II - TP 1

Parcourir un itérable

1 Itérables et parcours par valeur

Les codes suivants sont équivalents : ils affichent tous successivement les nombres 0, 1, . . ., 5.

Code python
```
for i in range(5):
    print(i)
```

Code python
```
for i in [0, 1, 2, 3, 4]:
    print(i)
```

Code python
```
for i in 0, 1, 2, 3, 4:
    print(i)
```

Code python
```
for i in (0, 1, 2, 3, 4):
    print(i)
```

On dit que `range(5)`, la liste `[0, 1, 2, 3, 4]`, et le tuple `(0, 1, 2, 3, 4)` sont des **itérables**. On appelle `i` la **variable de boucle** (on peut lui donner un autre nom si on le souhaite). On exécute alors le corps de la boucle autant de fois qu'il y a d'élément présent dans l'itérable, la variable de boucle prenant successivement chacune des valeurs de l'itérable.
Les chaînes de caractères sont également des itérables. Les éléments qui le constituent sont alors chacun des caractères de la chaîne de caractères. Par exemple, les deux codes ci-dessous sont équivalents (on utilise `c` comme nom pour la variable de boucle car les valeurs successives de celle-ci sont les **c**aractères du mot `"bonjour"`) :

Code python
```
for c in "bonjour":
    print(c)
```

Code python
```
for c in "b", "o", "n", "j", "o", "u", "r":
    print(c)
```

Attention toutefois à ce sur quoi on itère :

Code python
```
for m in "bonjour", "tout", "le", "monde !":
    print(m)
```

Dans cet exemple, l'itérable est constitué de quatre éléments, la variable de boucle `m` (pour **m**ot) prend successivement les valeurs `"bonjour"`, `"tout"`, `"le"`, et enfin `"monde !"`.

1.1 Compter le nombre d'éléments d'un itérable

La **longueur** d'un itérable `iter` est définie comme le nombre d'éléments qui le composent.
Par exemple :

- `[7, 8, 9]` est constitué de 3 éléments : 7, 8 et 9 ;
- `"bonjour"` est constitué de 7 éléments : `"b"`, `"o"`, . . ., `"r"` ;
- `["salut", "toi"]` est constitué de 2 éléments `"salut"` et `"toi"`.

Écrire une fonction `longueur` qui prend en argument un itérable `iter` et qui en renvoie la longueur.
Vous **ne pouvez** pas utiliser la fonction python `len`, et vous **devez** utiliser une boucle `for`.

Code python

```python
def longueur(iterable):
    """ Iterable -> int
    Compte le nombre d'éléments dont est constitué iterable """
    n = 0
    for e in iterable:
        # À compléter
    return # À compléter
```

Code python

```python
assert longueur("bonjour") == 7
assert longueur([7, 8, 9]) == 3
assert longueur(["salut", "toi"]) == 2
```

1.2 Déterminer si un élément fait partie d'un itérable

Écrire une fonction `appartient` qui prend en argument un itérable `iter` et un élément recherché `elem` renvoie `True` si `elem` est un des éléments de l'itérable et `False` sinon.
Par exemple :

- `appartient([7, 8, 9], 9)` renvoie `True` car 9 est le troisième élément de l'itérable `[7, 8, 9]` ;
- `appartient([7, 8, 9], 5)` renvoie `False` car 5 n'est pas un des éléments de l'itérable ;
- `appartient("bonjour", "b")` renvoie `True` car `"b"` est le premier élément de l'itérable `"bonjour"` ;
- `appartient(["salut", "toi"], "t")` renvoie `False` car `"t"` n'est pas un des éléments de l'itérable.

Code python

```python
def appartient(iterable):
    """ Iterable, Element -> bool
    Renvoie True si elem est l'un des éléments de l'itérable, False sinon
    ↪ """
    pass
```

Code python

```python
assert appartient([7, 8, 9], 9) == True
assert appartient([7, 8, 9], 5) == False
assert appartient("bonjour", "b") == True
assert appartient(["salut", "toi"], "t") == False
```

Question 1. Que renvoie `appartient(["bonjour"], "b")` ? Justifier votre réponse.

