



MINISTÈRE  
DE L'ÉDUCATION  
NATIONALE  
ET DE LA JEUNESSE

Liberté  
Égalité  
Fraternité



ACADÉMIE  
DE VERSAILLES

Liberté  
Égalité  
Fraternité



ACADÉMIE  
DE TOULOUSE

Liberté  
Égalité  
Fraternité

---

# Olympiades numérique et sciences informatiques (NSI)

*Académie de Versailles - 24 avril 2024*

---

Cette épreuve est individuelle et dure trois heures.

Aucun document n'est autorisé. Aucun matériel électronique n'est autorisé, en particulier **les calculatrices et les ordinateurs ne sont pas autorisés**.

Le seul langage de programmation autorisé dans cette épreuve est Python. Le code proposé doit impérativement être proprement indenté afin d'éviter toute ambiguïté quant à sa validité. De plus, pour les mêmes raisons, on veillera à ne pas écrire le code d'une fonction à cheval sur deux pages. Plus généralement, on veillera à la lisibilité, la simplicité et l'efficacité du code proposé. De plus, l'utilisation d'identifiants significatifs pour les fonctions et les variables, ainsi que l'emploi judicieux de commentaires dans le code seront appréciés.

Ce sujet comporte deux exercices totalement indépendants. Ces deux exercices doivent être **traités sur des copies séparées**.

À tout moment vous pouvez faire appel à une fonction définie plus haut dans l'exercice, même si vous n'avez pas traité la question correspondante. Si vous ne pouvez pas formuler une réponse complète à une question, il vous est néanmoins conseillé d'exposer le bilan de vos pistes de recherches.

Le sujet comporte 16 pages. Avant de commencer, vérifiez que votre exemplaire est complet.

**Le verso de cette feuille est une annexe détachable, le premier exercice commence à la page 3.**

## Rappel

On rappelle qu'en Python, l'instruction `t = (24, "avril", 2024)` crée une variable `t` de type `Tuple` telle que `t[0]` vaut l'entier 24, `t[1]` la chaîne de caractères "avril" et `t[2]` l'entier 2024. On peut accéder aux différentes composantes d'un tel tuple `t` par leurs indices (avec `t[0]`, `t[1]` ou `t[2]`) ou bien récupérer ces composantes dans des variables par une instruction comme `(jour, mois, annee) = t`, après laquelle `jour` vaut 24, `mois` vaut "avril" et `annee` vaut 2024. Il n'est **pas possible** en revanche de modifier les composantes d'un tuple.

## Annexe détachable - Exercice 2

Nom :

Prénom :

Utiliser la grille ci-dessous pour répondre à la question 7 de l'exercice 2.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0															
1															
2															
3															
4															
5															
6															
7															
8															
9															
10															
11															
12															
13															
14															

# Exercice 1 : Montre GPS

Une montre destinée à la pratique du trail enregistre des données en utilisant un système de navigation par satellite (GPS) qui permet de déterminer l'heure et la position de l'athlète. De plus, cette montre détermine l'altitude en utilisant les différences de pression atmosphérique.

Par exemple, à la fin d'une course, l'ensemble des données enregistrées par la montre peut être la liste suivante où chaque caractère '␣' correspond à une espace.

```
["2023:08:16␣08:31:54␣N42.91343␣E00.13707␣1048",  
"2023:08:16␣08:32:02␣N42.91399␣E00.13708␣1050",  
"2023:08:16␣08:32:09␣N42.91421␣E00.13596␣1051",  
"2023:08:16␣08:32:17␣N42.91475␣E00.13561␣1054",...]
```

Un élément de cette liste est appelé une **trame**. La première trame correspond au moment où l'athlète active sa montre et la dernière est enregistrée quand la montre est désactivée. Une trame est une chaîne de caractères qui comporte toujours 44 caractères. La première trame de l'exemple précédent, à savoir "2023:08:16␣08:31:54␣N42.91343␣E00.13707␣1048", est constituée de 5 sous-chaînes séparées par des espaces :

- la sous-chaîne de 10 caractères "2023:08:16" correspond à la date ;
- la sous-chaîne de 8 caractères "08:31:54" correspond à l'heure ;
- la sous-chaîne de 9 caractères "N42.91343" correspond à la latitude ;
- la sous-chaîne de 9 caractères "E00.13707" correspond à la longitude ;
- la sous-chaîne de 4 caractères "1048" correspond à l'altitude en mètres entiers.

Dans tout l'exercice, on suppose que les données sont collectées à une même date.

## Partie 1 : Gestion de l'heure

On souhaite pouvoir vérifier qu'une chaîne de caractères est au format des heures dans les trames (comme la sous-chaîne "08:31:54" de la première trame donnée en exemple).

### Question 1

Écrire une fonction `format` qui prend en argument une chaîne de caractères `heure` et qui renvoie un booléen : elle renvoie `True` si `heure` comporte 8 caractères et si le caractère ':' est situé aux bons emplacements et elle renvoie `False` sinon. Par exemple :

- `format("04:12:55")` et `format("23:07:56")` valent `True` ;
- `format("12:2:55")` et `format("12;10:25")` valent `False`.

