

Mise à Niveau

Sébastien Gagné, Université d'Orléans

Cours 03

Types itérables, Boucles for

Plan du cours 3

- Exemple des chaînes de caractères
- Les autres types itérables : *range*, *tuple*, *list*
- Opérateur [] : Accès à un élément, Extraction, Priorité et Associativité
- Divers : Taille, Appartenance, Concaténation

Exemple des chaînes de caractères

Une chaîne de caractères est une structure qui peut être parcourue caractère par caractère (itération).

Ce parcours utilise une boucle *for*, selon le format

```
for lettre in chaine :  
    <instruction_1>  
    <instruction_2>  
    ...
```

Le bloc d'instructions sera exécuté pour chaque valeur donnée à la variable nommée ici *lettre* (pour chaque lettre de la variable *chaine*).

Exemple des chaînes de caractères

```
1 # entrée d'un mot
2 mot = input('entrez un mot : ')
3 numLettre = 1
4
5 # parcours de la variable mot
6 for lettre in mot :
7     print('- lecture de la lettre', numLettre, ':', lettre)
8     numLettre = numLettre + 1
9
10 print('lecture terminée')
```

```
entrez un mot : Algo
- lecture de la lettre 1 : A
- lecture de la lettre 2 : l
- lecture de la lettre 3 : g
- lecture de la lettre 4 : o
lecture terminée
```

Exemple des chaînes de caractères

Voici un programme.

```
1  # entrée d'un mot
2  mot = input('entrez un mot : ')
3  separateur = input('entrez le séparateur : ')
4  mot2 = separateur
5
6  # parcours de la variable mot
7  for lettre in mot :
8      mot2 = mot2 + lettre + separateur
9
10 print('mot transformé : ', mot2)
```

Décrivez l'affichage final pour les entrées suivantes :

- mot : abc
- separateur : |

Exemple des chaînes de caractères

Ecrivez un programme Python qui demande une chaîne de caractères à l'utilisateur, puis qui crée et affiche une nouvelle chaîne construite ainsi, à partir de la chaîne donnée :

- chaque minuscule entre 'a' et 'z' est remplacée par 'm' ;
- chaque majuscule entre 'A' et 'Z' est remplacée par 'M' ;
- chaque caractère entre '0' et '9' est remplacé par '0' ;
- tout autre caractère est remplacé par '#'.

Exemple d'exécution :

```
entrez une chaîne : a5+L4ty=éP[  
chaîne transformée : m0#M0mm##M#
```

Les types itérables

On parle de *type itérable* pour évoquer un type dont les valeurs peuvent être parcourues au moyen d'une boucle *for*.

On a déjà vu les chaînes de caractères : type *str*.

Il y a aussi :

- les suites arithmétiques de nombres entiers : type *range* ;
- les suites de valeurs : type *tuple* ;
- les listes de valeurs : type *list*.

Le type *range*

```
1 deb = int(input("début : "))
2 fin = int(input("fin : "))
3 pas = int(input("pas : "))
4
5 chaine = ""
6
7 for nombre in range(deb, fin, pas) :
8     chaine = chaine + str(nombre) + "; "
9
10 print("contenu du range : ", chaine)
```

```
début : 4
fin : 79
pas : 7
4; 11; 18; 25; 32; 39; 46; 53; 60; 67; 74;
```

```
début : 9
fin : -1
pas : -1
9; 8; 7; 6; 5; 4; 3; 2; 1; 0;
```

Le type *range*

- *range*(5) génère la série 0 ; 1 ; 2 ; 3 ; 4
- *range*(-2, 4) génère la série -2 ; -1 ; 0 ; 1 ; 2 ; 3
- *range*(2, 33, 9) génère la série 2 ; 11 ; 20 ; 29 ;

Plus généralement

$$\textit{range}(d, f, p)$$

génère la séquence des entiers entre d (inclus) et f (exclus), avec un pas égal à p .

