Jeudi 30 janvier 2025 Term Spé N.S.I.

Interrogation (1h50)

(Calculatrice non autorisée)

Exercice 1 *(6 points)*

Dans le tableau ci-dessous, on donne les caractéristiques nutritionnelles, pour une quantité de 100 grammes, de quelques aliments.

	Lait entier UHT	Farine de blé	Huile de tournesol
Énergie (kcal)	65.1	343	900
Protéines (grammes)	3.32	11.7	0
Glucides (grammes)	4.85	69.3	0
Lipides (grammes)	3.63	0.8	100

Figure 1 : Caractéristiques nutritionnelles

Pour chaque aliment, on souhaite stocker les informations dans un objet de la classe `Aliment` définie ci-dessous, où `e,p,g` et `l` sont de type `float` et désignent respectivement les quantités d'énergie, de protéines, de glucides et de lipides de l'aliment.

```
1.  class Aliment:
2.    def __init__(self, e, p, g, l):
3.      self.energie = e
4.      self.proteines = p
5.      self.glucides = g
6.      self.lipides = l
```

1.

a. Ecrire, à l'aide du tableau des caractéristiques nutritionnelles de la Figure 1, l'instruction en langage Python pour instancier l'objet `lait`.

b. Donner l'instruction qui permet d'obtenir la valeur 65.1 de l'objet `lait` instancié dans la question précédente.

Une erreur s'est introduite dans le tableau de la Figure 1 : la masse de protéines dans le lait est 3.4 au lieu de 3.32.

c. Donner l'instruction qui modifie la masse de protéines de l'objet `lait` instancié dans la question **1.a.**

On souhaite ajouter une méthode "`energie_reelle`" à la classe `Aliment` qui calcule l'énergie en kcal d'un aliment en fonction d'une masse donnée.

Par exemple :
Pour 245 grammes de lait, l'énergie réelle sera 245 × 65.1 ÷ 100 = 159.495 kcal.
L'instruction `lait.energie_reelle(245)` renvoie alors `159.495`

2. Recopier et compléter les lignes n°1 et n°2 dans la méthode ci-dessous.

```
1.    def energie_reelle(…………,masse):
2.        return ………………
```

3.

On regroupe les caractéristiques nutritionnelles du tableau de la Figure 1 dans le dictionnaire suivant, les clés étant des chaînes de caractères donnant le nom de l'aliment et les valeurs associées des objets de la classe `Aliment` :

```
nutrition = {'lait' : Aliment(65.1,3.4,4.85,3.63),
             'farine' : Aliment(343,11.7,69.3,0.8),
             'huile' : Aliment(900,0,0,100)
             }
```

a. Donner l'instruction qui permet d'obtenir la valeur énergétique en kcal du lait à partir des données de ce dictionnaire.

b. Donner l'instruction qui permet d'obtenir la valeur énergétique réelle de 220 grammes de lait à partir des données de ce dictionnaire.

Une recette de gâteau (sans œuf) utilise les ingrédients suivants :

- 230 g de farine ;
- 220 g de lait ;
- 100 g d'huile.

Les quantités d'ingrédients, en grammes, sont regroupées dans le dictionnaire suivant :

```
recette_gateau={'lait' : 220, 'farine' :230, 'huile':100}
```

4.

Ecrire, en utilisant la classe `Aliment` et la méthode `energie_reelle`, les instructions nécessaires pour calculer l'énergie réelle totale du gâteau.

Exercice 2 *(7 points)*

Pour tout l'exercice on notera `n` le nombre de cartes et on considérera qu'il est pair.

Pour modéliser le jeu de cartes, on décide d'utiliser une pile qui sera une instance de la classe `Pile` dont on donne ici l'interface.

- Le constructeur Pile ne prend pas de paramètres et renvoie une pile vide.
  `jeu = Pile() # crée une pile vide référencée par jeu`
- La méthode `empile` prend en paramètre une valeur et l'empile sur la pile.
  `jeu.empile(1) # empile la valeur 1 sur la pile jeu`
- La méthode `depile` ne prend pas de paramètres et retire le dernier élément empilé d'une pile non vide et renvoie sa valeur.
  `print(jeu.depile()) # dépile 1 et affiche cette valeur`
- La méthode `est_vide` ne prend pas de paramètres et renvoie un booléen indiquant si la pile est vide.
  `print(jeu.est_vide()) # affiche True puisque la pile est vide`

Le jeu de cartes est alors modélisé par une pile appelée `jeu` de sommet 1, puis 2 en dessous, *et cætera* jusqu'au bas de la pile qui contient n, comme illustré sur la figure ci-dessous.