On souhaite maintenant pouvoir vérifier qu'une chaîne de caractères au bon format<sup>1</sup> est bien une heure valide. Par exemple :

- "04:12:55" et "23:07:56" sont des heures valides ;
- "18:62:55" et "24:00:00" ne sont pas des heures valides.

---

1. c'est-dire une chaîne constituée de deux chiffres, un caractère ':', deux chiffres, un caractère ':' et enfin deux chiffres

On propose ci-contre le début de la fonction `heure_valide` qui prend en argument une chaîne de caractères `heure` et qui renvoie `True` si `heure` a le bon format et est valide, et `False` sinon.

### Question 2

Compléter le code de la fonction `heure_valide`. On rappelle que `int("25")` renvoie l'entier 25.

```
1 def heure_valide(heure) :
2     if format(heure) == False :
3         return False
4     else :
5         h = int(heure[0] + heure[1])
6         m = int(heure[3] + heure[4])
7         s = int(heure[6] + heure[7])
8         # à compléter
```

### Question 3

Écrire une fonction `extrait_temps_ecoule` qui prend en argument une chaîne de caractères `heure` représentant une heure valide et qui renvoie un entier correspondant au nombre de secondes depuis minuit. Par exemple, `extrait_temps_ecoule("00:02:12")` vaut l'entier 132.

Il arrive que certaines trames soient interverties ou dupliquées lors du transfert de données, on veut pouvoir vérifier que ce n'est pas le cas dans les données reçues.

### Question 4

Écrire une fonction `verification_ordre` qui prend en arguments deux chaînes de caractères `h1` et `h2` représentant des heures valides et qui renvoie un booléen. Ce booléen vaut `True` si `h2` est strictement postérieure à `h1` et `False` sinon.

### Question 5

Écrire une fonction `duree` qui prend en arguments deux chaînes de caractères `h1` et `h2` représentant des heures valides et qui renvoie un entier. Cet entier représente la durée, en secondes, entre les instants `h1` et `h2`. Par exemple, `duree("15:48:12", "15:50:18")` vaut l'entier 126.

## Partie 2 : Validité et extraction des données d'une trame

On souhaite vérifier qu'au cours du transfert d'une trame, aucune des cinq sous-chaînes (date, heure, latitude, longitude et altitude) n'a été oubliée ni dégradée. On dit qu'une trame est **valide** si elle comporte bien ces cinq éléments, séparés par des espaces, et que chacun d'eux est lui-même valide. De même que nous disposons de la fonction `heure_valide` depuis la question 2, on suppose disposer dans la suite des fonctions suivantes :

- `date_valide` qui prend en argument une chaîne de caractères `date` et qui renvoie `True` si `date` est de longueur 10, au bon format, et représente une date valide et renvoie `False` sinon ;
- `lat_valide` qui prend en argument une chaîne de caractères `lat` et qui renvoie `True` si `lat` est de longueur 9, au bon format, et représente une latitude valide et renvoie `False` sinon ;
- `long_valide` qui prend en argument une chaîne de caractères `long` et qui renvoie `True` si `long` est de longueur 9, au bon format, et représente une longitude valide et renvoie `False` sinon ;
- `alt_valide` qui prend en argument une chaîne de caractères `alt` et qui renvoie `True` si `alt` est de longueur 4, au bon format, et représente une altitude valide et renvoie `False` sinon.

On rappelle que si `t` est une chaîne de caractères, `t[i:j]` est la sous chaîne de `t` constituée des caractères d'indices  $m$  tels que  $i \leq m < j$ . Si `t="2023:08:16 08:31:54 N42.91343 E00.13707 1048"` par exemple, `t[3:10]` vaut `"3:08:16"`.

### Question 6

Écrire une fonction `trame_valide` qui prend en argument une chaîne de caractères `trame` et qui renvoie `True` si `trame` représente bien une trame valide et `False` sinon.

De même que nous disposons de la fonction `extrait_temps_ecoule` depuis la question 3, on suppose disposer dans la suite des fonctions suivantes :

- `extrait_lat` qui prend en argument une chaîne de 9 caractères `lat` qui est une latitude valide et qui renvoie un flottant représentant la latitude ;
- `extrait_long` qui prend en argument une chaîne de 9 caractères `long` qui est une longitude valide, et qui renvoie un flottant représentant la longitude ;
- `extrait_alt` qui prend en argument une chaîne de 4 caractères `alt` qui est une altitude valide, et qui renvoie un entier représentant l'altitude.

### Question 7

Écrire une fonction `extrait_trame` qui prend en argument une chaîne de caractères `trame` représentant une trame valide. et qui renvoie les informations de cette trame sous la forme d'un tuple `(t, lati, longi, alt)` où :

- `t` est de type `int` et représente le temps écoulé depuis minuit en secondes ;
- `lati` est de type `float` et représente la latitude ;
- `longi` est de type `float` et représente la longitude ;
- `alt` est de type `int` et représente l'altitude en mètres.

