Python – Binaire, mentalisme et parachutes

Préparation

Pour pouvoir suivre cette activité, il faut copier le dossier `Initiation` qui se trouve dans le dossier `Echange (P:) / Tous / NSI` et le mettre dans vos documents. Ce dossier contient un raccourci vers `Thonny` qu'il faut lancer et `feuille-python-initiation-nsi.pdf` qu'il faut ouvrir.

Binaire et décimal

Pour pouvoir faire cette partie, il faut ouvrir depuis `Thonny` le fichier `initiation.py`. Ce fichier contient les fonctions que nous allons compléter pour pouvoir faire les conversions.
Pour la conversion en binaire, nous allons utiliser la méthode des divisions. Tant que le nombre est supérieur ou égal à 2, on note la retenue qu'on met à gauche du nombre en binaire. On divise alors le nombre par deux et on recommence.
Pour faire les divisions, on va utiliser `a // b` pour le quotient et `a % b` pour le reste.

```
>>> 15 // 5
3
>>> 16 // 5
3
>>> 17 // 5
3
```

```
>>> 15 % 5
0
>>> 16 % 5
1
>>> 17 % 5
2
```

Les nombres en binaire seront représentés par des textes. Pour écrire un texte en Python, il suffit de le mettre entre guillemets ou apostrophes : `"un texte"` ou `'un autre texte'`. Pour le texte vide, on met juste les quillemets ou les apostrophes. Enfin, pour coller deux textes (on dit **concaténer**), il suffit d'écrire `texte1+texte2`.

```
>>> texte1 = "bonjour"
>>> texte2 = "les gens"
>>> texte1 + texte2
'bonjourles gens'   # il manque un espace
>>> texte2 = " " + texte2   # on rajoute l'espace à gauche de texte2
>>> texte2
' les gens'   # il y est bien
>>> texte1 + texte2
'bonjour les gens'
>>> texte1 + " !"  # on peut aussi rajouter à droite
'bonjour !'
```

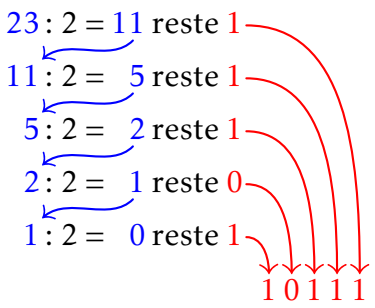
Par contre, on ne peut pas mettre un chiffre à côté d'un texte. Il faut d'abord le convertir en texte avec la commande `str(chiffre)`.

```
>>> 1 + "110"    # On va avoir une erreur
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> str(1) + "110"   # Là, ça marche
'1110'
```

Nous avons maintenant tout ce qu'il faut pour faire la conversion.

Exercice 1 : Compléter la fonction `conversion(nb)` qui prend un entier `nb` et renvoie le nombre binaire correspondant, représenté par un texte.

```
def conversion(nb):
    binaire = ""
    while nb >= 2:
        reste = ...
        binaire = str(reste) + binaire
        nb = ...
    binaire = str(...) + binaire
    return binaire
```




```
>>> conversion(19)
'10011'
>>> conversion(102)
'1100110'
>>> conversion(0)
'0'
```

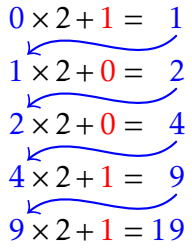
Pour repasser en décimal, il faut pouvoir parcourir les chiffres du nombre en binaire, pour utiliser la méthode des multiplications. Pour cela on utilise `for b in binaire` qui va faire une boucle où `b` va prendre successivement chacun des symboles de `binaire` de gauche à droite.

```
>>> binaire = "1101"
>>> for b in binaire: print(b)   # on affiche les chiffres un par un

1
1
0
1
```

Exercice 2 : Compléter la fonction `deconversion(binaire)` qui prend un nombre binaire représenté par un texte et renvoie le nombre décimal correspondant. On utilise la méthode des multiplications en multipliant à chaque nouveau chiffre le résultat par 2, en partant de 0, et en ajoutant 1 si le chiffre est `"1"`.

```
def deconversion(binaire):
    nb = ...
    for b in binaire:
        nb = ...
        if b == "1":
            nb = ...
    return nb
```




```
>>> deconversion("110")
6
>>> deconversion("0")
0
>>> deconversion("001101")
13
```

Place au mentalisme

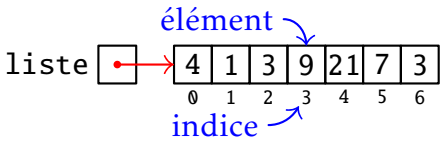
Pour pouvoir faire le tour de mentalisme, il faut générer les grilles de nombres. Pour cela nous utiliseront les listes Python. Une liste est une suite d'éléments séparés par des virgules et mis entre crochets. La liste vide est représentée par `[]`. La fonction **`len`**`(liste)` permet de connaître le nombre d'éléments dans `liste`.

```
>>> liste_vide = []
>>> liste = [4, 1, 3, 9, 21]
>>> len(liste)   # le nombre d'éléments dans la liste
5   # il y en a bien 5
```

