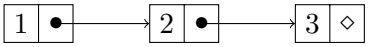
Liste chaînées

Exo 1 : à partir de la classe `ListeChainees` du cours,

a) proposer une fonction `listeN(n : int) -> 'ListeChainee'` qui renvoie une liste chaînée de taille n avec les valeurs de 1 à n.

◇ `listeN(0)` renvoie la liste chaînée vide 

◇ `listeN(3)` renvoie la liste chaînée 

b) proposer une méthode `recherche(val : object) -> bool` qui renvoie `True` si `val` est une valeur de la liste chaînée.

Exo 2 : à partir de la classe `ListeChainees` du cours,

a) définir la méthode `__setitem__(i : int, val : object) -> None`, qui permet d'affecter `val` à la i^è valeur de la liste.

b) définir la méthode `__str__() -> str`, qui permettra d'afficher les valeurs de la liste comme en `python`, autrement dit si `liste = listeN(3)` alors `print( liste )` affichera `[1, 2, 3]`.

rq : puisque la fonction `__getitem__()` a été définie, il est possible de faire une boucle `for`.

Exo 3 :

a) proposer la méthode `__eq__(lc : 'ListeChainee') -> bool`, qui renvoie si deux listes sont égales

b) proposer une méthode `occurence(val : object) -> int`,qui renvoie le nombre d'occurence de `val` dans la liste.

c) proposer une méthode `tri() -> 'ListeChainee'`, qui renvoie une nouvelle liste chaînée triée dans l'ordre croissant (on supposera que la liste chainée ne contiendra alors que des entiers ou des chaînes de caractères).

Exo 4 : proposer une classe `ListeDoubleChaine` qui implémente une chaîne doublement chaînée.

a) définir le constructeur,

b) définir les méthodes `len()`, `__getitem__(i : int) -> object` et `__setitem__(i : int, val : object) -> None`,

c) définir les méthodes `ajout_i(i : int, val : object) -> None` et `supprime_i(i : int) -> None`,

d) définir la méthode `__str__() -> str` qui renverra `"[1]<->[2]<->[3]"` si la liste contien les valeurs $1, 2, 3$.

e) définir la méthode `recherche(val : object) -> bool`.

Exo 5 : on considère le code suivant

```
1 from point import Point
2 from cercle import Cercle
3 p1 = Point(1, 2, 'x')
4 c1 = Cercle(centre = p1, rayon = 1.2, color = (255, 0, 255))
5 print( f"Un cercle de rayon {c1.rayon} a une aire de {c1.aire()}" )
```

a) Que représentent / à quoi réfèrent `point` et `Point` ligne 1 ?

b) Quel est le type des paramètres `centre`, `rayon` et `color` ligne 4 ?

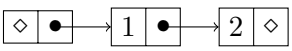
c) Que représentent / à quoi réfèrent `c1.rayon` et `c1.aire()` ligne 5 ?

Exo 6 : proposer une classe `ListeSentinelle` qui propose une liste chaînée avec un maillon sentinelle.

a) définir le constructeur,

b) définir les méthodes `len()`, `__getitem__(i : int) -> object` et `__setitem__(i : int, val : object) -> None`,

c) définir les méthodes `ajout_i(i : int, val : object) -> None` et `supprime_i(i : int) -> None`,

d) définir la méthode `__str__() -> str`, qui renverra `"[1]->[2]"` pour 

e) définir la méthode `recherche(val : object) -> bool`

f) définir la méthode `inversion() -> None`

Piles et files

Exo 7 : Proposer une méthode `vider_pile() -> None`, qui vide la pile.

Exo 8 : Proposer une méthode `vider_file() -> None`, qui vide la file.

Exo 9 : À l'aide des méthodes existentes de la classe 'Pile', proposer une fonction qui

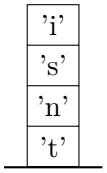
a) dépile k éléments d'une pile (si elle contient moins de k éléments, alors elle sera vidée)

b) inverse toute la pile,

c) inverse le sommet et le pénultième élément de la pile, 

d) inverse le sommet et le premier élément de la pile. 

