

Terminale - Spécialité NSI

Modules pile et file

Dans un premier temps, nous allons créer les différentes fonctions qui permettront de simuler le fonctionnement d'une pile et d'une file à partir d'une liste, en python.

À travers ces exercices, vous serez amené à créer plusieurs **modules** (ensemble de code regroupé dans un fichier qui peut-être importé dans un programme ensuite)

1 - Pile *

Pour cette suite d'exercices, créer un fichier **"pile.py"** dans lequel vous placerez et testerez vos fonctions.

Exercice 1 : Création d'une pile

Définissez et testez une fonction **creer_pile()** qui crée une pile vide.

```
def creer_pile():  
    """  
    Crée une pile vide  
    """
```

Exercice 2 : Taille de la pile

Définissez et testez une fonction **taille_pile(p)** qui renvoie la taille de la pile p.

```
def taille_pile(p):  
    """  
    Renvoie la taille de la pile  
    """
```

Exercice 3 : Pile vide

Définissez et testez une fonction **pile_vide(p)** qui renvoie True si la pile p est vide et False sinon.

```
def pile_vide(p):  
    """  
    Renvoie True si la pile est vide et False sinon.  
    """
```

Exercice 4 : Empilement

Définissez et testez une fonction **empiler(p, x)** qui empile l'élément x sur la pile p.

```
def empiler(p, x):  
    """  
    Empile l'élément x sur la pile p  
    """
```

Exercice 5 : Dépilement

Définissez et testez une fonction **depiler(p)** qui dépile (retire et renvoie de la pile p) le sommet de la pile p. (si la pile n'est pas vide)

```
def depiler(p):  
    """  
    Retire et renvoie le sommet de la pile p si elle n'est pas vide.  
    """
```

Exercice 6 : Sommet de la pile

Définissez et testez une fonction **sommet__pile(p)** qui renvoie le sommet de la pile p sans le dépiler.

```
def sommet_pile(p):  
    """  
    Renvoie le sommet de la pile p sans le dépiler.  
    """
```

Test de votre structure pile

A l'aide de la **console** testez votre structure. Par exemple :

```
>>> pile_test = creer_pile()  
>>> taille_pile(pile_test)  
0  
>>> pile_vide(pile_test)  
True
```

```

>>> empiler(pile_test, 'A')
>>> empiler(pile_test, 'B')
>>> empiler(pile_test, 'C')
>>> empiler(pile_test, 'D')
>>> taille_pile(pile_test)
4
>>> pile_vide(pile_test)
False
>>> sommet_pile(pile_test)
'D'
>>> depiler(pile_test)
'D'
>>> sommet_pile(pile_test)
'C'
>>> taille_pile(pile_test)
3

```

2- File *

Pour cette suite d'exercices, créer un fichier **"file.py"** dans lequel vous placerez et testerez vos fonctions.

Exercice 1 : Création d'une file

Définissez et testez une fonction **creer_file()** qui crée une file vide.

```

def creer_file():
    """
    Crée une file vide
    """

```

Exercice 2 : Taille de la file

Définissez et testez une fonction **taille_file(f)** qui renvoie la taille de la file f.

```

def taille_file(f):
    """
    Renvoie la taille de la file
    """

```

Exercice 3 : File vide

Définissez et testez une fonction **file_vide(f)** qui renvoie True si la file f est vide et False sinon.

```

def file_vide(f):
    """

```

```
Renvoie True si la file est vide et False sinon.
"""
```

Exercice 4 : Ajout d'un élément à la file

Définissez et testez une fonction `ajouter_file(f, x)` qui ajoute un élément `x` en bout de file.

```
def ajouter_file(f,x):
    """
    Ajoute l'élément x à la file
    """
```

Exercice 5 : Retrait de l'élément en bout de file

Définissez et testez une fonction `retirer_file(f)` qui retire et renvoie l'élément situé en bout de file. (par exemple, dans une file d'attente d'un supermarché, le prochain client à faire passer à la caisse)

```
def retirer_file(f):
    """
    Retire et renvoie l'élément situé en bout de file
    """
```

Exercice 6 : Élément en bout de file

Définissez et testez une fonction `premier_file(f)` qui renvoie l'élément situé en bout de file sans le retirer de la file.

```
def premier_file(f):
    """
    Renvoie l'élément situé en bout de file
    """
```

