

**OBJECTIFS** : L'objectif de ce TP est encore d'écrire des codes pythons. Comme le chapitre vu en cours est actuellement celui sur les nombres binaires et hexadécimaux, ces codes reprendront cette thématique.

**DOCUMENT A RENDRE** : Ce travail est évalué. Vous en rédigez un compte-rendu au format `.doc` ou `.odt` et vous le transférez en fin d'activité par l'intermédiaire de l'onglet **Mon Compte** du site <https://nsibranly.fr> en utilisant le code : **tp5** . Ce compte-rendu contiendra :

- les réponses aux différentes questions posées,
- les captures d'écran **des morceaux de codes** écrits **et** celles **des résultats des exécutions** données dans le shell. Pour faire ces captures, utiliser *l'Outil Capture d'écran* de Windows.

### 1. Code qui calcule le nombre de caractère d'une chaîne

On vous demande de coder une fonction que vous nommerez `nb_caractere()`. Elle prend en argument une chaîne de caractère et retourne le nombre de caractères qui composent cette chaîne. On donne ci-dessous un exemple d'exécution :

```
n = nbr_caractere("j'ai 10000 ans en base 2")
print(n)
```

Donne :

```
>>> (executing file "exercicel.py")
24
```

Pour vous aider, on donne le code incomplet de la fonction :

```
def nbr_caractere(chaine) :
    _____
    for c in chaine :
        _____
    return n
```

## 2. Code qui convertit un nombre de la base 2 vers la base 10

L'objectif de cet exercice est de coder une fonction que vous nommerez `conversion_binaire()`. Elle prend en argument un nombre en base 2 et le retourne en base 10. On donne ci-après, quelques exemples d'exécution :

Retour : nombre en base 10

Argument : nombre en base 2

```
n_dec = conversion_binaire(10)
print(n_dec)
```

donnera :

```
>>> (executing file "exercice2.py")
2
```

```
n_dec = conversion_binaire(11111111)
print(n_dec)
```

donnera :

```
>>> (executing file "exercice2.py")
255
```

Pour arriver à cet objectif, il faut bien sûr savoir comment on convertit un nombre binaire en base 10. N'ayant pas encore eu le temps de voir cela en cours, on le voit ici :

a- Conversion du nombre binaire  $n_{bin} = 11111111$  en base 10 :

Pour trouver  $n_{dec} = 255$ , on réalise le calcul suivant :

$$\begin{aligned}
 n_{bin} &= 11111111 \\
 n_{dec} &= 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\
 &= 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 \\
 &= 255
 \end{aligned}$$

b- Conversion du nombre binaire  $n_{bin} = 10$  en base 10 :

Pour trouver  $n_{dec} = 2$ , on réalise le calcul suivant :

$$\begin{aligned}
 n_{bin} &= 10 \\
 n_{dec} &= 1 \times 2^1 + 0 \times 2^0 \\
 &= 2 + 0 \\
 &= 2
 \end{aligned}$$

Pour convertir de la base 2 à la base 10 un nombre binaire à  $n$  chiffres, on prend le 1<sup>er</sup> chiffre que l'on multiplie par  $2^0 = 1$ , on l'ajoute au 2<sup>nd</sup> chiffre que l'on multiplie par  $2^1 = 2$ , puis on l'ajoute au 3<sup>ième</sup> chiffre que l'on multiplie par  $2^2 = 4$ , puis on l'ajoute au 4<sup>ième</sup> chiffre que l'on multiplie par  $2^3 = 8$ , ..., on l'ajoute au  $n$ <sup>ième</sup> chiffre que l'on multiplie par  $2^{n-1}$ .

- c- Utiliser cette méthode pour convertir le nombre binaire  $n_{bin} = 10001$  et retrouver  $n_{dec} = 17$ . Mettre le détail du calcul.
- d- Utiliser cette méthode pour convertir le nombre binaire  $n_{bin} = 1000101$  et retrouver  $n_{dec} = 69$ . Mettre le détail du calcul.
- e- Utiliser cette méthode pour convertir le nombre binaire  $n_{bin} = 1111100100$  et retrouver  $n_{dec} = 2020$ . Mettre le détail du calcul.
- f- S'inspirer de cette méthode pour écrire le code de la fct `conversion_binaire()`.  
Pour vous aider, 2 conseils :
- ⇒ le nombre  $n_{bin}$  pris en argument devra être converti en chaîne de caractère afin de pouvoir utiliser une boucle du genre : `for c in n_bin :`
  - ⇒ pour connaître le nombre de caractères composant le nombre  $n_{bin}$ , vous pourrez utiliser la fonction écrite dans l'exercice 1 précédent.

