50 Questions – Variables

1. Declare three variables in one line and assign values 1, 2, and 3. Print their sum.
2. Swap two variables' values without using a temporary variable.
3. Store your name and age in variables and print: `"My name is X and I am Y years old."`
4. Assign the same value to multiple variables in one line and verify they share the same reference using `id()`.
5. Show how variable names are case-sensitive by creating two variables `A` and `a` with different values.
6. Write a program where a variable changes type (int → str → float) and print its type at each step.
7. Demonstrate variable shadowing inside a function vs outside (global vs local).
8. Assign a large integer to a variable and print its memory size using `sys.getsizeof()`.
9. Create variables of all basic data types and print their types in one loop.
10. Change a variable's value based on user input (e.g., increment a counter).
11. Show how Python allows dynamic typing by assigning a string, then a number, then a list to the same variable.
12. Create multiple variables and swap their values circularly (a → b, b → c, c → a).
13. Use tuple unpacking to assign values from a tuple to separate variables.
14. Assign values from a list to separate variables using unpacking.
15. Demonstrate constants in Python using naming conventions (all caps) and explain mutability.
16. Store a lambda function in a variable and call it to perform a calculation.

17. Create a variable whose value is computed from two other variables dynamically (e.g., sum).

18. Implement variable reassignment inside a function using `global` keyword.

19. Demonstrate `nonlocal` keyword with nested functions and variable modification.

20. Check if two variables refer to the same object using `is` operator.

21. Assign a mutable object to two variables and modify via one variable; observe changes in the other.

22. Show variable name rules by attempting invalid names and catching syntax errors.

23. Use `del` keyword to delete a variable and try accessing it afterward.

24. Write a program to create alias variables for an object and prove both names refer to the same object.

25. Track variable reference count using `sys.getrefcount()`.

26. Assign variables inside a loop and check their value outside the loop.

27. Assign value using conditional expression (`x = 5 if condition else 10`).

28. Demonstrate immutability of tuples by attempting to modify them via variables.

29. Create a variable in one module and import it into another module to show accessibility.

30. Create variables with same name in global and local scope and print both values.

31. Use `id()` to prove two identical strings may share the same memory address.

32. Store environment variable value using `os.environ.get()` in a Python variable.

33. Write code to show variable lifetime (global persists, local destroyed after function ends).

34. Assign a function to a variable and call it via that variable.

35. Show multiple assignment to same variable type vs different types and observe effects.

36. Assign variable names dynamically using `globals()` or `locals()`.

37. Create variable names from user input dynamically (dictionary or eval).
38. Show chained variable assignment (`a = b = c = 0`) and prove all reference same value.
39. Create a mutable variable and pass it into multiple functions; modify and observe effects.
40. Assign a default value to variable if undefined using `try/except NameError`.
41. Demonstrate use of f-strings with variables to format output.
42. Assign binary, octal, and hex literals to variables and print their decimal equivalents.
43. Increment and decrement variables manually (Python has no ++/--).
44. Show how reassignment of immutable types creates new object references.
45. Use variables to store function return values and chain them.
46. Demonstrate overwriting built-in variable name (e.g., `list`) and consequences.
47. Create variable with underscore `_` and use it interactively in Python shell context.
48. Use annotated variables with type hints (`age: int = 25`) and verify with `__annotations__`.
49. Show how variable scope works inside comprehensions vs outside.
50. Write program to create variables for constants like PI and use them in calculations.

---

50 Questions – Data Types & Casting

1. Create variables for all primitive data types and print their types.
2. Demonstrate implicit type casting during arithmetic (int + float).
3. Convert string `"123"` to int and add 10 to it.

4. Convert integer 255 to binary, octal, and hexadecimal using built-in functions.
5. Cast a float to int and observe data loss.
6. Cast a string with spaces (`" 123 "`) to int after trimming.
7. Attempt to cast an invalid string to int and handle exception gracefully.
8. Convert boolean `True` to integer and vice versa.
9. Convert a list of strings to a list of integers using `map()`.
10. Convert tuple to list, modify, and convert back to tuple.
11. Convert set to list and sort it.
12. Cast dictionary keys to a list and values to a tuple.
13. Show difference between `str()` and `repr()` when converting objects.
14. Convert integer to string and concatenate with another string.
15. Convert string `"3.14"` to float and multiply by 2.
16. Demonstrate `ord()` and `chr()` conversions (char ↔ Unicode code point).
17. Convert list of tuples to dictionary using casting.
18. Show that casting large float to int truncates rather than rounds.
19. Convert list of integers to single concatenated string (`[1,2,3]` → `"123"`).
20. Convert nested list to flat list via comprehension and casting.
21. Convert boolean list `[True, False, True]` to integers `[1,0,1]`.
22. Convert string `"True"` to actual boolean value safely.
23. Demonstrate `bytes` and `bytearray` type creation and conversion.
24. Convert number to bytes and back using `to_bytes` and `from_bytes`.

25. Convert string to uppercase and check type remains `str`.

26. Convert complex number to string and parse real/imaginary parts.

27. Convert integer timestamp to `datetime` object.

28. Convert `datetime` object back to string in custom format.

29. Cast mutable objects (lists) to immutable (tuple) and attempt modification.

30. Convert keys of dictionary to set and check uniqueness.

31. Show casting of negative numbers to binary using `bin()`.

32. Convert decimal to fraction using `fractions.Fraction`.

33. Convert floating-point number to exact decimal using `decimal.Decimal`.

34. Convert list of numbers to string and back to list of numbers.

35. Show how JSON string can be cast to Python dictionary using `json.loads()`.

36. Cast Python dictionary to JSON string using `json.dumps()`.

37. Convert Python object to string with `str()` vs using `f-string`.

38. Cast `range` object to list and print.

39. Convert a list of mixed types to strings using comprehension.

40. Demonstrate `bool()` conversion on different data types (`0`, `""`, `[]`, `None`).

41. Convert iterable to set to remove duplicates and back to list.

42. Convert number to scientific notation string format.

43. Convert lowercase string to title case (capitalize each word).

44. Convert int to ASCII character if within range using `chr()`.

45. Convert ASCII character to int using `ord()`.

46. Convert float to percentage string (e.g., `0.85 → "85%"`).
47. Show casting using constructor of custom class (`class A: def __init__(self, val)`).
48. Convert string `"1,2,3"` into list of integers `[1,2,3]`.
49. Convert dictionary into list of `(key,value)` tuples and back.
50. Convert Python object to bytes via `pickle` and back.

- **Strings – 50 questions**
- **Lists – 50 questions**
- **Tuples – 50 questions**
- **Sets – 50 questions**
- **Dictionaries – 50 questions**

---

50 Questions – Strings

1. Reverse a string without using slicing (`[::-1]`).
2. Check if two strings are anagrams.
3. Find the first non-repeating character in a string.
4. Count frequency of each character using `collections.Counter`.
5. Remove duplicate characters but keep order.
6. Check if a string is palindrome ignoring case and spaces.

7. Find all substrings of a given string.
8. Replace all spaces with `-` in a string.
9. Capitalize first letter of every word.
10. Remove vowels from a string.
11. Extract digits from a string and return as integer.
12. Count words in a string without using `split()`.
13. Check if string contains only alphabets (no digits/symbols).
14. Implement `strStr()` – return index of first substring occurrence.
15. Find longest common prefix of a list of strings.
16. Find longest palindrome substring.
17. Compress a string (e.g., `"aaabb"` → `"a3b2"`).
18. Expand compressed string (e.g., `"a3b2"` → `"aaabb"`).
19. Count occurrences of each word in a string.
20. Remove punctuation from a string.
21. Convert camelCase to snake_case.
22. Convert snake_case to camelCase.
23. Find smallest and largest word in a string.
24. Check if two strings are rotations of each other.
25. Find first repeated character in a string.
26. Print all permutations of a string.
27. Print all combinations of characters in a string.

