### Python Exception

**Types of errors**

Errors in your Python programs can be categorized into at least two types-

* Syntax errors
* Exceptions

Syntax errors or compile time errors are the errors due to which your program fails to compile. Such errors are detected at the compile time itself, file name, line number, and error description are displayed so you know where to look for the error and how to correct it.

For example, if you don’t write a colon after the if statement-

def check\_equality(str1, str2):

if str1 > str2

print(str1, 'is greater than', str2)

### Exceptions

Even if a statement or expression is syntactically correct it may result in an error at run time. Errors detected during runtime are called exceptions and they disrupt the normal execution flow of the program.

For example in the following program there is an attempt to divide by zero which results in ZeroDivisionError at runtime.

defdivide\_num(num1, num2):

return num1/num2

divide\_num(10,0)

An exception can be defined as an abnormal condition in a program resulting in a disruption in the flow of the program.

Exceptions are 5 types:

1. ZeroDivisionError
2. NameError
3. IndentationError
4. IOError
5. EOFError
6. ValueError

**Exception Handling in Python using Try Block:**

Here the code which might have exception have to place in try block. The keywords used are **try, except, finally**

Syntax:

try:

suspicious code

except exception1:

block1 code

except exception2:

block2 code

Finally:

Other code

Ex1: example of ZeroDivisionError

try:

x = int(input('Enter the first number: '))

y = int(input('Enter the second number: '))

print(x/y)

except ZeroDivisionError:

print ('The second number cannot be zero!')

o/p:

Enter the first number: 10

Enter the second number: 0

The second number cannot be zero!

Ex2:example of ZeroDivisionError

try:

x = int(input('Enter the first number: '))

y = int(input('Enter the second number: '))

z=(x/y)

print("a/b = %d"%z)

except ZeroDivisionError:

print("cannot divide by zero")

else:

print("division successful")

print("end of program")

o/p:

Enter the first number: 10

Enter the second number: 2

a/b = 5

division successful

end of program

ex3: Example of IOError

try: #try block

fileptr=open("file7.txt","r")

except IOError: #except block

print("the file is not found")

else: #else block

print("the file opened successfully")

fileptr.close()

print("end of program") #other code block

o/p:

the file opened successfully

end of program

**Finally:**

A final block will be executed regardless of an error. It is an optional clause. It generally executes for external resources. It will execute even if an exception exists or else not exist.

Ex4: Example of IOError

try:

fileptr=open("file1.txt","r")

except IOError:

print("the file is not found")

else:

print("the file opened successfully")

fileptr.close()

finally:

print("i am finally")

print("end of program")

o/p:

the file is not found

i am finally

end of program

ex5: Example of IOError

try:

fileptr=open("file7.txt","r")

#fileptr.write("sample text")

f=fileptr.read()

print(f)

except IOError:

print("the file error")

else:

print("let us read")

finally:

fileptr.close()

print("file closed")

o/p:

this is file seven

let us read

file closed

ex3: Example of IOError

try:

fileptr=open("file7.txt","r")

fileptr.write("sample text")

except IOError:

print("the file error")

else:

print("let us read")

finally:

fileptr.close()

print("file closed")

o/p:

the file error

file closed

ex4: Example of ValueError

def inputnumber():

try:

r=int(input("enter a no"))

except ValueError:

print("wrong entry")

else:

print("right entry")

finally:

print("end of try")

inputnumber()

o/p:

enter a no AA

wrong entry

end of try

ex5: Example of ValueError

def inputnumber():

while(True):

try:

r=int(input("enter a no"))

except ValueError:

print("wrong entry")

continue

else:

print("right entry")

break

finally:

print("end of exceptions")

inputnumber()

o/p:

enter a noA

wrong entry

end of exceptions

enter a noA

wrong entry

end of exceptions

enter a no2

right entry

end of exceptions

**raise:**

An exception can be raised using the raise clause in python.

