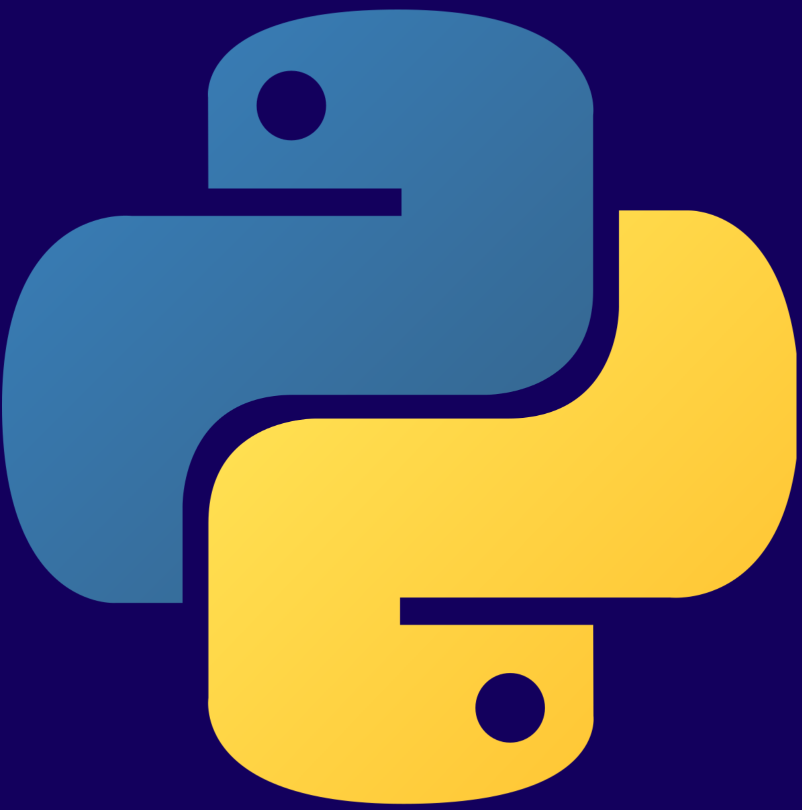


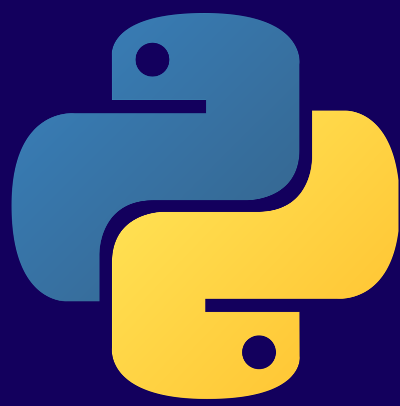
Prof R Madana Mohana

OBJECT ORIENTED PROGRAMMING USING PYTHON

ERROR & EXCEPTION HANDLING

Introduction to Errors and Exceptions | Handling Exceptions

https://www.youtube.com/c/RASINENIMADANAMOHANA



OUTLINE

- Introduction to Errors and Exceptions
- Handling Exceptions
- Multiple Except Blocks
- Multiple Exceptions in a Single Block
- Except Block Without Exception
- The else Clause

Introduction to Errors and Exceptions

- The programs that we write may behave abnormally or unexpectedly because of some errors and/or exceptions.
- The two common types of errors that we very often encounter are ***syntax errors*** and ***logic errors***.
- While ***logic errors*** occur due to poor understanding of problem and its solution, ***syntax errors***, on the other hand, arises due to poor understanding of the language.

Introduction to Errors and Exceptions

- However, *such errors* can be detected by exhaustive debugging and testing of procedures.
- But many times, we come across some peculiar problems which are often categorized as **exceptions**.
- **Exceptions** are run-time anomalies or unusual conditions (such as divide by zero, accessing arrays out of its bounds, running out of memory or disk space, overflow, and underflow) that a program may encounter during execution.

Introduction to Errors and Exceptions

- Like **errors**, **exceptions** can also be categorized as **synchronous** and **asynchronous exceptions**.
- While **synchronous exceptions** (like divide by zero, array index out of bound, etc.) can be controlled by the program, **asynchronous exceptions** (like an interrupt from the keyboard, hardware malfunction, or disk failure), on the other hand, are caused by events that are beyond the control of the program.