28. Validate password (must contain uppercase, lowercase, number, special char).
29. Encode a string using run-length encoding.
30. Decode run-length encoded string.
31. Count consonants and vowels separately.
32. Replace multiple spaces with a single space.
33. Check if string is pangram (contains all alphabets).
34. Find common characters between two strings.
35. Implement `startswith()` and `endswith()` manually.
36. Remove all occurrences of a character from a string.
37. Reverse words in a sentence (word order).
38. Reverse each word in place (but keep order of words).
39. Convert Roman numeral to integer.
40. Convert integer to Roman numeral.
41. Find first index of substring without using `find()` or `index()`.
42. Find length of last word in a string.
43. Remove HTML tags from a string using regex.
44. Count how many times each substring of length 2 appears.
45. Generate all possible valid IP addresses from a string of digits.
46. Find longest substring without repeating characters.
47. Find longest substring with at most 2 distinct characters.
48. Find minimum window substring containing all characters of another string.
49. Implement a simple Caesar cipher for encryption/decryption.

50. Detect if string contains balanced parentheses/brackets.

---

50 Questions – Lists

1. Reverse a list without using `reverse()` or slicing.
2. Find second largest element without sorting.
3. Merge two sorted lists into one sorted list.
4. Rotate list elements by `k` steps to the right.
5. Remove duplicates from list without using `set()`.
6. Find all pairs of numbers whose sum equals target.
7. Flatten a nested list (arbitrary depth).
8. Split a list into chunks of size `n`.
9. Find intersection of two lists.
10. Find union of two lists (remove duplicates).
11. Find missing numbers in a sequence.
12. Partition list into even and odd numbers.
13. Sort list of tuples by second element.
14. Find longest increasing subsequence.
15. Find product of all elements except self (no division).
16. Find equilibrium index (sum left = sum right).
17. Move all zeros to end maintaining order.

18. Find duplicate elements and count their occurrences.
19. Find first repeating element.
20. Find first non-repeating element.
21. Remove `None` values from list.
22. Generate Pascal's triangle up to n rows.
23. Generate all sublists of a list.
24. Rotate list left by one element repeatedly (simulate queue).
25. Merge `n` sorted lists into one sorted list.
26. Find common elements in three sorted lists.
27. Find kth largest and kth smallest element.
28. Check if list is palindrome.
29. Separate positive and negative numbers.
30. Find maximum sum subarray (Kadane's algorithm).
31. Find minimum sum subarray.
32. Count frequency of each element.
33. Implement binary search manually.
34. Implement linear search manually.
35. Find leaders in array (element greater than all to its right).
36. Replace each element with product of previous and next.
37. Rearrange list so that positive and negative numbers alternate.
38. Find first missing positive integer.
39. Sort list of 0, 1, 2 without using sort().

40. Count inversions in list (pairs i<j where a[i]>a[j]).
41. Find longest consecutive sequence.
42. Implement insertion sort manually.
43. Implement selection sort manually.
44. Implement bubble sort manually.
45. Remove elements that appear more than once.
46. Find sum of subarrays of length `k`.
47. Find subarray with sum equal to target.
48. Generate power set of list elements.
49. Implement custom `zip()` function.
50. Implement custom `map()` function using list comprehension.

---

50 Questions – Tuples

1. Reverse a tuple without converting to list.
2. Find element-wise sum of two tuples.
3. Check if tuple is palindrome.
4. Convert nested tuple to flat tuple.
5. Find min and max of tuple without built-ins.
6. Concatenate multiple tuples into one.
7. Slice tuple without using slicing operator.

8. Count occurrences of element in tuple.
9. Find index of element without using `index()`.
10. Convert tuple of tuples to list of lists and back.
11. Sort tuple of numbers in ascending order.
12. Sort tuple of strings by length.
13. Create tuple from dictionary keys.
14. Create tuple from dictionary values.
15. Check if two tuples are equal.
16. Swap elements of two tuples element-wise.
17. Merge tuple into single string.
18. Find common elements between two tuples.
19. Remove duplicates from tuple.
20. Convert list of tuples to tuple of lists.
21. Convert tuple of lists to list of tuples.
22. Group tuple elements by even and odd.
23. Find frequency of each element in tuple.
24. Check if tuple contains only unique elements.
25. Find longest increasing subsequence in tuple.
26. Find subtuple with maximum sum.
27. Extract every second element of tuple.
28. Convert tuple to dictionary with index as key.
29. Convert dictionary to tuple of (key, value) pairs.

30. Find intersection of multiple tuples.

31. Merge sorted tuples into single sorted tuple.

32. Find kth smallest and largest elements.

33. Split tuple into two halves.

34. Rearrange tuple so that negatives come first.

35. Rotate tuple elements by `k` steps.

36. Create tuple comprehension (using generator then convert).

37. Compare memory size of tuple vs list.

38. Count total number of digits across tuple of numbers.

39. Check if tuple is subset of another tuple.

40. Find unique characters in tuple of strings.

41. Convert tuple of characters to single string.

42. Find all possible pairings of tuple elements.

43. Check if tuple contains nested tuple and flatten it.

44. Find tuples with same first element from list of tuples.

45. Combine tuple of keys and tuple of values into dictionary.

46. Partition tuple into chunks of `n`.

47. Sum of diagonal elements if tuple represents matrix.

48. Convert tuple to namedtuple and access attributes.

49. Create tuple with repeated pattern (e.g., `(1,2)` repeated n times).

50. Find difference between two tuples (elements present in first but not in second).

---

50 Questions – Sets

1. Find union, intersection, difference between two sets.
2. Check if one set is subset of another.
3. Find symmetric difference between sets.
4. Remove duplicates from list using set.
5. Find all unique characters in a string.
6. Check if two strings have common characters.
7. Count unique words in a sentence.
8. Find elements in first list but not in second.
9. Check if two sets are disjoint.
10. Convert list of lists to set of tuples.
11. Remove element safely using `discard()` (avoid error).
12. Check if set is superset of another.
13. Generate power set of given set.
14. Find pairs of numbers with sum divisible by `k`.
15. Remove duplicates from nested lists using frozenset.
16. Compare performance of membership test in list vs set.
17. Find duplicates in list using set logic.
18. Use set comprehension to create squares of numbers.
19. Find common words between multiple sentences.
20. Count unique vowels in string.

21. Check if sentence is pangram using set.

22. Create immutable set (frozenset) and try modifying it.

23. Find all unique subsets of string characters.

24. Implement custom intersection logic manually.

25. Convert two lists into set and find symmetric difference.

26. Find union of multiple sets at once.

27. Create set from dictionary keys and values.

28. Remove elements from set based on condition (e.g., even numbers).

29. Find difference between two sets without using `-` operator.

30. Convert set to sorted list.

31. Implement algorithm to check if two arrays are permutations using sets.

32. Find common prime factors of two numbers using sets.

33. Implement simple spell checker using set of dictionary words.

34. Generate all possible pairs from two sets.

35. Find distinct absolute values from list using set.

36. Implement custom subset check using loops (no built-in).

37. Find missing letters from alphabet using set difference.

38. Count distinct digits across list of numbers using sets.

39. Detect duplicates in list by comparing length vs set length.

40. Find intersection size of multiple sets.

41. Remove all vowels from set of characters.

42. Find common divisors of two numbers using set.

43. Combine two sets and remove common elements.
44. Create frozen set of tuples and check membership.
45. Find first unique character in string using set and loop.
46. Find uncommon words between two sentences.
47. Use set comprehension to filter prime numbers in range.
48. Implement manual union of sets using loops.
49. Create nested set (set of frozensets).
50. Find unique elements that appear only once across multiple lists.

