Spécification et test

Nous avons vu qu'un **algorithme** est correctement défini lorsqu'on précise les **données** et le **résultat** de cet algorithme

Nous allons procéder de même pour chaque **fonction d'un programme** on parle alors de spécification de fonction

Nous allons aussi intégrer des tests à certaines fonctions

1 Annotation de type

Contrairement à des langages comme Java ou C++, Python n'est pas un langage **typé**, c'est à dire que l'on n'est pas obligé de préciser le type des variables d'un programme

Par contre pour la lisibilité et la compréhension d'un programme il est utile de mettre des informations concernant le type des variables dans le programme sous forme d'annotations

Dans un premier temps on va annoter le type des paramètres de la fonction s'il y en a et de ce qui est retourné par la fonction (ceci est possible depuis Python 3.5)

Regardons cela sur la fonction

```
def dec_to_bin(nombre : int, taille : int) -> list:
    quotient = nombre
    octet = [0]*taille
    i = 0
    while quotient > 0:
        octet[i] = quotient \% 2
        quotient  = quotient // 2
        i += 1
    return octet
```

Commentaires

1. On précise le type des paramètres de la fonction, `nombre` et `taille` sont de type `int` (pour integer qui signifie entier en anglais)
2. On précise le type de ce qui est retourné entre la "flèche" (formée du signe moins et du signe >) et les deux points

   Ici on retourne un tableau de type `list`
3. Si la fonction ne retourne rien on écrit `None` entre la "flèche" et les deux points par exemple

   ```
   def complementaire(octet : list) -> None:
       for i in range(len(octet)):
           octet[i] = 1 - octet[i]
   ```

4. Ces informations sont utiles à l'utilisateur de cette fonction pour éviter des mauvais usages de la fonction ainsi un utilisateur particulier pourrait croire que cette fonction retourne une chaîne de caractère (str) ce qui n'est pas le cas

2 Spécifier

On aimerait cependant avoir un peu plus d'informations sur cette fonction, il est d'usage de rajouter en dessous de la ligne où se trouve `def`, une phrase explicative décrivant un peu plus la fonction comme ceci

```
def dec_to_bin(nombre : int, taille : int) -> list:
"""retourne la représentation binaire de nombre
dans un tableau de longueur taille
"""
    quotient = nombre
    octet = [0]*taille
    i = 0
    while quotient > 0:
        octet[i] = quotient \% 2
        quotient  = quotient // 2
        i += 1
    return octet
```

Ensuite suivant le contexte, on précise un peu plus ou pas les paramètres à la manière de ce qui est fait dans la documentation du module `matplotlib`
`https://matplotlib.org/stable/devel/documenting_mpl.html#writing-docstrings`

```
def dec_to_bin(nombre : int, taille : int) -> list:
"""retourne la représentation binaire de nombre
dans un tableau de longueur taille

Paramètres
------------
nombre : int
        un entier naturel < 2**taille

taille : int
        la longueur du tableau >= 1

Résultat:
--------
         un tableau de longueur taille contenant les bits
         de la représentation binaire de nombre
         et où l'élément d'indice i du tableau est le chiffre
         associé à 2**i

"""
    quotient = nombre
    octet = [0]*taille
    i = 0
    while quotient > 0:
        octet[i] = quotient \% 2
        quotient  = quotient // 2
```

```
        i += 1
    return octet
```

Commentaires

1. En tant que programmeur utilisateur nous sommes contents de trouver des informations sur telle ou telle fonction en utilisant la fonction `help()`

   Aussi si on fait `help(dec_to_bin)` on pourra voir la spécification dans la console (docstring)

```
>>> help(dec_to_bin)
Help on function dec_to_bin in module __main__:

dec_to_bin(nombre: int, taille: int) -> list
    retourne la représentation binaire de nombre
    dans un tableau de longueur taille

    Paramètres
    ------------
    nombre : int
            un entier naturel < 2**taille

    taille : int
            la longueur du tableau >= 1

    Résultat:
    --------
            un tableau de longueur taille contenant les bits
            de la représentation binaire de nombre
            et où l'élément d'indice i du tableau est le chiffre
            associé à 2**i
```

2. On conseille d'**écrire la spécification avant le code** comme une sorte de cahier des charges pour nous guider dans l'écriture du code

3 Tester

Annoter et spécifier une fonction sont des moyens pour nous aider à mieux définir une fonction. Cependant cela ne suffit pour nous assurer que la fonction une fois exécutée fera le travail escompté

Comment être sûr que la fonction à l'exécution fera le travail et ceci dans tous les cas possibles ?