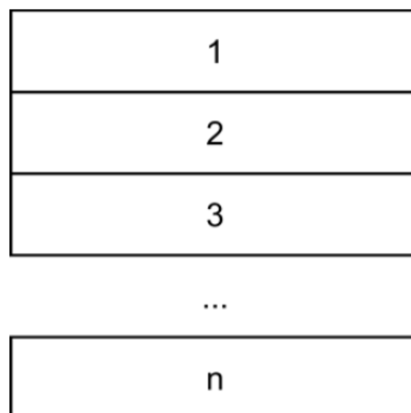


Figure 1. Pile représentant un jeu de cartes

Le mélange faro est réalisé ainsi :

- **Étape 1** : on dépile la moitié de `jeu` et chaque élément dépilé est empilé dans une deuxième pile appelée `moitie1` ;
- **Étape 2** : on dépile le reste de `jeu` et chaque élément dépilé est empilé dans une troisième pile appelée `moitie2` ;
- **Étape 3** : on empile alternativement dans `jeu` et dans cet ordre un élément de `moitie1` puis un élément de `moitie2` jusqu'à vider ces 2 piles.

Dans l'exemple suivant les contenus initiaux de `jeu`, `moitie1` et `moitie2` sont représentés ci-dessous :

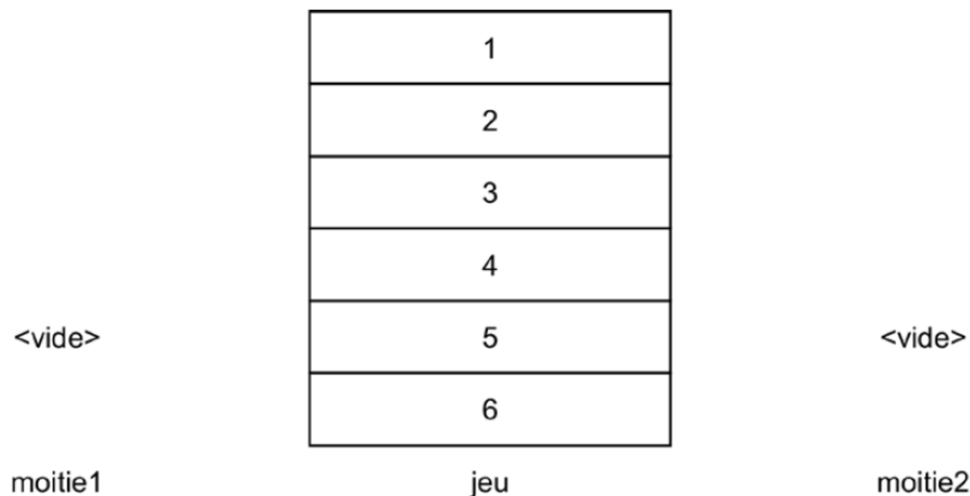


Figure 2. Contenus initiaux des 3 piles

1. Représenter sur votre copie les contenus de ces trois piles à la fin de chaque étape du mélange faro.

Voici le code de la fonction `produire_jeu` qui prend en paramètre un entier `n` supposé pair et qui renvoie une instance de la classe `Pile` qui représente le jeu de cartes.

```
1 def produire_jeu(n):
2     resultat = Pile()
3     for i in range(...):
4         resultat.empile(...)
5     return resultat
```

2. Recopier et compléter sur votre copie le code de la fonction `produire_jeu`.

Ci-après figure le code de la fonction `scinder_jeu` qui prend en paramètres une instance de taille paire de la classe `Pile` qui est le jeu que l'on veut partager en 2 moitiés, un entier `n` qui est la taille de la pile et qui renvoie deux piles qui sont les deux moitiés du jeu.

```
1 def scinder_jeu(p, n):
2     m1 = Pile()
3     m2 = Pile
4     for i in range(n):
5         m1.empile(p.depile())
6     for i in range(n):
7         m2.empile(p.depile())
8     return m1, m2
```

3. Ce code comporte des erreurs. Indiquer les numéros de lignes à rectifier ainsi que les rectifications à apporter.

4. Écrire une fonction `recombiner` qui prend en paramètres deux instances `m1` et `m2` de la classe `Pile` qui sont respectivement la première et la deuxième moitié d'un jeu de cartes et qui renvoie une instance de la classe `Pile` qui est le jeu obtenu en y empilant alternativement et dans cet ordre les éléments de `m1` et de `m2`.

5. Écrire une fonction `faro` qui prend en paramètres une instance de la classe `Pile` qui est le jeu que l'on veut mélanger, un entier `n` qui est la taille de la pile et qui renvoie une instance de la classe `Pile` qui contient le jeu obtenu en appliquant le mélange faro.

Une propriété mathématique assure qu'étant donné un jeu de n cartes (n pair), en répétant suffisamment de fois le mélange faro, on finira par remettre le jeu dans l'ordre initial. On aimerait trouver, pour un entier n donné, ce nombre minimal de répétitions nécessaires. Pour cela, on considère une fonction `identiques` qui prend en arguments deux instances de la classe `Pile` et qui renvoie un booléen indiquant si ces deux piles ont les mêmes éléments, en même nombre et dans le même ordre.

La fonction `identiques` ne modifie pas les piles données en entrée.