---

50 Questions – Dictionaries

1. Merge two dictionaries into one.
2. Sort dictionary by keys.
3. Sort dictionary by values.
4. Find key with maximum value.
5. Invert a dictionary (values become keys).
6. Count frequency of words in sentence using dictionary.
7. Find keys common in two dictionaries.
8. Remove key safely using `pop()` with default value.
9. Create dictionary from two lists (keys and values).
10. Create nested dictionary dynamically.
11. Flatten nested dictionary.

12. Update dictionary with another dictionary.
13. Check if dictionary is subset of another.
14. Find sum of all dictionary values.
15. Multiply all values of dictionary.
16. Remove all keys with value `None`.
17. Group words by their first letter using dictionary.
18. Implement custom `defaultdict` behavior manually.
19. Count character frequency using dictionary comprehension.
20. Convert dictionary to JSON string and back.
21. Find all keys having maximum value.
22. Reverse dictionary mapping (handle duplicate values).
23. Check if two dictionaries are equal.
24. Merge dictionaries using `|` operator (Python 3.9+).
25. Create dictionary of squares using comprehension.
26. Find difference between two dictionaries (keys present in first but not second).
27. Implement `get()` manually with default return.
28. Create dictionary of lists and append values dynamically.
29. Check if key exists and increment its value.
30. Remove all duplicate values in dictionary.
31. Convert dictionary keys to tuple.
32. Convert dictionary values to list.
33. Find intersection of dictionaries (common key-value pairs).

34. Create dictionary of even and odd numbers separately.
35. Implement simple phonebook using dictionary.
36. Count frequency of elements in list using dictionary.
37. Convert nested list of tuples to dictionary.
38. Implement caching (memoization) using dictionary.
39. Merge multiple dictionaries into one.
40. Implement manual `popitem()` method.
41. Filter dictionary by value condition.
42. Create dictionary from string (char count).
43. Convert dictionary to list of tuples and back.
44. Update multiple keys at once in dictionary.
45. Find dictionary difference (symmetric).
46. Create dictionary from sequence using `zip()`.
47. Implement lookup table for Roman numerals.
48. Find common elements across multiple dictionaries by keys.
49. Combine values of same key across two dictionaries.
50. Create immutable mapping using `MappingProxyType`.

Great! Now I'll continue with **Batch 3**, which includes:

- **Booleans + Operators (50 questions combined)**
- **Conditional Statements (50 questions)**
- **Match Statement (50 questions)**

Total = **150 questions** in this batch.

---

50 Questions – Booleans + Operators

1. Write a program that prints the result of all comparison operators (`==, !=, >, <, >=, <=`) between two numbers.
2. Demonstrate the difference between `is` and `==` using integers and lists.
3. Use `and`, `or`, `not` to create a truth table for two boolean variables.
4. Show short-circuit behavior of `and` and `or` operators.
5. Check if a given number is between 10 and 50 using chained comparisons.
6. Implement a custom XOR logic using boolean operators.
7. Count how many conditions out of three are true using boolean logic.
8. Write a program that returns `True` if a string contains both uppercase and lowercase letters.
9. Demonstrate bitwise AND, OR, XOR, NOT on two integers.
10. Check if a number is a power of 2 using bitwise operations.
11. Swap two numbers using XOR bitwise operator.
12. Find whether a given number is odd or even using bitwise operator.
13. Count set bits (1s) in binary representation of a number.
14. Write a program to perform left and right shift operations on integers.
15. Implement custom boolean class overriding `__bool__()` method.
16. Check if two lists share any common element using boolean operators.

17. Use `bool()` conversion to check truthiness of various Python objects (`[], {}, 0, ""`).

18. Create a function that mimics logical AND without using `and` keyword.

19. Create a function that mimics logical OR without using `or` keyword.

20. Implement a toggle switch using boolean value.

21. Check if given number is within a specific range using boolean expressions.

22. Validate if sum of digits of a number is even using boolean operators.

23. Combine three conditions using `and` and `or` in a single expression.

24. Check if two conditions are mutually exclusive.

25. Write program to check De Morgan's law equivalence.

26. Implement a simple binary calculator using bitwise operators.

27. Write a program that flips a boolean value without using `not`.

28. Use `any()` and `all()` functions to test boolean values in a list.

29. Count number of `True` values in a boolean list.

30. Find index of first `True` value in a boolean list.

31. Demonstrate operator precedence between boolean and arithmetic operators.

32. Implement a function to compare two numbers without using comparison operators.

33. Check if integer is multiple of 8 using bitwise `&` operation.

34. Implement a boolean function that checks parity (odd/even) of number using XOR.

35. Demonstrate how `bool` inherits from `int` in Python (`True == 1`).

36. Show result of `True + True`, `True * False`, and explain.

37. Write a program to negate a boolean list.
38. Demonstrate `not not x` to convert value to boolean equivalent.
39. Check if integer has alternating bits (e.g., 1010).
40. Validate if two integers have opposite signs using XOR.
41. Write custom implementation of `nand` and `nor` operators using boolean logic.
42. Show difference between bitwise and logical operators using same input.
43. Combine conditions for password strength validation using booleans.
44. Show usage of walrus operator (`:=`) in boolean expressions.
45. Implement binary AND truth table using nested loops.
46. Implement binary OR truth table using nested loops.
47. Implement binary XOR truth table using nested loops.
48. Use `reduce()` to combine boolean list values with AND logic.
49. Use `reduce()` to combine boolean list values with OR logic.
50. Implement function that checks if count of True values in list is prime.

---

50 Questions – Conditional Statements

1. Check if a number is positive, negative, or zero.
2. Find the largest of three numbers using nested `if-else`.
3. Check if a year is a leap year.
4. Determine grade based on marks (A, B, C…).
5. Check if number is divisible by 3 and 5 but not 15.

6. Print whether a character is vowel or consonant.
7. Check if given age is eligible for voting and driving.
8. Write program to classify triangle (equilateral, isosceles, scalene).
9. Calculate electricity bill based on usage slabs using `if-elif-else`.
10. Find day of week name given number 1–7.
11. Check if string is empty using conditional.
12. Compare lengths of two strings and print which is longer.
13. Check if entered password matches stored password.
14. Determine if entered character is uppercase, lowercase, digit, or special symbol.
15. Print absolute value of number using conditional.
16. Implement program that checks if number is prime using conditional.
17. Implement program that checks if number is Armstrong number.
18. Classify person as child/teen/adult/senior based on age ranges.
19. Determine discount percentage based on total purchase amount.
20. Implement simple calculator with `if-elif-else`.
21. Determine if point lies in which quadrant (x,y plane).
22. Check if number is multiple of another number.
23. Print whether number is single-digit, two-digit, or three-digit.
24. Validate login with multiple conditions (username & password).
25. Determine bonus eligibility based on performance rating.
26. Compare two dates and print which is earlier.
27. Check if year, month, day combination is valid date.