## Partie 3 : Analyse des performances de l'athlète

Dans toute la suite, le parcours réalisé par l'athlète est représenté par une liste dont chaque élément est un tuple de la forme `(h, lat, long, alt)` qui, comme expliqué à la question précédente, représente une trame. On appellera **parcours** de telles listes. Par exemple, le parcours correspondant à la liste de trames donnée en exemple en début de sujet (page 3) est la liste suivante.

```
[(30714, 42.91343, 0.13707, 1048),  
(30722, 42.91399, 0.13708, 1050),  
(30729, 42.91421, 0.13596, 1051),  
(30737, 42.91475, 0.13561, 1054),  
...]
```

### Question 8

Écrire une fonction `alt_max` qui prend en argument un parcours non vide `p` (représenté par une liste comme décrit ci-avant) et qui renvoie l'altitude maximale, en mètres, atteinte par l'athlète durant le parcours `p`.

Le dénivelé positif réalisé lors d'un parcours est le total des montées réalisées lors de ce parcours. Par exemple, si les altitudes successives d'un parcours sont 1048, 1051, 1054, 1049 et 1053, le dénivelé positif de ce parcours vaut 10 mètres.

### Question 9

Écrire une fonction `denivele_positif` qui prend en argument un parcours `p` et qui renvoie, sous la forme d'un entier, le total des montées, en mètres, du parcours `p`.

Dans la suite, on dispose d'une fonction `distance(lati1, longi1, lati2, longi2)` qui renvoie la distance sur le globe terrestre, en mètres, entre des points de coordonnées géographiques respectives `(lat1, long1)` et `(lat2, long2)`.

### Question 10

Écrire la fonction `vitesse_moyenne` qui prend en argument un parcours `p` et qui renvoie la vitesse

moyenne en mètres par seconde de l'athlète durant le parcours  $p$ . On suppose que le parcours  $p$  contient au moins deux éléments correspondant à des instants distincts.

On considère que l'athlète est en pause entre 2 instants  $t_1$  et  $t_2$  consécutifs si la distance entre ses positions aux instants  $t_1$  et  $t_2$  est inférieure ou égale à 3 mètres.

### Question 11

Écrire la fonction `en_pause` qui prend en argument quatre flottants `lati1, longi1, lati2` et `longi2` et qui renvoie `True` si l'athlète est en pause entre les points de coordonnées géographiques `(lati1, longi1)` et `(lati2, longi2)` et `False` sinon.

### Question 12

Écrire une fonction `temps_pause` qui prend en argument un parcours  $p$  et qui renvoie, sous la forme d'un nombre entier de secondes, la durée totale des pauses de l'athlète durant le parcours  $p$ .

### Question 13

Écrire la fonction `nombre_pause` qui prend en argument un parcours  $p$  qui renvoie le nombre de pauses effectuées durant le parcours  $p$ .

## Partie 4 : Gestion des performances des participants au trail

Le but de cette partie est de gérer ou d'utiliser les données des athlètes participant à une course organisée sur un circuit. Pour participer à un trail, les athlètes doivent s'inscrire via un site internet et renseigner leur nom, leur prénom, leur sexe, leur année de naissance et une adresse mail.

Chaque athlète porte une montre connectée et au franchissement de la ligne d'arrivée, le temps réalisé par l'athlète est enregistré dans la montre comme un nombre entier de secondes. Les gestionnaires de la course récupèrent les données de l'ensemble des participants sous la forme d'un dictionnaire dont les clés sont les numéros de dossard des athlètes. La valeur associée à un numéro de dossard dans ce dictionnaire est elle-même un dictionnaire regroupant les données de l'athlète. Les clés et les valeurs associées dans un tel dictionnaire d'athlète sont explicitées dans le tableau suivant.

Clés	Valeurs attendues	Type de valeur
"mail"	Chaîne de caractères	str
"sexe"	'F' ou 'H'	str
"nom"	Chaîne de caractères	str
"prénom"	Chaîne de caractères	str
"annee_naissance"	Entier (entre 1000 et 9999)	int
"temps"	Entier positif	int

Par exemple, à la fin d'un trail, l'ensemble des données du trail peut être le dictionnaire `Trail_Montcalm` décrit ci-après.

```

Trail_Moncalm = {
5: {"annee_naissance": 1998, "temps": 9891, "sexe": 'F',
    "mail": "GermainL@geaymail.com",
    "nom": "Germain", "prenom": "Lucille"},
2: {"annee_naissance": 1989, "temps": 8327, "sexe": 'H',
    "mail": "loicrobert@violet.fr",
    "nom": "Robert", "prenom": "Loïc"},
1: {"annee_naissance": 1994, "temps": 9672, "sexe": 'F',
    "mail": "clemgeoffray@brulantmail.fr",
    "nom": "Geoffray", "prenom": "Clémentine"},
6: {"annee_naissance": 1989, "temps": 8252, "sexe": 'H',
    "mail": "BaronianThib@vodaifone.fr",
    "nom": "Baronian", "prenom": "Thibaut"},
3: {"annee_naissance": 1998, "temps": 9819, "sexe": 'F',
    "mail": "louiseSB@libre.fr",
    "nom": "Serban-Penhoat", "prenom": "Louise"},
4: {"annee_naissance": 1995, "temps": 8464, "sexe": 'H',
    "mail": "ThomCard@essaifair.fr",
    "nom": "Cardin", "prenom": "Thomas"}
}

```

Dans toutes les fonctions suivantes, l'argument `trail` est un dictionnaire rassemblant les données des athlètes ayant participé à un trail au format décrit ci-avant, comme par exemple le dictionnaire `Trail_Moncalm`.

