Séance 2: Tests, chaînes de caractères, boucles, etc.

Université Paris Cité

1 Tests

Objectifs:

- Tester des fonctions selon un contrat
- Utiliser les conditionnelles et les boucles
- Manipuler le type `int` des entiers
- Manipuler le type `String` des chaînes de caractères

L'une des grandes problématiques de la programmation informatique est le "bug", c'est-à-dire un programme qui ne termine pas ou qui ne renvoie pas le résultat attendu.

Le résultat attendu d'un programme est généralement spécifié sous la forme d'un *contrat* : il spécifie quelle est la forme des entrées que le programme attend et ce que le programme garantit alors sur ses sorties.

Il existe des techniques de preuve mathématique permettant de s'assurer qu'un programme respecte son contrat. Cependant, ces techniques ne sont pas au programme d'IP1.

Une autre méthode, moins sûre mais très répandue, permet de se convaincre de la correction du programme en faisant passer au programme un grand nombre de *tests*.

Un *test* est la donnée d'un couple formé d'une entrée I et d'une sortie attendue O respectant le contrat. Pour effectuer un test sur une fonction, on appelle cette fonction avec le paramètre I et on vérifie que la valeur de retour de la fonction est égale à la sortie attendue O définie par le test.

Important : les fonctions testées dans cette partie sont fournies mais il ne faut pas les modifier. Les tests sont en effet à programmer dans le `main`. En pratique, les fonctions ou les programmes testés ne sont pas toujours connus par le testeur. D'ailleurs, à la fin de ce TP, vous verrez comment en testant une fonction on peut arriver à comprendre ce qu'elle calcule.

Exercice 1 (Valeur Absolue, $\star$)
Le fichier `AbsoluteValue.java` *propose deux fonctions* `myAbs` *et* `newAbs` *qui prennent en paramètre un entier et retournent sa valeur absolue. Le but de cet exercice est de les tester. Les tests seront à placer dans la fonction* `main` *de ce fichier.*

1. *À l'aide de la procédure* `System.out.println`*, afficher dans le terminal la valeur de retour de la fonction* `myAbs` *évaluée sur l'entier* 0.
2. *On souhaite vérifier que la fonction* `myAbs` *vérifie le contrat suivant :*
   Contrat:

$$\begin{array}{lcl} x = \text{-}10 & \to & 10 \\ x = 0 & \to & 0 \\ x = 1 & \to & 1 \end{array}$$

Recopiez et modifiez la suite d'instructions :

```
System.out.println("Entrée : " + 0);
System.out.println("Sortie myAbs : " + myAbs(0));
```

pour afficher le résultat de chaque test de manière lisible :

```
Entrée : -10
Sortie myAbs : 10
Entrée : 0
Sortie myAbs : 0
Entrée : 1
Sortie myAbs : 1
```

3. *A l'aide d'une boucle, répétez ces tests pour toutes les entrées entre -10 et 10 inclus.*
4. *Testez les entiers entre -10 et 0 d'une part, et entre 1 et 10 d'autre part, à l'aide de deux boucles, en précisant désormais dans l'affichage si chaque test est positif par rapport au contrat suivant :*

   Contrat:
   Pour tout entier `i>=0` *passé en paramètre, la valeur de retour de la valeur absolue est* `i`.
   Pour tout entier `i<0` *passé en paramètre, la valeur de retour de la valeur absolue est* `-i`.

   Par exemple, si myAbs renvoyait -1 pour l'entrée 1 (mais ce n'est pas le cas), on afficherait

```
...
Entrée : -1
Sortie myAbs : 1 (valide)
Entrée : 0
Sortie myAbs : 0 (valide)
Entrée : 1
Sortie myAbs : -1 (erreur)
...
```

   On rappelle que la procédure `System.out.print`*, à la différence de la procédure* `System.out.println`*, ne passe pas à la ligne à la fin de l'affichage.*
5. *Modifiez votre code pour tester tous les entiers entre -100 et 99 inclus à l'aide d'une seule et même boucle, et si possible d'une seule conditionnelle.*
6. *Testez si la fonction* `newAbs` *vérifie le contrat suivant sur les paramètres compris entre* -10 *et* 100*, en affichant* `"ok"` *si le test est passé, et un message d'erreur indiquant le paramètre qui ne passe pas sinon. Pour se faire, aidez-vous d'une boucle et d'une conditionnelle.*

