11.128 - Leaflet - Exceptions

July 23, 2018

1 Exceptions

An error may occur when a program is executed.

For example, this can happen when dividing by 0, or when you try to access a file that does not (no longer) exist:

```
In [1]: print(5 / 0)
```

```
        ---------------------------------------------------------------------------

        ZeroDivisionError                         Traceback (most recent call last)

        <ipython-input-1-34630cd80504> in <module>()
    ----> 1 print(5 / 0)


        ZeroDivisionError: division by zero
```

```
In [2]: with open("file.xyz", "r") as file:
            print(file)
```

```
        ---------------------------------------------------------------------------

        FileNotFoundError                         Traceback (most recent call last)

        <ipython-input-2-c081770b5d0c> in <module>()
    ----> 1 with open("file.xyz", "r") as file:
          2     print(file)


        FileNotFoundError: [Errno 2] No such file or directory: 'file.xyz'
```

Sometimes you don't want the program to close immediately if an error occurs.

With a `try ... except` - block you can catch these errors and react to them:

```
In [4]: try:
            print(5 / 0)
            print(4)
        except ZeroDivisionError:
            print("Dividing by zero is not allowed!")
        print(5)
```

```
Dividing by zero is not allowed!
5
```

1.1 Several `try` ... `except` - Blocks

Your program can also catch and respond to multiple errors via `except`:

```
In [5]: try:
            with open("file.xyz", "r") as file:
                print(file)
            print(5 / 0)
        except ZeroDivisionError:
            print("You may not divide by 0")
        except FileNotFoundError:
            print("FileNotFoundError has occured")
```

```
FileNotFoundError has occured
```

1.2 Raise your own errors

You can use the `raise` command to cause your own errors:

```
In [6]: class InvalidEmailError(Exception):
            pass

        def send_mail(email, subject, content):
            if not "@" in email:
                raise InvalidEmailError("email does not contain an @")
        try:
            send_mail("hello", "Subject", "Content")
        except InvalidEmailError:
            print("Please enter a valid email")
```

```
Please enter a valid email
```

1.3 Clean up with `finally`

If you want a particular block of code to run in any case, whether an error occurs or not, you can write that code in a `finally` block. This code is always executed, even if an error occurred before.

In this case, for example, you can guarantee that once you close a file using `.close()` (necessary if you do not open the file using `with file = open("exists.txt", "r")`).

Other examples could be, for example, that a network connection is still disconnected in any case, etc.

```
In [7]: try:
            file = open("exists.txt", "r")
            print(file)
            print(5 / 0)
        except FileNotFoundError:
            print("File not found")
        finally:
            print("FINALLY!!!")
            file.close()
```

```
<_io.TextIOWrapper name='exists.txt' mode='r' encoding='cp1252'>
FINALLY!!!
```

```
        ---------------------------------------------------------------------------

        ZeroDivisionError                         Traceback (most recent call last)

        <ipython-input-7-87bf4e8156b9> in <module>()
          2     file = open("exists.txt", "r")
          3     print(file)
    ----> 4     print(5 / 0)
          5 except FileNotFoundError:
          6     print("File not found")


        ZeroDivisionError: division by zero
```

1.3.1 The `with` - construct

In practice, however, the `with` construct is primarily suitable for files. Python has already implemented that the file is closed in any case - regardless of whether an error occurs or not.

Our code becomes much clearer:

```
In [8]: with open("exists.txt", "r") as file:
            print(file)
```

```
<_io.TextIOWrapper name='exists.txt' mode='r' encoding='cp1252'>
```