TD n° 1 : les piles

Dans tout ce TD, on fera attention à **n'utiliser que les opérations valides sur les piles** (empiler et dépiler) et non les autres opérations utilisables sur les listes en général (par exemple lecture d'un élément autre que celui au sommet de la pile).

Pour faciliter l'écriture des fonctions, on pourra utiliser simplement `p = []` pour créer une pile vide ainsi que `p.append(v)` et `p.pop()` pour respectivement empiler l'élément `v` sur la pile `p` ou dépiler `p`.

Enfin, on pourra à chaque exercice réutiliser si besoin les fonctions des exercices précédents.

1 Quelques fonctions sur les piles

Exercice 1. Écrire une fonction `est_vide` qui

- prend en argument une pile `p` ;
- renvoie le booléen `True` si `p` est vide et `False` sinon.

Exercice 2. Écrire une fonction `observer_sommet` qui

- prend en argument une pile `p` ;
- donne la valeur de l'élément du sommet de la pile `p` ;
- ne modifie au final pas la pile `p`.

Exercice 3. Écrire une fonction `echange` qui

- prend en argument une pile `p` ;
- ne retourne aucune valeur ;
- modifie la pile `p` en échangeant les deux valeurs du dessus.

Exercice 4. Écrire une fonction `renverse` qui

- prend en argument une pile `p` ;
- renvoie une pile dont les éléments sont ceux de `p` dans l'ordre inverse. Par exemple `renverse([1,2,3])` doit renvoyer `[3,2,1]`.
- ★ ne modifie au final pas la pile `p`.

Donner sa complexité en fonction de la hauteur n de la pile `p` donnée en argument.

Exercice 5. Écrire une fonction `taille` qui

- prend en argument une pile `p` ;
- renvoie la hauteur de la pile ;
- n'utilise pas la fonction `len`.
- ★ ne modifie au final pas la pile `p`.

2 Première application : bon parenthésage

On s'intéresse dans cette partie à des chaînes de caractères qui ne contiennent que des parenthèses ouvrantes et fermantes. On dit qu'une telle chaîne est *bien parenthésée* si :

- *condition 1* : il y a autant de parenthèses ouvrantes que de parenthèses fermantes ;
- *condition 2* : à toute parenthèse fermante correspond une parenthèse ouvrante située avant.

Par exemple, les chaînes `"(()())"` et `"()((())())"` sont bien parenthésées alors que les chaînes `"()(()"` et `"())("` ne le sont pas.

Afin de tester si une chaîne est bien parenthésée, on commence par lui associer une pile dont chaque élément est un caractère (soit `"("`, soit `")"`), celui du sommet de la pile étant le premier de la chaîne et ainsi de suite. Par exemple :

`"(()"` $\rightsquigarrow$

`"("`
`"("`
`")"`

Exercice 6. Écrire une fonction suivante `chaine_en_pile` qui prend en argument une chaîne de caractères `ch` et qui renvoie la pile associée à cette chaîne de caractères.

Par exemple `chaine_en_pile("()((")` doit renvoyer `['(', '(', ')', '(']`.

Exercice 7. Compléter le code de la fonction

```
def bien_parenthesee(ch: str) -> bool:
    pile = chaine_en_pile(ch)
    # A completer
```

qui renvoie `True` si la chaîne de caractères `ch` est bien parenthésée et `False` sinon.

Bonus. Pour aller un peu plus loin, on pourra faire l'épreuve 3.11 du challenge.

3 Deuxième application : notation polonaise inverse

Dans toute cette partie, on se limitera aux entiers naturels et aux opérateurs $+$ et $*$.

La *notation polonaise inverse* (NPI) permet d'écrire des opérations arithmétiques sans utiliser de parenthèses. On y écrit les opérateurs après les nombres sur lesquels s'effectuent l'opération, c'est pourquoi on parle aussi de *notation postfixe*.

Par exemple, l'opération que l'on note usuellement « $5 + 7$ » s'écrit en NPI « $5\ 7\ +$ ». De façon plus élaborée, l'opération « $3 * (2 + 9)$ » s'écrit en NPI « $2\ 9 + 3\ *$ ».

De manière générale, il est assez aisé de passer de la notation classique à la NPI : il suffit de respecter l'ordre dans lequel s'effectuent les calculs.

Par exemple, pour effectuer « $((4 + 6) * 7) + 1$ », on effectue les étapes suivantes :

- on commence par additionner 4 et 6 donc on note « $4\ 6\ +$ » ;
- on multiplie ensuite le résultat par 7 donc on note « $4\ 6 + 7\ *$ » ;
- enfin on ajoute 1 d'où « $4\ 6 + 7 * 1\ +$ ».

Comme on effectue les opérations dans l'ordre d'affichage, la structure de pile est bien adaptée à l'évaluation d'expressions en NPI. Par exemple, la pile associée au calcul précédent est donnée ci-contre.

4
6
`'+'`
7
`'*'`
1
`'+'`

Remarque. Il serait plus naturel de considérer la pile dans l'ordre inverse afin de correspondre à l'ordre des étapes décrites précédemment (qui est aussi celui des frappes sur les touches d'une calculatrice ou d'un ordinateur) et de commencer le traitement en la renversant (cf exercice 4).

Exercice 8. Écrire une fonction `evaluation_npi` qui prend en argument une pile et qui renvoie la valeur de l'opération arithmétique associée. Par exemple `evaluation_npi(['+', 1, '*', 7, '+', 6, 4])` renverra `71`.

Bonus. Faire l'épreuve 4.13 du challenge.