   Contrat:
   Si la fonction prend en paramètre un entier, alors elle renvoie un entier positif.
7. *À l'aide d'une boucle et d'une conditionnelle, comparer les fonctions* `Math.abs` *(valeur absolue de Java) et* `newAbs` *sur les entiers compris entre* -200 *et* 199 *inclus, en affichant* `"ok"` *si les deux fonctions retournent la même valeur et* `"ko"` *sinon. À partir de quelle valeur les deux fonctions ne coïncident plus ?*

□

2 Chaînes, booléens et boucles

Objectifs:

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

Exercice 2 (Cadre  , ⋆)

1. *Écrire une fonction* `line` *qui prend en paramètre un entier* `n` *et qui renvoie une chaîne de caractères contenant* `n` *fois le caractère* `'#'`.

   Contrat:
   Par exemple, l'appel `line(7)` *renvoie la chaîne de caractères "#######".*

2. *Écrire une procédure* `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, l'appel `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'`. *Dans un premier temps, ne gérer que le cas où il y a un seul retour à la ligne, puis étendre au cas d'un nombre arbitraire de retours à la ligne.*

   Contrat:
   Ainsi, l'appel `frame("Hello\nWorld!")` *doit afficher :*

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

   et l'appel `frame("Ligne 1\nLigne numéro deux\nLigne no 3")` *doit afficher :*

   ```
   +-------------------+
   | Ligne 1           |
   | Ligne numéro deux |
   | Ligne no 3        |
   +-------------------+
   ```

□

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

Contrat:
Par exemple, l'appel `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'`. □

Exercice 4 (Concaténation, ⋆)
Écrire une fonction `concatNTimes` *qui prend en paramètres 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...s` *où le caractère* `s` *est présent* `n` *fois.*

□

Exercice 5 (Palindromes  , ⋆)
Un palindrome est une chaîne de caractères dont la lecture 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, l'appel `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 chaînes de caractères* `s1` *et* `s2` *sont égales.*
3. *Écrire une autre fonction* `palindrome_bis` *qui fait la même chose sans utiliser la fonction* `reverse`. *Votre fonction compare-t-elle chaque paire de lettres une seule fois ? Si ce n'est pas le cas, comment l'améliorer ?*

□

Exercice 6 (Premier, ⋆⋆)

Rappelons qu'un nombre est premier s'il a pour seuls diviseurs `1` *et lui-même. Ecrire une fonction* `isPrime` *qui renvoie un booléen indiquant si son argument est un nombre premier.*

Contrat:

Par exemple, l'appel `isPrime(17)` *renvoie* `true`, *alors que les appels* `isPrime(12)` *et* `isPrime(1)` *renvoient* `false`.

□

Exercice 7 (Nombres amicaux 🔍, ⋆⋆⋆)

Un diviseur d'un entier n est dit propre *s'il est différent de n. Par exemple, les diviseurs propres de 6 sont 1, 2 et 3.*

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. *Ecrire une fonction* `trouverAmicaux` *permettant de trouver un couple de nombres amicaux inférieurs à* 500.
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 ? Si non, décrire une méthode qui vous permettra de trouver un tel couple.*

□

Exercice 8 (Conjugaison, ⋆⋆)

1. *Écrire une procédure* `conjugate` *qui prend en paramètre une chaîne de caractères contenant un verbe à l'infinitif, et conjugue ce verbe.*
   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, se 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, l'appel `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 !)*

□

3 Chaînes de caractères, suite

Objectifs:

- Manipuler les chaînes de caractères.
- Définir et utiliser des fonctions.
- Concevoir et programmer des algorithmes.

Dans cette partie, vous résoudrez des exercices et des problèmes sur des chaînes de caractères. Vous écrirez des boucles et définirez des fonctions intermédiaires pour rendre votre code plus lisible et concis.

Pour cette partie, on utilisera le type `char` qui représente un caractère. Si la variable `s` contient une chaîne de caractères, l'appel à la méthode `s.charAt(i)` renvoie le caractère à la position $i+1$. Par exemple si `s` vaut `"ordinateur"`, alors l'appel `s.charAt(0)` renvoie `'o'` et l'appel `s.charAt(2)` renvoie `'d'`.

