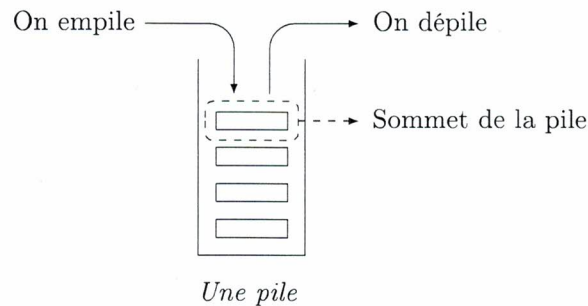


Exercice 1

Cet exercice porte sur la notion de pile et sur la programmation de base en Python.

On rappelle qu'une pile est une structure de données abstraite fondée sur le principe « dernier arrivé, premier sorti » :



On munit la structure de données Pile de quatre fonctions primitives définies dans le tableau ci-dessous. :

Structure de données abstraite : Pile	
Utilise : Éléments, Booléen	
Opérations :	
— <code>creer_pile_vide</code> : $\emptyset \rightarrow \text{Pile}$	<code>creer_pile_vide()</code> renvoie une pile vide
— <code>est_vide</code> : $\text{Pile} \rightarrow \text{Booléen}$	<code>est_vide(pile)</code> renvoie <code>True</code> si <code>pile</code> est vide, <code>False</code> sinon
— <code>empiler</code> : $\text{Pile}, \text{Élément} \rightarrow \text{Rien}$	<code>empiler(pile, element)</code> ajoute <code>element</code> au sommet de la <code>pile</code>
— <code>depiler</code> : $\text{Pile} \rightarrow \text{Élément}$	<code>depiler(pile)</code> renvoie l'élément au sommet de la <code>pile</code> en le retirant de la <code>pile</code>

Question 1 On suppose dans cette question que le contenu de la pile P est le suivant (les éléments étant empilés par le haut) :

4
2
5
8

Quel sera le contenu de la pile Q après exécution de la suite d'instructions suivante ?

```
1   Q = creer_pile_vide()
2   while not est_vide(P):
3       empiler(Q, depiler(P))
```

P → 4, 2, 5, 8

Question 2

- On appelle *hauteur* d'une pile le nombre d'éléments qu'elle contient. La fonction `hauteur_pile` prend en paramètre une pile P et renvoie sa hauteur. Après appel de cette fonction, la pile P doit avoir retrouvé son état d'origine.

Exemple : si P est la pile de la question 1 : `hauteur_pile(P) = 4`.

Recopier et compléter sur votre copie le programme Python suivant implémentant la fonction `hauteur_pile` en remplaçant les ??? par les bonnes instructions.

```

1      def hauteur_pile(P):
2          Q = creer_pile_vide()
3          n = 0
4          while not(est_vide(P)):
5              ??? n += 1
6              x = depiler(P)
7              empiler(Q, x)
8          while not(est_vide(Q)):
9              ??? x = depiler(Q)
10             empiler(P, x)
11         return ??? n

```

- Créer une fonction `max_pile` ayant pour paramètres une pile P et un entier i. Cette fonction renvoie la position j de l'élément maximum parmi les i derniers éléments empilés de la pile P. Après appel de cette fonction, la pile P devra avoir retrouvé son état d'origine. La position du sommet de la pile est 1.

Exemple : si P est la pile de la question 1 : `max_pile(P, 2) = 1`

Question 3 Créer une fonction `retourner` ayant pour paramètres une pile P et un entier j. Cette fonction inverse l'ordre des j derniers éléments empilés et ne renvoie rien. On pourra utiliser deux piles auxiliaires.

Exemple : si P est la pile de la question 1(a), après l'appel de `retourner(P, 3)`, l'état de la pile P sera :

def retourner(P, j):

```

    Q = creer_pile_vide()
    R = creer_pile_vide()
    for i in range(j):
        x = depiler(P)
        empiler(Q, x)
    for i in range(j):
        x = depiler(Q)
        empiler(R, x)
    for i in range(j):
        x = depiler(R)
        empiler(P, x)

```

5
2
4
8

Question 4 L'objectif de cette question est de trier une pile de crêpes.

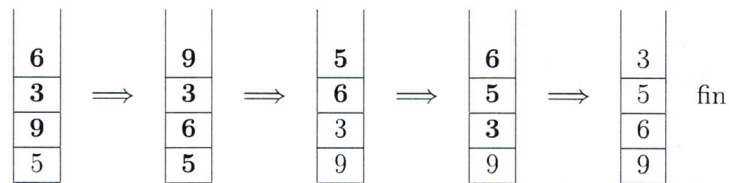
On modélise une pile de crêpes par une pile d'entiers représentant le diamètre de chaque crêpe. On souhaite réordonner les crêpes de la plus grande (placée en bas de la pile) à la plus petite (placée en haut de la pile).