Exo 10 : Proposer une fonction `chaine2Pile(chaine : str) -> 'Pile'`, qui remplit une pile avec chaque caractères de la chaîne.



expl : `chaine2Pile("tnsi")` renverra la pile ci-contre :

Exo 11 : Proposer une fonction `chaine2File(chaine : str) -> 'File'`, qui remplit une file avec chaque caractères de la chaîne.

expl : `chaine2File("tnsi")` renverra la file ⟶ | 'i' | 's' | 'n' | 't' | ⟶

Que faut-il changer pour avoir `chaine2File("tnsi")` qui renvoie la file ⟶ | 't' | 'n' | 's' | 'i' | ⟶ ?

Exo 12 : Dans certaines applications, il est possible d'annuler une action. Pour cela, toutes les actions sont sauvegardées dans une pile d'actions (nommée `pAction`). Lorsqu'une action est annulée, la pile d'actions est dépiler et l'ancienne action part dans la pile d'annulation (nommée `pAnnul`). À chaque nouvelle action, cette pile d'annulation est vidée.

a) dessiner les états des piles en fonction des actions suivantes :

1- place un point A
2- place un point B
3- trace le segment [AB]
4- annule le tracé
5- place un point C
6- trace le segment [AB]
7- annule le tracé
8- trace le segment [AC]

b) proposer deux fonctions `actionFaite(a : 'action') -> None` et `actionAnnulee() -> None` qui auront les deux piles voulues en tant que variables globales.

c) Souvent il est possible d'annuler l'annulation via une fonction re-faire. Proposer une nouvelle implémentation des deux fonctions précédentes ainsi que de la fonction `redo() -> None`.

Exo 13 : Le code `python` est dit bien parenthésé s'il y a autant de parenthèses ouvrantes que fermantes et écrites dans le bon ordre. Ainsi `print) "tnsi"(` ou bien `print("tnsi"))` sont incorrect tandis que `print("tnsi")` est correct.

On admet que le code `python` sera fourni dans une longue chaîne de caractères dont les parenthèses ne servent qu'à la structure du code (l'utilisateur n'affichera pas de message contenant des parenthèses).

a) Proposer une fonction `bienParenthese(code : str) -> bool`, qui renvoie `True` si le code est bien parenthésé. _On pourra considérer qu'à une parenthèse ouvrante correspond +1_

Tester votre code sur les expressions mathématiques : "((1+2)*4" ; "(1+2)*4" et ")(1+2)*4(" et "(1+2))*4"

b) Modifier le code afin d'utiliser une pile au lieu d'une variable entière.

c) En mathématiques, on peut également écrire des crochets. Ainsi "[(2+3)*4-(2+1)]*6" sera dit bien parenthésé.

Proposer une nouvelle fonction `bienParenthese2(code : str) -> bool` qui répond à la question en utilisant une pile.

Exo 14 : Comment créer une file à l'aide de deux piles ?

File
pEntree ('Pile')
pSortie ('Pile')

a) Proposer une implémentation de votre nouvelle classe `'File'` en définissant le constructeur afin que `File` vérifie

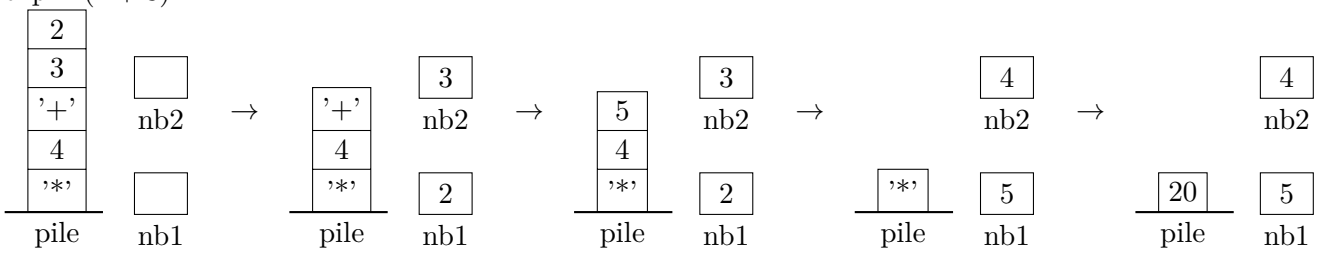
b) définir la méthode `est_vide() -> bool`

c) définir les méthodes `enfiler(e : object) -> None` et `defiler() -> object`

Exo 15 : Dans les années 1 960-1 990, les calculatrices *Hp* proposaient la notation polonaise inverse. En notation polonaise inverse, aucune parenthèse n'est utilisée pour marquer les priorités opératoires mais c'est l'ordre d'entrée qui définit les priorités.

