**TRY EXCEPTION BLOCK-Sandip B.**

In any programming language there are **2 types of errors** are possible:

**1. Syntax Errors**

**2. Runtime Errors**

**1. Syntax Errors/Compile time error:**

The errors which occurs because of **invalid syntax** are called syntax errors.

**e.g.**

**prnt("Hello")**

name 'prnt' is not defined

**e.g.**

**x=Lambda x,y:x+y**

**print(x(4,5))**

invalid syntax

**Programmer is responsible to correct these syntax errors.**

**2. Runtime Errors:**

If **something goes wrong while executing program** k/a Runtime error.

Also called as Logical Errors or Exceptions .

**Reasons:**

**Wrong end user input:**

**e.g.**

x=int(input("Enter number 1:")) #input1 -10

y=int(input("Enter number 2:")) #input2 -sandip or whitespace

print(x+y)

**Output:**

**ValueError**: invalid literal for int() with base 10: 'sandip'

**e.g. print(10/"ten") ==>TypeError:** unsupported operand type(s) for /: 'int' and 'str'

**Logical Error:**

**Wrong programming logic:**

**e.g.**

x=int(input("Enter number 1:")) #input 12

y=int(input("Enter number 2:")) #input 2

print("addition is",**x/y**) #outputaddition is 6.0 ,wrong output

**e.g.**

**Eg:** print(10/0) **==>ZeroDivisionError**: division by zero

**Other reasons:**

Device failures -Battery issue /short,etc

the loss of a network connection -Wifi

insufficient memory -RAM

**EXCEPTION:**

An **unwanted and unexpected event** that **disturbs normal flow** of program is called exception.

It is highly **recommended** to handle exceptions. The main **objective** of exception handling

is **Graceful Termination of the program**.

It does **not mean we are repairing exception**. We have to define **alternative way** to continue rest of the program normally. (Show must go on)

**If never handled**, an **error** message is displayed and our program comes to a sudden **unexpected halt.**

**Real time example : Aeroplane crash**

In 1996,**flight 501** resulted into Catastrophic Explosion and hundreds of people were killed,because **the airline company has avoided exception handling policy** of **aborting computation on arithmetic error**.

**Example of Exceptions :**

**ZeroDivisionError**

print(10/0) ==>ZeroDivisionError: division by zero

**TypeError**

If you attempt to **divide an integer with a string**, the data types of the integer and the string object will not be compatible. Due to this,  the Python interpreter will raise a TypeError exception .

**print(10/"ten")**

e.g.

|  |
| --- |
| x = "one"  y = 2  print(x+y) |

**ValueError:**

when a user gives an **invalid value** to a function but is of a valid argument.

x=int(input("Enter number 1:"))

y=int(input("Enter number 2:")) #space or unsupported data type

print("addition is",x+y)

**Floatingpoint Error**

print(0.9-0.6)

if 0.9-0.6==0.3:

print("Right")

else:

print("Wrong")

**FileNotFoundError:**

a file that the system cannot find

**Reasons:**

**wrong file path** specified

the file is present in **another directory**

an **extension** **error**

e.g. In file handling if we give wrong filename name in read file command

f=open("f8.txt",'w')

f1=open("f88.txt",'r')  
data=f1.read()

print(data)

**FileNotFoundError: [Errno 2] No such file or directory: 'f88.txt'**

**EOFError**

for i in range(10):

print(i)

for j in range(4):

**Normally shows in online interpreter SyntaxError: unexpected EOF while parsing**

**Overflow Error**

import math  
print("The exponential value is")  
print(math.exp(10)) #If we give 1000

 e^x means 2.7 power 10

**Flow:**

**Every exception** in Python is an **object**. For every exception type the **corresponding classes**

**are available.**

Whevever an exception occurs **PVM will create the corresponding exception object** and

**will check for handling code**. If handling code is **not available** then Python interpreter

**terminates the program abnormally** and prints corresponding exception information to

the console.(Errors)

The rest of the program won't be executed.

e.g. **Without try except**

print("this is statement 1")

print(10/0) # ZeroDivisionError k/a Risky Code which raise exception

print("this is statement 2") #Not executed ZeroDivError

The code which may raise exception is called **risky code** and we have to take risky code

inside **try** block. The **corresponding handling code** we have to take inside **except** block.

**Format:**

**try:**

**Risky Code**

**except XXX:**

**Handling code/Alternative Code**

**e.g. With Try Except**

print("this is statement 1")

try:

print(10/0)

except:

print("this is statement 2")

**Note:**

Whenever we are writing try block, compulsory we should write except or finally

Block and vice versa.i.e without except or finally block we cannot write try block.