28. Implement rock-paper-scissors game logic with conditionals.
29. Validate time input (hh:mm:ss) using conditions.
30. Check if three lengths can form a triangle.
31. Implement program to find maximum of four numbers.
32. Check if two strings are equal ignoring case.
33. Validate if number lies within specific closed interval.
34. Implement program to classify BMI into categories.
35. Check if given temperature is freezing, moderate, or hot.
36. Check if given character is alphabet, digit, or whitespace.
37. Implement eligibility check for scholarship with multiple criteria.
38. Implement simple menu-driven program using `if-elif-else`.
39. Determine if year is century year or not.
40. Check if number is perfect square using conditional.
41. Print whether three numbers form arithmetic progression.
42. Print whether three numbers form geometric progression.
43. Check if entered password meets length and character conditions.
44. Classify employee tax bracket based on salary slabs.
45. Implement logic to print smallest and largest of three numbers.
46. Validate marks input (0–100) and print pass/fail.
47. Determine if string is palindrome using conditional only.
48. Print name of month given month number.
49. Classify blood pressure readings into categories.

50. Implement a basic traffic light simulation using conditionals.

---

50 Questions – Match Statement (Python 3.10+)

1. Implement calculator using `match` with operations `+`, `-`, `*`, `/`.
2. Map day number to weekday name using `match`.
3. Map month number to season (winter, summer, etc.).
4. Classify character type (vowel, consonant, digit, special) using `match`.
5. Implement menu-driven program for restaurant ordering using `match`.
6. Create program to match HTTP status codes (200, 404, 500).
7. Implement simple grading system using `match`.
8. Check if number is even or odd using `match`.
9. Print corresponding zodiac sign based on month/day using `match`.
10. Match file extensions to file types (e.g., `.txt` → Text file).
11. Match integer to Roman numeral (1–10).
12. Implement switch-like logic for traffic light colors.
13. Map key presses (WASD) to movement directions.
14. Implement a program to check multiple ranges using `match` with guards.
15. Use `match` to classify age groups.
16. Implement currency conversion menu using `match`.
17. Implement discount calculation based on customer type.

18. Implement temperature classification using `match`.
19. Map input number (1–12) to month name using `match`.
20. Match string commands to function calls.
21. Implement login system role-based using `match` (admin/user/guest).
22. Create simple chatbot response logic using `match`.
23. Use `match` to handle basic geometric shape area calculation.
24. Implement multiple cases falling through to same block.
25. Implement nested `match` statements (submenus).
26. Match tuple of coordinates to check quadrant.
27. Use `match` with sequence pattern to match list structure.
28. Implement program to validate file permissions using `match`.
29. Handle multiple possible inputs for same action using `|` (OR patterns).
30. Implement match case for math operations using symbols.
31. Match string to detect palindrome vs non-palindrome.
32. Map planet names to their distance from sun using `match`.
33. Implement simple bank ATM menu using `match`.
34. Match input against multiple data types (int, str, bool).
35. Implement number spelling (1 → one, 2 → two) using `match`.
36. Use `match` to detect basic errors (404, 403, etc.).
37. Implement dice roll result classification using `match`.
38. Implement match for days left until weekend logic.

39. Implement system command handler using `match`.
40. Match tuple pattern `(x,y)` to classify points on axis or origin.
41. Use `match` with dictionary unpacking patterns.
42. Implement recursive menu navigation using `match`.
43. Match specific substrings to commands using `match`.
44. Implement rock-paper-scissors logic using `match`.
45. Create custom routing system simulation using `match`.
46. Implement program to classify animals (mammal, bird, reptile).
47. Use match with pattern guards (`if condition`) for range checks.
48. Map numerical grade to letter grade using `match`.
49. Implement `match` case for multiple exception messages.
50. Create program to identify shape type by number of sides using `match`.

Batch 4

- **Functions (50 questions)**
- **Higher-Order Functions (50 questions)**
- **Iterators + Scope (50 questions combined)**

---

50 Questions – Functions

1. Write a function that returns the factorial of a number.

2. Create a function that checks if a string is a palindrome.
3. Write a function to find the nth Fibonacci number using recursion.
4. Implement a function with default arguments and call it with and without them.
5. Write a function that returns multiple values and unpack them.
6. Write a function that accepts arbitrary number of arguments using `*args`.
7. Write a function that accepts arbitrary keyword arguments using `**kwargs`.
8. Implement a function to check if a number is prime.
9. Write a function to compute the sum of all even numbers in a list.
10. Create a function that filters out vowels from a string.
11. Write a recursive function to compute the sum of digits of a number.
12. Write a function that accepts another function as argument and applies it to a list.
13. Write a function that returns a lambda function to add a fixed number.
14. Implement a function to reverse a list without using built-in functions.
15. Write a function that counts the number of words in a sentence.
16. Create a function that merges two dictionaries.
17. Write a function to check if two strings are anagrams.
18. Implement a function that calculates compound interest.
19. Write a function to convert Celsius to Fahrenheit.
20. Write a function that returns the largest element in a list.
21. Write a function that takes a string and returns the most frequent character.
22. Implement a function to calculate the greatest common divisor (GCD) of two numbers.
23. Write a function that flattens a nested list by one level.

24. Write a function to remove duplicates from a list while preserving order.
25. Write a function to check if a string contains balanced parentheses.
26. Write a function that calculates the area of a circle given its radius.
27. Write a function that accepts a list of numbers and returns a list of their squares.
28. Write a function to check if a number is Armstrong number.
29. Write a function to generate a list of prime numbers up to n.
30. Write a function that calculates the length of a string without using `len()`.
31. Write a function that takes a list and returns a dictionary with element counts.
32. Write a function that swaps two variables using tuple unpacking.
33. Write a function that returns the Fibonacci sequence up to n terms as a list.
34. Write a function to calculate the sum of digits of an integer.
35. Write a function to convert a decimal number to binary as a string.
36. Write a function to check if a number is a perfect square.
37. Write a function to generate a multiplication table for a given number.
38. Write a function to find all divisors of a number.
39. Write a function to remove all whitespace from a string.
40. Write a function that returns True if a list is sorted in ascending order.
41. Write a function that returns the second largest number in a list.
42. Write a function that computes the power of a number without using `**` or `pow()`.
43. Write a function that accepts a string and returns a dictionary with character frequencies.
44. Write a function to rotate a list left by k positions.
45. Write a function that converts a Roman numeral to integer.

46. Write a function that calculates the sum of digits raised to the power of their position.
47. Write a function that counts vowels and consonants in a string.
48. Write a function that checks if two lists have at least one common element.
49. Write a function that merges two sorted lists into one sorted list.
50. Write a function that finds the longest word in a list of strings.

---

50 Questions – Higher-Order Functions

1. Use `map()` to convert a list of strings to their lengths.
2. Use `filter()` to extract even numbers from a list.
3. Use `reduce()` to calculate the product of all elements in a list.
4. Write a function that returns another function which adds a fixed number to its argument.
5. Use `map()` with a lambda to convert a list of temperatures from Celsius to Fahrenheit.
6. Use `filter()` with a function to remove vowels from a list of characters.
7. Write a higher-order function that takes a function and applies it twice to a value.
8. Use `reduce()` to concatenate a list of strings into a single string.
9. Use `map()` to square all elements in a list of numbers.
10. Use `filter()` to select strings longer than 5 characters.
11. Use `reduce()` to find the maximum number in a list.
12. Write a function that returns a lambda function for exponentiation with fixed exponent.
13. Use `map()` and `filter()` together to find squares of even numbers only.