Test de votre structure file

A l'aide de la **console** testez votre structure. Par exemple :

```
>>> file_test = creer_file()
>>> taille_file(file_test)
0
>>> file_vide(file_test)
True
>>> ajouter_file(file_test, 'A')
>>> ajouter_file(file_test, 'B')
>>> ajouter_file(file_test, 'C')
>>> ajouter_file(file_test, 'D')
>>> taille_file(file_test)
```

```

4
>>> file_vide(file_test)
False
>>> premier_file(file_test)
'A'
>>> retirer_file(file_test)
'A'
>>> premier_file(file_test)
'B'
>>> taille_file(file_test)
3

```

Exercices sur les piles

Les exercices suivants vont vous demander d'utiliser le module **pile** précédemment créé. Vous pouvez l'importer en début de fichier ainsi :

```
import pile
```

Attention, afin d'utiliser les fonctions de ce module, vous devez écrire "**pile.nom_de_la_fonction**", par exemple, "**pile.creer_pile()**"

Navigateur internet

Dans la plupart des navigateurs web, lors de la navigation à travers différentes pages, l'utilisateur a la possibilité de revenir en arrière, vers les pages précédemment consultées (via un bouton).

Lorsque l'utilisateur retourne en arrière, il peut alors également "avancer d'une page" pour revenir à la page où il se trouvait avant de revenir en arrière.

Un tel comportement peut être implémenté grâce aux **piles**.

Exercice 1 - Reculer d'une page **

Afin d'implémenter le comportement du "retour arrière" nous allons utiliser une variable de type pile nommée "**memoire_pages**", fonctionnant de la manière suivante :

1. Le sommet de la pile représente la page actuellement consultée
2. Lorsqu'on consulte un nouveau site, celui-ci est empilé
3. Lorsqu'on retourne en arrière, le site courant est donc dépilé.

Définissez et testez une fonction **consulter_page(page, memoire_pages)** qui simule la consultation d'une nouvelle page et modifie **memoire_pages** en conséquence. La "page" sera représentée par la variable **page**, une chaîne de caractère (par exemple l'adresse d'un site web).

```
def consulter_page(page, memoire_pages):
    """
    Entrées :
        - page : Une chaîne de caractère (l'adresse de la page consultée)
        - memoire_pages : Une pile stockant les pages consultées

    Simule la consultation d'une nouvelle page web
    """
```

Définissez et testez une fonction `page_courante(memoire_pages)` qui renvoie la page actuellement consultée.

```
def page_courante(memoire_pages):
    """
    Entrée :
        - memoire_pages : Une pile stockant les pages consultées

    Renvoie la page actuellement consultée.
    """
```

Définissez et testez une fonction `reculer_page(memoire_pages)` qui simule la fonctionnalité de recul d'une page en modifiant `memoire_pages` en conséquence.

```
def reculer_page(memoire_pages):
    """
    Entrée :
        - memoire_pages : Une pile stockant les pages consultées

    Simule la fonctionnalité de retour en arrière.
    """
```

Test de la fonction

```
>>> memoire_pages = pile.creer_pile()
>>> consulter_page("https://google.fr", memoire_pages)
>>> page_courante(memoire_pages)
"https://google.fr"
>>> consulter_page("https://pixees.fr", memoire_pages)
>>> consulter_page("https://twitter.com/home", memoire_pages)
>>> page_courante(memoire_pages)
"https://twitter.com/home"
>>> reculer_page(memoire_pages)
>>> page_courante(memoire_pages)
"https://pixees.fr"
```

Exercice 2 - Avancer d'une page ***

Pour implémenter la fonctionnalité permettant d'avancer d'une page après avoir reculé, nous aurons besoin d'une seconde pile (que nous appellerons "**memoire_bis**") fonctionnant de la manière suivante :

1. Le sommet de la pile représente la page d'où venait l'utilisateur avant de faire un retour arrière.
2. Lorsqu'on consulte un nouveau site, cette pile est vidée (réinitialisée).
3. Lorsqu'on avance d'une page, le sommet de la pile est dépilé pour être empilé dans la première pile (**memoire_pages**)

Tout d'abord, **ajoutez une procédure (fonction) "reinitialiser_pile(p)"** à votre module **pile**, qui aura pour effet de vider le contenu de la liste.

```
def reinitialiser_pile(p):  
    """  
    Entrée :  
    - p : Une pile  
    Supprime tout le contenu de la pile (vide la liste)  
    """
```

Ensuite, **modifiez et testez** les fonctions **consulter_page** et **reculer_page** afin d'accueillir un nouveau paramètre **memoire_bis** de type pile qui sera modifié en conséquence lors de l'exécution de ces fonctions.

```
def consulter_page(page, memoire_pages, memoire_bis):  
    """  
    Entrées :  
    - page : Une chaîne de caractère (l'adresse de la page consultée)  
    - memoire_pages : Une pile stockant les pages consultées  
    - memoire_bis : Une seconde pile mémoire pour les pages  
  
    Simule la consultation d'une nouvelle page web  
    """  
  
def reculer_page(memoire_pages, memoire_bis):  
    """  
    Entrée :  
    - memoire_pages : Une pile stockant les pages consultées  
    - memoire_bis : Une seconde pile mémoire pour les pages  
  
    Simule la fonctionnalité de retour en arrière.  
    """
```

