Challenges Python

(Discord « Docstring »)

@bucdany

2023

Table des matières

Avant propos

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Challenge N° 1

Compter le nombre de voyelles

1.1 Énoncé

Ce premier challenge est très simple, il est de niveau « débutant », mais si vous avez plus d'expérience, vous pouvez essayer de trouver de belles astuces pour un code propre, rapide et concis.

Ici, il va nous falloir créer une fonction `nb_voyelles(phrase: str)->int` qui retourne le résultat du nombre total de voyelles dans une phrase passée en paramètre.

Conditions

- Les voyelles sont : `aeiou`, `y` n'est pas pris en compte.
- Les voyelles accentuées ne sont pas prises en compte.
- La phrase passée en paramètre doit être écrite en minuscule.
- Une chaîne vide, passée en paramètre, doit renvoyer `0`.

Exemples

- `nb_voyelles("bonjour, comment allez-vous ?")` doit retourner `9`.
- `nb_voyelles("je vais à paris")` doit retourner `5`.
- `nb_voyelles("docstring")` doit retournet `2`.
- `nb_voyelles("")` doit retourner `0`.

1.2 Solution et explications

Voici donc ma solution [1] :

```python
def nb_voyelles(phrase: str)->int:
    return sum(phrase.count(el) for el in "aeiou")
```

- La phrase doit toujours être en minuscule, donc pas besoin de la méthode `lower()`.
- Ici, on compte chaque voyelle dans la phrase à l'aide de la méthode `count`.
- La fonction `sum()` renvoie ensuite la somme du résultat obtenu.
- Si l'on passe un générateur ou une liste de compréhension dans la fonction `sum()`, la paire de crochets supplémentaire peut-être éliminée [2]. De cette manière :

  `sum([phrase.count(el) for el in "aeiou"])`

  est l'équivalent de :

  `sum(phrase.count(el) for el in "aeiou")`

Voici aussi le code pour mes tests unitaires :

```python
import pytest

@pytest.mark.parametrize("sentence, expected", [
    ("", 0),
    ("docstring", 2),
    ("bonjour comment allez-vous ?", 9),
    ("je vais à paris", 5),
    ("vas-y !", 1),
])
def test_should_return_the_sum(sentence, expected):
    got = nb_voyelles(sentence)
    assert got == expected
```

`@OsKaR31415` a par ailleurs apporté plusieurs solutions pour résoudre ce challenge [3].

1. Fil de discussion de ce challenge: https://discord.com/channels/396825382009044994/1142617945139335189
2. Attention, car cela n'est par contre pas compatible avec la fonction `len()`. Pour plus d'information on se reportera au PEP-289: https://peps.python.org/pep-0289/#the-details.
3. https://discord.com/channels/396825382009044994/1142617945139335189/1144591818516860998

Challenge N° 2

Jeu du « Pierre - Papier - Ciseaux »

2.1 Énoncé

On va jouer un peu en développant un petit jeu très simple.

Le but du challenge est de développer le célèbre jeu « *pierre - papier - ciseaux* »[1] en essayant de trouver un algorithme astucieux et un code à la fois simple, propre et efficace.

Étapes

1. Générer un choix aléatoire pour votre session de jeu : « pierre », « papier » ou « ciseaux ».
2. Demander au joueur d'écrire son choix entre trois propositions : « pierre », « papier » ou « ciseaux ».
3. Afficher qui a gagné en dévoilant le choix aléatoire du point n°1.

Conditions

- L'affichage, le prompt et la réponse seront affichées par écrit sur la console.
- Le fonctionnement du jeu est simple : la pierre gagne sur les ciseaux, les ciseaux gagnent sur le papier, le papier gagne sur la pierre, deux éléments identiques correspondent à une égalité.
- Toutes les chaînes de caractères, « pierre », « papier » et « ciseaux » doivent toujours être entrées en minuscule, le joueur devra donc écrire correctement ces mots, sinon vous devrez lui demander de redéfinir son choix.
- S'il y a égalité, vous devrez relancer automatiquement votre programme (en regénérant un nouveau choix aléatoire pour la nouvelle session de jeu), jusqu'à ce qu'il y ait un gagnant à la partie.

1. https://fr.wikipedia.org/wiki/Pierre-papier-ciseaux

Exemples

- Le choix aléatoire donne « pierre » et le joueur a choisi « papier » -> `Vous avez gagné ! Le papier enveloppe la pierre`
- Le choix aléatoire donne « ciseaux » et le joueur a choisi « papier » -> `Vous avez perdu ! Les ciseaux coupent le papier`
- Le choix aléatoire donne « pierre » et le joueur a choisi « pierre » -> `Égalité ! Recommencez...`

2.2 Ma solution

```
from random import randint

BDD = {
    "element": ["papier", "pierre", "ciseaux"],
    "gagnant": ["10", "21", "02"],
    "phrase": ["Le papier enveloppe la pierre",
               "Les ciseaux coupent le papier",
               "La pierre casse les ciseaux"]
}

while True:
    joueur = input("pierre, papier ou ciseaux: ")
    if joueur in BDD["element"]:
        choix, joueur = (randint(0, 2),
                         BDD["element"].index(joueur))

        if choix != joueur: break
        print("Ègalité, recommencez...")

    else: print("Faute de frappe !")

print(f'Vous avez {"gagné" if f"{choix}{joueur}" in BDD["g
agnant"] else "perdu"} ! {BDD["phrase"][choix + joueur - 1]
} !')
```

2.3 Explication de l'algorithme

Contexte

- Trois éléments : « pierre », « papier » ou « ciseaux ».
- Un choix aléatoire fait par la machine et un joueur qui entre son choix au clavier.

Il suffit donc de réfléchir à un algorithme sympathique pour présenter le code de manière élégante et éviter bien sûr les répétitions.

Rangement des données

Chaque élément est rangé dans cet ordre particulier dans la liste, afin de les faire correspondre en triade logique.

- papier = index(0)
- pierre = index(1)
- ciseaux = index(2)

Algorithme pour un choix triangulaire

Si on fait l'addition `0+1`, on obtient `1`, alors le jeu se fait entre « papier » et « pierre ». On cherche ensuite la phrase dans `phrase` en faisant juste un calcul grâce à la somme `-1` des deux éléments. Donc en index: `1-0 = 0`, et on trouve donc la chaîne de caractères `"Le papier enveloppe la pierre"`.

Si on fait l'addition `1+2`, on obtient `3`, alors le jeu se fait entre la « pierre » et les « ciseaux ». Donc en index, on obtient `3-1`, soit `2`. On trouve donc la chaîne de caractères `"La pierre casse les ciseaux"`.

De la même façon, si on fait l'addition `2+0`, on obtient `2` et le jeu se fait entre les « ciseaux » et le « papier ». En index cela donne `2-1`, soit `1`, et on tombe sur la chaîne de caractères `"Les ciseaux coupent le papier"`.