14. Write a higher-order function that times the execution of another function.
15. Use `filter()` to remove None values from a list.
16. Write a function that composes two functions.
17. Use `map()` with a built-in function like `str.upper`.
18. Use `reduce()` to calculate factorial of a number.
19. Write a higher-order function that caches the result of an expensive function.
20. Use `filter()` to remove duplicates from a list using a helper function.
21. Use `map()` to convert integers to their hexadecimal string representation.
22. Use `reduce()` to sum numbers but stop early if sum exceeds a threshold (custom logic).
23. Write a function that returns a function which multiplies its input by a given number.
24. Use `filter()` to keep only palindromic strings from a list.
25. Use `map()` and `lambda` to add 5 to each element of a list.
26. Write a function that takes a list of functions and applies them sequentially to a value.
27. Use `reduce()` to find the longest string in a list.
28. Use `filter()` with a lambda that checks string length is even.
29. Write a higher-order function that logs input and output of another function.
30. Use `map()` to extract first character of each string in a list.
31. Use `filter()` to select prime numbers from a list.
32. Write a function that takes a function and returns a memoized version of it.
33. Use `reduce()` to flatten a list of lists.

34. Write a higher-order function that repeats execution of another function `n` times.
35. Use `map()` with a function that returns True if string starts with 'a'.
36. Use `filter()` to remove strings containing digits.
37. Write a function that takes another function and applies it conditionally.
38. Use `reduce()` to combine dictionaries into one.
39. Use `map()` with a lambda to convert temperatures from Fahrenheit to Celsius.
40. Write a higher-order function that adds logging to any function.
41. Use `filter()` to find elements greater than the average of a list.
42. Use `map()` to extract last character of each string.
43. Write a function that returns a function to check if number is divisible by n.
44. Use `reduce()` to calculate the greatest common divisor of a list of numbers.
45. Use `filter()` and `map()` to transform and filter lists in pipeline.
46. Write a higher-order function that retries execution on exception.
47. Use `map()` to convert list of booleans to integers.
48. Use `filter()` to select tuples where second element is even.
49. Write a function that composes multiple functions into one.
50. Use `reduce()` to compute the sum of digits of a number repeatedly until single digit.

---

50 Questions – Iterators + Scope

1. Create a custom iterator that returns even numbers up to n.

2. Write a generator function that yields Fibonacci numbers.
3. Implement an iterator class that iterates over a list in reverse order.
4. Use `iter()` and `next()` to manually iterate over a list.
5. Write a generator that yields prime numbers indefinitely.
6. Implement an iterator for a binary tree (in-order traversal).
7. Create a generator to yield all substrings of a string.
8. Write a function using generator expression to produce squares of numbers.
9. Implement a class with iterator protocol (`__iter__` and `__next__`).
10. Use generator to read a large file line-by-line.
11. Write code to demonstrate difference between global and local scope.
12. Use `nonlocal` keyword in nested functions to modify variable.
13. Explain LEGB rule with code examples.
14. Write a function that modifies a global variable using `global` keyword.
15. Show variable shadowing inside functions and classes.
16. Write code to demonstrate closures capturing variables.
17. Use function default argument as closure example.
18. Write a generator that yields infinite sequence but stops after some condition externally.
19. Explain what happens if `StopIteration` is not handled.
20. Write a function that uses generator and `send()` method.
21. Demonstrate generator delegation using `yield from`.
22. Write an iterator that cycles through a list infinitely.

23. Write code showing scope of variables in list comprehensions.

24. Write a recursive generator that yields factorial values.

25. Explain scope of variables in nested classes.

26. Write code to show effect of modifying mutable default arguments.

27. Write function with local variable having same name as global variable.

28. Write code to demonstrate difference between class variables and instance variables.

29. Write a generator expression to filter even numbers from a list.

30. Write a function that returns a generator expression.

31. Show how generator can maintain state between calls.

32. Demonstrate use of `global` in nested scopes.

33. Write code to demonstrate name mangling with double underscores.

34. Write a generator to iterate over lines in a file that contain a specific word.

35. Show how to manually close a generator and handle `GeneratorExit`.

36. Write a function that demonstrates variable lifetime.

37. Explain the effect of `nonlocal` in closures with nested functions.

38. Write code to demonstrate variable access in comprehensions.

39. Write code showing difference in variable lookup between locals and globals.

40. Write a function that creates and returns a closure.

41. Write a generator that produces infinite primes with sieve of Eratosthenes.

42. Use iterator protocol to iterate over custom data structure.

43. Write code showing how nested functions access outer variables.

44. Write a generator that yields elements from multiple iterables in round robin.

45. Show difference between iterator and iterable with examples.
46. Write a function demonstrating `nonlocal` modifying variable multiple levels up.
47. Explain use of `global` and `nonlocal` in nested classes.
48. Write code that shows scope issues in list comprehensions in Python 2 vs 3.
49. Write generator to produce permutations of a string.
50. Demonstrate scope of variables inside try-except blocks.

Batch 5: Object-Oriented Programming (OOP)

- **Classes and Objects (50 questions)**
- **Inheritance (50 questions)**
- **Polymorphism (50 questions)**

---

50 Questions – Classes and Objects

1. Define a class `Person` with attributes `name` and `age`. Create an object and print attributes.
2. Add a method `greet()` to the `Person` class that prints a greeting message.
3. Implement a class `Rectangle` with attributes `width` and `height` and a method to calculate area.
4. Add a constructor (`__init__`) to initialize `Rectangle` attributes.
5. Implement a class method to return number of instances created.
6. Write a class `Circle` with a class attribute `pi` and instance attribute `radius`.

7. Override `__str__` method in `Circle` to print radius and area.
8. Implement getters and setters using `@property` decorator for `Rectangle`.
9. Create a class `BankAccount` with methods `deposit()`, `withdraw()`, and `get_balance()`.
10. Add a private attribute to `BankAccount` and access it using methods.
11. Implement static method in a class that returns current date.
12. Use `__del__` destructor method to print a message when object is deleted.
13. Implement a `Counter` class that counts how many times a method is called.
14. Write a class with class-level attribute shared by all instances.
15. Create two classes `Point2D` and `Point3D` with inheritance later.
16. Implement method overloading by default parameters in a class.
17. Write a class that tracks all its instances in a class-level list.
18. Create a class `Employee` with class attribute `raise_amount` and method to apply raise.
19. Override `__repr__` method to provide unambiguous string representation.
20. Write a class that implements addition of two objects using `__add__`.
21. Implement a class `Car` with methods to start, stop, and display speed.
22. Create a class `Student` that keeps track of grades and calculates average.
23. Write a class with method that raises custom exceptions on invalid input.
24. Create a class that supports item access with `__getitem__`.
25. Write a class implementing iterator protocol to iterate over a range.
26. Implement a class with a class variable counting total instances.

