Exception Handling

Overview

- Introduction to Exception Handling
- Errors, Run Time Errors
- Handling IO Exception
- Try....except statement
- Raise & Assert
- Exception classes

Errors and Exception Handling

- **What is an Exception?**
  - An Exception is an error that happens during execution of a program. When that error occurs, Python generates an exception that can be handled, which avoids your program to crash.

- **Why use Exceptions?**
  - Exceptions are convenient in many ways for handling errors and special conditions in a program. When you think that you have a code which can produce an error then you can use exception handling

Where Exception may Occur?

- **Hardware/operating system level.**
  - Arithmetic exceptions; divide by 0, under/overflow.
  - Memory access violations, stack over/underflow.
- **Language level.**
  - Bounds violations: illegal indices.
  - Value Error: invalid literal, improper casts.
- **Program level.**
  - User defined exceptions.

Exception Handling Keywords

try

except

raise

else

finally

Common Exception/Errors in Python

- ✓IOError: If the file cannot be opened
- ✓ImportError: If Python cannot find the module
- ✓ValueError: Raised when a built-in operation or function receives an argument that has the right type but an inappropriate value
- ✓EOFError: Raised when one of the built-in functions (input()) hits an end-of-file condition (EOF) without reading any data

try…except…else…finally clause

```
try:
        data = something_that_can_go_wrong
except  ValueError :
        handle_the_exception_error
else:
        doing_different_exception_handling
finally:
        executes_under_all_circumstances
```

- ✓ The else clause in a **try, except** statement must follow all except clauses
- ✓ It is useful for code that must be executed if the try clause does not raise an exception

Note 1: Exceptions in the else clause are not handled by the preceding except clauses.

Note 2: Make sure that the else clause is executed before the finally block

Errors vs Runtime Errors

- Syntax Error (Missing parenthesis)

```
print("Hello)
# SyntaxError: EOL while scanning
string literal
```

- Runtime Error (File not found)

```
open("missing_file.txt")
# FileNotFoundError
```

Handling I/O Exceptions

Common when working with files, databases, or network requests.

```
try:
    file = open("data.txt", "r")
    print(file.read())
except FileNotFoundError:
    print(“Error: File not found!”)
finally:
    file.close()
    # Ensures file is always closed
```

Assert Statement

- ✓The assert statement is intended for debugging statements
- ✓It raises an exception as soon as the condition is False
- ✓The caller gets an exception which will go into **stderr** or **syslog**

```
assert <some_test>, <message>
```

- ✓The line above can be "read" as: If <some_test> evaluates to False, an exception is raised and <message> will be output

Assert Statement (cont.)

Example:

```
while (True):
    try:
        x=int( input("input value for x:\n"))
        assert(x>500) , "Value must be greater than 500"
        y=int (input ("input value of y:\n"))
        z=x/y
        print("result is:"+str(z))
    except (ZeroDivisionError ,ValueError,AssertionError ) as v:
        print(v)
    else:
        break
```

Custom/User Defined Exceptions

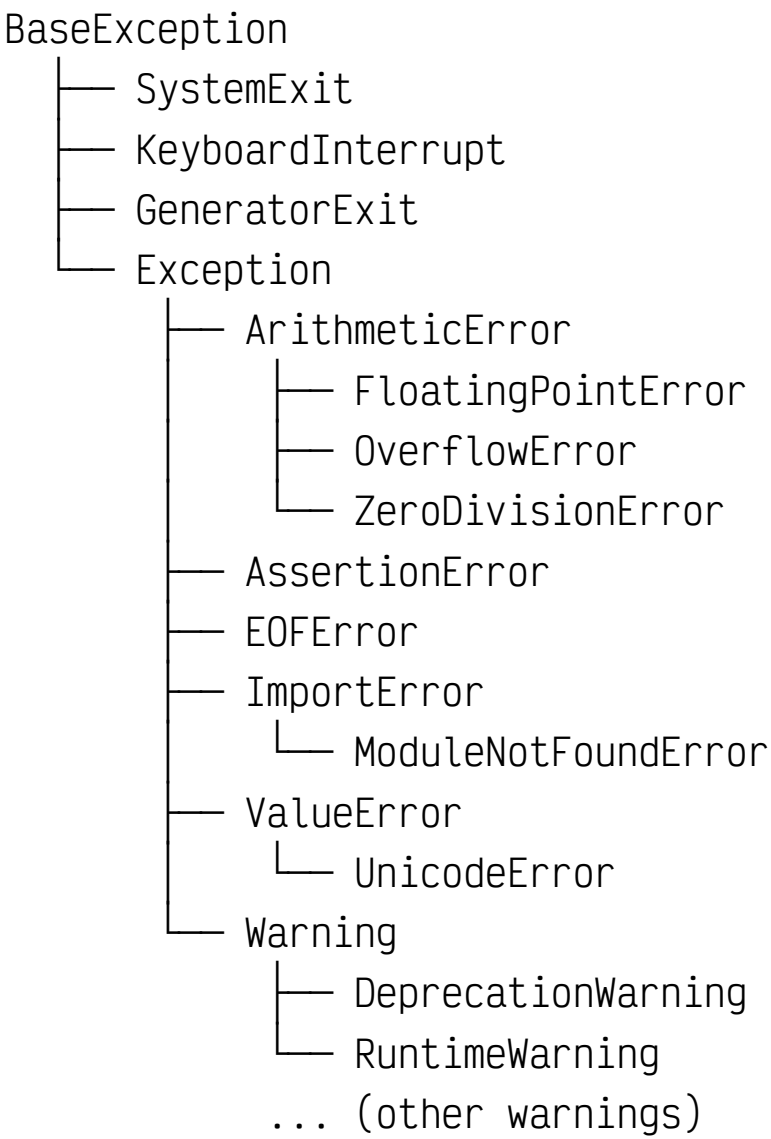
```
#Creating Custom Exception:
class MyException(Exception):
    def __init__(self,message="Salary must be greater than
10000"):
        self.message=message

#Raising Custom Exception:
def inputSalary(sal):
    if sal < 10000:
        raise MyException()
    print("salary is:"+str(sal))

#Using Custom Exception:
try:
    sal = int(input("Input your salary:\n"))
    inputSalary(sal)
except MyException as e:
    print(e.message)
```

Ignore Errors

Errors can be ignored without handling them in the program. We can do this using pass in except block of error handling section like below.

```
try:
      data = something_that_can_go_wrong
except:
      pass
```

Exception Classes



1. Base Exception

- Root class (avoid catching directly; use Exception instead).
- Includes system-exiting exceptions (SystemExit, KeyboardInterrupt).

2. Exception

- Parent class for all user-facing exceptions.
- Example:

```
try:
    x = 1 / 0
except Exception as e:
    # Catches most errors
    print(f"Error: {e}")
```

3. Common Subclasses

- ValueError: Invalid argument (e.g., int("abc"))
- TypeError: Incorrect type (e.g., "hello" + 5)
- IndexError: List/string index out of range
- KeyError: Missing dictionary key
- FileNotFoundError: File doesn’t exist
- ArithmeticError -> ZeroDivisionError : Division by zero
- AssertionError: assert condition fails

THANK YOU ☺