1.3 Compter le nombre d'occurences d'un élément

Lorsqu'un élément apparaît dans un itérable, on parle d'une occurrence de cet élément dans l'itérable. Si celui-ci apparaît plusieurs fois, alors on dit qu'il y a plusieurs occurrences de cet élément dans l'itérable.
Par exemple, `"n"` apparaît 2 fois dans `"bonjour"`, on dit que `n` possède deux occurences dans `"bonjour"`. De même 1 possède 4 occurences dans `[0, 1, 0, 0, 1, 1, 1]`, mais aucune dans `[4, 5, 6]`.
Écrire une fonction `nombre_occurrences` qui prend en arguments un itérable `iterable` et un éléments `elem` et qui renvoie le nombre d'occurrences de `elem` dans `iterable`.

Code python

```python
def nombre_occurrences(iterable, elem):
    """ Iterable, Element -> int
    Renvoie le nombre de fois qu'apparaît elem dans iter """
    pass
```

Question 2. Écrire les tests correspondant aux exemples donnés dans l'énoncé. Vérifier que votre fonction passe les tests.

1.4 Plus long mot d'une liste

Écrire une fonction `plus_long_mot` qui prend en argument une liste de mots et qui renvoie le nombre de lettre du plus long mot de la liste.
Pour cela, vous parcourerez tous les mots de la liste `mots`. Pour chaque mot `m` de la liste de `mots`, vous calculerez le nombre de lettres `n` du mot `m`. Puis, vous comparerez `n` au `record` du plus long mot découvert jusqu'à présent :

- si `n` est supérieur à `record`, alors c'est que le mot `m` est le plus long parmi les mots parcourus, il faut donc mettre à jour `record`.
- sinon, le `record` du mot le plus long reste inchangé.

Lorsque tous les éléments ont été parcourus, on renvoie la valeur de `record`.

Code python

```python
def plus_long_mot(mots):
    """ [str] -> int
    Renvoie la longueur du plus long mot parmis les mots """
    record = 0 # aucun mot n'a été étudié
    for ...:
        if ...:
            record = ...
    return ...
```

Code python

```python
assert plus_long_mot(["bonjour", "tout", "le", "monde"]) == 7
assert plus_long_mot(["vive", "la", "nsi!!!"]) == 6
assert plus_long_mot(["Je", "m'appelle", "Alfred Dupond"]) == 13
```

Question 3.

1. Justifier les assertions ci-dessus.

2. Compléter les `...` du code de la fonction `plus_long_mot` et tester votre fonction.

   Vous pouvez éventuellement insérer une ou des lignes de code supplémentaires par rapport au code donné dans l'énoncé si vous le souhaitez, bien que cela ne soit pas obligatoire.

3. Adapter le code de la fonction `plus_long_mot` en une fonction `mot_plus_long` qui prend en entrée une liste de `mots` et qui renvoie le **mot** le plus long de la liste. Par exemple, `mot_plus_long(["bonjour", "tout", "le", "monde"])` devra renvoyer `"bonjour"`.

1.5 Liste de conditions

Dans cet exercice, on considère une liste `lst` de booléens. Par exemple, `lst` peut valoir `[True, True, False]`, ou bien `[False, True]`.
L'objectif de cet exercice est d'écrire le code des quatres fonctions `existe_vrai`, `existe_faux`, `tout_vrai`, `tout_faux` dont on donne les spécifications ci-dessous.

Code python

```
def existe_vrai(lst):
    """ [bool] -> bool
    Renvoie True si un des booléens
    ↪ de lst est True, renvoie
    ↪ False sinon """
    pass
```

Code python

```
def tout_faux(lst):
    """ [bool] -> bool
    Renvoie True si tous les
    ↪ booléens de lst sont False,
    ↪ renvoie False sinon """
    pass
```

Code python

```
def tout_vrai(lst):
    """ [bool] -> bool
    Renvoie True si tous les
    ↪ booléens de lst sont True,
    ↪ sinon renvoie False """
    pass
```

Code python

```
def existe_faux(lst):
    """ [bool] -> bool
    Renvoie True si un des booléens
    ↪ de lst est False, renvoie
    ↪ False sinon """
    pass
```

Question 4. **1.** Écrire un jeu de trois test pour chacune des fonctions.