27. Write a class method to create an object from a string (alternative constructor).

28. Create a class `Temperature` that converts Celsius to Fahrenheit and vice versa.

29. Implement a class to simulate a stack with push and pop methods.

30. Write a class that implements context manager protocol (`__enter__` and `__exit__`).

31. Create a class that prevents adding new attributes dynamically (`__slots__`).

32. Write a class that supports comparison operators (`__eq__`, `__lt__`).

33. Create a class that stores RGB color and converts to HEX.

34. Write a class that counts occurrences of each word in text.

35. Implement a singleton pattern using a class.

36. Create a class with a method that modifies a class attribute.

37. Write a class to represent a library book with borrow and return methods.

38. Implement a `Person` class with private attributes and public getter/setter.

39. Create a class that caches results of expensive calculations.

40. Write a class that logs method calls using a decorator inside class.

41. Implement a class that overloads `__len__` to return number of elements.

42. Write a class that implements multiplication of objects (`__mul__`).

43. Implement a class with a static method to validate email format.

44. Write a class that uses composition by including another class as attribute.

45. Implement a class that simulates a simple stopwatch.

46. Create a class that keeps track of total area of all shape instances.

47. Write a class that supports item assignment (`__setitem__`).

48. Implement a class that converts between metric and imperial units.
49. Write a class with a method to serialize object to JSON.
50. Implement a class method that returns all instance attributes as a dictionary.

---

50 Questions – Inheritance

1. Implement base class `Animal` and derived class `Dog` with overridden method `speak()`.
2. Create a class `Vehicle` and subclass `Car` that inherits and adds new attributes.
3. Implement multiple inheritance with classes `Flyable` and `Swimmable`.
4. Use `super()` to call base class constructor from derived class.
5. Override a method in derived class and call base class method inside it.
6. Create abstract base class `Shape` with abstract method `area()`.
7. Implement concrete subclasses `Circle` and `Square` of `Shape`.
8. Write a class hierarchy for employees: `Employee` base, `Manager` derived with extra perks.
9. Demonstrate method resolution order (MRO) with diamond inheritance problem.
10. Use mixins to add logging functionality to multiple unrelated classes.
11. Override `__str__` in subclass to extend base class string representation.
12. Implement class with protected attributes (by convention with underscore).
13. Show how derived class can add new methods without affecting base class.
14. Create a subclass that changes default behavior of base class method.

15. Implement a `Person` base class and subclass `Student` that adds grades.
16. Demonstrate inheritance from built-in types like `list` or `dict`.
17. Write a class that inherits from `Exception` and customize error message.
18. Create a base class with a class variable and show how derived classes share it.
19. Implement method overriding with different argument signatures.
20. Create a subclass that disables a method from base class by raising exception.
21. Implement polymorphic behavior in inheritance hierarchy.
22. Use `isinstance()` to check object type in inheritance.
23. Create a `Shape` base class with common attributes and derived classes with specific attributes.
24. Show difference between class variables and instance variables in inheritance.
25. Write a base class with static methods and override them in subclass.
26. Implement private variables using double underscore and show name mangling.
27. Demonstrate extending base class method with additional functionality.
28. Create base class with a method that raises `NotImplementedError` and subclasses implement it.
29. Write a class that inherits from two parent classes and calls both constructors.
30. Use `super()` in multi-level inheritance.
31. Write a subclass that modifies a class attribute inherited from base class.
32. Show how `issubclass()` works with custom classes.
33. Create a base class with a protected method and override it in subclass.
34. Implement a vehicle class hierarchy demonstrating inheritance of attributes and methods.

35. Write a subclass that uses composition to delegate some functionality.
36. Create a subclass that calls base class method conditionally.
37. Demonstrate how to use `@classmethod` in inheritance and override it.
38. Create a subclass that inherits from immutable built-in type (tuple).
39. Implement an inheritance hierarchy to simulate geometric shapes with area and perimeter.
40. Show effect of overriding `__init__` without calling `super()`.
41. Demonstrate class attribute shadowing in subclass.
42. Create base class with method that uses another method overridden in subclass (template method pattern).
43. Implement multiple inheritance with mixins adding different features.
44. Write a base class that counts instances and subclass that inherits count.
45. Demonstrate inheritance with dataclasses.
46. Write a subclass that changes the behavior of `__str__` and `__repr__`.
47. Implement base class with `__call__` and subclass overriding it.
48. Write a base class with properties and override them in subclass.
49. Demonstrate how exceptions propagate in inheritance hierarchies.
50. Create a subclass that restricts or validates inherited attributes on set.

---

50 Questions – Polymorphism

1. Write a function that takes objects of different classes but calls same method on all (duck typing).

2. Implement polymorphism with a base class `Animal` and subclasses `Dog`, `Cat` overriding `speak()`.
3. Use polymorphism to calculate area for different shape objects in a list.
4. Show how method overriding implements runtime polymorphism.
5. Write code that demonstrates polymorphism using abstract base classes.
6. Use polymorphism to process different types of payment methods.
7. Write polymorphic code that serializes objects differently depending on class.
8. Implement function that calls `.draw()` on various graphic objects.
9. Demonstrate polymorphism with operator overloading.
10. Write polymorphic class hierarchy for employees with different salary calculations.
11. Show how polymorphism allows replacing subclass object where base class expected.
12. Implement polymorphism using interfaces (ABC module).
13. Demonstrate polymorphism with methods that take variable arguments.
14. Write a function that accepts different data structures (list, tuple, set) and processes them uniformly.
15. Use polymorphism to implement different sorting strategies.
16. Implement polymorphism in exception handling.
17. Demonstrate polymorphism in method calls across different classes.
18. Show polymorphism by overriding `__str__` for different classes.
19. Use polymorphism to write flexible logging system supporting different log handlers.
20. Demonstrate polymorphic serialization for JSON, XML, and CSV formats.
21. Use polymorphism in visitor pattern example.
22. Write code that uses polymorphism for UI widget rendering.

23. Implement polymorphism in database ORM models.

24. Use polymorphism to handle different file formats in one function.

25. Write polymorphic arithmetic class hierarchy.

26. Implement polymorphic behavior in networking protocols.

27. Use polymorphism in game character actions.

28. Demonstrate polymorphism with class methods and static methods.

29. Show polymorphism in Python's built-in container types.

30. Use polymorphism to handle different types of shapes in a drawing app.

31. Implement polymorphism for discount strategies in a sales app.

32. Write polymorphic code for processing different sensor inputs.

33. Show polymorphism in logging messages of different severity.

34. Use polymorphism to create interchangeable data parsers.

35. Write a function that uses polymorphism to aggregate data from various objects.

36. Demonstrate polymorphism with plugins architecture.

37. Use polymorphism to simulate different transport methods (car, bike, bus).

38. Write polymorphic code for authentication strategies (OAuth, JWT, Basic).

39. Show polymorphism in arithmetic with fractions, decimals, and integers.

40. Implement polymorphism in a task scheduler handling different task types.

41. Use polymorphism to customize report generation.

42. Write code that uses polymorphism in exception hierarchy.

43. Demonstrate polymorphism in serialization/deserialization methods.

44. Use polymorphism for different notification methods (email, SMS, push).

45. Write polymorphic class to handle different image file formats.
46. Show polymorphism in shape collision detection in games.
47. Implement polymorphism in payment gateways integration.
48. Write polymorphic code to convert units in physics calculations.
49. Use polymorphism for customizable input validation.
50. Demonstrate polymorphism in machine learning pipeline steps.

- **Modules (50 questions)**
- **Date & Time (50 questions)**
- **Math (50 questions)**
- **Regex (50 questions)**
- **Try/Except (Exception Handling) (50 questions)**
- **User Input (50 questions)**

---

50 Questions – Modules

1. Import and use the `math` module to calculate square root of a number.
2. Write a script that imports `random` and generates 5 random integers.
3. Create a module `calculator.py` with functions `add()`, `subtract()`. Import and use them in another script.
4. Use `from module import function` syntax and call the function.