The Winner is...

Pour connaître qui gagne, il suffit de convertir en *string* et de joindre les deux caractères d'index du choix et du joueur. Ainsi, `10` dans `gagnant` veut dire que le choix aléatoire donne la « pierre » (index `1`) et que le joueur a saisi le « papier » (index `0`). La « pierre » contre le « papier » fait donc gagner le joueur.

On affiche ainsi `"gagné"` puis la phrase qui suit s'obtient grâce à l'index de la liste `phrase` de la `BDD`, calculée par l'addition des deux index: `1+(0-1)`, ce qui nous donne `0`, ce qui correspond à la chaîne de caractères `"Le papier enveloppe la pierre"`.

De la même façon pour `21` et `02`, cela représente la combinatoire complète des choix gagnants pour le joueur par rapport au choix aléatoire.

Conclusion

On utilise le calcul de la somme `-1` qui renvoie un objet de type `int` et qui pemret l'association des deux chaînes de caractères (*string*) pour connaître le gagnant.

Challenge N° 3

Couleur complémentaire

3.1 Énoncé

Pour ce challenge j'ai choisi un niveau intermédiaire. Les débutants pourront cependant résoudre la première étape en s'aidant de librairies.

Le but de ce challenge est de trouver la **couleur complémentaire** [1]

Étapes

1. Créer la fonction `get_color_types(color:str)->dict` qui permet de convertir le format *RVB hexadécimal* d'une couleur au format *RVB décimal* et *TSL* [2] (anglais : ***H**ue**L**ight**S**aturation*).
   - **color : [string]** : la couleur RVB codée en hexadécimal, envoyée en paramètre.
   - **dict : [dict]** : contient le résultat de la conversion en différents styles d'écriture, contenant les clés et valeurs suivantes :

     hex : [str] : valeur hexadécimale de la couleur passée en paramètre.

     rvb : [list] : valeurs de chaque éléments RVB en décimal.

     tsl_norm : [tuple] : valeurs de chaque élément TSL (teinte en degrés (360°), saturation et luminosité en pourcentage).

     tsl : [tuple] : valeurs de chaque élément TSL (teinte, saturation et luminosité au format [0-1], soit de type `float`).

2. Afficher le contenu du dictionnaire retourné par cette fonction.

3. Créer la fonction `get_complementary(color:str)->str` pour trouver la couleur complémentaire et la retourne au format hexadécimal.

1. https://fr.wikipedia.org/wiki/Couleur_complÃ¯mentaire
2. https://fr.wikipedia.org/wiki/Teinte_saturation_lumiÃ¯re

Conditions

- L'affichage se fera via la console.
- Les valeurs hexadécimales sont précédées du symbole « # » et les lettres sont en minuscules.

Exemples

```
- get_color_types("#19021e") -> {'hex': '#19021e', 'rvb': [25, 2, 30],
                                 'tsl_norm': ('289°', '88%', '6%'),
                                 'tsl': (0.8035714285714285, 0.875,
                                         0.06274509803921569)}
- get_complementary("#19021e") -> "#071e02"
```

Ressource

Vous pouvez vous aider du site `colorpicker`[3] pour vos tests.

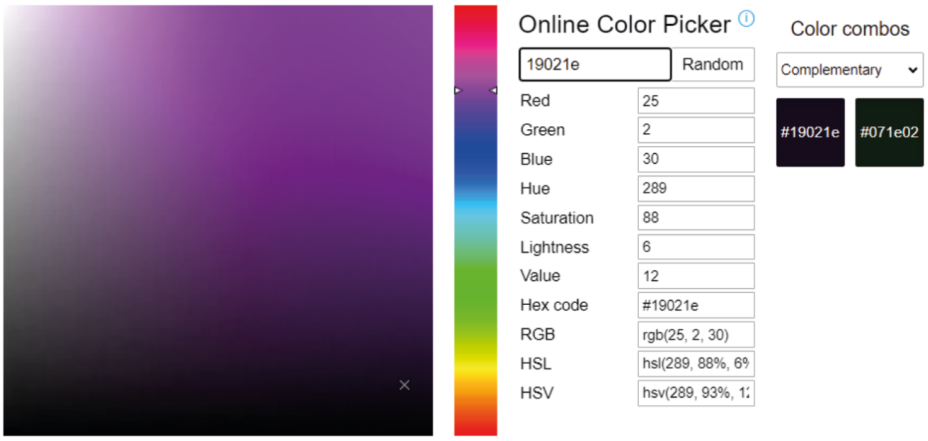

FIGURE 3.1 – `colorpicker`

Indices

- Le nombre hexadécimal d'une couleur représente ses valeurs *RougeVertBleu* codées avec six nombres. Les deux premiers correspondent à la couleur rouge, les deux suivants au vert et les deux derniers au bleu. Pour transformer cette valeur hexadécimale en décimal, il vous suffit de convertir chacune des paires de ce nombre.
- Vous pouvez vous aider de la librairie `colorsys` pour vous permettre de réaliser facilement les conversions.
- Ce sont principalement les fonctions `colorsys.rgb_to_hls` et `colorsys.hls_to_rgb` qui pourront être utilisées.

3. https://colorpicker.me/#00ee7b

- Pour chercher la complémentaire d'une couleur, il faut passer par le format *TSL* en faisant une rotation de 180° sur la teinte et trouver ainsi la position de la couleur diamétralement opposée.
- Pour faire une rotation, il faudra bien sûr penser à normaliser la valeur de la teinte *TSL* qui par défaut est de `[0, 1]` en `[0°, 360°]` puis faire la rotation en additionnant avec 180°.

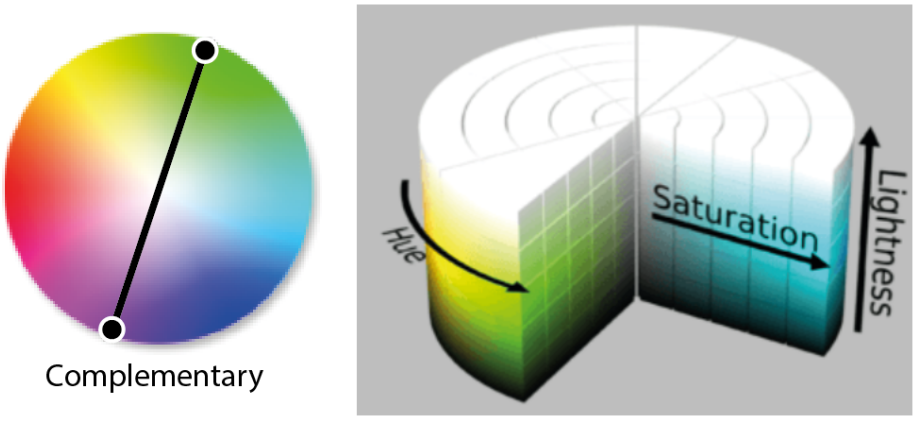

FIGURE 3.2 – Couleur complémetaire et TSL

3.2 Mon code

```python
from colorsys import rgb_to_hls, hls_to_rgb


def get_color_types(color: str) -> dict:
    rvb = [int(color[i:i+2], 16) for i in
           range(1, len(color), 2)]
    tsl = list(rgb_to_hls(*list(map(lambda x: x/255,
                                    rvb))))
    tsl.insert(-1, tsl.pop())
    tsl_norm = (f"{round(tsl[0]*360)}°",
                *[f"{el:.0%}" for el in tsl[1:3]])
    return {"hex": color, "rvb": rvb,
            "tsl_norm": tsl_norm, "tsl": tuple(tsl)}


def get_complementary(color: str) -> str:
    t, s, l = get_color_types(color)["tsl"]
    return f'#{"".join([f"{round(el*255):02x}" for el
                        in hls_to_rgb(t+.5 % 1, l, s)])}'
```

