



# PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

## Exception handling in Python

---

**Prof. Sindhu R Pai**

PCPS Theory Anchor - 2024

Department of Computer Science and Engineering

#### Introduction:

- Programming error is an error which may occur because of various reasons.

Program can produce some errors even if the program is perfect.

- This may be because of
  1. Exceptional situations that may occur during the execution of a program.
  2. The data coming from outside the program which is malformed .

Based on when the error occurs, errors are classified into two types:

#### 1. Syntax errors

This occurs when there is a deviation from the rules of the language.

A component of python's interpreter called parser discovers these errors. If a hybrid interpreter, compiler finds this error.

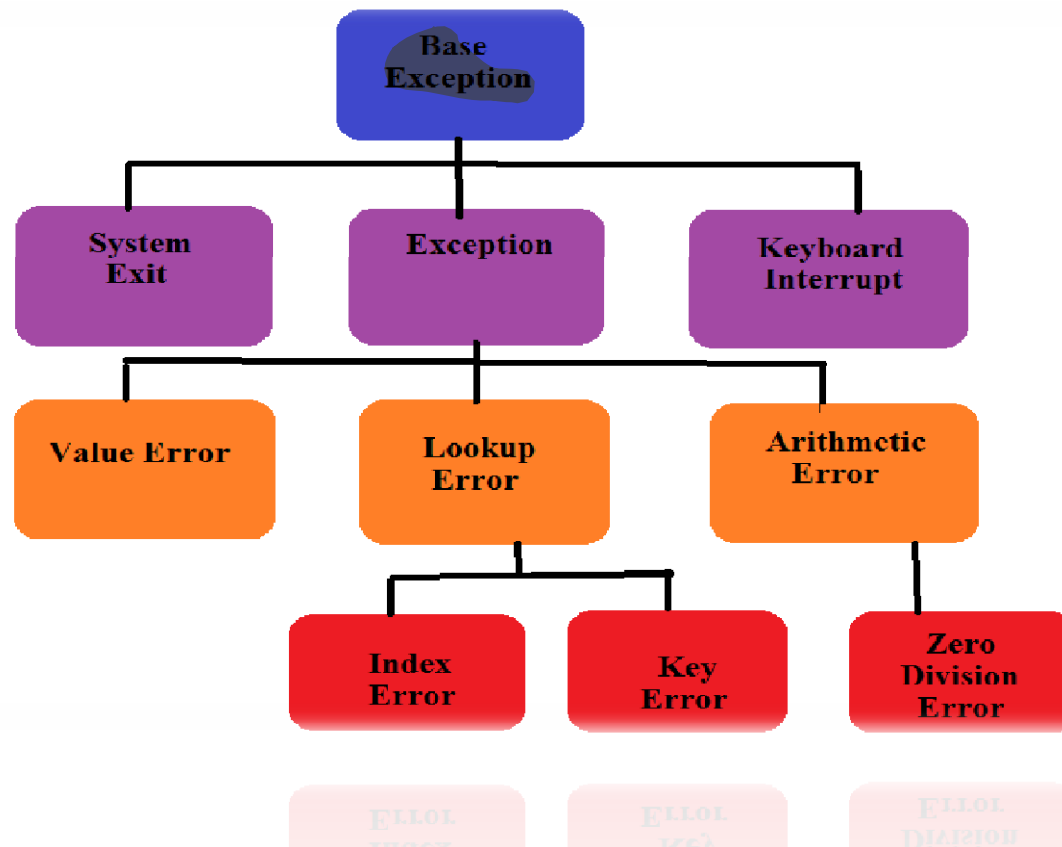
```
Ex: print("good morning")  
    print("good morning"
```

#### 2. Runtime Errors (Exceptions)

These are the errors which are detected during execution. This disrupts the normal flow of execution of instructions.

```
Ex: Print("Good Morning")  
    #NameError: No name 'Print' found
```

### Exception Tree-Partial View



**Two categories** of Exceptions:

1. **Built-In exception (System defined exceptions )**  
Those exceptions which Python knows by default.
2. **User defined exceptions**  
User-defined type/class written for handling the exceptions

# PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

## Exception Handling



Exceptions	Explanation
<b>Keyboard Interrupt</b>	Raised when users hit Ctrl-C, the interrupt key
<b>Overflow Error</b>	Raised when a floating-point expression evaluates to a value that is too large
<b>ZeroDivision Error</b>	Raised when attempting to divide by 0
<b>IO Error</b>	Raised when a sequence index is outside the range of valid indexes
<b>Name Error</b>	Raised when attempting to evaluate an unassigned identifier
<b>Type Error</b>	Raised when an operation or function is applied to an object wrong type

#### Exception Handling( How to deal with errors?)

- Exception handling is a way to deal with errors or exceptional situations that may occur during the execution of a program.
- It allows you to gracefully manage these situations instead of letting the program crash.
- Python provides try, except, else, and finally blocks for handling exceptions.

#### Constructs used in Exception handling -

try, except, else, finally, raise

try:

- Contains the code which might cause an exception
- It may have more than one except clause to specify handlers for different exceptions.
- At most one handler(except) will be executed.
- Handlers only handle exceptions that occur in the corresponding try-block, not in the other handlers of the same try block.



except :

- The exception thrown in try block, is caught in except block and the statements in this block are executed.
- Can have multiple exceptions as a parameterized tuple.
- If many except clauses are there, then the **specific classes are specified first** and **then the parent classes are specified**.
- Last except clause provide a common and default way of handling all exceptions.

## PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

### Exception Handling

---



else :

- This block will be executed if no errors were raised.
- That is , this block gets executed when try is successful

Finally:

- The statements in this clause will be executed regardless of whether an exception occurred or not in the try block.
- Its primary purpose is to perform **clean up actions or tasks and close the resources** used in the program.

raise

- Is used to forcefully throw an exception

#### Basic Structure

```
try:
    # This is a block of code where an exception might occur

except SomeException:
    # This is the except block to handle the exception

else:
    # This is executed if no exceptions were raised in the "try" block

finally:
    #This is executed whether an exception occurs or not
```

## PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

### Exception Handling

---

Example 1: (with try and except only)

try:

```
    print(x)
```

except: #default block

```
    print("An exception occurred as x is not defined ")
```

Output

An exception occurred as x is not defined

## PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

### Exception Handling

---



Example 2: (with try and except only-ZeroDivision Error)

try:

```
y = 10 / 0
```

except ZeroDivisionError:

```
print("Python exception raised")
```

Output

Python exception raised

## PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

### Exception Handling

---



Example 3: (with try,except ,else and finally)

**try:**

```
result = 10 / 0          # This will raise a ZeroDivisionError
```

**except** ZeroDivisionError as e:

```
print("Error:", e)      # Handling the ZeroDivisionError
```

**else:**

```
print("No exceptions occurred.")
```

**finally:**

```
print("This will always execute, regardless of exceptions.")
```

**Output:**

Error: division by zero

This will always execute,  
regardless of exceptions.

## PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

### Exception Handling

---



#### Example 4 (with try except else and finally)

**try:**

```
file = open("example.txt", "r")
```

**except** FileNotFoundError:

```
print("File not found.")
```

**else:**

```
print("File operations completed successfully.")
```

**finally:**

```
print("File closure commands can be given here")
```

Output:

File not found.

File closure commands can be given here

## PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

### Exception Handling



#### Raising Exceptions:

Exceptions can be raised manually using the raise statement, which raises the custom exceptions. Can also be used to raise builtin exceptions.

#### Example 5

**try:**

```
age = int(input("Enter your age: "))
```

```
if age < 0:
```

```
    raise ValueError("Age cannot be negative.")
```

```
except ValueError as e: #as keyword is used to create an object
    print("Error:", e)
```

**Output:**

Enter your age: -7

Error: Age cannot be negative.



## PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

### Exception Handling

---



#### Accessing Exception Information:

When an exception occurs, you can access information about the exception using “as” to assign it to a variable.

```
try:  
  
    res= 20 / 0  
  
except Exception as e:  
  
    print("Exception type:", type(e).__name__)  
    print("Exception details:", e)
```

#### Output:

Exception type: ZeroDivisionError  
Exception details: division by zero

#### Matching of except blocks:

- The raised or thrown exception object is matched with the except blocks in the order in which they occur in the try-except statement.
- The code following **the first match is executed.**
- It is **always the first match and not the best match.**
- If no match occurs and if there is a default except block, this will be executed.

## PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

### Exception Handling

---



#### Handling Multiple Exceptions:

##### Example 6

try:

```
num1 = int(input("Enter first number: "))
```

```
num2 = int(input("Enter second number: "))
```

```
result = num1 / num2      # This might raise a ZeroDivisionError
```

```
my_list = [1, 2, 3]
```

```
index = int(input("Enter an index for the list: "))
```

```
value = my_list[index]    # This might raise an IndexError
```

## PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

### Exception Handling

---



```
except ValueError:  
    print("ValueError: Please enter a valid number.")
```

```
except ZeroDivisionError:  
    print("ZeroDivisionError: Cannot divide by zero.")
```

```
except IndexError:  
    print("IndexError: Index is out of range.")
```

```
else:  
    print("No exceptions occurred.")
```

```
finally:  
    print("This will always execute, regardless of exceptions.")
```

#### Output:

```
Enter first number: 4  
Enter second number: 2  
Enter an index for the list: 6  
IndexError: Index is out of range.  
This will always execute, regardless  
of exceptions.
```

#### 2. User defined exceptions

- Users can define custom exceptions by creating a new class.
- **This exception class has to be derived, either directly or indirectly, from the built-in Exception class.**
- Most of the built-in exceptions are also derived from this class.
- The new exception, like other exceptions, can be raised using the raise statement with an optional error message.

## PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

### Exception Handling

---



#### Example 7

```
class MyException(Exception): #inherits from the Exception class
```

```
    def __init__(self, str):
        self.str = str
```

```
    def __str__(self):
        return self.str
```

```
# check whether n is between 1 and 100
```

```
n = int(input("Enter a number:"))
```

## PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

### Exception Handling

---



```
try:
    n = int(input("Enter the number"))
    if not 1 <= n <= 100 :
        raise MyException("Number not in range")
except MyException as e:
    print(e)
else:
    print("Number well within the range")

print("Program Terminated")
```

#### Output:

```
Enter the number200
Number not in range
Program Terminated
```

#### Example 8

```
class FiveDivisionError(Exception):  
    pass  
  
try:  
    n1=int(input("Enter the first number"))  
    n2=int(input("Enter the second number"))  
    if n2==5:  
        raise FiveDivisionError("Dividing by 5 not possible")  
    d=n1/n2  
    print("The answer is",d)  
except (FiveDivisionError,ZeroDivisionError) as e:  
    print(e)  
print("Program ends")
```

#### Output:

```
Enter the first number 7  
Enter the second number 5  
Dividing by 5 not possible  
Program ends
```





## THANK YOU

---

Department of Computer Science and Engineering

Dr. Shylaja S S, Director, CDSAML & CCBD, PESU

Prof. Sindhu R Pai – [sindhurpai@pes.edu](mailto:sindhurpai@pes.edu)

Prof. C N Rajeswari