TD12. Redonnons la main

Bases de la Programmation Impérative – Chapitre 3 : references
version du 25 décembre 2023

L'objectif de ce TD est d'introduire les générateurs Python, c'est à dire le mot-clef `yield`, et le concept fondamental d'itérateur qui se cache derrière ainsi que le concept d'itérable spécifique à Python.

Exercice 1 : quel est le problème ?

Question 1

Analysez attentivement le code ci-dessous. Combien de lignes du fichier d'entrée faut-il lire pour que le programme affiche la note de l'étudiant situé en toute première position dans le fichier ?

```python
#!/usr/bin/env python3

import sys

def traite_fichier(nom_fichier):
    fichier = open(nom_fichier, "r")
    resultats = []
    for ligne in fichier:
        ligne_decoupee = ligne.split(" ")
        prenom = ligne_decoupee[0]
        note = int(ligne_decoupee[1])
        resultats.append((prenom, note))
    fichier.close()
    return resultats

def cherche_note(prenom, prenoms_notes):
    for resultat in prenoms_notes:
        if resultat[0] == prenom:
            return resultat[1]
    return None

def get_note():
    if len(sys.argv) != 3:
        print("Usage :", sys.argv[0], "nom_fichier prenom")
        return
    prenoms_notes = traite_fichier(sys.argv[1])
    prenom = sys.argv[2]
    print("La note de", prenom, "est", cherche_note(prenom, prenoms_notes))

if __name__ == "__main__":
    get_note()
```

Question 2

Cette fois, on cherche à calculer la moyenne des notes se trouvant dans le fichier. Que peut-on dire sur l'espace mémoire occupé par ce programme ?

```python
#!/usr/bin/env python3

import sys
```

```python

def traite_fichier(nom_fichier):
    fichier = open(nom_fichier, "r")
    resultats = []
    for ligne in fichier:
        ligne_decoupee = ligne.split(" ")
        prenom = ligne_decoupee[0]
        note = int(ligne_decoupee[1])
        resultats.append((prenom, note))
    fichier.close()
    return resultats

def calcule_moyenne(prenoms_notes):
    somme = 0
    nb_etudiants = 0
    for resultat in prenoms_notes:
        somme += resultat[1]
        nb_etudiants += 1
    return somme / nb_etudiants

def teste():
    """Teste les fonctions ci-dessus."""
    if len(sys.argv) != 2:
        print("Usage :", sys.argv[0], "nom_fichier")
    else:
        prenoms_notes = traite_fichier(sys.argv[1])
        print("La note moyenne est", calcule_moyenne(prenoms_notes))

if __name__ == "__main__":
    teste()
```

Question 3

> Comment corriger le problème de calculs inutiles et de mémoire dans le cas de la recherche d'un étudiant uniquement avec ce que nous avons vu jusqu'ici, c'est-à-dire sans avoir recours à `yield` ?

Question 4

> Quel est l'inconvénient de cette solution ?

Exercice 2 : il n'y a pas de problème, il n'y a que des solutions

Voici donc comment, à l'aide de l'utilisation d'un `yield` Python, avoir une recherche d'étudiant dans laquelle :

- nous ne faisons aucun calcul pour rien ;
- nous ne créons pas de `list` d'étudiants, donc utilisation mémoire constante ;
- le traitement du fichier et la recherche sont séparés dans deux fonctions.

```python
#!/usr/bin/env python3

import sys

def traite_fichier(fichier):
    for ligne in fichier:                                   # tf1 (bof, non ?)
        ligne_decoupee = ligne.split(" ")                   # tf2
        prenom = ligne_decoupee[0]                          # tf3
        note = int(ligne_decoupee[1])                       # tf4
```

```
10          yield (prenom, note)                                    # tf5
11
12  def cherche_note(prenom, resultats):
13      for resultat in resultats:                                  # cn1
14          if resultat[0] == prenom:                               # cn2
15              return resultat[1]                                  # cn3
16      return None                                                 # cn4
17
18  def get_note():
19      if len(sys.argv) != 3:                                      # gn1
20          print("Usage :", sys.argv[0], "nom_fichier prenom")     # gn2
21          return                                                  # gn3
22      fichier = open(sys.argv[1], "r")                            # gn4
23      iterateur_prenoms_notes = traite_fichier(fichier)           # gn5
24      prenom = sys.argv[2]                                        # gn6
25      note = cherche_note(prenom, iterateur_prenoms_notes)        # gn7
26      print("La note de", prenom, "est", note)                    # gn8
27      fichier.close()                                             # gn9
28
29  if __name__ == "__main__":                                      # 1
30      get_note()                                                  # 2
```

