TP2 : Conversion et Python, le retour

Nous allons faire la suite du premier TP où nous avions codé un programme qui permettait la conversion vers la base 10.

Votre objectif sera de faire un programme qui permettra la conversion au départ de la base 10 vers n'importe quelle base…

Point Python :

Python et les divisions :

Testez les lignes suivantes :

```
16 / 3
16 // 3
16 % 3
```

`a / b`	retourne le résultat de la division décimale de a par b.
`a // b`	retourne le quotient de la division euclidienne de a par b.
`a % b`	retourne le reste de la division euclidienne de a par b.

Table des matières

Niveau expert : ***2***

Niveau Newbie : ***2***

Niveau expert :

Réaliser une fonction `conversion_depart_10` qui prendra comme arguments le nombre et la base d'arrivée et retournera l'écriture de ce nombre en base 10 dans la base choisie.

Niveau Newbie :

Mise en place et rappel

Rappel sur cette conversion.
On peut la faire par division successives ou par la méthode des plus grandes puissances

Nous allons opérer avec la méthode des divisions successives.

Si nous avons un nombre en base 10 et que nous voulons sont écrire en base b, alors nous faisons des divisions euclidiennes successives par b, jusqu'à obtenir un quotient de 0.
La conversion est le nombre écrit à partir des restes obtenus en partant du dernier reste.

Voici un exemple pour comprendre :
Convertissons $(41)_{10}$ vers la base 2 :

41	2					
1	20	2				
	0	10	2			
		0	5	2		
			1	2	2	
				0	1	2
					1	0

$$(41)_{10} = (101001)_2$$

Nous allons donc devoir faire un certain nombre de divisions euclidienne, récupérer et stocker les restes dans le bon sens et le retourner.

Donc, on sent le besoin d'une boucle.
Pour les boucles, nous avons 2 choix : le `for` ou le `while`.
Comment choisir ???
Si vous savez combien de fois boucler, le `for` est pour vous, si vous avez une condition de fin de bouclage, le `while` est à vous.
Dans notre cas, nous savons que nous devons boucler jusqu'à obtenir un quotient de 0, donc partons pour le `while`.

Contrairement au dernier programme où nous l'avons transformé à la fin en fonction, nous allons le structurer comme tel dès le début.

Notre programme s'appellera donc `conversion_depart_10,` il prendra comme argument le nombre à convertir `N` et la base `b` d'arrivée.

Nous allons ensuite devoir faire un certain nombre de divisions euclidienne, récupérer et stocker les restes dans le bon sens et le retourner.
Mais combien de division à faire ???
On ne peut pas le savoir simplement.
Par contre, nous savons que nous stopperons nos divisions successives lorsque le quotient obtenu sera nul.
Par conséquent, nous allons nous orienter vers une boucle en `while`, plutôt qu'une boucle en `for`…

On obtient un code de départ comme suit :

```
def conversion_depart_10 (N, base):
    q = N   #initialisation du quotient
    M = ""   #initialisation de la réponse

    while q > 0:
        …

    return M
```

Pour le stockage, nous allons utiliser l'addition de chaînes de caractères ou concaténation.
Pour comprendre le fonctionnement, teste les ligne suivantes :

```
a = "azerty"
b = "123"
print(a + b)
print(b + a)
```

Facile de stocker des valeurs…

À partir de là, tu peux essayer de compléter le code au-dessus…

Réponse page suivante…

```
def conversion_depart_10 (N, base):
    q = N   #initialisation du quotient
    M = ""   #initialisation de la réponse

    while q > 0:
        r = q % base
        q = q // base
        M = r + M
    return M
```

Vous pouvez la tester avec une conversion vers la base 2 :

```
conversion_depart_10 (10, 2)
```

Ou vers la base 16 :

```
conversion_depart_10 (15, 16)
```

Vous voyez un problème ???

Vous avez dû avoir ce message d'erreur :

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Mais pourquoi?

Et bien tout simplement car `M` a été défini comme une chaîne de caractère, mais quand nous faisons la division euclidienne, nous travaillons avec des entiers, et notre reste est un entier…
Et quand nous faisons notre concaténation (`M = r + M`), nous faisons une opération avec un entier et une chaîne de caractères… Et on ne le peut pas.

Comment corriger cela ?
Testez les lignes suivantes :

```
a = 123
print(type (a))
a = str(a)
print(type(a))
```

Nous avons un convertisseur en chaîne de caractères avec le `str()`.

Modifie ton code pour que ça fonctionne.

Réponse page suivante…

```
def conversion_depart_10 (N, base):
    q = N   #initialisation du quotient
    M = ""   #initialisation de la réponse

    while q > 0:
        r = q % base
        q = q // base
        M = str(r) + M
    return M
```

On repart pour un test :

```
conversion_depart_10 (10, 2)
```

Ou vers la base 16 :

```
conversion_depart_10 (15, 16)
```

Vous voyez un problème ???

Et oui, avec la base 16. On a un retour de 15, alors qu'il nous fallait un F.

On n'a pas géré la conversion des résultats des restes en lettres si le reste est supérieur à 10…

Point python :

Teste les lignes suivantes :

```
print(ord('A'))
print(chr(65))
print(chr(66))
```

`chr()` prend un Unicode et fournit le caractère correspondant, le `ord()` (que l'on connaissait) fournit l'Unicode du caractère.

Voici une trame que tu vas essayer de compléter :

```
def conversion_depart_10 (N, base):
    q = N   #initialisation du quotient
    M = ""   #initialisation de la réponse

    while q > 0:
        r = q % base
        q = q // base
        if r < 10:
            …
        else:
            …
        M = r + M
    return M
```

Réponse page suivante…

```
def conversion_depart_10 (N, base):
    q = N   #initialisation du quotient
    M = ""   #initialisation de la réponse

    while q > 0:
        r = q % base
        q = q // base
        if r < 10:
            r = str(r)
        else:
            r = chr(r + 55)
        M = r + M
    return M
```

À toi de tester…

Ta fonction … fonctionne.

Félicitations pour le suivi, dans les prochain TP nous allons revoir les bases entrevues sur ces 2 TPs puis faire notre premier projet !!!