Langage de programmation (3/3)

Chapitre 4

I- La boucle répétitive for

1) Rappels sur le pseudo-code

La boucle pour est utilisée quand on connait le nombre de fois où l'on doit répéter une suite d'instructions.
Voici comment écrire cette boucle répétitive dans le cas où on désire n répétitions, n étant un nombre entier naturel non nul.

- **Structure de la boucle "pour" en pseudo-code :**

```
1 for i←0  to  n-1
2       ÉCRIRE i
```

Pour i allant de 0 à n-1 faire

```
Pour i allant de 0 à n-1 faire :
    ...
    les différentes instructions à répéter
    ...
Fin pour
```

- **Autre écriture de la boucle "pour" en pseudo-code :**

```
for i←0  to  n-1
   ...
   les différentes instructions à répéter
   ...
```

Exemple 1 :

On reprend et prolonge un exercice fait dans le chapitre 1. On veut un algorithme qui donne la somme des n premiers entiers naturels. Par exemple, pour n=4, on veut obtenir la valeur de 0 + 1 + 2 + 3 = 6.
Voici une écriture du programme de cet exemple en pseudo-code :

```
1 S ← 0
2 for i←0  to  n-1
3    S ← S+i
4 ÉCRIRE S
```

Pour plus de rappels sur la boucle for en pseudo-code, relire le chapitre 1.

2) La structure de la boucle for en python avec range

La première ligne fait que la suite du code va être répétée n fois en incrémentant de manière automatique la variable i de 0 à n-1. Dans un premier temps, nous allons utiliser l'instruction range.

Propriété 1 :

Syntaxe à utiliser pour répéter un bloc d'instruction un nombre n de fois connu à l'avance :

```
for i in range(n):
    ...
    les différentes instructions à répéter
    ...
```

Remarques :

- Il n'y a pas de Fin_Pour : C'est en fait l'indentation qui sert de limitateur de boucle. Attention ! L'indentation en python n'est pas uniquement un élément d'hygiène de programmation mais un élément faisant partie intégrante des structures.
- L'initialisation et l'incrémentation du compteur est gérée automatiquement par la boucle for.
- On utilise habituellement les lettres i, j, k, ... pour les compteurs, mais ce n'est pas une obligation.
- On peut imbriquer plusieurs boucles for.

Propriété 2 :

```
for i in range(n):
```

Equivaut à :

```
for i in range(0,n,1):
```

```
for i in range(m,n):
```

Equivaut à :

```
for i in range(m,n,1):
```

Exemple 2 :

On peut réécrire en langage Python, sous forme d'une fonction l'algorithme vu dans l'exemple précédent ainsi :

```
def somme(n: int)->int: #les int sont ici pour indiquer le typage des variables d'entrée et de sortie. Ici le paramètre d'ent
    """La fonction somme renvoie la somme des n premiers entiers naturels.
    la fonction somme a pour attribut un entier (int) : n et renvoie un autre entier :S
    Pour n=4 : S=0+1+2+3+4=10"""
    S=0         #initialisation de la somme à 0
    for i in range(n+1):
        S=S+i
    return S
```

Vous pouvez visualiser la trace d'exécution de cet algorithme grâce au pythontutor.

Exercice 1 :

1. Modifier la fonction somme de l'exemple précédent en une fonction somme1 qui renvoie la somme des n+1 premiers entiers naturels.
   Exemple : pour somme1(4) renvoie 10 car S=0+1+2+3+4=10.
2. Modifier la fonction somme de l'exemple précédent en une fonction somme2 qui prend toujours comme paramètre un entier naturel n et renvoie la somme de tous les nombres entiers **impairs** inférieurs ou égaux à n.
   Exemple : pour somme2(6) renvoie 9 car S=1+3+5=9.

