



## Affichage d'un sablier

Écrivez une **fonction** nommée `afficher_sablier` qui accepte en argument un nombre  $n$  et qui **affiche** à la console  $2n + 1$  lignes de points qui ressemblent à un **sablier**.

Par exemple, pour  $n = 2$ , votre fonction doit afficher les caractères suivants :

```
.....
....
...
.
...
....
.....
```

**Notez bien** que nous utilisons ci-dessus le symbole  pour vous indiquer graphiquement l'emplacement des caractères de **fin de ligne** (immédiatement après le dernier point), mais votre code n'a **pas** à afficher ces symboles.

Pour  $n = 3$ , votre fonction doit afficher : (ici nous omettons les )

```
.....
.....
....
...
.
...
....
.....
.....
```

Pour  $n < 0$ , votre fonction ne doit **rien** afficher.

## Solution du professeur

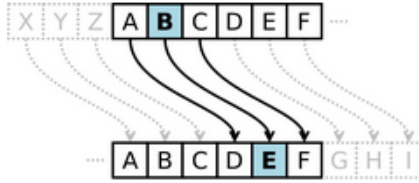
Notez que cette solution n'est généralement **pas** unique.



```
1 def afficher_sablier(n):
2     if n > 0:
3         for i in range(n+1):
4             print(' '*i+'.'(2*(n-i)+1))
5         for i in range(1, n+1):
6             print(' '*(n-i)+'.'*(2*i+1))
```

## Chiffrement de César

Une méthode simple pour chiffrer un message consiste à substituer les lettres de celui-ci par d'autres lettres, afin de le rendre illisible par le non initié. Parmi ce type de méthode, la plus connue est sans doute celle qu'utilisait Jules César pour communiquer avec ses généraux. Dans ses missives secrètes, il substituait simplement chaque lettre par la lettre située 3 rangs plus loin dans l'alphabet. Par exemple, il remplaçait tous les **a** par des **d**, tous les **b** par des **e**, etc. Sa clé de chiffrement était donc +3:



On vous demande de définir deux fonctions distinctes. La première fonction, nommée `chiffrer`, devra accepter en argument un message à chiffrer, représenté sous la forme d'une chaîne de caractères, ainsi qu'une clé de chiffrement sous la forme d'un nombre entier (positif ou négatif), et devra retourner le message chiffré par la clé. Vous pouvez considérer que le message ne contient que des lettres minuscules de l'alphabet (**a** à **z**) et des signes de ponctuation, sans aucun accent. Pour le cas des signes de ponctuation, ne faites aucune substitution. Par exemple, l'expression `chiffrer('secret:', 3)` doit retourner la chaîne `vhfuhw:`.

La deuxième fonction, nommée `dechiffrer`, devra accepter en argument un message chiffré ainsi que la clé qui a servi au chiffrement, et retourner le message déchiffré. Par exemple, l'expression `dechiffrer('vhfuhw:', 3)` doit retourner la chaîne `secret:`.

Notez bien qu'il est possible d'implanter la 2e fonction en faisant appel à la 1re. Notez aussi le contenu de la cellule de contexte. Nous vous fournissons dans cette cellule une chaîne contenant les 26 lettres de l'alphabet; utilisez-la à votre guise. Nous y avons aussi inclus un poème que vous pouvez utiliser pour faire des tests. Finalement, nous vous rappelons l'existence de la [méthode find](#) pour les chaînes de caractères. Cette méthode permet de retourner l'indice d'une sous-chaîne dans une chaîne.

### Contexte de l'exercice

```
1 alphabet = 'abcdefghijklmnopqrstuvwxyz'
2 poeme = '''
3 souvent, pour s'amuser, les hommes d'équipage
4 prennent des albatros, vastes oiseaux des mers,
5 qui suivent, indolents compagnons de voyage,
6 le navire glissant sur les gouffres amers.
7
8 à peine les ont-ils déposés sur les planches,
9 que ces rois de l'azur, maladroits et honteux,
10 laissent piteusement leurs grandes ailes blanches
11 comme des avirons traîner à côté d'eux.
12
13 ce voyageur aile, comme il est gauche et veule !
14 lui, naguère si beau, qu'il est comique et laid !
15 l'un agace son bec avec un brûle-gueule,
16 l'autre mime, en boitant, l'infirme qui volait !
17
18 le poète est semblable au prince des nuées
19 qui hante la tempête et se rit de l'archer ;
20 exilé sur le sol au milieu des huées,
21 ses ailes de géant l'empêchent de marcher.
22 '''
```

```
alphabet = 'abcdefghijklmnopqrstuvwxyz'
poeme = '''\nsouvent, pour s'amuser, les hommes d'équipage\nprennent des
albatros, vastes oiseaux des mers,\nqui suivent, indolents compagnons de
voyage,\nle navire glissant sur les gouffres amers.\n\n\na peine les ont-ils
deposes sur les planches,\nque ces rois de l'azur, maladroits et
honteux,\nlaissent piteusement leurs grandes ailes blanches\ncomme des
avirons traîner a cote d'eux.\n\n\nce voyageur aile, comme il est gauche et
veule !\n\nlui, naguere si beau, qu'il est comique et laid !\n\nl'un agace son
bec avec un brule-gueule,\n\nl'autre mime, en boitant, l'infirme qui volait
!\n\n\nle poete est semblable au prince des nuees\nqui hante la tempete et se
rit de l'archer ;\nexile sur le sol au milieu des huees,\n\nses ailes de geant
l'empechent de marcher.\n'''
```