**Error : SyntaxError:** expected 'except' or 'finally' block

**Control Flow in try-except:**

**try:**

**stmt-1**

**stmt-2**

**stmt-3**

**except:**

**stmt-4**

**stmt-5**

**case-1: If there is no exception**

1,2,3,5 and Normal Termination

**case-2: If an exception raised at stmt-2 and corresponding except block matched**

1,4,5 Normal Termination

**case-3: If an exception raised at stmt-2 and corresponding except block not matched**

1, Abnormal Termination

**case-4: If an exception raised at stmt-4 or at stmt-5 then it is always abnormal**

**termination.**

Note: If any statement which is not part of try block raises an exception then it is always

abnormal termination.

**try with multiple except blocks:**

**try:**

**-------**

**-------**

**-------**

**except ZeroDivisionError:**

alternative /handling code

perform alternative arithmetic operations

**except FileNotFoundError:**

use local file instead of remote file

**Note:**

If try with multiple except blocks available then based on raised exception the

corresponding except block will be executed.

**e.g.**

try:

x=int(input("Enter number 1:"))

y=int(input("Enter number 2:"))

print(x/y)

except ZeroDivisionError :

print("Can't Divide with Zero")

except ValueError:

print("Value Error: please provide int value only")

except:

print("Unknown error occured..please check") #Default except block ,always at the end

**Single except block that can handle multiple exceptions:**

We can write a **single except block** that can handle multiple **different types of exceptions**.

except (Exception1,Exception2,exception3,..): or

except (Exception1,Exception2,exception3,..) as msg :

**try**:  
 x=int(input(**"Enter First Number: "**))  
 y=int(input(**"Enter Second Number: "**))  
 print(x/y)  
**except** (ZeroDivisionError,ValueError) **as** msg:  
 print(**"Plz Provide valid numbers only and Error is: "**,msg)  
**except**:  
 print(**"Provide valid INPUT"**)

**output:**

1)ZeroDivisionError:

Enter First Number: 12

Enter Second Number: 0

Plz Provide valid numbers only and Error is: division by zero

2)

ValueError:

Enter First Number: 12

Enter Second Number: y

Plz Provide valid numbers only and Error is: invalid literal for int() with base 10: 'y'

**Finally Block:**

The finally block will be **executed no matter if the try block raises an error or not.**

place to maintain **clean up code** which should be executed.

always irrespective of whether exception raised or not raised and whether exception

handled or not handled.

Note:

**Finally** block should be **after try except** block

Whenever we are writing finally block, **compulsory** we should write **try** block. i.e finally without try is always invalid.

We can write multiple except blocks for the same try,but we **cannot** write **multiple**

**finally blocks for the same try**

**Syntax**:

try:

Risky Code

except:

Handling Code

finally:

Cleanup code

e.g.

try:

print(10/2)

except:

print("except")

finally:

print("finally")

Output:

5.0

finally

**Case-2**: If there is an exception raised but handled:

e.g.2

try:

print(10/0)

except:

print("except")

finally:

print("finally")

**output**:

except

finally

**Case-3**: If there is an exception raised but not handled:

try:

print(10/0)

except ValueError:

print("except")

finally:

print("finally")

**Output**:

finally

**ZeroDivisionError**: division by zero

**Control flow in try-except-finally:**

try:

stmt-1

stmt-2

stmt-3

except:

stmt-4

finally:

stmt-5

stmt6

**Case-1: If there is no exception**

1,2,3,5,6 Normal Termination

**Case-2: If an exception raised at stmt2 and the corresponding except block matched**

1,4,5,6 Normal Termination

**Case-3: If an exception raised at stmt2 but the corresponding except block not matched**

1,5 Abnormal Termination

**Case-4:If an exception raised at stmt4** then it is always abnormal termination but before

that finally block will be executed.

**Case-5: If an exception raised at stmt-5 or at stmt-6** then it is always abnormal termination

**Exception in finally block:**

try:

print("Statement 1")

except ValueError:

print("except")

finally:

print(10/0) #Exception

print("statement 2")

**Output:**

Statement 1

ZeroDivisionError

Exception in statement after finally:

try:

print("Statement 1")

except ValueError:

print("except")

finally:

print("finally")

print(10/0)

**Output**:

Statement 1

Finally

ZeroDivisionError

**Nested try-except-finally blocks:**

We can take try-except-finally blocks **inside try or except or finally blocks**.i.e nesting of try-except-finally is possible.

try:

---------- Risky Code

----------

try:

------------- To Much Risky Code

except:

--------------

except:

-----------

-----------

General **Risky code** we have to take **inside outer try** block and **too much risky code** we

have to take **inside inner try block**.