Pour s'assurer que la fonction `identiques` fonctionne correctement, on a commencé à produire un jeu de tests :

```
1 p1 = Pile()
2 p1.empile(1)
3 p2 = Pile()
4 assert not identiques(p1, p2)
```

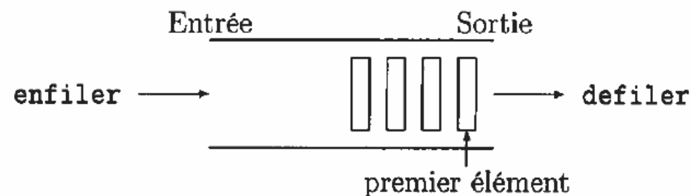
6. Compléter ce jeu de tests pour s'assurer que l'on couvre les cas suivants : les piles sont différentes, mais de même taille ; les piles sont identiques.

7. Écrire une fonction `ordre_faro` qui prend en paramètres un entier `n` pair et qui renvoie le plus petit nombre de répétitions du mélange faro pour qu'un jeu de n cartes soit remis dans son ordre initial.

Exercice 3 *(7 points)*

L'objectif de cet exercice est de travailler sur les températures relevées par une station météorologique. Les données sont enregistrées une fois par jour, à la même heure, et traitées dans l'ordre dans lequel elles arrivent.

On choisit d'utiliser une file : une file est une structure de données abstraite fondée sur le principe « premier arrivé, premier sorti ».



On munit la structure de données `File` des quatre fonctions primitives définies ci-dessous :

Structure de données abstraite : File
Utilise : Élément, Booléen

Opérations :
• `creer_file_vide : ∅ ⟶ File`
`creer_file_vide()` renvoie une file vide
• `est_vide : File ⟶ Booléen`
`est_vide(F)` renvoie `True` si la file `F` est vide, `False` sinon
• `enfiler : File, Élément ⟶ ∅`
`enfiler(F, element)` ajoute `element` en entrée de la file `F`
• `defiler : File ⟶ Élément`
`defiler(F)` renvoie l'élément en sortie de la file `F` (premier élément) en le retirant de la file `F`

1. Les températures relevées ont été 15, puis 17, puis 14.

   (a) Parmi les quatre propositions suivantes, indiquer celle qui représente correctement cette file :

	Entrée — Sortie	
Proposition 1 :	15 17 14	Le premier élément est 14
Proposition 2 :	14 17 15	Le premier élément est 15
Proposition 3 :	15 17 14	Le premier élément est 15
Proposition 4 :	14 17 15	Le premier élément est 14

   (b) En utilisant les fonctions primitives précédentes, donner les instructions permettant de créer cette file.

2. On appelle longueur d'une file le nombre d'éléments qu'elle contient.
   La fonction `longueur_file` prend en paramètre une file `F` et renvoie sa longueur `n`.
   Après appel de cette fonction, la file `F` doit avoir retrouvé son état d'origine.
   Exemple :
   Si F = | 10 | 10 | 12 | 12 | alors `longueur_file(F)` vaut 4.
   Recopier et compléter le programme Python suivant, implémentant la fonction `longueur_file`.
   Dans le code de la fonction, les trois points (...) peuvent correspondre à une ou plusieurs lignes de programme.

```
1 def longueur_file(F):
2     """File -> Int"""
3     G = creer_file_vide() # file temporaire
4     n = 0 # initialisation du nombre d'elements
5     while not(est_vide(F)):
6         ...
7     while not(est_vide(G)): # reconstruction de la file initiale
8         ...
9     return ...
```

3. On s'intéresse à la variation de température d'un jour sur l'autre.
   Par exemple, lorsque les températures relevées sont dans l'ordre d'arrivée 15, 17 et 14, les variations sont 2 et −3.
   Recopier et compléter le programme Python implémentant la fonction `variations` qui prend en paramètre une file non vide `F` et qui renvoie le tableau `tab` contenant les variations successives, ou un tableau vide si la file `F` ne contient qu'une seule température. Il n'est pas demandé ici que la file `F` retrouve son état d'origine après appel de la fonction `variations`.
   Exemple : si `F` est la file qui contient dans l'ordre des relevés les valeurs 15, 17 et 14, `variations(F)` vaut `[2, -3]`.
   Dans le code de la fonction, les trois points (...) peuvent correspondre à une ou plusieurs lignes de programme.

```
1 def variations(F):
2     """File -> Tableau"""
3     taille = longueur_file(F)
4     if taille == 1:
5         ...
6     else:
7         tab = [0 for k in range(taille - 1)]
8         element1 = defiler(F)
9         for i in range (taille - 1):
10            element2 = defiler(F)
11            ...
12    return ...
```

4. Écrire une fonction `nombre_baisses` qui prend en paramètre un tableau `tab`, non vide, des variations des températures et qui renvoie un p-uplet contenant le nombre de jours où la

température a baissé par rapport au jour précédent (soit le nombre de valeurs strictement négatives de `tab`), ainsi que la baisse journalière la plus importante (soit la valeur minimale de `tab`).

S'il n'y a aucune baisse (toutes les valeurs de `tab` sont positives), la fonction renvoie le p-uplet `(0,0)` .

Exemple 1 : `nombre_baisses([1, -4, 2, -1, 3])` vaut `(2,-4)`.

Exemple 2 : `nombre_baisses([1, 5, 3, 1])` vaut `(0,0)`.