Dans toute cette partie, on suppose que :

- tous les athlètes au départ d'un trail le terminent ;
- les temps réalisés par les athlètes sont différents les uns des autres ;
- le trail compte au minimum trois athlètes au départ.

#### Question 14

On donne ci-contre le code de la fonction `fonction_mystere`. Pour un dictionnaire `trail` représentant les données des athlètes d'un trail comme expliqué plus haut, expliquer ce que renvoie l'appel de fonction `fonction_mystere(trail)`.

```

1 def fonction_mystere(trail) :
2     c = 0
3     for cle in trail.keys() :
4         if trail[cle]["sexe"] == 'F' :
5             c = c + 1
6     return c

```

On suppose dans la suite que l'on dispose d'une fonction `conversion` qui prend en argument un entier `n` tel que  $0 \leq n \leq 60 \times 60 \times 24$  et qui renvoie une chaîne de caractères au format `"hh:mm:ss"` décrivant la durée de `n` secondes en heures, minutes et secondes. Par exemple, `conversion(6205)` renvoie la chaîne de caractères `"01:43:25"`.

#### Question 15

Écrire une fonction `info_dossard` qui prend en arguments un dictionnaire `trail` et un entier `num_dossard` et qui renvoie un tuple de deux chaînes de caractères décrivant l'athlète de numéro de dossard `num_dossard` dans `trail` s'il existe, et `None` sinon. Plus précisément, dans le cas où cet athlète existe,

- la première chaîne donne les nom et prénom de l'athlète au format `"nom prenom"` ;
- la seconde donne son temps au format `"hh:mm:ss"`.

### Question 16

Écrire une fonction `temps_max` qui prend en paramètre un dictionnaire `trail` et renvoie un tuple dont le premier élément est le numéro de dossard et dont le deuxième élément est le temps, en secondes, de l'athlète ayant réalisé le temps maximal au trail décrit par `trail`.

### Question 17

Écrire une fonction `temps_min` qui prend en argument un dictionnaire `trail` et qui renvoie un tuple dont le premier élément est le numéro de dossard et dont le deuxième élément est le temps, en secondes, de l'athlète ayant réalisé le temps minimal au trail décrit par `trail`.

### Question 18

Écrire une fonction `podium` qui prend en argument un dictionnaire `trail` et qui renvoie un tuple de trois éléments décrivant les athlètes ayant réalisé les trois plus petits temps lors du trail décrit par `trail`. Chaque athlète sera décrit par un tuple de trois éléments :

- le premier est son numéro de dossard (un entier donc) ;
- le second donne ses nom et prénom dans une chaîne au format "`nom prenom`" ;
- le troisième donne son temps dans une chaîne au format "`hh:mm:ss`".



## Exercice 2 : Le Scrabble

Le Scrabble est un jeu de société où chaque joueur cherche à créer des mots à l'aide des lettres dont il dispose, puis à placer ces mots sur un plateau de jeu. Les lettres sont écrites en majuscules sur des jetons. Dans cette version du Scrabble, il n'y a pas de jetons blancs. On appellera « lettre » un jeton sur lequel est indiquée une lettre. Chaque joueur dispose d'un chevalet où il place 7 lettres qu'il a piochées au hasard. Ci-dessous un exemple de chevalet.



Lorsque c'est son tour, le joueur doit positionner un mot sur le plateau de jeu quadrillé à partir des lettres qu'il a sur son chevalet : il place une lettre par case et doit obligatoirement utiliser au moins une des lettres déjà placées sur le plateau de jeu (sauf pour le premier mot posé). Après avoir posé ses lettres, le joueur repioche autant de lettres pour en avoir à nouveau 7 disponibles. Tous les mots doivent être positionnés soit verticalement avec un sens de lecture du haut vers le bas, soit horizontalement avec un sens de lecture de la gauche vers la droite.

Pour exemple, voici le début d'une partie à deux joueurs Alice et Bob :

1. Alice forme le mot **BINAIRE** en utilisant les 7 lettres de son chevalet puis repioche 7 lettres ;
2. Bob forme le mot **WEB** en utilisant 2 lettres de son chevalet et la lettre B de **BINAIRE** puis repioche 2 lettres ;
3. Alice forme le mot **LANGAGE** en utilisant 6 lettres de son chevalet et le A de **BINAIRE** puis repioche 6 lettres ;
4. Bob forme le mot **MEMOIRE** en utilisant 6 lettres de son chevalet et le E de **BINAIRE** puis repioche 6 lettres.

La figure 1 montre le plateau du jeu à l'issue de ces quatre tours.