Pour l'instant on se contentera de quelques cas en intégrant des tests à la spécification, quand c'est possible, grâce au module `doctest`

```
def dec_to_bin(nombre : int, taille : int) -> list:
"""retourne la représentation binaire de nombre
dans un tableau de longueur taille

Paramètres
------------
```

```
nombre : int
         un entier naturel < 2**taille

taille : int
         la longueur du tableau >= 1

Résultat:
--------
          un tableau de longueur taille contenant les bits
          de la représentation binaire de nombre
          et où l'élément d'indice i du tableau est le chiffre
          associé à 2**i

>>> dec_to_bin(2, 8)
[0, 1, 0, 0, 0, 0, 0, 0]
>>> dec_to_bin(255, 8)
[1, 1, 1, 1, 1, 1, 1, 1]
>>> dec_to_bin(0, 8)
[0, 0, 0, 0, 0, 0, 0, 0]

"""
    quotient = nombre
    octet = [0]*taille
    i = 0
    while quotient > 0:
        octet[i] = quotient \% 2
        quotient  = quotient // 2
        i += 1
    return octet
```

Ensuite on écrit dans le programme principal pour lancer les tests

```
import doctest
doctest.modtest(verbose=True)
```

Ensuite dans la console on observe le message suivant dans lequel on peut observer que les tests sont effectués :

On compare les résultats "espérés" aux résultats obtenus, si ce sont les mêmes on aura le message "Test passed"

Dans ce message on précise aussi les fonctions qui n'avaient pas de tests

```
Trying:
    dec_to_bin(2, 8)
Expecting:
    [0, 1, 0, 0, 0, 0, 0, 0]
ok
Trying:
    dec_to_bin(255, 8)
Expecting:
    [1, 1, 1, 1, 1, 1, 1, 1]
ok
Trying:
    dec_to_bin(0, 8)
Expecting:
    [0, 0, 0, 0, 0, 0, 0, 0]
ok
3 items had no tests:
    __main__
    __main__.complementaire
    __main__.complementaire2
1 items passed all tests:
   3 tests in __main__.dec_to_bin
3 tests in 4 items.
3 passed and 0 failed.
Test passed.
```

Attention

1. Après les trois symboles > il faut laisser un espace avant d'appeler la fonction, ainsi

   `> > > dec_to_bin(2, 8)` est correct par contre `> > >dec_to_bin(2, 8)` produira une erreur

2. Quand le résultat d'un test est un tableau il faut laisser un espace après la virgule
3. Si on exécute `help(dec_to_bin)` à nouveau on verra les tests comme des exemples d'utilisation ce qui aide l'utilisateur dans la compréhension de la fonction
4. Ce n'est pas la peine de faire de nombreux tests mais il faut envisager quelques cas "intéressants", par exemple, ne pas oublier les cas "limites"

Exercices

Ex 1

Annoter les fonctions suivantes :

1. `nb_chiffres(n)` qui retourne le nombre de chiffres d'un entier n
2. `complementaire(octet)` qui ne retourne rien mais qui transforme un octet de type `list` en son complémentaire
3. `distance(t1,t2)` qui retourne le nombre de différences entre deux tableaux `t1` et `t2` de même longueur et contenant des entiers

Ex 2

Spécifier les fonctions suivantes :

1. `nb_chiffres(n)` qui retourne le nombre de chiffres d'un entier n
2. `complementaire(octet)` qui ne retourne rien mais qui transforme un octet de type `list` en son complémentaire
3. `distance(t1,t2)` qui retourne le nombre de différences entre deux tableaux `t1` et `t2` de même longueur et contenant des entiers

Ex 3

Ajouter des tests aux spécifications des fonctions suivantes

1. `nb_chiffres(n)` qui retourne le nombre de chiffres d'un entier n
2. `distance(t1,t2)` qui retourne le nombre de différences entre deux tableaux `t1` et `t2` de même longueur et contenant des entiers

Ex 4

Voici la définition d'une fonction

```
def f(t):
    s = 0
    for i in range(len(t)):
        s += t[i]
    return s
```

1. Que fait elle ?
2. Rendre plus explicite la définition de cette fonction (mieux la nommer, mieux nommer les variables)
3. Annoter, spécifier et tester

Ex 5

Avant de définir une fonction `nombre_de_max(tableau)` qui retourne le nombre de maximum contenu dans un tableau `tableau` contenant des entiers, procéder dans l'ordre :

1. Annoter
2. Spécifier
3. Inclure des tests