3.3 Explications

Contexte

Il s'agit soit de faire tous les calculs à la main pour les conversions entre les formats de couleur, soit trouver une bibliothèque qui nous permet de nous simplifier la vie. Mon choix s'est porté sur la bibliothèque `colorsys` qui est plutôt sympathique pour cela. Elle nous permet de passer au format `TSL` via `RGB` et vice et versa.

Le but est aussi de trouver la couleur complémentaire, qui est la couleur diamétralement opposée sur le cercle chromatique. Pour cela `TSL`, nous permet de faire une rotation et trouver directement cette valeur sans difficulté particulière.

Différentes problématiques

- **Le dièze « # »... gênant hein !** : il y a donc plusieurs façon de faire, soit un simple `lstrip()` et hop viré, soit débuter son index à `1`.
- **Conversion d'un format hexadécimal à un format décimal pour le `RVB`** : un simple `int(..., 16)` nous fait le calcul directement !
- **Normaliser ses valeurs** : avec la bibliothèque `colorsys`, attention à toujours rester dans un domaine de valeurs comprises entre `0` et `1`, comme stipulé dans la documentation officielle de `colorsys`[4].
- **`HLS` vs `TSL`** : attention à bien penser à inverser la saturation de la luminosité si vous utilisez la bibliothèque `colorsys`.
- ***Tuple*, *liste*** : l'énoncé précise de bien respecter les différents types dans le dictionnaire renvoyé par la fonction.
- **Utilisation des fonctions déjà crées** : on a déjà développé la fonction `get_color_type()` alors pourquoi ne pas l'utiliser pour créer la fonction `get_complementary()` afin d'avoir directement la valeur `TSL`.
- **Trouver la valeur diamétralement opposée** : une rotation de `180°` ou de `0.5` dans la plage de limites [`0 - 1`] puis un *modulo* de `1` à appliquer pour rester dans cette plage de valeurs.

Explication du code et astuces d'optimisation / *refactoring*

```
1 rvb = [int(color[i:i+2], 16) for i in
2             range(1, len(color), 2)]
```

On pointe avec un `range` sur la chaîne de caractères `color` avec un index de `1` jusqu'à sa taille totale suivant un pas de `2`, puis avec `[i:i+2]` on vient chercher deux par deux les caractères hexadécimaux de chaque couleur à convertir en décimal.

4. https://docs.python.org/3/library/colorsys.html : « the coordinates are all between 0 and 1... »

```
tsl = list(rgb_to_hls(*list(map(lambda x: x/255, rvb))))
```

Pour constituer notre `TSL`, on utilise la fonction `colorsys.rgb_to_hls()`. J'ai volontairement utilisé la fonction `map()` pour vous permettre de voir de nouvelles choses en plus de la compréhension de liste. Ceci `list(map(lambda x: x/255, rvb))` est la même chose que cela : `[x/255 for x in rvb]`. C'est juste une autre façon de faire ! Donc on vient diviser chaque couleur `R`, `V` et `B` par `255` pour respecter la norme `0` à `1` qui sera nécessaire pour l'argument de la fonction `rgb_to_hls()`. Notez l'astérisque (`*`) qui permet de faire un *unpacking* pour dispatcher les éléments `RVB` en trois arguments, la fonction `rgb_to_hls()` nécessitant trois arguments pour fonctionner. Et l'on convertit en `list()` car nous avons ensuite besoin d'intervertir les deux derniers éléments, luminosité et saturation étant inversés par rapport à l'énoncé du challenge.

```
tsl.insert(-1, tsl.pop())
```

C'est la phrase magique qui permet d'inverser les places des deux derniers éléments de la liste. `pop()` permet de retirer le dernier élément, `insert()` permet d'insérer ce que `pop()` renvoie à l'index `-1`, qui est l'avant dernière position... `-1` étant la dernière position mais l'insertion se passe juste avant la position indiquée, donc juste avant `-1`. La liste `tsl` sera alors modifiée directement.

```
tsl_norm = (f"{round(tsl[0]*360)}°",
                *[f"{el:.0%}" for el in tsl[1:3]])
```

On utilise le résultat de la conversion `TSL` pour obtenir `TSL_norm`:

- La teinte est définie en degrés, comprise dans une limite entre 0 et 360°. `round()` permet d'arrondir pour avoir une valeur sans chiffre après la virgule.
- La luminosité et la saturation quant à elles sont exprimées en pourcentage grâce à l'utilisation d'une *f-string* qui avec « `%` » permet à la fois de multiplier par `100` et d'afficher le pourcentage. Le `.0` permet de supprimer les chiffres après la virgule. `tsl[1:3]` c'est la lecture des élément `1` et `2` de la liste `tsl`. On utilise encore l'astérisque (`*`) pour avoir en tout trois arguments pour `tsl_norm`.

```
return {"hex": color, "rvb": rvb,
            "tsl_norm": tsl_norm, "tsl": tuple(tsl)}
```

Juste un simple `return` pour renvoyer un dictionnaire. Notez le `tuple()` qui permet de convertir notre précédente liste en tuple afin de suivre l'énoncé du challenge.

```
t, s, l = get_color_types(color)["tsl"]
```

On envoie le résultat *TSL* du dictionnaire de la fonction précédente dans les variables `t`, `s` et `l`. Il est important d'utiliser `tsl` plutôt que `tsl_norm` afin de travailler sur des valeurs non arrondies et avoir ainsi plus de précision.

```
return f'#{"".join([f"{round(el*255):02x}" for el
                        in hls_to_rgb(t+.5 % 1, l, s)])}'
```

Ce `return` peut paraître un peu complexe mais tout deviendra simple une fois décomposé :

`t+.5 % 1`

On effectue une rotation de la teinte de `180°` mais comme on se trouve dans un espace compris entre 0 à 1, alors cela devient `180/360 = 1/2 = 0.5`. Pour rester dans la norme, on fait un *modulo* `1`.

`hls_to_rgb(t+.5 % 1, l, s)`

On convertit en *RVB* via *TSL*, avec la teinte, la luminosité et la saturation.

