Université Paris Dauphine **Licence MIE 1re année**
Algorithmique et programmation 1 **2025-2026**

5. Chaînes de caractères en Python (cours)

5.1 Type `str`

Par caractère, on entend l'ensemble des caractères alphabétiques latins minuscules ou majuscules 'a', 'b', 'A', 'B',... les chiffres '0','1',... les signes de ponctuations, l'espace, les caractères spéciaux...

Caractères spéciaux :

— `'\n'` : caractère de saut de ligne
— `'\t'` : caractère de tabulation

Les caractères non accentués en minuscule et en majuscule, les signes de ponctuation, les caractères spéciaux et les chiffres sont codés en machine de façon normalisée. Ainsi quel que soit le système utilisé (Mac, Windows ou Unix), chaque caractère est représenté par une même valeur numérique (appelée son code ASCII = *American Standard Code for Information Interchange).* Pour les caractères accentués (dont l'utilisation doit être à proscrire dans les identificateurs), plusieurs normalisations coexistent. On risque donc de faire face à des problèmes d'encodage. Par exemple, le 'é' pouvant ne pas être représenté par le même code sur un Mac ou sur un PC, la lecture du fichier contenant ce caractère ne donnera pas le même résultat sur les deux systèmes.

Une suite de caractères est appelée une chaîne de caractères. En Python, une chaîne de caractère est de type `str`. Une valeur de type `str` est donc une suite ordonnée de n caractères (si $n = 0$, il s'agit de la chaine vide).

La longueur d'une chaîne de caractères est égale au nombre de caractères qu'elle contient (incluant les caractères spéciaux, caractères d'espacement...).

Une valeur de type `str` est toujours notée entre guillemets ou entre apostrophes (appelées *quotes* en informatique) : ce sont les **délimiteurs** de la chaîne. Ces délimiteurs permettent de distinguer la valeur de type `str` des valeurs numériques, des identificateurs de variables, des mots-clés du langage. Par exemple, `423` est le nombre quatre cent vingt-trois alors que `"423"` représente la suite des caractères chiffres '4', '2', '3'. L'autre confusion, bien plus grave – et bien plus fréquente – concerne les noms de variables et les chaînes de caractères : `bonjour` sans guillemets représente le nom d'une variable qui sera évaluée pour en connaître son contenu, alors que `"bonjour"` est une valeur de type `str` qui ne sera pas évaluée.

Pour délimiter une valeur de type `str`, on peut utiliser indifféremment des guillemets ou des quotes. Cela présente l'avantage d'insérer des quotes ou des guillemets dans la chaîne de caractères sans poser de problème. Exemples :

— pour la chaîne `c'est`, on doit écrire : `"c'est"`. Dans ce cas, le délimiteur de chaîne est le guillemet et le caractère apostrophe n'est pas interprété comme un délimiteur de chaîne (alors que `'c'est'` va renvoyer une erreur) ;
— pour la chaîne `il a dit "bonjour"`, on écrit : `'il a dit "bonjour"'`. Ici, le délimiteur de chaîne est la quote et les guillemets ne sont pas interprétés comme des délimiteurs de chaîne (alors que `"il a dit "bonjour""` va renvoyer une erreur) ;

— si on veut stocker la chaîne `c'est "bonjour"`, on ne peut utiliser ni les quotes, ni les guillemets sans commettre une erreur. Une solution est alors d'indiquer à l'interpréteur Python, que les caractères `'` et `"` ne doivent pas être interprétés : on doit écrire :

```
"c\'est \"bonjour\""
```

Le caractère `\` est spécial et empêche les caractères `'` et `"` de jouer leur role de délimiteur de chaîne.

Une valeur de type `str` est une suite de caractères ; il est donc possible d'accéder à chacun de ses caractères. Par exemple,
`'bonjour'[0]` renvoie `'b'`
`'bonjour'[i]` avec `i` compris entre 0 et 6 renvoie le caractère en position `i+1`.

