LMI 1 - Semestre 1

Université d'Artois
Faculté des Sciences Jean Perrin

ALGO1 – Algorithmique et Programmation 1 – Groupe 4

Fiche de TD numéro 2 - Boucles et chaînes de caractères

Exercice 1 : Spécifiez, puis écrivez une fonction `somme_diviseurs` qui retourne la somme des diviseurs propres d'un entier n fourni en paramètre. a est un diviseur propre de b si $a < b$ et si $b\%a == 0$.

```
>>> somme_diviseurs(4)
3
>>> somme_diviseurs(220)
284
```

Exercice 2 : Spécifiez puis écrivez une fonction `est_parfait` qui teste si un entier fourni en paramètre est un nombre parfait. Un entier est parfait s'il est égal à la somme de ses diviseurs propres.

```
>>> est_parfait(8)
False
>>> est_parfait(6)
True
```

Exercice 3 : Spécifiez puis écrivez une fonction `sont_amis` qui teste si deux entiers fournis en paramètres sont des nombres amis. Deux entiers sont amis si la somme des diviseurs de l'un est égal à l'autre, et réciproquement.

```
>>> sont_amis(220,284)
True
```

Exercice 4 : Spécifiez puis écrivez une fonction `nbre_parfaits_sous` qui retourne le nombre d'entiers parfaits inférieurs à une valeur donnée en paramètre.

```
>>> nbre_parfaits_sous(500)
3
```

Exercice 5 : Spécifiez puis écrivez une fonction `les_nbres_parfaits_sous` qui retourne les entiers parfaits inférieurs à une valeur donnée en paramètre. Quel est le type de retour adapté ?

```
>>> les_nbres_parfaits_sous(1000)
[6, 28, 496]
>>> les_nbres_parfaits_sous(10000)
[6, 28, 496, 8128]
```

Exercice 6 : Pour effectuer une multiplication, les égyptiens utilisaient uniquement la multiplication par deux, la division par deux, et l'addition :

$$\text{mult\_egyptienne}(m,n) \mapsto \begin{cases} 0 & si\ m = 0 \\ \text{mult\_egyptienne}(m/2, 2 \times n) & si\ m\ pair \\ n + \text{mult\_egyptienne}(m-1, n) & si\ m\ impair \end{cases}$$

Regardez ce qu'il se passe pour `15 * 387` (non, on ne vous demande pas de calculer le résultat, mais de dérouler le calcul). Quel algorithme itératif proposer ?

Spécifiez puis écrivez une fonction `mult_egyptienne`.

Exercice 7 : Spécifiez puis écrivez la fonction `fibo(n)` qui retourne la valeur du terme `n` de la suite de Fibonacci.

La suite de Fibonacci est définie par :

$$\begin{aligned} u_0 &= 0 \\ u_1 &= 1 \\ u_n &= u_{n-2} + u_{n-1} \quad n > 1 \end{aligned}$$

Exercice 8 : Spécifiez, puis écrivez une fonction `est_valide` qui prend en paramètres deux chaînes de caractères, un *mot* et un *schéma*. Le schéma comporte des caractères `'?'` qui correspondent à des jokers. Un mot est valide par rapport à un schéma, s'il est identique au schéma sur tous les caractères, sauf sur les `'?'` où n'importe quel caractère peut être accepté.

```
>>> estValide("mémé","m?m?")
True
>>> estValide("momi","m?m?")
True
>>> estValide("mmmmmhhh","m?m?")
False
```

Exercice 9 : Spécifiez puis écrivez une fonction `est_prefixe` qui prend en paramètres deux chaînes de caractères et dit si la première est un préfixe de l'autre.

```
>>> est_prefixe("pre","prefixe")
True
>>> est_prefixe("pro","prefixe")
False
>>> est_prefixe("fi","prefixe")
False
```

Exercice 10 : Spécifiez puis écrivez une fonction `est_suffixe` qui prend en paramètres deux chaînes de caractères et dit si la première est un suffixe de l'autre.

```
>>> est_suffixe("fixe","suffixe")
True
>>> est_suffixe("fix","suffixe")
False
>>> est_suffixe("suffixe","suffixe")
True
```

Exercice 11 : Spécifiez puis écrivez une fonction `contient_dans_ordre` qui prend en paramètres deux chaînes de caractères et dit si toutes les lettres de la première sont présentes dans l'autre dans le même ordre.

```
>>> contient_dans_ordre("aeiou","blablebliboblublublu")
True
>>> contient_dans_ordre("aeiyu","blablebliboblublublu")
False
```

Exercice 12 : Spécifiez puis écrivez une fonction `miroir` qui retourne la chaîne miroir d'une chaîne donnée.

```
>>> miroir("machin")
'nihcam'
```

Exercice 13 : Spécifiez et écrivez une fonction `palindrome` qui teste une chaîne de caractères donnée est ou non un palindrome.

```
>>> palindrome("toto")
false
>>> palindrome("laval")
true
```

Exercice 14 : Spécifiez puis écrivez la fonction `trim` qui prend en paramètre une chaîne de caractères et qui supprime les espaces en début et en fin de chaîne, et ramène à un seul espace les blancs entre les mots :

```
>>> trim("    il     va falloir    enlever des     espaces     !    ")
'il va falloir enlever des espaces !'
>>> trim("           ")
''
```