TP3 : Typage en Python & manipulation binaire

Les fichiers Python de ce TP sont disponibles sur mon site internet : bouillotvincent.github.io .

I. Résumé de l'épisode précédent (TP2)

20 min 

1) Boucles en Python

Le TP2 nous faisait à nouveau manipuler les boucles.

```
for i in range(4):
    fred.forward(5)
    fred.left(90)
```

Les boucles bornées **`for`** sont utiles lorsque l'on souhaite :
❖ réaliser les instructions dans la boucle un nombre fixé de fois ;
❖ épeler une liste d'éléments.

```
for i in 'abcd':
    print(i)
```

Les boucles bornées **`while`** sont utiles lorsque l'on ne connait pas le nombre de répétitions à l'avance. Elles comportent une initialisation, une condition et une incrémentation.

```
nombre = 0
while nombre**2 < 125:
   nombre = nombre + 0.1
```

2) Fonctions en Python

Le TP2 introduisait surtout la notion de **fonctions.** Celles-ci sont des "boites noires" permettant de réaliser des calculs complexes qui peuvent être réutilisées à souhait.

En tant que boites noires, les variables apparaissant à l'intérieur d'une fonction **n'existent qu'au sein de ces fonctions**. Elles sont détruites à la fin de la fonction.
Certains fonctions, appelées **procédures**, ne renvoient d'ailleurs aucun nombre.

En général, nous souhaitons obtenir un résultat de la part d'une fonction. Pour se faire, nous utilisons le mot clé **`return`**.

Exemple 1 (bouillotvincent.github.io) :

Ceci est une fonction qui prend un nombre **n** en paramètre et renvoie $\frac{n}{3}$ si **n** est divisible par 3 et n^2 sinon. Oui, oui, ça ne sert à rien…

```
def fonctionInutile(n):
    reste = n%3
    if reste == 0:
        return n/3
    else:
        return n**2

print(fonctionInutile(10))
print(fonctionInutile(12))
```

❐ — Exercice **1** —

Code divisibilite.py à télécharger sur bouillotvincent.github.io.

En moins de 10 minutes, écrivez une fonction dont le nom est **`estDivisiblePar`** . Elle teste si un entier **`n`** est divisible par un entier **`q`**.

`estDivisiblePar` prend donc deux paramètres en entrée :

❖ un entier **`n`** ;

❖ un entier **`q`**.

`estDivisiblePar` renvoie **`True`** si n est divisible par q et **`False`** sinon.

Jeu de tests :

print(estDivisiblePar(18, 3)) doit renvoyer True

print(estDivisiblePar(100, 50)) doit renvoyer True

print(estDivisiblePar(101, 45)) doit renvoyer False

II. Typage en Python

30 min

Les données et les variables sont **typées** dans un langage de programmation, c'est-à-dire que l'on doit préciser si nos variables sont des entiers, des nombres à virgules, des chaines de caractères, des tableaux ou autres.

En Python, le typage est **très souple** : le type d'une même variable n'est pas fixé et peut être modifié selon les besoins. On appelle cela le **duck typing**.

Néanmoins, il est fondamental de savoir quels types existent pour en connaitre les avantages et les limitations.

L'instruction **`type(variable)`** permet d'obtenir le type d'une variable. Cette instruction renvoie un résultat de la forme <class '……'>, ce qui signifie que la variable est du type indiqué entre les guillemets.

❐ — Exercice **2** —

Code exercice2_3.py à télécharger sur bouillotvincent.github.io.

En utilisant les instructions **`type`** et **`print`**, complétez le tableau de la page suivante.

	Variable	Type
Types simples	`a = 1000`	
	`b = 1000.0`	
	`c = '1000'`	
	`d = True`	
Types construits	`cadeau = ['tablette', 'livre', 'lego']`	
	`constantes = (sqrt(2), pi, e, (1+sqrt(5))/2)`	
	`annuaire = {'Neumann': '0612670912', 'Ada': '0612379112', 'Berners': '0965267712'}`	

Une opération appelée le **transtypage** permet de changer le type d'une variable :

- en entier en utilisant `int(variable) ;`
- en nombre à virgules en utilisant `float(variable) ;`
- en chaine de caractères avec `str(variable) ;`

❐ — Exercice **3** —

Changez le type de la variable **c** en entier, puis modifiez à nouveau **c** afin de la transformer en nombre à virgules.

Vous utilisez **`print`** afin d'afficher la variable et le type de **c** à chaque changement de type.

❐ — Exercice **4** —

Code exercice4.py à télécharger sur bouillotvincent.github.io.

