Compréhension de la liste du jour 13

Compréhension de la liste

La compréhension de la liste dans Python est une manière compacte de créer une liste à partir d'une séquence. C'est un moyen court de créer une nouvelle liste. La compréhension de la liste est considérablement plus rapide que le traitement d'une liste à l'aide de la boucle *for*.

```
# syntaxe
[je pour je dans itérable si l'expression                    ]]
```

Exemple: 1

Par exemple, si vous souhaitez modifier une chaîne en une liste de caractères. Vous pouv ez utiliser quelques méthodes. Voyons certains d'entre eux:

```
# Language unique = 'Python' LST = List (Langue) # Modification de la chaîne e
n Liste PRINT (Type (LST)) # List Print (LST) # ['P', 'Y', 'T', 'H', 'O', 'N']




# Second Way: List Comprehension LST = [i For i in Language] print (type
(lst)) # list print (lst) # ['p', 'y', 't', 'h', 'o', 'n']
```

Exemple: 2

Par exemple, si vous souhaitez générer une liste de nombres

```
# Génération de nombres Nombres = [i pour i dans la plage (11)] # pour générer des nombres d
e 0 à 10 imprimer (nombres) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]




# Il est possible de faire des opérations mathématiques pendant les carrés d'itération = [i * i
pour i dans la plage (11)] imprimer (carrés) # [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]



# Il est également possible de faire une liste de numéros de tuples = [(i, i * i) pour i dans la pl
age (11)] imprimer (nombres) # [(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]]
```

Exemple: 2

La compréhension de la liste peut être combinée avec l'expression si

```
# Génération des nombres pair pair_numbers = [i pour i dans la plage (21) Si i% 2 == 0] # p
our générer une liste de nombres uniformes dans la plage 0 à 21 imprimer (même_numbres)
# [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]


# Génération de nombres impairs odd_numbers = [i pour i dans la plage (21) Si i% 2! = 0] # po
ur générer des nombres impairs dans la plage de 0 à 21 imprimer (odd_numbers) # [1, 3, 5, 7, 9
, 11, 13, 15, 17, 19] # Numbers de filtre: - - Filtrons-vous, les nombres de la liste inférieurs aux
numéros = [-8, -3, -1, 0, 0, 0, 0, V5} [-8, - - -1 1, 3, 4, 5, 7, 6, 8, 10] positif_even_numbers = [i
pour i en nombres si je% 2 == 0 et i > 0] imprimer (positif_even_numbers) # [2, 4, 6, 8, 10, 12,
14, 16, 18, 20] # a aplatissant un arrivée dimensionnelle 2, 3], [4, 5, 6], [7, 8, 9]] aplatis_list = [
numéro pour la ligne dans list_of_lists pour le nombre en ligne] print (aplated_list) # [1, 2, 3, 4,
5, 6, 7, 8, 9]
```

Fonction lambda

La fonction lambda est une petite fonction anonyme sans nom. Cela peut prendre n'importe q
uel nombre d'arguments, mais ne peut avoir qu'une seule expression. La fonction Lambda est
similaire aux fonctions anonymes en JavaScript. Nous en avons besoin lorsque nous voulons
écrire une fonction anonyme dans une autre fonction.

Création d'une fonction lambda

Pour créer une fonction Lambda, nous utilisons *lambda* mot-clé suivi d'un (s) paramètre (s), s
uivi d'une expression. Voir la syntaxe et l'exemple ci-dessous. La fonction lambda n'utilise pa
s le retour mais il renvoie explicitement l'expression.

```
# syntaxe x = lambda param1, param2, param3: param1 + param2 + param2 print (x (arg1,
arg2, arg3))
```

Exemple:

```
# Nommé fonction def add_two_nu
ms (a, b): renvoie un + b


print (add_two_nums (2, 3)) # 5 # permet de modifier la fonction ci-dessus en un
e fonction lambda add_two_nums = lambda a, b: a + b print (add_two_nums (2,3
)) # 5


# Fonction Lambda auto-invoquant (Lambda A, B: A + B) (2,3) # 5 - Besoin de le résu
mer dans Print () pour voir le résultat dans la console


carré = lambda x: x ** 2 print (carré (3))
# 9 cube = lambda x: x ** 3 print (cube (
3)) # 27


# Variables multiples multiples_variable = lambda a, b, c: a ** 2 - 3 * b + 4 * c print (mu
ltiple_variable (5, 5, 3)) # 22
```

Fonction lambda dans une autre fonction

Utilisation d'une fonction lambda à l'intérieur d'une autre Functi on

```
Def Power (x): retourne lambda n: x ** n

Cube = Power (2) (3) # Fonction Power a maintenant besoin de 2 arguments pour s'exécuter,
dans des supports arrondis séparés imprimer (Cube) # 8 Two_Power_Of_Five = Power (2) (5
) Print (Two_Power_Of_Five) # 32
```

Continuez votre bon travail. Gardez l'élan, le ciel est la limite! Vous venez de terminer les défis du jour 13 et vous êtes à 13 pas de la grandeur à la grandeur. Faites maintenant quelque s exercices pour votre cerveau et vos muscles.

Exercices: Jour 13

1. Filtre uniquement négatif et zéro dans la liste à l'aide de la compréhension de la liste

```
Nombres = [-4, -3, -2, -1, 0, 2, 4, 6]
```

2. Aplatir la liste suivante des listes de listes à une liste unidimensionnelle:

```
list_of_lists = [[[1, 2, 3]], [[4, 5, 6]], [[7, 8, 9]]]

Sortie [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

3. Utilisation de la compréhension de la liste Créez la liste suivante des tuples:

```
[(0, 1, 0, 0, 0, 0, 0), (1, 1, 1, 1, 1, 1, 1), (2, 1, 2, 4, 8, 16, 32
), (3, 1, 3, 9, 27, 81, 243), (4, 1, 4, 16, 64, 256, 1024), (5, 5
, 25, 125, 625, 3125), (6, 3, 5, 255, 625, 3125), (6, 3, 5, 25
5, 625, 3125). 216, 1296, 7776), (7, 1, 7, 49, 343, 2401, 16
807), (8, 1, 8, 64, 512, 4096, 32768), (9, 1, 9, 81, 729, 656
1, 59049), (10, 1, 10, 100, 1000, 10000, 10000)]
```

4. Aplatir la liste suivante à une nouvelle liste:

```
pays = [[('Finlande', 'Helsinki')], [('Suède',
«Stockholm»)], [(«Norvège», «Oslo»)] Sortie: [[«Finlande», «Fin», «Helsi
nki»], [«Suède», «Swe», «Stockholm»], [«Norvège», «Nor», «Oslo»]]]
```

5. Changer la liste suivante en une liste de dictionnaires:

```
pays = [[('Finlande', 'Helsinki')], [('Suède',
«Stockholm»)], [(«Norvège», «Oslo»)]]
sortir:
[{'country': 'Finlande', 'City': 'Helsinki'},
{'Country': 'Suède', 'City': 'Stockholm'}, {'country': 'Norway', 'City
': 'Oslo'}]
```

6. Modifiez la liste des listes suivantes en une liste de chaînes concaténées:

```
Noms = [[('Asabeneh', 'encoreayeh')], [('David', 'Smith')], [('Donald', 'Trump')], [('Bill', '
Gates')]
```

7. Écrivez une fonction lambda qui peut résoudre une pente ou une interception Y des fonctions linéaires.

Félicitations!