Question 1

Analyser attentivement le code ci-dessus. En essayant de “deviner” ce que fait le `yield`, prendre quelques minutes pour dérouler le programme sur papier en écrivant les numéros de lignes successivement exécutées dans le cas d’un appel correct, c’est à dire avec un fichier qui existe et un nom d’étudiant qui existe en troisième position du fichier.

Question 2

À l’aide d’un générateur, réécrire le programme qui calcule la moyenne des notes. Quel est l’avantage par rapport au calcul de moyenne de l’exercice 1 ?

Exercice 3 : générer les jours de la semaine.

Question 1

Écrire une fonction génératrice renvoyant un itérateur sur les chaînes de caractères représentant les jours de la semaine.

Question 2

Qu’affiche le programme suivant ?

```
1  #!/usr/bin/env python3
2
3  """Petit exemple pour illustrer le contexte de CHAQUE itérateur 1 et 2"""
4
5  def get_first_five():
6      """Fonction génératrice des 5 premiers nombres."""
7      yield "one"
8      yield "two"
```

```
9       yield "three"
10      yield "four"
11      yield "five"
12
13  def affiche():
14      """Affiche des trucs en utilisant DEUX itérateurs."""
15      iterateur_1 = get_first_five()
16      iterateur_2 = get_first_five()
17
18      print("De l'itérateur 1", next(iterateur_1))
19      print("De l'itérateur 1", next(iterateur_1))
20      print("De l'itérateur 2", next(iterateur_2))
21      print("De l'itérateur 1", next(iterateur_1))
22
23  if __name__ == "__main__":
24      affiche()
```

Exercice 4 : analyse de code

Question 1

> Qu'affiche le programme ci-dessous ?

```
1   #!/usr/bin/env python3
2
3   """Train reading someone else (bad) code"""
4
5
6   def mystery_function(first_parameter, second_parameter, everything):
7       """What am I doing?"""
8       for variable in first_parameter:
9           everything.append(variable[second_parameter])
10          yield variable[second_parameter]
11
12
13  def main():
14      """Script's entry point"""
15      a_list = []
16      something = mystery_function([("to", 12), ("ti", 17), ("ta", 47)], 1, a_list)
17      print("type of something is", type(something))
18      for element in something:
19          print(element, end=" ")
20      print()
21      something_else = mystery_function(("to", "ti", "\u0634 what is that?"), 0,
            a_list)
22      print("type of something_else is", type(something_else))
23      for element in something_else:
24          print(element, end=" ")
25      print()
26      print(a_list)
27      other_thing = mystery_function({(1, 2, 3), (4, 5, 6), (7, 8)}, 2, a_list)
28      print("type of other_thing is", type(other_thing))
29      for element in other_thing:
30          print(element, end=" ")
31      print()
32
33
34  if __name__ == "__main__":
35      main()
```

Exercice 5 : quand aurions nous pu/dû utiliser `yield` ? (pour aller plus loin)

Question 1

> Chercher dans ses notes, sa mémoire, son ordinateur, à quels endroits nous aurions pu/dû utiliser `yield` dans le cadre des TD et TP BPI en justifiant pourquoi ?