Python advanced class

Module 6, Exceptions

- Catch of exceptions
- Standard exception hierachy
- Own exception classes
- Raise and re-raise

Catch of exceptions

Why exceptions

- An error should never pass silently ... unless explicitely silenced
- If an operation is expected to raise an exception it should run in a try block
- The default is that any error that is not handled makes the script stop running
- In old days it was the responsibility of the programmer to notice that an error occured

try except

- If an exception occur within a try block the execution is abandonned and the except block is executed

In [1]:
```
while True:
    try:
        filename = input("Enter a file to open (or enter to abort) : ")
        if filename == "":
            text = None
            break
        f = open(filename, "r", encoding="utf-8")
        text = f.read()
        f.close()
        break
    except OSError as e:
        print("The file could not be opened :", e)
```

```
The file could not be opened : [Errno 2] No such file or directo
ry: 'xxx.txt'
```

The exception object

- When an exception is raised, an exception object is passed
- It contains a text message that descibes the error
- The exception object may carry addition information

In [2]:
```
try:
    f = open("badfile.txt", "r", encoding="utf-8")
except OSError as e:
    print(e)
    print(e.errno)
    print(e.strerror)
    print(e.filename)
```

```
[Errno 2] No such file or directory: 'badfile.txt'
2
No such file or directory
badfile.txt
```

else

- If the try block when well without exception, an else block will be executed

In [3]:
```
try:
    print("The try block")
except ValueError:
    pass
else:
    print("Everything went well")
```

```
The try block
Everything went well
```

finally

- finally is always executed! Even if the exception isn't caught
- Any except blocks or else block is runned before

In [4]:
```python
try:
    x = 1/0
except ZeroDivisionError:
    print("The exception")
finally:
    print("The finally block")
```

```
The exception
The finally block
```

Standard exception hierachy

Specific to broad

- Always specify exceptions from the most specific to the broadest
- The first exception type that matches gets the exception

In [5]:
```python
def bad():
    raise FileNotFoundError("No such file!")
try:
    bad()
except FileExistsError:
    pass
except OSError as e:
    print("This")
except Exception:
    print("Catches everything")
```

```
This
```

OSError

- The `OSError` exception also carries OS Specific error information
    - `errno`: A numeric error code from the C variable errno.
    - `winerror`: Under Windows, this gives you the native Windows error code.
    - `strerror`: The corresponding error message, as provided by the operating system.
    - `filename`: For exceptions that involve a file system path (such as open() or os.unlink()), filename is the file name passed to the function.
    - `filename2`: For functions that involve two file system paths (such as os.rename()), filename2 corresponds to the second file name passed to the function.

Own exception classes

- When designing a library or application it's best to define your own exception hierachy
- If an exception is close to a built-in exception, derive from that

In [6]:
```python
class XYZBaseError(Exception):
    """The base/abstract exception of the XYZ"""
    pass
class XYZDataError(XYZBaseError):
    """Wrong format of the data for XZY"""
    pass
class XYZIllegalOrderError(XYZBaseError):
    """Illegal order of elements for XYZ"""
    pass
```

Raise and re-raise

Raise

- If a function or method encounters a fault or unsolvable situation, it should raise an exception
- Raising an exception is always better than returning a "magic" value, like -1 or an empty string

In [7]:
```python
def fac(n):
    if n < 0:
        raise ValueError(f"Negative n is not allowed, {n=}")
try:
    fac(-1)
except ValueError as e:
    print(e)
```

```
Negative n is not allowed, n=-1
```

re-raise

- When an exception is caught, but can't be handled, it must be re-raised
    - A simple `raise` alone will just raise the same exception again
    - A `raise OtherError from e` will raise the exception as another exception but keep the traceback

In [8]:

```python
import traceback
try:
    try:
        x = 1/0
    except ZeroDivisionError as e:
        raise ValueError("Zero is illegal") from e
except ValueError as err:
    print(traceback.format_exc())
```

```
Traceback (most recent call last):
  File "/tmp/ipykernel_853938/3373147313.py", line 4, in <module
>
    x = 1/0
        ~^~
ZeroDivisionError: division by zero

The above exception was the direct cause of the following except
ion:

Traceback (most recent call last):
  File "/tmp/ipykernel_853938/3373147313.py", line 6, in <module
>
    raise ValueError("Zero is illegal") from e
ValueError: Zero is illegal
```