On dispose uniquement d'une spatule que l'on peut insérer dans la pile de crêpes de façon à retourner l'ensemble des crêpes qui lui sont au-dessus.

Le principe est le suivant :

- On recherche la plus grande crêpe.
- On retourne la pile à partir de cette crêpe de façon à mettre cette plus grande crêpe tout en haut de la pile.
- On retourne l'ensemble de la pile de façon à ce que cette plus grande crêpe se retrouve tout en bas.
- La plus grande crêpe étant à sa place, on recommence le principe avec le reste de la pile.

Exemple :



Créer la fonction `tri_crepes` ayant pour paramètre une pile P. Cette fonction trie la pile P selon la méthode du tri crêpes et ne renvoie rien. On utilisera les fonctions créées dans les questions précédentes.

Exemple : Si la pile P est

7
14
12
5
8

, après l'appel de `tri_crepes(P)`, la pile P devient

5
7
8
12
14

.

EXERCICE 3 (8 points)

Cet exercice porte sur les bases de numération, la structure de données PILE et la POO.

La civilisation *Maya* est une ancienne civilisation de Mésopotamie principalement connue pour ses avancées dans les domaines de l'écriture, de l'art, de l'architecture, de l'agriculture, des mathématiques et de l'astronomie.

La numération *Maya* est une numération positionnelle de base 20 (dite vigésimale) utilisant trois symboles pour former les "chiffres" :

- une coquille pour le zéro ,
- un point pour l'unité ●,
- un trait pour la valeur 5 .

Les "chiffres" sont les suivants. Ils utilisent une numération additive :

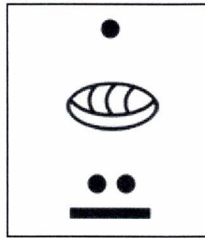
0	1	2	3	4
	●	● ●	● ● ●	● ● ● ●
5	6	7	8	9
	● 	● ● 	● ● ● 	● ● ● ● 
10	11	12	13	14
	● 	● ● 	● ● ● 	● ● ● ● 
15	16	17	18	19
	● 	● ● 	● ● ● 	● ● ● ● 

Figure 1. Table des "chiffres" et valeurs correspondantes (source : Wikipédia)

Dans une version simplifiée de ce système, l'écriture d'un nombre se fait par empilement de "chiffres". Chaque étage correspond à un chiffre de poids 20 fois supérieur au poids du chiffre de l'étage inférieur.

Ainsi la valeur du chiffre de l'étage le plus bas est multipliée par 20^0 soit 1, du second étage par 20^1 , du troisième étage par 20^2 , et ainsi de suite.

Exemple : la représentation *Maya* de l'entier 407 est la suivante.



1. Compléter le tableau donné en **annexe à rendre avec la copie**.
2. Justifier que l'écriture *Maya* de l'entier 3435 est :

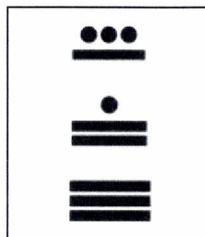


Figure 2. Ecriture *Maya* de l'entier 3435

On **modélise l'écriture d'un entier** dans sa représentation *Maya* par une pile formée de listes. Chacune de ces listes est composée de trois entiers et modélise le "chiffre" d'un étage :

- le premier entier vaut 0 ou 1 suivant s'il y a ou non une coquille ;
- le deuxième représente le nombre de points ;
- le troisième représente le nombre de traits.

Ainsi, la modélisation *Maya* de l'entier 3435 est $[[0, 0, 3], [0, 1, 2], [0, 3, 1]]$ et celle de l'entier 407 est $[[0, 2, 1], [1, 0, 0], [0, 1, 0]]$.

Le sommet de la pile se situe en fin de liste.

