Introduction to Information Systems and Programming

Exceptions

CentraleSupélec

SG1

Errors

A python program terminates as soon as it encounters an error :

- Syntax error : the parser detects an incorrect statement
- Exception : syntactically correct Python code results in an error

```
>>> x = 0
>>> y = 1.0 / x
Traceback (most recent call last):
  File "<input>", line 2, in <module>
ZeroDivisionError: float division by zero
```

Errors

```python
def inverse (x):
    y = 1.0 / x
    return y

a = inverse(2)
print(a)
b = inverse(0)
print(b)
```

The call stack

```
Traceback (most recent call last):
  File "/Users/hudelotc/Documents/Pistus/Data/
    exceptions.py", line 7, in <module>
    b = inverse(0)
  File "/Users/hudelotc/Documents/Pistus/Data/
    exceptions.py", line 2, in inverse
    y = 1.0 / x
ZeroDivisionError: float division by zero
```

Capturing an Exception

The `try` `and` `except` block in Python is used to catch and handle exceptions.

- Python executes code following the try statement as a "normal" part of the program.
- The code that follows the except statement is the program's response to any exceptions in the preceding try clause.

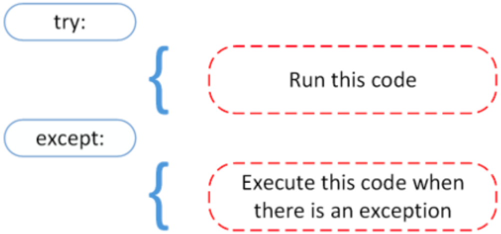

Figure – Source : RealPython

Capturing an Exception

```python
def inverse(x):
    y = 1.0 / x
    return y

try:
    a = inverse(2)
    print(a)
    b = inverse(0)  # launch an exception
    print(b)
except:
    print("The program as launched an error")
```

Capturing an Exception

General Syntax

```
try:
    # ... statements to protect
except:
    # ... what has to be done if an error occurs
else:
    # ... will be done is no error occurs
```

Capturing an Exception

Error and Exception type

By getting a variable of type `Exception`

```
def inverse(x):
    y = 1.0 / x
    return y

try:
    print(inverse(2))
    print(inverse(0))
except Exception as exc:
    print("exception of type ", type(exc).__name__)
    print("message", exc)

>>>
0.5
exception de type  ZeroDivisionError
message float division by zero
```

Capturing an Exception

The `finally` clause

An optional clause which is intended **to define clean-up actions** that must be executed under all circumstances.

```
def inverse(x):
    y = 1.0 / x
    return y
try:
    print(inverse(0))
except ZeroDivisionError:
    print("Zero division")
except Exception as exc:
    print("Error not predicted :", exc.__class__)
    print("message ", exc)
else:
    print("All is Ok")
finally:
    print("I am mastering Exception")
```

Built-in Exception

- `AttributeError`
- `OSError`
- `ImportError`
- `IndexError`
- `KeyError`
- `NameError`
- `TypeError`
- `UnicodeError`
- `ValueError`

Raising an Exception

We can use `raise` to throw an exception if a condition occurs

```python
def inverse(x):
    if x == 0:
        raise ValueError
    y = 1.0 / x
    return y

try:
    print(inverse(0))  # erreur
except ValueError:
    print("erreur de type ValueError")
```