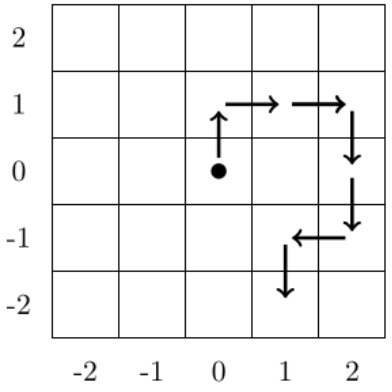
Déplacement d'un mobile sur une grille

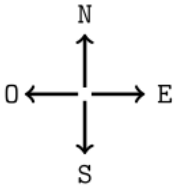
Un mobile se déplace sur une grille quadrillée. À partir d'une case de départ, il peut à chaque étape se déplacer sur une case adjacente.

On peut coder les déplacements par leur direction : Nord, Sud, Est, Ouest. Ces directions sont notées respectivement `'N'`, `'S'`, `'E'`, `'O'`.

On peut aussi coder les positions du mobile par la liste des coordonnées des cases occupées. La case de départ a pour coordonnées `[0, 0]`. Les coordonnées d'une case sont représentées par une liste `[x, y]` où `x` est l'abscisse et `y` l'ordonnée.

Exemple. Sur la figure ci-dessous, le point de départ est représenté par le petit disque noir. L'ensemble des déplacements du mobile, à partir de la case de départ et en suivant les flèches, est codé par la chaîne de longueur 7 : `'NEESSOS'`. L'ensemble des positions du mobile, en incluant la case départ, est codé par la liste de longueur 8 : `[[0, 0], [0, 1], [1, 1], [2, 1], [2, 0], [2, -1], [1, -1], [1, -2]]`.





A. Étude des codages

Question 1. On considère la chaîne `ch1 = 'NEESSOS'`.

a) Que renvoie `ch1[3]` ? Quel est le type du résultat ?

b) On exécute l'instruction `"NON" in ch1`. Parmi les quatre propositions suivantes, déterminer celle qui correspond au résultat obtenu.

`"OUI"` `"NON"` `True` `False`

Question 2. On considère la liste suivante :

`L1 = [[0, 0], [0, 1], [1, 1], [2, 1], [2, 0], [2, -1], [1, -1], [1, -2]]`.

a) Que renvoie `L1[3]` ? Quel est le type du résultat ?

b) Que renvoie `L1[3][1]` ? Quel est le type du résultat ?

Question 3. Recopier et compléter la fonction `DirEnPos` suivante qui prend en argument une chaîne `ch` dont tous les caractères sont dans l'ensemble {`'N'`, `'S'`, `'E'`, `'O'`} et qui renvoie la liste `L` des positions successives du mobile.

```
def DirEnPos(ch):
    x, y = 0, 0
    L = [[0, 0]]
    for lettre in ch:
        if lettre == 'N':
            y = y + 1
        if ... == ... :
            ... = ...
        if ... == ... :
            ... = ...
        if ... == ... :
            ... = ...
        L.append(...)
    return L
```

Question 4. On veut écrire une fonction `retour` qui prend en argument une chaîne `ch` représentant les directions de déplacements et qui renvoie `True` si le mobile est passé au moins 2 fois par une même case et `False` sinon.

Parmi les fonctions suivantes, déterminer l'unique fonction qui répond à la question.

```
def retour1(ch):
    n = len(ch)
    for i in range(n):
        for j in range(i+1, n):
            if ch[i] == ch[j]:
                return True
    return False
```

```
def retour2(ch):
    L = DirEnPos(ch)
    n = len(L)
    for i in range(n):
        for j in range(n):
            if L[i] == L[j]:
                return True
    return False
```

```
def retour3(ch):
    L = DirEnPos(ch)
    n = len(L)
    for i in range(n):
        for j in range(i+1, n):
            if L[i] == L[j]:
                return True
    return False
```

```
def retour4(ch):
    L = DirEnPos(ch)
    n = len(L)
    for i in range(n):
        for j in range(i+1, n):
            if L[i] == L[j]:
                return True
            else:
                return False
```

Question 5. Soit `L` une liste de longueur n dont tous les éléments sont des listes de 2 entiers. Soit k un entier compris entre 0 et $n-2$. On considère la fonction `difference` suivante d'arguments `L` et `k`.

```
def difference(L, k):
    return [L[k+1][0] - L[k][0], L[k+1][1] - L[k][1]]
```

On considère la liste suivante :

`L1 = [[0, 0], [0, 1], [1, 1], [2, 1], [2, 0], [2, -1], [1, -1], [1, -2]]`.

