Exercise 61:

### 1. ****Syntax Errors****

* **Description**: These occur when the code violates the grammar rules of the programming language. For example, missing colons, incorrect indentation, or mismatched parentheses.
* **How to Catch**: Python automatically detects syntax errors during the initial parsing of code before execution and will raise an error message indicating the location of the mistake.

**Example**:

if True

print("Syntax Error") # Missing colon after 'if'

**Solution**: The interpreter will point to the line with the syntax error. Correct the code by following Python's syntax rules.

### 2. ****Runtime Errors (Exceptions)****

* **Description**: These errors occur during the execution of the program, typically when Python encounters something unexpected that it cannot handle. Common runtime errors include:
  + **TypeError**: When an operation is performed on an inappropriate data type.
  + **ValueError**: When a function receives an argument of the correct type but an inappropriate value.
  + **ZeroDivisionError**: When attempting to divide by zero.
  + **FileNotFoundError**: When attempting to open a non-existent file.
* **How to Catch**: Runtime errors are handled using **try-except** blocks in Python.

**Example**:

try:

result = 10 / 0 # This will raise ZeroDivisionError

except ZeroDivisionError:

print("Cannot divide by zero!")

#### General Syntax for Catching Errors:

try:

# Code that may raise an error

except ErrorType:

# Code to handle the error

finally:

# Code that always runs (optional)

* **Multiple Exceptions**: You can catch different types of exceptions.

**Example**:

try:

num = int(input("Enter a number: "))

result = 10 / num

except ValueError:

print("That's not a valid number!")

except ZeroDivisionError:

print("Cannot divide by zero!")

### 3. ****Logical Errors****

* **Description**: These occur when the program runs without crashing, but produces incorrect results. Logical errors are due to mistakes in the algorithm or logic, and Python cannot detect these errors. They require careful debugging and testing.
* **How to Detect**: Logical errors are detected by testing the program and ensuring it produces the expected outputs.

**Example**:

def calculate\_area(length, width):

return length + width # Logical error (should be length \* width)

**Solution**: You must manually review the code logic and fix errors through testing.

### Catching Errors in Python

In Python, the most common way to handle exceptions (runtime errors) is by using **try-except** blocks. Here's a breakdown:

#### 1. **Basic Try-Except Block**

* Use try to wrap the code that might throw an error, and except to handle specific or generic exceptions.

**Example**:

try:

number = int(input("Enter a number: "))

print(10 / number)

except ZeroDivisionError:

print("Error: Cannot divide by zero.")

except ValueError:

print("Error: Invalid input. Please enter a number.")

#### 2. **Catching Multiple Exceptions**

* You can catch different exceptions using multiple except clauses or a tuple for multiple exception types in one block.

**Example**:

try:

num = int(input("Enter a number: "))

result = 10 / num

except (ValueError, ZeroDivisionError) as e:

print(f"Error occurred: {e}")

#### 3. **Catching All Exceptions**

* You can catch any kind of error using a generic except block. However, it's not recommended unless you're handling the error in a very specific way, as it can make debugging harder.

**Example**:

try:

# some code

except Exception as e:

print(f"An error occurred: {e}")

#### 4. **Using Finally**

* The finally block is optional and is used to execute code regardless of whether an exception occurred. It is commonly used for cleanup tasks like closing files or releasing resources.

**Example**:

try:

file = open("data.txt", "r")

# Some file operations

except FileNotFoundError:

print("File not found.")

finally:

file.close() # Always close the file

#### 5. **Raising Exceptions**

* You can explicitly raise exceptions in your code using the raise keyword when certain conditions are met.

**Example**:

def divide(a, b):

if b == 0:

raise ZeroDivisionError("Cannot divide by zero!")

return a / b

### Common Python Built-in Exceptions:

* **IndexError**: When trying to access an element outside the valid range of an indexable collection.
* **KeyError**: When a dictionary key is not found.
* **AttributeError**: When an object does not have the attribute you are trying to access.
* **ImportError**: When an import statement fails.
* **NameError**: When a variable is not defined.