Ainsi la calculatrice lisait dans la pile d'éxécution deux nombres puis un opérateur, elle effectuait l'opération et mettait le résultat dans la pile d'éxécution.

expl : $(2+3)*4$



a) En montrant les états de la pile et des variables `nb1` et `nb2`, calculer les expressions $2-(3+4)$ et $5*(3+2*7)$

b) À partir de l'exo 10, créer une fonction `chaine2PilePolo(chaine : str) -> 'Pile'` pour obtenir une pile d'entiers et d'opérateurs (seuls les quatre opérateurs classiques seront reconnus).

c) Proposer une fonction `poloInverse(p : 'Pile') -> int`, qui renvoie le calcul voulu

d) Proposer un jeu de tests et tester votre fonction `poloInverse()`.

Exo 16 : Dans un supermarché, deux caisses (avec deux files d'attentes) fonctionnent. Soudainement, une caisse ne fonctionne plus. Comment répartir les clients dans l'unique nouvelle file ?

On considère qu'à l'entrée de chaque file, les clients ont un ticket avec leur heure d'arrivée.

a) Proposer une foncion `arrivee(f : 'File') -> None`, qui remplit la file avec un tuple contenant l'heure et le temps pour passer la caisse.

expl : si la personne arrive à 17h01 et mettra 10 min pour passer la caisse, le tuple sera `(17h01, 10)`.

Pour faciliter l'implémentation, l'heure sera donnée par `time.time()` et le temps de passage en caisse sera un nombre aléatoire entre 1 et 20.

Ainsi la file suivante $\longrightarrow \boxed{\binom{1.83}{5}}\boxed{\binom{1.4}{17}}\boxed{\binom{1.21}{3}}\boxed{\binom{0.9}{12}} \longrightarrow$ doit être comprise comme : il y a quatre personnes dans la file. La 1ère personne est arrivée à 0,9 (unité de temps) et mettra 12 min en caisse, la 2n personne est arrivée à 1,21 u.t. et mettra 3 min en caisse, la 3è est arrivée à 1,4 u.t. et mettra 17 min tandis que la 4è est arrivée à 1,83 u.t. et mettra 5 min en caisse.

b) Proposer une fonction `fusion(f1 : 'File', f2 : 'File') -> 'File'`, qui fusionne vos deux files en une selon l'heure d'arrivée des clients.

c) on souhaite simuler l'arrivée de 20 clients aléaoirement, compléter le code suivant :

```
def simul() -> None :
    f1, f2 = ......, ......
    for i in range(...) :
        if 2*random.random() < 1 : # la client choisit 1 fois sur 2 la file 1
            f1.enfiler( ( ......... , random.randint(..., ...) ) )
        else :
            f2.enfiler( ( ......... , random.randint(..., ...) ) )
    ff = fusion(... , ...)
```

d) Proposer une fonction `tempsTotal(f1 : 'File') -> int`, qui renvoie le temps total pour vider la file.

expl : si la file est $\longrightarrow \boxed{\binom{1.83}{5}}\boxed{\binom{1.4}{17}}\boxed{\binom{1.21}{3}}\boxed{\binom{0.9}{12}} \longrightarrow$, la fonction renverra $12+3+17+5=37$.

e) Simuler et vérifier que quasi-toujours le temps total de la file unifiée (`tempsTotal(ff)`) est supérieur au plus grand temps total des deux autres files. Pourquoi ?

Exo 17 : Lire un fichier revient à utiliser une pile contenant toutes les lignes du fichier et où la méthode `depiler()` se nomme `readline()`.

On considère un fichier contenant un ensemble de nombres. Proposer un code qui lit ce fichier et : **a)** fait la somme de tout les nombres ; **b)** en fait la moyenne.

```
myfile = open( "./fichier.txt", 'r')
line = myfile.readline()
while line != "" :
    print( line )
    line = myfile.readline()
myfile.close()
```