Il est possible d'ajouter des éléments en fin de la liste en faisant `liste.append(element)`. On peut bien entendu rajouter plusieurs fois le même élément dans la liste.

```
>>> liste.append(7)
>>> liste.append(3)
>>> liste
[4, 1, 3, 9, 21, 7, 3]   # On a bien mis 7 puis 3
```

Pour accéder à un élément d'une liste, il faut mettre son indice entre crochets après le nom de la liste. Par exemple `liste[3]`. Les indices commencent à 0 et vont de gauche à droite.




```
>>> liste[0]
4
>>> liste[1]
1
>>> liste[4]
21
>>> liste[6]
3
```

L'indice du dernier élément est toujours **`len`**`(liste)-1`.

Mais nous n'allons pas avoir besoin de juste une liste, mais de plusieurs listes. Nous allons faire une liste de listes. En effet, les listes peuvent contenir n'importe quel type de valeur ou d'objet, et donc peuvent contenir des listes.

```
>>> liste_de_listes = [[1, 2], [3, 4]]
>>> liste1 = liste_de_listes[0]  # la première sous-liste
>>> liste2 = liste_de_listes[1]  # la deuxième sous-liste
>>> liste1
[1, 2]
>>> liste1[0]
1
>>> liste2[1]
4
>>> liste_de_listes.append([14, 89, 54])  # ajout d'une nouvelle sous-liste
>>> liste_de_listes
[[1, 2], [3, 4], [14, 89, 54]]
```

Enfin, lorsqu'on modifie une sous-liste, la modification se voit aussi dans la liste de listes :

```
>>> liste1.append(9)
>>> liste_de_listes
[[1, 2, 9], [3, 4], [14, 89, 54]] # on a bien rajouté 9 dans la première liste
```

Exercice 3 : Compléter la fonction `creer_grilles(n)` qui renvoie une liste de `n` listes vides.

```
def creer_grilles(n):
    grilles = [] # la liste des grilles
    for i in range(...):
        grilles.append(...) # on rajoute une liste vide
    return grilles
```

```
>>> creer_grilles(2)
[[], []]
>>> creer_grilles(6)
[[], [], [], [], [], []]
```

Il faut maintenant remplir les grilles en y mettant tous les nombres qui sont dedans. Si on pose n le nombre de grilles, alors dans la grille d'indice i va contenir tous les nombres entre 1 et $2^n - 1$ qui ont un 1 dans la colonne de 2^i. Par exemple, $26 = 11010$ sera dans les grilles d'indices 1 et 3 et 4. Pour remplir les grilles, on va donc utiliser la même méthode que pour la conversion, sauf qu'au lieu de construire le nombre binaire, on va utiliser la retenue pour savoir s'il faut mettre le nombre ou pas dans les grilles.

Exercice 4 : Compléter la fonction `placer_dans_grilles(nb, grilles)` qui va rajouter `nb` dans toutes les grilles dans lesquelles il doit être.

```
def placer_dans_grilles(nb, grilles):
    copie = nb
    n = ... # n est le nombre de sous grilles
    for i in range(n):
        grille = grilles[i] # la sous grille d'indice i
        if copie % 2 == ...: # dans quel cas il faut rajouter quelque chose ?
            grille.append(...) # qu'est-ce qu'on rajoute à grille ?
        copie = ... # et ensuite, on fait quoi à copie ?
```

```
>>> grilles = creer_grilles(6)
>>> placer_dans_grilles(26, grilles)
>>> grilles
[[], [26], [], [26], [26], []]  # 26 est bien dans les grilles d'indices 1, 3 et 4
```

Exercice 5 : Compléter la fonction `construire_grilles(n)` qui va créer les `n` grilles et les remplir avec les nombres entre 1 et $2^n - 1$ inclus.

```
def construire_grilles(n):
    grilles = creer_grilles(...)
    for nb in range(1, 2**n): # tous les nombres de 1 à (2^n)-1 inclus
        placer_dans_grilles(..., ...)
    return grilles
```

```
>>> construire_grilles(4)
[[1, 3, 5, 7, 9, 11, 13, 15], [2, 3, 6, 7, 10, 11, 14, 15],
 [4, 5, 6, 7, 12, 13, 14, 15], [8, 9, 10, 11, 12, 13, 14, 15]]
```

