# Plus sur les chaînes de caractères

## 1 Préambule

Nous avons déjà abordé les chaînes de caractères. Ici nous allons un peu plus loin, notamment avec les méthodes associées aux chaînes de caractères.

## 2 Chaînes de caractères et listes

Les chaînes de caractères peuvent être considérées comme des listes (de caractères) un peu particulières :

```
animaux = "girafe tigre"
print(animaux)
print(len(animaux))
print(animaux[3])
```

Après exécution

```
girafe tigre
12
a
```

Nous pouvons donc utiliser certaines propriétés des listes comme les tranches :

```
animaux = "girafe tigre"
print(animaux[0:4])
print(animaux[9:])
print(animaux[:-2])
print(animaux[1:-2:2])

Après exécution:
gira
gre
girafe tig
iaetg
```

Mais *a contrario* des listes, les chaînes de caractères présentent toutefois une différence notable, ce sont **des listes non modifiables**. Une fois une chaîne de caractères définie, vous ne pouvez plus modifier un de ses éléments. Le cas échéant, Python renvoie un message d'erreur :

```
animaux = "girafe tigre"
print(animaux[4])
animaux[4] = "F"

après l'exécution :
```

```
f
Traceback (most recent call last):
  File "C:/Users/Smart 140S/tp python/chaine1.py", line 3, in <module>
    animaux[4] = "F"
TypeError: 'str' object does not support item assignment
```

Par conséquent, si vous voulez modifier une chaîne de caractères, vous devez en construire une nouvelle. Pour cela, n'oubliez pas que les opérateurs de concaténation (+) et de duplication (*) peuvent vous aider. Vous pouvez également générer une liste, qui elle est modifiable, puis revenir à une chaîne de caractères (voir plus bas).

# 3 Caractères spéciaux

Il existe certains caractères spéciaux comme `\n` que nous avons déjà vu (pour le retour à la ligne). Le caractère `\t` produit une tabulation. Si vous voulez écrire des guillemets simples ou doubles et que ceux-ci ne soient pas confondus avec les guillemets de déclaration de la chaîne de caractères, vous pouvez utiliser `\'` ou `\"`.

```
print("Un retour à la ligne\npuis une tabulation\t puis un
guillemet\"")
print('J\'affiche un guillemet simple')

après exécution

Un retour à la ligne
puis une tabulation     puis un guillemet"
J'affiche un guillemet simple
```

Vous pouvez aussi utiliser astucieusement des guillemets doubles ou simples pour déclarer votre chaîne de caractères :

```
print("Un brin d'ADN")
print('Python est un "super" langage de programmation')

après exécution :
Un brin d'ADN
Python est un "super" langage de programmation
```

Quand on souhaite écrire un texte sur plusieurs lignes, il est très commode d'utiliser les guillemets triples qui conservent le formatage (notamment les retours à la ligne) :

```
x = """souris
chat
abeille"""
print(x)

après exécution :
```

```
souris
chat
abeille
```

# 4 Méthodes associées aux chaînes de caractères

Voici quelques méthodes spécifiques aux objets de type `str` :

```
>>> x = "girafe"
>>> x.upper()
'GIRAFE'
>>> x
'girafe'
>>> 'TIGRE'.lower()
'tigre'
```

Les méthodes `.lower()` et `.upper()` renvoient un texte en minuscule et en majuscule respectivement. On remarque que l'utilisation de ces méthodes n'altère pas la chaîne de caractères de départ mais renvoie une chaîne de caractères transformée.

Pour mettre en majuscule la première lettre seulement, vous pouvez faire :

```
>>> x[0].upper() + x[1:]
'Girafe'
```

ou plus simplement utiliser la méthode adéquate :

```
>>> x.capitalize()
'Girafe'
```

Il existe une méthode associée aux chaînes de caractères qui est particulièrement pratique, la méthode `.split()` :

```
>>> animaux = "girafe tigre singe souris"
>>> animaux.split()
['girafe', 'tigre', 'singe', 'souris']
>>> for animal in animaux.split():
...     print(animal)
...
girafe
tigre
singe
souris
```

La méthode `.split()` découpe une chaîne de caractères en plusieurs éléments appelés *champs*, en utilisant comme séparateur n'importe quelle combinaison « d'espace(s) blanc(s) ».

**Définition**

Un espace blanc (*whitespace* en anglais) correspond aux caractères qui sont invisibles à l'œil, mais qui occupent de l'espace dans un texte. Les espaces blancs les plus classiques sont l'espace, la tabulation et le retour à la ligne.

Il est possible de modifier le séparateur de champs, par exemple :

```
>>> animaux = "girafe:tigre:singe::souris"
>>> animaux.split(":")
['girafe', 'tigre', 'singe', '', 'souris']
```

Attention, dans cet exemple, le séparateur est un seul caractères « : » (et non pas une combinaison de un ou plusieurs :) conduisant ainsi à une chaîne vide entre `singe` et `souris`.

