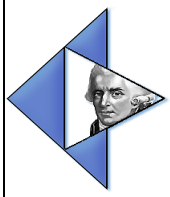

Université de Corse - Pasquale PAOLI

***Diplôme :* Licence SPI 3ème année** | 2023-2024

UE : Ateliers de programmation

Atelier 6 : Listes en compréhension et lambda expressions

Enseignants : Paul-Antoine BISGAMBIGLIA, Marie-Laure NIVET, Evelyne VITTORI

Partie 1 - Listes en compréhension

Une liste définie en compréhension caractérise un sous-ensemble d'une liste lst contenant tous les éléments de lst qui vérifient une condition C :

[x for x in lst if C(x)]

Syntaxe : [expression for élément in itérable if condition]

Exemples

```
lst=list(range(0,31))
lst1 = [x for x in lst if x%3==0] # liste des éléments de lst
divisibles par 3
lst2= [x for x in lst if 10<= x <=14] # liste des éléments de
lst compris entre 10 et 14
mots = ["chat", "chien", "éléphant", "crocodile"]
longueurs = [len(mot) for mot in mots]
# Résultat: [4, 5, 8, 9]
```

Questions

Définissez les fonctions suivantes en utilisant la définition de listes en compréhension et définissez progressivement une procédure de test :

1) fonction **listeMultiples(binf,bsup,nb)** qui admet en paramètres 3 entiers et retourne la liste des entiers multiples de nb compris entre les nombres binf et bsup.
2) fonction **ajouter(lst,nb)** qui admet en paramètres une liste lst d'entiers et un entier nb et retourne une nouvelle liste en ajoutant le nombre nb à chaque élément de la liste lst.
3) fonction **ajouterSiSup(lst,val,nb)** qui admet en paramètres une liste lst d'entiers et deux entiers val et nb, et retourne une nouvelle liste en ajoutant le nombre nb à chaque élément de la liste lst supérieur ou égal au nombre val.
4) fonction **bissextiles(adeb,afin)** qui admet en paramètres deux entiers et retourne la liste des années bissextiles comprises entre l'année adeb et l'année afin.
   Principe : une année bissextile est divisible par 4 et non par 100 ou par 400.
5) fonction **premiers(binf,bsup)** qui admet en paramètres 2 entiers et retourne la liste des nombres premiers strictement compris entre les nombres binf et bsup.

Principe :

- créer une liste lst d'entiers entre binf+1 et bsup

- pour chaque entier n compris entre 2 et 9
construire une liste lst en compréhension à partir de la liste existante lst
en ne conservant que les nombres inférieurs ou égaux à n
ou non multiples de n

Partie 2 – Lambda Expressions en python

Les expressions lambda en Python fournissent une façon d'écrire des fonctions de manière concise pour des utilisations simples. Une fonction lambda peut avoir n'importe quel nombre d'arguments, mais elle ne peut avoir qu'une seule expression. L'expression est évaluée puis renvoyée.

Syntaxe

```
lambda arguments: expression
```

Exemples

- **Fonction lambda simple** : Une lambda qui prend un nombre et renvoie son carré.

```
carre = lambda x: x**2
print(carre(5))  # Résultat: 25
```

- **Utilisation avec `sorted`** : Si vous avez une liste de tuples où chaque tuple a deux éléments, et que vous voulez trier cette liste en fonction du second élément de chaque tuple, vous pouvez utiliser une lambda.

```
liste_tuples = [(1, 20), (3, 5), (9, 16)]
sorted_list = sorted(liste_tuples, key=lambda x: x[1])
print(sorted_list)  # Résultat: [(3, 5), (9, 16), (1, 20)]
```

- **Utilisation avec `filter`** : Pour filtrer une liste de nombres et ne garder que les nombres pairs.

```
numeros = [1, 2, 3, 4, 5, 6]
pairs = list(filter(lambda x: x % 2 == 0, numeros))
print(pairs)  # Résultat: [2, 4, 6]
```

- **Lambda avec plusieurs arguments** : Une lambda qui prend deux arguments et les additionne.

```
somme = lambda x, y: x + y
print(somme(5, 3))  # Résultat: 8
```

Questions

1) **Conversion de Température** : Écrivez une expression lambda qui convertit les températures de Celsius à Fahrenheit. Utilisez-la pour convertir 25°C.
2) **Manipulation de chaînes** : Écrivez une expression lambda qui prend une chaîne en entrée et retourne la même chaîne avec la première lettre en majuscule et le reste en minuscules. Testez-le avec le mot "pYTHON".
3) **Filtrage d'une liste** : Étant donnée la liste `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`, utilisez `filter` avec une expression lambda pour ne garder que les nombres impairs.