Exercice 2 :

Écrire une procédure table_multi qui reçoit en entrée un nombre entier n compris entre 0 et 10 et affiche en sortie la table de multiplication de ce nombre.
Par exemple, table_multi(5) conduit à l'affichage suivant (la partie centrale n'a pas été reproduite) :

- 0 fois 5 = 0
- 1 fois 5 = 5
- 2 fois 5 = 10
- [...]
- 9 fois 5 = 45
- 10 fois 5 = 50

Exercice 3 :

À la naissance de Crésus, son grand-père Midas, lui ouvre un compte bancaire. Ensuite, à chaque anniversaire, le grand père de Crésus verse sur son compte 100 €, auxquels il ajoute le double de l'âge de Crésus.
Par exemple, lorsqu'elle a deux ans, son grand-père lui verse 104 € ce qui fait un total versé de 100+102+104=306 € à la deuxième année.
Écrire une fonction tresorApo() qui en entrée reçoit un entier n et en sortie renvoie la somme présente sur le compte de Crésus à sa n-ième année écoulée entièrement.

3) for avec un itérable

Définition 1 :

Dans la partie précédente, l'instruction range(m,n) créé un ensemble de nombres dont le compteur i parcourait les valeurs. Un tel ensemble dont un peut parcourir les valeurs avec une boucle for est appelé un **itérable**.

Exemple 3 :

Au cours de cette année nous verrons progressivement différents itérables :

- Une chaîne de caractères (type str) est un **itérable**,
- Une liste (type list) est un **itérable**,
- Un dictionnaire (type dict) est un **itérable**.

Propriété 3 :

En Python, pour utiliser un itérable dans une boucle for, il suffit d'utiliser le mot-clé in. Voici la structure générale :

```
for elt in iterable:
    instruction 1
    instruction 2
    ...
```

Dans cette boucle, elt va parcourir les éléments de iterable.

Exemple 4 :

Le programme ci-dessous affiche séparément chaque caractère d'une chaîne de caractères :

```
ch="Bonjour à tou.te.s"
for elt in ch:
    print(elt)
```

Exercice 4 :

Tester le script ci-dessus pour vérifier qu'il réalise bien ce qui a été annoncé et modifier la chaîne de caractères avec une autre phrase (Vous pouvez aussi rajouter des nombres et des symboles).

Exercice 5 :

1. Que fait le code suivant :

```
citation="Je ne cherche pas à connaître les réponses, je cherche à comprendre les questions."
compteur=0
for elt in citation:
    if elt=="e":
        compteur=compteur+1
print(compteur)
```

2. Transformer le code pour qu'il compte le nombre de i.

Exercice 6 :

1. Écrire une fonction est_dans_liste qui prend en paramètres une liste lst et un entier n et qui renvoie True si l'entier est dans la liste et False sinon.
2. Tester la fonction est_dans_liste, entre autres, est_dans_liste([8,4,7,2,5,4],6) renvoie False et est_dans_liste([8,4,7,2,5,4],4) renvoie True.
3. Proposer au moins deux autres tests pour valider cette fonction sous forme de postconditions. Vous utiliserez la commande assert pour vos postconditions. Par exemple assert est_dans_liste([8,4,7,2,5,4],6) == False

Exercice 7 :

Voici un programme qui permet de représenter le coût en temps du parcours sur un itérable comme range :