Introduction to Errors and Exceptions

Syntax Errors:

- **Syntax errors** occurs when we violate the rules of **Python** and they are the most common kind of error that we get while learning a new language.

For example, consider the lines of code given below.

```
>>>x=0
>>>if x == 0 print(x)
SyntaxError: invalid syntax
```

Logic Error:

- **Logic error** specifies all those type of errors in which the program executes but gives incorrect results.
- **Logical error** may occur due to wrong algorithm or logic to solve a particular program.

Introduction to Errors and Exceptions

Logic Error:

- In some cases, **logic errors** may lead to divide by zero or accessing an item in a list where the index of the item is outside the bounds of the list. In this case, the **logic error** leads to a run-time error that causes the program to terminate suddenly. These types of run-time errors are known as **exceptions**.

Exceptions:

- Even if a statement is syntactically correct, it may still cause an error when executed. Such errors that occur at run-time (or during execution) are known as ***exceptions***.
- An **exception** is an event, which occurs during the execution of a program and disrupts the normal flow of the program's instructions.

Introduction to Errors and Exceptions

Exceptions:

- When a program encounters a situation which it cannot deal with, it raises an exception.
- Therefore, we can say that an **exception** is a **Python object** that represents an error.
- When a program raises an exception, it must handle the **exception** or the program will be immediately terminated.

Exceptions:

- We can handle exceptions in our programs to end it gracefully, otherwise, if **exceptions** are not handled by programs, then error messages are generated.

Exceptions: *Examples*

```
>>>5/0
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    5/0
ZeroDivisionError: division by zero
```

Exceptions: *Examples*

```
>>>var + 5
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    var + 5
NameError: name 'var' is not defined. Did you
mean: 'vars'?
```

Exceptions: *Examples*

```
>>>'Python' + 3.10
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    'Python' + 3.10
TypeError: can only concatenate str (not
"float") to str
```

HANDLING EXCEPTIONS

- We can handle exceptions in our program by using **try** block and **except** block.
- A critical operation which can raise exception is placed inside the **try block** and the code that handles exception is written in **except block**.

HANDLING EXCEPTIONS

The syntax for **try-except** block can be given as,

```
try:
    statements
except ExceptionName:
    statements
```

HANDLING EXCEPTIONS

The **try** statement works as follows:

Step-1:

First, the ***try block*** (statements between the ***try*** and ***except*** keywords) is executed.

Step-2a:

If no exception occurs, the ***except block*** is skipped.

HANDLING EXCEPTIONS

Step-2b:

If an **exception** occurs, during execution of any statement in the ***try block***, then,

i) Rest of the statements in the ***try block*** are skipped.

ii) If the **exception** type matches the exception named after the ***except*** keyword, the ***except block*** is executed and then execution continues after ***try*** statement.