Boucle *for* et type *range*

Pour répéter n fois un bloc d'instructions, on utilise une boucle *for* combinée à un itérable de type *range*

Souvent, la variable qui sert de compteur de boucle (ici i) est utilisée dans le bloc d'instructions.

Dans ce cas, l'exécution de l'instruction peut donner un résultat différent à chaque tour de boucle.

```
>>> for i in range(4) :  
        print('Hello n°' + str(i+1))  
  
Hello n°1  
Hello n°2  
Hello n°3  
Hello n°4
```

Boucle *for* et type *range*

Parfois, la variable de boucle n'est pas utilisée dans le bloc d'instructions. L'exécution de l'instruction donne alors un résultat indépendant de cette variable.

```
>>> for i in range(4) :  
        print('Hello')
```

```
Hello  
Hello  
Hello  
Hello
```

Le nom de cette variable est alors sans importance. La variable de boucle peut être anonymisée.

```
for _ in range(4) :  
    print("hello")
```

Le type *tuple*

Les valeurs de type tuple sont des séquences **non modifiables** de diverses valeurs (elles-mêmes de types variables).

```
1 # un tuple de valeurs de type str
2 couleurs = ('rouge', 'vert', 'bleu')
3
4 # un tuple de valeurs de type float
5 point = (7.5, 3.2)
6
7 # un tuple de valeurs de type variés :
8 # un float, un bool, un str et même un tuple
9 tuple_bizarre = (2.3, True, 'Hello', (1, 3, 5))
```

Le type *list*

Les valeurs de type list sont des séquences **modifiables** de diverses valeurs (elles-mêmes de types variables).

```
1 # une list de valeurs de type str
2 couleurs = ['rouge', 'vert', 'bleu']
3
4 # une list de valeurs de type float
5 point = [7.5, 3.2]
6
7 # une list de valeurs de type variés :
8 # un float, un bool, un str, une list et un tuple
9 list_bizarre = [3.14, False, 'Bye', [1, 3], ('a', 0, True, 'toto')]
```

L'opérateur []

- L'opérateur [] permet d'accéder à un élément particulier de l'itérable.
- L'indice fourni précise l'élément.
- Pour des indices **positifs**, le premier indice est 0, le dernier est la taille de l'itérable - 1.

```
1 str1 = 'Bonjour'
2 range1 = range(1, 20, 3)
3 tuple1 = ('a', 'b', 'c', 'd')
4 list1 = [3.14, False, 'Bye', [1, 3], ('a', 0, True, 'toto')]
5
6 print(str1[0])
7 print(str1[3])
8 print(range1[4])
9 print(tuple1[2])
10 print(list1[4])
```

L'opérateur []

- Les indices fournis à l'opérateur [] peuvent aussi être **négatifs** !
- L'indice -1 désigne le "dernier" élément.
- Dans ce cas, le "premier" élément est celui d'indice -taille.

```
1 str1 = 'Bonjour'
2 range1 = range(1, 20, 3)
3 tuple1 = ('a', 'b', 'c', 'd')
4 list1 = [3.14, False, 'Bye', [1, 3], ('a', 0, True, 'toto')]
5
6 print(str1[-7])
7 print(str1[-1])
8 print(range1[-5])
9 print(tuple1[-3])
10 print(list1[-2])
```

L'opérateur []

Les indices d'accès autorisés sont donc

- De 0 à taille - 1 pour le sens "naturel", de gauche à droite ;
- De -1 à -taille pour le sens inverse.

En dehors de ces deux plages, se produit une erreur.

```
>>> ch1 = 'Bonjour'
>>> ch1[7]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range

>>> ch1[-8]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

Extraction avec []

Extraction d'un "sous-itérable" de même type :

$\langle iterable \rangle [\langle debut \rangle : \langle fin \rangle]$

- l'élément d'indice *debut* est inclus ;
- l'élément d'indice *fin* est exclu.

L'extraction se fait toujours de gauche à droite.