Afin de programmer le mentaliste, il faut faire la fonction qui permet de demander à l'utilisateur si le nombre se trouve dans une grille donnée ou non. Pour cela on va afficher la grille et lui demander de répondre par `"O"` ou `"N"`. On acceptera aussi les lettres en minuscules. Tant que l'utilisateur ne donne pas une réponse valide, on lui redemande.
Pour demander des valeurs à l'utilisateur, on utilise la fonction **`input`**`(texte)` qui affiche `texte` et attend que l'utilisateur tape quelque chose au clavier et appuie sur la touche Entrée. Le texte tapé est alors renvoyé.

```
>>> input("Ça va ? ")
Ça va ? oui
'oui'
```

Pour ne pas avoir à faire la différence entre minuscule et majuscule, on peut utiliser la commande `texte.upper()` qui renvoie le texte mis en majuscule.

```
>>> "bonjour".upper()
'BONJOUR'
>>> "Même avec des nombres comme 123, et de la ponctuation !".upper()
'MÊME AVEC DES NOMBRES COMME 123, ET DE LA PONCTUATION !'
>>> texte = "o O n N"
>>> texte.upper()
'O O N N'
```

Exercice 6 : Compléter la fonction `demander_utilisateur(grille)` qui affiche `grille` et qui demande à l'utilisateur se le nombre auquel il pense se trouve dans la liste. Si c'est le cas, elle renvoie **`True`** et sinon renvoie **`False`**.

```
def demander_utilisateur(grille):
    print("Est-ce que le nombre est dans cette grille :")
    print(...)   # on affiche la grille
    reponse = "reponse pas bonne"
    while reponse != "O" and reponse != "N": # toujours pas de bonne réponse
        reponse = input("O/N ? ")
        reponse = ... # on met en majuscule
    if reponse == ...: # pour quelle valeur est-ce qu'on renvoie True ?
        return True
    else:
        return ... # Sinon quoi ?
```

```
>>> demander_utilisateur([1, 3, 5, 7, 9, 11, 13, 15])
Est-ce que le nombre est dans cette grille :
[1, 3, 5, 7, 9, 11, 13, 15]
O/N ? Je ne veux pas le dire
O/N ? Oui   # ce n'est pas une réponse attendue
O/N ? o   # en majuscule, ça marchait aussi
True
```

L'ordinateur mentaliste

Nous pouvons maintenant programmer la fonction qui fera le tour de mentalisme. L'ordinateur va commencer par générer les grilles puis va demander à l'utilisateur, grille par grille, si le nombre auquel il pense se trouve dedans. Chaque fois qu'il dit oui, il suffit d'ajouter le premier élément de la grille au total pour trouver le nombre.

Dans l'exemple ci-contre, l'utilisateur a dit oui pour les trois grilles en bleu. Si on fait $4+8+16$, on trouve 28. C'est le seul nombre qui se trouve dans ces 3 et pas dans les autres. C'est parce qu'il s'écrit 011100 en binaire.

1	3	5	7	9	11	13	15
17	19	21	23	25	27	29	31
33	35	37	39	41	43	45	47
49	51	53	55	57	59	61	63

2	3	6	7	10	11	14	15
18	19	22	23	26	27	30	31
34	35	38	39	42	43	46	47
50	51	54	55	58	59	62	63

4	5	6	7	12	13	14	15
20	21	22	23	28	29	30	31
36	37	38	39	44	45	46	47
52	53	54	55	60	61	62	63

8	9	10	11	12	13	14	15
24	25	26	27	28	29	30	31
40	41	42	43	44	45	46	47
56	57	58	59	60	61	62	63

16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

Exercice 7 : Compléter la fonction `mentaliste(n)` qui fait le tour de mentalisme avec `n` grilles.

```
def mentaliste(n):
    grilles = ... # on fabrique les grilles
    print("Pense à un nombre entre 1 et", 2**n-1)
    nb = ... # quelle valeur initiale ?
    for i in range(...):
        grille = grilles[i]
        reponse = ... # on demande à l'utilisateur
        if ...: # dans quel cas est-ce qu'on fait quelque chose ?
            nb = ... # on rajoute le premier élément de la grille
        print("Le nombre auquel tu penses est", ...)
```

```
>>> mentaliste(4)
Pense à un nombre entre 1 et 15
Est-ce que le nombre est dans cette grille :
[1, 3, 5, 7, 9, 11, 13, 15]
O/N ? o
Est-ce que le nombre est dans cette grille :
[2, 3, 6, 7, 10, 11, 14, 15]
O/N ? n
Est-ce que le nombre est dans cette grille :
[4, 5, 6, 7, 12, 13, 14, 15]
O/N ? o
Est-ce que le nombre est dans cette grille :
[8, 9, 10, 11, 12, 13, 14, 15]
O/N ? n
Le nombre auquel tu penses est 5
```