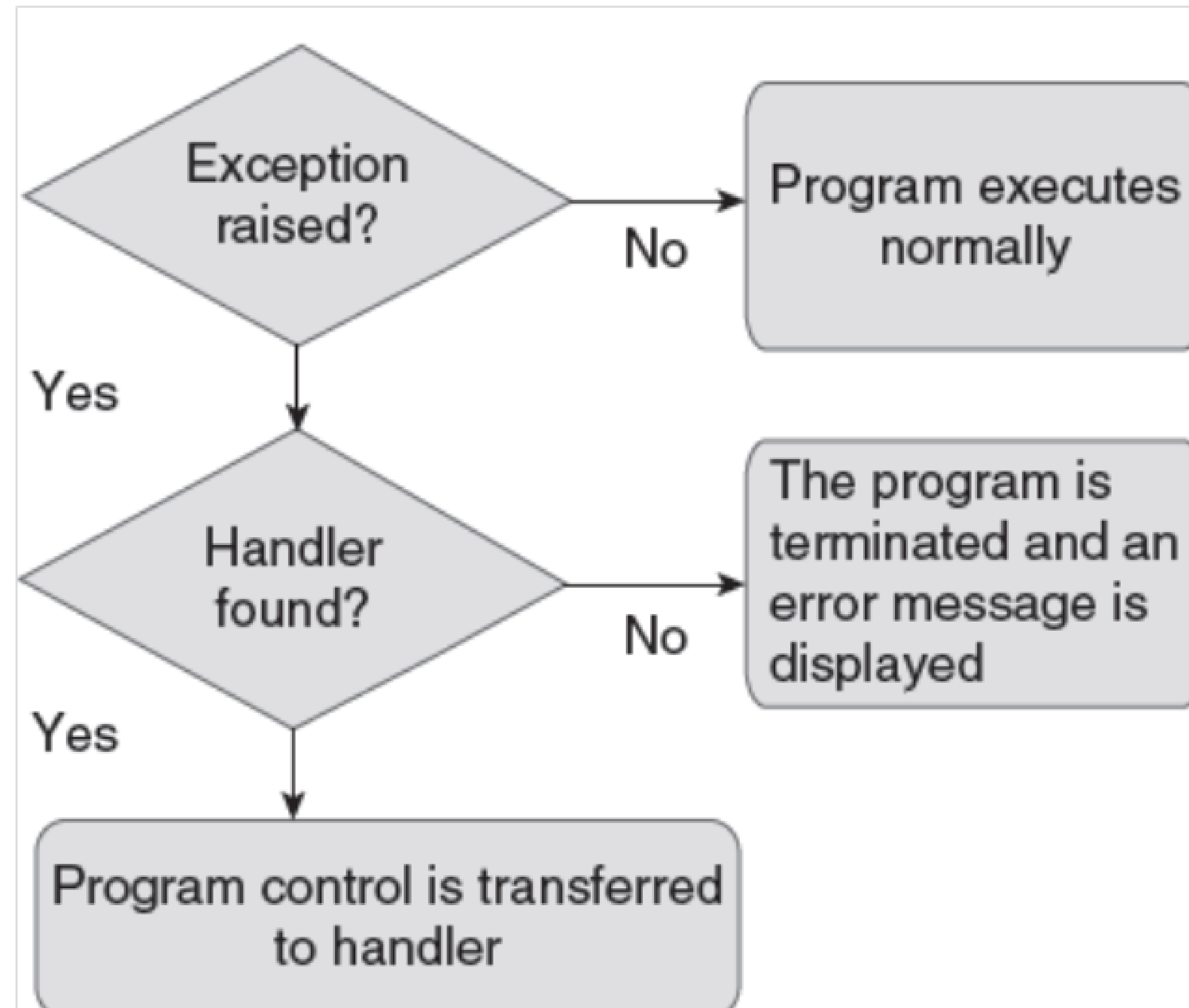
HANDLING EXCEPTIONS

Step-2b:

iii) If an **exception** occurs which does not match the **exception** named in the ***except block,*** then it is passed on to ***outer try block*** (in case of nested try blocks). If no exception handler is found in the program, then it is an unhandled exception and the program is terminated with an **error mesaage**.

HANDLING EXCEPTIONS

Step-2b:Case-iii



Ex: Program to handle the divide by zero exception

```
num = int(input("Enter the numerator:"))
denom = int(input("Enter the denominator:"))
try:
    quo = num/denom
    print("Quotient:",quo)
except ZeroDivisionError:
    print("Denominator cannot be zero..")
```

Ex: Program to handle the divide by zero exception

OUTPUT:

```
Enter the numerator:20
Enter the denominator:10
Quotient: 2.0
-------------------------
Enter the numerator:30
Enter the denominator:0
Denominator cannot be zero..
```

MULTIPLE EXCEPT BLOCKS

- **Python** allows you to have **multiple except blocks** for a **single try block**.
- The block which matches with the **exception** generated will get executed.
- A **try block** can be associated with more than one **except block** to specify handlers for different exceptions. However, only one handler will be executed.

MULTIPLE EXCEPT BLOCKS

- **Exception handlers** only handle exceptions that occur in the corresponding **try block**.
- We can write our programs that handle selected exceptions.