Exercice 9 (Codes de caractères, ⋆)

- *Un ordinateur représente chaque caractère par un entier appelé son code Unicode. Par exemple, le caractère* `'j'` *est représenté par le code* 106*. La fonction* `charToCode` *(fournie dans le fichier CharCode.java) prend en paramètre un caractère et renvoie son code unicode. Déterminer à l'aide de cette fonction les codes associés aux caractères* `'a'`*,* `'m'` *et* `'M'`*.*
- *Écrire une procédure* `minuscule` *qui prend un caractère en argument et affiche* `"minuscule"` *si ce caractère est une lettre minuscule,* `"MAJUSCULE"` *si le caractère est une majuscule, et* `"caractère spécial"` *sinon.*

  Contrat:
  `minuscule('a')` *doit afficher* `"minuscule"`*,* `minuscule('H')` *doit afficher* `"MAJUSCULE"` *et* `minuscule('é')` *doit afficher* `"caractère spécial"`*.*

  Indice : *Les codes des caractères* `'a'` *à* `'z'` *se suivent, idem pour les codes des caractères* `'A'` *à* `'Z'`*.*
- *La fonction inverse de* `charToCode` *est la fonction* `codeToChar` *(également fournie). Cette fonction prend en argument un code Unicode et renvoie le caractère associé. Par exemple,* `codeToChar(106)` *renvoie* `'j'`*. Écrire une procédure* `alphabet` *qui affiche* `abcdefghijklmnopqrstuvwxyz` *à l'aide d'une boucle.*

□

Exercice 10 (Recherche de caractère, ⋆)

- *Écrire une fonction* `cherche` *qui prend en arguments un caractère* `c` *et une chaîne de caractères* `s`*, qui renvoie* `true` *si* `c` *apparaît dans* `s` *et* `false` *sinon.*

  Contrat:
  Par exemple, l'appel `cherche('a', "cheval")` *renvoie* `true` *tandis que l'appel* `cherche('a', "école")` *renvoie* `false`*.*
- *Modifier la fonction* `cherche` *afin qu'elle renvoie la position de la première occurrence du caractère* `c` *dans la chaîne* `s` *si le caractère est présent, et* `-1` *sinon.*

  Contrat:
  L'appel `cherche('a',"ecole")` *doit toujours renvoyer* `-1` *et tandis que l'appel* `cherche('a',"babar")` *doit maintenant renvoyer 1.*

□

Exercice 11 (Distance de Hamming, ⋆)

La distance de Hamming entre deux mots est une notion utilisée dans de nombreux domaines (télécommunications, traitement du signal...). Elle est définie, pour deux mots de même longueur, comme le nombre de positions où les deux mots ont un caractère différent. Écrire une fonction `hamming` *qui calcule la distance de Hamming entre deux mots lorsqu'ils ont la même longueur, et qui renvoie* `-1` *sinon.*

Contrat:
Par exemple, l'appel `hamming("aaba", "aaha")` *renvoie* `1`*, l'appel* `hamming("poire", "pomme`

") *renvoie* 2 *et l'appel* `hamming("stylo", "bouteille")` *renvoie* -1.

□

Exercice 12 (Scrabble et anagrammes, ⋆⋆⋆)

— *Écrire une fonction* `suppression` *qui prend en arguments un caractère* `c` *et une chaîne de caractères* `s` *et qui renvoie* `s` *dans laquelle on a supprimé la première occurrence de* `c`. *Si* `c` *n'apparaît pas dans* `s`, `suppression` *renvoie la chaîne inchangée.*

Contrat:

Par exemple, l'appel `suppression('a', "baldaquin")` *renvoie* `"bldaquin"` *et l'appel* `suppression('d', "fleur")` *renvoie* `"fleur"`.

Indication : Si `c` *est de type* `char` *et* `s` *de type* `String`, *on peut écrire* `s+c` *pour ajouter le caractère* `c` *à la fin de la chaîne* `s`.

— *Écrire une fonction* `scrabble` *qui prend en arguments deux chaînes de caractères* `mot` *et* `lettres_disponibles`, *et qui renvoie* `true` *si on peut écrire* `mot` *en utilisant au plus une fois chaque lettre de la chaîne* `lettres_disponibles` *et* `false` *sinon.*

Contrat:

Par exemple, `scrabble("maison", "auiysmzanpo")` *renvoie* `true` *et* `scrabble("bungalows", "hbteslo")` *renvoie* `false`.