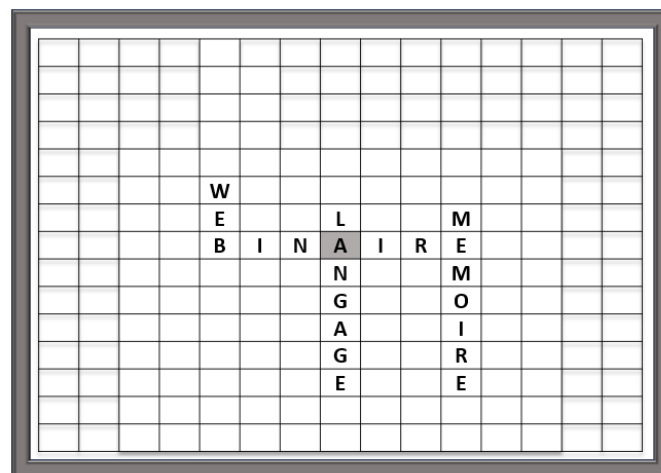


FIGURE 1 – Exemple de plateau

Le but de l'exercice est de modéliser un jeu de Scrabble en Python.

### Partie 1 : Formation des mots

Les mots acceptés selon les règles du Scrabble sont répertoriés dans un ouvrage spécifique à ce jeu, *L'officiel du Scrabble* (appelé plus couramment « ODS »). On suppose que tous les mots de cet ouvrage sont stockés dans une liste Python sous forme de chaînes de caractères.

Dans la suite de l'énoncé, on utilisera la convention suivante pour les arguments des fonctions :

► `liste_ODS` désigne une liste de chaînes de caractères représentant les mots acceptés au Scrabble.

Pour vérifier qu'un mot est autorisé ou non, on crée une fonction `est_autorise` qui prend en arguments `liste_ODS` et une chaîne de caractères `mot` et qui renvoie un booléen indiquant si `mot` est un mot autorisé selon `liste_ODS` ou non.

### Question 1

Indiquer sur la copie la ou les versions de la fonction `est_autorise` correspondant à la spécification demandée parmi les choix A, B, C, D ci-dessous.

Choix A :

```
1 def est_autorise(liste_ODS, mot) :
2     reponse = False
3     i = 0
4     while i < len(liste_ODS) and reponse == False :
5         if liste_ODS[i] == mot :
6             reponse = True
7             i = i + 1
8     return reponse
```

Choix B :

```
1 def est_autorise(liste_ODS, mot) :
2     reponse = False
3     i = 0
4     while i < len(liste_ODS) and reponse == False :
5         if liste_ODS[i] == mot:
6             reponse = True
7             i = i + 1
8     return reponse
```

Choix C :

```
1 def est_autorise(liste_ODS, mot) :
2     reponse = False
3     i = 0
4     while i < len(liste_ODS) or reponse == False :
5         if liste_ODS[i] == mot :
6             reponse = True
7             i = i + 1
8     return reponse
```

Choix D :

```
1 def est_autorise(liste_ODS, mot) :
2     reponse = False
3     i = 0
4     while i != len(liste_ODS) and reponse == False :
5         if liste_ODS[i] == mot :
6             reponse = True
7         else :
8             i = i + 1
9     return reponse
```

On rappelle que le **nombre d'occurrences** d'une lettre dans un mot est le nombre de fois que cette lettre apparaît dans ce mot. Par exemple, le nombre d'occurrences de la lettre 'E' dans le mot "EFFACE" est 2.

On souhaite écrire une fonction `table_occurrences` qui renvoie le nombre d'occurrences de chaque lettre de l'alphabet dans un mot donné. Le résultat est stocké dans une liste de 26 cases associées aux 26 lettres de l'alphabet et dont la valeur correspond au nombre d'occurrences de la lettre associée dans le mot. Par exemple `table_occurrences("EFFACE")` renvoie la liste suivante.

```
[1,0,1,0,2,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
```

On suppose qu'on dispose d'une fonction `indice` qui prend en paramètre une lettre et renvoie l'indice associé de la lettre dans la table d'occurrences. Ainsi, `indice('A')` vaut 0 et `indice('C')` vaut 2.

### Question 2

- Quelle est la table d'occurrences du mot "DECADE" ?
- Que vaut `table_occurrences("DECADE")[2]` ?

### Question 3

Écrire le code de la fonction `table_occurrences`.

On remarque qu'une telle liste permet aussi de modéliser un chevalet, c'est-à-dire un lot de lettres disponibles. Par exemple, `chevalet1 = [2,1,1,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]` modélise un chevalet avec deux 'A', un 'B', un 'C' et trois 'D'. Avec ces lettres on peut par exemple former le mot "BAC" mais pas le mot "BABA".

### Question 4

- Peut-on former le mot "CACHE" avec le chevalet modélisé par la liste suivante ?  
`[2,0,2,0,1,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]`
- Peut-on former le mot "ADA" avec le chevalet modélisé par la liste suivante ?  
`[1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]`

Dans la suite de l'énoncé, on utilisera les conventions suivantes pour les arguments des fonctions :

- ▶ `chevalet` désigne une liste de 26 entiers représentant la table d'occurrences d'un chevalet ;
- ▶ `mot` désigne une chaîne de caractères représentant un mot à former ou à placer.

