Fonctions Lambda en Python

1. Objectifs

- Comment, quand utiliser et quand ne pas utiliser les fonctions Lambda

2. Présentation

- En Python, le mot-clé `lambda` est utilisé pour déclarer une fonction anonyme, raison pour laquelle ces fonctions sont appelées "**fonctions lambda**". Une fonction anonyme se réfère à une fonction déclarée sans nom.
- Une **fonction lambda** est comme n'importe quelle fonction python normale, sauf qu'elle n'a pas de nom lors de sa définition et qu'elle est contenue dans une ligne de code.
- Les fonctions **lambda en Python** sont également appelées **fonctions anonymes** (fonctions sans nom). Ces fonctions remplacent les fonctions traditionnelles définies par l'utilisateur comportant deux lignes ou plus par une simple fonction à une ligne .



- Tout comme vous définissez une fonction définie par l'utilisateur à l'aide du mot-clé '`def`', vous devez définir une fonction lambda à l'aide du mot-clé '`lambda`'. Une fonction lambda peut avoir '**n**' **nombre d'arguments** mais une **seule expression**.

- **Exemple 01**

  - Imaginons qu'on voudrait créer une fonction pour calculer la **somme de deux valeurs** en python notre code ressemblera à ça.

apcpedagogie.com

```
def sommeClassique( a , b ):
  return a + b
```

- Avec les lambda on aura le code suivant

```
sommeLambda = lambda a,b : a+b
```

3. **Exemple 02**

   - écrire une fonction lambda, qui représente l'équation suivante : **2x-3y+z**

```
f = lambda x, y, z : 2*x - 3*y +z
print(f(5,3,0))
```

4. **Exemple 03**

   - Dans cet exemple, nous voulons calculer les indemnités spécifiques accordées aux agents d'une entreprise, suivant le chiffre d'affaire réalisé, le système de paie attribue à chaque semaine une somme bien définie. Donc, pour calculer l'indemnité d'un mois nous devons écrire une fonction qui fait la somme des montants de toute la semaine moins 19 pour 100 de tva.

```
indemnite = lambda sem1 ,sem2,sem3,sem4:(sem1+ sem2
             +sem3 +sem4) -(sem1+ sem2
             + sem3 + sem4)*0.19
print(indemnite (120, 144, 500,300))
```

5. Utilisation de la fonction lambda en Python

- **Syntaxe des fonctions Lambda en Python**
  - **`argument(s) lambda : expression`**

apcpedagogie.com



- Les fonctions **lambda en Python** sont utilisées pour créer de petites fonctions anonymes d'une ligne sans trop d'effort de codage. L'utilisation réelle des fonctions lambda ne peut être connue que lorsqu'elle est utilisée dans le cadre d'une autre fonction.
- **Voir la vidéo**

- **Utilisation de la fonction `lambda()` avec `map()`**



- *Jusqu'à présent, vous avez utilisé des **fonctions lambda** pour écrire des fonctions simples et courtes ainsi que pour redéfinir des fonctions avec des fonctionnalités simples. Cependant, le meilleur cas d'utilisation des fonctions lambda est lorsque vous souhaitez que ces fonctionnalités simples soient intégrées de manière anonyme dans des expressions plus larges.*
- La fonction `map()` applique une fonction donnée à chaque élément d'un itérable (liste, tuple, etc.) et renvoie une liste des résultats.
- La fonction `map()` exécute une fonction `lambda` sur un "**itérable**"
- La syntaxe de map() est: `map(function, iterable, ...)`
  - **Les paramètres**
    - **fonction** - map() passe chaque élément du itérable à cette fonction.
    - **itérable** - itérable qui doit être mappé

Vous pouvez passer plus d'un itérableà la map()fonction.

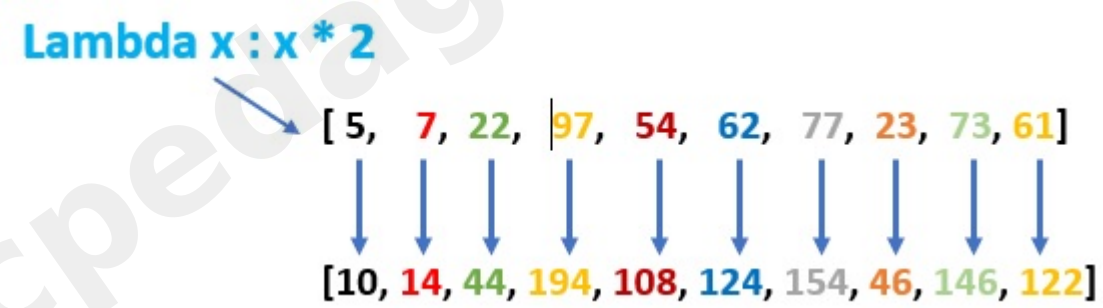
- La fonction `map()` en Python prend **une fonction** et une **liste** comme argument. La fonction est appelée avec une fonction **lambda** et une liste et une nouvelle liste est renvoyée qui contient tous les éléments modifiés lambda renvoyés par cette fonction pour chaque élément.
- La syntaxe de `map()` avec une fonction `lambda` est la suivante :
- `map(lambda item: item[] expression, iterable)`

