You have studied the core foundations of Python, including data types, control flow, iteration, and basic algorithmic patterns. This knowledge is sufficient to tackle a comprehensive set of introductory problems focusing heavily on **branching concepts** and elementary algorithms.

Here are 50 problem sets designed to reinforce your learning, drawing primarily from foundational Python materials in the sources, particularly those focused on Chapters 1 through 4.

Section 1: Python Basics, I/O, and Data Types (10 Problems)

These problems focus on expressions, input/output, variable assignment, and basic type conversion.

1. **Expression Evaluation:** Evaluate the result of arithmetic expressions that mix operators, for example: `(5 + 2 * 3) / 2 ** 2`.
2. **Identify Types:** Given a set of literals and operators (e.g., `'hello'`, `+`, `5`, `-88.8`), identify which are **operators** and which are **values**.
3. **Variable vs. String:** Determine which of the following is a variable name and which is a string: `spam` and `'spam'`.
4. **String Arithmetic:** Determine what the following expressions evaluate to: `'spam' + 'spamspam'` and `'spam' * 3`.
5. **Fixing Type Errors:** Explain why the expression `'I have eaten ' + 99 + ' burritos.'` causes an error and how to fix it using type conversion.
6. **Basic Input/Output:** Write a program that uses `input()` to ask the user for their name and then prints a personalised greeting.
7. **Calculate Length:** Ask the user to input a phrase and use the built-in **`len()` function** to print the number of characters in the input.
8. **Time Conversion:** Write expressions to calculate the total number of seconds contained in 42 minutes and 42 seconds.
9. **Numeric Input:** Write a program that asks a user for a float value (e.g., a temperature in Celsius) and prints that value converted to an **integer** using `int()`.
10. **Variable Assignment Logic:** Given the code snippet `bacon = 20; bacon + 1`, what is the final value stored in the variable `bacon`?

Section 2: Core Branching Concepts (`if`, `elif`, `else`) (15 Problems)

These problems are essential for mastering conditional execution and Boolean logic.

11. **Age Check (Simple `if`):** Write a program that takes an age as input and prints "You are old enough to vote!" only if the age is 18 or greater.

12. **Age Check (`if-else`):** Modify the above program to use an `if-else` block. If the age is under 18, print "Sorry, you are too young to vote."
13. **Even/Odd Check:** Use the **modulo operator (`%`)** and an `if-else` statement to determine whether an integer input by the user is **even** or **odd**.
14. **Membership Test:** Define a list of `available_toppings`. Ask the user for a `requested_topping`. Use an `if` statement and the `in` operator to check if the request can be fulfilled.
15. **Inequality Check:** Define a list of `banned_users`. Ask the user for their username and use the `!=` (not equal to) operator to check if they are permitted to post a response.
16. **Chained Conditional:** Write code that checks the value of a variable `spam`. If `spam` is 1, print "Hello". If `spam` is 2, print "Howdy". Otherwise, print "**Greetings!**"
17. **Amusement Park Pricing (Chained):** Use an **`if-elif-else` chain** to set the price variable based on age: under 4 is $0, 4 through 17 is $25, and 18 or older is $40.
18. **Senior Discount (Multiple `elif`):** Expand the pricing problem to include a senior discount: anyone 65 or older pays $20 (ensuring the logic runs correctly in order).
19. **Numerical Range (Boolean `and`):** Use the **`and`** keyword to check if a user's age is between 18 and 35, inclusive.
20. **Alternative Conditions (Boolean `or`):** Use the **`or`** keyword to check if a person is either eligible for retirement (age > 65) or still a minor (age < 18).
21. **Boolean Expression Evaluation:** Determine the result of the expression `(5 > 4) and (3 == 5)`.
22. **Logical Negation:** Determine the result of the expression `not (5 > 4)`.
23. **Finding the Largest Odd:** Write a program that examines three variables (`x`, `y`, and `z`) and prints the largest odd number among them. If none are odd, print the smallest value of the three.
24. **Checking List State:** Use a simple `if` statement to check if a list of requested items is currently **empty**. If it is, print a message, such as "We need to find some users!"
25. **Complex Topping Order (Multiple Independent `if`):** Write a program using a **series of independent `if` statements** to process a list of requested pizza toppings, ensuring *all* requested items are checked, even if the first item passes.