Indiquer ce que renvoient les trois instructions `difference(L1, 2)`, `difference(L1, 3)` et `difference(L1, 7)` ? On choisira les réponses parmi :

`[0, 1]` `[0, -1]` `[1, 0]` `[-1, 0]` Un message d'erreur

Question 6. On veut créer une fonction `PosEnDir` prenant en argument un liste `L` représentant les positions et qui renvoie la chaîne des directions des déplacements.

La fonction proposée ci-dessous comporte 3 erreurs. Indiquer les numéros des lignes à changer et les corrections nécessaires.

```
1  def PosEnDir(L):
2      n = len(L)
3      ch = ""
4      for k in range(n):
5          if difference(L, k) == [0, 1]:
6              ch = ch + 'N'
7          if difference(L, k) == [0, -1]:
8              ch = ch + 'S'
9          if difference(L, k) == [1, 0]:
10             ch = ch + 'O'
11         if difference(L, k) == [-1, 0]:
12             ch = ch + 'E'
13     return ch
```

B. Étude des répétitions dans une chaîne

Question 7. On considère la fonction `mystere` ci-dessous qui prend en arguments une chaîne `ch` et un entier `i` et la chaîne `ch2 = 'EONNNNSSEN'`, de longueur 10.

```
def mystere(ch, i):
    n = len(ch)
    j = 1
    while i+j < n and ch[i+j] == ch[i]:
        j += 1
    return j
```

Que renvoient les instructions `mystere(ch2, 0)`, `mystere(ch2, 1)` et `mystere(ch2, 2)` ?

Question 8. Recopier et compléter la fonction `maxRepet` suivante qui prend en argument une chaîne de directions `ch` et qui renvoie le nombre maximum de déplacements successifs du mobile dans la même direction.

Par exemple, `maxRepet(ch2)` doit renvoyer 4 car le caractère répété successivement le plus grand nombre de fois dans `ch2` est le `'N'` et il est répété à 4 reprises.

```
def maxRepet(ch):
    i = 0
    maxi = 1
    while i < len(ch):
        m = mystere(ch, i)
        if ...
            ...
        i = ...
    return ...
```

C. Génération de marches aléatoires

Question 9. Écrire une fonction `marche` qui prend en argument un entier `n` non nul et qui renvoie une chaîne de caractères de longueur `n`, dont tous les éléments sont choisis aléatoirement dans l'ensemble `{'N', 'S', 'E', 'O'}`. On pourra utiliser la fonction `randint` du module `random` qu'on supposera importé.

Question 10. On veut interdire le retour immédiat à la case que l'on vient de quitter. La chaîne codant les directions doit donc vérifier les contraintes suivantes :

- un `'N'` n'est pas suivi d'un `'S'` et, inversement, un `'S'` n'est pas suivi d'un `'N'` ;
- un `'E'` n'est pas suivi d'un `'O'` et, inversement, un `'O'` n'est pas suivi d'un `'E'`.

Parmi les fonctions suivantes qui prennent en argument une chaîne de caractères `ch`, déterminer les deux seules qui renvoient `True` si toutes ces contraintes sont respectées et `False` sinon.

```
def valide1(ch):
    n = len(ch)
    for k in range(n-1):
        if ch[k] == 'N' and ch[k+1] == 'S':
            return False
        if ch[k] == 'S' and ch[k+1] == 'N':
            return False
        if ch[k] == 'E' and ch[k+1] == 'O':
            return False
        if ch[k] == 'O' and ch[k+1] == 'E':
            return False
        return True
```

```
def valide2(ch):
    n = len(ch)
    for k in range(n-1):
        if ch[k] == 'N' and ch[k+1] == 'S':
            return False
        if ch[k] == 'S' and ch[k+1] == 'N':
            return False
        if ch[k] == 'E' and ch[k+1] == 'O':
            return False
        if ch[k] == 'O' and ch[k+1] == 'E':
            return False
    return True
```

```
def valide3(ch):
    n = len(ch)
    if ('NS' in ch) or ('SN' in ch) or ('EO' in ch) or ('OE' in ch):
        return False
    return True
```

```
def valide4(ch):
    n = len(ch)
    if ('NS' in ch) and ('SN' in ch) and ('EO' in ch) and ('OE' in ch):
        return False
    return True
```

Question 11. On considère la fonction `prochain` ci-dessous qui prend en argument un caractère `lettre`.

```
def prochain(lettre):
    if lettre == 'N':
        return 'NEO'
    if lettre == 'S':
        return 'SEO'
    if lettre == 'E':
        return 'ENS'
    if lettre == 'O':
        return 'ONS'
```

a) Que renvoie l'instruction `prochain('N')` ?

b) Que renvoie l'instruction `prochain('S')[0]` ?

c) Que renvoie l'instruction `prochain('E')[2]` ?

