Programming fundamentals with Python

Session 7

Pepe García jgarciah@faculty.ie.edu

2020-04-20

Today's materials

You can find today's materials in https://github.com/pfp-2020/session-7.

Plan for today

- Learn about exceptions

Plan for today

- Learn about exceptions
- Do some exercises about exceptions

SCHOOL OF SCIENCE & TECHNOLOGY

Exceptions

Exceptions, or errors, are exceptional events that happen in our program. They might happen because of a lot of different reasons, such as passing a value to a function that it doesn't expect, not passing enough values, trying to open a file that's not there, etc. Python libraries communicate errors via exceptions too.

Exceptions - recovering from exceptions

Let's see an example, let's try to **read a file that doesn't exist**.

SCHOOL OF SCIENCE & TECHNOLOGY

Exceptions - recovering from exceptions

Introducing two new Python keywords, **`try`** and **`except`**. We will use **`try`** when we know that some piece of code may fail, and you want to control that failure. **`except`** is used to control the failure.

Demo

Let's control the exceptions that happens when we try to read a file that doesn't exist.

SCHOOL OF SCIENCE & TECHNOLOGY

Exercise

Exercise time. Given the following program:

```
def divide(x, y):
    return x / y
```

Print `"try another number"` if an error happens.

SCHOOL OF SCIENCE & TECHNOLOGY

Exceptions - handing different exception types

Python provides a very powerful way of matching different exceptions. We can use different **except** blocks to handle different exception types.

```
try:
    some_code_that_may_raise_exceptions()
except ValueError:
    print("You received a value error")
except TypeError:
    function_to_handle_type_errors()
```

In this example we're doing different things for recovering from different exception types.

SCHOOL OF SCIENCE & TECHNOLOGY

Exercise

Let's revisit the previous exercise:

```
def divide(x, y):
    try:
        return x / y
    except:
        print("try another number")
```

Now, two different exceptions may happen, a **`TypeError`** if we try to divide things that aren't numbers, or a **`ZeroDivisionError`** if we try to pass zero as the denominator of the division.

Let's control these independently.

Interlude - recap

So far we have seen how to use **`try`** and **`except`** for handling parts of the code that may fail. Try is used to wrap the part that may fail, and except for reacting to the failure.

We have also learned that **`except`** can be used more than once for different errors.

SCHOOL OF SCIENCE & TECHNOLOGY

Exceptions - raising exceptions

Apart of handling exeptions that may occur int he code, we can also raise our own exceptions in Python. This is useful to tell the user of our code that something went wrong, and to let them handle the code using `try-except`.

SCHOOL OF SCIENCE & TECHNOLOGY

Exceptions - raising exceptions

In order to raise exceptions, Python provides the **`raise`** keyword.

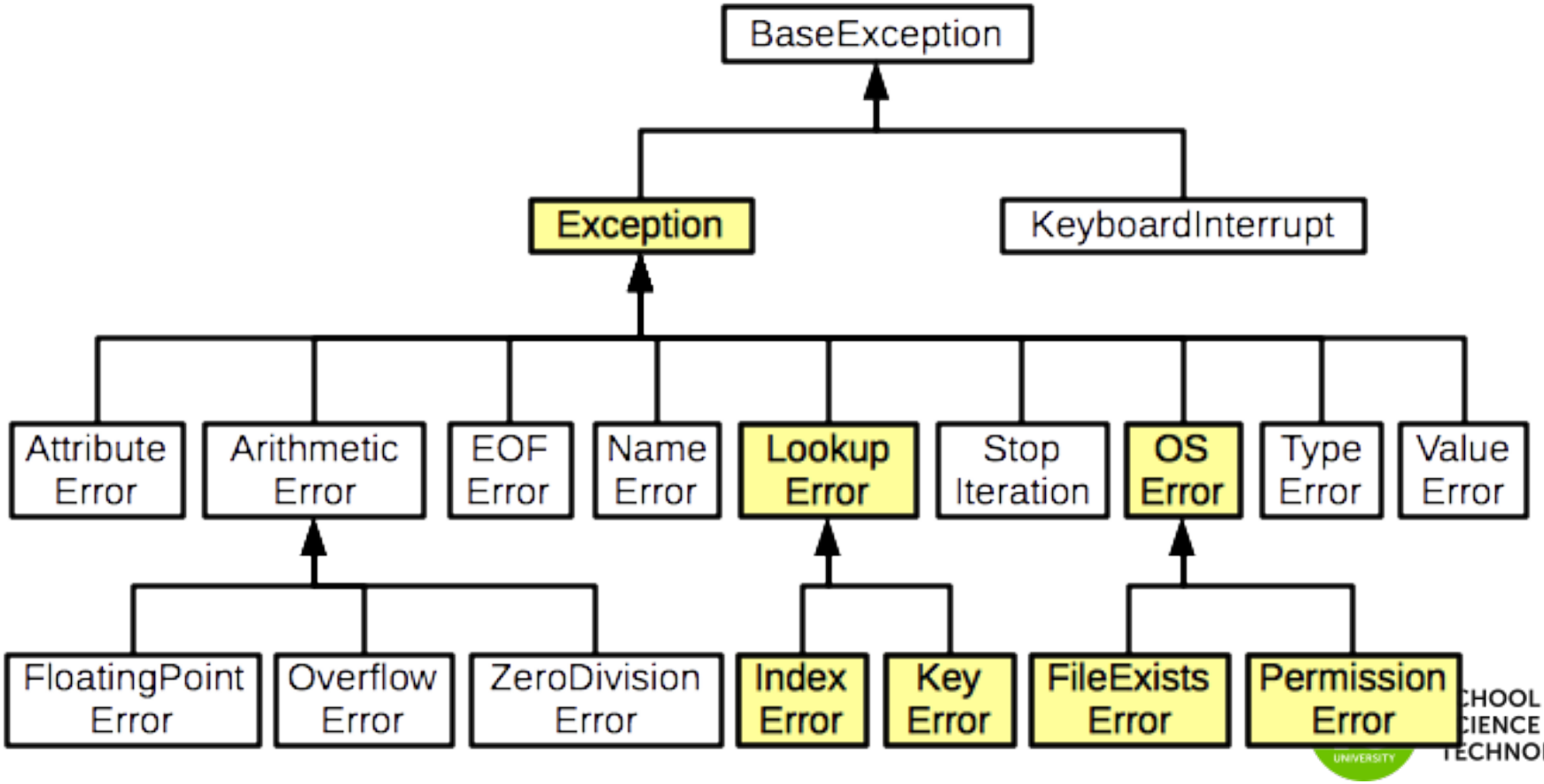
```
raise ValueError("the number is invalid")
```

As you can see, we get to choose the exception we want to raise.

SCHOOL OF SCIENCE & TECHNOLOGY

Exceptions - raising exceptions

There are a lot of exceptions built into Python. Depending on the kind of error that happened programs will raise one or another exception, or even create their own.



Exercise - raising exceptions

Let's create a function to validate emails. In this function we will do a silly validation, just ensuring that the string contains an `@` sign and a dot.

We will raise a `ValueError` in case it doesn't contain any of these, and a `TypeError` in case what we receive is not a string.

SCHOOL OF SCIENCE & TECHNOLOGY