**Error and Exception Handling**

1. What is the correct order of execution in a try-except-else-finally block?

a) try, except, else, finally

b) try, else, except, finally

c) try, finally, except, else

d) try, except, finally, else

1. Which of the following is not a built-in exception in Python?

a) ValueError

b) TypeError

c) IndexError

d) LoopError

1. What will be the output of the following code?

try:

x = 1 / 0

except ZeroDivisionError:

print("A")

except ArithmeticError:

print("B")

a) A

b) B

c) AB

d) No output

1. Which statement is used to raise an exception in Python?

a) throw

b) raise

c) except

d) error

1. What is the purpose of the finally clause in a try-except block?

a) To define the main code to be executed

b) To handle exceptions that weren't caught by except clauses

c) To execute code regardless of whether an exception occurred

d) To raise a final exception if none were caught

1. What will be the output of the following code?

def func():

try:

return 1

finally:

return 2

print(func())

a) 1

b) 2

c) None

d) Exception

1. Which of the following is true about custom exceptions in Python?

a) They must inherit from the BaseException class

b) They can't have custom attributes

c) They are defined using the exception keyword

d) They can inherit from any existing exception class

1. What does the else clause do in a try-except block?

a) It's executed if no exception occurs in the try block

b) It's always executed, regardless of exceptions

c) It's executed if a specific exception occurs

d) It's used to define additional exception handlers

1. How can you catch multiple exceptions in a single except clause?

a) except (Exception1, Exception2):

b) except Exception1 or Exception2:

c) except [Exception1, Exception2]:

d) except {Exception1, Exception2}:

1. What will be the output of the following code?

try:

assert False, "Error occurred"

except AssertionError as e:

print(str(e))

a) AssertionError

b) Error occurred

c) False

d) No output

1. Answer: a) try, except, else, finally Explanation: This is the correct order of execution for a try-except-else-finally block.
2. Answer: d) LoopError Explanation: LoopError is not a built-in exception in Python.
3. Answer: a) A Explanation: ZeroDivisionError is caught first, so "A" is printed. The more general ArithmeticError is not reached.
4. Answer: b) raise Explanation: The raise statement is used to raise exceptions in Python.
5. Answer: c) To execute code regardless of whether an exception occurred Explanation: The finally clause is always executed, whether an exception occurred or not.
6. **Answer: b) 2** The finally block in Python always executes, regardless of whether an exception occurs or not. In this case, even though the try block returns 1, the finally block will override it and return 2.
7. **Answer: d)** Custom exceptions in Python are created by inheriting from an existing exception class, usually Exception or one of its subclasses. This allows you to define your own exception types and provide specific error handling.
8. **Answer: a)** The else clause in a try-except block is optional and is executed only if no exception occurs within the try block.
9. **Answer: a)** You can catch multiple exceptions in a single except clause by using a tuple of exception types.
10. **Answer: b)** The assert statement raises an AssertionError if the given condition is false. In this case, the condition is False, so the AssertionError is raised. The except block catches the exception and prints the error message associated with it.