Débuter en python - Partie 2

Présenté par Guillaume Mousnier

Plan

1. Les erreurs
2. Les modules et les namespaces

Les erreurs

Les types d'erreurs / exceptions

Python est très expressif et n'hésite pas à lever des erreurs:

```
SyntaxError # Erreur de parsing
IndentationError # Mauvaise indentation
NameError # Variable non définie
TypeError # Erreur de typage
IndexError # Index inexistant
KeyError # Clef inexistante .....
```

Python en a beaucoup déjà définies, ce sont les built-in exceptions:

https://docs.python.org/3/library/exceptions.html

Les erreurs

Les types d'erreurs / exceptions (2)

Tout ces types d'erreurs ont un type commun: `Exception`. On dit que ces erreurs héritent de la classe ou du type `Exception`. (cf partie 2).

Il est également possible d'en créer de nouvelles, mais également des dérivés (cf partie 2).

Ainsi les modules et les librairies (modules externes) amènent leurs lots de nouveaux types d'erreurs.

Les erreurs

Gérer les erreurs

Lorsqu'une erreur est levée, elle met fin à l'éxecution du programme. Il faut les gérer pour garantir la continuité de notre programme.

On utilise alors les déclarations `try` et `except` pour les gérer.

Le `try` entoure du code à risque (susceptible de lever une exception).
Si il y a une erreur (attendue), `except` va éxecuter du code qui lui a été fourni. Le programme continue.

Les erreurs

Gérer les erreurs (2)

Programme sans try / except:

```
dict_students = {"John": 18.65, "Jacques": 10.00}
print(dict_students["Martine"])
# KeyError; Le programme s'arrête.
# Le code dessous ne s'exécute pas.
print(dict_students["John"])
```

Programme avec try / except gérant les KeyError:

```
try:
    dict_students = {"John": 18.65, "Jacques": 10.00}
    print(dict_students["Martine"])
except KeyError:
    # Le programme continue.
    print(dict_students["John"])
```

Gérer les erreurs (3)

On peut chaîner les `except` pour gérer différents types d'erreurs:

```
students_name = ["John", "Gilles", "Jacques", "Jean Charle
students_notes = {"John": 18.65, "Jacques": 10.00, "Gilles
for student in students_name:
    try:
        students_notes[student] += 1.00
    except TypeError:
        print(student, "'s score is wrong!")
        del students_notes[student]
    except KeyError:
        print(student, " doesn't have score!")

print(students_notes)
```

```
Gilles 's note is wrong!
Jean Charles  doesn't have score!
{'John': 19.65, 'Jacques': 11.0}
```

Gérer les erreurs (4)

/! Evitez d'utiliser le type Exception:

```
for student in students_name:
    try:
        students_notes[student] += 1.00
    except Exception: # Chaque erreur est une exception
         print("Il y a eu une erreur !!")
    except TypeError: # On ne passe plus ici
        print(student, "'s score is wrong!")
        del students_notes[student]
    except KeyError:  # On ne passe plus ici
        print(student, " doesn't have score!")

print(students_notes)
```

```
Il y a eu une erreur !!
Il y a eu une erreur !!
{'John': 19.65, 'Jacques': 11.0}
```

Les erreurs

Lever une erreur

Lorsque l'on écrit un programme, il peut être intéressant de lever des erreurs pour communiquer avec l'utilisateur (développeur ou autres).

On peut utiliser `raise` pour lever une erreur:

```
if value < 0:
    # on choisit de lever une erreur avec un message perso
    raise ValueError("The value can't be negative!")
```

Les modules et les namespaces

Les fonctions built-in

De nombreuses fonctions ou types sont disponibles par défaut dans le namespace (ou scope) de python, ce sont les **built-in functions**.

Par exemple, la fonction `print` est une fonction built-in.

La liste non exhaustive des built-in functions est disponible ici: https://docs.python.org/3/library/functions.html

Les modules et les namespaces

Les modules

Parfois on veut utiliser du code prêt à l'emploi autre que les built-ins. Ce sont les modules (1 module ~= 1 fichier python).

On peut ainsi importer différents modules dans son programme pour ajouter des fonctionnalités.

Certains modules sont préinstallés avec python: https://docs.python.org/3/py-modindex.html

Les modules et les namespaces

Importer un module

On peut importer un module dans son intégralité:

```
import math
print(math.pi)
```

Ou importer quelques parties du module:

```
from math import pi
print(pi)
```

Ou même tout importer d'un module: **/!**

```
from math import *
print(pi)
```

Les modules et les namespaces

Importer un module (2)

On peut aussi utiliser le système de modules pour découper son programme en plusieurs fichiers:

```
from .my_folder.my_file import my_function
```

On peut alors créer une arborescence de fichier `.py` contenant chacun une petite partie du code, que l'on importera au besoin.

Les modules et les namespaces

Comprendre les namespaces

Quand une fonction, une variable, ou un module est utilisable dans python, on dit qu'elle est présente dans le namespace (ou scope).

Les built-ins par exemple sont présents par défaut.

Pourcomprendre on utilise la fonction `dir` qui liste tout ce qui est dans le namespace:

```
print(dir())
```

```
['__annotations__', '__builtins__', '__doc__',
'__loader__', '__name__', '__package__', '__spec__']
```

Les modules et les namespaces

Comprendre les namespaces

Quand une fonction, une variable, ou un module est utilisable dans python, on dit qu'elle est présente dans le namespace (ou scope).

Les namespaces sont (~) une liste de noms de ce qui est accessible à un endroit donné du programme.

Les built-ins par exemple sont présents par défaut dans le namespace.

Les modules et les namespaces

Comprendre les namespaces (2)

Pour visualiser un namespace, on utilise la fonction `dir` qui liste tout ce qui est dans le namespace local:

```
print(dir())
```

```
['__annotations__', '__builtins__', '__doc__',
'__loader__', '__name__', '__package__', '__spec__']
```

Ainsi on voit que nos built-ins sont en effet présent dans le programme.

On peut même en voir le contenu: `print(dir(__builtins__))`

Les modules et les namespaces

Comprendre les namespaces (3)

Si on essaye d'accéder à une variable qui n'est pas présent dans le namespace local:

```
print(pi)
```

Python nous signal que le nom `pi` n'est pas présent dans le namespace local: `NameError: name 'pi' is not defined`.

Les modules et les namespaces

Comprendre les namespaces (4)

Lorsque l'on importe un module, ou que l'on déclare une variable, python l'ajoute alors au namespace:

```
from math import pi
print(pi)
print(dir())
```

Ce qui rend l'accés à une variable ou à une fonction possible:

```
3.141592653589793
[..., 'pi']
```