### Solution du professeur

Notez que cette solution n'est généralement **pas** unique.

```
1 def chiffrer(message, n):
2     chiffre = ''
3     for lettre in message:
4         if lettre in alphabet:
5             i = alphabet.find(lettre)
6             chiffre += alphabet[(i + n) % len(alphabet)]
7         else:
8             chiffre += lettre
9     return chiffre
10
11
12 def dechiffrer(message, n):
13     return chiffrer(message, -n)
```

## Validation d'un code IMEI

Le code IMEI («*International Mobile Equipment Identity*») est un identifiant unique de **15 chiffres** utilisée par les téléphones cellulaires qui respectent la norme **GSM** (la plupart des téléphones contemporains).

On vous demande de définir une **fonction** nommée `valider_imei` qui accepte en argument une **chaîne** de caractères, et qui retourne un **booléen** indiquant si oui ou non la chaîne reçue correspond à un **code IMEI** valide.

Le code IMEI est en fait constitué d'une séquence de **14 chiffres** suivis d'un **15e chiffre** de validation. Pour déterminer si le code est **valide**, on peut procéder de la façon suivante:

1. tout d'abord, un code qui n'a **pas** exactement 15 chiffres est **nécessairement** invalide;
2. pour les  $j = 0, 2, 4, \dots, 12$ , calculez la somme  $A = \sum c_i$ , où  $c_i$  est le chiffre à l'**indices**  $i$  de la séquence;
3. pour les  $j = 1, 3, 5, \dots, 13$ , calculez la somme  $B = \sum x_j$  des valeurs:

$$x_j = \begin{cases} 2 \times c_j & \text{si } c_j < 5 \\ (2 \times c_j) - 9 & \text{autrement} \end{cases}$$

4. pour que le code soit valide, la somme  $A + B + c_{14}$  doit être un **multiple** de 10.

Par exemple, la séquence de 15 chiffres `490154203237518` est un code IMEI valide:

IMEI	4	9	0	1	5	4	2	0	3	2	3	7	5	1	x
Double every other	4	18	0	2	5	8	2	0	3	4	3	14	5	2	x
Sum digits	4 + (1 + 8) + 0 + 2 + 5 + 8 + 2 + 0 + 3 + 4 + 3 + (1 + 4) + 5 + 2 + x = 52 + x														

avec  $x = 8$ .

**Rappel:** avec l'[opérateur](#) `[]`, il est aisé de **découper** dans une chaîne les caractères d'indice pair ou impair.

**Indices:** les [fonctions standards](#) suivantes pourraient sans doute vous être **utiles** pour résoudre cet exercice:

- `int` pour convertir une chaîne de chiffres en nombre;
- `len` pour déterminer la longueur d'une séquence;
- `sum` pour faire la somme des éléments d'une séquence.

## Solution du professeur

Notez que cette solution n'est généralement **pas** unique.



```
1 def valider_imei(code):
2     if len(code) != 15:
3         return False
4     A = sum(int(c) for c in code[0:14:2])
5     B = sum((2*c if c < 5 else 2*c-9) for c in (int(c) for c in code[1:14:2]))
6     return (A+B+int(code[14])) % 10 == 0
```

## Tri postal

Les codes postaux du Canada comportent 6 caractères **alternant** d'une lettre à un chiffre et d'un chiffre à une lettre. Par exemple, le code postal de l'Université Laval est **G1V0A6**. La **1<sup>re</sup> lettre** du code postal identifie la région. Les différentes régions du Canada sont illustrées sur la carte ci-contre. La lettre **G** désigne l'est du Québec, le **H** désigne la région de Montréal et le **J** est utilisé pour le reste du Québec.

Le **1<sup>er</sup> chiffre** spécifie quant à lui si l'adresse est urbaine ou rurale. Le **zéro** désigne une adresse **rurale**, alors que les **autres chiffres** sont réservés pour des adresses **urbaines**. La **2<sup>e</sup> lettre** représente soit une région rurale particulière, soit une ville entière de taille moyenne, ou encore un secteur spécifique d'une grande ville. Les trois derniers caractères désignent une **unité de distribution locale** comme, par exemple, un certain quartier résidentiel à l'intérieur d'un secteur d'une ville:



On vous demande de définir une **fonction** nommée `trier_colis` qui accepte en argument une **liste** de codes postaux et qui **retourne** un dictionnaire qui associe des **listes** de codes postaux à des **numéros** de bac d'expédition. Les numéros possibles pour les bacs d'expédition sont:

1. pour les colis à destination de **Montréal**;
2. pour les colis à destination d'une région **urbaine** du Québec (autre que Montréal);
3. pour les colis à destination d'une région **rurale** du Québec;
4. pour les colis à destination d'une province canadienne **autre** que le Québec;
5. pour les colis destinés à **l'étranger**.

