TD 6

Tuples

Un peu comme des listes (c'est une séquence qui peut contenir n'importe quoi), mais non modifiables.

```
a = 1, 2, 3
a = (1, 2, 3)
b = a
a[1] = 9  --> erreur
b?

b, c = a   --> erreur
b, c, d = a
(b, c, d) = a
b, c, d = (a)
```

Marche aussi avec une liste :

```
b, c, d = [1, 2, 3]
b, c, d = [1, 2, 3, 4]  --> erreur
```

Créer un tuple avec un seul élément :

```
t = ("truc",)
print(type(t))

# Ceci n'est pas un tuple
t = ("truc")
print(type(t))

u = t  --> copie le tuple
u, = t  --> copie le 1er élément du tuple
u[0] = t  --> copie le 1er élément du tuple
```

Exercice

Écrivez une fonction qui convertit un nombre d'heures en (jours, heures restantes).

Ex. 48 => (2, 0)
49 => (2, 0)
10 => (0, 10)

Tranches

Toutes les séquences peuvent être coupées en tranches :

```
séquence[start:stop:step]
                 ^--- s'arrête avant
```

Exercices

Créez des fonctions qui...

- les deux avant-derniers caractères d’une chaîne de longueur > 2
- radix(s) : retourne la moitié gauche d'une chaîne
- antep(s) : retourne l’avant-avant dernier caractère d'une chaîne, ou sinon l'avant dernier, ou sinon le dernier, si la chaîne est trop courte.
- straddle(s) : retourne un caractère sur deux d'un chaîne
- next_straddle(s) : retourne un caractère sur deux d'un chaîne à partir du 2e caractère
- reverse(s) : retourne la chaîne inversée

On a utilisé des chaînes pour ces exercices, mais ça marcherait pareil avec n’importe quelle séquence !

Substitutions par regex

regex.sub(pattern, repl, string, count=0, flags=0)

On peut utiliser des parenthèses dans *pattern*, et recopier leur contenu dans *repl*.

regex.sub(r"(\p{L}+) (\p{L}+)", r"\2 \1", "Nom Prénom")

D’autres parenthèses

Parenthèses non capturantes :

```
(?: parenthèse non capturante )
```

```
(?= positive lookahead )
regex.match(r"q(?=u)", "qu")  =>  q (et non qu)
```

```
(?! negative lookahead )
regex.match(r"q(?=u)", "qx")  =>  qx
regex.match(r"q(?=u)", "qu")  =>  None
```

Exercice

Écrire une fonction date_to_iso qui passe une date du format français 01/01/1900 au format international 1900-01-01.

Fréquences

Trouvez les 100 tokens les plus fréquents dans le fichier volume07.txt.

- (Avec un dictionnaire) < - - en cours
- Avec un Counter < - - **exercice A**

Initialiser un Counter :

```
Python >>>
>>> from collections import Counter

>>> Counter({"i": 4, "s": 4, "p": 2, "m": 1})
Counter({'i': 4, 's': 4, 'p': 2, 'm': 1})
```

Ajouter des trucs itérables :

update([*iterable-or-mapping*])

Elements are counted from an *iterable* or added-in from another *mapping* (or counter). Like `dict.update()` but adds counts instead of replacing them. Also, the *iterable* is expected to be a sequence of elements, not a sequence of `(key, value)` pairs.

Fonctions utiles :

elements()

Return an iterator over elements repeating each as many times as its count. Elements are returned in the order first encountered. If an element's count is less than one, `elements()` will ignore it.

```
>>> c = Counter(a=4, b=2, c=0, d=-2)
>>> sorted(c.elements())
['a', 'a', 'a', 'a', 'b', 'b']
```

most_common([*n*])

Return a list of the *n* most common elements and their counts from the most common to the least. If *n* is omitted or `None`, `most_common()` returns *all* elements in the counter. Elements with equal counts are ordered in the order first encountered:

```
>>> Counter('abracadabra').most_common(3)
[('a', 5), ('b', 2), ('r', 2)]
```

Exercice B

Comptez maintenant les tokens les + fréquents de chaque *Normalized Classification*.