On souhaite écrire une fonction `sont_dispo` qui prend en arguments `chevalet` et `mot` et qui teste si les lettres nécessaires pour former `mot` sont disponibles (en quantité suffisante) sur `chevalet`. Ainsi la fonction doit renvoyer `True` si les lettres sont disponibles et `False` sinon. Par exemple, `sont_dispo(chevalet1, "BAC")` renvoie `True` et `sont_dispo(chevalet1, "USB")` renvoie `False`.

### Question 5

Écrire le code de la fonction `sont_dispo`.

## Partie 2 : Placement du premier mot

Dans cette deuxième partie, on s'intéresse à la programmation de fonctions pour placer le premier mot sur le plateau de jeu. Ce plateau est constitué de 225 cases réparties sur 15 lignes et 15 colonnes.

On modélise le plateau de jeu par une liste de 15 lignes, chaque ligne étant elle-même modélisée par une liste de 15 éléments de type `str`. Si la case correspondante est vide, la valeur de cet élément est la chaîne vide `' '`, sinon la valeur de cet élément est la chaîne réduite à la lettre qui couvre la case, par exemple `'A'`. Le plateau est donc une liste de listes initialisée comme suit.

```
p1_0 = [  
    [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '], #ligne 0  
    [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '], #ligne 1  
    ...  
    [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']] #ligne 14
```

On numérote chaque ligne et chaque colonne de 0 à 14 en partant du coin supérieur gauche. Pour modéliser le fait de placer une lettre sur le plateau de jeu située ligne  $n^o i$  et colonne  $n^o j$ , il faut modifier l'élément situé à l'indice  $j$  de la ligne située à l'indice  $i$ . Ainsi pour placer la lettre `'A'` sur la case située à l'intersection de la ligne dont le numéro est 2 et de la colonne dont le numéro est 5 du plateau `p1_0` on effectue l'instruction suivante : `p1_0[2][5] = 'A'`.

### Question 6

On suppose qu'une lettre `'P'` se situe sur la case en haut à gauche du plateau de jeu, ce qui signifie que `p1_0[0][0]` vaut `'P'`.

- Quelles sont les instructions à effectuer pour placer le mot `"PYTHON"` horizontalement à partir de cette lettre `'P'` ?
- Quelles sont les instructions à effectuer pour placer le mot `"NET"` verticalement à partir de la dernière lettre du mot `"PYTHON"` ?

### Question 7

En considérant que le plateau `p1_1` est vide au départ (c'est-à-dire initialisé comme `p1_0` ci-avant), quel est l'état du plateau `p1_1` après les instructions ci-contre ? Pour répondre, compléter la grille donnée page 2 du sujet à la manière de la figure 1, c'est-à-dire en indiquant seulement les lettres et non les `' '`.

**Pensez à détacher la feuille du sujet, à y indiquer vos nom et prénom, et à la joindre à votre copie.**

```
1 mot = "WEBINAIRE"  
2 for k in range(3) :  
3     p1_1[1+k][11] = mot[k]  
4     p1_1[2][3+k] = mot[k]  
5     p1_1[7][7-k] = mot[2-k]  
6 for k in range(7) :  
7     p1_1[2][5+k] = mot[2+k]  
8     p1_1[1+k][6] = mot[2+k]
```

Dans la suite de l'énoncé, on utilisera les conventions suivantes pour les arguments des fonctions :

- `pos` désigne un tuple représentant la position de la première lettre du mot à poser ; le premier élément de `pos` est le numéro de ligne et le deuxième le numéro de colonne ;
- `lg` désigne un entier représentant la longueur du mot à placer.

On souhaite maintenant écrire une fonction `ne_depasse_pas_v` qui prend en arguments `pos` et `lg`, et qui renvoie `True` si on peut placer un mot de longueur `lg` verticalement à partir de la position `pos` sans dépasser du plateau (de taille standard  $15 \times 15$ ) et `False` sinon. Pour cela, on pourra s'inspirer de la fonction `ne_depasse_pas_h` ci-après, qui prend en entrée les mêmes arguments et qui effectue cette vérification pour un placement horizontal.

```
1 def ne_depasse_pas_h(pos, lg) :  
2     l, c = pos  
3     return (0 <= c) and (c + lg <= 15)
```

### Question 8

Écrire le code de la fonction `ne_depasse_pas_v`.

Une contrainte au premier tour est que le mot placé doit obligatoirement couvrir la case centrale du plateau (case grisée sur les figures).

### Question 9

Écrire une fonction `couvre_centre_h` qui prend en argument `pos` et `lg` et qui renvoie **True** si on couvre la case centrale du plateau en posant un mot de longueur `lg` horizontalement à partir de la position `pos`, et **False** sinon.

Dans la suite, on suppose que la fonction `couvre_centre_v` est définie de manière analogue pour un placement vertical.

Dans la suite de l'énoncé, on utilisera les conventions suivantes pour les arguments des fonctions :

- ▶ `p1` désigne une liste de 15 listes représentant le plateau du jeu.
- ▶ `d` désigne un chaîne de caractères valant **"vertical"** ou **"horizontal"** et représentant l'orientation du mot à placer.