Indice *: Parcourir les caractères de* `mot` *et les supprimer successivement dans* `lettres_disponibles` *avec la fonction* `suppression`.

— *Deux mots sont des anagrammes si on peut obtenir l'un à partir de l'autre en permutant les lettres. Par exemple,* `"police"` *et* `"picole"`.

Écrire une fonction `anagramme` *qui prend en argument deux chaînes* `u` *et* `v`, *renvoie* `true` *si* `u` *et* `v` *sont des anagrammes et* `false` *sinon.*

Contrat:

Par exemple, l'appel `anagramme("parisien", "aspirine")` *renvoie* `true` *tandis que l'appel* `anagramme("chaise", "disque")` *renvoie* `false`.

□

Exercice 13 (Calculatrice, ⋆⋆⋆)

Écrire une fonction `somme` *qui prend en argument une chaîne de caractères comprenant des entiers séparés par des symboles* +, *comme "7+52+3" ou "2+6+74+13" et qui renvoie le résultat de la somme. Si l'argument n'a pas le bon format, l'appel doit renvoyer* -1.

Contrat:

Par exemple, l'appel `somme("3+8")` *renvoie 11 mais les appels* `somme("+7+8")` *et* `somme("4+3+9+")` *renvoient* -1.

□

4 Encodage et décodage

Le but de cette partie est tout d'abord de comprendre l'encodage en binaire des entiers naturels et relatifs et les différentes formes d'encodage. Afin de rester dans un cadre simple, les encodages seront donnés sous forme de chaînes de caractères.

Nous verrons aussi comment encoder et décoder une chaîne de caractères à l'aide d'un codage transformant une chaîne de caractères en une autre chaîne.

Pour cette partie, il vous est donné à compléter un fichier par section (c'est à dire un fichier pour la section **??** et un pour la section **??**). Ces fichiers contiennent seulement les fonctions auxiliaires que vous aurez à utiliser, ainsi que la fonction main où vous écrirez vos tests. Il vous faudra écrire en entier les fonctions demandées.

4.1 Encodage des entiers en binaire

Exercice 14 (Encodage des entiers naturels non signés, ⋆)
Dans cet exercice. on s'intéresse à l'encodage en binaire des entiers naturels sur un octet. On rappelle qu'un octet contient 8 bits valant 0 ou 1. Les octets seront représentés par une chaîne de caractères de longueur 8 où le bit de poids faible sera situé le plus à droite. Ainsi une chaîne de caractères "$a_0a_1a_2a_3a_4a_5a_6a_7$" composée uniquement de 0 et de 1 correspondra à l'entier : $a_0*2^7+a_1*2^6+a_2*2^5+a_3*2^4+a_4*2^3+a_5*2^2+a_6*2+a_7$. Par exemple, l'encodage de l'entier 5 en binaire sera donné par la chaîne "00000101". Avec cet encodage on peut donc coder les entiers de 0 à 255.

1. *Écrire une fonction* `isBinaryEncoding` *qui prend en paramètre une chaîne de caractères et teste si elle correspond au codage d'un octet, c'est à dire si sa taille est 8 et si elle ne contient que des 0 ou des 1. Dans le cas positif elle renverra* `true`*, et sinon* `false`*. Testez votre fonction.*
2. *Écrire une fonction* `powerTwo` *qui prend en argument un entier* `n` *(supposé positif) et renvoie 2^n.*
3. *Écrire une fonction* `decode` *qui prend en paramètre une chaîne de caractères, et si cette chaîne correspond à un octet renvoie la valeur entière correspondante et sinon elle renvoie -1. Testez votre fonction.*

   Contrat:

   — `decode("11111110")` *renvoie* 254
   — `decode("10001000")` *renvoie* 136
4. *Écrire une procédure* `encodeAndPrint` *qui prend en paramètre un entier (supposé positif) et affiche son encodage en binaire à l'envers et va à la ligne. Si l'entier est négatif ou supérieur ou égal à 256, cette procédure va simplement à la ligne sans rien afficher.*

   Contrat:

   — `encodeAndPrint(254)` *affiche* 01111111
   — `encodeAndPrint(136)` *affiche* 00010001

   Indice : *Si l'on a un entier* `n` *et on veut connaître son encodage en octet "$a_0a_1a_2a_3a_4a_5a_6a_7$", on a a_7 qui vaut* `n % 2`*, a_6 qui vaut* `(n / 2) % 2`*, a_5 qui vaut* `((n / 2))/2 % 2` *etc.*
5. *Écrire une fonction* `encode` *qui prend en paramètre un entier (supposé positif) et renvoie une chaîne de caractères contenant un octet correspondant à son encodage en binaire. Si l'entier est négatif ou supérieur ou égal à 256, cette fonction renverra la chaîne vide* `""`*.*

   Contrat:

   — `encode(253)` *renvoie* `"11111101"`
   — `encode(15)` *renvoie* `"00001111"`
6. *Selon vous, à quoi doit être égal* `decode(encode(x))`*. Vérifier que cela est vrai pour toutes les valeurs de* `x` *prises entre 0 et 255.*

□

Exercice 15 (Codage inverse, ⋆⋆)
Un autre codage des entiers en binaire, similaire à celui proposé à l'exercice précédent, est obtenu en changeant les 0 en 1 et les 1 en 0. Ainsi dans l'encodage inverse, l'octet correspondant à 253 s'écrira `"00000010"`*.*

1. *Écrire une fonction* `inverse` *qui prend en paramètre une chaîne de caractères et retourne une chaîne de caractères identique dans laquelle chaque 0 est remplacé par un 1 et chaque 1 par un 0. Remarque : pour cette fonction, la chaîne de caractères peut comporter des caractères autres que 0 et 1 et n'est pas nécessairement de longueur 8.*

   Contrat:

   — `inverse("010")` *renvoie* `"101"`
   — `inverse("bob1081")` *renvoie* `"bob0180"`
2. *En utilisant les fonctions de l'exercice 1 et la fonction précédente, donnez une fonction* `encodeInv` *qui prend en paramètre un entier et renvoie une chaîne de caractères représentant un octet contenant*

l'encodage binaire inversé. Donnez également une fonction `decodeInv` *qui fait l'opération inverse, c'est-à-dire renvoie l'entier correspondant à un octet encodé en complément à un.*

Contrat:

— `encodeInv(253)` *renvoie* `"00000010"`
— `encodeInv(15)` *renvoie* `"11110000"`
— `decodeInv("11101101")` *renvoie* `18`

□

Exercice 16 (Encodage des entiers relatifs, ⋆)
Afin de coder des entiers relatifs, une technique consiste à utiliser le premier bit de l'octet comme bit de signe. Si ce bit vaut 1 alors l'entier est négatif et sinon il est positif. Avec cette technique sur un octet, on peut coder les nombres allant de 127 à -127. Avec le codage par chaîne de caractères de longueur 8 pour représenter des octets, l'entier 11 sera codé par la chaîne `"00001011"` *et l'entier -11 par la chaîne* `"10001011"`.

1. *Écrire une fonction* `isNegative` *qui prend en paramètre une chaîne de caractères correspondant à un octet et renvoie* `true` *si la chaîne est l'encodage d'un entier négatif et* `false` *sinon. On supposera dans cette question que la chaîne donnée en paramètre correspond toujours à un octet.*
2. *Écrire une fonction* `decodeNeg` *qui prend en paramètre une chaîne de caractères et qui renvoie l'entier relatif correspondant si cette chaîne correspond à l'encodage d'un entier relatif et renvoie -128 sinon.*

   Contrat:

   — `decodeNeg("00000110")` *renvoie 6*
   — `decodeNeg("11000001")` *renvoie -65*
3. *Écrire une fonction* `encodeNeg` *qui prend en paramètre un entier relatif et qui renvoie la chaîne de caractères correspondant à son encodage binaire. Si l'entier donné en paramètre n'est pas compris entre -127 et 127, la fonction renverra la chaîne vide* `""`.