Supposez que **tout** code postal qui ne débute **pas** par une lettre ou qui n'a **pas** 6 caractères est un code postal **étranger**. Par exemple, une série de cinq chiffres pourrait être un code postal américain ou français. Votre fonction doit **reconnaître** le code postal canadien **avec ou sans** espace pour séparer les deux groupes de trois caractères, et avec les lettres en **majuscules** ou en **minuscules**. Le dictionnaire retourné par votre fonction doit toujours contenir les **cinq** clés des cinq bacs d'expédition, même lorsqu'**aucun** colis n'est associé à la destination correspondante. Dans ce cas, la clé sera associée à une **liste vide**. Par exemple:

```
trier_colis(['G1X 1B5', 'J0A0A0', '12345', 'H3C 0A7', 'v5c3A7'])
```

doit produire:

```
{1: ['H3C 0A7'], 2: ['G1X 1B5'], 3: ['J0A0A0'], 4: ['v5c3A7'], 5: ['12345']}
```

## Solution du professeur

Notez que cette solution n'est généralement **pas** unique.



```
1 def trier_colis(codes):
2     tri = {1: [], 2: [], 3: [], 4: [], 5: []}
3     for code in codes:
4         tmp = code.replace(' ', '').upper()
5         if len(tmp) != 6 or tmp and not tmp[0].isalpha():
6             tri[5].append(code)
7         elif tmp[0] in 'H':
8             tri[1].append(code)
9         elif tmp[0] in 'GJ':
10            if tmp[1] != '0':
11                tri[2].append(code)
12            else:
13                tri[3].append(code)
14        else:
15            tri[4].append(code)
16    return tri
```

## Nombres d'arguments

Écrivez une fonction nommée `nargs` qui accepte un nombre **arbitraire** d'arguments et qui **retourne** le nombre d'arguments reçus. Pour ce faire, utilisez un **seul** argument étoilé.

Bravo!

Votre score est 100/100.

Il vous reste 1 soumission avec rétroaction.

### Entrez votre solution dans la cellule ci-dessous

Notez bien que **seul** le contenu de cette cellule sera évalué par le correcteur automatique.

```
1 ▶ def nargs(*b):  
2   return len(b)
```

### Faites vos tests ici

Notez qu'il est inutile de copier votre solution dans cette cellule, car **toutes** les cellules partagent le même interpréteur python. À partir de cette cellule, faites **directement** appel aux éléments de votre solution afin de **bien** les tester.

```
▶ 1
```

### Solution du professeur

Notez que cette solution n'est généralement **pas** unique.

```
▶ 1 def nargs(*args):  
2   return len(args)
```

## Arguments obligatoirement nommés

Écrivez une fonction nommée `fc3` qui accepte trois arguments nommés `x`, `y` et `z`, et qui **retourne** la somme des trois arguments.

Faites en sorte que ces trois arguments soit **obligatoirement** nommés, c'est-à-dire que l'on ne puisse **pas** appeler cette fonction sans spécifier explicitement les noms des arguments. Faites également en sorte que l'argument `z` possède une valeur **par défaut** égale à 17. Par exemple, on doit pouvoir appeler `fc3(x=1, y=2)` ou `fc3(z=3, y=2, x=1)`, mais pas `fc3(1, 2)` ni `fc3(1, 2, 3)`.

Bien que votre solution soit fonctionnelle, veuillez corriger les erreurs de style suivantes ([PEP-8](#))

```
stdin:1:19: E251 unexpected spaces around keyword / parameter equals  
def fc3(*, x, y, z = 17):  
    ^  
stdin:1:21: E251 unexpected spaces around keyword / parameter equals  
def fc3(*, x, y, z = 17):  
    ^
```

Votre score est 98/100.

### Entrez votre solution dans la cellule ci-dessous

Notez bien que **seul** le contenu de cette cellule sera évalué par le correcteur automatique.

```
1 ▶ def fc3(*, x, y, z = 17):  
2   return x + y + z
```

### Faites vos tests ici

Notez qu'il est inutile de copier votre solution dans cette cellule, car **toutes** les cellules partagent le même interpréteur python. À partir de cette cellule, faites **directement** appel aux éléments de votre solution afin de **bien** les tester.

```
▶ 1
```

### Solution du professeur

Notez que cette solution n'est généralement **pas** unique.

```
▶ 1 def fc3(*, x, y, z=17):  
2   return x+y+z
```