`[f"{round(el*255):02x}" for el in hls_to_rgb(t+.5 % 1, l, s)]`

Cette compréhension de liste nous fournit le *RVB* au format hexadécimal. Il faut pour ça changer la norme et passer de `[0-1]` à `[0-255]` en multipliant par `255` puis convertir en hexadécimal via `02x`, ce qui nous donne deux chiffres (les valeurs de 0 à 9 seront écrites à l'aide de deux chiffres : 01 02 03 04... 09), `x` permet la conversion grâce à une *f-string*.

```
f'#{"".join([f"{round(el*255):02x}" for el
                        in hls_to_rgb(t+.5 % 1, l, s)])}'
```

`"".join()` permet de réunir le résultat de chaque paire *RVB*/*hexadécimal* et enfin le « `#` », la cerise sur le gâteau !

Mon fichier d'*unitests*

```
import pytest

@pytest.mark.parametrize("hexcode, expected", [
    ("#19021e", {'hex': '#19021e', 'rvb': [25, 2, 30],
                 'tsl_norm': ('289°', '88%', '6%'),
                 'tsl': (0.8035714285714285, 0.875,
                         0.06274509803921569)}),
    ("#aedd5f", {'hex': '#aedd5f', 'rvb': [174, 221, 95],
```

```
                    'tsl_norm': ('82°', '65%', '62%'),
                    'tsl': (0.2288359788359884,
                            0.6494845360824743,
                            0.6196078431372549)}),
    ("#ff0000", {'hex': '#ff0000', 'rvb': [255, 0, 0],
                    'tsl_norm': ('0°', '100%', '50%'),
                    'tsl': (0.0, 1.0, 0.5)})
])
def test_different_colour_writing_formats(
        hexcode, expected):
    got = get_color_types(hexcode)

    assert got == expected

@pytest.mark.parametrize("hexcode, expected", [
    ("#19021e", "#071e02"),
    ("#aedd5f", "#8e5fdd"),
    ("#ff0000", "#00ffff"),
])
def test_for_complementary_color_in_hex(
        hexcode, expected):
    got = get_complementary(hexcode)
```

Quelques conseils

- Faire fonctionner son code le plus tôt possible en évitant les problèmes.
- Coder simplement pour obtenir un premier jet et optimiser/refactoriser par la suite
- Ne pas hésiter pas à s'aider de bibliothèques, mais il est possible de réaliser tous les calculs à la main pour s'entraîner si on le souhaite.
- Le but n'est pas d'obtenir le code le plus court, mais un code lisible et optimisé avec un algorithme efficace. Penser qu'un code n'est pas seulement pour son usage personnel, surtout si on travaille en *open source*, donc veiller a ce qu'il soit clair. Ne pas hésiter à ajouter des commentaires, cela nous servira tout autant quand nous reverrons notre code trois mois plus tard sans y avoir touché !

Challenge N° 4

Calculatrice romaine

4.1 Énoncé

Le but de ce challenge est de développer la fonction:

`add_romans(calculate:str)->str`

...qui permet de calculer la somme de deux ou plusieurs nombres écrits en chiffres romains [1].

Conditions

- L'affichage du prompt se fera via la console.
- L'opération est simple : deux ou plusieurs nombres écrits sur une seule ligne en chiffres romains séparés par un signe `+`.
- Le résultat de l'opération sera bien sûr donné en chiffres romains dans une plage comprise entre `I` et `MMMCMXCIX`.
- Vérifier que les chiffres romains et l'opération entrés par l'utilisateur sont valides. Les lettres doivent toujours êtres écrites en majuscules.
- Les chiffres romains sont : `IVXLCDM`.
- `I`, `X` et `C` ne peuvent pas être répétés plus de trois fois.
- `V`, `L` et `D` ne peuvent pas apparaître plus d'une fois.

Exemples

```
- X + IV -> XIV
- MXCIX + CIV -> MCCIII
- II + II -> IV
- MMMCX + DCII -> MMMDCCXII
```

1. https://fr.wikipedia.org/wiki/Num%C3%A9ration_romaine)

- MCMXCIX + I -> MM
- IVM + I -> erreur (IVM n'est pas valide)
- VVI -> erreur (2x V n'est pas valide)
- XXXX -> erreur (4x X n'est pas valide)
- TX -> erreur (T ne fait pas partie de la liste des chiffres romains)
- XX - IV -> erreur (seule l'addition est permise)
- IV -> erreur (2 éléments minimum)
- MMMCMXC + X -> erreur (débordement, doit respecter la plage de I à MMMCMXCIX inclus)

Ressource

Voir le site `Roman numerals calculator` pour réaliser des tests[2].

4.2 Ma proposition

```python
from re import search

ROMANS = {"I": 1, "V": 5, "X": 10, "L": 50,
          "C": 100, "D": 500, "M": 1000}
SPEC = {4: ("IV", "XL", "CD"), 9: ("IX", "XC", "CM")}
list_romans = list(ROMANS.keys())


def to_dec(roman_nb: str) -> int:
    nb = [ROMANS[el] for el in roman_nb]
    return sum([nb[i] - 2 * nb[i - 1] if nb[i] > nb[i - 1]
                else nb[i] for i in
                range(1, len(nb))]) + nb[0]


def n_to_9(nb: int, lv: int) -> str:
    base, v = list_romans[lv * 2], list_romans[1:6:2][lv]
    return f"{base * nb}" if nb < 4 else v if nb == 5 \
        else SPEC[nb][lv] if nb in SPEC \
        else f"{v}{base * (nb - 5)}"


def to_roman(nb: int) -> str:
    nb = [int(i) for i in reversed(str(nb))]
    return "".join([f"{'M' * nb[3]}" if i == 3
                    else n_to_9(nb[i], i) for i in
                    range(0, len(nb))][::-1])
```

2. https://www.hackmath.net/en/calculator/roman-numerals

```
28
29
30  def add_romans(calculate: str) -> str:
31      if [el for el in "-*/" if el in calculate]:
32          return "Seule l'addition est permise"
33
34      el_list = calculate.replace(" ", "").split("+")
35      if len(el_list) < 2:
36          return "2 éléments minimum"
37
38      calc = []
39      for el in el_list:
40          err = [lt for lt in el if lt not in ROMANS]
41          if err:
42              return (f"{err[0][0]} ne fait pas partie de "
43                      f"la liste des chiffres romains")
44
45          if search("IL|IC|ID|IM|XD|XM|VL|VC|VD|VM", el):
46              return f"{el} n'est pas valide"
47
48          err = search("V.*?V|L.*?L|D.*?D", el)
49          if err:
50              return f"2x {err[0][0]} n'est pas valide"
51
52          err = search(
53              rf'([{"".join(list_romans[0:5:2])}])\1{{3,}}',
54              el)
55          if err:
56              return f"4x {err[0][0]} n'est pas valide"
57
58          err = search(
59              rf'({"|".join(SPEC[4] + SPEC[9])}).*?\1', el)
60          if err:
61              return f"2x {err[0]} n'est pas valide"
62
63          calc.append(to_dec(el))
64      result = sum(calc)
65      return to_roman(result) if result < 4000 else \
66          ("Débordement, doit respecter la plage de I à "
67           "MMMCMXCIX inclus")
68
69
70  if __name__ == '__main__':
71      print(add_romans(input("une opération : ")))
```

