of numbers from 1 to N with one missing,

find the missing number.

Part 3: Advanced Problems (101-150)

These problems challenge you with more complex algorithms, advanced Python features, and applications of libraries.

101. **Merge Sort:** Implement the merge sort algorithm.
102. **Quick Sort:** Implement the quick sort algorithm.
103. **Graph Traversal: DFS:** Implement Depth-First Search (DFS) on a simple graph (represented as an adjacency list).
104. **Graph Traversal: BFS:** Implement Breadth-First Search (BFS) on a simple graph.
105. **Dijkstra's Algorithm:** Find the shortest path in a weighted graph using Dijkstra's algorithm.
106. **Knapsack Problem (0/1):** Solve the 0/1 Knapsack problem using dynamic programming.
107. **Longest Common Subsequence:** Find the longest common subsequence of two strings using dynamic programming.
108. **Generating Permutations:** Generate all permutations of a list of elements.
109. **Generating Combinations:** Generate all combinations of k elements from a list.
110. **Decorators:** Write a Python decorator that logs function calls (function name and arguments).
111. **Memoization (with Decorator):** Implement a memoization decorator for a recursive function (e.g., Fibonacci).
112. **Generators:** Write a generator function that yields Fibonacci numbers.
113. **Iterators:** Create a custom iterator class.
114. **Context Manager:** Write a custom context manager using __enter__ and __exit__ for file handling.
115. **Regular Expressions: Email Validation:** Use regex to validate if a string is a valid email address.
116. **Regular Expressions: Phone Number Extraction:** Extract phone numbers from a given text using regex.
117. **JSON File Handling:** Read data from a JSON file and write a Python dictionary to a JSON file.
118. **CSV File Handling:** Read and write data to a CSV file.
119. **Command Line Arguments:** Write a script that takes command-line arguments using argparse.
120. **Web Scraping (Basic):** Use requests and BeautifulSoup to scrape the title of

a webpage.

121. **Unit Testing:** Write unit tests for a simple function (e.g., add function) using unittest or pytest.
122. **Logging:** Implement basic logging in a Python script using the logging module.
123. **Multithreading:** Create a simple program that uses multiple threads to perform a task concurrently.
124. **Multiprocessing:** Create a simple program that uses multiple processes to perform a task in parallel.
125. **Producer-Consumer Problem:** Implement the producer-consumer problem using threads and queues.
126. **Database Interaction (SQLite):** Create a SQLite database, create a table, insert data, query data, update data, and delete data.
127. **Object Relational Mapping (ORM) Concept:** (No full ORM library, but simulate basic ORM concepts) Design classes that map to database tables and methods to interact with them.
128. **HTTP Requests:** Make a GET request to a public API and parse the JSON response.
129. **Basic Flask App:** Create a "Hello World" web application using Flask.
130. **URL Shortener (Concept):** Implement the core logic for a URL shortener (generating short codes, storing mappings).
131. **Basic Chat Server/Client (Sockets):** Implement a very basic text-based chat server and client using Python's socket module.
132. **Image Manipulation (Pillow):** (If allowed to assume Pillow is installed or just explain the concept) Open an image, resize it, and save it.
133. **PDF Processing (PyPDF2/fitz/PyMuPDF):** (Conceptually) Read text from a PDF, or merge two PDFs.
134. **Email Sender:** Send a simple email using Python's smtplib module.
135. **QR Code Generator:** (Conceptually) Generate a QR code for a given string.
136. **Decorator with Arguments:** Write a decorator that takes arguments.
137. **Abstract Base Classes (ABC):** Define an abstract base class and demonstrate its usage.
138. **Metaclasses (Basic):** Explain the concept of metaclasses and provide a simple example.
139. **Descriptors:** Implement a custom descriptor.
140. **Asyncio (Basic):** Write a simple asynchronous program using asyncio and async/await.
141. **Websocket Client (Concept):** Connect to a public WebSocket echo server and send/receive messages.

142. **Password Generator:** Generate a strong random password based on user criteria (length, inclusion of special chars, numbers).
143. **File Compression/Decompression (zipfile):** Compress and decompress files using the zipfile module.
144. **Data Visualization (Matplotlib/Seaborn basics):** (Conceptually) Plot a simple line graph or bar chart from data.
145. **Numerical Computing (Numpy basics):** (Conceptually) Perform array operations and linear algebra with NumPy arrays.
146. **Pandas Dataframe Basics:** (Conceptually) Create a DataFrame, perform basic data loading, selection, and filtering.
147. **Binary Tree Implementation:** Implement a binary tree with insertion and search operations.
148. **Binary Search Tree (BST):** Implement a Binary Search Tree with insertion, search, and deletion operations.
149. **Linked List:** Implement a singly linked list with insertion, deletion, and traversal.
150. **Queue Implementation:** Implement a queue data structure using a list.