- En utilisant le transtypage, complétez le programme de l'exercice 4 afin que la valeur des nombres n et q donnés par l'utilisateur soient des nombres entiers. Le programme renverra une erreur type **`AssertionError`** si vous n'avez pas fait le travail attendu.
- Plutôt que d'afficher les entiers **a** et **b** (ligne 21), on souhaite afficher une phrase comme indiquée dans la dernière ligne :

```
print('Le quotient vaut ' + a + ' et le reste vaut ' + b)
```

- Dans cette affichage, on **concatène** (ajoute avec des **+**) des chaines de caractères, ce qui n'est possible que si a et b sont aussi des chaines de caractères.
  Transtypez ces variables pour faire fonctionner l'affichage.

III. Manipulation binaire : conversion base 2 ⇔ base 10

60 min

Pour encoder des nombres en base 2, nous choisissons d'utiliser des chaines de caractères de la forme : `'0b101011'` avec **`'0b'`**, un préfixe indiquant que le nombre est en base 2.

Nous serons amené à pratiquer le transtypage afin de réaliser des calculs sur ces "nombres" binaires.

— Exercice 5 —

Code exercice5_7.py à télécharger sur bouillotvincent.github.io.

Complétez la fonction **`estBinaire`** qui teste si une chaine de caractère est en base 2.

Voici l'algorithme de cette fonction :

- Si le préfixe n'est pas égal à '**`0b`**', on renvoie le booléen **`False`**
- Sinon, on épelle **`bit`** par bit la chaine de caractères **`nbin`** et :
  - Si un bit est différent de '0' et de '1', on renvoie **`False`**
- Si on réussit à épeler toute la chaine de caractères **`nbin`** (pas de bit différent de 0 et 1), on renvoie **`True`**.

L'exercice est réussi si aucune AssertionError n'apparait lors de l'exécution du programme.

Remarque : les lignes 17 à 35 sont appelées "jeu de tests". Ils permettent de valider la fonction en testant tous les cas (problématiques ou non) possibles.

Très souvent, on a besoin de connaitre le nombre d'éléments dans une séquence (chaine de caractères ou tableau). Ce se fait grâce à l'instruction **`len`** :

```
len(nomDeLaSequence)
```

Exemple : `len('ABCDE')` vaut 5.

— Exercice 6 —

Code exercice5_7.py de l'exercice précédent.

- Faire la décomposition en base 2 de 1011_2.
- Rappelez la formule pour faire la décomposition en base 2 d'un nombre binaire contenant L bits et écrit sous la forme : $A_2 = \text{bit}_{L-1}\text{bit}_{L-2}\ldots\text{bit}_1\text{bit}_0$.
- À partir de cette formule, complétez la fonction **`bin2dec`** qui convertit en base 10 un nombre binaire **`nbin`** écrit sous la forme '0b…….' .
- En utilisant la fonction **`estBinaire`**, modifiez **`bin2dec`** afin d'être sur et certain que **`nbin`** écrit sous la forme '0b…….'. Si la valeur renvoyée par estBinaire est False, la fonction **`bin2dec`** renverra "Votre nombre n'est pas au format 0b……..".

Lancez le programme calculette.py. Celui-ci fait appel à la fonction **bin2dec** en utilisant le programme exercice5_7.py comme module. La calculatrice fonctionne-t-elle pour la conversion binaire vers décimale ?

Nous souhaitons maintenant faire une conversion base 10 vers base 2. Pour essayer de trouver l'algorithme en langage naturel (i.e. la suie d'opérations à réaliser), nous allons travailler sur un exemple.

❐ — Exercice **7** —

Code exercice5_7.py de l'exercice précédent.

- Rappeler les commandes permettant d'obtenir le quotient Q et le reste R de la division euclidienne de A par B en Python (éventuellement, faites une petite recherche internet) :

  _____

- À l'aide de l'algorithme de la division euclidienne, convertir 53_{10} en binaire. Indiquez à chaque étape A, B, Q et R et leurs équivalents en Python.

- À l'aide des questions précédentes, complétez la fonction **dec2bin** qui convertit un nombre décimal en nombre binaire. **dec2bin** prend comme paramètre un entier **n** et renvoie un nombre **nBin** en base 2 sous forme de chaine de caractères '0b……' .

  Exemple : `print( dec2bin(25) )` renvoie '0b11001'

  Rappel :
  pour ajouter un caractère '**a**' à la fin (au début) d'une chaine de caractères **chaine** et le sauvegarder en écrasant **chaine**, on fait :

```
chaine = chaine + 'a'
chaine = 'a' + chaine
```

- Lancez le programme calculette.py et testez la conversion base 10 vers base 2. Cela fonctionne-t-il ? Faites les modifications nécessaires pour la faire fonctionner.