Jeudi 10 janvier 2013, N-515, Pav. Roger Gaudry

Directives:

1. Toute documentation est **permise**.
2. Les calculatrices électroniques, ordinateurs, ipad, ipod, et autres appareils du genre sont **interdits**. Vous pouvez vous servir de votre téléphone cellulaire pour avoir l'heure.
3. Inscrivez tout de suite votre **nom**, votre **code permanent** et le **numéro de votre place**.
4. Répondez **sur le questionnaire**, dans l'espace libre qui suit chaque question. Si vous manquez de place, écrivez au verso en l'indiquant très clairement.
5. L'examen est noté sur 75. Une règle de trois vous donnera votre note à cet examen (afin de la ramener à 30). Le barème est donné à titre indicatif seulement. Votre note dépendra de l'élégance de vos solutions.
6. **Conseil:** ne restez pas bloqué sur une question.

BONNE CHANCE!!

1.	_____	/14
2.	_____	/10
3.	_____	/6
4.	_____	/15
5.	_____	/10
6.	_____	/20
Total	_____	/ 75

Nom: _____ **Code permanent:** _____

Numéro de votre place: _____

1. (14 points) — Échauffement

   Dans les questions qui suivent, on fera l'hypothèse que chaque instruction présentée est la première exécutée. Que produit l'exécution des instructions suivantes:

   (a) (1 point) `print(2+3+"5");`

   (b) (1 point) `print(2+"3"+"5");`

   (c) (1 point) `print("3"+5+1);`

   (d) (1 point) `print(3-2-1*-3);`

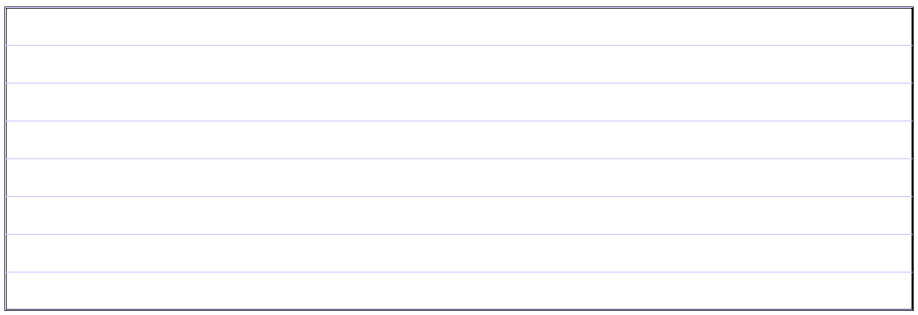
   (e) (1 point) Qu'affiche ce programme lorsqu'exécuté ?

   ```
   var a = 15;
   var f = function (a) {
       b = a+5;
       return b*b;
   };
   print(f(3), a, b);
   ```

   (f) (1 point) Qu'affiche ce programme lorsqu'exécuté ?

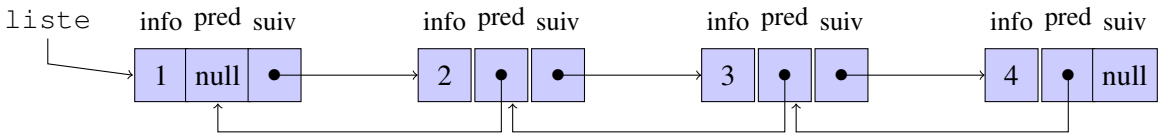
   ```
   var i = 3;
   var f2 = function () {
       for (var i=10; i>5; i--);
       return i;
   };
   print(f2(),i);
   ```

(g) (1 point) Écrire le code le plus simple pour représenter le tableau à 2 dimensions suivant qui contient 4 lignes de respectivement 4, 2 , 3 et 1 colonnes:

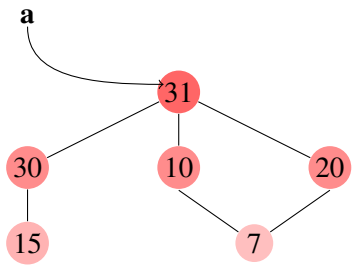


1	2	3	4
5	6		
7	8	9	
10			

(h) (3 points) Écrire le code le plus simple pour créer la liste doublement chaînée `liste`, puis le code pour retirer de cette liste le maillon portant l'information `3`.



