Exception Handling

Exception handling

- Python Exception Handling allows a program to gracefully deal with 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.

EXCEPTION HANDLING

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

```
try:
    # Code
except SomeException:
    # Code
else:
    # Code
finally:
    # Code
```

EXCEPTION HANDLING

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

Exception handling

```
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.")
```

O/P –
You can't divide by zero!
Execution complete.

try block attempts division, except blocks catch specific errors, else block would run if no errors occurred, while the finally block always runs, signaling the end of execution.

EXCEPTION HANDLING

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

User-Defined Exception

- In Python, we can define custom exceptions by creating a new class that is derived from the built-in Exception class.

```
class CustomError(Exception):
    ...
    pass

try:
    ...

except CustomError:
    ...
```

CustomError is a user-defined error which inherits from the Exception class.

User-Defined Exception

```
# define Python user-defined exceptions

class InvalidAgeException(Exception):
    pass

# you need to guess this number
number = 18

try:
    input_num = int(input("Enter a number: "))
    if input_num < number:
        raise InvalidAgeException
    else:
        print("Eligible to Vote")

except InvalidAgeException:
    print("Exception occurred: Invalid Age")
```

exception is fired when the input value is less than 18

pass is used for null statement

raise is used to call the exception

User-Defined Exception

```
class SalaryNotInRangeError(Exception):

    def __init__(self, salary, message="Salary is not in (5000, 15000) range"):
        self.salary = salary
        self.message = message
        super().__init__(self.message)


salary = int(input("Enter salary amount: "))

if not 5000 < salary < 15000:
    raise SalaryNotInRangeError(salary)
```

Exception raised for errors in the input salary.

salary -- input salary which caused the error

message -- explanation of the error

ASSERT – USED FOR CONSISTENCY TEST

- In Python, assertions use the assert keyword followed by an expression.
- If the expression evaluates to False, an AssertionError is raised.
- Following is the syntax of assertion –

```
assert condition, message
e.g

 x = 10
assert x > 0, "x must be positive"
print("Program continues...")
```

condition – A boolean expression that should be true.
message (optional) – An optional message to be displayed if the assertion fails..