On dispose de la classe suivante :

```

1  class Maya:
2      def __init__(self):
3          self.nombre = []
4
5      def ajouter(self, chiffre):
6          """ chiffre est une liste de longueur 3.
7          La méthode empile le chiffre au sommet de la pile """
8          self.nombre.append(chiffre)
9
10     def retirer(self):
11         """ depile et renvoie le chiffre qui etait au sommet de
12
13             la pile """
14         if not self.estVide():
15             return self.nombre.pop()
16
17     def estVide(self):
18         return self.nombre == []
19
20     def nbEtages(self) :
21         """ renvoie le nombre de chiffres de la pile """
22         ...
23
24     def MayaToDec(self):
25         """ renvoie le nombre entier correspondant a la
26         modelisation Maya de l'instance courante """
27         coeff = 20**...
28         ch_Dec = 0
29         while ... :
30             ch_Maya = ...
31             ch_Dec = ch_Dec + (valeurChiffre(ch_Maya)) * coeff
32             coeff = ...
33         return ch_Dec
34
35     def multiplie(self):
36         """ renvoie le resultat de la multiplication par 20 d'un
37         nombre en modelisation Maya. """
38         ...
39
40     def somme(self, maya2):
41         """ ajoute maya2 à l'instance courante et renvoie le
42         resultat en modelisation Maya """
43         if self.nbEtages() == maya2.nbEtages()
44         ...

```

3. Écrire une suite d'instructions permettant de créer une instance, nommée *M*, de la classe *Maya* qui modélise le nombre entier 3435.
4. Écrire la méthode *nbEtages* de la classe *Maya*. Celle-ci renvoie le nombre de "chiffres" utilisés pour écrire le nombre correspondant en écriture *Maya*.

De l'écriture *Maya* à l'écriture décimale

5. Écrire une fonction `valeurChiffre` ayant pour paramètre une liste `L`. Celle-ci renvoie la valeur de l'entier associé à la liste `L = [c, p, t]` où `c` (de valeur 0 ou 1) indique la présence d'une coquille, `p` est le nombre de points et `t` le nombre de traits composant un "chiffre" *Maya*.

Exemple :

```
>>> valeurChiffre([0, 2, 3])
>>> 17
>>> valeurChiffre([1, 0, 0])
>>> 0
```

6. Recopier et compléter les lignes 2, 4, 5 et 7 de la méthode `MayaToDec` suivante de la classe `Maya`. Cette méthode renvoie la valeur de l'entier associé à l'objet `Maya`. On pourra utiliser les méthodes `estVide`, `nbEtages` et `retirer`.

```
1 def MayaToDec(self):
2     coeff = 20**...
3     ch_Dec = 0
4     while ... :
5         ch_Maya = ...
6         ch_Dec = ch_Dec + (valeurChiffre(ch_Maya)) * coeff
7         coeff = ...
8     return ch_Dec
```

De l'écriture décimale vers sa modélisation *Maya*

On considère que la fonction `DecToVige` est déjà écrite. Celle-ci prend en paramètre un entier `n` et renvoie la décomposition en base 20 de celui-ci sous la forme d'une liste $[a_0, a_1, \dots, a_p]$ telle que :

$$n = a_0 \times 20^0 + a_1 \times 20^1 + a_2 \times 20^2 + \dots + a_p \times 20^p$$

Exemple :

```
>>> DecToVige(3435)
[15, 11, 8]
>>> DecToVige(407)
[7, 0, 1]
```

7. Écrire la fonction `decompChiffre` qui prend en paramètre un entier `n` compris entre 0 et 19 et renvoie la liste `[c, p, t]` où `c` vaut 0 ou 1 et indique la présence ou non d'une coquille, `p` est le nombre de points et `t` le nombre de traits composant le "chiffre" *Maya* correspondant.

Exemple :

```
>>> decompChiffre(17)
[0, 2, 3]
```

```
>>>decompChiffre(0)
[1, 0, 0]
```

8. Écrire la fonction `DecToMaya` qui prend en paramètre un entier `n` et renvoie la modélisation *Maya* d'un objet `M` de la classe `Maya` correspondant.

Exemple :

```
>>>DecToMaya(3435).nombre
[[0, 0, 3], [0, 1, 2], [0, 3, 1]]
>>>DecToMaya(407).nombre
[[0, 2, 1], [1, 0, 0], [0, 1, 0]]
```

Opérations sur les nombres en modélisation *Maya*

On souhaite additionner des nombres directement à partir de leur modélisation *Maya*.

9. Écrire la méthode `multiplie` de la classe `Maya` qui renvoie le résultat de la multiplication par 20 d'un nombre en modélisation *Maya*.