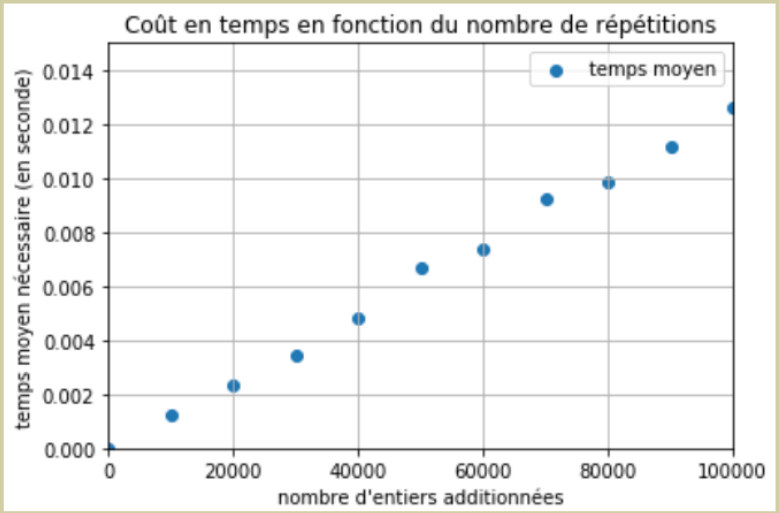
```python
# importation des bibliothèques
from time import perf_counter
import matplotlib.pyplot as plt

# fonction dont le coût moyen est évalué en fonction de n
def somme(n:int)->int:
    S=0 #
    for i in range(n+1):
        S=S+i
    return S

# calcul du coût moyen en temps
def temps() -> list:
    Ly = []
    for i in range(11):
        deb = perf_counter()
        # 100 répétitions du calcul de la somme pour la même valeur de n
        for j in range(100):
            somme(10000*i)
        Ly.append((perf_counter() - deb)/100)  # calcul du temps moyen pour cette valeur de n
    return Ly

# gestion du graphique
Lx = [10000*i for i in range(11)] # création de la liste des abcisses
Ly = temps() # Les différentes valeurs de Ly correspondront aux ordonnées des points
plt.axis([0,100000, 0, 0.015])    # fenêtre du graphique [x_min,x_max,y_min,y_max]
plt.xlabel("nombre d'entiers additionnées")       # légende de l'axe des abscisses
plt.ylabel('temps moyen nécessaire (en seconde)') # légende de l'axe des orodnnées
plt.title('Coût en temps en fonction du nombre de répétitions')  # titre du graphique
plt.grid()                                # graduation
plt.scatter(Lx,Ly, label='temps moyen')   # nuage de points avec une légende explicative
plt.legend()        # pour faire apparaître la légende explicative précédente
plt.show()          # À ne pas oublier : pour faire appraître le graphique
```

1. Exécuter le programme précédent (dans un jupyter-notebook).

Remarques :

Remarque(s) :

Normalement, vous verrez apparaître un graphique proche de celui-ci :



Si vous n'avez pas 11 points sur votre graphique, remplacer dans `plt.axis([0,100000, 0, 0.015])` 0.015 par un nombre un peu plus grand (tout dépend du processeur de votre ordinateur).

2. Que pouvez-vous dire quant à la relation liant le temps moyen de calcul et le nombre de répétitions ?
3. Que pouvez-vous dire quant à la relation liant le temps moyen de calcul et le nombre de répétitions ?

Définition 2 :

Quand le temps de calcul est **proportionnel** au nombre de répétitions, on dit que le **coût en temps est linéaire**.

II- La boucle répétitive while

1) Rappels sur la boucle répétitive conditionnelle en pseudo-code

La boucle répétitive conditionnelle **Tant que / while** est utilisée quand on ne connaît pas le nombre de fois où l'on doit répéter une suite d'instructions. Une autre différence majeure avec la boucle pour est l'absence d'incrémentation automatique.

Structure de la boucle "tant que" en pseudo-code :

```
1 while condition vraie
2       bloc d'instructions à répéter
```

Dans la boucle while :

- la condition est évaluée une première fois ;
- si la condition est **Fausse**, on n'entre pas dans la boucle et on continue après le bloc des instructions du while ;
- si la condition est **Vraie**, on entre dans la boucle :
  - les instructions du bloc des instructions du while sont exécutées,
  - le booléen est à nouveau évalué,
  - et ainsi de suite, autant de fois que nécessaire pour que la condition vaille Faux.

Remarques :

Deux écueils à éviter :

- Si la condition au départ vaut Faux, la boucle while ne sera jamais exécutée.
- Si la condition ne change jamais de valeur, la boucle while ne s'arrêtera pas et l'algorithme restera bloqué sur cette boucle.