On peut définir une variable référençant une valeur de type `str` et accéder à chacun de ses caractères. Par exemple,
— `ch = "bonjour"`
— `ch[i]` avec `i` compris entre 0 et 6 renvoie le caractère en position `i+1`
Pour définir une variable référençant une chaîne vide, on exécutera l'affectation suivante : `ch = ""`

En Python, une chaîne n'est pas une séquence de caractères car les éléments ne peuvent pas être modifiés : les opérateurs de suppression et d'insertion ne sont pas définis. Il s'agit d'une valeur **non modifiable** comme l'est une valeur de type `str`, `int`, `float` ou `bool`. En conséquence, soit la variable `ch = 'bonjour'`, l'instruction

```
ch[0] = 'B'
```

ne peux pas s'exécuter. Le message d'erreur est :

```
>>> ch[0] = 'B'
Traceback (most recent call last):
  File "<console>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

En Python on ne peut pas modifier une valeur de type `str`, mais il existe des opérateurs pour en construire une nouvelle.

5.2 Opérateurs Python sur les chaînes de caractères

L'extraction
`ch[n:m]` désigne la sous-chaîne de `ch` formée par les caractères situés entre les indices `n` et `m-1`. Si `n` est supérieur ou égal à `m`, `ch[n:m]` renvoie la chaîne vide.
Variantes :

`ch[n:]` est la sous-chaîne allant de l'indice `n` à la fin,
`ch[:m]` est la sous-chaîne allant de l'indice 0 à l'indice `m-1`,
`ch[::-1]` est la chaîne `ch` inversée (si `ch="bon"` alors `ch[::-1]` est égale à `"nob"`)

Concaténation = mise bout à bout de deux chaînes.

L'opérateur binaire + permet de concaténer deux chaînes de caractères.

```
ch = 'Python' + ' pour tous'
```

```
ch1 = 'Python'
ch2 = ' pour tous'
ch = ch1 + ch2
```

Dans les deux cas, `ch` contient `'Python pour tous'`.

Les opérateurs d'extraction et de concaténation permettant de créer de nouvelles chaînes de caractères. Par exemple, pour remplacer dans `ch` le premier caractère `'P'` par une minuscule `'p'`, on exécutera l'affectation suivante :

```
ch = 'p'+ch[1:]
```

Duplication = recopie plusieurs occurrences d'une même chaîne.

Après l'instruction suivante

```
ch = ch*2
```

`ch` contient `"Python pour tousPython pour tous"`.

Remarque : L'opérateur * est l'unique opérateur qui mélange dans une même expression des opérandes de type `str` et `int`.

Test d'appartenance.
L'opérateur binaire `in` est un opérateur booléen. Lorsque ses deux opérandes sont de type `str`, l'expression

```
ch1 in ch2
```

renvoie une valeur de type `bool`. Son évaluation renvoie `True` si `ch1` est incluse dans `ch2` et `False` sinon.

Exemple :

```
>>> ch = 'Python'
>>> 'thon' in ch
True
>>> 'toutes' in ch
False
```

Opérateurs de comparaison sur `str` Soit `ch1` et `ch2` deux variables référençant des valeurs de type `str`

- `ch1 == ch2` renvoie `True` si `ch1` et `ch2` référencent les mêmes suites de caractères (les mêmes caractères aux mêmes positions !) et `False` sinon (idem pour `!=`). Par exemple :
  ```
  >>> ch1, ch2 = 'bonjour', ' bonjour'
  >>> ch1 == ch2
  False
  ```
  car `ch2` commence par une espace !
- `ch1 < ch2` renvoie `True` si `ch1` précède `ch2` dans l'ordre alphabétique et `False` sinon (idem pour <=, >, >=)
  ```
  >>> 'ana' > 'aba'
  True
  >>> 'ana' <= 'Ana'
  False
  ```

5.3 Fonctions et méthodes prédéfinies en Python

Syntaxe pour l'appel de fonction : `nom_fonction(chaine)`

Étant donnée une variable `ch` de type `str`, on peut appeler les fonctions natives, déjà intégrées ('built-in') dans l'interpréteur, suivantes :

— `len(ch)` : `len` (abréviation de *length*) renvoie le nombre de caractères contenus dans la chaîne de caractères stockée dans `ch`. Ainsi, `len('python')` renvoie 6.