   Contrat:

   — `encodeNeg(-17)` *renvoie "10010001"*
   — `encodeNeg(9)` *renvoie "00001001"*
4. *Écrire une procédure* `testFinal` *qui ne prend pas de paramètre et qui affiche* `Tout va bien` *si pour les entiers de 0 à 127, les fonctions* `encode` *(de l'exercice 1) et* `encodeNeg` *renvoient les mêmes valeurs et qui affiche* `Il y a un souci` *si pour au moins un entier, ces fonctions ne retournent pas la même valeur. (*Attention : *il ne faut pas afficher 128 fois* `Tout va bien` *mais juste une fois).*

□

4.2 Encodage/Décodage de textes (optionnel)

Exercice 17 (Le chiffrement de César, ⋆⋆)
On souhaite utiliser une méthode connue pour encoder des chaînes de caractères qui s'appelle le chiffrement de César. Le chiffrement de César correspond à un décalage de lettre. On donne la lettre qui correspond à 'A' et on peut en déduire toutes les autres. Par exemple, si on dit qu'à 'A' correspond 'C' alors à 'B' on associe 'D', à 'C' on associe 'E', . . ., à 'Y' on associe 'A' et à 'Z' on associe 'B'. Ainsi dans l'encodage où à 'A' correspond 'C', le mot 'BONJOUR' s'encode 'DQPLQWT'. On supposera dans cet exercice, que les espaces ne sont pas codés et que toutes les chaînes de caractères contiennent uniquement des espaces et des lettres majuscules.

1. *Écrire une fonction* `letterToInt` *qui prend en paramètre une chaîne de caractères contenant un seul caractère et qui renvoie sa position dans l'alphabet en partant de* 0 *si c'est une lettre en majuscule, et* -1 *sinon.*

   Contrat:

   — `letterToInt("A")` *renvoie* 0
   — `letterToInt("Z")` *renvoie* 25

— `letterToInt("*")` *renvoie* -1

Indice : *On utilisera la chaîne de caractère* `alphabet= "ABCDEFGH...XYZ"`.

2. *Écrire une fonction* `cesarLetter` *qui prend en paramètre une chaîne de caractères contenant le caractère correspondant au codage de 'A' et une chaîne de caractères contenant un autre caractère et renvoie la chaîne de caractères codant ce second caractère. Si le second caractère est un espace, alors la fonction renvoie la chaîne* `" "`*. On supposera que les paramètres auront toujours le bon format (ce n'est pas la peine de le vérifier).*

   Contrat:

   — `cesarLetter("D","B")` *renvoie* `"E"`
   — `cesarLetter("H"," ")` *renvoie* `" "`
   — `cesarLetter("C","Z")` *renvoie* `"B"`

   Attention : comme on le voit dans le dernier exemple de ce contrat, quand on arrive à la fin de l'alphabet, il faut compter modulo 26.

3. *Écrire une fonction* `cesar` *qui prend en paramètre une chaîne de caractères contenant le caractère correspondant à 'A' et une chaîne de caractères* `s` *(ne contenant que des lettres majuscules et des espaces) et renvoie la chaîne de caractères correspondant au chiffrement de César de* `s`.

   Contrat:

   — `cesar("D","BONJOUR")` *renvoie "ERQMRXU"*

4. *Écrire une fonction* `deCesarLetter` *qui prend en paramètre une chaîne de caractères contenant le caractère correspondant à 'A' et une chaîne de caractères correspondant au codage d'un caractère et renvoie le caractère original associé à ce second caractère. Si le second caractère est un espace, alors la fonction renvoie le caractère espace. On supposera que les paramètres auront toujours le bon format (ce n'est pas la peine de le vérifier).*

   Contrat:

   — `deCesarLetter("D","E")` *renvoie "B"*
   — `deCesarLetter("D","F")` *renvoie "C"*
   — `deCesarLetter("Z","B")` *renvoie "C"*

   Indice : *Utiliser une technique similaire à celle indiquée à la question 1.*

5. *Écrire une fonction* `deCesar` *qui prend en paramètre une chaîne de caractères contenant le caractère correspondant à 'A' et une chaîne de caractères* `s` *(ne contenant que des lettres majuscules et des espaces) correspondant à un message chiffré en César et renvoie la chaîne de caractères originale.*

   Contrat:

   — `deCesar("D", "ERQMRXU")` *renvoie "BONJOUR"*

6. *Dans l'encodage où à 'A' on associe 'D', qu'a voulu dire Jules César avec ce message secret 'DOHD MDFWD HVW' ?*

□