Rappels de Python pour la sécurité informatique

Ryan Lahfa

Avant-propos

Pendant tout le long de ce cours, si vous avez un doute, vous oubliez quelque chose, posez des questions !

[1] Potable et en anglais!

Avant-propos

Pendant tout le long de ce cours, si vous avez un doute, vous oubliez quelque chose, posez des questions !

Mais aussi, utilisez abondamment:
https://perso.limsi.fr/pointal/_media/python:cours:mementopython3-english.pdf

Enfin, voici une vidéo [1] pour apprendre Python en 4 heures:
https://www.youtube.com/watch?v=rfscVS0vtbw

[1] Potable et en anglais!

Objets de base

Expressions ou déclarations

En général, dans un langage de programmation, il faut distinguer deux types de syntaxes: les expressions et les déclarations.

Objets de base

Expressions ou déclarations

En général, dans un langage de programmation, il faut distinguer deux types de syntaxes: les expressions et les déclarations.

Vous reconnaissez les déclarations car elles ne se suffisent pas à elle même: `3 + 2` est une expression, mais `x += 1` n'en est pas une, par contre le `1` dans `x += 1` est une expression.

Mais aussi, `x = x + 1`, le `x + 1` est une expression.

Tandis que `x = x + 1` est une déclaration.

Autour des fonctions

De la même façon, on déclare des fonctions généralement en Python avec `def`, mais `lambda x: x` est une expression de fonction.

Autour des fonctions

De la même façon, on déclare des fonctions généralement en Python avec `def`, mais `lambda x: x` est une expression de fonction.

Plus intéressant, on peut déclarer des fonctions dans des fonctions et se référer au contexte lexical de la fonction:

```
def parametrer_une_fonction_lineaire(a):
    def fonction_lineaire(x): return a*x
    return fonction_lineaire
```

Vous remarquez que `fonction_lineaire` a utilisé le paramètre `a` qui dépend de la fonction plus haut, on appelle `fonction_lineare` une clôture (ou une fermeture), ce concept est assez utile pour fabriquer des abstractions sans recourir à tout l'arsenal qu'une classe offre.

Tout est un objet

Mais à la fin de la journée en Python, tout est un objet, même les entiers ou les fonctions ont des méthodes et des attributs.

Tout est un objet

Mais à la fin de la journée en Python, tout est un objet, même les entiers ou les fonctions ont des méthodes et des attributs.

Par exemple: `int.__doc__`, ou alors:

```
def f(x): return x
print(dir(f)) # Pouvez vous deviner le résultat ?
```

Notion de classe

Une classe en Python est la donnée d'un nom, éventuellement des classes parentes (héritage multiple), des méthodes, des attributs mais aussi... des métaclasses. [2]

[2] Dont nous ne parlerons pas, car 99 % du temps, personne les utilise, mais elles sont très puissantes. Donc renseignez vous ou posez moi des questions à la fin sur le sujet !

Quelques remarques:

- En Python, il n'y a pas de visibilité des attributs, tout est public mais la convention est que tout ce qui commence par `_` est privé. [3]

[3] Bonus: Si votre attribut commence par `__`, alors il est inaccessible, une exception `AttributeError` sera levée, mais on peut quand même tricher avec `__dict__`, donc ne l'utilisez pas !

Quelques remarques:

- En Python, il n'y a pas de visibilité des attributs, tout est public mais la convention est que tout ce qui commence par `_` est privé. [3]
- Contrairement à C++, la notion de `virtual` et fonctions abstraites pures n'existent pas. [4]

[3] Bonus: Si votre attribut commence par `__`, alors il est inaccessible, une exception `AttributeError` sera levée, mais on peut quand même tricher avec `__dict__`, donc ne l'utilisez pas !

[4] Il y a des équivalents avec le module `abc`, mais c'est souvent overkill.

Quelques remarques:

- En Python, il n'y a pas de visibilité des attributs, tout est public mais la convention est que tout ce qui commence par `_` est privé. [3]
- Contrairement à C++, la notion de `virtual` et fonctions abstraites pures n'existent pas. [4]
- La surcharge des opérateurs peut se faire par la surcharge des « magic methods »: https://rszalski.github.io/magicmethods/ — n'en abusez pas !

---

[3] Bonus: Si votre attribut commence par `__`, alors il est inaccessible, une exception `AttributeError` sera levée, mais on peut quand même tricher avec `__dict__`, donc ne l'utilisez pas !

[4] Il y a des équivalents avec le module `abc`, mais c'est souvent overkill.

- L'appel à la classe parente se fait par `super()` [5]

[5] Dans le cas d'héritage multiple, l'ordre de résolution devient un peu délicat et il faut parler du MRO, ce qui n'est pas l'objet de ce cours. Posez moi des questions à la fin dessus, si vous voulez savoir !

- L'appel à la classe parente se fait par `super()` [5]
- Les méthodes statiques s'implèmentent par décoration avec `@staticmethod` et `@classmethod`, le premier ne prend pas la **classe** (et non pas une **instance**) en argument, le second si.

