

# Chapter 6

- Mrs. Apurwa A. Barve

# Exception Handling

- ❖ An EXCEPTION occurs when there is a mistake in the program.
- ❖ When a mistake / error or exception occurs, the program behaves unnaturally and either results in termination or incorrect output. Both the situations are unpleasant from user's point of view.
- ❖ Exception handling allows a developer to resolve / predict the errors that happen in the program and take corrective actions for effective damage control. It allows the program to continue to execute even if an error occurs.
- ❖ Exceptions are errors that occur at runtime when the program is being executed.
- ❖ In Python an ERROR can be a SYNTAX ERROR or an EXCEPTION

# Exception Handling - Advantages

- ❖ **Ease of debugging** - The traceback printed by the interpreter in case of an error allows the developer to locate the actual place of error and rectify it.
- ❖ **Clean separation of code** - Error handling mechanism in Python allows the developer to separate error prone code from regular code using proper syntactic constructs. This enhances the readability of the code and helps in isolating tricky parts of the code.
- ❖ **Better code readability** - In built error handling keywords such as try, except, etc, allow the developer to use them to enhance the readability of the code.
- ❖ **Better control over code execution** - Exception handling allows the developer to have inbuilt mechanism to handle any kind if expected or unexpected exceptions and deal them in a professional way. This gives better control on program execution in times of error.

# Exception Handling - Buzz words

- ❖ Python provides readymade exception handling mechanism for its developers to be able to handle all sorts of exceptions.
- ❖ It also gives a set of pre defined exceptions, ERRORS, which can be raised in the code to generate a specific error.
- ❖ The language also allows the developer to code CUSTOM exceptions based on business needs.
- ❖ **try, except, else, finally and raise** are the buzz words of Python's exception handling mechanism.

# Activity 6.1

1. Copy the following code and observe the output

```
num1 = int(input('Enter first number'))
num2 = int(input('Enter the second number'))
res = num1/num2
print(res)
```

Execute the above code with different combinations of num1 and num2 values. Try passing num1 as 12 and num2 as 0. What happens to the code? What happens to the output?

```
Traceback (most recent call last):
  File "main.py", line 3, in <module>
    res = num1/num2
ZeroDivisionError: division by zero
```

The code stays the same, but the execution results into **RUNTIME ERROR** namely, **ZeroDivisionError**

# Activity 6.1

2. Make the following changes to the previous code.

```
num1 = int(input('Enter first number'))
num2 = int(input('Enter the second number'))
try:
    res = num1/num2
    print(res)
except:
    print("Exception caught")
```

Suspicious or error prone code is enclosed within try BLOCK

With each try we need to have one (or more) except block(s)

Execute the above code with different combinations of num1 and num2 values. Try passing num1 as 12 and num2 as 0. What happens to the code? What happens to the output?

Using try-except block, it is possible to handle the ZeroDivisionException in a smooth manner.

# Activity 6.1

3. Make the following changes to the code and observe the output :

```
num1 = int(input('Enter first number'))
num2 = int(input('Enter the second number'))
try:
    res = num1/num2

except:
    print("Exception caught")
print(res)
```

Execute the above code with different combinations of num1 and num2 values. Try passing num1 as 12 and num2 as 0.  
What happens to the code? What happens to the output?

```
Traceback (most recent call last):
  File "main.py", line 8, in <module>
    print(res)
NameError: name 'res' is not defined
```

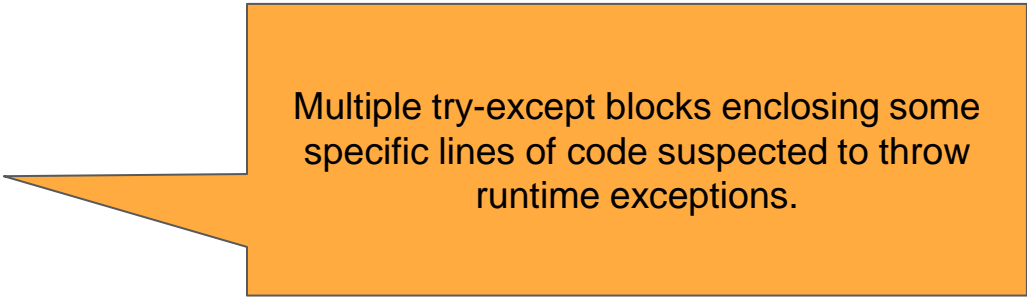
The code stays the same, but the execution results into **TWO RUNTIME ERRORS** namely, ZeroDivisionError and NameError. Why do we see **ONLY** one UNHANDLED exception then???

# Exception Handling - Buzz words

- ❖ In the previous code snippet, try-except block catches only that exception which occurs in the lines enclosed within the try block. The next exception, `NameError` is thrown by the print statement outside the try-except blocks when it tries to access the `res` variable.
- ❖ We can resolve this by the following approach:

```
num1 = int(input('Enter first number'))
num2 = int(input('Enter the second number'))
try:
    res = num1/num2

except:
    print("Exception caught")
try:
    print(res)
except:
    print("One more exception caught")
```



Multiple try-except blocks enclosing some specific lines of code suspected to throw runtime exceptions.



## Activity 6.2

1. Copy the following code and run the program. Observe the output

```
num1 = int(input('Enter first number'))
num2 = int(input('Enter the second number'))
try:
    res = num1/num2
    print(result)
except ZeroDivisionError:
    print("Exception caught")
except:
    print('Any other exception caught')
```

Execute the above code with different combinations of num1 and num2 values. What happens to the code? What happens to the output?

