

# Challenges Python

(Discord « Docstring »)



@bucdany

2023

# Table des matières

<b>1</b>	<b>Compter le nombre de voyelles</b>	<b>5</b>
1.1	Énoncé . . . . .	5
1.2	Solution et explications . . . . .	6
<b>2</b>	<b>Jeu du « Pierre - Papier - Ciseaux »</b>	<b>7</b>
2.1	Énoncé . . . . .	7
2.2	Ma solution . . . . .	8
2.3	Explication de l'algorithme . . . . .	8
<b>3</b>	<b>Couleur complémentaire</b>	<b>11</b>
3.1	Énoncé . . . . .	11
3.2	Mon code . . . . .	13
3.3	Explications . . . . .	14



## Avant propos

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur 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 retourner 2.
- `nb_voyelles("")` doit retourner 0.

## 1.2 Solution et explications

Voici donc ma solution<sup>1</sup> :

```
1 def nb_voyelles(phrase: str) -> int:
2     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<sup>2</sup>. 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 :

```
1 import pytest
2
3 @pytest.mark.parametrize("sentence, expected", [
4     ("", 0),
5     ("docstring", 2),
6     ("bonjour comment allez-vous ?", 9),
7     ("je vais à paris", 5),
8     ("vas-y !", 1),
9 ])
10 def test_should_return_the_sum(sentence, expected):
11     got = nb_voyelles(sentence)
12     assert got == expected
```

@OsKaR31415 a par ailleurs apporté plusieurs solutions pour résoudre ce challenge<sup>3</sup>.

---

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* »<sup>1</sup> 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 régénérant 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

```
1 from random import randint
2
3 BDD = {
4     "element": ["papier", "pierre", "ciseaux"],
5     "gagnant": ["10", "21", "02"],
6     "phrase": ["Le papier enveloppe la pierre",
7               "Les ciseaux coupent le papier",
8               "La pierre casse les ciseaux"]
9 }
10
11 while True:
12     joueur = input("pierre, papier ou ciseaux: ")
13     if joueur in BDD["element"]:
14         choix, joueur = (randint(0, 2),
15                         BDD["element"].index(joueur))
16
17         if choix != joueur: break
18         print("Égalité, recommencez...")
19
20     else: print("Faute de frappe !")
21
22 print(f'Vous avez {"gagné" if f"{choix}{joueur}" in BDD["gagnant"] 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 permet 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 bibliothèques.

Le but de ce challenge est de trouver la **couleur complémentaire**<sup>1</sup>

#### 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*<sup>2</sup> (anglais : *HueLightSaturation*).
  - **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\\_complémentaire](https://fr.wikipedia.org/wiki/Couleur_complémentaire)

2. [https://fr.wikipedia.org/wiki/Teinte\\_saturation\\_luminance](https://fr.wikipedia.org/wiki/Teinte_saturation_luminance)

## 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](https://colorpicker.me/#00ee7b)<sup>3</sup> pour vos tests.

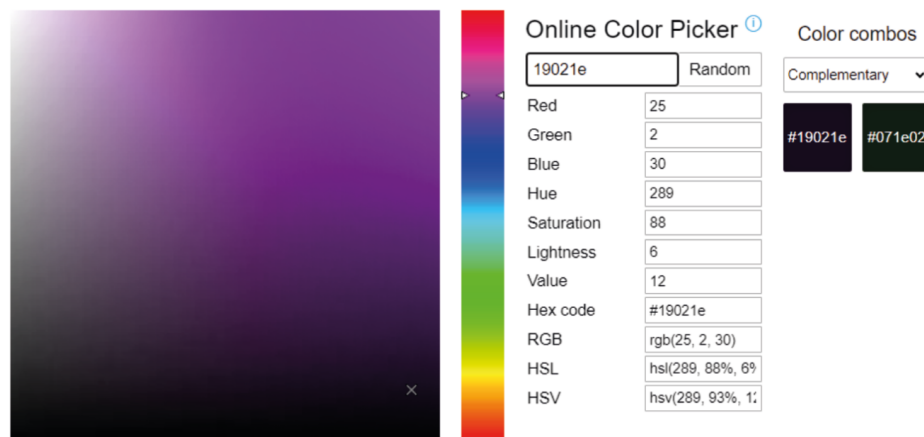


FIGURE 3.1 – colorpicker

## 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  $180^\circ$  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  $[0^\circ, 360^\circ]$  puis faire la rotation en additionnant avec  $180^\circ$ .

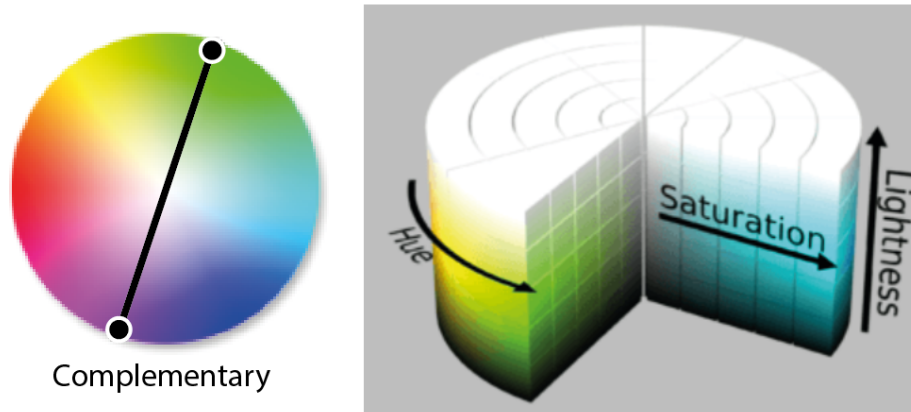


FIGURE 3.2 – Couleur complémentaire et TSL

## 3.2 Mon code

```

1 from colorsys import rgb_to_hls, hls_to_rgb
2
3
4 def get_color_types(color: str) -> dict:
5     rvb = [int(color[i:i+2], 16) for i in
6             range(1, len(color), 2)]
7     tsl = list(rgb_to_hls(*list(map(lambda x: x/255,
8                                     rvb))))
9     tsl.insert(-1, tsl.pop())
10    tsl_norm = (f"{round(tsl[0]*360)}°",
11               *[f"{el:.0%}" for el in tsl[1:3]])
12    return {"hex": color, "rvb": rvb,
13           "tsl_norm": tsl_norm, "tsl": tuple(tsl)}
14
15
16 def get_complementary(color: str) -> str:
17     t, s, l = get_color_types(color)["tsl"]
18     return f'#{"".join([f"{round(el*255):02x}" for el
19                           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 dièse « # »... gênant hein !** : il y a donc plusieurs façon de faire, soit un simple `lstrip()` et hop viré, soit débiter 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`<sup>4</sup>.
- **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 créées** : 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... »

```
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'invertir les deux derniers éléments, luminosité et saturation étant inversés par rapport à l'énoncé du challenge.

```
1 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.

```
1 tsl_norm = (f"{round(tsl[0]*360)}°",
2             *[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 éléments 1 et 2 de la liste `tsl`. On utilise encore l'astérisque (\*) pour avoir en tout trois arguments pour `tsl_norm`.

```
1 return {"hex": color, "rvb": rvb,
2         "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.

```
1 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.

```
1 return f'#{"".join([f"{round(el*255):02x}" for el
2                       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  $180^\circ$  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'*unittests*

```
1 import pytest
2
3 @pytest.mark.parametrize("hexcode, expected", [
4     ("#19021e", {'hex': '#19021e', 'rvb': [25, 2, 30],
5           'tsl_norm': ('289°', '88%', '6%'),
6           'tsl': (0.8035714285714285, 0.875,
7                 0.06274509803921569)}),
8     ("#aedd5f", {'hex': '#aedd5f', 'rvb': [174, 221, 95],
```

```

9         'tsl_norm': ('82°', '65%', '62%'),
10        'tsl': (0.22883597883597884,
11                0.6494845360824743,
12                0.6196078431372549)}) ,
13    ("#ff0000", {'hex': '#ff0000', 'rvb': [255, 0, 0],
14                'tsl_norm': ('0°', '100%', '50%'),
15                'tsl': (0.0, 1.0, 0.5)})
16])
17def test_different_colour_writing_formats(
18    hexcode, expected):
19    got = get_color_types(hexcode)
20
21    assert got == expected
22
23@pytest.mark.parametrize("hexcode, expected", [
24    ("#19021e", "#071e02"),
25    ("#aedd5f", "#8e5fdd"),
26    ("#ff0000", "#00ffff"),
27])
28def test_for_complementary_color_in_hex(
29    hexcode, expected):
30    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 à 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 !