**Question 1**:

The try and except blocks in Python are used for exception handling, which allows you to gracefully handle errors or exceptional situations that may occur during the execution of your code.

Roles of try and except block are –

* Making code more robust and resilient to unexpected inputs or conditions.
* We can define specific actions to take when certain types of errors occur.
* We can use try and except blocks to catch and log errors, and thereby identifying and diagnosing problems in code.
* Can provide alternative actions or error messages when specific exceptions occur.
* Can prevent your program from crashing when it encounters an error.

**Question 2**:

try:  
 # Code that may raise an exception  
 # ...  
except ExceptionName:  
 # Code to handle the exception  
 # ...

**Question 3**:

If no suitable except block is found anywhere in the call stack, the program will terminate with an unhandled exception error, and a traceback will be printed to the console.

**Question 4**:

* Using a specific exception type allows you to handle different types of exceptions differently.
* Using a bare except block may catch unexpected exceptions that you did not anticipate, potentially masking errors or making it harder to diagnose problems in your code.
* Using specific exception types makes your code clearer and more readable because it explicitly states the types of exceptions that are expected and handled.

**Question 5**:

Yes, we can have nested try-except blocks in Python.

Example code:

try:  
 num1 = int(input("Enter the numerator: "))  
 num2 = int(input("Enter the denominator: "))  
 try:  
 result = num1 / num2  
 print("Result:", result)  
 except ZeroDivisionError:  
 print("Error: Division by zero!")  
except ValueError:  
 print("Error: Invalid input! Please enter integers.")

**Question 6**:

Yes, we can use multiple except blocks to handle different types of exceptions.

Example code:

try:  
 file = open("nonexistent\_file.txt", "r")  
 contents = file.read()  
 file.close()  
 print("File contents:", contents)  
except FileNotFoundError:  
 print("Error: File not found!")  
except PermissionError:  
 print("Error: Permission denied while accessing the file!")  
except IOError:  
 print("Error: Input/Output error occurred!")

**Question 7**:

1. EOFError: This error is raised when the built-in function input() reaches the end of the file (EOF) without reading any data.
2. FloatingPointError: This error is raised when a floating-point arithmetic operation fails to produce a valid result due to numerical instability or precision issues.
3. IndexError: This error is raised when trying to access an index that is outside the bounds of a sequence (e.g., list, tuple, string).
4. MemoryError: This error is raised when an operation fails due to insufficient memory available for allocation.
5. OverflowError: This error is raised when an arithmetic operation exceeds the limits of the numerical data type used to represent the result. It typically occurs when performing arithmetic operations that result in a value larger than the maximum representable value for the data type.
6. TabError: This error is raised when inconsistent use of tabs and spaces is encountered in indentation.
7. ValueError: This error is raised when a function or operation receives an argument of the correct type but with an invalid value. Example: *number = int("abc")*