# Exception Handling - Buzz words

- ❖ In the previous code snippet, we see one try block followed by more than one except blocks. The first except block is dedicated to `ZeroDivisionError` while the lowest is for any and all other errors which could be occurring at runtime.
- ❖ The except block without any specific error is called the `DEFAULT` except block. It should always be mentioned at the `END` of the except blocks chain.
- ❖ As soon as Python raises an exception, it checks the except clauses from top to bottom and executes the first matching one that it finds.

## Activity 6.3

1. Copy the following code and run the program. Observe the output

```
num1 = int(input('Enter first number'))
num2 = int(input('Enter the second number'))
try:
    res = num1/num2
except ZeroDivisionError:
    print("Exception caught")
except:
    print('Any other exception caught')
else:
    print(res)
```

else block is used in Python with try-except to execute those lines of code which should be executed IF NO EXCEPTION occurs.

Execute the above code with different combinations of num1 and num2 values. What happens to the code? What happens to the output?

Execute the above code with num1=12 and num2=0. What output do you see and why?

## Activity 6.3

2. Copy the following code and run the program. Observe the output

```
num1 = int(input('Enter first number'))
num2 = int(input('Enter the second number'))
try:
    res = num1/num2
except ZeroDivisionError:
    print("Exception caught")
except:
    print('Any other exception caught')
else:
    print(res)
finally:
    print('Thank you for using this code')
```

finally block will be executed ALWAYS whether or not any exception occurs. This should be used to close resources, clear memory and nullify objects.

Execute the above code with different combinations of num1 and num2 values. What happens to the code? What happens to the output?

Execute the above code with num1=12 and num2=0. What output do you see and why?

## Activity 6.3

3. Look at the three examples given below and discuss which is the best way to implement exception handling in Python and why?

Sample A :

```
num1 = int(input('Enter first number'))
num2 = int(input('Enter the second number'))
try:
    res = num1/num2
    print(res)
except:
    print("Exception caught")
```

## Activity 6.3

Sample B :

```
num1 = int(input('Enter first number'))
num2 = int(input('Enter the second number'))
try:
    res = num1/num2
except ZeroDivisionError:
    print("Exception caught")
except:
    print('Any other exception caught')
else:
    print(res)
```

## Activity 6.3

Sample C :

```
num1 = int(input('Enter first number'))
num2 = int(input('Enter the second number'))
try:
    res = num1/num2
    print(result)
except ZeroDivisionError:
    print("Exception caught")
except NameError:
    print("NameError caught")
except Exception as ex:
    print(type(ex))
    print('Any other exception caught')
else:
    _
    print(res)
finally:
    print('Thank you for using this')
```

# Exception Handling - Buzz words

- ❖ From the previous samples, Sample C is the correct way to implement try-except mechanism due to the following reasons:
  1. Each except block is handling some named exception. This helps in easing the process of debugging and informs the developer exactly which exception was thrown from where.
  2. Except blocks are placed in the right order with the most general Exception being caught by the bottom-most except block.
  3. There are NO BARE excepts
  4. else block is added to make sure that the code which should be executed in case of no exceptions is placed within it. This ensure execution of that part of the business logic.
  5. finally block is added to provide scope for all kind of clean-up activities whether or not there is any exception



# Exception Handling - Custom exception

- ❖ If the business logic makes it mandatory to have a different exception apart from those which are already given by the language, then we need to design Custom Exception.
- ❖ Python allows easy ways to add Custom Exceptions to the already present list of predefined exceptions.
- ❖ Steps :
  1. Define custom exception class
  2. Raise custom exception from the necessary place in the business logic
  3. Handle custom exception using exception handling mechanism

## Activity 6.4

You are developing a portal to register new voters for the upcoming assembly elections in your state. If the age of any voter is less than 18, throw, UnderAgeUser exception. (Refer UnderAgeUserError.py)

Step 1 : Define Custom exception class - UnderAgeUserError. To this class add a constructor with a message.

```
class UnderAgeUserError(Exception):  
    """This exception is raised when an under age voter tries to register in the voting app"""  
    def __init__(self,message):  
        super().__init__(message)  
        self.message = message
```

The custom exception class is a SUB CLASS of Exception class

This is class doc to give more information about the class. Adding this is OPTIONAL

Custom Exception class constructor with the call to super()

## Activity 6.4

You are developing a portal to register new voters for the upcoming assembly elections in your state. If the age of any voter is less than 18, throw, UnderAgeUser exception.

Step 2 : Raise custom exception from the necessary place in the business logic - Accept age from the user. If the age is less than 18, raise UnderAgeUserError

```
ag = int(input('Enter your age'))  
if(ag < 18):  
    raise UnderAgeUserError("You are UNDER AGE to be a voter")
```

Execute the above code with different values for age. Observe what happens when the age is less than 18.

## Activity 6.4

You are developing a portal to register new voters for the upcoming assembly elections in your state. If the age of any voter is less than 18, throw, UnderAgeUser exception.

Step 3 : Handle custom exception using exception handling mechanism - Add necessary try-except blocks

```
class UnderAgeUserError(Exception):
    """This exception is raised when an under age voter tries to register in the voting app"""
    def __init__(self,message):
        super().__init__(message)
        self.message = message

ag = int(input('Enter your age'))
try:
    if(ag < 18):
        raise UnderAgeUserError("You are UNDER AGE to be a voter")
except UnderAgeUserError as uag:
    print("Error message is:",uag.message)
else:
    print("Age appropriate")
```

# Multithreading

❖ Self study

Thank you