LAB # 16

EXCEPPTIONAL HANDLING

OBJECTIVE

To learn how to handle errors in Python programs using exception handling and rograms using exception handling, understand common exceptions.

THEORY

Programs often encounter unexpected events, such as invalid input, missing files, or division by zero. Without handling, these events can crash the program. **Exception handling** allows Python programs to detect and respond to errors gracefully without terminating abruptly.

Python Exception Handling

- It allows a program to gracefully handle unexpected events (like invalid input or missing files) without crashing.
- Instead of terminating abruptly, Python lets you detect the problem, respond to it, and continue execution when possible.

1. Errors vs Exceptions

Type	Description	Example
Error	Serious problems in program logic that cannot be handled.	SyntaxError: print("Hello"
Exception	Runtime issues that can be caught and handled.	ZeroDivisionError: 10 / 0

Example:

```
# Syntax Error (cannot run)
# print("Hello World"

# Exception (can be handled)
n = 10
try:
   res = n / 0
except ZeroDivisionError:
   print("Cannot divide by zero!")
```

Output

```
>>> %Run Lab_16.py
Cannot divide by zero!
>>>
```

LAB # 16

Four Keywords in Handling Exception

Python provides four main keywords for handling exceptions: try, except, else and finally each plays a unique role. Let's see syntax:

1. **try:** Runs the risky code that might cause an error.
2. **except:** Catches and handles the error if one occurs.
3. **else:** Executes only if no exception occurs in try.
4. **finally:** Runs regardless of what happens useful for cleanup tasks like closing files.

Example:

```
try:
   n = 0
   res = 100 / n

except ZeroDivisionError:
   print("You can't divide by zero!")

except ValueError:
   print("Enter a valid number!")

else:
   print("Result is", res)

finally:
   print("Execution complete.")
```

Output

```
>>> %Run Lab_16.py
You can't divide by zero!
Execution complete.
>>>
```

Raise an Exception

- ✓ We raise an exception in Python using the raise keyword followed by an instance of the exception class that we want to trigger.
- ✓ We can choose from built-in exceptions or define our own custom exceptions by inheriting from Python's built-in Exception class.

Example:

```
def set_age(age):
   if age < 0:
      raise ValueError("Age cannot be negative")
   print(f"Age set to {age}")
```

LAB # 16

```
try:
   set_age(-8)

except ValueError as e:
   print(e)
```

Output

```
>>> %Run Lab_16.py
Age cannot be negative
>>>
```

EXERCISE

A. Point out the errors, if any, in the following Python programs.
Identify the exception and write a **try-except** block to handle it.

1. Code

```
n = 10
res = n / 0
```

Output:

2. Code : Handle the ValueError with a proper message.

```
x = int("abc")
```

Output:

B. Practice Try-Except-Else-Finally

- Write a program that asks the user to input a number and divides 100 by it.
- Use try, except, else, and finally blocks.

C. Catch Multiple Exceptions

- Create a list with mixed types: ["10", "abc", 20]
- Attempt to sum the first two elements after converting to integers.

LAB # 16

- Handle **ValueError, TypeError, IndexError** using **except**.

D. Raising Exceptions

- Write a function set_age(age)
- Raise ValueError if age is negative.
- Catch the exception and print a friendly message.