Deviner le nombre auquel pense l'ordinateur

Il est également possible de faire l'inverse. L'ordinateur choisit un nombre, indique les grilles dans lesquelles il se trouve ou pas, et c'est l'utilisateur qui doit trouver le nombre. Pour cela, l'ordinateur va devoir tester si le nombre qu'il a choisi se trouve dans une grille. Au lieu de refaire la conversion en binaire, on va juste utiliser la commande Python qui teste l'appartenance à une liste : `val in liste`.

```
>>> liste = [2, 3, 6, 7, 10, 11, 14, 15]
>>> 1 in liste
False
>>> 7 in liste
True
>>> 12 in liste
False
```

Exercice 8 : Compléter la fonction `mentaliste_inverse(n)` qui correspond à une partie où l'ordinateur met au défi l'utilisateur de trouver le nombre, entre 1 et $2^n - 1$ auquel il pense.

```
def mentaliste_inverse(n):
    print("Je pense à un nombre entre 1 et", ...)
    nb_secret = random.randint(1, 2**n-1) # on tire au hasard une valeur
    grilles = ...
    for i in range(n):
        if ...:
            print("Il est dans", grilles[i])
        else:
            print("Il n'est pas dans", ...)
    reponse = int(input("Devine le nombre auquel je pense : "))
    if ...:
        print("Tu es trop fort !")
    else:
        print("Tu as tout faux, c'était", ...)
```

```
>>> mentaliste_inverse(4)
Je pense à un nombre entre 1 et 15
Il n'est pas dans [1, 3, 5, 7, 9, 11, 13, 15]
Il n'est pas dans [2, 3, 6, 7, 10, 11, 14, 15]
Il est dans [4, 5, 6, 7, 12, 13, 14, 15]
Il n'est pas dans [8, 9, 10, 11, 12, 13, 14, 15]
Devine le nombre auquel je pense : 4
Tu es trop fort !
```

Exercice 9 : Essayer de deviner le nombre auquel pense l'ordinateur en prenant $n = 8$.

Version graphique

Le fichier `mentaliste.py` contient un ensemble de fontions permettant de proposer une interface graphique pour le tour du mentaliste. Il utilise le fichier `initiation.py` que nous venons de compléter, en important toutes ses fonctions.
Il suffit de cliquer sur des grilles pour sélectionner les grilles dans lesquelles se trouve le nombre cherché et d'appuyer sur le bouton en bas pour afficher le nombre trouvé par l'ordinateur.
Pour changer le nombre de grilles, il faut changer la valeur de `NB_BITS` qui se trouve à la ligne 14.

Exercice 10 : Ouvrir le fichier avec Thonny et tester si l'ordinateur a toujours raison.

Le parachute

De la même manière le fichier `parachute.py` permet de générer des messages codés par des parachutes, mais aussi de les décoder. Il utilise également le fichier `initiation.py`.
Pour générer une image, il faut utiliser la fonction suivante qui renvoie l'image :
`dessiner_parachute(texte, gps, quartier_debut)`
Si le texte est composé de 3 mots de 7 lettres maximum, alors il y a un mot par disque. Sinon, le texte doit faire 21 caractères au maximum et il est découpé en 3 parties réparties sur les 3 disques. Les coordonnées GPS servent pour le dernier disque. Vous pouvez utiliser `GPS` pour les coordonnées du parachute original ou `GPS_LYCEE` pour utiliser les coordonnées du lycée.
Le paramètre `quartier_debut` permet de décider du quartier d'où part le premier mot. Le quartier 0 correspond au quartier en haut, légèrement à droite et les autres suivent dans le sens des aiguilles d'une montre.
Le parachute de Perseverance est généré par cette fonction :
`dessiner_parachute("DARE MIGHTY THINGS", GPS, 1)`.
Pour le voir il faut utiliser la fonction `afficher_image(image)`. Il est aussi possible de le sauvegarder avec la fonction `sauver_image(image, nom)`.
Voici un exemple d'utilisation :

```
>>> parachute = dessiner_parachute("DARE MIGHTY THINGS", GPS, 1)
>>> afficher_image(parachute)
>>> sauver_image(parachute, "parachute.png")
```

Vous pouvez également décoder le message contenu dans une image avec la fonction :
`decoder_parachute(nom)`.

```
>>> decoder_parachute("parachute.png")
Si c'est 3 mots :
DARE MIGHTY THINGS
Sinon :
DAREMIGHTYTHINGS
Coordonnées GPS :
34 11 58 N 118 10 31 W
```

La fonction affiche les deux versions possibles du message (en 3 mots ou en collant les symboles des 3 disques), ainsi que les coordonnées GPS.

Exercice 11 : Ouvrir le fichier `parachute.py` dans Thonny et essayer de générer le parachute original ou d'autres parachutes.