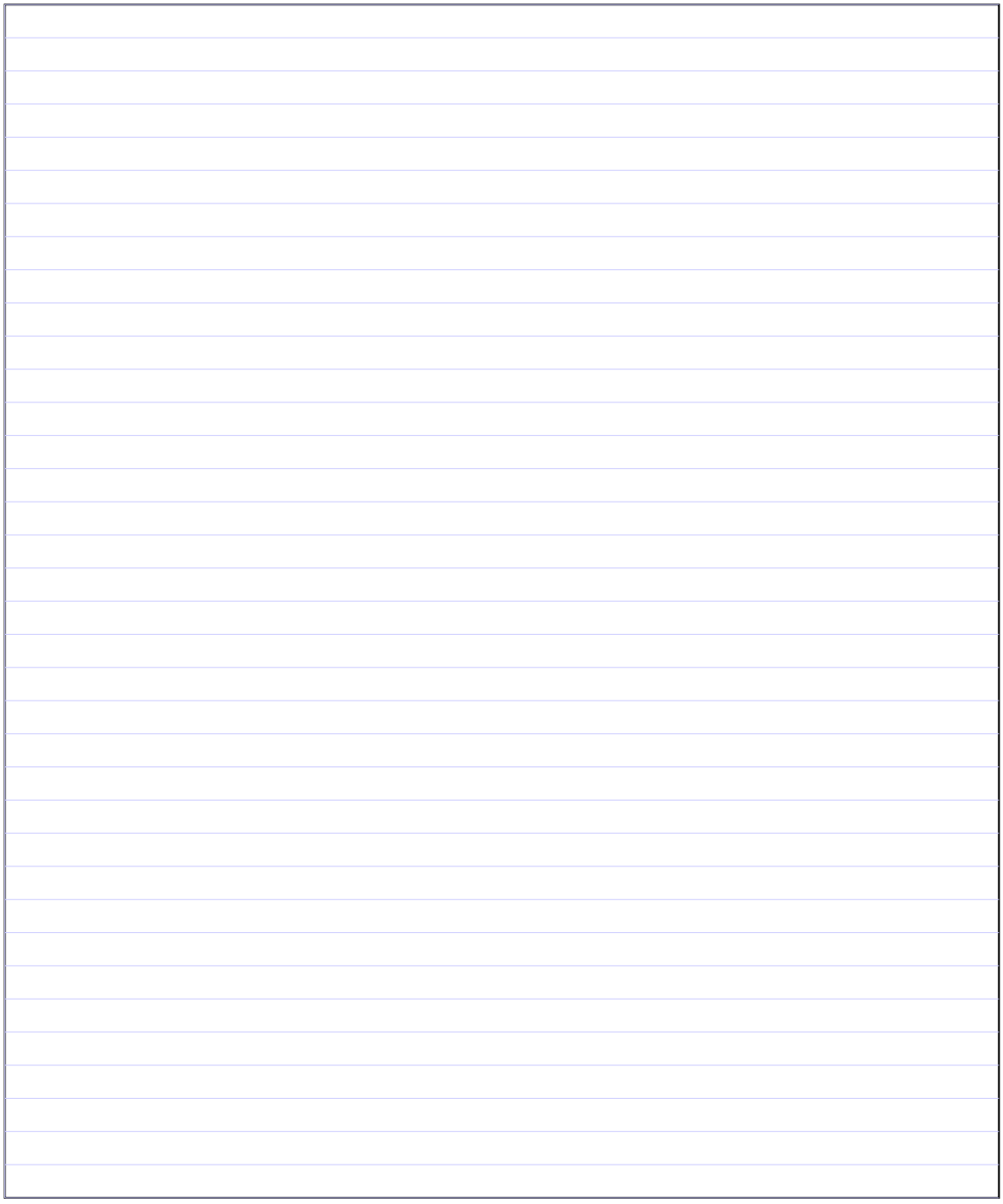
(i) (3 points) Écrire le code le plus simple pour représenter la structure référencée par `a`.



(j) (1 point) Dans quel ordre seront visités les nœuds de cette structure lors d'un parcours **en ordre infixe** si l'on supprime 30 et 15 de l'arbre de la question précédente?

2. (10 points) — Boucles simples

   (a) (5 points) Écrivez une fonction `separe1` qui prend en argument un tableau de valeurs entières et qui retourne un **nouveau tableau** tel que les valeurs paires du tableau initial soit placées avant les valeurs impaires. L'ordre relatif des éléments du tableau initial doit être respecté, de sorte que l'appel `separe([5,4,3,2,1,3,6,7,8])` retournera le tableau `[4,2,6,8,5,3,1,3,7]` et pas un autre.



