

Informatique S3 :
Structures de données en Python
Outils Python pour les data science
L2 MIASHS

Laurent Boyer

Aujourd'hui – cours 4

Chaînes de caractères

Lecture et écriture dans des fichiers

Section 1

Chaînes de caractères

Chaînes de caractères

En python c'est le type *String*, noté `str`.

► **Définition :**

Ce sont des séquences immuables de caractères.

Chaînes de caractères

En python c'est le type *String*, noté `str`.

► **Définition :**

Ce sont des séquences immuables de caractères.

```
1 s1 = "La maison est "  
2 s2 = "rouge."  
3 s3 = s1 + s2  
4 print(s3)  
5 print(len(s2))
```



```
La maison est rouge.  
6
```

Création et conversion

On peut obtenir un `str` de différente manière :

- ▶ déclaration littérale (guillemets simples ou doubles):

```
1 s1 = "bonjour"  
2 s2 = 'rouge'
```

- ▶ conversion à partir d'un autre type :

```
1 s3 = str(1.3)  
2 s4 = str([1,2,3])
```

Ici l'affichage des chaînes donne :

```
bonjour  
rouge  
1.3  
[1, 2, 3]
```

Parcours et appartenance

On peut parcourir des chaînes ou tester l'appartenance avec la même syntaxe que pour les listes :

```
1 s = "ro ge"
2 for x in s :
3     print(x)
4 if "r" in s and "ro" in s :
5     print("r et ro y sont.")
```

Parcours et appartenance

On peut parcourir des chaînes ou tester l'appartenance avec la même syntaxe que pour les listes :

```
1 s = "ro ge"  
2 for x in s :  
3     print(x)  
4 if "r" in s and "ro" in s :  
5     print("r et ro y sont.")
```



```
r  
o  
  
g  
e  
  
r et ro y sont.
```

Découpage (1)

► On accède aux caractères et on fait des tranches (slicing) comme pour les listes :

```
1 s1 = "bonjour"
2 s2 = s1[3]
3 s3 = s1[4:6]
4 s4 = s1[:6:2]
5 print(s2)
6 print(s3)
7 print(s4)
```



Découpage (1)

► On accède aux caractères et on fait des tranches (slicing) comme pour les listes :

```
1 s1 = "bonjour"
2 s2 = s1[3]
3 s3 = s1[4:6]
4 s4 = s1[:6:2]
5 print(s2)
6 print(s3)
7 print(s4)
```



j
ou
bno

Découpage (2)

```
1 s1 = "bonjour"
2 s5 = s1[5:1:-2]
3 s6 = s1[5::2]
4 s7 = s1[:3:-1]
5 print(s5)
6 print(s6)
7 print(s7)
```



Découpage (2)

```
1 s1 = "bonjour"
2 s5 = s1[5:1:-2]
3 s6 = s1[5::2]
4 s7 = s1[:3:-1]
5 print(s5)
6 print(s6)
7 print(s7)
```



uj
u
ruo

Découpage (2)

```
1 s1 = "bonjour"
2 s5 = s1[5:1:-2]
3 s6 = s1[5::2]
4 s7 = s1[:3:-1]
5 print(s5)
6 print(s6)
7 print(s7)
```



uj
u
ruo

Retenir :

- ▶ rôle des indices négatifs,
- ▶ valeur par défaut des *paramètres*.

split

► `split` est très pratique pour découper une chaîne de caractères selon un délimiteur.

`s.split(t)` retourne une liste des morceaux de `s`, découpés selon `t`.

```
1 s1 = "Là-bas, nous passons quelques jours."  
2 s2 = s1.split("ou")  
3 print(s2)  
4 s3 = s1.split()  
5 print(s3)
```



split

► `split` est très pratique pour découper une chaîne de caractères selon un délimiteur.

`s.split(t)` retourne une liste des morceaux de `s`, découpés selon `t`.

```
1 s1 = "Là-bas, nous passons quelques jours."  
2 s2 = s1.split("ou")  
3 print(s2)  
4 s3 = s1.split()  
5 print(s3)
```



```
['Là-bas, n', 's passons quelques j', 'rs.']  
['Là-bas,', 'nous', 'passons', 'quelques', 'jours.']
```

join

► join fait l'inverse : à partir d'une liste de strings, il les recolle en intercallant un délimiteur.

`t.join(l)` retourne un string constitué des morceaux de `l` recollés avec `t`.

```
1 l1 = ["H","e","l","l","o"," ","W"]
2 l2 = ["histoire", "de", "toute", "les"]
3 s1 = " ".join(l1)
4 s2 = "".join(l2)
5 print (s1)
6 print (s2)
```



join

► join fait l'inverse : à partir d'une liste de strings, il les recolle en intercallant un délimiteur.

`t.join(l)` retourne un string constitué des morceaux de `l` recollés avec `t`.

```
1 l1 = ["H","e","l","l","o"," ", "W"]
2 l2 = ["histoire", "de", "toute", "les"]
3 s1 = " ".join(l1)
4 s2 = "".join(l2)
5 print (s1)
6 print (s2)
```



```
H e l l o W
histoiredetouteles
```

join

► join fait l'inverse : à partir d'une liste de strings, il les recolle en intercallant un délimiteur.

`t.join(l)` retourne un string constitué des morceaux de `l` recollés avec `t`.

```
1 l1 = ["H","e","l","l","o"," ","W"]
2 l2 = ["histoire", "de", "toute", "les"]
3 s1 = " ".join(l1)
4 s2 = "".join(l2)
5 print (s1)
6 print (s2)
```



```
H e l l o W
histoiredetouteles
```

► Attention à l'ordre des paramètres...

Méthodes

Nombreuses pour les `str`.

(`str` immuable $\Rightarrow$ méthodes fabriquent de nouvelles chaînes.)

- ▶ Manipulation

- ▶ +
- ▶ upper / lower / capitalize
- ▶ find / replace
- ▶ ...

Méthodes

Nombreuses pour les `str`.

(`str` immuable $\Rightarrow$ méthodes fabriquent de nouvelles chaînes.)

- ▶ Manipulation

- ▶ +
- ▶ upper / lower / capitalize
- ▶ find / replace
- ▶ ...

- ▶ Information

- ▶ isalnum / isnumeric / isspace /

Méthodes

Nombreuses pour les `str`.

(`str` immuable $\Rightarrow$ méthodes fabriquent de nouvelles chaînes.)

► Manipulation

- +
- upper / lower / capitalize
- find / replace
- ...

► Information

- isalnum / isnumeric / isspace /

La référence :

<https://docs.python.org/fr/3.6/library/stdtypes.html#text-sequence-type-str>

Méthodes

Nombreuses pour les `str`.

(`str` immuable $\Rightarrow$ méthodes fabriquent de nouvelles chaînes.)

► Exemple :

```
1 s1 = "Bonjour !"
2 s2 = s1.upper()
3 print(s1)
4 print(s2)
```



Méthodes

Nombreuses pour les `str`.

(`str` immuable $\Rightarrow$ méthodes fabriquent de nouvelles chaînes.)

► Exemple :

```
1 s1 = "Bonjour !"
2 s2 = s1.upper()
3 print(s1)
4 print(s2)
```



Bonjour !

BONJOUR !

- aucune méthode sur les `String` ne peut modifier un objet `String`,
- souvent, on doit passer par des affectations.

Fromatage des chaînes

Python a introduit une nouvelle manière riche de formater les chaînes de caractères.

```
1 s1="C'est Mr {} et il a {} ans !".format("X",20)
2 s2="C'est Mr {0} et il a {1} ans !".format("X",20)
3 s3="C'est Mr {0} et il a {1} ans !".format("X",20)
4 s4="C'est Mr {nom} et il a {age} ans !".format(nom="X",age=20)
5 s5="C'est Mr {0} et il a {age} ans !".format("X",age=20)
6
7 print(s1)
8 print(s2)
9 print(s3)
10 print(s4)
11 print(s5)
```



Fromatage des chaînes

Python a introduit une nouvelle manière riche de formater les chaînes de caractères.

```
1 s1="C'est Mr {} et il a {} ans !".format("X",20)
2 s2="C'est Mr {0} et il a {1} ans !".format("X",20)
3 s3="C'est Mr {0} et il a {1} ans !".format("X",20)
4 s4="C'est Mr {nom} et il a {age} ans !".format(nom="X",age=20)
5 s5="C'est Mr {0} et il a {age} ans !".format("X",age=20)
6
7 print(s1)
8 print(s2)
9 print(s3)
10 print(s4)
11 print(s5)
```



C'est Mr X et il a 20 ans !

Formatage des chaînes (nombres)

On a la possibilité de spécifier notamment :

- ▶ la largeur d'affichage, l'alignement, la position d'une variable
- ▶ l'affichage des nombres (base, précision, etc.)
- ▶ ...

Formatage des chaînes (nombres)

On a la possibilité de spécifier notamment :

- ▶ la largeur d'affichage, l'alignement, la position d'une variable
- ▶ l'affichage des nombres (base, précision, etc.)
- ▶ ...

```
1 print('{:.3}'.format(3.14159))
2 print('{:8}'.format(14159))
3 print('{1:<8} aa {0}'.format(12,14159))
```



Formatage des chaînes (nombres)

On a la possibilité de spécifier notamment :

- ▶ la largeur d'affichage, l'alignement, la position d'une variable
- ▶ l'affichage des nombres (base, précision, etc.)
- ▶ ...

```
1 print('{:.3}'.format(3.14159))  
2 print('{:8}'.format(14159))  
3 print('{1:<8} aa {0}'.format(12,14159))
```



```
3.14  
    14159  
14159  aa 12
```

Fromatage des chaînes (nombres)

```
1 w = 5
2 for n in range(0,12):
3     for base in ["d","b"]:
4         print("{0:{w}{b}}".format(n, b=base, w=w), end=" ")
5     print("")
```



Formatage des chaînes (nombres)

```
1 w = 5
2 for n in range(0,12):
3     for base in ["d","b"]:
4         print("{0:{w}{b}}".format(n, b=base, w=w), end=" ")
5     print("")
```



0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001

Que faut-il retenir ?

- ▶ Ce qui apparaît dans ces supports doit être appris.
- ▶ Les exemples vus en cours doivent être compris.
- ▶ Les méthodes, fonctions, paramètres optionnels, syntaxes étranges, etc. qui apparaissent dans ces exemples doivent être compris et appris.

Section 2

Lecture et écriture dans des fichiers

Lecture dans un fichier

- Un programme Python peut lire ou écrire dans un fichier.
pour enregistrer des résultats, charger des données, etc.

Lecture dans un fichier

► Un programme Python peut lire ou écrire dans un fichier.
pour enregistrer des résultats, charger des données, etc.

```
1 fichier = open("te.txt", "r")  
2 c = fichier.read()  
3 print(c)
```

Lecture dans un fichier

- Un programme Python peut lire ou écrire dans un fichier.
pour enregistrer des résultats, charger des données, etc.
-

```
1 fichier = open("te.txt", "r")  
2 c = fichier.read()  
3 print(c)
```

- open permet d'ouvrir le fichier,
- read permet de récupérer le contenu du fichier.
Il est retourné sous forme de chaîne de caractère (puis ici stocké dans c).

Paramètres de open, chemins.

```
1 fichier = open("te.txt","r")
2 c = fichier.read()
```

Paramètres de open, chemins.

```
1 fichier = open("te.txt", "r")
2 c = fichier.read()
```

- ▶ le premier paramètre est le chemin du fichier.
 - ▶ si on en met que le nom comme ici, cela signifie que le fichier est rangé dans le même répertoire que le code Python.
 - ▶ On peut aussi mettre un chemin absolu,
"/home/olivier/scripts/projets.txt"
 - ▶ ou un chemin relatif :
"data/thefile.txt"
- ▶ "r" signifie qu'on ouvre le fichier pour lire (read en anglais) son contenu.

Écriture dans un fichier

```
1 fichier = open("te.txt", "w")  
2 fichier.write("Bonjour")
```

Écriture dans un fichier

```
1 fichier = open("te.txt", "w")
2 fichier.write("Bonjour")
```

- ▶ Notez le "w" comme write.
- ▶ L'instruction write écrit dans le fichier l'information passée en paramètre.

Écriture dans un fichier

```
1 fichier = open("te.txt", "w")
2 fichier.write("Bonjour")
```

- ▶ Notez le "w" comme write.
- ▶ L'instruction write écrit dans le fichier l'information passée en paramètre.
- ▶ Avec "w" on écrit à la place de ce qui était éventuellement dans le fichier.
- ▶ On peut aussi utiliser "a" qui permet qu'on écrive à la suite dans le fichier.