Exception Handling in Python

- Python Exception Handling run-time error : division by zero

Exception Handling in Python. Exception handling is crucial for making your code more robust and preventing it from crashing due to unexpected errors.

1. **Try-Except Block**

- Python uses the `try`, `except` block to handle exceptions.
- Code that might raise an error is placed in the `try` block, and the `except` block contains code to handle the error if it occurs.

Example:

```
try:
    x = 10 / 0  # This will raise a ZeroDivisionError
except ZeroDivisionError:
    print("Cannot divide by zero!")
```

2. **Catching Multiple Exceptions**

- You can handle different types of exceptions using multiple `except` blocks.
- Example:

```
try:
    num = int(input("Enter a number: "))
    result = 10 / num
except ZeroDivisionError:
    print("Cannot divide by zero!")
except ValueError:
    print("That's not a valid number!")
```

3. **The `else` Block**

- The `else` block runs if no exceptions are raised in the `try` block.
- Example:

```
try:
    num = int(input("Enter a number: "))
    print(f"Success! You entered {num}.")
except ValueError:
    print("Invalid input!")
else:
    print("No exceptions occurred.")
```

4. **The `finally` Block**

- The `finally` block runs no matter what, whether or not an exception was raised. It's often used for cleanup tasks (e.g., closing files or releasing resources).

Example:

```
try:
    file = open("example.txt", "r")
    content = file.read()
except FileNotFoundError:
    print("File not found!")
finally:
    print("Closing the file.")
    file.close()  # This will always execute
```

5. **Raising Exceptions**

- You can raise exceptions manually using the `raise` keyword. This can be useful for custom error handling.

Example:

```
def check_age(age):
    if age < 18:
        raise ValueError("Age must be 18 or older.")
    else:
        print("Age is valid.")

try:
    check_age(15)
except ValueError as e:
    print(f"Error: {e}")
```

6. **Creating Custom Exceptions**

- You can create your own exception classes by inheriting from Python's built-in `Exception` class.

Example:

```
class NegativeNumberError(Exception):
    pass

def check_number(num):
    if num < 0:
        raise NegativeNumberError("Negative numbers are not allowed.")
    else:
        print("Number is valid.")
```

```
try:
    check_number(-5)
except NegativeNumberError as e:
    print(f"Error: {e}")
```

Exercises:

- **Exercise 1:** Write a program that asks the user for two numbers and divides them. Handle exceptions for `ValueError` (non-numeric input) and `ZeroDivisionError` (division by zero).
- **Exercise 2:** Create a custom exception called `TooYoungError` that is raised if a user enters an age below 18. Handle this exception in the program.

Would you like to try these exercises, or would you like to explore another topic such as file handling with exceptions, or more about the `try`, `except` blocks?

Key Terms

Fix the Error in Python

7. `else` Block After `try` Without `except`

Code:

```
try:
    result = 10 / 2
else:
    print("Division successful!")
```

Mistake:
An `else` block is used incorrectly after `try` without an `except` block.

Corrected Code:

```
try:
    result = 10 / 2
except ZeroDivisionError:
    print("You can't divide by zero!")
else:
    print("Division successful!")
```

- Now, the `else` block runs only if no exception occurs.

True/False (Mark T for True and F for False)

Answer Key (True/False):

Multiple Choice (Select the best answer)

1. **Which function would you use to determine the type of a variable in Python?**
    - A) id()
    - B) type()
    - C) str()
    - D) isinstance()

What is the correct way to raise an exception in Python? a) raise Exception("error message") b) throw Exception("error message") c) exit("error message") d) halt("error message")

Watch this video for the answer:

Answer key (Mutiple Choice):

Fill in the Blanks

Answer Key (Fill in the Blanks):

Exercises

Review Questions

References and Bibliography