Exception Handling

- Exception
- Assertion
- Raising Error
- Proper Error Handling Technique

Exception

- Runtime error on code
- If not handled, program terminated at the point of error giving user unfriendly message
- Exception has to be handled to avoid these scenarios
- Four exception blocks

Exception Blocks

try	➔	Check for potential exception
except	➔	Code to run on exception. Multiple are valid
else	➔	Code to run if exception doesn't occur
finally	➔	Code to run no matter if exception occurs or not

Syntax

```
try:
    -- statements --
except:
    -- statement on error --
else:
    -- statement if no error –
finally:
    -- statement --
```

Example

```
usr_input = input("Enter Num separated by comma: ").split(",")
try:
    num_1, num_2 = map(lambda num: float(num.strip()), usr_input)
    div_result = num_1 / num_2
except:
    print("Incorrect input. Please try again with correct data")
else:
    print(f"{num_1} + {num_2} = {div_result}")
finally:
    print("Program ended")
```

Example

```
usr_input = input("Enter Num separated by comma: ").split(",")
try:
    num_1, num_2 = map(lambda num: float(num.strip()), usr_input)
    div_result = num_1 / num_2
except ZeroDivisionError:
    print(”Second parameter can’t be zero”)
except Exception as e:
    print(e)
    print("Incorrect input. Please try again with correct data")
else:
    print(f"{num_1} + {num_2} = {div_result}")
finally:
    print(“Program ended")
```

Common Exception

Exception	Cause of error
ZeroDivisionError	Second operand of division or modulo operation is zero
TypeError	Function or operation is applied to an object of incorrect data type
ValueError	Function gets an argument of correct type but improper value
KeyError	Key is not found in a dictionary
IndexError	Index of a sequence is out of range
NameError	Variable is not found in local or global scope
AssertionError	Assert statement fails

Assertion

- Used when programmer knows the condition and always want that to be valid
- Takes Boolean condition as input. If it evaluates to False then error is generated
- Optionally we can also pass message

Syntax:

assert condition [, error_message]

Assertion

```
def rect_area(length, breadth):
    assert length > 0, "Length Cannot be Zero"
    assert breadth > 0, "Breadth cannot be Zero"
    return length * breadth

area_1 = rect_area(4, 5)
print(area_1)

area_2 = rect_area(-2, 3)
print(area_2)
```

Raising Error

- Allows programmer to force error based on condition

Syntax:

raise ExceptionName("Exception Message")

Raising Error

```
def license_eligibility(age):
    if type(age) != int:
        raise TypeError("Age has to be integer number")
    if age <= 0:
        raise ValueError("Age has to be positive")
    if age > 16:
        return "Eligible"
    else:
        return "Ineligible"
```

Proper Technique

```
def send_welcome_sms(number, name):
    number_txt = str(number)
    if len(number_txt) != 3:
        raise ValueError("Invalid Number")
    print(f"{name}, Welcome to python class")

user_info = {'983': 'Rahul', '07': 'Rajiv', '567': 'Sita'}
for mobile_num, user_name in user_info.items():
    send_welcome_sms(mobile_num, user_name)
```

Exercise