2. Écrire le code des quatre fonctionsen parcourant la liste `lst` à l'aide d'une boucle `for`. Vérifier que les fonctions ainsi écrites passent les tests proposés.

3. **a.** Donner deux liens logiques entre deux paires de fonctions parmi `existe_vrai`, `tout_faux`, `tout_vrai` et `existe_faux`.

b. Simplifier le code des fonctions.

2 Parcours par indice d'un itérable

Parfois, parcourir les valeurs des éléments d'un itérable les unes après les autres ne suffit pas. Pour comprendre cela, considérons le problème suivant. Une station météorologique relève la température extérieure chaque jour. À la fin du mois, elle calcule la température moyenne sur le mois. Les données sont stockées dans une liste `temperatures`.

Code python

```
mois = ["Janvier", "Février", "Mars", "Avril", "Mai", "Juin",
        "Juillet", "Août", "Septembre", "Octobre", "Novembre", "Décembre"]
temperatures = [24.8, 25.8, 25.7, 25.2, 24.9, 23.9,
                23.3, 23.2, 23.7, 24.1, 24.5, 24.4]
```

Question 5. **1.** D'après les données présentes dans le relevé `temperatures` :

a. Quel est le premier mois pendant lequel la température moyenne est inférieure à $24^\circ C$ en moyenne ?

b. Quel est le mois le plus chaud en moyenne ? Le mois le plus froid en moyenne ?

2. Expliquer pourquoi on ne peut pas écrire une fonction python qui réponde aux questions précédentes en parcourant par valeur la liste `temperatures`.

Dans une liste, chaque élément est repéré de manière unique à l'aide de son **indice**, en commençant à partir de 0.
Par exemple, on définit `liste` de la manière suivante :

Code python
```
liste     = ["b", "o", "n", "j", "o", "u", "r"]
# indices    0    1    2    3    4    5    6
```

On a alors :

- l'élément d'indice 0 est `"b"`, c'est le premier élément de `liste`. On le note `liste[0]`.
- l'élément d'indice 6 est `"r"`, c'est le dernier élément de `liste`. On le note `liste[6]`.
- les deux occurrences de `"o"` ont pour indice 1 et 4. Ce sont les deuxièmes et cinquièmes éléments de `liste`. On les note `liste[1]` et `liste[4]`

Code python
```
print(liste[0])
print(liste[6])
print(liste[1], liste[4])
```

Résultat
```
b
r
o o
```

Question 6. **1.** Donner `mois[1]`, `mois[4]`, `temperatures[6]`, `temperatures[7]`.

2. Donner `mois[longueur(mois) - 2]`

3. `lst` est une liste de `n` éléments, et `i` est un indice d'un élément de cette liste.

a. Quel est le lien entre `lst[i]` et `lst[i + 1]` ? Entre `lst[i - 1]` et `lst[i]` ?

b. À quoi correspond `lst[n - 1]` ?

4. On considère le code de la fonction `premier_mois_sous` ci-dessous.

Code python
```
def premier_mois_sous24(temperatures):
    """ float -> int
    Renvoie l'indice du premier mois dont la température mensuelle
    ↪ moyenne est inférieure à 24C.
    Renvoie -1 si aucun mois ne possède une température mensuelle
    ↪ moyenne inférieure à 24C. """
    for i in range(len(temperatures)):
        t = temperatures[i]
        if ...:
            return ...
    return ...
```

a. Compléter et tester le code.

b. Adapter le code de `premier_mois_sous24` en une fonction `premier_mois_sous` qui prend en arguments une liste de `temperatures` et une température `seuil` et qui renvoie le **nom** du premier mois pendant lequel la température mensuelle moyenne est inférieure à `seuil`.

Votre fonction renverra `""` si aucune température n'est inférieure à `seuil`. Vous utiliserez la liste `mois` pour déterminer le nom du mois à partir de son indice.

5. Challenge. Écrire une fonction `mois_plus_chaud` qui prend en argument une liste `temperatures` et qui renvoie le nom du mois le plus chaud correspondant.

2.1 Prix total d'un panier

Un supermarché souhaite écrire un code python pour faire fonctionner sa caisse automatique. Lorsqu'un client passe en caisse, il scanne son panier et trois listes sont automatiquement créées :

- une liste `objets` qui contient le nom des objets scannés par le client ;
- une liste `quantite` qui contient la quantité de chaque objet ;
- une liste `prix` qui contient le prix à l'unité de chaque objet.