Section 3: Iteration, Control Flow, and Algorithms (15 Problems)

These problems focus on `for` and `while` loops, recursion (introductory/conceptual), and application of simple numerical algorithms.

26. **Simple `while` Loop:** Write a `while` loop that initializes a counter to 1 and prints the numbers up to 5, inclusive.
27. **User Quit Loop:** Use a `while` loop that prompts the user for input and stops running only when the user enters the sentinel value **`'quit'`**.

28. **Infinite Loop and `break`:** Rewrite the user quit loop using `while True:` and a **`break` statement** within an `if` block.
29. **Loop Skip (`continue`):** Write a `while` loop that iterates through numbers 1 to 10 but uses the **`continue` statement** within an `if` block to print only the odd numbers.
30. **Range Step:** Use the three-argument version of the `range()` function (`start`, `stop`, `step`) within a `for` loop to list the multiples of 3 between 3 and 30 (inclusive).
31. **Loop Equivalence:** Write a `for` loop that prints numbers 1 to 10, and then write an **equivalent `while` loop** that produces the exact same output.
32. **List Iteration:** Define a list of names. Use a `for` loop and string formatting (`f-strings`) to print a personalised message to each name in the list.
33. **Maximum Search:** Write a loop that accepts 10 integer inputs from the user and keeps track of the **largest odd number** entered. Print a message if no odd number was entered.
34. **Loop with Exception Handling:** Use a `while True` loop and a **`try-except`** block to repeatedly ask the user for an integer, handling `ValueError` if non-numeric input is given, until valid integer input is received.
35. **Exhaustive Enumeration (Cube Root):** Write a script using a loop (exhaustive enumeration) to find the approximate integer **cube root** of a large input integer `x`.
36. **Guess the Number:** Write a short program that uses a loop to implement a "Guess the Number" game, incorporating `random.randint()` and conditional logic to provide hints ("too high," "too low").
37. **Fibonacci (Conceptual):** Explain the difference between an iterative and a recursive function for computing the factorial or Fibonacci sequence in terms of execution flow and memory consumption.
38. **Break/Continue Use:** Explain the conceptual difference between the **`break`** and **`continue`** statements in relation to loop execution.
39. **Sum of Primes (Nested Logic):** Write a program that prints the sum of all **prime numbers** greater than 2 and less than 1000. (Requires a primality test loop nested inside a numerical iteration loop).
40. **Lists with Conditional Loop Logic:** Use a `for` loop over a list of strings, and inside the loop, use an `if` statement to print a warning only if an item has a specific negative characteristic (e.g., a "green pepper" shortage).

Section 4: Strings, Lists, and Functions (10 Problems)

These problems require working with string manipulation and basic list functions.

41. **String Indexing:** Define a string variable (`word`). Use **indexing** (including negative indexing) to print the first character and the last character.
42. **String Slicing:** Use **slicing** to extract a substring composed of the characters from index 2 up to, but not including, index 5.
43. **Case-Insensitive Check:** Write an `if` statement that checks if a user's input matches the secret word 'python', regardless of the input's **case** (e.g., 'PYTHON', 'Python', 'python' should all match).

44. **String Searching:** Use the **`in` operator** within an expression to check if the phrase "Python" is contained within a longer paragraph.
45. **Prefix Check:** Write an `if` statement using the **`startswith()`** string method to determine if a path string begins with "C:\Users".
46. **List Concatenation:** Demonstrate two ways to add elements to the end of an existing list: one method that mutates the original list (e.g., **`append()`**) and one that returns a new list (e.g., **`+` concatenation**).
47. **List Modification by Index:** Given a list `L = ['A', 'B', 'C']`, write a single line of code that replaces the element at index 1 with `'Z'`.
48. **Splitting and Indexing:** Take a string representing a date in "DD-MM-YYYY" format. Use the **`split()` method** to break it apart and use indexing to print the year only.
49. **List Formatting (`.join`):** Take a list of individual words and use the **`.join()` method** to combine them into a single sentence, separated by spaces.
50. **List Containment Logic (Conceptual):** Write a brief plan outlining how you would programmatically check if a given list contains any **duplicate elements** using only basic loops and branching (no use of `set()` or special counting methods allowed).