Syntax:

raise exception\_class

Ex1: example of ValueError

try:

age=int(input("enter age"))

if(age<18):

raise ValueError;

else:

print("the age is valid")

except ValueError:

print("the age is invalid")

finally:

print("end of exceptions")

o/p:

enter age14

the age is invalid

ex2: example of ZeroDivisionError

try:

a=int(input("enter a"))

b=int(input("enter b"))

if(b==0):

raise ZeroDivisionError;

else:

print("a/b =",a/b)

except ZeroDivisionError:

print("b cannot be 0")

o/p:

enter a12

enter b0

b cannot be 0

**Some important built in exceptions are:**

AttributeError, EOFError, FloatingPointError, ImportError, IndexError, KeyError,MemoryError, NameError, OverlfowError, RuntimeError, SyntaxError, IndentationError, TypeError, ValueError, ZeroDivisionError

**Python Custom Exceptions/User-Defined Exception:**

In Python, we can create custom exceptions by creating a new class. This exception class has to be derived either directly or indirectly.

Ex1:

class error(Exception):

pass

class Derived\_smallvalue(error):

pass

class Derived\_largevalue(error):

pass

n=10

while True:

try:

i=int(input("enter no"))

if(i<n):

raise Derived\_smallvalue

elif(i>n):

raise Derived\_largevalue

break

except Derived\_smallvalue:

print("it is less than 10")

except Derived\_largevalue:

print("it is greater than 10")

print("congrats, it is", 10)

o/p:

enter no 11

It is greater than 10

|  |  |  |
| --- | --- | --- |
| **Sr.No.** | **Name of the Exception** | **Description of the Exception** |
| **1** | **Exception** | All exceptions of Python have a base class. |
| **2** | **StopIteration** | If the next() method returns null for an iterator, this exception is raised. |
| **3** | **SystemExit** | The sys.exit() procedure raises this value. |
| **4** | **StandardError** | Excluding the StopIteration and SystemExit, this is the base class for all Python built-in exceptions. |
| **5** | **ArithmeticError** | All mathematical computation errors belong to this base class. |
| **6** | **OverflowError** | This exception is raised when a computation surpasses the numeric data type's maximum limit. |
| **7** | **FloatingPointError** | If a floating-point operation fails, this exception is raised. |
| **8** | **ZeroDivisionError** | For all numeric data types, its value is raised whenever a number is attempted to be divided by zero. |
| **9** | **AssertionError** | If the Assert statement fails, this exception is raised. |
| **10** | **AttributeError** | This exception is raised if a variable reference or assigning a value fails. |
| **11** | **EOFError** | When the endpoint of the file is approached, and the interpreter didn't get any input value by raw\_input() or input() functions, this exception is raised. |
| **12** | **ImportError** | This exception is raised if using the import keyword to import a module fails. |
| **13** | **KeyboardInterrupt** | If the user interrupts the execution of a program, generally by hitting Ctrl+C, this exception is raised. |
| **14** | **LookupError** | LookupErrorBase is the base class for all search errors. |
| **15** | **IndexError** | This exception is raised when the index attempted to be accessed is not found. |
| **16** | **KeyError** | When the given key is not found in the dictionary to be found in, this exception is raised. |
| **17** | **NameError** | This exception is raised when a variable isn't located in either local or global namespace. |
| **18** | **UnboundLocalError** | This exception is raised when we try to access a local variable inside a function, and the variable has not been assigned any value. |
| **19** | **EnvironmentError** | All exceptions that arise beyond the Python environment have this base class. |
| **20** | **IOError** | If an input or output action fails, like when using the print command or the open() function to access a file that does not exist, this exception is raised. |
| **22** | **SyntaxError** | This exception is raised whenever a syntax error occurs in our program. |
| **23** | **IndentationError** | This exception was raised when we made an improper indentation. |
| **24** | **SystemExit** | This exception is raised when the sys.exit() method is used to terminate the Python interpreter. The parser exits if the situation is not addressed within the code. |
| **25** | **TypeError** | This exception is raised whenever a data type-incompatible action or function is tried to be executed. |
| **26** | **ValueError** | This exception is raised if the parameters for a built-in method for a particular data type are of the correct type but have been given the wrong values. |
| **27** | **RuntimeError** | This exception is raised when an error that occurred during the program's execution cannot be classified. |
| **28** | **NotImplementedError** | If an abstract function that the user must define in an inherited class is not defined, this exception is raised. |