Question 12. Écrire une fonction `marcheContrainte` qui prend en argument un entier non nul `n` et qui renvoie une chaîne de caractères de longueur `n`, dont les éléments sont des directions de l'ensemble `{'N', 'S', 'E', 'O'}`, avec les 2 contraintes suivantes :

- la première direction suivie est imposée vers le Nord.
- à chaque étape, on exclut le retour à la case que l'on vient de quitter et la direction suivante est choisie aléatoirement entre les 3 directions restantes.

Indication. On pourra utiliser la fonction `prochain` de la question 11.

Solution.

A. Étude des codages

Question 1.

a) L'instruction `ch[3]` renvoie `'S'` et ce résultat est de type `str` (chaîne de caractères).

b) L'instruction `"NON" in ch1` renvoie le booléen `False` car `"NON"` n'est pas une sous-chaîne de `ch1`.

Question 2.

a) L'instruction `L1[3]` renvoie `[2, 1]`. Ce résultat est de type `List` (liste).

b) L'instruction `L1[3][1]` renvoie `1`. Ce résultat est de type `int` (entier).

Question 3.

```
def DirEnPos(ch):
    x, y = 0, 0
    L = [[0, 0]]
    for lettre in ch:
        if lettre == 'N':
            y = y + 1
        if lettre == 'S' :
            y = y - 1
        if lettre == 'E' :
            x = x + 1
        if lettre == 'O' :
            x = x - 1
        L.append([x, y])
    return L
```

Question 4. La seule fonction qui répond à la question est `retour3`. En effet, la fonction `retour1` compare les déplacements et non pas les cases. Dans la fonction `retour2`, le compteur `i` et le compteur `j` commence tous les deux à `0` donc la fonction renvoie toujours `True` (puisque pour `i` et `j` valant `0`, le test `L[i]=L[j]` est vrai. Enfin, la fonction `retour4` renvoie `False` dès que la chaîne la liste `L` contient deux listes différentes en raison du `else`.

Question 5. L'instruction `difference(L1,2)` renvoie `[1, 0]`, l'instruction `différence(L1,3)` renvoie `[0, -1]` et l'instruction `difference(L1,7)` renvoie une erreur car pour si `k` vaut 7 alors `k+1` vaut 8 et la liste `L1` admet 8 éléments numérotés de `0` à `7` donc `L1[8][0]` n'existe pas.

Question 6. À la ligne `4`, il faut remplacer `range(n)` par `range(n-1)` car l'indice `k` ne doit pas dépasser `n-2`. Ensuite, aux lignes `10` et `12`, il faut échanger `'O'` et `'E'`.

B. Étude des répétitions dans une chaîne

Question 7. Les instructions `mystere(ch2,0)` et `mystere(ch2,1)` renvoie `1` et L'instruction `mystere(ch2,2)` renvoie `4`. En effet, `mystere(ch, i)` renvoie le nombre de caractères consécutifs dans `ch` à partir de `ch[i]` qui sont égaux à `ch[i]`.

Question 8.

```
def maxRepet(ch):
    i = 0
    maxi = 1
    while i < len(ch):
        m = mystere(ch, i)
        if m > maxi:
          maxi = m
        i = i + m       # ou bien i = i + 1
    return maxi
```

Question 9.

```
def marche(n):
    L = ['N', 'S', 'E', 'O']
    ch = ""
    for i in range(n):
      ch += L[randint(0,3)]
    return ch
```

Question 10. Les deux fonctions qui conviennent sont les fonctions `valide2` et `valide3`. En effet, dans la fonction `valide1`, le `return True` est mal indenté et, dans la fonction `valide4` ne renvoie `False` qui si les 4 contraintes ne sont pas respectées alors qu'il en suffit d'une.

Question 11. L'instruction `prochain('N')` renvoie `'NEO'`, l'instruction `prochain('S')[0]` renvoie `'S'` et l'instruction `prochain('E')[2]` renvoie `'S'`.

Question 12.

```
def marche(n):
    ch = "N"
    for i in range(n-1):
      ch += prochain(ch[i])[randint(0,2)]
    return ch
```