



**Department of Computer Science and Engineering  
PES University, Bangalore, India**

**Lecture Notes  
Python for Computational Problem Solving  
UE23CS151A**

**Lecture #108  
*Exception Propagation mechanism***

**By,  
Prof. Sindhu R Pai,  
Anchor, PCPS - 2023  
Assistant Professor  
Dept. of CSE, PESU**

**Many Thanks to  
Dr. Shylaja S S (Director, CCBD and CDSAML Research Centers, Former  
Chairperson, CSE, PES University)  
Prof. Chitra G M (Asst. Prof, Dept. of CSE, PCPS Anchor – 2022**

## Introduction

When an **exception is raised**, **exception-propagation mechanism takes control**. The normal control flow of the program stops, and Python looks for a suitable exception handler. **Python's try statement establishes an exception handler via its except clauses**.

The **handlers deal with those exceptions that are raised in the body of the try clause [as discussed in the previous lecture #107]**, and **those exceptions that propagate from any of the functions called by that code, directly or indirectly [Will discussed in this lecture #108]**.

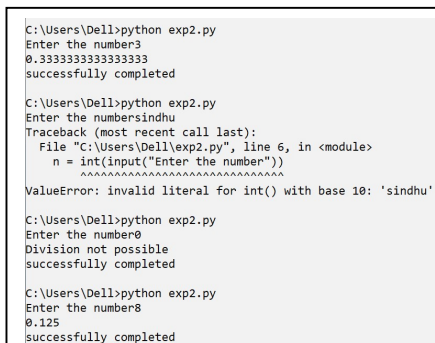
If an exception is raised within a try clause that has an applicable except handler, the try clause terminates and the handler executes. When the handler finishes, execution continues with the statement after the try statement.

If an **exception is thrown or raised inside a function**, it can be handled in **two ways**:

- Inside the function
- Outside the function

Consider,

```
def mathOp(n):  
    try:  
        return 1 / n  
    except ArithmeticError:  
        return "Division not possible"  
n = int(input("Enter the number"))  
print(mathOp(n))  
print("successfully completed")
```



```
C:\Users\Dell>python exp2.py  
Enter the number3  
0.3333333333333333  
successfully completed  
  
C:\Users\Dell>python exp2.py  
Enter the numbersindhu  
Traceback (most recent call last):  
  File "C:\Users\Dell\exp2.py", line 6, in <module>  
    n = int(input("Enter the number"))  
ValueError: invalid literal for int() with base 10: 'sindhu'  
  
C:\Users\Dell>python exp2.py  
Enter the number0  
Division not possible  
successfully completed  
  
C:\Users\Dell>python exp2.py  
Enter the number8  
0.125  
successfully completed
```

Please observe that there is **normal flow of execution if user enters valid integer**. If user enters **0**, **exception is raised inside the function** and handled in the same function using the **ArithmeticError**, the super class of **ZeroDivision Exception**. But if user enters something

other than these two, ValueError occurred. But not handled at all . Hence the code stops executing abruptly.

Now, let us make small modifications by **changing the position of try and except to outside the function definition**. This results in the same output as ValueError is not handled. But since **the function call is a part of try block, any exception raised during the execution of the function is handled in the corresponding except block**.

```
def mathOp(n):
    return 1 / n
n = int(input("Enter the number"))
try:
    print(mathOp(n)) #function call is inside try block
except ArithmeticError:
    print("Division not possible" )
print("successfully completed")
```

```
C:\Users\Dell>python exp2.py
Enter the number8
0.125
successfully completed

C:\Users\Dell>python exp2.py
Enter the numbersindhu
Traceback (most recent call last):
  File "C:\Users\Dell\exp2.py", line 3, in <module>
    n = int(input("Enter the number"))
    ~~~~~^~~~~~
ValueError: invalid literal for int() with base 10: 'sindhu'

C:\Users\Dell>python exp2.py
Enter the number0
Division not possible
successfully completed
```