5. Import a module using alias with `import math as m` and use it.
6. Write code to list all functions and attributes of a module using `dir()`.
7. Use `sys` module to get command line arguments.
8. Write a program that imports only specific functions from a module.
9. Create a module with a variable and access it from another file.
10. Explain and demonstrate the difference between `import module` and `from module import *`.
11. Use `os` module to get current working directory.
12. Write code to check if a module is installed and import it dynamically.
13. Use `math` module's `factorial()` function.
14. Write a module with a class and import it.
15. Use `datetime` module inside a custom module.
16. Create a module and demonstrate `if __name__ == "__main__"` block.
17. Use `random.choice()` to select a random element from a list.
18. Write code to reload a module using `importlib.reload()`.
19. Use `pickle` module to serialize and deserialize an object.
20. Write a script that uses `collections` module's `Counter` class.
21. Import and use `json` module to parse JSON string.
22. Create a package with multiple modules and import between them.
23. Write a module with a function that raises custom exception.
24. Use `math` module constants like `pi` and `e`.

25. Write code to dynamically import a module by name string.
26. Use `time` module to measure execution time of code block.
27. Use `pathlib` module to manipulate file paths.
28. Write a script to list all files in a directory using `os` module.
29. Create a module that uses functions from another custom module.
30. Use `functools` module to memoize a recursive function.
31. Use `argparse` module to parse command-line options.
32. Write code to check Python version using `sys` module.
33. Use `subprocess` module to execute an external command.
34. Write a module that contains constants and import it.
35. Use `heapq` module to implement a priority queue.
36. Use `statistics` module to calculate mean, median, and mode.
37. Write a module that exports a list and a dictionary.
38. Use `urllib` module to fetch contents of a web page.
39. Write a module with function to convert temperature units.
40. Use `re` module inside a custom module for pattern matching.
41. Create a module with private variables (prefix with underscore).
42. Use `dataclasses` module to create simple data classes.
43. Write a module with generator function and import it.
44. Use `itertools` module to create combinations of a list.
45. Write a script to demonstrate `__all__` variable in a module.

46. Use `email` module to parse email messages.
47. Write a module with a function to log messages to a file.
48. Use `decimal` module for precise decimal arithmetic.
49. Write a module that handles file compression using `zipfile`.
50. Use `uuid` module to generate unique identifiers.

---

50 Questions – Date & Time

1. Get current date and time using `datetime.now()`.
2. Extract year, month, day from current date.
3. Format date as `YYYY-MM-DD`.
4. Parse a string into a `datetime` object.
5. Calculate difference between two dates.
6. Add 10 days to current date.
7. Convert timestamp to datetime and vice versa.
8. Use `time` module to get current time in seconds since epoch.
9. Use `strftime()` to format date in different styles.
10. Use `strptime()` to parse date from string.
11. Create a `timedelta` object representing 1 week.
12. Calculate age in years given birthdate.
13. Get day of week for a given date.

14. Convert datetime to UTC and back to local time.
15. Use `calendar` module to print a month's calendar.
16. Find out if a year is a leap year using `calendar` module.
17. Use `dateutil` (third-party) to parse fuzzy dates.
18. Calculate number of days between two dates.
19. Get current time with timezone info using `pytz` or `zoneinfo`.
20. Write a countdown timer using `time.sleep()`.
21. Display current time in 12-hour format with AM/PM.
22. Convert between different time zones.
23. Use `datetime` to calculate next Monday from today.
24. Get week number of the year for a date.
25. Extract hour, minute, second from current time.
26. Use `timeit` module to time execution of a function.
27. Format datetime to ISO 8601 string.
28. Calculate business days between two dates.
29. Get timestamp for the start of the day.
30. Use `time` module to measure elapsed time with high precision.
31. Parse and format date strings with milliseconds.
32. Write function to check if date is weekend.
33. Create a recurring event every first Monday of the month.
34. Convert datetime to Unix timestamp.

35. Use `datetime.combine()` to merge date and time objects.
36. Get current time in different locales.
37. Use `date` object to create calendar events.
38. Write code to calculate the number of weeks in a year.
39. Use `datetime` and `timedelta` to simulate a countdown.
40. Create a function that returns time elapsed in human-readable format.
41. Use `calendar` module to find all Fridays in a month.
42. Display time elapsed since a specific event.
43. Use `zoneinfo` module to convert time zones (Python 3.9+).
44. Calculate age from birthdate string input.
45. Parse ISO 8601 formatted datetime strings.
46. Write code to check if two dates are in the same month.
47. Create datetime object for a leap second.
48. Write a function to get the last day of the current month.
49. Calculate duration between two `datetime` objects in hours and minutes.
50. Use `time` and `datetime` to simulate stopwatch functionality.

---

50 Questions – Math

1. Use `math.sqrt()` to compute square root of a number.
2. Calculate factorial using `math.factorial()`.

3. Use `math.gcd()` to find greatest common divisor.
4. Calculate sine, cosine, and tangent of an angle in radians.
5. Convert degrees to radians and vice versa.
6. Use `math.log()` and `math.log10()` to compute logarithms.
7. Use `math.pow()` to compute powers.
8. Use `math.ceil()` and `math.floor()` to round numbers.
9. Calculate combinations and permutations using `math.comb()` and `math.perm()`.
10. Use `math.isclose()` to compare floating-point numbers.
11. Write code to generate random floats in a range using `random.uniform()`.
12. Calculate the hypotenuse of a right triangle using `math.hypot()`.
13. Use `math.copysign()` to copy sign from one number to another.
14. Write code that calculates e^x using `math.exp()`.
15. Use `math.fabs()` to get absolute value as float.
16. Generate random integers within a range using `random.randint()`.
17. Use `random.choice()` to select a random element from a list.
18. Use `random.shuffle()` to shuffle a list in place.
19. Write code to round to n decimal places using `round()`.
20. Calculate arc sine and arc cosine using `math.asin()` and `math.acos()`.
21. Use `math.tan()` to find tangent of an angle.
22. Generate Gaussian distributed random numbers with `random.gauss()`.

23. Use `math.modf()` to separate fractional and integer parts of a float.
24. Write code to generate a random sample without replacement using `random.sample()`.
25. Use `math.trunc()` to truncate float to integer.
26. Calculate the factorial of a large number efficiently.
27. Use `math.prod()` to compute product of all elements in an iterable.
28. Write code to calculate distance between two points using `math.dist()`.
29. Use `random.getrandbits()` to generate random bits.
30. Write a program that simulates rolling a dice.
31. Calculate the cube root of a number.
32. Use `math.frexp()` and `math.ldexp()` to manipulate floating point numbers.
33. Use `random.seed()` to make random numbers reproducible.
34. Write code to calculate nth root of a number.
35. Calculate the exponential moving average of a list of numbers.
36. Use `math.isfinite()` to check if number is finite.
37. Write code to find the minimum and maximum in a list using built-in functions.
38. Use `random.betavariate()` to generate beta distribution numbers.
39. Calculate the natural logarithm base e.
40. Use `math.nextafter()` to find the next floating point value after a number.
41. Implement a random walk simulation.
42. Use `math.comb()` to calculate binomial coefficients for probability.

43. Calculate standard deviation using `statistics` module.
44. Generate random integers with weights using `random.choices()`.
45. Use `math.gamma()` function for advanced factorial calculations.
46. Write code to compute dot product of two vectors.
47. Use `random.expovariate()` for exponential distribution.
48. Calculate distance on Earth between two lat/lon points using haversine formula.
49. Write code to normalize a vector.
50. Use `math.log2()` to compute logarithm base 2.