Note : filter `filter(f, iterable)f est une fonction booléenne, filter retourne les éléments de l'itérable qui vérifie f`

Partie 3 – Exercice : Analyse financière des transactions bancaires

Vous êtes un analyste financier chez BigBank Inc. On vous fournit des données de transactions de plusieurs clients sur une année entière. Chaque transaction est décrite par un dictionnaire contenant les clés suivantes :

- `client_id` : Identifiant unique du client.
- `date` : Date de la transaction au format 'JJ/MM/AAAA'.
- `type` : Type de transaction - "dépôt" ou "retrait".
- `montant` : Montant de la transaction.
- `devise` : Devise de la transaction - 'EUR', 'USD', etc.

Questions

1) **Écrire une fonction `filtrer_par_date(transactions, date_debut, date_fin)`:**
   - **Paramètres** : Liste de transactions, date de début et date de fin.
   - **Résultat** : Liste des transactions qui ont eu lieu entre la date de début et la date de fin (inclus).
   - **Contrainte** : Utiliser une liste en compréhension.
2) **Écrire une fonction `solde_client(transactions, client_id)`:**
   - **Paramètres** : Liste de transactions, identifiant du client.
   - **Résultat** : Solde net du client (somme des dépôts moins somme des retraits).
   - **Contrainte** : Utiliser une liste en compréhension pour chaque type de transaction (dépôt et retrait).
3) **Écrire une fonction `transactions_max(transactions, n)`:**
   - **Paramètres** : Liste de transactions, un nombre entier n.
   - **Résultat** : Les n transactions ayant le plus grand montant.
   - **Contrainte** : Utiliser une liste en compréhension.
4) **Écrire une fonction `clients_sans_retraits(transactions)`:**
   - **Paramètres** : Liste de transactions.
   - **Résultat** : Liste des clients (identifiants) qui n'ont effectué aucun retrait.
   - **Contrainte** : Utiliser `filter` avec une expression lambda.
5) **Écrire une fonction `convertisseur(transactions, taux_conversion)`:**
   - **Paramètres** : Liste de transactions, dictionnaire des taux de conversion par devise (par exemple : {'USD': 0.85, 'GBP': 1.1} où les valeurs sont les taux de conversion par rapport à l'euro).
   - **Résultat** : Nouvelle liste de transactions où tous les montants sont convertis en euros.
   - **Contrainte** : Utiliser une expression lambda pour effectuer la conversion.

6) **Écrire une fonction `retraits_supérieurs_a(transactions, montant)`:**
   - **Paramètres** : Liste de transactions, un montant.
   - **Résultat** : Liste des transactions qui sont des retraits d'un montant supérieur ou égal au montant donné.
   - **Contrainte** : Utiliser `filter` avec une expression lambda.
7) **Test**

Testez vos fonctions en exécutant le script suivant :

```
transactions = [
{'client_id': 1, 'date': '01/01/2022', 'type': 'dépôt', 'montant': 1000,
'devise': 'EUR'},
{'client_id': 2, 'date': '03/01/2022', 'type': 'retrait', 'montant': 200,
'devise': 'USD'},
{'client_id': 1, 'date': '04/01/2022', 'type': 'dépôt', 'montant': 500,
'devise': 'EUR'},
{'client_id': 2, 'date': '05/01/2022', 'type': 'dépôt', 'montant': 300,
'devise': 'USD'},
{'client_id': 3, 'date': '05/01/2022', 'type': 'dépôt', 'montant': 700,
'devise': 'EUR'}
]

# Taux de conversion pour la devise par rapport à l'EUR
taux_conversion = {'USD': 0.85, 'GBP': 1.1}

# Tests
# 1. filtrer_par_date
print("Transactions entre le '02/01/2022' et le '05/01/2022':")
print("-----------------------------------------------------")
print(filtrer_par_date(transactions, '02/01/2022', '05/01/2022'))
print()

# 2. solde_client
print("Solde du client avec ID 1:")
print("--------------------------")
print(solde_client(transactions, 1))
print()

# 3. transactions_max
print("Les 2 transactions avec le plus grand montant:")
print("----------------------------------------------")
print(transactions_max(transactions, 2))
print()

# 4. clients_sans_retraits
print("IDs des clients sans transactions de retrait:")
print("---------------------------------------------")
print(clients_sans_retraits(transactions))
print()

# 5. convertisseur
print("Transactions converties en EUR selon les taux donnés:")
print("-----------------------------------------------------")
print(convertisseur(transactions, taux_conversion))
print()

# 6. retraits_supérieurs_a
```

```
print("Retraits supérieurs ou égaux à 250:")
print("----------------------------------")map
print(retraits_supérieurs_a(transactions, 250))
```