Exemple 5 :

On reprend et prolonge un exercice fait dans le chapitre 1. On veut un algorithme qui donne la somme des n premiers entiers naturels. Par exemple, pour n=4, on veut obtenir la valeur de 0+1+2+3=6.
Voici une écriture du programme de cet exemple en pseudo-code avec l'instruction répétitive conditionnelle while :

```
1 S ← 0
2 i ← 0
3 while i < n :
4     S ← S+i
5     i ← i+1
6 ÉCRIRE S
```

Pour plus de rappels sur la boucle while en pseudo-code, relire le cours chapitre 1.

2) La boucle répétitive conditionnelle en langage Python

Propriété 4 :

Syntaxe à utiliser avec l'instruction while :

```
while condition:
    ...
    les différentes instructions à répéter
    ...
```

Remarques :

- Il n'y a pas de Fin_while : C'est en fait l'**indentation** qui sert de limitateur de boucle.
- Contrairement à la boucle for, l'initialisation et l'incrémentation du compteur ne sont pas gérée automatiquement par la boucle while.
- On utilise habituellement les lettres i, j, k, ... pour les compteurs, mais ce n'est pas une obligation.
- On peut imbriquer plusieurs boucles while.

Exemple 6 :

On peut réécrire en langage Python, sous forme d'une fonction l'algorithme vu dans l'exemple précédent ainsi :

```
def somme(n: int) -> int:
    """La fonction somme renvoie la somme des n premiers entiers naturels.
    la fonction somme a pour attribut un entier (int) : n et renvoie un autre entier :S
    Pour n=4 : S=0+1+2+3=6"""
    S = 0 #initialisation de la somme à 0
    i = 0
    while i < n-1:
        S = S+i
        i += 1 #o n aurait pu écrire aussi i=i+1 ; incrémente i de 1 à chaque passage dans la boucle
    return S
```

Vous pouvez visualiser la trace d'exécution de cet algorithme grâce au pythontutor.

Exercice 8 :

1. Modifier la fonction somme de l'exemple précédent en une fonction somme1 qui renvoie la somme des n+1 premiers entiers naturels.
   Exemple : pour somme1(4) renvoie 10 car S=0+1+2+3+4=10.
2. Modifier la fonction somme de l'exemple précédent en une fonction somme2 qui prend toujours comme paramètre un entier naturel non nul n et renvoie la somme de tous les nombres entiers **impairs** inférieurs ou égaux à n.
   Exemple : pour somme2(6) renvoie 9 car S=1+3+5=9.
3. Rajouter une précondition pour assurer le bon fonctionnement de somme2.
4. Proposer au moins deux tests pour valider cette fonction.

Exercice 9 :

Une association militant pour la sauvegarde d'une espèce protégée fait appel à un financement participatif. Cette association a besoin de 13500€ pour mener à bien sa campagne annuelle. Elle aimerait savoir combien de jours seront nécessaires pour obtenir la somme de 13500€ en fonction de la somme moyenne recueillie sur le site. Elle fait appel à vous afin que vous lui créiez un programme répondant à la question.

1. Écrire une fonction attendre qui prend comme paramètre d'entrée un flottant somme indiquant la somme en moyenne recueillie quotidiennement sur le site et renvoie un entier indiquant le nombre de jours nécessaire pour atteindre 13500€.
2. Proposer des préconditions à cette fonction.

Exercice 10 :

On reprend l'exercice qui permettait de savoir si un nombre entier n fait partie d'une liste lst ou non. (Exercice 6)

1. En utilisant une boucle while, écrire une fonction est_dans_liste2 qui prend en paramètres une liste lst et un entier n et qui renvoie True si l'entier est dans la liste et False sinon.