MULTIPLE EXCEPT BLOCKS

The **syntax** for specifying **multiple except blocks** for a **single try block** can be given as:

```
try:
    operations are done in this block
    .....................
except Exception1:
    If there is Exception1, then execute this block.
except Exception2:
    If there is Exception2, then execute this block.
    ..................
else:
    If ther is no exception then execute this block.
    ...................
```

MULTIPLE EXCEPT BLOCKS

Ex. Program with multiple except block.

```
try:
    num = int(input("Enter the number:"))
    print(num**2)
except (KeyboardInterrupt):
        print("You shoud have entered a command from keyboard....Program Terminating..")
except (ValueError):
        print("Please check before you enter...Program Terminating..")
print("Bye..")
```

Ex. Program with multiple except block.

OUTPUT:

```
Enter the number:10
100
Bye..
Enter the number:"Hi"
Please check before you enter...Program Terminating..
Bye..
Enter the number:abc
Please check before you enter...Program Terminating..
Bye..
Enter the number: PRESS CTRL+C Command
You shoud have entered a command from keyboard....Program Terminating..
Bye..
```

MULTIPLE EXCEPTIONS IN A SINGLE BLOCK

- An **except clause** may name **multiple exceptions** as a parenthesized tuple as shown in the program given below.
- So whatever **exception** is raised, out of the three exceptions specified, the **same except block** will be executed.

MULTIPLE EXCEPTIONS IN A SINGLE BLOCK

Ex. Program having an except clause handling multiple exceptions simultaneously.

```
try:
    num = int(input("Enter the number:"))
    print(num**2)
except (KeyboardInterrupt, ValueError, TypeError):
     print("Please check before you
enter...Program Terminating..")
print("Bye..")
```

MULTIPLE EXCEPTIONS IN A SINGLE BLOCK

Ex. Program having an except clause handling multiple exceptions simultaneously.

OUTPUT:

```
Enter the number:10
100
Bye..
Enter the number:abc
Please check before you enter...Program Terminating..
Bye..
Enter the number:PRESS CTRL+C Command
Please check before you enter...Program Terminating..
Bye..
```

EXCEPT BLOCK WITHOUT EXCEPTION

- We can even specify an **except block** without mentioning any exception (i.e., **except:**).
- This type of **except block** if present should be the last one that can serve as a wildcard (when multiple except blocks are present). But use it with extreme caution, since it may mask a real programming error.

EXCEPT BLOCK WITHOUT EXCEPTION

- In large software programs, many times, it is difficult to anticipate all types of possible exceptional conditions.
- Therefore, the programmer may not be able to write a different handler (**except block**) for every individual type of exception.
- In such situations, a better idea is to write a **handler** that would catch all types of exceptions.

EXCEPT BLOCK WITHOUT EXCEPTION

The **syntax** to define a **handler** that would catch every possible **exception** from the **try block** is:

```
try:
    Write the operations here
    ....................
except:
    If there is any exception, then execute this block.
    ......................
else:
    If there is no exception then execute this block.
```

EXCEPT BLOCK WITHOUT EXCEPTION

Ex. Program to demonstrate the use of except: block

```
try:
    #f1 = open('File1.txt','r')
    str1 = f1.readlines()
    print(str1)
except IOError:
    print("Error occured during Input...Program
Terminated..")
except ValueError:
    print("Could not convert data to an integer..")
except:
    print("Unexpected Error..Program Terminating..")
```

EXCEPT BLOCK WITHOUT EXCEPTION

Ex. Program to demonstrate the use of except: block

OUTPUT:

By including f1 = open('File1.txt','r')

```
['Welcome to the course\n', 'Object Oriented
Porgramming using Python']
```

By excluding #f1 = open('File1.txt','r')

```
Unexpected Error..Program Terminating..
```

THE else CLAUSE

- The **try … except** block can optionally have an **else clause**, which, when present, must follow all except blocks.
- The statement(s) in the **else block** is executed only if the **try clause** does not **raise** an **exception**.

THE else CLAUSE

Ex. Program to demonstrate else block.

```
try:
    f1 = open('File1.txt')
    str1 = f1.readline()
    print(str1)
except IOError:
    print("Error occured during Input...Program Terminated..")
except:
    print("Error occured..Program Terminated..")
else:
    print("Program Terminated successfully..")
```

Ex. Program to demonstrate else block.

OUTPUT:

```
By including f1 = open('File1.txt')
Welcome to the course


Program Terminated successfully..
```