[5] Dans le cas d'héritage multiple, l'ordre de résolution devient un peu délicat et il faut parler du MRO, ce qui n'est pas l'objet de ce cours. Posez moi des questions à la fin dessus, si vous voulez savoir !

L'introspection en Python

Qui dit langage dynamique dit assez facilement introspection.

L'introspection est un concept dans les langages de programmation qui permet d'inspecter les informations techniques d'un objet Python, e.g. sa documentation ou son code source.

L'introspection en Python

Qui dit langage dynamique dit assez facilement introspection. L'introspection est un concept dans les langages de programmation qui permet d'inspecter les informations techniques d'un objet Python, e.g. sa documentation ou son code source.

Le module `inspect` en est le plus gros fournisseur.

[6] Par contre, n'utilisez pas les commentaires comme des marqueurs, ce n'est pas une bonne idée.

L'introspection en Python

Qui dit langage dynamique dit assez facilement introspection. L'introspection est un concept dans les langages de programmation qui permet d'inspecter les informations techniques d'un objet Python, e.g. sa documentation ou son code source.

Le module `inspect` en est le plus gros fournisseur.

Cela est très utile quand on écrit des systèmes de plugins, ou qu'on manipule du métacode qu'on trouve dans des commentaires [6], bref plein de choses intéressantes quand on fait de la sécurité parfois, cela peut aider à échapper de certaines sandbox un peu faiblardes.

[6] Par contre, n'utilisez pas les commentaires comme des marqueurs, ce n'est pas une bonne idée.

Librairie standard

`collections` — **structures de données**

Le module `collections` est rempli de jolies choses dont il faut se servir sans modération, mais peu de gens ont connaissance de ce module.

Compteurs

Il s'agit de `Counter`:

- compter le nombre d'objets vérifiant une propriété
- avoir un multi-ensemble (`set` avec la propriété d'avoir plusieurs fois les mêmes objets)

Illustration : analyse statistique.

Dictionnaires à valeurs par défaut

Il s'agit de `defaultdict`:

- avoir un dictionnaire de listes, d'ensembles, d'un objet dont vous avez un constructeur par défaut.

Illustration : une sparse grid.

Liste doublement chaînées (pile & file)

Il s'agit de `deque`:

- avoir une file
- avoir une pile

Illustration : avoir une vraie file en Python.

`ctypes` — FFI

Le module `ctypes` vous permet d'utiliser les FFI[7] en Python, i.e. manipuler des fonctions venu d'un autre langage (du C/C++ souvent).

[7] Foreign Function Library

Liste doublement chaînées (pile & file)

Il s'agit de `deque`:

- avoir une file
- avoir une pile

Illustration : avoir une vraie file en Python.

`ctypes` — FFI

Le module `ctypes` vous permet d'utiliser les FFI[7] en Python, i.e. manipuler des fonctions venu d'un autre langage (du C/C++ souvent).

En pratique, on les utilise pour exposer les API systèmes d'un système d'exploitation par exemple.

[7] Foreign Function Library

Liste doublement chaînées (pile & file)

Il s'agit de `deque`:

- avoir une file
- avoir une pile

Illustration : avoir une vraie file en Python.

`ctypes` — FFI

Le module `ctypes` vous permet d'utiliser les FFI[7] en Python, i.e. manipuler des fonctions venu d'un autre langage (du C/C++ souvent).

En pratique, on les utilise pour exposer les API systèmes d'un système d'exploitation par exemple.

Illustration : `printf` en Python.

[7] Foreign Function Library

Librairie externes

Scapy & `dpkt` & `impacket`

Ce sont un ensemble de librairies qui permettent de manipuler les paquets réseaux selon des niveaux de couches plus ou moins bas, notamment pour construire des paquets réseaux malformés de protocoles, pour les tester ou pour développer des outils par dessus.

Librairie externes

Scapy & `dpkt` & `impacket`

Ce sont un ensemble de librairies qui permettent de manipuler les paquets réseaux selon des niveaux de couches plus ou moins bas, notamment pour construire des paquets réseaux malformés de protocoles, pour les tester ou pour développer des outils par dessus.

Nous aurons l'occasion de les revoir plus précisément durant le cours.

`pwnlib`

C'est un ensemble d'outils pour à la fois réussir les CTF[8] ou fabriquer des vrais exploits qui requiert de l'ingénierie classique (calculer des offsets mémoires, générer du shellcode pour une architecture donnée, etc.)

[8] Capture the Flag, concours de sécurité où on doit récupérer un flag sur une machine en piratant diverses éléments.

Unicorn Engine

Ce n'est pas une librairie Python mais un ensemble d'outils afin d'émuler les processeurs de plusieurs plateformes et plusieurs architectures assez léger.

Unicorn Engine

Ce n'est pas une librairie Python mais un ensemble d'outils afin d'émuler les processeurs de plusieurs plateformes et plusieurs architectures assez léger.

Basé sur QEMU et un outil de sécurité assez puissant et qui a mené à des tas d'outils d'analyse intéressant.