Séance 3b: Chaînes, booléens et boucles

Université Paris-Diderot

Objectifs:

- S'exercer à écrire du Java.
- Manipuler des chaînes de caractères.
- Utiliser des expressions booléennes.
- 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 (Accord, ⋆)
Écrire une fonction `withSIfNeeded` *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. On ne tentera pas de tenir compte des exceptions.*

Contrat:
Par exemple, `withSIfNeeded("pomme", 2)` *renvoie* `"pommes"` *alors que* `withSIfNeeded("poire", 1)` *renvoie* `"poire"`.

□

Exercice 2 (Cadre, ⋆)

1. *Écrire une procedure* `line` *qui prend en paramètre un entier* `n` *et qui affiche une ligne de* `n` *fois le caractère* #.

   Contrat:
   Par exemple, l'appel `line(7)` *affiche sur le terminal :*
   ```
   #######
   ```
2. *Écrire une procedure* `frame` *qui prend en paramètre une chaîne de caractères, et affiche cette chaîne de caractères entourée d'un cadre de taille adaptée.*

   Contrat:
   Par exemple, `frame("Hello World!")` *doit afficher :*
   ```
   +--------------+
   | Hello World! |
   +--------------+
   ```
   On rappelle qu'on peut obtenir la taille (le nombre de caractères) d'une chaine de caractères `s` *grâce à l'expression* `s.length`.
3. *(⋆⋆⋆)* ***Question bonus, à faire à la fin*** *Modifier* `frame` *pour qu'elle se comporte correctement dans le cas d'une chaîne de caractères qui contient le caractère* `'\n'`.

   Contrat:
   Ainsi, `frame("Hello\nWorld!")` *doit afficher :*
   ```
   +--------+
   | Hello  |
   | World! |
   ```

```
+--------+
```

□

Exercice 3 (Voyelles, ⋆)
Écrire une fonction `vowels` *qui renvoie le nombre de voyelles dans une chaîne de caractères.*

> ***Contrat:***
> *Par exemple,* `vowels("Hello World!")` *doit renvoyer* `3`.

Pour obtenir le caractère à l'indice `n` *d'une chaine de caractère* `str`*, on peut utiliser l'expression* `str.charAt(n)`*. Par exemple,* `"alibaba".charAt(2)` *est égal à* `'i'`*. On peut aussi utiliser la fonction* `characterAtPos` *définie en cours-TD, qu'il faudra recopier.* □

Exercice 4 (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 renvoyer* `ss...ss` *où* `s` *est présente* `n` *fois.*

□

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. On pourra utiliser la fonction* `reverse` *définie juste avant. On rappelle que l'on peut utiliser* `s1.equals(s2)` *pour déterminer si les chaîne de caractères* `s1` *et* `s2` *sont égales.*
3. *Écrire une autre fonction* `palindrome_bis` *qui fait la même chose sans utiliser la fonction* `reverse`.

□

Exercice 6 (Fizz Buzz, ⋆⋆)
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`. □

Exercice 7 (Premier, ⋆⋆)
Ecrire une fonction `isPrime` *qui renvoie un Booléen indiquand si son argument est un nombre premier.*

> ***Contrat:***
> *Par exemple,* `isPrime(17)` *renvoie* `true`*, alors que* `isPrime(12)` *et* `isPrime(1)` *renvoient* `false`.

□

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. Indication : On pourra utiliser une boucle imbriquée.*
4. *(Bonus :) On veut maintenant trouver un couple de nombres amicaux supérieurs à 10000. Est-ce-que la méthode que vous avez utilisée à la question précédente fonctionne encore ? Sinon, trouver une méthode qui vous permettra de déterminer un tel couple.*

□

Exercice 9 (Conjugaison, ⋆⋆)

1. *Écrire une fonction* `conjugate` *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, `conjugate` *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, `conjugate("parler")` *doit afficher :*
   ```
   je parle
   tu parles
   il parle
   nous parlons
   vous parlez
   ils parlent
   ```
2. *Corriger* `conjugate` *pour qu'elle conjugue correctement les verbes comme* `"manger"`*. (Si votre fonction est déjà correcte, tant mieux !)*

□