---

50 Questions – Regex

1. Write a regex to validate an email address.
2. Extract all phone numbers from a text string.
3. Check if a string contains only digits using regex.
4. Replace all whitespace characters with a single space.
5. Find all words starting with capital letters.
6. Split a string by commas or semicolons.
7. Match URLs starting with http or https.
8. Validate a date string in `YYYY-MM-DD` format.
9. Extract hashtags from a tweet.
10. Check if a password contains at least one digit, uppercase, lowercase, and special character.

11. Write a regex to match IPv4 addresses.
12. Replace multiple consecutive punctuation marks with a single one.
13. Extract all HTML tags from a string.
14. Match words that end with “ing”.
15. Validate US zip codes (5 digits or 5-4 digits).
16. Write regex to match a valid username (letters, digits, underscores, 3–16 chars).
17. Find all email domains from a list of emails.
18. Use regex to remove all non-alphanumeric characters from a string.
19. Extract all capitalized words not at the beginning of a sentence.
20. Match hexadecimal color codes like `#A3F9C8`.
21. Replace all dates in text with a standardized format.
22. Validate MAC addresses.
23. Extract all words that have double letters (e.g., “letter”).
24. Write a regex that matches empty lines.
25. Extract all numbers with optional decimal points.
26. Validate credit card numbers with regex.
27. Write regex to check if string starts and ends with the same character.
28. Extract all substrings enclosed in quotes.
29. Replace all tabs with four spaces.
30. Match valid Python variable names using regex.
31. Extract all email addresses ignoring case.
32. Write a regex that matches palindrome strings (bonus challenge).

33. Extract all words of length 4 or more.
34. Validate time strings in HH:MM format.
35. Use regex to count number of sentences in a text.
36. Replace all URLs in a string with `<URL>`.
37. Extract domain names from URLs.
38. Match floating point numbers including scientific notation.
39. Validate password strength with minimum length and character classes.
40. Extract all acronyms (all uppercase letters).
41. Write regex to find duplicate words consecutively repeated.
42. Replace email usernames with anonymized string.
43. Extract all dates in DD/MM/YYYY or MM-DD-YYYY formats.
44. Write a regex that matches balanced parentheses (bonus challenge).
45. Extract all hashtags and mentions from social media posts.
46. Validate ISBN-10 and ISBN-13 numbers.
47. Replace multiple spaces with single space.
48. Match multiline comments in code.
49. Extract all words starting with vowels.
50. Write a regex to match valid floating point numbers only.

---

50 Questions – Try/Except (Exception Handling)

1. Write a try-except block that catches division by zero error.
2. Handle file not found exception when opening a file.
3. Write code that catches multiple exceptions in one block.
4. Use `else` block with try-except and explain its use.
5. Use `finally` block to clean up resources.
6. Raise a custom exception and handle it.
7. Write a function that validates input and raises `ValueError` if invalid.
8. Demonstrate nested try-except blocks.
9. Catch exceptions and print error messages.
10. Use `assert` to check conditions.
11. Write a function that retries operation on failure using exceptions.
12. Handle `KeyboardInterrupt` gracefully.
13. Create custom exception class inheriting from `Exception`.
14. Log exceptions to a file in except block.
15. Write code that re-raises an exception after handling.
16. Use `with` statement and catch exceptions inside context manager.
17. Catch and handle `IndexError` and `KeyError`.
18. Demonstrate exception chaining with `from` keyword.
19. Use exception to validate user input in a loop.
20. Write code to catch exception but ignore it (pass).
21. Catch and handle `TypeError` in a function.

22. Handle exceptions in list comprehensions (hint: use helper function).

23. Write a decorator that handles exceptions for any function.

24. Use `try-except` inside a generator function.

25. Handle exceptions in multi-threaded code.

26. Write a function that raises `NotImplementedError`.

27. Handle exceptions raised by external library calls.

28. Use `contextlib.suppress` to ignore specific exceptions.

29. Handle `ZeroDivisionError` and ask user to re-enter value.

30. Write a program that catches all exceptions but logs them.

31. Demonstrate difference between `Exception` and `BaseException`.

32. Catch `AttributeError` and fix missing attribute dynamically.

33. Use exception to implement fallback mechanism.

34. Write code that catches exceptions in nested function calls.

35. Use `try-except` to validate JSON parsing.

36. Handle exceptions when connecting to a network socket.

37. Use exceptions to break out of deeply nested loops.

38. Write a context manager class that handles exceptions on enter and exit.

39. Demonstrate raising exceptions in property setters.

40. Catch exceptions during arithmetic operations.

41. Use `finally` to close database connection.

42. Write code to catch exception and retry operation 3 times.

43. Handle `ImportError` when dynamically importing modules.
44. Write a function that logs exception traceback.
45. Catch and handle exceptions raised by recursion depth exceeded.
46. Handle exceptions when reading user input from console.
47. Write unit tests that expect exceptions.
48. Use `sys.exc_info()` to get exception details.
49. Catch exceptions when parsing command line arguments.
50. Use `warnings` module to raise and handle warnings as exceptions.

---

50 Questions – User Input

1. Write a program to take integer input from user and print it.
2. Accept a string input and print its length.
3. Take floating point input and round it to 2 decimal places.
4. Write code to handle invalid integer input with error message.
5. Take multiple inputs in one line separated by spaces and convert to list of integers.
6. Accept user input until user types ‘exit’.
7. Take input for name and age and print formatted message.
8. Take password input (masking input is bonus).
9. Write code that prompts for a date in YYYY-MM-DD format and validates it.
10. Accept multiple strings and store them in a list.

11. Use `input()` to take expression as string and evaluate it safely.

12. Take input for a list of floats separated by commas.

13. Accept yes/no input and convert to boolean.

14. Take user input and check if it is palindrome.

15. Take a single character input and check if vowel or consonant.

16. Write code to take numeric input with range validation.

17. Take input for email and validate with regex.

18. Accept input for file path and check if file exists.

19. Take input for multiple words and print them in reverse order.

20. Prompt for username and password and validate length.

21. Take date input and print day of the week.

22. Write a program that asks for user confirmation (y/n).

23. Take input and split into dictionary key-value pairs.

24. Write a loop that asks for numbers and sums them until user inputs zero.

25. Accept input for RGB color code and validate format.

26. Take input for math expression and safely compute result.

27. Accept input and convert to title case.

28. Prompt for IP address and validate with regex.

29. Take input for list of integers and find max and min.

30. Accept multiple inputs and sort them.

31. Take input for time in HH:MM and validate.

32. Take multi-line input until blank line is entered.

33. Take input for hex color code and convert to RGB tuple.

34. Write code to take CSV input and parse into list of lists.

35. Prompt for a string and count vowels and consonants.

36. Take input for URL and parse domain name.

37. Accept inputs for dimensions of a rectangle and compute area.

38. Take a date input and convert it to Unix timestamp.

39. Accept a list of space separated numbers and calculate average.

40. Write a program that continuously asks for input and echoes back until ‘quit’.

41. Take input for two fractions and compute sum.

42. Accept input for sentence and count number of words.

43. Take input for file name and check extension.

44. Take password input and confirm by asking twice.

45. Accept a list of words and output the longest word.

46. Write code that takes input for chess move notation and validates it.

47. Accept input for a JSON string and parse it.

48. Take input and replace all spaces with underscores.

49. Accept multiple email addresses and validate all.

50. Write a program to take command line arguments and print them.