**Fonctions Lambda, Map, Reduce et Filter**

**La fonction map()**

La fonction map() utilise une fonction et une liste (ou tout autre itérable) et exécute la fonction sur chaque objet de la liste. Elle renvoie ensuite la séquence contenant tous les résultats.

Par exemple:

def upper\_string(s):

return s.upper()

fruit = ["Apple", "Banana", "Pear", "Apricot", "Orange"]

map\_object = map(upper\_string, fruit)

print(list(map\_object))

>> ['APPLE', 'BANANA', 'PEAR', 'APRICOT', 'ORANGE']

L'objet map\_object renvoyé contiendra tous les fruits en majuscules en raison de chaque fruit passé à la fonction upper\_string.

**La fonction filter()**

Similaire à map(), filter() prend un objet fonction et un itérable et crée une nouvelle liste.

Comme son nom l'indique, filter() crée une nouvelle liste contenant uniquement les éléments satisfaisant une condition particulière. Cette condition correspond à la fonction passée en argument. Si elle renvoie True, l'élément est ajouté à la nouvelle liste ; sinon, il ne l'est pas.  
Par exemple, l'extrait suivant ne renvoie que les chaînes commençant par A.

def starts\_with\_A(s):

return s[0] == "A"

fruit = ["Apple", "Banana", "Pear", "Apricot", "Orange"]

filtered\_object = filter(starts\_with\_A, fruit)

print(list(filtered\_object))

>> ['Apple', 'Apricot']

**La fonction functools.reduce()**

reduce() fonctionne différemment de map() et filter(). Elle ne renvoie pas une nouvelle liste basée sur la fonction et l'itérable passés. Elle renvoie une valeur unique.

Par exemple, la fonction somme est une opération réductrice, elle prend une liste comme argument et renvoie un seul nombre.

Une fonction passée à reduce doit recevoir deux arguments qui représenteront le premier et le deuxième élément de l'itérable, puis le deuxième et le troisième, puis le troisième et le quatrième, etc…

Par exemple, un extrait qui recrée la fonction somme :

from functools import reduce

def sum\_numbers(first, second):

return first+second

my\_list = [1, 3, 5, 7]

reduced\_list = reduce(sum\_numbers, my\_list)

print(reduced\_list)

>> 16

**Fonctions lambda**

Les fonctions Lambda sont des « fonctions sur une seule ligne », elles reçoivent uniquement des arguments et renvoient une valeur, et elles n'ont pas besoin d'être stockées dans des variables (mais elles le peuvent !).

Voici la syntaxe d'une fonction lambda :

lambda arg1, arg2 : valeur\_retournée

Par exemple, une fonction qui renvoie une chaîne en majuscules :

lambda s: s.upper()

Vous pouvez également stocker cette fonction dans une variable si vous souhaitez la réutiliser :

my\_function = lambda s: s.upper()

# This is the same as doing:

def my\_function(s):

return s.upper()

Ces fonctions peuvent s'avérer utiles lors de l'utilisation de cartes, de filtres et de réductions, car elles permettent une création rapide. Par exemple, recréons les extraits précédents à l'aide de fonctions lambda.

L'exemple de carte :

fruit = ["Apple", "Banana", "Pear", "Apricot", "Orange"]

map\_object = map(lambda s: s.upper(), fruit)

print(list(map\_object))

L'exemple de filtre :

fruit = ["Apple", "Banana", "Pear", "Apricot", "Orange"]

filtered\_object = filter(lambda s: s[0] == "A", fruit)

print(list(filtered\_object))

L'exemple de réduction :

from functools import reduce

my\_list = [1, 3, 5, 7]

reduced\_list = reduce(lambda first, second: first+second, my\_list)

**🚀 Défi**

personnes = ["Rick", "Morty", "Beth", "Jerry", "Snowball"]

À l'aide de la carte et du filtre, essayez de dire bonjour à tous ceux dont le nom est inférieur ou égal à 4 lettres

**Conclusion**

Cette leçon vous a présenté les puissants concepts de programmation fonctionnelle en Python, facilitant la création d'un code concis et lisible. En maîtrisant lambda, map, reduce et filter, vous améliorerez votre capacité à écrire du code Python de haute qualité, exploitant efficacement les capacités de Python pour la programmation fonctionnelle.

© 2025 **École supérieure de technologie et de commerce de Paris** . Tous droits réservés.