On souhaite finalement écrire une fonction `est_valide_1` qui prend en argument `p1`, `pos`, `mot` et `d`, et qui renvoie **True** si on peut placer , au premier tour, le mot `mot` à la position `pos` et dans la direction `d` sur le plateau `p1` (c'est-à-dire si le mot ainsi placé ne dépasse pas du plateau et couvre bien la case centrale du plateau) et **False** sinon.

### Question 10

Écrire le code de la fonction `est_valide_1`.

Afin de pouvoir effectuer le premier coup, on souhaite finalement écrire une fonction `place_mot_h` qui prend en argument `p1`, `pos` et `mot` et qui place le mot `mot` horizontalement à la position `pos` sur le plateau `p1`, **seulement si ce placement est valide en tant que premier coup de la partie.**

### Question 11

Écrire le code de la fonction `place_mot_h`.

Dans la suite, on suppose que la fonction `place_mot_v` est définie de manière analogue pour un placement vertical.

## Partie 3 : Placement des mots suivants

On travaillera dans cette partie sur les fonctions permettant de placer les mots après le premier tour, c'est-à-dire une fois qu'au moins un mot a déjà été posé sur le plateau. Il faut alors que le nouveau mot formé utilise au moins une lettre déjà sur le plateau et au moins une nouvelle lettre.

Dans la suite de l'énoncé, on utilisera la convention suivante pour les arguments des fonctions :

- ▶ `p1` pour le plateau, `pos` pour la position, `d` pour la direction ;
- ▶ `mot` est une chaîne de caractères représentant le mot à placer sur le plateau.

Pour poser un mot, on doit s'assurer qu'on ne crée pas un nouveau mot plus long engendré par des lettres en périphérie du mot. On dit que le mot à poser doit être **sans extension**. Considérons le plateau `p1_2` représenté sur la figure 2 pour illustrer cette notion.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0															
1													L		
2													O		
3		D											G		
4		O											I		
5		S			W								C		
6		S	I	T	E			L			M	A	I	L	
7		I			B	I	N	A	I	R	E		E		
8		E						N			M		L		
9		R			B	U	G				O				
10					U		A				I				
11					R		G				R	A	M		
12					E		E				E				
13					A		S								
14					U										

FIGURE 2 – Représentation du plateau p1\_2

- Alice ne peut pas placer le mot "CLE" en position (12,8) horizontalement. En effet ce mot possède une extension à gauche du fait de la présence de la lettre 'E' du mot "LANGAGES".
- De même Bob ne peut pas placer le mot "RESEAU" en position (12,4) horizontalement car ce mot possède une extension à droite du fait de la présence de la lettre 'E' du mot "MEMOIRE".
- En revanche, Alice peut poser le mot "CLE" en position (1,11) horizontalement, car il n'y a pas de lettre juste à gauche du "C" qui sera à la position (1,11) ni de lettre juste à droite du "E" qui sera à la position (1,13).

### Question 12

Écrire le code de la fonction `sans_extension_h` qui prend en arguments `pl`, `pos` et `mot`, et qui renvoie un booléen indiquant si le mot `mot` est sans extension horizontale.

On considérera que la fonction `sans_extension_v` est également créée.

Une autre vérification à faire avant de poser le mot consiste à contrôler que les cases qui vont contenir les lettres du mot à poser sont, soit vides, soit occupées par la bonne lettre déjà présente. Par exemple, sur le plateau de la Figure 2, en position (12,4) direction horizontale :

- Alice peut poser le mot "FENETRE" car la lettre 'E' déjà placée en (12,5), la lettre 'E' déjà placée en (12,7) et la lettre 'E' déjà placée en (12,10) sur le plateau correspondent aux trois lettres 'E' dans le mot "FENETRE" à placer.
- En revanche, Alice ne peut pas poser le mot "TEST" en (12,4) horizontalement car se trouve en (12,6) un 'E' alors que pour poser le mot "TEST", il faut un 'T'.

### Question 13

Écrire le code de la fonction `coincide_h` qui prend en arguments `pl`, `pos` et `mot` et qui renvoie un booléen indiquant si en plaçant `mot` horizontalement en position `pos` sur le plateau `pl`, les lettres de ce mot coïncident avec les lettres déjà posées sur le plateau.

Dans la suite, on suppose que la fonction `coincide_v` est définie de manière analogue pour un placement vertical.

Nous pouvons désormais vérifier qu'un mot coïncide avec le plateau (pour une position et une direction données). Il faut maintenant identifier les lettres qu'il faut effectivement poser sur le plateau

(par opposition aux lettres déjà placées sur le plateau) et à quelles positions celles-ci doivent être placées. Toujours avec le plateau de la Figure 2, si Alice veut placer le mot "SPAMS" en position (13,3) horizontalement, les lettres qu'elle placera effectivement sont 'S' en position (13,3), 'P' en position (13,4) et 'M' en position (13,6). Les lettres 'A' et 'S' sont déjà placées sur le plateau.