### 3. Code facile pour préparer la suite :

L'objectif de cet exercice est de coder une fonction que vous nommerez `affectation()`. Elle prend en argument un caractère correspondant à un **chiffre hexadécimal** égal à **0** ou **1** ou **2** ou ... **9** ou **a** ou **b** ou **c** ou **d** ou **e** ou **f** et retourne la valeur en base 10 de celui-ci.

On donne ci-dessous, quelques exemples d'exécution :

```
# Main *****  
val = affectation("7")  
print(val)
```

donnera :

```
>>> (executing file "exercice3.py")  
7
```

```
# Main *****  
val = affectation("d")  
print(val)
```

donnera :

```
>>> (executing file "exercice3.py")  
13
```

## 4. Code qui convertit un nombre de la base 16 vers la base 10

L'objectif de cet exercice est de coder une fonction que vous nommerez `conversion_hexa()`. Elle prend en argument un nombre en base 16 et le retourne en base 10. On donne ci-après, quelques exemples d'exécution :

Retour : nombre en base 10

Argument : nombre en base 16

```
# Main *****
n_dec = conversion_hexa("ff")
print(n_dec)
```

donnera :

```
>>> (executing file "exercice4.py")
255
```

```
# Main *****
n_dec = conversion_hexa("a1f8")
print(n_dec)
```

donnera :

```
>>> (executing file "exercice4.py")
41464
```

Pour arriver à cet objectif, il faut bien sûr savoir comment on convertit un nombre hexadécimal en base 10. N'ayant pas encore eu le temps de voir cela en cours, on le voit ici :

- a- Conversion du nombre hexadécimal  $n_{hex} = ff$  en base 10 :

Pour trouver  $n_{dec} = 255$ , on réalise le calcul suivant :

$$\begin{aligned}
 n_{hex} &= \mathbf{ff} \\
 n_{dec} &= \mathbf{15} \times 16^1 + \mathbf{15} \times 16^0 \\
 &= 15 \times 16 + 15 \times 1 \\
 &= \mathbf{255}
 \end{aligned}$$

- b- Conversion du nombre binaire  $n_{hex} = a1f8$  en base 10 :

Pour trouver  $n_{dec} = 41464$ , on réalise le calcul suivant :

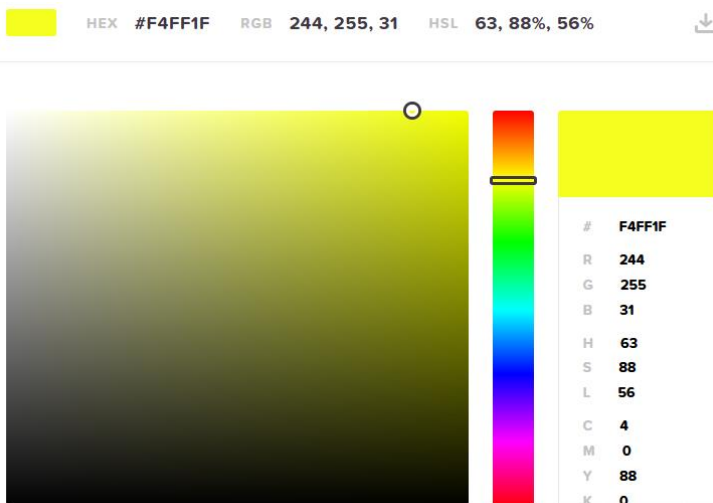
$$\begin{aligned}
 n_{hex} &= \mathbf{a1f8} \\
 n_{dec} &= \mathbf{10} \times 16^3 + \mathbf{1} \times 16^2 + \mathbf{15} \times 16^1 + \mathbf{8} \times 16^0 \\
 &= 10 \times 4096 + 1 \times 256 + 15 \times 16 + 8 \times 1 \\
 &= 40960 + 256 + 240 + 8 \\
 &= \mathbf{41464}
 \end{aligned}$$

Pour convertir de la base 16 à la base 10 un nombre hexadécimal à  $n$  chiffres, on prend le 1<sup>er</sup> chiffre que l'on multiplie par  $16^0 = 1$ , on l'ajoute au 2<sup>nd</sup> chiffre que l'on multiplie par  $16^1 = 16$ , puis on l'ajoute au 3<sup>ème</sup> chiffre que l'on multiplie par  $16^2 = 256$ , puis on l'ajoute au 4<sup>ème</sup> chiffre que l'on multiplie par  $16^3 = 4096$ , ..., on l'ajoute au  $n$ <sup>ème</sup> chiffre que l'on multiplie par  $16^{n-1}$ .

