# Python, Day 8.5: Exception Handling

Andrew Bydlon

January 22, 2019

## What is exception handling?

Up until now, the best we could do to deal with bad user input is use if-else statements to check the **type** of variable being input by the user. Today we will learn a better way to handle such issues.

We are trying to mitigate the program aborting against our wishes. As we talked about on the first day, there are many types of errors that can occur:

1. Syntax Errors: Errors in the code.
2. Runtime Errors: Errors during execution.
3. Logical Errors: Errors in logic that result in incorrect assertions being made.

## The try-except method

The idea of the try-except statement is to attempt the code in **try** section until an error is reached. If no error is reached, then the code executes in whole. Otherwise it stops at the moment of error and goes to the **except** section. The except section holds some code to execute if an exception is raised.

**Syntax**

```
try:
    SomeCodeToExecute1
    SomeCodeToExecute2
except TypeOfException:
    SomeCodeToHandleException1
    SomeCodeToHandleException2
```

## Types of exceptions

The following are possible to fill in TypeOfException:

- TypeError: Raised when two objects of the wrong type are being combined with an operator.
- ZeroDivisionError: Exactly as it sounds.
- IndexError: When the index of a list doesn't exist.
- AttributeError: Determined when a class doesn't have an assigned attribute, but you call it any ways.
- FileNotFoundError: If a file doesn't exist is read mode.
- IOError: If input/output command fails, e.g. if the disk runs out of space.
- RunTimeError: A last resort exception (no others are satisfied).

For a more complete list, see

`https://pymotw.com/2/exceptions/#typeerror`

## An example: Division by zero

Example

```
try:
    denominator = int(input("Enter a number: "))
    result = 10/denominator
    print("Result: ", result)

except ZeroDivisionError:
    print("Exception Handler for ZeroDivisionError")
    print("We cant divide a number by 0")
```

## Example execution

Example

```
>>> Enter a number: >? 5
2.0
>>> Enter a number: >? 0
Exception Handler for ZeroDivisionError
We cant divide a number by 0
```

**Note:** The program doesn't end with an error, but a print statement. Therefore, if there was more to the program, it could execute.

## Multiple Exception Types

You can add multiple exception types to increase the returned information to the user.

Example (Fix previous exercise)

```
MyFile = input("Enter file name: ")
try:
    f = open(MyFile, "r")
    for line in f:
        print(line, end="")
    f.close()
except FileNotFoundError:
    print("File not found")
except PermissionError:
    print("You don’t have the permission to read the file")
except:
    print("Unexpected error while reading the file")
```

## Finally or else

There are 2 additional commands that can be added to a **try**-statement.

1. **else:** Much like with if-elif-else statements, we can use else to execute code if none of the exceptions are raise:

2. **finally:** Can be used to execute commands **even if** an exception is raised.

Both statements are adjoined with identical syntax after all exceptions are raised.

## Raising exceptions

You can also raise your own exceptions with the raise command. It will immediately run the exception which is being raised.

Example

```
def factorial(n):
    if n < 0:
        raise RuntimeError("Factorials are for positive integers")
    if type(n)==float:
        raise RuntimeError("You can’t take the ‘factorial’ if not a \
        non-integer. I recommend the gamma function.")
    else:
        MyFactorial = 1
        for i in range(0,n+1):
            MyFactorial *= i
        return MyFactorial
```

## Finally...

**Example (Using Finally to release resources)**

```
MyFile = input("Enter file name: ")
try:
    f = open(MyFile, "a")
    f.write("I've appended some new information!")
except FileNotFoundError:
    print("File not found")
except PermissionError:
    print("You don't have the permission to read the file")
except:
    print("Unexpected error while reading the file")
finally:
    f.close()
```

## A small note about personalized exceptions

You can create your own exceptions using the class command that we have used previously. You can do this with the parent class "Exception"

**Syntax**

```
class MyException(Exception):
    def __init__(self,message):
        super().__init__()
        self.message = message
    def __str__(self):
        return self.message
```

## Assignment 13

Write a few functions to check the type of input: integer, floating point, and strings.

Then as the users for 3 pieces of input; an integer, decimal, and an alphanumeric string. As soon as they do one of these things wrong, using your functions, cause an exception to be raised (with good detailed explanation for the user). Otherwise, print good job at the end.

Try to use one of each of try/except, a raise exception, and an else or finally statement.

Upload your .py when you finish!