Exemple :

```
>>> M = Maya()
>>> M.ajouter([0, 0, 3])
>>> M.ajouter([0, 1, 2])
>>> M.multiplie().nombre
[[1, 0, 0], [0, 0, 3], [0, 1, 2]]
```

On donne la fonction `mystere` suivante :

```
1 def mystere(m1, m2, ret):
2     c = 0
3     p = (m1[1] + m2[1] + ret)%5
4     if m1[1] + m2[1] + ret >= 5:
5         ret = 1
6     else:
7         ret = 0
8     t = (m1[2] + m2[2] + ret)%4
9     if m1[2] + m2[2] + ret < 4:
10        ret = 0
11    else:
12        ret = 1
13    if (m1[0] == 0 and m2[0] == 1) or (p + t = 0 and ret == 1):
14        c = 1
15    return ([c, p, t], ret)
```

10. Donner les résultats renvoyés par les deux appels suivants :

- `mystere([0, 1, 1], [0, 3, 1], 0)`
- `mystere([0, 1, 1], [0, 4, 2], 0)`

11. Écrire une méthode `somme` de la classe `Maya` permettant d'ajouter à l'instance courante un autre nombre `maya2` de même taille en modélisation *Maya*.

ANNEXE À RENDRE AVEC LA COPIE

Exercice 3 – Question 1.

Étage	Écriture <i>Maya</i>	Valeur du "chiffre" de l'étage	Valeur dans la conversion
3		$1 \times 5 + 3 \times 1 = 8$	$8 \times 20^2 = 3200$
2			
1			

EXERCICE 1 (6 points)

Cet exercice porte sur la programmation Python, la programmation orientée objet, les structures de données (file), l'ordonnancement et l'interblocage.

On s'intéresse aux processus et à leur ordonnancement au sein d'un système d'exploitation. On considère ici qu'on utilise un monoprocesseur.

1. Citer les trois états dans lesquels un processus peut se trouver.

On veut simuler cet ordonnancement avec des objets. Pour ce faire, on dispose déjà de la classe `Processus` dont voici la documentation :

Classe `Processus`:

```
p = Processus(nom: str, duree: int)
    Crée un processus de nom <nom> et de durée <duree> (exprimée en
    cycles d'ordonnancement)

p.execute_un_cycle()
    Exécute le processus donné pendant un cycle.

p.est_fini()
    Renvoie True si le processus est terminé, False sinon.
```

Pour simplifier, on ne s'intéresse pas aux ressources qu'un processus pourrait acquérir ou libérer.

2. Citer les deux seuls états possibles pour un processus dans ce contexte.

Pour mettre en place l'ordonnancement, on décide d'utiliser une file, instance de la classe `File` ci-dessous.

Classe `File`

```
1 class File:
2     def __init__(self):
3         """ Crée une file vide """
4         self.contenu = []
5
6     def enqueue(self, element):
7         """ Enfile element dans la file """
8         self.contenu.append(element)
9
10    def dequeue(self):
11        """ Renvoie le premier élément de la file et l'enlève de
12        la file """
13        return self.contenu.pop(0)
14
15    def est_vide(self):
```

```

15         """ Renvoie True si la file est vide, False sinon """
16         return self.contenu == []

```

Lors de la phase de tests, on se rend compte que le code suivant produit une erreur :

```

1 f = File()
2 print(f.defile())

```

3. Rectifier sur votre copie le code de la classe `File` pour que la fonction `defile` renvoie `None` lorsque la file est vide.

On se propose d'ordonnancer les processus avec une méthode du type *tourniquet* telle qu'à chaque cycle :

- si un nouveau processus est créé, il est mis dans la file d'attente ;
- ensuite, on défile un processus de la file d'attente et on l'exécute pendant un cycle ;
- si le processus exécuté n'est pas terminé, on le replace dans la file.

Par exemple, avec les processus suivants

Liste des processus		
processus	cycle de création	durée en cycles
A	2	3
B	1	4
C	4	3
D	0	5

On obtient le chronogramme ci-dessous :

