Computing Fundamentals using Python

SUBJECT CODE : UQ25CA151A

Samyukta D Kumta

Computer Applications

Computing Fundamentals using Python

Exception handling and error management

Exceptions in Python

- An exception is an error that occurs during program execution.
- For example: dividing by zero, accessing an invalid index, or using an undefined variable.

Errors are serious problems in the program.

- They usually occur because of wrong syntax or something Python itself cannot interpret.
- Errors generally stop the program immediately.

Example:

- # Syntax Error (wrong code structure)

```
print("Hello"
```

- # NameError (using undefined variable)

```
print(x)
```

Common Errors:

- **SyntaxError** : invalid Python syntax
- **IndentationError** : wrong indentation
- **NameError** : variable not defined

Exceptions

- An exception is an error that occurs during program execution.
- For example: dividing by zero, accessing an invalid index, or using an undefined variable.
- Exceptions can be handled using try-except.
- Exceptions don't always mean the code is wrong, but something unexpected happened.

Example:1

```
print(10 / 0)   # ZeroDivisionError
```

Example:2

Imagine you're writing a program to read a file.

If the file does not exist, Python will raise an error which occurs during execution #FileNotFoundError

Common Exceptions:

- **ZeroDivisionError** : dividing by zero
- **ValueError :** invalid type conversion
- **FileNotFoundError** : file not found
- **TypeError** : wrong type of data used

Example without Exception Handling

```
x = "5"
y = 10
# This will raise a TypeError automatically
print(x + y)
```

Example with Exception Handling

```
x = "5"
y = 10

try:
    result = x + y   # Python will throw TypeError here
    print("Result:", result)
except TypeError as e:
    print("Handled Exception:", e)
```

PES UNIVERSITY
CELEBRATING 50 YEARS

Basic Syntax

```
try:
    # Code that may cause an exception

except SomeException:
    # Code to handle the exception
    handler_code
```

try

- Used to wrap the block of code that may cause an exception.
- Prevents abrupt termination by shifting control to except if error occurs.
- Must be followed by at least one except or finally.
- **Example**

```
try:
    num = 10 / 0
```

except

- Handles the exception that occurs in the try block.
- Can handle specific exceptions (ZeroDivisionError, ValueError) or multiple exceptions.
- If no exception occurs, except block is skipped.
- Example

```
except ZeroDivisionError:
        print(" Cannot divide by zero")
```

Example : try with except

```
try:
    num1 = int(input("Enter numerator: "))
    num2 = int(input("Enter denominator: "))
    result = num1 / num2   # risky code
    print("Result:", result)
except ZeroDivisionError:
    print("Error: Cannot divide by zero!")
except ValueError:
    print("Error: Please enter numbers only.")
```

Try with else and finally Syntax

```
try:
    # Code that may raise an exception
except SomeException as e:
    print("Error:", e)
except AnotherException:
    print("Another error occurred")
else:
    # Runs if no exception occurs
    print("Everything worked fine!")
finally:
    # Always runs (cleanup code)
    print("Execution finished.")
```

Try with else and finally

else

- Optional block.
- Runs **only if no exception occurs** in the try block.
- Useful for code that should execute only when everything goes well.
- **Example**

```
else:
    print("No error, operation successful")
```

Try with else and finally

Finally

- Optional block.
- Always executed whether an exception occurs or not.
- Often used for cleanup tasks (closing files, releasing resources).
- **Example**

```
finally:
        print(" This always runs")
```

THANK YOU

Samyukta D Kumta
Department of Computer Applications
samyuktad@pes.edu