Aide-mémoire : Exceptions/modules Python

Xavier Clerc – xavier.clerc@enpc.fr – Octobre 2020

Exceptions

Déclaration

```
class MyException(Exception):
  def __init__(self, ...):
    super().__init__()
    # ...
  def __repr__(self): # communément définie
    # ...
  def __str__(self): # communément définie
    # ...
  # ...
```

Lancement

```
raise MyException(...)
raise MyException # () omis
raise # seulement dans un bloc de traitement,
      # re-lance l'exception attrapée
```

Traitement basique

```
try:
  # ...
  # bloc protégé
  # ...
except: # attrape toute exception
  # ...
  # bloc de traitement
  # ...

try:
  # ...
  # bloc protégé
  # ...
except E1: # attrape les intances de E1
  # ...
  # bloc de traitement
  # ...
except E2: # attrape les intances de E2
  # ...
  # bloc de traitement
  # ...
except: # attrape toute exception
  # ...
  # bloc de traitement
  # ...
```

```
try:
  # ...
  # bloc protégé
  # ...
except E as e: # e référence l'instance
  # ...
  # bloc de traitement
  # ...
```

Traitement avancé

```
try:
  # ...
except ...:
  # ...
else:
  # ...
  # exécuté après le bloc try si aucune exception
  # n'est levée durant son exécution
  # ...

try:
  # ...
except ...:
  # ...
finally:
  # ...
  # exécuté après que le bloc try ou except
  # a été entièrement exécuté
  # ...
```

Si `else` et `finally` sont présent, `else` est exécuté en premier.

Exceptions built-ins

Liste non-exhaustive :

- `AttributeError` : quand la réfrence à un attribut est incorrecte
- `FileExistsError` : quand un fichier existe déjà
- `FileNotFoundError` : quand un fichier n'existe pas
- `ImportError` : quand un import échoue
- `IndexError` : quand un index est en dehors des bornes
- `KeyError` : quand une clef n'existe pas
- `ModuleNotFoundError` : quand un module n'existe pas
- `NameError` : quand un nom n'est pas défini
- `RecursionError` : quand la limite de récursion est atteinte
- `ValueError` : quand une fonction/méthode reçoit un argument incorrect
- `ZeroDivisionError` : quand une division par zéro est exécutée (entiers)

Ressources

```
with expr0 as id0, ..., exprn as idn:
  # ...

# ou de manière équivalente :

with expr0 as id0:
  # ...
  with exprn as idn:
    # ...
```

Les ressources créées par les `expr`*i* sont automatiquement fermées / nettoyées quand le programme sort du bloc `with`, même si le bloc se termine du fait d'une exception.

```
with open("my-file-name.ext", ...) as f:
  # ...
  # f peut être utilisé pour lire/écrire
  # ...
# le fichier est automatiquement fermé quand le
# block with prend fin
```

Modules

Déclaration

Un module est simplement un fichier `.py`. La hiérarchie de modules est fondée sur la hiérarchie du systme de fichiers, ainsi le fichier nommé `a/b.py` définit-il le module `a.b`. Le chemin de recherche est `sys.path` (inclus `.` et la variable d'environnement `PYTHONPATH`).

Utilisation qualifiée (préférée)

```
import module
module.element

import module as m
m.element

import module1, module2 as m, ...
module1.element
m.element
```

Utilisation non qualifiée (découragée)

```
from module import element
element

from module import element as e
e

from module import *
element
other_element
```