Mon fichier d'*unitests*

```python
import pytest

@pytest.mark.parametrize("calculate, expected", [
    ("X + IV", "XIV"),
    ("MXCIX + CIV", "MCCIII"),
    ("II + II", "IV"),
    ("MMMCX + DCII", "MMMDCCXII"),
    ("MCMXCIX + I", "MM"),
    ("IVM + I", "IVM n'est pas valide"),
    ("VVI + I", "2x V n'est pas valide"),
    ("XXXX + I", "4x X n'est pas valide"),
    ("TX + I", "T ne fait pas partie de la liste des "
               "chiffres romains"),
    ("XX - IV", "Seule l'addition est permise"),
    ("IV", "2 éléments minimum"),
    ("MMMCMXC + X", "Débordement, doit respecter la plage "
                    "de I à MMMCMXCIX inclus"),
])
def test_should_return_the_sum_of_roman_numbers(calculate,
                                                expected):
    got = add_romans(calculate)

    assert got == expected
```

4.3 Explications

Contexte

Il s'agit de trouver un moyen pour additionner des chiffres romains entre eux. Pour cela, il est préférable de rester dans un domaine de nombres arabes pour effectuer l'opération arithmétique puis reconvertir la sommes finales en chiffres romains.

Les différentes problématiques

- **Conversion chiffres romains en chiffres arabes :** Il faut trouver une astuce pour faire correspondre ces chiffres romains en chiffres arabes. La solution est dans cette phrase : « *Un nombre écrit en chiffres romains se lit de gauche à droite. En première approximation, sa valeur se détermine en faisant la somme des valeurs individuelles de chaque symbole, sauf quand l'un des symboles précède un symbole de valeur supérieure ; dans ce cas, on soustrait la valeur du premier symbole au deuxième* »[3].

3. Source `Wikipédia`: https://fr.wikipedia.org/wiki/NumÃ©ration_romaine

- **Conversion du résultat en chiffres romains :** Il y a plusieurs façons de faire, soit faire correspondre chaque symbole romain, soit reconstituer le nombre par dizaine, centaine et millier.
- **Gestion des erreurs :** Pour ce challenge, il s'agissait de simplement suivre la liste dans les exemples, car plusieurs écritures sont possible en fonction de la version romaine utilisée. Dans ce cas, faire un simple `return` du texte est accepté, sinon faire l'habituel `raise` pour *lever l'erreur*.

Explication du code

Mon programme se divise donc en trois, voire en quatre parties :

La conversion des chiffres romains en chiffres arabes

Avec la fonction :

`to_dec(roman_nb:str)->int`

Ici, de simples additions tant qu'on ne rencontre pas de paires contenant un élément de valeur supérieur, auquel cas, il faudra réaliser une soustraction. La conversion s'opère principalement par ces lignes :

```
1 def to_dec(roman_nb: str) -> int:
2     nb = [ROMANS[el] for el in roman_nb]
3     return sum([nb[i] - 2 * nb[i - 1] if nb[i] > nb[i - 1]
4                 else nb[i] for i in
5                 range(1, len(nb))]) + nb[0]
```

La conversion des chiffres arabes en chiffres romains

Avec la fonction :

`to_roman(nb:int)->str`

Là, c'est plus compliqué. Ici je décompose mon nombre en unité, dizaine, centaine et millier et m'occupe de chaque éléments un par un. Les éléments particulier 4, 5 et 9 sont traités dans une sous-fonction :

```
1 def n_to_9(nb: int, lv: int) -> str:
2     base, v = list_romans[lv * 2], list_romans[1:6:2][lv]
3     return f"{base * nb}" if nb < 4 else v if nb == 5 \
4         else SPEC[nb][lv] if nb in SPEC \
5         else f"{v}{base * (nb - 5)}"
```

Le code de la fonction est le suivant :

```python
def to_roman(nb: int) -> str:
    nb = [int(i) for i in reversed(str(nb))]
    return "".join([f"{'M' * nb[3]}" if i == 3
                    else n_to_9(nb[i], i) for i in
                    range(0, len(nb))][::-1])
```

La fonction principale

`add_romans(calculate:str)->str`

Corps du programme dans lequel la gestion des erreurs se fait à l'aide principalement des *expressions régulières* (*regex*).

```python
def add_romans(calculate: str) -> str:
    if [el for el in "-*/" if el in calculate]:
        return "Seule l'addition est permise"

    el_list = calculate.replace(" ", "").split("+")
    if len(el_list) < 2:
        return "2 éléments minimum"

    calc = []
    for el in el_list:
        err = [lt for lt in el if lt not in ROMANS]
        if err:
            return (f"{err[0][0]} ne fait pas partie de "
                    f"la liste des chiffres romains")

        if search("IL|IC|ID|IM|XD|XM|VL|VC|VD|VM", el):
            return f"{el} n'est pas valide"

        err = search("V.*?V|L.*?L|D.*?D", el)
        if err:
            return f"2x {err[0][0]} n'est pas valide"

        err = search(
            rf'([{"".join(list_romans[0:5:2])}])\1{{3,}}',
            el)
        if err:
            return f"4x {err[0][0]} n'est pas valide"

        err = search(
            rf'({"|".join(SPEC[4] + SPEC[9])}).*?\1', el)
        if err:
            return f"2x {err[0]} n'est pas valide"

```

```
34          calc.append(to_dec(el))
35      result = sum(calc)
36      return to_roman(result) if result < 4000 else \
37          ("Débordement, doit respecter la plage de I à "
38           "MMMCMXCIX inclus")
```

Notes sur les *regexes* utilisées

Le *regex* est un langage à part entière, il permet de chercher des occurrences plus ou moins complexes dans un bloc de texte. C'est grâce à la bibliothèque `re` que l'on peut l'intégrer facilement à *Python*.

```
search("IL|IC|ID|IM|XD|XM|VL|VC|VD|VM", el)
```

`search` cherche soit `IL`, `IC`, `ID`, `IM`... le caractère « | » représente le « `or` ».

```
search("V.*?V|L.*?L|D.*?D", el)
```

Cela représente n'importe quel caractère (`*?` signifie « *jusqu'à* ». Donc de `V` jusqu'à `V`, ce qui permet d'indiquer que deux `V` peuvent être présents. De même à l'aide de « or » pour chercher aussi deux `L` et deux `D`.

```
search(rf'([{"".join(list_romans[0:5:2])}])\1{{3,}}', el)
```