Il est également intéressant d'indiquer à `.split()` le nombre de fois qu'on souhaite découper la chaîne de caractères avec l'argument `maxsplit` :

```
>>> animaux = "girafe tigre singe souris"
>>> animaux.split(maxsplit=1)
['girafe', 'tigre singe souris']
>>> animaux.split(maxsplit=2)
['girafe', 'tigre', 'singe souris']
```

La méthode `.find()`, quant à elle, recherche une chaîne de caractères passée en argument :

```
>>> animal = "girafe"
>>> animal.find("i")
1
>>> animal.find("afe")
3
>>> animal.find("z")
-1
>>> animal.find("tig")
-1
```

Si l'élément recherché est trouvé, alors l'indice du début de l'élément dans la chaîne de caractères est renvoyé. Si l'élément n'est pas trouvé, alors la valeur `-1` est renvoyée.

Si l'élément recherché est trouvé plusieurs fois, seul l'indice de la première occurrence est renvoyé :

```
>>> animaux = "girafe tigre"
>>> animaux.find("i")
1
```

On trouve aussi la méthode `.replace()` qui substitue une chaîne de caractères par une autre :

```
>>> animaux = "girafe tigre"
>>> animaux.replace("tigre", "singe")
'girafe singe'
>>> animaux.replace("i", "o")
'gorafe togre'
```

La méthode `.count()` compte le nombre d'occurrences d'une chaîne de caractères passée en argument :

```
>>> animaux = "girafe tigre"
>>> animaux.count("i")
2
>>> animaux.count("z")
0
>>> animaux.count("tigre")
1
```

La méthode `.startswith()` vérifie si une chaîne de caractères commence par une autre chaîne de caractères :

```
>>> chaine = "Bonjour monsieur le capitaine !"
>>> chaine.startswith("Bonjour")
True
>>> chaine.startswith("Au revoir")
False
```

Enfin, la méthode `.strip()` permet de « nettoyer les bords » d'une chaîne de caractères :

```
>>> chaine = "  Comment enlever les espaces au début et à la fin ?       "
>>> chaine.strip()
'Comment enlever les espaces au début et à la fin ?'
```

La méthode `.strip()` enlève les espaces situés sur les bords de la chaîne de caractère mais pas ceux situés entre des caractères visibles. En réalité, cette méthode enlève n'importe quelle combinaison « d'espace(s) blanc(s) » sur les bords, par exemple :

```
>>> chaine = "  \tfonctionne avec les tabulations et les retours à la
ligne\n"
>>> chaine.strip()
'fonctionne avec les tabulations et les retours à la ligne'
```

La méthode `.strip()` est très pratique quand on lit un fichier et qu'on veut se débarrasser des retours à la ligne.

# 5 Extraction de valeurs numériques d'une chaîne de caractères

Une tâche courante en Python est de lire une chaîne de caractères (provenant par exemple d'un fichier), d'extraire des valeurs de cette chaîne de caractères puis ensuite de les manipuler.

On considère par exemple la chaîne de caractères `val` :

```
>>> val = "3.4 17.2 atom"
```

On souhaite extraire les valeurs `3.4` et `17.2` pour ensuite les additionner.

Dans un premier temps, on découpe la chaîne de caractères avec la méthode `.split()` :

```
>>> val2 = val.split()
>>> val2
['3.4', '17.2', 'atom']
```

On obtient alors une liste de chaînes de caractères. On transforme ensuite les deux premiers éléments de cette liste en *floats* (avec la fonction `float()`) pour pouvoir les additionner :

```
>>> float(val2[0]) + float(val2[1])
20.599999999999998
```

**Remarque**

Retenez bien l'utilisation des instructions précédentes pour extraire des valeurs numériques d'une chaîne de caractères. Elles sont régulièrement employées pour analyser des données depuis un fichier.

# 6 Conversion d'une liste de chaînes de caractères en une chaîne de caractères

On a vu dans le chapitre 2 *Variables* la conversion d'un type simple (entier, *float* et chaîne de caractères) en un autre avec les fonctions `int()`, `float()` et `str()`. La conversion d'une liste de chaînes de caractères en une chaîne de caractères est un peu particulière puisqu'elle fait appelle à la méthode `.join()`.

```
>>> seq = ["A", "T", "G", "A", "T"]
>>> seq
['A', 'T', 'G', 'A', 'T']
>>> "-".join(seq)
'A-T-G-A-T'
>>> " ".join(seq)
'A T G A T'
>>> "".join(seq)
'ATGAT'
```

Les éléments de la liste initiale sont concaténés les uns à la suite des autres et intercalés par un séparateur qui peut être n'importe quelle chaîne de caractères. Ici, on a utilisé un tiret, un espace et rien (une chaîne de caractères vide).

Attention, la méthode `.join()` ne s'applique qu'à une liste de chaînes de caractères.

```
>>> maliste = ["A", 5, "G"]
>>> " ".join(maliste)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: sequence item 1: expected string, int found
```