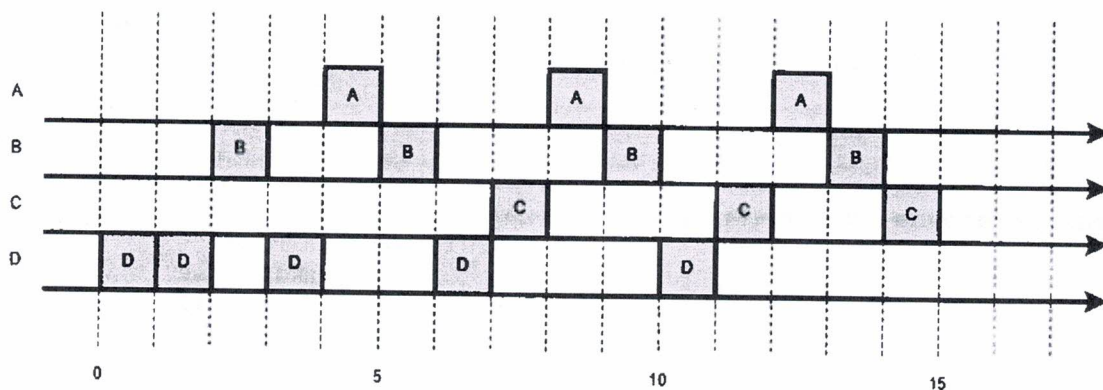


Figure 1. Chronogramme pour les processus A, B, C et D

Pour décrire les processus et le moment de leur création, on utilise le code suivant, dans lequel `depart_proc` associe à un cycle donné le processus qui sera créé à ce moment :

```

1 p1 = Processus("p1", 4)
2 p2 = Processus("p2", 3)
3 p3 = Processus("p3", 5)
4 p4 = Processus("p4", 3)
5 depart_proc = {0: p1, 1: p3, 2: p2, 3: p4}

```

Il s'agit d'une modélisation de la situation précédente où un seul processus peut être créé lors d'un cycle donné.

4. Recopier et compléter sur votre copie le chronogramme ci-dessous pour les processus p1, p2, p3 et p4.

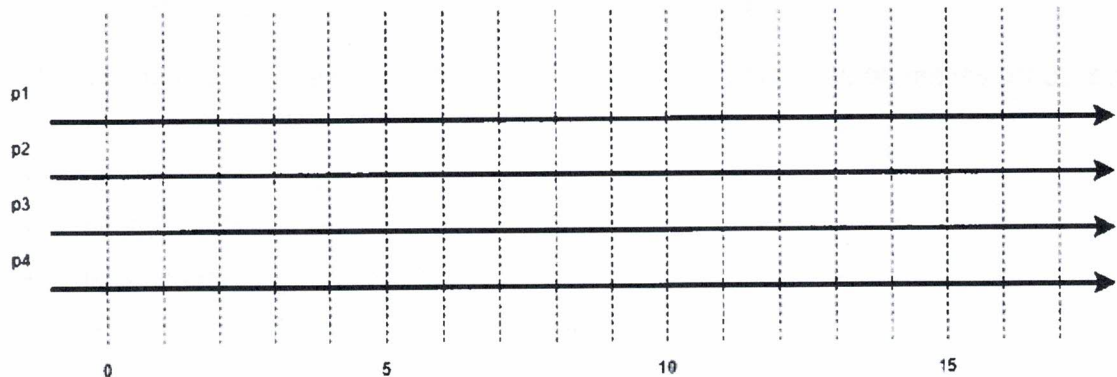


Figure 2. Chronogramme pour les processus p1, p2, p3 et p4

Pour mettre en place l'ordonnancement suivant cette méthode, on écrit la classe `Ordonnanceur` dont voici un code incomplet (l'attribut `temps` correspond au cycle en cours) :

```

1 class Ordonnanceur:
2
3     def __init__(self):
4         self.temps = 0
5         self.file = File()
6
7     def ajoute_nouveau_processus(self, proc):
8         '''Ajoute un nouveau processus dans la file de
9         l'ordonnanceur. '''
10        ...
11
12    def tourniquet(self):
13        '''Effectue une étape d'ordonnancement et renvoie le nom
14        du processus élu.'''
15        self.temps += 1
16        if not self.file.est_vide():
17            proc = ...
18            ...
19            if not proc.est_fini():
20                ...
21            return proc.nom
22        else:
23            return None

```

5. Compléter le code ci-dessus.

À chaque appel de la méthode `tourniquet`, celle-ci renvoie soit le nom du processus qui a été élu, soit `None` si elle n'a pas trouvé de processus en cours.

6. Écrire un programme qui :

- utilise les variables `p1`, `p2`, `p3`, `p4` et `depart_proc` définies précédemment ;
- crée un ordonnanceur ;
- ajoute un nouveau processus à l'ordonnanceur lorsque c'est le moment ;
- affiche le processus choisi par l'ordonnanceur ;
- s'arrête lorsqu'il n'y a plus de processus à exécuter.

Dans la situation donnée en exemple (voir Figure 1), il s'avère qu'en fait les processus utilisent des ressources comme :

- un fichier commun aux processus ;
- le clavier de l'ordinateur ;
- le processeur graphique (GPU) ;
- le port 25000 de la connexion Internet.

Voici le détail de ce que fait chaque processus :

Liste des processus			
A	B	C	D
acquérir le GPU	acquérir le clavier	acquérir le port	acquérir le fichier
faire des calculs	acquérir le fichier	faire des calculs	faire des calculs
libérer le GPU	libérer le clavier	libérer le port	acquérir le clavier
	libérer le fichier		libérer le clavier
			libérer le fichier

7. Montrer que l'ordre d'exécution donné en exemple aboutit à une situation d'interblocage.