Q1. What is an Exception in Python? Write the difference between Exceptions and Syntax Errors

Answer:

In Python, an exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions. When an exception occurs, Python raises an exception object, which contains information about the error, and the program can choose to handle this exception gracefully or let it propagate up the call stack. Exceptions are used to handle runtime errors that occur during program execution.

Key differences between Exceptions and Syntax Errors:

Exceptions: These are errors that occur during the execution of a program, such as dividing by zero, accessing a non-existent file, or trying to use an undefined variable. Exceptions are runtime errors. Syntax Errors: These are errors that occur when you write code that does not conform to the rules and structure of the Python language. Syntax errors prevent your code from being executed at all and are detected by the Python interpreter before runtime.

Q2. What happens when an exception is not handled? Explain with an example.Q2. What happens when an exception is not handled? Explain with an example.

Answer:

When an exception is not handled, it propagates up the call stack until it either encounters a suitable exception handler or reaches the top level of the program. If it reaches the top level without being caught, the program will terminate, and Python will display an error message along with a traceback. This can result in an abrupt termination of the program.

In [1]:
```
def divide(x, y):
    return x / y

try:
    result = divide(10, 0)
except ZeroDivisionError as e:
    print(f"Error: {e}")
```

```
Error: division by zero
```

Q3. Which Python statements are used to catch and handle exceptions? Explain with an example.

Answer:

Python uses try, except, else, and finally statements to catch and handle exceptions.

try: It encloses the code that may raise an exception. except: It specifies the code to be executed when an exception is raised. else: It contains code to be executed if no

exceptions are raised in the try block. finally: It contains code that is always executed, whether an exception is raised or not.

In [2]:
```
try:
    result = 10 / 0
except ZeroDivisionError as e:
    print(f"Error: {e}")
else:
    print("No exceptions were raised.")
finally:
    print("This block always executes.")
```

```
Error: division by zero
This block always executes.
```

Q4. Explain with an example: try and else blocks, and finally block.

In [ ]:
```
try:
    value = int(input("Enter a number: "))
except ValueError:
    print("Invalid input. Please enter a valid number.")
else:
    print(f"You entered: {value}")
finally:
    print("This block always executes.")
```

Q5. What are Custom Exceptions in Python? Why do we need Custom Exceptions? Explain with an example.

Answer:

Custom exceptions are user-defined exceptions that allow you to create specific exception classes tailored to your application's needs. They are typically used when the built-in exceptions don't fully capture the nature of the error you want to handle. Custom exceptions enhance code readability and make it easier to differentiate between different types of errors in your code.

Why we need Custom Exceptions:

To provide more specific information about the nature of an error. To create a hierarchy of related exceptions. To make code more readable and maintainable.

In [ ]:
```
class MyCustomException(Exception):
    def __init__(self, message):
        super().__init__(message)

def divide(x, y):
    if y == 0:
        raise MyCustomException("Division by zero is not allowed")
    return x / y

try:
    result = divide(10, 0)
except MyCustomException as e:
    print(f"Custom Exception: {e}")
```

Q6. Create a custom exception class. Use this class to handle an exception.

In [ ]:
```python
class MyCustomException(Exception):
    def __init__(self, message):
        super().__init__(message)

def divide(x, y):
    if y == 0:
        raise MyCustomException("Division by zero is not allowed")
    return x / y

try:
    result = divide(10, 0)
except MyCustomException as e:
    print(f"Custom Exception: {e}")
```