- c- Réaliser manuellement la conversion du nombre  $n_{hex} = 45$  en base 10 afin de trouver comme résultat la valeur de  $n_{dec} = 69$
  
- d- Réaliser manuellement la conversion du nombre  $n_{hex} = 7e4$  en base 10 afin de trouver comme résultat la valeur de  $n_{dec} = 2020$
  
- e- Réaliser manuellement la conversion du nombre  $n_{hex} = 7d00$  en base 10 afin de trouver comme résultat la valeur de  $n_{dec} = 32000$
  
- g- S'inspirer de cette méthode pour écrire le code de la fct `conversion_hexa()`. Pour vous aider, 2 conseils :
  - ⇒ le nombre  $n_{hexa}$  pris en argument sera déjà une chaîne de caractère afin de pouvoir utiliser une boucle du genre : `for c in n_hex :`
  - ⇒ on conseille d'utiliser la fonction `affectation()` écrite dans l'exercice 3 précédent.

5. Code qui convertit un nombre de la base 16 vers la base 10

On se propose d'utiliser les fonctions créées dans les exercices précédents pour convertir le format hexadécimal d'une couleur en format décimal r,g,b.



⇒ Retrouver la figure ci-contre sur : <https://htmlcolorcodes.com/fr/selecteur-de-couleur/>

Dans ce cas exposé ici, la couleur jaune est définie par l'intensité de rouge, vert et bleu généré par l'écran. Ces intensités sont codées sur 1 octet chacune. Ici :



|      |                |               |
|------|----------------|---------------|
| red  | 243 en base 10 | F3 en base 16 |
| blue | 12 en base 10  | 0C en base 16 |

Le codage de la couleur en hexadécimal commence toujours par # suivi des 6 chiffres hexa définissant la valeur des 3 octets correspondant aux intensités du rouge, vert et bleu émis par l'écran.

On vous demande de coder une fonction que vous nommerez couleur(). Elle prend en argument un code de couleur hexadécimal et retourne les valeurs r, g, b qui correspondent à la conversion en décimal des 3 x 2 caractères qui suivent le #. On donne ci-après, quelques exemples d'exécution :

```
# Main *****
r,g,b = couleur("#f3ff0c")
print(r,g,b)
```

donnera :

```
>>> (executing file "exercice5.py")
243 255 12
```

```
# Main *****
r,g,b = couleur("#ffffff")
print(r,g,b)
```

donnera :

```
>>> (executing file "exercice5.py")
255 255 255
```

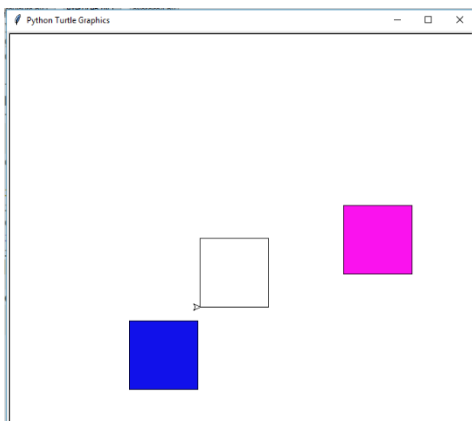
6. Code qui trace des carrés de 100px de coté et de couleurs différentes

Le programme principal ci-dessous, demande à l'utilisateur un nombre hexadécimal à 6 chiffres et renvoie dans une fenêtre turtle un carré de 100px de côté et dont la couleur est celle définie par le nombre saisi :

```
# Importation des bibliothèques
from turtle import *
from random import randint

# Main *****
for i in range(100) :
    coul_hexa = textinput("saisir un nombre hexadécimal à 6 chiffres : ","")
    if coul_hexa == "" :
        exitonclick()
        break
    else :
        coul_hexa = "#" + coul_hexa
        carré(coul_hexa)
```

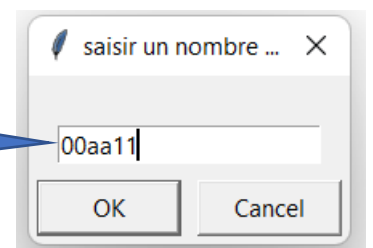
La fonction *textinput()* de la bibliothèque *turtle* renvoie la chaîne de caractère saisie par l'utilisateur dans une fenêtre de dialogue




On donne ci-contre un exemple d'exécution.

Dans ce bout de code, on appelle la fonction *carré()* dont on donne le code incomplet ci-après :

On saisie un code couleur en hexadécimal



```
def carre(coul_hexa) :  
    """  
    Prend en argument une couleur définie en hexadécimal. La  
    convertit en mode rgb. Trace un carré de coté 100px à  
    partir d'un point défini aléatoirement.  
    """  
    colormode(255)  
    up()  
    x = randint(-300,200)  
    y = randint(-300,200)  
    goto(x,y)  
    down()  
    r,g,b = couleur(coul_hexa)  
    fillcolor(r,g,b)
```



⇒ Compléter ce code en utilisant les fonctions créées dans les exercices précédents, afin que l'exécution de l'ensemble soit cohérent avec l'attendu.