`list_romans[0:5:2]` correspond à `["I", "X", "C"]`. On réalise ici un `.join()`, ce qui nous donne `IXC`. Ce `IXC` est entre parenthèses car nous faisons ainsi un groupe qui enregistre le caractère trouvé entre `I` ou `X` ou `C`. Le `\1` permet de récupérer ce groupe et recherche alors les deux mêmes caractères qui se suivent. `{3,}` veut dire : répéter au moins trois ou plus. Donc en tout, on cherche la répétition du même caractère quatre fois de suite (une fois dans le premier groupe, puis trois fois à l'aide de `\1`).

```
search(rf'({"|".join(SPEC[4]+SPEC[9])}).*?\1', el)
```

On cherche si `SPEC[4]` est dans `("IV", "XL", "CD")` ou si `SPEC[9]` est dans ("IX", "XC", "CM"), et s'ils sont présents deux fois. Pour information, le `r` avant le `f` du *f-string* permet d'ignorer le caractère d'échappement « \ », car sans ce `r` il aurait fallu écrire « \\1 » plutôt que « \1 ».

4.4 En complément

- A noter deux vidéos existantes sur le sujet présentes sur `Youtube`: *Convertir des nombres entiers en chiffres romains*[4] et *Convertir des chiffres romains en nombres entiers*[5].
- *Comment vérifier si une liste est vide en Python ?*[6]

Maintenant je pense que pour vous, c'est très facile de lire cette date, n'est-ce pas?

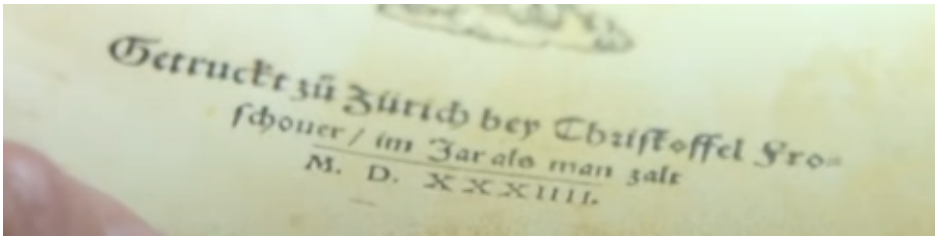

FIGURE 4.1 – Extrait de la première page d'une bible imprimée

4. https://www.youtube.com/watch?v=fD9aw0dZtjc
5. https://www.youtube.com/watch?v=WFyrryNO9Nk
6. https://flexiple.com/python/check-if-list-is-empty-python

Challenge N° 5

Affichage digital 7 segments

5.1 Énoncé

Le but de ce challenge est de développer la fonction `digital_time()->str` qui retourne l'heure actuelle au format d'affichage digital à sept segments [1].

Étapes

1. Transcrire les chiffres de 0 à 9 en sept segments digitaux.
2. Transcrire plusieurs chiffres à la suite sur une seule ligne.
3. Afficher l'heure en la retranscrivant.

Conditions

- L'affichage se fait via la console
- L'heure doit être affichée au format 24h.
- L'heure et les minutes sont séparées par deux points, le tout sur une même ligne d'affichage avec deux espaces de séparation.
- Le résultat de l'affichage de l'heure actuelle doit correspondre aux exemples ci-dessous à l'aide du caractère « `#` ».
- Les sept segments `a`, `b`, `c`, `d`, `e`, `f`, `g` sont les suivants :

1. https://fr.wikipedia.org/wiki/Affichage_%C3%A0_sept_segments

```
     a
     v
    ####
f-> #  # <- b
    #### <- g
e-> #  # <- c
    ####
     ^
     d
```

- Pour des raisons d'ergonomie et d'esthétisme, les segments se superposent et s'affichent de manière plus allongée, ainsi le chiffre `2` s'affichera de la façon suivante:

```
             ####                          ##
                #                            #
comme ceci : ####   et pas comme cela :  ##
             #                           #
             ####                          ##
```

Exemples

- À `17h26`, le résultat de `digital_time()` donne :

```
#  ####     ####  ####
#     #  #     #  #
#     #     ####  ####
#     #  #  #     #  #
#     #     ####  ####
```

- À `23h59`, le résultat de `digital_time()` donne :

```
####  ####     ####  ####
   #     #  #  #     #  #
####  ####     ####  ####
#        #  #     #     #
####  ####     ####  ####
```

- À `4h08`, le résultat de `digital_time()` donne : -

```
#  #     ####  ####
#  #  #  #  #  #  #
####     #  #  ####
   #  #  #  #  #  #
   #     ####  ####
```

5.2 Ma proposition avec l'utilisation de Numpy

```
from numpy import hstack, delete, full, ndarray
from datetime import datetime

SEG_MAP = (((0, 0), (0, 1), (0, 2), (0, 3)),  # a
           ((0, 3), (1, 3), (2, 3)),  # b
           ((2, 3), (3, 3), (4, 3)),  # c
           ((4, 0), (4, 1), (4, 2), (4, 3)),  # d
           ((4, 0), (3, 0), (2, 0)),  # e
           ((2, 0), (1, 0), (0, 0)),  # f
           ((2, 0), (2, 1), (2, 2), (2, 3)),  # g
           ((1, 0), (3, 0)))


def display(el: str) -> ndarray:
    digit = full((5, 6), [" "], dtype=str)
    match el:
        case " ": return digit
        case ":":
            s = [*[0]*7, 1]
            digit = delete(digit, slice(3), 1)
        case _:
            b = [int(b) for b in f"{int(el):04b}"][::-1]
            s = (b[3] or b[1] or b[0] and b[2] or not b[0]
                 and not b[2],
                 b[0] and b[1] or not b[1] and not b[0]
                 or not b[2],
                 b[2] or b[3] or not b[1] or b[0],
                 b[3] or b[1] and not b[0] or not b[2]
                 and not b[0] or not b[1] and b[0] and b[2]
                 or b[1] and not b[2],
                 b[1] and not b[0] or not b[0]
                 and not b[2],
                 b[3] or not b[0] and b[2] or not b[1]
                 and b[2] or not b[1] and not b[0],
                 b[3] or b[1] and not b[0] or not b[1]
                 and b[2] or not b[2] and b[1])
    digit[*zip(*[(r, c) for i, el in enumerate(s) if el
                 for r, c in SEG_MAP[i]])] = "#"
    return digit


def digital_time():
    time = datetime.now().strftime("%H:%M").lstrip("0")
    return "\n".join("".join(r) for r in
                     hstack([display(el) for el in
```

```
46                                  (time if len(time) == 5 else
47                                   " " + time)]))
48
49
50 if __name__ == "__main__":
51     print(digital_time())
```

5.3 Explications

Contexte

Il s'agit de trouver le moyen d'afficher un nombre sous forme digitale en sept segments comme on peut le voir sur certaines horloges numériques, puis d'afficher l'heure actuelle.

