assignment=8

July 27, 2023

1 Q1. What is an Exception in python? Write the difference between Exceptions and Syntax errors.

In Python, an Exception is an error that occurs during the execution of a program. When an Exception occurs, the program stops executing and Python raises an error message that describes the type of exception that occurred and the line of code where the exception occurred.

Exceptions can occur for a variety of reasons, such as when a program tries to perform an operation on data that is not the expected type or when a program tries to access a resource that is not available. Python provides a set of built-in Exception classes that can be used to handle different types of exceptions.

Syntax errors, on the other hand, occur when there is a problem with the syntax of the Python code itself. These errors occur before the program is executed and are caught by the Python interpreter before the code is run. Examples of syntax errors include forgetting to close a parentheses or a quotation mark, or using an incorrect keyword or operator.

Here's an example that demonstrates the difference between Exceptions and Syntax errors:

[1]:
```
# Example of an Exception
try:
    a = 10 / 0
except ZeroDivisionError:
    print("Error: Division by zero")

# Example of a Syntax error
b = 10
if b = 5:
    print("b is equal to 5")
```

```
  Cell In [1], line 9
    if b = 5:
       ^
SyntaxError: invalid syntax. Maybe you meant '==' or ':=' instead of '='?
```

In the first example, we try to divide the number 10 by 0, which is not allowed. This causes a ZeroDivisionError Exception to be raised, which we catch and handle by printing an error message.

In the second example, we have a syntax error in the if statement, where we are using the assignment operator (=) instead of the equality operator (==). This causes a syntax error, which is caught by the Python interpreter and prevents the program from executing.

In summary, Exceptions are errors that occur during the execution of a program, while Syntax errors are errors that occur before the program is executed, due to problems with the syntax of the Python code itself

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

When an exception is not handled in a Python program, it will cause the program to terminate and an error message will be displayed that describes the type of exception that occurred and the line of code where the exception occurred.

Here's an example that demonstrates what happens when an exception is not handled:

[2]:
```
# Example of an unhandled exception
a = 10 / 0
print("Result: ", a)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Cell In [2], line 2
      1 # Example of an unhandled exception
----> 2 a = 10 / 0
      3 print("Result: ", a)

ZeroDivisionError: division by zero
```

As we can see from the error message, Python tells us that a ZeroDivisionError occurred and that it was caused by a division by zero on line 1 of the code. The "Traceback" section of the error message shows the sequence of function calls that led to the exception.

In summary, if an exception is not handled in a Python program, it will cause the program to terminate and display an error message that describes the type of exception that occurred and the line of code where the exception occurred. It's important to handle exceptions in your code to prevent unexpected program termination and to provide a graceful way to handle errors.

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

In Python, the try and except statements are used to catch and handle exceptions. The try statement is used to enclose a block of code that might raise an exception, while the except statement is used to specify how to handle the exception if one is raised.

Here's an example that demonstrates how to catch and handle an exception in Python using try and except:

[3]:
```
try:
    a = 10 / 0
except ZeroDivisionError:
    print("Error: Division by zero")
```

```
Error: Division by zero
```

4 Q4. Explain with an example:

a. try and else b. finally c. raise

In Python, the try statement can be followed by an optional else block. The else block is executed only if the try block does not raise an exception. It is often used to specify code that should be executed regardless of whether an exception occurred or not. Here's an example:

python Copy code try:

[5]:
```
try:
    a = 10 / 5
except ZeroDivisionError:
    print("Error: Division by zero")
else:
    print("a =", a)
```

```
a = 2.0
```

b. finally:

In Python, the finally block can be used to specify code that should be executed regardless of whether an exception was raised or not. The finally block is always executed, even if an exception occurred and was caught by an except block. Here's an example:

c. raise:

In Python, the raise statement can be used to raise an exception manually. It is often used to indicate that an error has occurred and to stop the execution of the program. Here's an example:

[6]:
```
def divide(a, b):
    if b == 0:
        raise ZeroDivisionError("Error: Division by zero")
    else:
        return a / b

try:
    result = divide(10, 0)
except ZeroDivisionError as e:
    print(e)
```

```
Error: Division by zero
```

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

In Python, custom exceptions can be defined by creating a new class that inherits from the built-in Exception class. These custom exceptions can be used to represent specific error conditions that are not covered by the built-in exceptions, and can provide more specific information about what went wrong in the code.

We need custom exceptions in Python when we want to raise exceptions that are specific to our own code or application. By creating custom exceptions, we can give more meaningful error messages and provide better feedback to the user. This can help with debugging and troubleshooting, and can make our code more readable and maintainable.

Here's an example of how to create a custom exception in Python:

[7]:
```
class NegativeNumberError(Exception):
    def __init__(self, number):
        self.number = number
        self.message = "Error: Negative number not allowed"
        super().__init__(self.message)

    def __str__(self):
        return f"{self.message}: {self.number}"

def divide(a, b):
    if b < 0:
        raise NegativeNumberError(b)
    else:
        return a / b

try:
    result = divide(10, -5)
except NegativeNumberError as e:
    print(e)
```

```
Error: Negative number not allowed: -5
```

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

[10]:
```
class CustomException(Exception):
    def __init__(self, message):
        self.message = message
        super().__init__(self.message)

def divide(a, b):
    if b == 0:
```

```
        raise CustomException("Error: Division by zero not allowed")
    else:
        return a / b

try:
    result = divide(10, 0)
except CustomException as e:
    print(e.message)
```

```
Error: Division by zero not allowed
```

[ ]:

[ ]: