Python - Chaînes de caractères

[Mise à jour le : 2/8/2021]

- **Sources**
  - **Documentation** sur Python.org : référence du langage, opérations usuelles sur les chaînes, fonctions natives (built-in)
- **Lectures connexes**
  - **Real Python**
    - Basic Data Types in Python
    - Strings and Character Data in Python
    - Python String Formatting Best Practices
    - How to Convert a Python String to int
    - How to Use sorted() and sort() in Python
- **Mots-clés** : collection, séquence, indexation, insertion, concaténation, formatage, accès, parcours, sélection.

> Les mots ci-dessous sont dits “réservés”. Ils ne peuvent pas être utilisés comme nom de variable. Les mots soulignés sont une nouveauté de Python 3. Les mots en **gras** sont utilisés dans cette page.

and	continue	finally	is	raise
as	def	**for**	lambda	return
assert	del	from	None	True
async	elif	global	nonlocal	try
await	else	if	not	**while**
break	except	import	or	with
class	False	**in**	pass	yield

- Fonctions natives **(built-in)**[1] utilisées dans les exemples : **print**(), **len**(), **ord**(), **chr**().

---

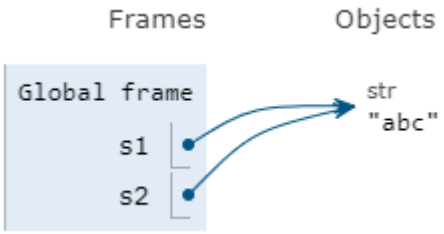
1. Introduction

> En programmation, le texte s'appelle chaîne de caractères. Pour créer une chaîne de caractères, il faut encadrer le texte de guillemets **“ “** ou d'apostrophes **' '**. Une chaîne de caractères est une **séquence**, ce qui signifie que c'est une **collection ordonnée** de valeurs. Le premier élément de la chaîne est **indexé** par **0**. Les chaînes de caractères sont des instances de la classe **str**.

> En Python une chaîne de caractères est **immuable** (ou **non mutable**) c'est-à-dire q'elle ne peut être modifiée après sa création.

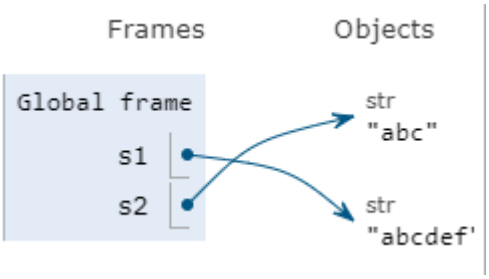
> Toutes les **méthodes** de manipulation des chaînes **renvoient** une **chaîne de caractères**.
>
> Il faut encadrer le texte de **guillemets** simples ' ' ou doubles **” “**.

Exemple



*.py

```
# deux variables pointent vers le même objet chaîne de caractères
s1 = 'abc'
s2 = s1
```



*.py

```
# on essaie de modifier l'objet
s1 += 'def' # une deuxième chaîne est créée pour s1,
            # s2 continue à pointer vers s1 initial
```

> La liste des méthodes peut être obtenue dans l'interpréteur python avec **dir(str)** et une aide sur une méthode avec **help(str**.*méthode*)**.

Exemple

```
>>> dir(str)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__dc
__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__',
'__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__s
ncode', 'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index
isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'lju
 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith',
>>> help(str.istitle)
Help on method_descriptor:

istitle(self, /)
    Return True if the string is a title-cased string, False otherwise.

    In a title-cased string, upper- and title-case characters may only
    follow uncased characters and lowercase characters only cased ones.
```

2. Mise en forme du texte

Les caractères d'une chaîne sont mis en forme avec les méthodes de la classe str. Par exemple en minuscules avec la méthode **lower**, en majuscules avec la méthode **upper**, etc.

Exemple

*.py

```
texte = "Le manuel de Python"
# Conversion en majuscules
print(texte.upper()) # Résultat : LE MANUEL DE PYTHON
# Conversion en minuscules
print(texte.lower()) # Résultat : le manuel de python
```

> Par défaut l'instruction **print** provoque un retour à la ligne après l'affichage. On peut changer ce comportement en fournissant une autre chaîne de caractère à accoler à l'affichage comme ci-dessus ou même rien.

Exemple

*.py

```
print("Ce texte s'affiche", end=" ") # Renvoie : Ce texte s'affiche sur
une seule ligne.
print("sur une seule ligne.", end=" ")
```

3. Formatage

- **Depuis Python 3.6**
  - Source : Bibliothèques Python

L'utilisation de **f-strings** permet d'insérer des expressions dans des chaines de caractères en utilisant une syntaxe minimale.

Exemple

*.py

```
f"Vous avez obtenu {monscore} points" # Résultat : Vous avez obtenu 1000 points
```

Pour **associer** des chaînes (**concaténation**) on utilise l'opérateur **+**.

Exemple

*.py

```
texte1 = "Hello"
texte2 = " World"
texte3 = texte1 + texte2 # Résultat texte3 = "Hello World"
```

- Pour **répéter** des chaînes, on utilise l'opérateur ***** .

Exemple

*.py

```
texte2 = " World"
texte3 = texte2*3 # Résultat texte3 = " World World World
```

- **Avant python 3.6**

Pour **insérer des valeurs** dans une chaîne on utilise **%s** à l'emplacement retenu, la méthode **format** ou des **virgules**.

Exemple

*.py

```
score = 1000

# Utilisation de %s (à éviter, car lourd par rapport aux autres
```

```
méthodes)
points = "Vous avez obtenu %s points"
print(points % score) # Résultat : Vous avez obtenu 1000 points

# Utilisation de la méthode format()
# Solution 1
print("Vous avez obtenu {0} points".format(score)) # Résultat : Vous
avez obtenu 1000 points
# Solution 2
print("Vous avez obtenu {score} points".format(score=monscore)) #
Résultat : Vous avez obtenu 1000 points

# Utilisation de virgules
print("Vous avez obtenu",monscore,"points") # Résultat : Vous avez
obtenu 1000 points
```

4. Longueur d'une chaîne, parcours et sélection (slice)

- **Ressource**
  - Python Indexing and Slicing: Complete Tutorial With Hands-On Exercises

> Une chaîne de caractères est une séquence constituée de chaînes de caractères constituées d'un seul caractère.

- **Longueur d'une chaîne**

> La longueur d'une chaîne (son **nombre d'éléments**) est déterminée avec la fonction built-in **len**().
>
> Exemple : **len**(“Le jour le plus long”) # renvoie 20

- **Accès** aux caractères d'une chaîne

> Pour accéder à un caractère dans une chaîne, on précise son indice entre crochets **[***indice***]**. **L'indice du premier caractère est 0**.

Exemple

*.py

```
texte = "Le manuel de Python 3"
```

```
texte[0]  # Résultat : L
texte[13] # Résultat : P
```

- **Parcours** des éléments d'une chaîne

Exemple

*.py

```
texte = "Le manuel de Python 3"

# Parcours avec une boucle while
i=0
while i<len(texte):
    print(texte[i])
    i+=1

# Parcours avec une boucle for
for i in texte:
    print(i)
```

- **Sélection** de chaîne (slice)

La sélection consiste à extraire une partie de la chaîne (*slicing*). Pour sélectionner une partie d'une chaîne, on précise la valeur du premier et du dernier indice entre crochet et éventuellement un pas.

> *Chaine***[***début* **:** *fin* **:** *pas***]**

> L'élément du **dernier indice** est **exclu**.

Exemple

*.py

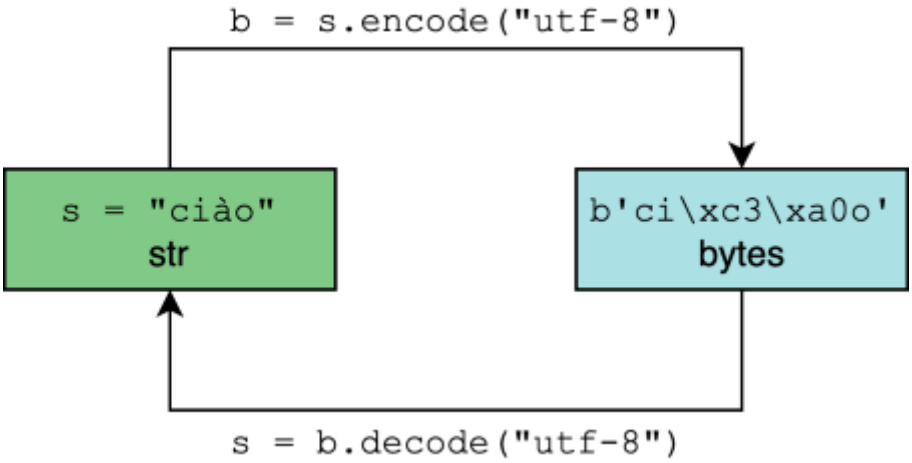
```
texte = "Le manuel de Python 3"
```

```
texte[:9]             # Résultat : 'Le manuel' (borne début omise)
texte[3:12]           # Résultat : 'manuel de'
texte[13:len(texte)] # Résultat : 'Python'
# ou
texte[13:]            # Résultat : 'Python' (borne fin omise)
texte[:]              # Résultat : 'Le manuel de Python 3' (shadow copy)
# Utilisation du pas
texte[:9:2]           # Résultat : 'L aul'
texte[9::2]           # Résultat : ' ePto '
# Indices négatifs
texte[3:-2]           #  Résultat : 'manuel de Python'
texte[-21:-14]        #  Résultat : 'Le manu'
texte[::-1]           #  Résultat : '3 nohtyP ed leunam eL' (inverse le
texte)
```

5. Encodage, décodage

- **Ressource** : Guide Unicode

"Le type de chaîne de caractères de Python utilise le standard Unicode pour représenter les caractères, ce qui permet aux programmes Python de travailler avec tous ces différents caractères possibles. UTF-8 est l'un des encodages les plus couramment utilisés et Python l'utilise souvent par défaut. "



- **Caractère → Unicode**

La fonction **ord**(*car*) envoie le nombre entier représentant le code Unicode du caractère représenté par la chaîne *car*donnée.

Exemple

*.py

```
ord('a') # donne 97
```

```
hex(ord('a')) # donne 0x61
```

- **Code Unicode → caractère**

La fonction **chr**(*i*) renvoie la chaîne représentant un caractère dont le code de caractère Unicode est le nombre entier *i*.

Exemple

*.py

```
chr(ox26) # donne '&'
```

6. Quelques méthodes de la classe str

Mise en forme	

format(arguments)	Place la liste des arguments, dans l'ordre, aux emplacements réservés par {0} dans la chaîne.
	*.py
	`nom = "Célestin"`
	`age = 4`
	`print("{} a {} ans".format(nom,age)) # Célestin à 4 ans`

Découpage - assemblage	

split()	Transforme une chaîne en une liste de sous-chaînes. Le **séparateur par défaut** est un **espace**, mais il est possible de lui donner en argument n'importe quel autre séparateur.
	*.py
	`s="Le petit chat de la voisine boit du lait"`
	`s.split() # Renvoie : ['Le', 'petit', 'chat', 'de', 'la', 'voisine', 'boit', 'du', 'lait']`

join(**liste**)	Effectue l'opération **inverse** de split. Commence par des **guillements** dans lesquels on place le **caractère de séparation**.
	*.py
	`l=['Le', 'petit', 'chat', 'de', 'la', 'voisine', 'boit', 'du', 'lait']`
	`" ".join(l) # Renvoie : 'Le petit chat de la voisine boit du lait'`

Remplacement	

replace(**old,new**)	Remplace le ou les caractères *old* par le ou les caractères *new*.
	*.py
	`s = "Le petit chat de la voisine boit du lait"`
	`s.replace("chat","chien") # Renvoie : 'Le petit chien de la voisine boit du lait'`

Nettoyage	

strip()	Supprime les espaces, tabulations et retours à la ligne éventuels en début et en fin de chaîne.
	*.py
	`" \tune chaîne avec des trucs qui dépassent \n".strip()`
	`# Renvoie : 'une chaîne avec des trucs qui dépassent'`

lstrip()	Enlève les espaces éventuels seulement en début de chaîne.

Mise en forme	
rstrip()	Enlève les espaces éventuels en fin de chaîne.
Recherche d'une sous-chaîne	

find(**x**)	Fait exactement la même chose que index() sauf que si elle ne trouve pas la sous-chaîne, elle renvoie -1.
	*.py
	`# L'exemple ci-dessous renvoie l'indice du début de la première occurrence`
	`"abcdefcdefghefghijk".find("def") # soit ici 3`
	`# rfind fait la même chose que find mais en partant de la fin`
	`"abcdefcdefghefghijk".rfind("fgh") # Renvoie 13`

index(**y**)	Renvoie l'indice de la première occurrence de la chaîne passée en argument. Lève une exception en cas d'absence.
count()	Renvoie le nombre d'occurrences d'une sous-chaîne. *.py `"abcdefcdefghefghijk".count("ef") # renvoie 3`
startswith()	Renvoie True si la chaîne commence par la sous-chaîne passée en argument. Sinon, elle renvoie False. *.py `"abcdefcdefghefghijk".startswith("abcd") # Renvoie True`
endswith()	Renvoie True si la chaîne se termine par la sous-chaîne passée en argument. Sinon, elle renvoie False. *.py `"abcdefcdefghefghijk".endswith("ghijk") # Renvoie True`
Changement de casse	
upper()	Convertit la chaîne de caractères en majuscules. *.py `"monty PYTHON".upper() # Renvoie : 'MONTY PYTHON'`
lower()	Convertit la chaîne de caractères en minuscules. *.py `"monty PYTHON".lower() # Renvoie : 'monty python'`
capitalize()	Convertit la première lettre d'un mot en majuscule. *.py `"monty PYTHON".capitalize() # Renvoie : 'Monty python'`
Ajout de caractères	
zfill(**n**)	Complète la chaîne par des zéros (à gauche) jusqu'à ce qu'elle contienne le nombre **n** de caractères passés en paramètres. *.py `bin(0)[2:].zfill(8) # Renvoie '00000000'`

Résumé

- Les chaînes sont des **objets**.
- Un objet est une **instance** de **classe**.
- Une classe possède des **méthodes**, accessibles à partir de l'objet grâce à **objet.methode(arguments)**.
- On peut accéder à un caractère dans la chaîne grâce à *chaine[indice]* ou à une partie de la chaîne grâce à *chaine[premier_indice:dernier_indice]*

Quiz

- Basic Data Types in Python
- Splitting, Concatenating, and Joining Strings in Python Quiz

Pour aller plus loin

- A Guide to the Newer Python String Format Techniques
- How to Convert a Python String to int
- Regular Expressions: Regexes in Python (Part 1) (Part 2)
- Les expressions régulières dans la documentation Python - Outil pythex
- Splitting, Concatenating, and Joining Strings in Python
- A Comprehensive Guide to Slicing in Python

1)

Fonctions toujours disponibles.