Remarque :

1. Vous pouvez utiliser l'instruction len qui renvoie la taille d'une liste donnée en paramètre.
Exemple : len([1,4,6,5,8,3]) renvoie 6 car 6 termes composent la liste.
2. Attention ! Les éléments d'une liste sont référencés à partie de 0 et non de 1.
Exemple : Si lst = [10,4,6,5,8,3], lst[0] renvoie 10 car 10 est le premier terme de la liste.
lst[1] renvoie 4 car 4 est le deuxième terme de la liste.

2. Tester la fonction est_dans_liste2, entre autres, est_dans_liste2([8,4,7,2,5,4],6) renvoie False et est_dans_liste2([8,4,7,2,5,4],4) renvoie True.

3. On considère dans cette question une liste de taille importante.
   À votre avis les fonctions est_dans_liste, écrite avec un for, et est_dans_liste2, écrite avec un while ont-elles un coût en temps d'exécution proche ou l'une des deux, en général, pourrait être plus rapide à renvoyer un résultat ?

Exercice 11 :

Vous allez programmer un jeu de nombre à deviner contre un ordinateur. Pour commencer l'ordinateur va choisir au hasard un nombre compris entre 1 et 100.
Ensuite, l'utilisateur doit alors deviner ce nombre comme ceci :

- l'utilisateur propose un nombre,
- l'ordinateur lui dit s'il est trop petit ou trop grand,
- et ainsi de suite jusqu'à ce que l'utilisateur ait trouvé le bon nombre.

1. Rappeler les lignes de codes à écrire en Python afin d'obtenir un nombre entier entre 1 et 100.
2. Écrire une procédure plus_ou_moins, sans paramètre d'entrée ni retour qui correspond au jeu du nombre à deviner.

Voici un exemple de traces d'affichages successifs obtenus sur un jupyter-notebook dans le cas où l'ordinateur avez choisi au hasard le nombre 41 :

```
Donnez un entier entre 1 et 100. 50
La réponse est inférieure à votre proposition. Essayez encore !25
La réponse est supérieure à votre proposition. Essayez encore !32
La réponse est supérieure à votre proposition. Essayez encore !41
Bravo ! Vous avez gagné !
```

3. On veut complexifier le jeu en imposant un nombre d'essais maximal.

Modifier la fonction précédente en une nouvelle plus_ou_moins2 qui :

- prend un nombre entier comme paramètre, nombre entier qui correspond au nombre maximum d'essais autorisés,
- permet de jouer comme précédent sans permettre plus d'essais que le nombre maximum entré,
- affiche soit la défaite soit la victoire en précisant le nombre d'essais nécessaires.

4. Proposer des préconditions à cette fonction plus_ou_moins2.

Exercice 12 :

Ayant travaillé durant les grandes vacances dernières, vous avez un pécule de 500€. Vous décidez de placer cet argent en banque.
La banque vous propose un compte rémunéré à 1% par an, c'est-à-dire que chaque fin d'année vous recevez 1% d'intérêt sur la somme présente sur le compte en début d'année.
Vous aimeriez savoir quand vous aurez sur votre compte 700€.
Proposez une fonction, sans paramètre, qui renvoie le nombre d'années d'attente nécessaires pour avoir 700€ sur le compte.

Exercice 13 :

Lorsque l'utilisateur saisit une donnée, il peut être utile de vérifier qu'elle est conforme à ce qu'on attend de lui.

1. Proposer un programme qui demande à l'utilisateur de rentrer un nombre positif.
2. Compléter ce programme afin de demander à l'utilisateur de saisir un nouveau nombre si le nombre entré ne correspond pas à la demande.
3. Modifier le programme précédent pour limiter la répétition à au plus 10 demandes successives.
   Rajouter un message d'arrêt si l'utilisateur "s'obstine" à saisir un nombre strictement négatif.