Différentes problématiques

- **Conversion d'un chiffre en sept segments pour simuler un affichage digital**: Il s'agit de repérer la position de chacun des segments puis « d'allumer » les segments spécifiques au chiffre à afficher.
- **Afficher plusieurs chiffres sur une ligne**: L'étape suivante est d'afficher plusieurs chiffres sur une même ligne en transposant les « matrices » des chiffres à afficher.
- **Ajouter le séparateur heure/minute**: Il nous faut donc ajouter le caractère « : » entre les heures et les minutes.
- **Formatage particulier**: Les heures de 0 à 9 ne doivent pas afficher un zéro mais laisser l'espace d'un *digit* à gauche.

Explication du code

Le code se divise en trois, voire quatre parties :

- **Définir les segments à allumer en fonction du chiffre à afficher**

Ma méthode ici consiste à chercher des solutions combinatoires pour chaque segment en fonction du chiffre à afficher. Les chiffres sont au nombre de 10 et peuvent varier de 0 à 9. Ce qui nécessite quatre *bits* pour les coder en binaire :

```
0000 -> 0
0001 -> 1
0010 -> 2
...
1000 -> 8
1001 -> 9
```

Il y a ensuite sept solutions combinatoires à rechercher pour chacun des segments de « `a` » à « `g` ».

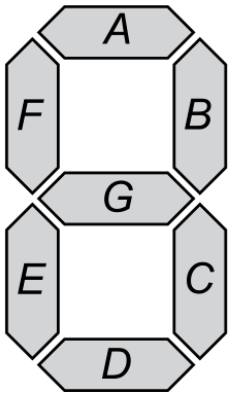

FIGURE 5.1 – Représentation visuelle des sept solutions combinatoires

Ainsi il faut allumer les segments `B` et `C` pour coder le chiffre `1`, `BCFG` pour le chiffre `4`, `ABCDEF` pour le chiffre `0`, etc. On peut donc poser tous ces résultat dans une *table de vérité* :

	b3	b2	b1	b0	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

FIGURE 5.2 – *Table de vérité*

A gauche les chiffres transcrits en quatre *bits* de `b0` à `b3` et à droite les segments à allumer.

La table de vérité est incomplète dans le sens où on n'utilise pas la totalité des possibilités en quatre *bits*. Ainsi, on pourra être libre de choisir n'importe quels états durant la simplification de nos équations à l'aide des *tableaux de Karnaugh* (ces combinaisons seront notées en rouge dans les tableaux).

Le chiffre est d'abord converti en binaire sur quatre *bits* pour procéder au calcul avec l'*algèbre de Boole* :

```
b = [int(b) for b in f"{int(el):04b}"][::-1]
```

Pour faire bien correspondre à mes tableaux je suis obliger de faire une inversion via `[::-1]`.

Pour simplifier chacune des sept solutions on posera nos *tableaux de Karnaugh*[2] de cette manière :

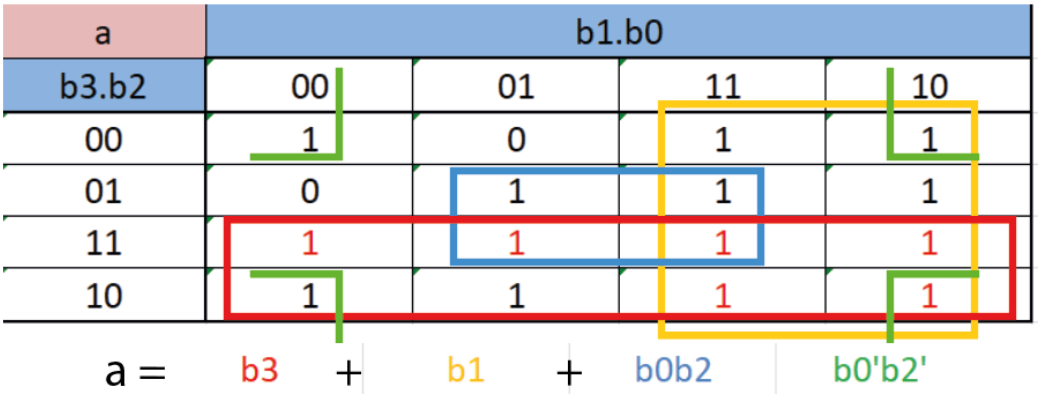

FIGURE 5.3 – *Tableau de Karnaugh* - 1

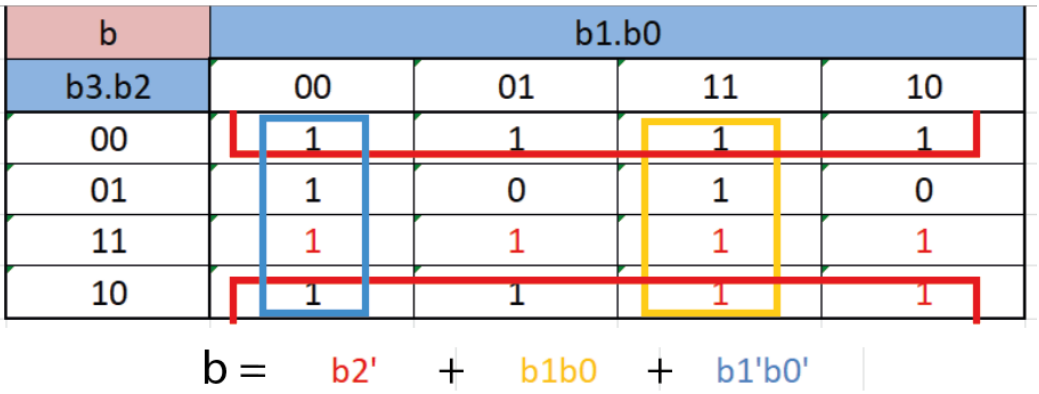

FIGURE 5.4 – *Tableau de Karnaugh* - 2

2. *Tableaux de Karnaugh* - simplification d'expression en *algèbre de Boole*: https://www.youtube.com/watch?v=2tULBk6V9ZE

c	b1.b0			
b3.b2	00	01	11	10
00	1	1	1	0
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

c = b1' + b0 + b2 + b3

De Morgan

FIGURE 5.5 – *Tableau de Karnaugh* - 3

d	b1.b0			
b3.b2	00	01	11	10
00	1	0	1	1
01	0	1	0	1
11	1	1	1	1
10	1	1	1	1

d = b1b0' + b2'b0' + b1'b0b2 + b3 + b1b2'

FIGURE 5.6 – *Tableau de Karnaugh* - 4

e	b1.b0			
b3.b2	00	01	11	10
00	1	0	0	1
01	0	0	0	1
11	1	1	1	1
10	1	0	1	1

e = b0'b2' + b1b0'

