Exception Handling Questions (1–8)

Note: There are some concepts we haven't heard about. If you find the question/hints difficult or have no idea about it, then make habit for searching in internet. This searching habit will help you the most in your programming journey.

---

1. Write a program to take two numbers from the user and divide them. Handle `ZeroDivisionError`.

> *Hint:* Use a `try-except` block to prevent program crash when dividing by zero.

---

2. Take a number input and convert it to integer. Handle `ValueError`.

> *Hint:* Wrap the `int()` conversion in a `try-except` block.

---

3. Take a list of numbers. Ask the user for an index and print the value at that index. Handle `IndexError`.

> *Hint:* Use a fixed list like `[10, 20, 30]` and get user input for index.

---

4. Create a program that takes two numbers, performs division, and uses multiple `except` blocks to handle both `ValueError` and `ZeroDivisionError`.

> *Hint:* Use nested input handling and separate `except` for each error type.

---

5. Write a program to perform safe integer conversion from input using `try-except` and print "Invalid input" if it fails.

> *Hint:* Catch only `ValueError` to handle invalid integer conversion.

---

6. Use `try-except-finally` to demonstrate a risky division and always print "Done" at the end.

Hint: Place `print("Done")` in the `finally` block to ensure it always runs.

---

7. Demonstrate nested `try-except` blocks for taking user input and dividing two numbers.

Hint: Outer block for input conversion, inner for division.

---

8. Take input from user for an index and access that element in a list. If any error occurs, handle it using a generic `except:` block.

Hint: Use `except:` without specifying the error type.

List and Tuple Questions (9–25)

---

9. Create a list of 5 student names. Print each name using a loop.

Hint: Use a `for` loop to iterate over the list.

---

10. Write a program that appends 5 numbers entered by the user into a list and prints it.

Hint: Use `append()` inside a loop to add elements.

---

11. Write a function that takes a list of numbers and returns the maximum value.

Hint: You can use the `max()` function or loop manually to compare values.

---

12. Sort a list of integers and print the result in both ascending and descending order.

Hint: Use `sort()` or `sorted()` with the `reverse=True` argument.

---

13. Write a program to remove duplicates from a list.

Hint: loop through and add only unique items to a new list. use membership operator to check existence of an item in list

---

14. Create a tuple of 5 cities and print them using a loop.

Hint: Tuples are similar to lists in terms of iteration.

---

15. Ask the user to input 3 numbers and store them in a tuple. Then print the tuple.

Hint: Collect numbers in a list and convert it to a tuple using `tuple()` .

---

16. Write a function that takes a tuple of numbers and returns their sum.

Hint: Use the built-in `sum()` function or add manual addition logic

---

17. Given a list of numbers, write a function that returns a list containing only the even numbers.

Hint: Use a loop or list comprehension with `if num % 2 == 0` .

---

18. Create a list of words and count how many start with the letter ‘a’.

Hint: Use a loop and the `.startswith('a')` method.

---

19. Replace the second item in a list with a new value and print the updated list.

Hint: Use indexing like `list[1] = new_value`.

---

20. Demonstrate slicing on a list to get the first 3 and last 2 elements.

Hint: `list[:3]` gives first 3; `list[-2:]` gives last 2.

---

21. Write a program that joins two tuples and prints the combined result.

Hint: Tuples support `+` operator for concatenation.

---

22. Ask the user for a sentence and count how many words it contains using a list.

Hint: Use `split()` and `len()`.

---

23. Create a program that stores names in a list and allows the user to search for a name using the `in` operator.

Hint: Use `if name in list:` to check membership.

---

24. Take a list of numbers and reverse it using both slicing and the `reverse()` method.

Hint: Use `list[::-1]` and `list.reverse()` separately.

**25. Given a list of numbers, create a new list with their squares. **

Hint: use for loop to traverse through each number, calculate square (num**2) and then append to new list.