Let me introduce one more exception inside a function and handle that exception by adding a generic Exception handler outside the function.

```
def mathOp(n):
    print(a) #exception raised here always
    #expression for return statement not executed at all.
    #Hence no ZeroDivisionError if you enter 0
    return 1 / n
n = int(input("Enter the number"))
try:
    print(mathOp(n))
except ArithmeticError:
    print("Division not possible" )
except Exception: #parent of NameError
    print("There is a NameError")
print("successfully completed")
```

```
C:\Users\Dell>python exp2.py
Enter the number5
There is an Exception: name 'a' is not defined
successfully completed

C:\Users\Dell>python exp2.py
Enter the number0
There is an Exception: name 'a' is not defined
successfully completed

C:\Users\Dell>python exp2.py
Enter the numbersindhu
Traceback (most recent call last):
  File "C:\Users\Dell\exp2.py", line 6, in <module>
    n = int(input("Enter the number"))
    ~~~~~^~~~~~
ValueError: invalid literal for int() with base 10: 'sindhu'
```

Any kinds of exception raised during the function execution are handled properly because of two handlers but not ValueError which occurred while taking the user input. What can be done to handle this? This is due to the presence of code that might throw an exception

outside the try block. So it is not handled at all. Let us handle this exception by adding this as a part of try block.

```
def mathOp(n):
    print(a) #exception raised here always
    #expression for return statement not executed at all.
    #Hence no ZeroDivisionError if you enter 0
    return 1 / n
try:
    n = int(input("Enter the number"))
    print(mathOp(n))
except ArithmeticError:
    print("Division not possible")
except Exception as e: #parent of NameError
    print("There is an Exception:",e)
    print("successfully completed")
```

```
C:\Users\Dell>python exp2.py
Enter the number0
There is an Exception: name 'a' is not defined
successfully completed

C:\Users\Dell>python exp2.py
Enter the number8
There is an Exception: name 'a' is not defined
successfully completed

C:\Users\Dell>python exp2.py
Enter the numbersindhu
There is an Exception: invalid literal for int() with base 10: 'sindhu'
successfully completed
```

If any exception occurs in the except block, will this be handled in the further except block?

– No. Code abruptly terminates.

```
def mathOp(n):
    return 1 / n
try:
    n = int(input("Enter the number"))
    print(mathOp(n))
except ArithmeticError:
    Print("Division not possible" ) #NameError here
except Exception as e: #parent of NameError
    print("There is an Exception:",e)
    print("successfully completed")
```

```
C:\Users\Dell>python exp2.py
Enter the number0
Traceback (most recent call last):
  File "C:\Users\Dell\exp2.py", line 5, in <module>
    print(mathOp(n))
    ~~~~~
  File "C:\Users\Dell\exp2.py", line 2, in mathOp
    return 1 / n
ZeroDivisionError: division by zero

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "C:\Users\Dell\exp2.py", line 7, in <module>
    Print("Division not possible")
    ~~~~~
NameError: name 'Print' is not defined. Did you mean: 'print'?
```

Solution is to **nest a try except block again inside except block.**

```
def mathOp(n):
    return 1 / n
try:
    n = int(input("Enter the number"))
    print(mathOp(n))
except ArithmeticError:
    try:
        Print("Division not possible" ) #NameError here
```

```
C:\Users\Dell>python exp2.py
Enter the number2
0.5
successfully completed

C:\Users\Dell>python exp2.py
Enter the number0
Nameerror inside except
successfully completed
```

```
except NameError:
    print("Nameerror inside except")
except Exception as e: #parent of NameError
    print("There is an Exception:",e)
print("successfully completed")
```

**Points to think!**

- If we remove except block from the nested try, will the exception be handled from outside handler?
- Can we just have try and finally blocks?
- Can we have only try block inside a function? If any exception in this, will this be handled by an except handler outside the function?
- Can the custom defined exception be used inside function?

**-END-**