```
ch = 'python'
longueur = len(ch)
car = ch[longueur-1]
```

`car` est une variable de type `str` qui contient alors `'n'`

— `chr(i)` avec `i` une valeur de type `int` : renvoie le caractère dont la valeur numérique de codage est `i`.

— `ord(car)` avec `car` une valeur de type `str` de longueur 1 : renvoie la valeur numérique de codage du caractère `car`.

— `eval(ch)` : évalue le contenu de la chaîne `ch` comme étant une expression Python. Par exemple, `eval('5+7')` renvoie la valeur 12 de type `int`.

En Python, une chaîne de caractères est un **objet**. Comme on vient de le voir, on peut appeler des fonctions prédéfinies sur un objet mais pas seulement... Un objet dispose d'une partie **données** (la chaîne de caractères elle-même) et d'une partie **méthodes**. Une méthode d'un objet est une fonction qui s'applique à l'objet lui-même et qui renvoie une valeur.

Syntaxe pour l'appel de méthode : `chaine.nom_methode()`

Étant données des variables `ch` et `ch1` de type `str`, et `p` de type `int`, on peut appeler les méthodes suivantes qui renvoient une valeur de type `int` :

— `ch.find(ch1,p)` : renvoie l'indice de début de la sous-chaîne `ch1` dans `ch` (en commençant la recherche à partir de l'indice `p`) et renvoie -1 si `ch1` n'est pas une sous-chaîne de `ch` à partir de l'indice `p`. L'argument `p` est optionnel : quand il est omis – `ch.find(ch1)` – `ch1` est cherché dans `ch` à partir de l'indice 0.

— `ch.count(ch1)` : renvoie le nombre d'occurrences de `ch1` dans `ch`.

Étant données des variables `ch`, `ch1` et `ch2` de type `str`, on peut appeler les méthodes ci-dessous qui renvoient une valeur de type `str`. Attention, dans chacune de ces trois méthodes, `ch` est inchangée après l'exécution de la méthode. Quand on applique une méthode sur une variable de type `str`, la méthode ne modifie pas le contenu de la variable. Si on veut récupérer le résultat de la méthode pour modifier le contenu de la variable, il faut l'affecter à cette variable.

— `ch.replace(ch1,ch2)` : renvoie une **nouvelle** chaîne construite à partir de `ch` dans laquelle toutes les occurrences de `ch1` sont remplacées par `ch2` (attention, `ch` n'est pas modifiée).

```
>>> ch = "Langage C"
>>> ch.replace("C","Python")
 'Langage Python'
>>> ch
 'Langage C'
>>> ch = ch.replace("C","Python")
```

```
>>> ch
 'Langage Python'
```

- `ch.lower()` et `ch.upper()` : renvoie une nouvelle chaîne construite à partir de `ch`, respectivement transformée en minuscule et en majuscule (attention, `ch` n'est pas modifiée).
- `ch.isalpha()` : renvoie la valeur `True` si tous caractères de ch sont des lettres de l'alphabet et la valeur `False` sinon.
- `ch.strip()` : renvoie une valeur de type `str` qui correspond à `ch` dans laquelle ont été enlevées les espaces en début et en fin de chaîne. Si on veut enlever d'autres caractères en début et fin de chaîne `ch`, on écrit : `ch.strip(ch1)` et tous les caractères contenus dans `ch1` seront enlevés en début et fin de `ch`.

`dir(str)` pour afficher toutes les méthodes Python qui s'appliquent au type `str` (de façon générale on peut écrire `dir(nom_type)`).
`help(type.methode)` pour afficher des explications à propos d'une méthode. Exemple : `help(str.strip)`.

5.4 Parcours d'une chaîne de caractères

Pour parcourir les caractères d'une chaîne `ch` à l'aide d'une boucle `while` on exécutera :

```
i = 0
while i < len(ch):
    Bloc d'instructions (ch[i] permet d'accéder aux différents caractères
    i = i + 1
```

Exemple 1 : Compter le nombre de virgules dans un texte

```
texte =  "L'accord de Paris, souvent appelé accord de Paris sur le climat
cpt = 0
i = 0
while i < len(texte):
    if texte[i] == ',':
        cpt += 1
    i = i + 1
print('Le nombre de virgules est :', cpt)
```

Exemple 2 : Calculer la longueur d'une chaîne sans utiliser la fonction `len()`

```
ch = 'biodiversité'
chTemp = ch
longueur = 0
while chTemp != '':
    longueur = longueur + 1
    chTemp = chTemp[1:]
print('La longueur de', ch, 'est :', longueur)
```

Dans cet exemple, plutôt que de faire varier les indices de la chaîne, on modifie la chaîne en la parcourant.