FIGURE 5.7 – *Tableau de Karnaugh* - 5

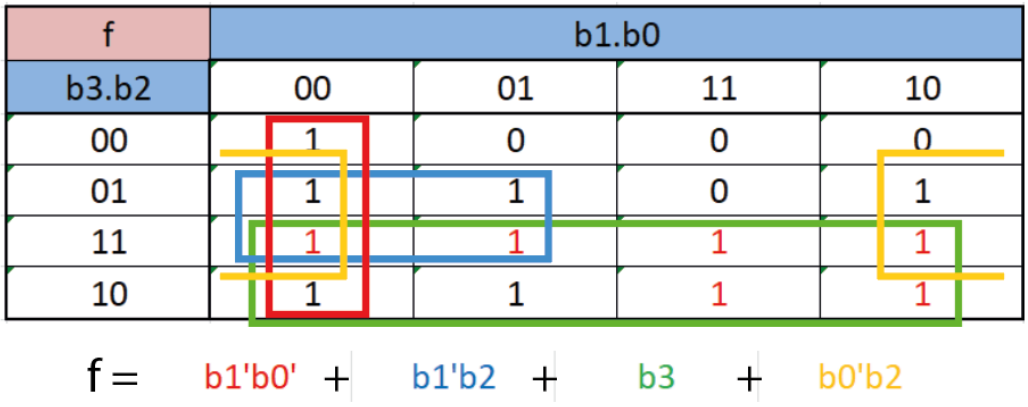

FIGURE 5.8 – *Tableau de Karnaugh* - 6

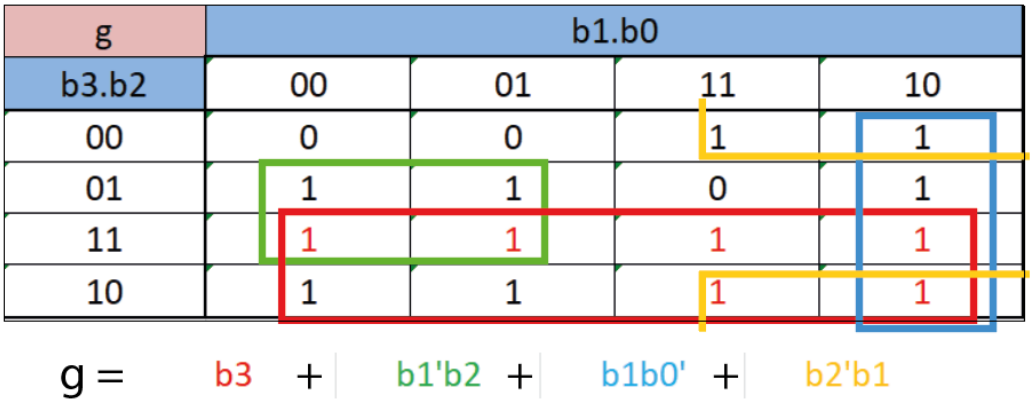

FIGURE 5.9 – *Tableau de Karnaugh* - 7

On peut donc poser les solutions sous forme de listes qu'on pourra interpréter respectivement via les index de `0` à `6` pour les segments de `a` à `g`.

```
s = (b[3] or b[1] or b[0] and b[2] or not b[0] and not
     b[2],
     b[0] and b[1] or not b[1] and not b[0] or not b[2],
     b[2] or b[3] or not b[1] or b[0],
     b[3] or b[1] and not b[0] or not b[2] and not b[0]
     or not b[1] and b[0] and b[2] or b[1] and not b[2],
     b[1] and not b[0] or not b[0] and not b[2],
     b[3] or not b[0] and b[2] or not b[1] and b[2]
     or not b[1] and not b[0],
     b[3] or b[1] and not b[0] or not b[1] and b[2]
     or not b[2] and b[1])
```

Puis la liste `SEG_MAP` me permet de fixer la position des segments en posant leur coordonnées dans la matrice d'un chiffre digital :

```
SEG_MAP = (((0, 0), (0, 1), (0, 2), (0, 3)), # a
           ((0, 3), (1, 3), (2, 3)), # b
```

```
            ((2, 3), (3, 3), (4, 3)), # c
            ((4, 0), (4, 1), (4, 2), (4, 3)), # d
            ((4, 0), (3, 0), (2, 0)), # e
            ((2, 0), (1, 0), (0, 0)), # f
            ((2, 0), (2, 1), (2, 2), (2, 3)), # g
            ((1, 0), (3, 0))) # :
```

Notez que la dernière ligne correspond au caractère de séparation « : » entre l'heure et les minutes.

- **Affichage d'un *digit***

La variable `digit` est un tableau *Numpy* initialisé avec des espaces :

```
digit = full((5, 6), [" "],dtype=str)
```

Je teste ici avec `match case` les caractères à afficher, soit `" "` pour un *digit* « éteint », puis les deux points (« : ») pour le séparateur et le reste pour les chiffres de 0 à 9.

```
match el:
        case " ": ...
        case ":":
            ...
        case _:
            ...
```

Une fois les solutions de ma table de vérité trouvées, alors on récupère les coordonnées pour afficher un « # » à chaque fois pour former les segments à allumer :

```
digit[*zip(*[(r, c) for i, el in enumerate(s) if el
                    for r, c in SEG_MAP[i]])] = "#"
    return digit
```

Cette fonction peut être éclatée de cette manière pour plus de lisibilité :

```
pos = [(r, c) for i, el in enumerate(s) if el
              for r, c in SEG_MAP[i]]
r, c = zip(*pos)
digit[r, c] = "#"
```

Ce n'est qu'une simple transposition afin de réunir les coordonnées `x` et `y` et les appliquer d'un coup et ainsi remplir le tableau *Numpy* `digit`.

- **Afficher sur une ligne**

Pour ce faire on doit procéder à une lecture ligne par ligne de chaque `digit`. On utilise pour cela la fonction `hstack()` [3] qui permet de concaténer ligne par ligne mes tableaux *Numpy*.

Tout se fait à cette ligne :

```
hstack([display(el) for el in time])
```

- **Afficher l'heure en suivant l'énoncé**

L'affichage de l'heure n'est pas bien complexe: heure sur un chiffre et les minutes avec deux chiffres. Il faut penser à effacer le premier zéro de l'heure si l'heure est entre 0 et 9 puis d'ajouter un *digit* blanc pour l'affichage.

```
1 time = datetime.now().strftime("%H:%M").lstrip("0")
2 ...
3 ...time if len(time) == 5 else " "+time...
```

Pour cela je cherche la taille de l'heure et si l'on a que cinq caractères alors on se trouve dans le cas d'un affichage avec deux *digits* pour l'heure sinon ce sera une longueur de quatre :

- 3:24 <- une longueur de quatre
- 15h58 <- une longueur de cinq

Le caractère spécial « : » est calculé au sein de la fonction `display()` en ajoutant une solution supplémentaire pour afficher les positions de mon « deux-points » à la fin de la liste `SEG_MAP`.

3. https://numpy.org/doc/stable/reference/generated/numpy.hstack.html