

Séance 3b: SÉANCE 3B

Université Paris-Diderot

Objectifs:

- | | |
|--|--|
| — S'exercer à écrire du PYTHON. | — Utiliser des expressions booléennes. |
| — Manipuler des chaînes de caractères. | — Utiliser des boucles. |

Dans cette séance, vous résoudrez des exercices de difficulté et longueur variées, concernant surtout les chaînes de caractères, l'utilisation de boucles et d'expressions booléennes.

Exercice 1 (Concaténation, ☆)

Écrire une fonction `concatNTimes` qui prend en paramètre une chaîne de caractères `s` et un entier `n`, et renvoie la chaîne de caractères `s` répétée `n` fois.

Contrat:

Si `n` est négatif, on renverra la chaîne de caractères vide.

Si `n` est positif, `concatNTimes(s, n)` doit être `ss...ss` où `s` est présente `n` fois.

□

Exercice 2 (Accord, ☆)

Écrire une fonction `accord` qui prend en paramètre un nom `name`, un entier `n`, et renvoie `name` suivi d'un '`s`' si `n` est supérieur ou égal à 2, `name` sinon.

Contrat:

Par exemple, `accord("pomme", 2)` renvoie `"pommes"` alors que `accord("poire", 1)` renvoie `"poire"`.

□

Exercice 3 (Cadre, ☆)

1. Écrire une fonction `cadre` qui prend en paramètre une chaîne de caractères, et affiche cette chaîne de caractères entourée d'un cadre de taille adaptée.

Contrat:

Par exemple, `cadre("Hello World!")` doit afficher :

```
+-----+
| Hello World! |
+-----+
```

2. (☆☆) Modifier `cadre` pour qu'elle se comporte correctement dans le cas d'une chaîne de caractères qui contient le caractère '`\n`'.

Contrat:

Ainsi, `cadre("Hello\nWorld!")` doit afficher :

```
+-----+
| Hello |
```

```
| World! |  
+-----+
```

□

Exercice 4 (Voyelles, ★)

Écrire une fonction `voyelles` qui renvoie le nombre de voyelles dans une chaîne de caractères.

Contrat:

Par exemple, `voyelles("Hello World!")` doit renvoyer 3.

□

Exercice 5 (Palindromes, ★)

Un palindrome est une chaîne de caractères qui est identique dans les deux sens. Par exemple, "rotor" ou "ressasser".

1. Écrire une fonction `reverse` qui inverse le sens d'une chaîne de caractères.

Contrat:

Par exemple, `reverse("hello")` doit renvoyer "olleh".

2. Utiliser la fonction `reverse` pour écrire une fonction `palindrome` qui renvoie `True` si son argument est un palindrome, `False` sinon.
3. Écrire une autre fonction `palindrome_bis` qui fait la même chose sans utiliser la fonction `reverse`.

□

Exercice 6 (Fizz Buzz, ★★)

1. Écrire une fonction `fizzbuzz` qui prend un entier `n` en argument et affiche les entiers de 1 jusqu'à `n`, en respectant les règles suivantes :
 - les multiples de 3 sont remplacés par Fizz ;
 - les multiples de 5 sont remplacés par Buzz.Les règles se cumulent. Par exemple, à la place de 15, il faut afficher Fizz Buzz.
2. Écrire une fonction `fizzbuzzwoof` qui fait la même chose que `fizzbuzz` mais, en plus, remplace les nombres qui **contiennent** 7 par Woof.
Par exemple, à la place de 75 il faut afficher Fizz Buzz Woof, alors que 14 doit être affiché tel quel.

□

Exercice 7 (Conjugaison, ★★)

1. Écrire une fonction `conjugaison` qui prend en paramètre une chaîne de caractères et conjugue le verbe qui lui est donné.
On se limitera aux verbes du premier groupe. Ainsi, `conjugaison` doit d'abord vérifier que la chaîne de caractères qu'on lui donne ne contient que des lettres minuscules, termine par "er" et n'est pas "aller".
En cas d'erreur, afficher à la place un message qui explique le problème.

Contrat:

Par exemple, `conjugaison("parler")` doit afficher :

```
je parle  
tu parles  
il parle  
nous parlons  
vous parlez  
ils parlent
```

2. Corriger `conjugaison` pour qu'elle conjugue correctement les verbes comme "manger". (Si votre fonction est déjà correcte, tant mieux!)

□

Exercice 8 (Nombres amicaux, ***)

Vous avez déjà vu les nombres parfaits dans le Cours-TD 2, les nombres amicaux sont une notion proche.

1. Écrire une fonction `sumDiv` qui renvoie la somme des diviseurs propres d'un entier.

Contrat:

Par exemple, `sumDiv(6)` vaut 6, `sumDiv(1184)` vaut 1210.

2. Deux entiers n et m sont dits amicaux si `sumDiv(n) == m` et `sumDiv(m) == n`.
Vérifier que 1184 et 1210 sont amicaux.
3. Utiliser cette caractérisation pour trouver un couple de nombres amicaux inférieurs à 500, à l'aide d'une boucle imbriquée.
4. On veut maintenant trouver un couple de nombres amicaux supérieurs à 10000. Malheureusement, une boucle imbriquée ne sera pas assez efficace pour ça. Trouver une méthode plus directe!

□

Exercice 9 (Des nombres en toutes lettres, ***)

Le but de cet exercice est d'écrire une fonction `toLetters` qui prend en paramètre un entier positif, et le renvoie en toutes lettres.

On ne gèrera que les nombres entre 0 et 999999.

1. Écrire une fonction `oneDigit` qui prend en paramètre un entier n entre 1 et 9 et renvoie son écriture en toutes lettres.
2. Écrire une fonction `twoDigits` qui prend en paramètre un entier n entre 1 et 99 et renvoie son écriture en toutes lettres. Un peu d'aide :
 - La division entière `n // 10` permet d'obtenir le chiffre des dizaines, et le modulo `n%10` le chiffre des unités.
 - "quatre-vingt" ne prend un 's' que s'il est tout à la fin du nombre, on ne s'occupe donc pas de ce cas particulier dans cette fonction.
 - Attention aux exceptions pour 11, ..., 16 et de même, 71, ..., 76 et 91, ..., 96.
 - "un" est toujours précédé d'un "et", sauf après "quatre-vingt".
 - Tous les mots d'un nombre entre 1 et 99 sont séparés par des traits d'union, sauf "et".N'hésitez pas à écrire d'abord cette fonction sans tous les cas particuliers, et à les ajouter au fur et à mesure après-coup.
3. Écrire une fonction `threeDigits` qui prend en paramètre un entier n entre 1 et 999 et renvoie son écriture en toutes lettres.
4. Écrire la fonction `toLetter`. On pensera au cas particulier de 0, et enfin, à l'ajout d'un 's' si le nombre finit par 80 ou par une centaine au pluriel.
5. Tester la fonction `toLetter` ! On pourra par exemple vérifier manuellement les nombres de 0 à 100, puis tester sur des cas particuliers ou des nombres pris au hasard.

□