Exercice 14 :

Votre enseignant vous exploite : il vous demande de créer un programme qui lui renverra la moyenne des notes saisies.
Pour cela, il vous donne le cahier des charges suivant :

- Créer une fonction continuer, sans paramètre d'entrée qui demande à l'utilisateur s'il veut poursuivre ou non la saisie de notes et qui renvie un booléen True en cas de poursuite et False sinon.
- Créer une fonction moyenne qui calcule la nouvelle moyenne liée à l'ajout d'une note.
- Créer dans le programme principal une fonction main qui conduit à demander une nouvelle note et à calculer la moyenne obtenue alors jusqu'à ce que l'utilisateur décide d'arrêter la saisie.

1. Proposer une fonction continuer qui répond au cahier des charges. Faire en sorte que la casse (=minuscule ou majuscule) ne soit pas un problème pour la saisie, par exemple en utilisant ma méthode upper.

Exemple 7.

`"mélanGes".upper()` renvoie `'MÉLANGES'`.

2. Tester la fonction continuer créée.
3. Proposer une fonction moyenne qui répond au cahier des charges. Cette fonction aura trois paramètres :
    - moy un flottant qui correspond à la moyenne pour l'instant obtenue,
    - nb_notes un entier qui correspond au nombre de notes déjà pris en compte dans la moyenne moy,
    - new_note un flottant qui correspond à la nouvelle note à prendre en compte pour la moyenne.

Cette fonction renverra la moyenne de l'ensemble des notes saisies (en comptant aussi la dernière).

Remarque(s) :

Aide mathématique :

On considère deux ensembles de notes A et B de moyenne respective moy_A et moy_B comprenant respectivement nb_A et nb_B notes.
La moyenne des notes formant la réunion de ces deux ensembles est donnée par la formule :

$$moyenne = \frac{moy_A \times nb_A + moy_B \times nb_B}{nb_A + nb_B}$$

4. Tester la fonction moyenne créée.
5. Proposer une fonction main qui répond au cahier des charges. Cette fonction fera appel aux deux fonctions continuer et moyenne et n'aura pas de paramètre ; elle renverra la moyenne de toutes les notes successivement saisies.
6. Tester votre programme principal, par exemple en saisissant successivement les valeurs 10, 19 et 13.

Exercice 15 :

Vous avez décidé de partir en randonnée avec un ami. Celui possède un très grand sac pesant 5.2kg et a une technique très particulière pour le remplir : il prend au hasard des éléments parmi tout le nécessaire à emporter et s'arrête dès que le poids obtenu dépasse 20 kg. Les objets choisis au hasard pèsent tous entre 10 et 500 grammes.
Comme vous êtes très intrigué par sa manière de faire, vous décider de modéliser la situation informatiquement afin d'évaluer en moyenne combien d'éléments va prendre votre ami dans son sac.
Pour cela, vous disposez des informations suivantes :

- Pour modéliser informatiquement l'ensemble de masses des objets pris, il est préférable d'utiliser une liste,
- pour créer une liste vide qui s'appelle lst, on écrit en Python :

```
lst = []
```

- pour rajouter un élément à cette liste lst, par exemple 25, on écrit en Python :

```
lst.append(25)
```

- pour obtenir un nombre entier aléatoire entre deux nombres aa et bb, il suffit d'importer la fonction randint du module random et d'écrire :

```
randint(a,b)
```

1. Proposer une fonction remplir_sac, sans paramètre, qui renvoie la liste des masses des objets du sac de sorte que le remplissage s'arrête dès que la masse totale dépasse 20 kg.
2. Proposer une fonction nb_objets qui prend en paramètres une liste et renvoie le nombre d'éléments de cette liste.
3. Proposer une fonction qui renvoie une moyenne du nombre d'éléments que va prendre cet ami dans son sac.

Exercice 16 :

Écrire une fonction triangle qui prend en paramètre un nombre entier n et qui affiche un triangle isocèle formé d'étoiles.
La hauteur du triangle (c'est-à-dire le nombre de lignes) correspond à l'argument n, comme dans l'exemple ci-dessous.
On s'arrangera pour que la dernière ligne du triangle s'affiche sur le bord gauche de l'écran.
Exemple d'exécution de triangle(6) :