```
1 str1 = 'Bonjour'
2 list1 = [3.14, False, 'Bye', [1, 3], ('a', 0, True, 'toto')]
3
4 print(str1[1 : 4])
5 print(str1[-5 : 5])
6 print(str1[-4 : -2])
7
8 print(list1[1 : 3])
9 print(list1[-3 : 4])
10 print(list1[-4 : -1])
```

Extraction avec []

Si une borne est manquante, cela revient à "tout jusqu'à" ou à "tout à partir de".

```
1 str1 = 'Algorithme'
2
3 print(str1[3 :])    # à partir de l'indice 3 (inclus)
4 print(str1[: 5])    # jusqu'à l'indice 5 (exclus)
5
6 print(str1[-4 :])   # à partir de l'indice -4 (inclus)
7 print(str1[: -6])   # jusqu'à l'indice -6 (exclus)
```

Extraction avec []

Contrairement à l'accès à un élément particulier, qui ne tolère pas les dépassements de bornes, une extraction peut être faite avec une borne hors-limite.

```
1 str1 = 'Algorithme'
2
3 print(str1[4 : 17])
4 print(str1[-11 : 6])
5 print(str1[-11 : 17])
```

Priorité et associativité avec []

- Priorité : comme l'appel de fonction, supérieure à tout le reste ;

'Sa' + 'Hello'[3] + 'Bonjour'[-2] + 't'

'Sa'	+	<u>'Hello'[3]</u>	+	<u>'Bonjour'[-2]</u>	+	't'
		'l'		'u'		
		'Sal'				
				'Salu'		
						'Salut'

Priorité et associativité avec []

- Associativité : de gauche à droite

```
1 list1 = [3.14, False, 'Bye', [1,3], ('a', [0,2,7,8,4,3], True, 0)]  
2  
3 print(list1[4])  
4 print(list1[4][1])  
5 print(list1[4][1][2:5])
```

('a', [0, 2, 7, 8, 4, 3], True, 0)

[0, 2, 7, 8, 4, 3]

[7, 8, 4]

Taille d'un itérable

La taille (*len*) est le nombre d'éléments de l'itérable.

```
1 str1 = 'Bonjour'
2 range1 = range(1, 28, 3)
3 tuple1 = ('a', 'b', 'c', 'd')
4 list1 = [3.14, False, 'Bye', [1, 3], ('a', 0, True, 'toto')]
5
6 ma_liste = [str1, range1, tuple1, list1]
7
8 for iterable in ma_liste :
9     print(len(iterable))    # len donne la taille de l'itérable
```

```
1 str1 = 'Algo'
2
3 for i in range(len(str1)) : # 0; 1; 2; 3;
4     print('caractère d\'indice', i, ':', str1[i])
```

Test d'appartenance - *in*

Utilisé seul, Le mot clé *in* permet de dire si oui ou non l'élément est présent dans l'itérable.

<element> in <iterable>

Le résultat est donc un booléen.

```
1 str1 = 'Bonjour'
2 tuple1 = (1, 2, 7, 4)
3
4 print('B' in str1)      # dit si 'B' est dans str1
5 print('j' not in str1)  # dit si 'w' n'est pas dans str1
6
7 print(5 in tuple1)      # dit si 2 est dans tuple1
8
9 print('onj' in str1)    # dit si 'onj' est une sous-chaîne de str1
10                        # attention, ça ne fonctionne que pour les str !
```

Concaténation d'itérables : +

On peut concaténer, grâce à l'opérateur + et **sans mélanger les types** :

- deux valeurs *str*,
- deux valeurs *tuple*,
- deux valeurs *list*.

```
>>> 'Hello' + " world !"
'Hello world !'
>>> (1, 2, 3) + (8, 7)
(1, 2, 3, 8, 7)
>>> ['a', 'bc', 'd'] + ['ef', 'g']
['a', 'bc', 'd', 'ef', 'g']
```

Les valeurs de type *range* ne sont pas concaténables.