Note:

**Inside Inner try** block **if an exception** raised then inner except block is responsible to handle. If it is **unable to handle then outer except block** is responsible to handle.

e.g.

try:

print("outer try block")

try:

print("Inner try block")

except ZeroDivisionError:

print("Inner except block")

finally:

print("Inner finally block")

except:

print("outer except block")

finally:

print("outer finally block")

**Output**:

outer try block

Inner try block

Inner finally block

outer finally block

e.g.2

try:

print("outer try block")

print(10/0)

try:

print("Inner try block")

except ZeroDivisionError:

print("Inner except block")

finally:

print("Inner finally block")

except:

print("outer except block")

finally:

print("outer finally block")

**Output:**

outer try block

outer except block

outer finally block

e.g.3

try:

print("outer try block")

try:

print("Inner try block")

print(10/0)

except ZeroDivisionError:

print("Inner except block")

finally:

print("Inner finally block")

except:

print("outer except block")

finally:

print("outer finally block")

**Output**:

outer try block

Inner try block

Inner except block

Inner finally block

outer finally block

e.g.

try:

print("outer try block")

try:

print("Inner try block")

print(10/0)

except NameError:

print("Inner except block")

finally:

print("Inner finally block")

except:

print("outer except block")

finally:

print("outer finally block")

Note:**Exception of inner try handled by outer Except**

**Output**:

outer try block

Inner try block

Inner finally block

outer except block

outer finally block

**else block with try-except-finally:**

We can use else block with try-except-finally blocks.

else block will **be executed if and only if there are no exceptions inside try block**.

The **try** block lets you **test a block of code for errors**. The **except** block lets you **handle** the error. **The else block lets you execute code when there is no error.**

**Syntax:**

try:

Risky Code

except:

will be executed if exception inside try

else:

will be executed if there is no exception inside try

finally:

will be executed whether exception raised or not raised and handled or not handled

Note:

Whenever we are writing else block **compulsory except block** should be there. i.e

without except we cannot write else block.

In try-except-else-finally order is important.

Eg 1:

**If No exception in try block**

try:  
 print("try")  
except:  
 print("except")  
else:  
 print("else")  
finally:  
 print("finally")

**Output**:

try

else

finally

**Eg 2: If there is exception in try block**

try:  
 print(10/0)  
except:  
 print("except")  
else:  
 print("else")  
finally:  
 print("finally")

**Output**:

Except

Finally

**CASES :**

|  |  |
| --- | --- |
| **try:**  **print("try")** | **No** |
| **except:**  **print("Hello")** | **No** |
| **else:**  **print("Hello")** | **No** |
| **finally:**  **print("Hello")** | **No** |
| **try:**  **print("try")**  **except:**  **print("except")** | **Yes** |
| **try:**  **print("try")**  **finally:**  **print("except")** | **Possible** |
| **try:**  **print("try")**  **except:**  **print("except")**  **else:**  **print("else")** | **Yes** |
| **try:**  **print("try")**  **else:**  **print("else")** | **No**  expected 'except' or 'finally' block |
| **try:**  **print("try")**  **else:**  **print("else")**  **finally:**  **print("finally")** | **No**  expected 'except' or 'finally' block |
| **try:**  **print("try")**  **except XXX:**  **print("except-1")**  **except YYY:**  **print("except-2")** | **Yes** |
| **try:**  **print("try")**  **except:**  **print("except")**  **except:**  **print("except")** | **No**  **Two default except** |
| **try:**  **print("try")**  **except ZeroDivisionError:**  **print("except")**  **except:**  **print("except")** | **Yes** |
| **try:**  **print("try")**  **except :**  **print("except-1")**  **else:**  **print("else")**  **else:**  **print("else")** | **No**  **Extra else** |
| **try:**  **print("try")**  **except :**  **print("except-1")**  **finally:**  **print("finally")**  **finally:**  **print("finally")** | **No**  **Extra finally** |
| **try:**  **print("try")**  **print("Hello")**  **except:**  **print("except")** | **No**  expected 'except' or 'finally' block |
| **try:**  **print("try")**  **except:**  **print("except")**  **print("Hello")**  **finally:**  **print("finally")** | **Not continuous flow** |
| **try:**  **print("try")**  **except:**  **print("except")**  **print("Hello")**  **else:**  **print("finally")** | **No** |
| **try:**  **print("try")**  **except:**  **print("except")**  **try:**  **print("try")**  **except:**  **print("except")** | **possible** |
| **try:**  **print("try")**  **except:**  **print("except")**  **try:**  **print("try")**  **finally:**  **print("finally")** | **possible** |
| **try:**  **print("try")**  **except:**  **print("except")**  **if 10>20:**  **print("if")**  **else:**  **print("else")** | **Try will execute**  **If-else will execute** |
| **try:  print("try")  try:  print("inner try")  except:  print("inner except block")  finally:  print("inner finally block") except:  print("except")** | **Possible**  **Try**  **Inner try**  **Inner finally** |
| **try:  print(10/0)  print("try") except:  print("except")  try:  print("inner try")  except:  print("inner except block")  finally:  print("inner finally block")**  **try:  print("try") except:  print("except") finally:  try:  print("inner try")  except:  print("inner except block")  finally:  print("inner finally block")** | **except**  **inner try**  **inner finally block** |
| **try:  print("try") except:  print("except") try:  print("try") else:  print("else")** | **SyntaxError: expected 'except' or 'finally' block** |
| **try:  print("try")  try:  print("inner try") except:  print("except")** | **SyntaxError: expected 'except' or 'finally' block** |
| **try:  print("try") else:  print("else") except:  print("except") finally:  print("finally")** | **Sequence**  **Try**  **Except**  **Else**  **Finally** |