Par exemple, on peut avoir :

Code python
```
objets = ["pomme", "ananas", "prune"]
quantite = [3, 1, 6]
prix = [100, 500, 50]
```

Dans cette situation, le client souhaite acheter 3 pommes à 100 francs, 1 ananas à 500 francs, et 6 prunes à 50 francs. Il doit donc payer 1100 francs.

Compléter les ... présents dans le code de la fonction `prix_total` ci-dessous.

Code python
```
def prix_total(quantite, prix):
    """ [int], [int] -> int
    quantite et prix sont deux listes de même taille
    Renvoie le prix total que doit payer le client """
    p = 0
    for i in ...:
        p = p + ...
    return ...
```

Code python
```
assert prix_total([3, 1, 6], [100, 500, 50]) == 1100
```

2.2 Deux éléments consécutifs identiques

Écrire une fonction `deux_suivants_egaux` qui prend en argument une liste `lst` et qui renvoie `True` s'il existe deux éléments consécutifs identiques, `False` sinon.
On rappelle que si `i` est l'indice d'un élément de `lst`, alors `i + 1` est l'indice de l'élément suivant dans `lst`.

Code python
```
def deux_suivants_egaux(lst):
    """ list -> bool
    Renvoie True si deux éléments consécutifs de la liste sont égaux,
    ↪ False sinon """
    pass
```

Code python
```
assert deux_suivants_egaux([1, 2, 3, 1, 2, 3]) == False
assert deux_suivants_egaux([3, 4, 5, 5, 1]) == True
assert deux_suivants_egaux([True, True, False]) == True
```

Question 7. Adapter le code de la fonction `deux_suivants_egaux` en une fonction `est_trie_croissant` qui prend en argument une liste d'entiers `lst` et qui renvoie `True` si celle-ci est triée par ordre croissant, `False` sinon.

Par exemple :

Résultat

```
>>> est_trie_croissant([1, 2, 3])
True
>>> est_trie_croissant([1, 3, 2, 4, 5, 6])
False
```

Pour cela, on pourra parcourir les éléments de la liste les uns après les autres: si on trouve deux éléments `a` et `b` consécutifs (`a` apparaissant en premier premier dans la liste, `b` suivant `a`) tels que `a` est plus grand que `b`, alors on est certain que la liste n'est pas triée par ordre croissant, et on peut renvoyer `False` immédiatement.
Si on a parcouru toute la liste sans rencontrer de "problème", c'est que la liste est triée par ordre croissant et on peut renvoyer `True`.

2.3 Palindrome

On appelle palindrome un mot qui se lit de la même manière de la droite vers la gauche ou bien de la gauche vers la droite.
Par exemple, `"radar"`, `"ici"`, `"kayak"` sont des palindromes.

Question 8. **1.** `mot` est un palindrome constitué de $n = 9$ lettres (par exemple `"ressasser"`).

Quels sont les paires d'indices qui doivent nécessairement correspondre à des caractères égaux de `mot` ?

2. `mot` est une chaîne de caractère constiuée de `n` caractères.

a. Quel est l'indice du premier caractère ? Quel est l'indice du dernier caractère ?

b. Quel est l'indice du second caractère ? Quel est l'indice de l'avant dernier caractère ?

c. Quel est l'indice du troisième caractère ? Quel est l'indice de l'antépénultième caractère ?

3. Écrire une fonction `est_palindrome` qui prend en argument une chaîne de caractère `mot` et qui renvoie `True` si `mot` est un palindrome, `False` sinon.

On parcourra par indice le `mot` et on renverra `False` dès que l'on est certain que le mot n'est pas un palindrome. On renverra `True` si on a parcouru l'intégralité des caractères de `mot` sans "problème".

Code python

```
def est_palindrome(mot):
    """ str -> bool
    Renvoie True si mot est un palindrome, False sinon """
    pass
```

Code python

```
assert est_palindrome("kayak") == True
assert est_palindrome("nsi") == False
assert est_palindrome("ici") == True
```

Question 9. L'eibohphobie est la peur des palindromes.
Que renvoie `est_palindrome("eibohphobie")` ?