- **Exemple**

```
liste = [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]

listeDouble = list(map(lambda x: x*2, liste))
print(listeDouble)
#Production:
[10, 14, 44, 194, 108, 124, 154, 46, 146, 122]
```

- Avec une liste comme celle ci-dessus, nous pouvons mettre en œuvre une fonction **lambda** avec une expression que nous voulons appliquer à chaque élément de notre liste



- **Application**

- Soit la liste nums suivante: **[2, 4, 6, 8, 10]**
- Utiliser la fonction **lambda()** avec la fonction **map()**, pour mettre tous les éléments à la puissance 2

```
resultat = list(map(lambda x: x ** 2, nums))
print(resultat)
```

6. **Voir la vidéo**

7. **Utilisation de la fonction lambda() avec `filter()`**

   - Comme son nom l'indique, `filter` est une autre fonction intégrée qui filtre une séquence ou tout objet itératif.
   - En d'autres termes, étant donné tout objet itératif (comme une liste), la fonction de `filter()` filtre certains des éléments tout en gardant certains selon des critères.
   - Ce critère est défini par l'appelant de `filter` en passant une fonction comme argument. Cette fonction est appliquée à chaque élément de l'itératif. Si la valeur de retour est **True**, l'élément est conservé. Sinon, l'élément est ignoré.

   - **Exemples:**

     - Considérez l'exemple ci-dessous où la fonction `filter()` filtre et imprime les nombres pairs de l'itérateur donné à l'aide d'une fonction définie par l'utilisateur.

```
def nbrPaire(x):
    if(x % 2 == 0):
        return x

listeNombres = [12, 5, 18, 22, 97, 44, 45, 88]
resultat = filter(nbrPaire, listeNombres)
print(list(resultat))

#Output:
#[12, 18, 22, 44]
```

   - Vous trouverez ci-dessous le même code ci-dessus implémenté à l'aide de la fonction lambda.

```
listeNombres = [12, 5, 18, 22, 97, 44, 45, 88]
resultat = list(filter( lambda x: (x % 2 == 0 ) , listeNombres))
print(resultat)
```

8. Soit la liste des notes suivante: **notes = [45 , 70 , 94, 28, 75, 76, 51, 35, 50, 48, 78, 88, 90]**
9. Maintenant, vous aimerez filtrer les scores inférieurs à la moyenne. Écrivons le faire de la manière normale. Ensuite, nous le faisons également en utilisant la fonction de filtre avec la fonction lambda. Ici, nous supposons que pour une liste de scores, 50 est le score moyen.

apcpedagogie.com

```
def infMoyenne(x):
    if(x <= 50):
        return x

notes = [45 , 70 , 94, 28, 75, 76, 51, 35, 50, 48, 78, 88, 90]
resultat = filter(infMoyenne, notes)
print(list(resultat))
```

10. Vous trouverez ci-dessous le même code ci-dessus implémenté à l'aide de la fonction lambda.

```
notes = [45 , 70 , 94, 28, 75, 76, 51, 35, 50, 48, 78, 88, 90]
resultat = list(filter( lambda x: (x <= 50 ) , notes))
print(resultat)
```

11. ## Avantages & inconvénients de la fonction lambda en Python

    - **Avantages**
        - Bon pour les opérations logiques simples qui sont faciles à comprendre. Cela rend également le code plus lisible.
        - Bon quand vous voulez une fonction que vous n'utiliserez qu'une seule fois.
    - **Inconvénients**
        - Ils ne peuvent exécuter qu'une seule expression. Il n'est pas possible d'avoir plusieurs opérations indépendantes dans une seule fonction lambda.
        - Mauvais pour les opérations qui s'étendraient sur plus d'une ligne dans une def fonction normale (par exemple, les opérations conditionnelles imbriquées). Si vous avez besoin d'une minute ou deux pour comprendre le code, utilisez plutôt une fonction nommée.
        - Mauvais car vous ne pouvez pas écrire une chaîne de documentation pour expliquer toutes les entrées, opérations et sorties comme vous le feriez dans une fonction `def` normale .

Sommaire du cours Python

apcpedagogie.com

Source : https://apcpedagogie.com/fonctions-lambda-en-python · téléchargé le 07/11/2026 · © APC Pédagogie · apcpedagogie.com

apcpedagogie.com