Définissez et testez une fonction **avancer_page(memoire_pages, memoire_bis)** qui simule la fonctionnalité d'avancement d'une page en modifiant **memoire_bis** et **memoire_pages** en conséquence.

```
def avancer_page(memoire_pages, memoire_bis):
    """
    Entrée :
        - memoire_pages : Une pile stockant les pages consultées
        - memoire_bis : Une seconde pile mémoire pour les pages
    Simule la fonctionnalité "Avancer d'une page"
    """
```

Test de la fonction

```
>>> memoire_pages = pile.creer_pile()
>>> memoire_bis = pile.creer_pile()
>>> consulter_page("https://google.fr", memoire_pages, memoire_bis)
>>> page_courante(memoire_pages)
"https://google.fr"
>>> consulter_page("https://pixees.fr", memoire_pages, memoire_bis)
>>> consulter_page("https://twitter.com/home", memoire_pages, memoire_bis)
>>> page_courante(memoire_pages)
"https://twitter.com/home"
>>> reculer_page(memoire_pages, memoire_bis)
>>> page_courante(memoire_pages)
"https://pixees.fr"
>>> avancer_page(memoire_pages, memoire_bis)
>>> page_courante(memoire_pages)
"https://twitter.com/home"
>>> avancer_page(memoire_pages, memoire_bis)
>>> page_courante(memoire_pages)
"https://twitter.com/home"
>>> reculer_page(memoire_pages, memoire_bis)
>>> consulter_page("https://youtube.com", memoire_pages, memoire_bis)
>>> reculer_page(memoire_pages, memoire_bis)
>>> page_courante(memoire_pages)
"https://pixees.fr"
>>> avancer_page(memoire_pages, memoire_bis)
>>> page_courante(memoire_pages)
"https://youtube.com"
```

Exercices sur les files

Les exercices suivants vont vous demander d'utiliser le module **file** précédemment créé. Vous pouvez l'importer en début de fichier ainsi :

```
import file
```

Attention, afin d'utiliser les fonctions de ce module, vous devez écrire "**file.nom_de_la_fonction**", par exemple, "**file.creer_file()**"

Le jeu de la bataille

Voici la description du jeu de la bataille, adapté aux besoins de l'exercice :

- Chaque joueur possède un paquet de 52 cartes, numérotées de 1 à 13 (nous ignorerons les couleurs pour simplifier l'exercice)
- A chaque tour, les deux joueurs révèlent la carte au sommet de leur paquet.
- Le joueur possédant la carte avec la plus grande valeur récupère les deux cartes et les placent sous son paquet.
- Si les deux valeurs sont égales, les deux cartes sont "perdues", personne ne les récupère.
- La partie se termine après un nombre de tours défini, ou bien quand un des deux paquets est vide.
- Le gagnant est le joueur avec le plus de cartes dans son paquet.

A travers cette suite d'exercices, nous allons créer une simulation du jeu de la bataille (adapté). Pour commencer, créez un fichier **bataille.py** et importez le module file.

Exercice 1 : Génération aléatoire d'un paquet *

Une carte est représenté par une valeur (entre 1 et 13). On aura donc quatre fois la même valeur dans un paquet. Un paquet de carte est une file contenant les 52 cartes possibles dans un ordre quelconque.

Reprenez, complétez et testez le code suivant afin d'obtenir une fonction de génération aléatoire d'un paquet de carte.

```
import random
import file #Votre module

def generer_paquet():
    paquet = file.creer_file()
    for ... in ...:
        for ... in ...:
            file.a_completer(paquet, ...)
    random.shuffle(paquet) #Mélange le paquet
    return paquet
```

Vous devriez avoir un rendu similaire dans la console :

```
[4,1,2,5,4,13,12,11,1,2,5,4,...]
```

Exercice 2 : Un tour de jeu ***

Le tour de jeu désigne le moment où les deux joueurs révèlent la première carte de leur paquet (en bout de file) et obtiennent ou perdent la carte en en question.

