Nom : .....

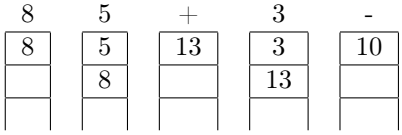
Terminale NSI
Devoir surveillé
durée : 1 heure

Le langage *Forth* a été créé au début des années 1960 par Chuck Moore (né en 1938, photo ci-jointe). C'est un langage important du point de vue de l'histoire de l'informatique, d'une part parce qu'il a été développé pour l'informatique embarquée (suivi de satellites, guidage de téléscopes, ...), d'autre part parce qu'il est basé sur la structure de pile (*stack*). Forth s'inscrit dans le paradigme de **programmation impérative** et c'est un langage interprété (il n'a pas de compilateur). Un interpréteur Forth occupe typiquement moins de 10 ko...

1. Parmi les langages de programmation suivants, lesquels sont également des langages interprétés ? Cocher la case si le langage est interprété.
   - ☐ Lisp (programmation fonctionnelle)
   - ☐ Prolog (programmation logique)
   - ☐ C (programmation impérative)
   - ☐ Java (programmation objet)
   - ☐ JavaScript (programmation événementielle)
   - ☐ Python (programmation objet)
   - ☐ Haskell (programmation fonctionnelle)

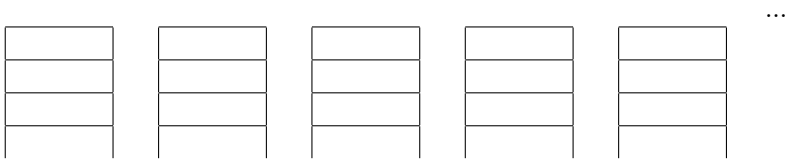
2. **Notation polonaise inversée**

   La pile de Forth est utilisée pour évaluer des expressions algébriques comme `8 5 + 3 -`, écrites en notation polonaise inversée (*RPN*[1]). L'expression `8 5 + 3 -` s'évalue à 10 comme le montrent les états successifs de la pile de Forth :



   Que donne l'évaluation par Forth, de l'expression `2 3` × `5 +` ? .....

   On pourra s'aider de ces dessins d'états successifs de la pile de Forth :



3. **Pile**

   L'interpréteur de Forth est basé sur une pile [2] (`stack`), dotée de trois méthodes :
   - `push` (fonction ayant pour argument l'objet à empiler) pour ajouter un élément au sommet de la pile,

1. Reversed Polish Notation : proposée par Arthur Burks (1915-2008) en 1945, sur l'ordinateur `ENIAC`.
2. Proposition faite par Alan Turing (1912-1954) sur l'ordinateur `ACE`

- `pop` pour dépiler (enlever le dernier élément de la pile), renvoie l'élément dépilé,
- `empty` renvoyant un booléen disant si la pile est vide.

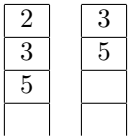
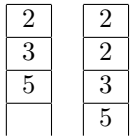
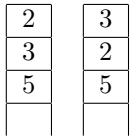
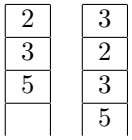
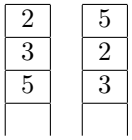
Cette pile peut être simulée en Python sous forme d'une variable globale appelée `stack`, initialisée à la liste vide, et sur laquelle agissent les trois fonctions `push`, `pop` et `empty`. Noter que seule la fonction `pop` renvoie quelque chose : l'élément dépilé.

Compléter le script Python ci-dessous, en écrivant les noms des fonctions, parmi `push`, `pop` et `empty` :

```
stack = []
def ........(elt):
        stack.append(elt)
def ........():
        return stack.pop()
def ........():
        return len(stack)==0
```

4. **Autres instructions de Forth**

La pile de Forth est plus riche que celle de Turing : outre `push`, `pop` et `empty`, elle possède les instructions suivantes (chacune illustrée après `5 3 2`) :

drop	**dup**	**swap**	**over**	**rot**
				

Plus précisément,

- `drop` diminue d'une unité la longueur de la pile, en supprimant le haut de la pile (*drop* veut dire jeter).
- `dup` au contraire, allonge la pile, en *dup*licant le haut de celle-ci.
- `swap` échange (*to swap* en anglais) les contenus des deux cellules du haut de la pile.
- `over` duplique l'avant-dernier élément de la pile, en plaçant sa copie au-dessus (*over* en anglais) de la pile.
- `rot` déplace l'avant-avant-dernier élément de la pile vers le haut de celle-ci (ce qui revient à une *rot*ation de ses trois derniers éléments).

Le but de cet exercice est de voir comment on peut programmer ces instructions de Forth avec le modèle de pile de Turing (`push` et `pop`). On utilisera des variables (autres que la pile `stack`) temporaires.

Par exemple, l'instruction `top` qui permet de voir le haut (*top* en anglais) de la pile, sans modifier celle-ci, peut s'écrire ainsi en Python :

```
def top():
    x = pop()
    push(x)
    return x
```

Un autre exemple est `drop` :

```
def drop():
        pop()
```

Un autre est `over` :

```
def over():
        x = pop()
        y = pop()
        push(y)
        push(x)
        push(y)
```

et `rot` :

```
def rot():
        x = pop()
        y = pop()
        z = pop()
        push(y)
        push(x)
        push(z)
```

(a) Définir à partir de `push` et `pop` (plus éventuellement des affectations de variables et `return`) l'instruction `dup` :

```
def dup():
        ...
        ...
        ...
        ...
```

(b) Définir à partir de `push` et `pop` (plus éventuellement des affectations de variables et `return`) l'instruction `swap` :

```
def swap():
        ...
        ...
        ...
        ...
```

(c) Quelle instruction de Forth obtient-on avec la séquence `swap dup rot swap` ? Compléter :

```
def ....():
        swap()
        dup()
        rot()
        swap()
```