Vous devez obtenir l'affichage suivant :

```
Transactions entre le '02/01/2022' et le '05/01/2022':
--------------------------------------------------
[{'client_id': 2, 'date': '03/01/2022', 'type': 'retrait', 'montant': 200, 'devise': 'USD'}, {'client_id': 1, 'date': '04/01/2022', 'type': 'dépôt', 'montant': 500, 'devise': 'EUR'}, {'client_id': 2, 'date': '05/01/2022', 'type': 'dépôt', 'montant': 300, 'devise': 'USD'}, {'client_id': 3, 'date': '05/01/2022', 'type': 'dépôt', 'montant': 700, 'devise': 'EUR'}]

Solde du client avec ID 1:
------------------------
1500

Les 2 transactions avec le plus grand montant:
--------------------------------------------
[{'client_id': 1, 'date': '01/01/2022', 'type': 'dépôt', 'montant': 1000, 'devise': 'EUR'}, {'client_id': 3, 'date': '05/01/2022', 'type': 'dépôt', 'montant': 700, 'devise': 'EUR'}]

IDs des clients sans transactions de retrait:
-------------------------------------------
[1, 3]

Transactions converties en EUR selon les taux donnés:
---------------------------------------------------
[{'client_id': 1, 'date': '01/01/2022', 'type': 'dépôt', 'montant': 1000, 'devise': 'EUR'}, {'client_id': 2, 'date': '03/01/2022', 'type': 'retrait', 'montant': 170.0, 'devise': 'EUR'}, {'client_id': 1, 'date': '04/01/2022', 'type': 'dépôt', 'montant': 500, 'devise': 'EUR'}, {'client_id': 2, 'date': '05/01/2022', 'type': 'dépôt', 'montant': 255.0, 'devise': 'EUR'}, {'client_id': 3, 'date': '05/01/2022', 'type': 'dépôt', 'montant': 700, 'devise': 'EUR'}]

Retraits supérieurs ou égaux à 250:
---------------------------------------------------
[]
```

Rappels de cours : Dictionnaires en Python

Un dictionnaire est une collection non ordonnée, modifiable et indexée. En Python, les dictionnaires sont écrits entre accolades `{}`, et ils sont composés de clés et de valeurs.

1. **Création d'un dictionnaire** :

```
mon_dictionnaire = {
    "nom": "Dupont",
    "prénom": "Jean",
    "âge": 30
}
```

2. **Accéder à une valeur** : Utilisez la clé pour obtenir la valeur associée :

```
x = mon_dictionnaire["prénom"]  # Jean
```

3. **Modification d'une valeur**

```
mon_dictionnaire["âge"] = 31  # Change l'âge de 30 à 31
```

4. **Ajout d'une paire clé-valeur**

```
mon_dictionnaire["ville"] = "Paris"  # Ajoute une nouvelle paire
clé-valeur
```

5. **Supprimer un élément** : Utilisez la méthode `pop()` ou le mot-clé `del` :

```
mon_dictionnaire.pop("ville")       # Supprime l'élément avec la
clé "ville"
del mon_dictionnaire["prénom"]      # Supprime également l'élément
avec la clé "prénom"
```

6. **Parcourir un dictionnaire**:

- Par clé:
```
for clé in mon_dictionnaire:
print(clé)
```
- Par valeur
```
for valeur in mon_dictionnaire.values():
print(valeur)
```
- Par clé et valeur :
```
for clé, valeur in mon_dictionnaire.items():
p rint(clé, valeur)
```