Programming Fundamentals (CT-175) **PRACTICAL WORKBOOK**

**LAB # 12**

**OBJECTIVE**

Understanding exception handling in python.

**THOERY**

An exception is an error that happens during execution of a program. When that error occurs, Python generate an exception that can be handled, which avoids your program to crash.

The try block lets you test a block of code for errors.

The except block lets you handle the error.

The finally block lets you execute code, regardless of the result of the try- and except blocks.

Exception Handling

When an error occurs, or exception as we call it, Python will normally stop and generate an error message.

These exceptions can be handled using the try statement:

Example

The try block will generate an exception, because x is not defined:

try:

print(x)

except:

print("An exception occurred")

Since the try block raises an error, the except block will be executed.

Without the try block, the program will crash and raise an error:

Example

This statement will raise an error, because x is not defined:

print(x)

56

Programming Fundamentals (CT-175) **PRACTICAL WORKBOOK**

Many Exceptions

You can define as many exception blocks as you want, e.g. if you want to execute a special block of code for a special kind of error:

Example

Print one message if the try block raises a NameError and another for other errors:

try:

print(x)

except NameError:

print("Variable x is not defined")

except:

print("Something else went wrong")

Else

You can use the else keyword to define a block of code to be executed if no errors were raised:

Example

In this example, the try block does not generate any error:

try:

print("Hello")

except:

print("Something went wrong")

else:

print("Nothing went wrong")

Finally

The finally block, if specified, will be executed regardless if the try block raises an error or not.

Example

try:

print(x)

except:

print("Something went wrong")

finally:

print("The 'try except' is finished")

This can be useful to close objects and clean up resources:

57

Programming Fundamentals (CT-175) **PRACTICAL WORKBOOK**

Example

Try to open and write to a file that is not writable:

try:

* = open("demofile.txt") f.write("Lorum Ipsum")

except:

print("Something went wrong when writing to the file")

finally:

f.close()

The program can continue, without leaving the file object open.

**Raise an exception**

As a Python developer you can choose to throw an exception if a condition occurs.

To throw (or raise) an exception, use the raise keyword.

Example

Raise an error and stop the program if x is lower than 0:

x = -1

if x < 0:

raise Exception("Sorry, no numbers below zero")

The raise keyword is used to raise an exception.

You can define what kind of error to raise, and the text to print to the user.

Example

Raise a TypeError if x is not an integer:

x = "hello"

if not type(x) is int:

raise TypeError("Only integers are allowed")

58

Programming Fundamentals (CT-175) **PRACTICAL WORKBOOK**

**Some Built in Exception in Python Classes**

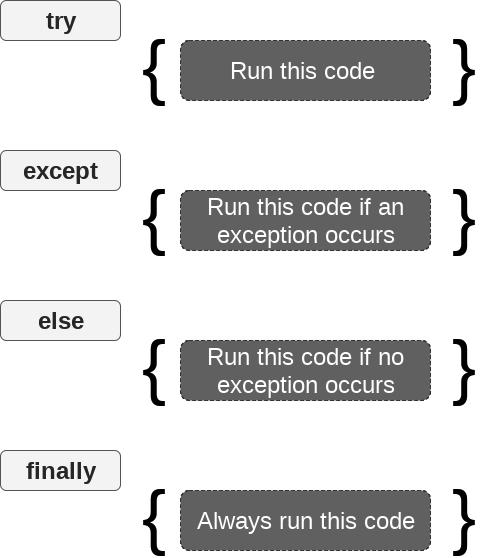
There are some built in exception in Python classes that are already defined for generic cases. They are mentioned in the table below:

|  |  |  |
| --- | --- | --- |
|  |  |  |
| **Exception Class** | | **Event** |
|  |  |  |
|  |  |  |
| IOError |  | It gets raised when an input/output operation fails. |
|  | |  |
|  | |  |
| Arithmetic Error | | It gets raised when numeric calculations fail. |
|  | |  |
|  | |  |
| Floating-point | | It gets raised when a floating-point calculation fails. |
| Error |  |  |
|  |  |  |
|  |  |  |
| Zero | Division | It gets raised when division or modulo by zero takes place for all |
| Error |  | numeric types. |
|  | |  |
|  |  |  |
| Assertion Error | | It gets raised when the assert statement fails. |
|  | |  |
|  | |  |
| Overflow Error | | It gets raised when the result of an arithmetic operation is too |
|  |  | large to be represented. |
|  | |  |
|  | |  |
| Import Error | | It gets raised when the imported module is not found. |
|  | |  |
|  | |  |
| Index Error | | It gets raised when the index of a sequence is out of range. |
|  | |  |
|  |  |  |
| Keyboard | | It gets raised when the user interrupts program execution, |
| Interrupt Error | | usually by pressing (Ctrl+C). |
|  | |  |
|  | |  |
| Indentation Error | | It gets raised when there is an incorrect indentation. |
|  | |  |
|  | |  |
| Syntax Error | | It gets raised by the parser when a syntax error is encountered. |
|  |  |  |
|  |  |  |
| Key Error |  | It gets raised when the specified key is not found in the |
|  |  | dictionary. |
|  | |  |
|  |  |  |
| Name Error | | It gets raised when an identifier is not found in the local or global |
|  |  | namespace. |
|  | |  |
|  | |  |
| Type Error | | It gets raised when a function or operation is applied to an object |
|  |  | of an incorrect type. |
|  |  |  |

59

Programming Fundamentals (CT-175) **PRACTICAL WORKBOOK**

|  |  |
| --- | --- |
|  |  |
| Value Error | It gets raised when a function gets an argument of the correct |
|  | type but of an improper value. |
|  |  |
|  |  |
| Runtime Error | It gets raised when a generated error does not fall into any |
|  | category. |
|  |  |



60

Programming Fundamentals (CT-175) **PRACTICAL WORKBOOK**

**Lab Exercise:**

1. **What will be the output (with justification) of followings: Program # 1**

|  |  |
| --- | --- |
| **Line** |  |
| **#** | **Python Code** |
|  |  |
| 1 | data = 50 |
|  |  |
| 2 | try: |
|  |  |
| 3 | data = data/0 |
|  |  |
| 4 | except ZeroDivisionError: |
| 5 | print('Cannot divide by 0 ', end = '') |
|  |  |
| 6 | else: |
|  |  |
| 7 | print('Division successful ', end = '') |
|  |  |
| 8 |  |
|  |  |
| 9 | try: |
|  |  |
| 10 | data = data/5 |
|  |  |
| 11 | except: |
|  |  |
| 12 | print('Inside except block ', end = '') |
|  |  |
| 13 | else: |
|  |  |
| 14 | print('GFG', end = '') |
|  |  |

Output:

Cannot Divide by 0 GFG

Justification: When the first try block executes, it faces the ZeroDivisionError and

falls into the except block printing the first line. Similarly, during the execution of second try block, it faces no errors and whenever there’s no error in try block, the

else part is executed….

61

Programming Fundamentals (CT-175) **PRACTICAL WORKBOOK**

|  |  |  |
| --- | --- | --- |
|  | **Program # 2** |  |
|  |  |  |
| **Line** |  |  |
| **#** | **Python Code** |  |
|  |  |  |
| 1 | value = [1, 2, 3, 4] |  |
|  |  |  |
| 2 | data = 0 |  |
|  |  |  |
| 3 | try: |  |
| 4 | data = value[3] |  |
|  |  |  |
| 5 | except IndexError: |  |
|  |  |  |
| 6 | print('CSIT IndexError ', end = '') |  |
|  |  |  |
| 7 | except: |  |
|  |  |  |
| 8 | print('NEDUET IndexError ', end = '') |  |
|  |  |  |
| 9 | finally: |  |
|  |  |  |
| 10 | print('Python IndexError ', end = '') |  |
|  |  |  |
| 11 |  |  |
| 12 | data = 10 |  |
|  |  |  |
| 13 | try: |  |
|  |  |  |
| 14 | data = data/0 |  |
|  |  |  |
| 15 | except ZeroDivisionError: |  |
|  |  |  |
| 16 | print('CSIT ZeroDivisionError ', end = '') |  |
|  |  |  |
| 17 | finally: |  |
| 18 | print('Python ZeroDivisionError ') |  |
|  |  |  |
|  |  |  |

Output:

Python IndexError CSIT ZeroDivisionError Python ZeroDivisionError

Justification: The first try block faces no error, so the except blocks are not executed

The finally block is to be executed anyhow, so we get a line printed. The second try

block faces an error, so it executes the except block, which is specifically for the

ZeroDivisionError. The finally part, same as above will be executed every time.

62