12 et 14 sept. 2018 | # Les Piles | I2 — Promo 122

A. Notions à connaître sur les piles

1. Définition

Une **pile** (*stack* en anglais) est une collection d'objets sur laquelle deux opérations sont possibles :

- ajouter un élément,
- retirer l'élément le plus récemment inséré.

On prend souvent l'image d'une pile d'assiettes ou d'une pile de journaux : insérer un élément dans la pile se dit **empiler** (*push* en anglais) et retirer un élément de la liste se dit **dépiler** (*pop* en anglais).

On définit 4 opérations permettant de manipuler les piles :

1. créer une pile vide
2. déterminer si une pile est vide
3. ajouter un élément au sommet de la pile
4. retirer un élément au sommet de la pile

2. Implémentation en Python

L'implémentation de ces opérations dépendant du langage utilisé et des choix du programmeur. Une possibilité d'implémentation d'une pile sous Python est d'utiliser une liste. Les quatre opérations peuvent alors se coder de la façon suivante :

```
1  # 1 - Créer une pile vide
2  def pilevide():
3  """  retourne une pile vide
4  exemple : P=pilevide() """
5   →return []
6
7  # 2 - Déterminer si une pile est vide
8  def estvide(P):
9  """  retourne TRUE si une pile est vide
10 exemple : print(estvide(pilevide())) """
11  →return P==pilevide()
12
13 # 3 - Empiler un élément dans une pile
14 def empiler(P,e):
15 """ Ajoute l'élément e au sommet de la pile P
16 Exemple : empiler(pilevide(),5) """
17  →P.append(e)
18
19 # 4 - Dépiler un élément
20 def depiler(P):
21 """ retourne l'élément se trouvant sommet de la pile P
22 exemple : print(depile(P)) """
23  →return(P.pop())
```

B. Exercices d'application

En utilisant uniquement les 4 primitives définies ci-dessus, déterminer les arguments et coder les fonctions suivantes :

Exercice 1 Ecrire une fonction `sommet` qui retourne l'élément `s` au sommet de la pile sans la modifier.

Exercice 2 Améliorer la fonction précédente pour ne pas avoir une erreur en cas de pile vide.

Exercice 3 Ecrire une fonction `taille` qui renvoie la taille d'une pile.

Exercice 4 Ecrire une fonction `taille2` qui renvoie la taille de la pile mais cette fois sans la modifier.

Exercice 5 Ecrire une fonction `inverse` qui inverser l'ordre d'une pile.

Exercice 6 Ecrire une fonction `verif` qui vérifie si une pile donnée contient un objet donné.

Exercice 7 Ecrire une fonction `element` qui retourne le k-ième élément de la pile à partir du haut (s'il y en a !)

Exercice 8 Ecrire une fonction `circ` qui modifie une pile en envoyant à son sommet L'Élément qui se trouve tout en bas de la pile (on appelle cela une permutation circulaire).

C. Evaluation d'une expression postfixée

Définition

En 1920, le mathématicien polonais Lukasiewicz introduit une notation permettant de noter les calculs sans utiliser de parenthèse : au lieu de placer l'opérateur entre les deux opérandes, l'opérateur est placé après ses opérandes. Quelques exemples :

- le calcul `4 + 2` se note `4 2 +`
- le calcul `sin(2)` se note `2 sin`
- le calcul `1+sin(2+3)` se note `1 2 3 + sin +`
- le calcul `2×(3+5)` se note `2 3 5 + ×`

Exercice 9
1. Traduire `(1+2)×(3+4)` en notation postfixée.
2. Que dire de `a b + c +` et `a b c + +`?
3. Traduire en notation classique `a b + 2 / sin a b - 2 / cos +`
   On se limitera dans un premier temps aux opérateurs $\times$ et `+`.

Exercice 10 Ecrire une fonction `estReel(string)` qui renvoie "true" si la chaîne de caractères string est un réel et "false" sinon.

Exercice 11 Ecrire une fonction `operation(string1,string2,string3)` telle que

```
operation("a","b","+")
```

renvoie la chaîne de caractère `"(a+b)"`

Le programme principal

Pour évaluer la valeur d'une expression postfixée, l'idée est la suivante :

1. On parcourt la liste de la gauche vers la droite et on empile les réels rencontrés
2. Dès que l'on rencontre un opérateur : on dépile les deux derniers réels (ou le dernier si l'opérateur est unitaire), on leur applique l'opérateur en question et on empile le résultat.

Exercice 12 Traduire ce principe de calcul en un pseudo-code ou un arbre programmatique.

Exercice 13 Ecrire une fonction `traduction(string)` qui traduit une expression postfixée en une expression usuelle. Par exemple :

```
print(traduction("2 3 + 5 *"))
```

retournera la chaîne de caractère `"((2+3)*5)"`.