**Types of Exceptions:**

**In Python there are 2 types of exceptions are possible.**

**1. Predefined Exceptions**

**2. User Definded Exceptions**

**1. Predefined Exceptions:**

Also known as in-built exceptions

The exceptions which are raised **automatically** by **Python virtual machine** whenver a

particular event occurs, are called pre defined exceptions.

Eg 1: Whenever we are trying to perform Division by zero, automatically Python will raise

ZeroDivisionError.

print(10/0)

**2. User Defined Exceptions:**

Also known as **Customized Exceptions or Programatic Exceptions**

Some time we have to **define** and raise exceptions **explicitly to indicate that something**

**goes wrong** ,such type of exceptions are called User Defined Exceptions or Customized

Exceptions

**Programmer is responsible** to define these exceptions and Python not having any idea

about these. Hence we have to raise explicitly based on our requirement by using "**raise"**

**keyword.**

**Creating** an Exception Class in Python is done the **same way as a regular class**. The main difference is you have to **include the Python’s base**[**Exception**](https://docs.python.org/3/library/exceptions.html#base-classes)**class** to inform the compiler that the class you’re making is an exception class. exceptions are just **regular classes that inherit from the Exception class.**

e.g.

class DemoException(Exception):

class DemoException(Exception):

pass

raise DemoException

**It’s OK to have a custom Exception subclass that only pass-es -** your exception doesn’t need to do anything **fancy** to be useful. Having custom exceptions - tailored to your specific use cases and that you can raise and catch in specific circumstances - can make your code much more readable and robust, and reduce the amount of code you write later to try and figure out what exactly went wrong.

e.g.

class IncorrectValueError(Exception):  
 def \_\_init\_\_(self, value):  
 message = f"Got an incorrect value of {value}"  
 super().\_\_init\_\_(message)  
  
my\_value = 9999  
if my\_value > 100:  
 raise IncorrectValueError(my\_value)

else:  
 print("Correct value")

**e.g.**

class DemoException(Exception):  
 def \_\_init\_\_(self, message):  
 super().\_\_init\_\_(message)  
  
  
message = "Exception Triggered! Something went wrong."  
  
  
def triggerException(num):  
 if (num == 0):  
 raise DemoException(message)  
 else:  
 print(num)  
  
  
try:  
 triggerException(0)  
 print("Code has successfully been executed.")  
except DemoException:  
 print("Error: Number should not be 0.")

**e.g.**

class TooYoungException(Exception):  
 def \_\_init\_\_(self,arg):  
 self.msg=arg  
  
class TooOldException(Exception):  
 def \_\_init\_\_(self,arg):  
 self.msg=arg  
  
age=int(input("Enter Age:"))  
if age>60:  
 raise TooOldException("Your age already crossed marriage age...no chance of getting marriage")  
  
elif age<18:  
 raise TooYoungException("Plz wait some more time you will get best match soon!!!")  
else:  
 print("You will get match details soon by email!!!")

**e.g.**

class NotfromgroupException(Exception):  
 def \_\_init\_\_(self, name):  
 message = f"{name} is not member of this group"  
 super().\_\_init\_\_(message)  
  
name=input("Enter person's name: ")  
if name == 'Priyanka' or name=='Shubham' or name=='Rohit' or name=='Madhuri' or name=='Seema' or name=='Nikhil' or name=='Sandip':  
 print("person is in group..")  
else:  
 raise NotfromgroupException(name)

In summary, creating custom exceptions in Python is as **simple as creating a new class**, but with the **Exception class as an extra argument in the class definition**. The **raise keyword is used to trigger exceptions** given the Exception Class. The try...except blocks are used to wrap one or more exceptions within a code block and modify what the code does when handling that exception and not just shutting down the program entirely.