On souhaite donc écrire une fonction `lettres_a_poser_h` qui prend en arguments `pl`, `pos` et `mot` et qui renvoie la liste des tuples représentant les lettres à poser effectivement (pour former horizontalement `mot` sur `pl` à partir de la position `pos`) ainsi que leur position. Plus précisément :

- Le 1<sup>er</sup> élément de chaque tuple est une chaîne de caractères `lettre` de longueur 1 indiquant la lettre à poser ;
- le 2<sup>ème</sup> élément de chaque tuple est lui-même un tuple `pos` indiquant la position de la case où poser `lettre`.

Par exemple, `lettres_a_poser_h(pl_2, (13,3), "SPAMS")` renvoie la liste suivante.  
[ ('S', (13,3)), ('P', (13,4)), ('M', (13,6)) ].

#### Question 14

Que renvoie l'exécution de `lettres_a_poser_h(pl_2, (4,9), "WIFI")` ?

#### Question 15

Écrire le code de la fonction `lettres_a_poser_h`.

Dans la suite, on suppose que la fonction `lettres_a_poser_v` est définie de manière analogue pour un placement vertical.

Dans suite de l'énoncé, on utilise la convention suivante pour les arguments des fonctions :

- `liste_lp` désigne une liste de tuples composés des lettres à poser suivies de leur position.

Afin de pouvoir utiliser la fonction `sont_dispo` écrite à la question 5, on souhaite écrire une fonction `extrait_lettres` qui prend en argument `liste_lp` et qui renvoie la chaîne de caractères formée par toutes les lettres de `liste_lp`.

Par exemple, `extrait_lettres([('S', (13,3)), ('P', (13,4)), ('M', (13,6))])` renvoie "SPM".

#### Question 16

Écrire le code de la fonction `extrait_lettres`.

Lorsqu'on place un mot horizontalement sur le plateau, il se peut que des chaînes de caractères (non réduites à une lettre) soient créées perpendiculairement à ce mot. Il est important de pouvoir lister ces chaînes annexes créées pour vérifier qu'elles correspondent à des mots autorisés. Par exemple, sur le plateau de la figure 2, si Bob plaçait le mot "DEBIT" en position (12,9) horizontalement, deux chaînes annexes seraient créées verticalement : "AB" et "MI", or "AB" n'est pas autorisé.

En fait, pour un placement horizontal, il s'agit de trouver, pour chaque lettre posée, la chaîne de caractères maximale créée verticalement, et dans le cas où cette chaîne a au moins deux lettres de vérifier que c'est un mot autorisé.

Dans la suite de l'énoncé, on utilisera la convention suivante pour les arguments des fonctions :

- `lettre` est une chaîne de caractères à un seul caractère.

On souhaite dans un premier temps écrire une fonction `chaine_cree_v` qui prend en arguments `pl`, `lettre`, et `pos`, et qui renvoie la chaîne de caractères maximale créée verticalement à partir de `lettre` posée à la position `pos` sur `pl`. Si aucune lettre n'est voisine verticalement de cette lettre posée en position `pos`, la fonction renvoie la chaîne réduite à `lettre`. Par exemple :

- `chaine_cree_v(pl_2, "I", (12,9))` renvoie "I";
- `chaine_cree_v(pl_2, "A", (12,11))` renvoie "AA";
- `chaine_cree_v(pl_2, "E", (8,6))` renvoie "NEU".

### Question 17

Que renvoient `chaine_cree_v(pl_2, "I", (12,12))` et `chaine_cree_v(pl_2, "S", (6,14))` ?

### Question 18

Recopier et compléter le code de la fonction `chaine_cree_v` ci-dessous.

```

1  def chaine_cree_v(plateau, pos, lettre) :
2      l,c = pos
3      # Identifier les bornes de la chaîne créée
4      ind_deb = l
5      while (ind_deb-1 >= 0) and ... :
6          ind_deb ...
7      ind_fin = ...
8      while ...
9          ...
10     # Former la chaîne créée
11     chaine = ""
12     for ...
13         ...
14     chaine += lettre
15     for ...
16         ...
17     return chaine

```

On souhaite maintenant écrire une fonction `liste_chaines_creées_v` qui prend en arguments `pl` et `liste_lp` et qui renvoie la liste des chaînes de caractères, y compris celles de longueur 1, créées lors du placement des lettres de `liste_lp`.

Avec `llp = [("D", (12,9)), ("B", (12,11)), ("I", (12,12)), ("T", (12,13))]` par exemple, l'appel `liste_chaines_creées_v(pl_2, (12,9), llp)` retourne `['D', 'AB', 'MI', 'T']`.

### Question 19

Écrire le code de la fonction `liste_chaines_creées_v`.

Dans la suite, on suppose que la fonction `liste_chaines_creées_h` est définie de manière analogue pour les chaînes de caractères créées horizontalement lors d'un placement vertical.

Il est désormais possible d'écrire une fonction `est_valide_2` permettant de s'assurer qu'un joueur peut former le mot qu'il propose sur le plateau de jeu, horizontalement et à la position qu'il propose.

### Question 20

Écrire le code de la fonction `est_valide2_h` qui prend en argument `liste_ODS`, `chevalet` (voir partie 1), `pl`, `pos` et `mot` et qui réalise ce test. Toutes les vérifications n'ont pas nécessairement été évoquées dans les questions précédentes.