Définissez et testez une fonction `tour_bataille(paquet1, paquet2)` prenant deux files en entrées (les paquets des deux joueurs) et réalise un tour de jeu. Ces deux paquets seront donc modifiés lors de l'exécution de la fonction.

```
def tour_bataille(paquet1, paquet2):  
    """  
    Entrées : 2 files représentant les paquets de cartes de chaque joueur  
    Exécute un tour du jeu de la bataille, en modifiant les deux paquets.  
    """
```

Un exemple de test en console :

```
p1 = [3,5]  
p2 = [1,9]  
tour_bataille(p1,p2)  
p1  
Out[380]: [5, 3, 1]  
p2  
Out[381]: [9]
```

Exercice 3 : Le jeu **

Il ne reste plus qu'à implémenter la fonction réalisant la boucle de jeu! Pour cela, **définissez et testez** une procédure `jeu_bataille(max_tours)`, paramétré avec une variable `max_tours` indiquant le tour où le jeu se termine, et qui va réaliser les actions suivantes :

- Initialise les deux paquets de cartes pour les deux joueurs.
- Effectue une boucle en réalisant un tour de jeu à chaque itération.
- Arrête le jeu quand on dépasse le nombre de tours fixé par la variable `max_tours` ou bien qu'un des deux paquets est vide.
- Affiche le vainqueur (celui qui a le plus de cartes dans son jeu à la fin) ou bien égalité, ainsi que le nombre de points (nombre de cartes du paquet) de chaque joueur.

```
def jeu_bataille(max_tours):  
    """  
    Entrée : Le nombre de tours de jeu  
    Effectue une partie du jeu de la bataille entre deux joueurs  
    """
```

Un exemple d'affichage :

```
>>> jeu_bataille()  
Gagnant : Joueur 1  
Joueur 1 : 42 points  
Joueur 2 : 36 points
```

```
>>> jeu_bataille()
Egalite
Joueur 1 : 44 points
Joueur 2 : 44 points
```

Bibliothèque et temps d'exécution

Comme nous l'avons vu, les structures permettant de manipuler les piles ou les files ne sont pas natives à python. Dans ce TP, vous avez implémenté ces structures à l'aide d'une liste, mais il existe des méthodes plus efficaces, notamment en terme de complexité, pour implémenter ces structures. Par exemple, avec la programmation objet que nous verrons dans de futures activités. Néanmoins, vous pouvez déjà accéder à ce type d'implémentation à l'aide de bibliothèques! Pour la suite, nous allons donc utiliser le module **deque** issu de la bibliothèque **collections** qui permet à la fois d'implémenter les **piles** et les **files**.

Importez le ainsi :

```
from collections import deque
```

Exercice 1 : Analyse de code *

Analysez , testez (en console) et commentez le code suivant. Le code est constitué de deux blocs. Identifiez lequel correspond à la structure pile et lequel correspond à la structure file puis identifiez les différentes opérations que vous avez codé pour ces deux structures dans la première section de ce TP.

```
#Bloc 1
a = deque()
a.append(5)
a.append(7)
a[0]
a.popleft()
a[0]
```

```
#Bloc 2
b = deque()
b.append(5)
b.append(7)
b[-1]
b.pop()
b[-1]
```

Exercice 2 : Complexité et temps d'exécution *

Cet exercice a pour but de vous montrer que l'implémentation de la pile ou de la file avec une liste n'est pas forcément adéquate. En effet, si on n'observera que très peu, voire pas du tout de différences pour de petits volumes de données, cette implémentation s'avère peu efficace pour de plus gros volumes. Commencez par importer la bibliothèque **time** qui permet de mesurer le temps.

```
import time
```

Recopiez et analysez cette fonction :

```
def comparer_temps_files(taille):
    f1 = file.creer_file()
    for _ in range(taille): file.ajouter_file(f1, 0)
    f2 = deque([0] * taille)

    debut = time.time()
    file.retirer_file(f1)
    fin = time.time()
    t1 = fin - debut

    debut = time.time()
    f2.popleft()
    fin = time.time()
    t2 = fin - debut

    print("Temps file avec liste : " + str(t1))
    print("Temps file avec deque : " + str(t2))
```

Testez d'appeler la fonction avec différentes valeurs, de plus en plus grosses, par exemple :

```
#Certains appels peuvent prendre du temps, c'est normal
comparer_temps_files(1000)
comparer_temps_files(100000)
comparer_temps_files(1000000)
comparer_temps_files(10000000)
comparer_temps_files(100000000)
```

Qu'observez vous ? Qu'en déduisez vous ?

Exercice 3 : Retour sur la bataille **

Pour terminer ce TP, **reprenez le code de la bataille** de la section sur les files dans un nouveau fichier et **remplacez votre file par la structure deque** de la bibliothèque **collections**, comme utilisée lors des précédents exercices.