PY0101EN-3-1.2ExcecptionHandling

May 11, 2022

1 Exception Handling

Estimated time needed: **15** minutes

1.1 Objectives

After completing this lab you will be able to:

- Understand exceptions
- Handle the exceptions

1.2 Table of Contents

1.3 What is an Exception?

In this section you will learn about what an exception is and see examples of them.

1.3.1 Definition

An exception is an error that occurs during the execution of code. This error causes the code to raise an exception and if not prepared to handle it will halt the execution of the code.

1.3.2 Examples

Run each piece of code and observe the exception raised

[1]:
```
1/0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
/tmp/ipykernel_1897/2354412189.py in <module>
----> 1 1/0

ZeroDivisionError: division by zero
```

ZeroDivisionError occurs when you try to divide by zero.

[2]:
```
y = a + 5
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipykernel_1897/1161414849.py in <module>
----> 1 y = a + 5

NameError: name 'a' is not defined
```

NameError – in this case, it means that you tried to use the variable a when it was not defined.

[3]:
```
a = [1, 2, 3]
a[10]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
/tmp/ipykernel_1897/1542938915.py in <module>
      1 a = [1, 2, 3]
----> 2 a[10]

IndexError: list index out of range
```

IndexError – in this case, it occured because you tried to access data from a list using an index that does not exist for this list.

There are many more exceptions that are built into Python, here is a list of them https://docs.python.org/3/library/exceptions.html

1.4 Exception Handling

In this section you will learn how to handle exceptions. You will understand how to make your program perform specified tasks instead of halting code execution when an exception is encountered.

1.4.1 Try Except

A try except will allow you to execute code that might raise an exception and in the case of any exception or a specific one we can handle or catch the exception and execute specific code. This will allow us to continue the execution of our program even if there is an exception.

Python tries to execute the code in the try block. In this case if there is any exception raised by the code in the try block, it will be caught and the code block in the except block will be executed. After that, the code that comes after the try except will be executed.

[ ]:
```
# potential code before try catch

try:
```

```
    # code to try to execute
except:
    # code to execute if there is an exception

# code that will still execute if there is an exception
```

1.4.2 Try Except Example

In this example we are trying to divide a number given by the user, save the outcome in the variable a, and then we would like to print the result of the operation. When taking user input and dividing a number by it there are a couple of exceptions that can be raised. For example if we divide by zero. Try running the following block of code with b as a number. An exception will only be raised if b is zero.

[4]:
```
a = 1

try:
    b = int(input("Please enter a number to divide a"))
    a = a/b
    print("Success a=",a)
except:
    print("There was an error")
```

```
Please enter a number to divide a 0

There was an error
```

1.4.3 Try Except Specific

A specific try except allows you to catch certain exceptions and also execute certain code depending on the exception. This is useful if you do not want to deal with some exceptions and the execution should halt. It can also help you find errors in your code that you might not be aware of. Furthermore, it can help you differentiate responses to different exceptions. In this case, the code after the try except might not run depending on the error.

Do not run, just to illustrate:

[ ]:
```
# potential code before try catch

try:
    # code to try to execute
except (ZeroDivisionError, NameError):
    # code to execute if there is an exception of the given types

# code that will execute if there is no exception or a one that we are handling
```

[ ]:
```
# potential code before try catch
```

```
try:
    # code to try to execute
except ZeroDivisionError:
    # code to execute if there is a ZeroDivisionError
except NameError:
    # code to execute if there is a NameError

# code that will execute if there is no exception or a one that we are handling
```

You can also have an empty except at the end to catch an unexpected exception:

Do not run, just to illustrate:

[ ]:
```
# potential code before try catch

try:
    # code to try to execute
except ZeroDivisionError:
    # code to execute if there is a ZeroDivisionError
except NameError:
    # code to execute if there is a NameError
except:
    # code to execute if ther is any exception

# code that will execute if there is no exception or a one that we are handling
```

1.4.4 Try Except Specific Example

This is the same example as above, but now we will add differentiated messages depending on the exception, letting the user know what is wrong with the input.

[ ]:
```
a = 1

try:
    b = int(input("Please enter a number to divide a"))
    a = a/b
    print("Success a=",a)
except ZeroDivisionError:
    print("The number you provided cant divide 1 because it is 0")
except ValueError:
    print("You did not provide a number")
except:
    print("Something went wrong")
```

1.4.5 Try Except Else and Finally

else allows one to check if there was no exception when executing the try block. This is useful when we want to execute something only if there were no errors.

do not run, just to illustrate

[ ]:
```
# potential code before try catch

try:
    # code to try to execute
except ZeroDivisionError:
    # code to execute if there is a ZeroDivisionError
except NameError:
    # code to execute if there is a NameError
except:
    # code to execute if ther is any exception
else:
    # code to execute if there is no exception

# code that will execute if there is no exception or a one that we are handling
```

finally allows us to always execute something even if there is an exception or not. This is usually used to signify the end of the try except.

[ ]:
```
# potential code before try catch

try:
    # code to try to execute
except ZeroDivisionError:
    # code to execute if there is a ZeroDivisionError
except NameError:
    # code to execute if there is a NameError
except:
    # code to execute if ther is any exception
else:
    # code to execute if there is no exception
finally:
    # code to execute at the end of the try except no matter what

# code that will execute if there is no exception or a one that we are handling
```

1.4.6 Try Except Else and Finally Example

You might have noticed that even if there is an error the value of a is always printed. Let's use the else and print the value of a only if there is no error.

[ ]:
```
a = 1
```

```
try:
    b = int(input("Please enter a number to divide a"))
    a = a/b
except ZeroDivisionError:
    print("The number you provided cant divide 1 because it is 0")
except ValueError:
    print("You did not provide a number")
except:
    print("Something went wrong")
else:
    print("success a=",a)
```

Now lets let the user know that we are done processing their answer. Using the finally, let's add a print.

[ ]:
```
a = 1

try:
    b = int(input("Please enter a number to divide a"))
    a = a/b
except ZeroDivisionError:
    print("The number you provided cant divide 1 because it is 0")
except ValueError:
    print("You did not provide a number")
except:
    print("Something went wrong")
else:
    print("success a=",a)
finally:
    print("Processing Complete")
```

1.5 Authors

Joseph Santarcangelo

1.6 Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-09-02	2.0	Simran	Template updates to the file

##