(b) (5 points) Écrivez une variante de la fonction précédente (`separe2`) qui réalise la séparation des valeurs paires et impaires d'un tableau d'entiers en modifiant directement le tableau passé en argument. L'ordre relatif des valeurs n'est pas important. Vous n'avez pas le droit d'utiliser une structure (tableau, string, etc.) intermédiaire pour la réalisation de cette fonction (et donc pas non plus la fonction `separe1`).

3. (6 points) — Simplifiez-vous la vie

   (a) (2 points) Réécrivez la fonction `avant` sans en modifier la sémantique (le résultat de votre fonction doit être le même pour un argument identique) de manière à utiliser un seul `return`. Vous ne devez pas utiliser d'autre variable que celles déjà utilisées. Vous ne devez pas utiliser de `break`, de `continue` ou `goto`.

```
var avant = function (t) {
    for (var i=0; i<t.length; ++i)
        if (t[i] > 5)
            return true;
    return false;
};
```

   (b) (2 points) Réécrivez la fonction `avant2` sans en modifier la sémantique de manière à n'utiliser qu'un seul `return`. Vous ne devez pas utiliser d'autre variable que celles déjà utilisées. Vous ne devez pas utiliser de `break`, de `continue` ou `goto`.

```
var avant2 = function (t) {
    for (var i=0; i<t.length; ++i) {
        if (t[i] < 10)  return true;
        if (t[i] >= 20) return true;
    }
    return false;
};
```

(c) (2 points) Simplifiez le plus possible la fonction `avant3` sans en modifier la sémantique. Vous supposerez que `i` est toujours positif ou nul lors de l'appel de cette fonction. Vous ne devez pas utiliser de `break`, de `continue` ou `goto`.

```
var avant3 = function (i) {
    var sum = 0;
    do {
        i += 2;
        if (i >= 10) i = 0;
        else
            sum += i;
    } while (i > 0);
    return sum;
};
```

4. (15 points) — Récursivité

Considérez le code suivant où `even` est la fonction qui retourne `true` si et seulement si son argument est un nombre entier pair:

```
var quiz1 = function(i,r) {
    if (i === 0) return r;
    if (even(i))
        return quiz1(i/2,r*2);
    return quiz1(i-1,r);
};
```

(a) (2 points) Que retourne l'exécution de: `quiz1(-2,1)` ?

(b) (2 points) Que retourne l'exécution de: `quiz1(136,1)` ?

(c) (2 points) Que retourne l'exécution de: `quiz1(256,1);` ?

(d) (3 points) Expliquez de manière claire ce que retourne l'appel `quiz1(n,1)` pour tout `n` positif.

Considérez le code suivant où `s.slice(i,j)` retourne (sans modifier `s`) la sous-chaîne de `s` comprise entre les indices `i` (inclus) et `j` (exclus):

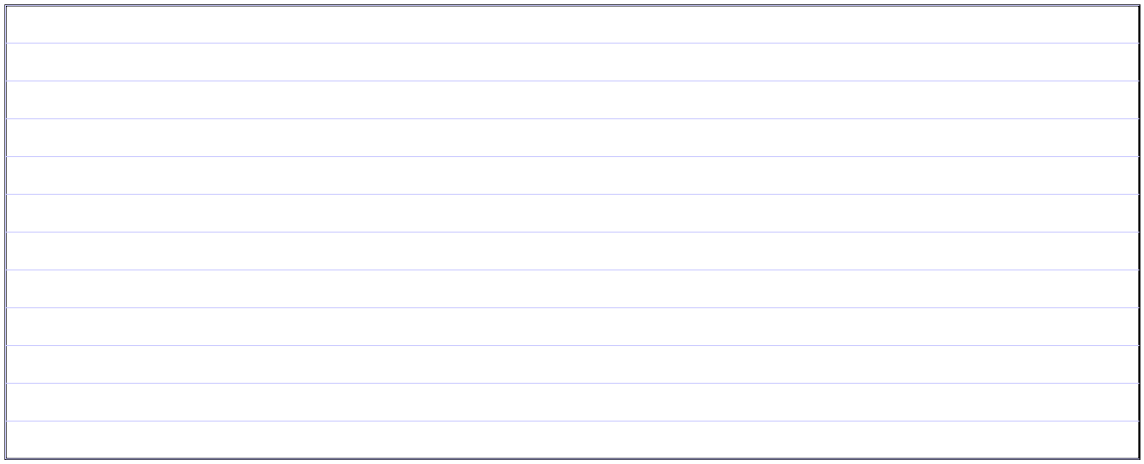
```
var quiz2 = function (s) {
   if (s.length <= 1)
       return s;
    return s.charAt(s.length-1)
       + quiz2(quiz2(s.slice(1,s.length-1)))
       + s.charAt(0);
};
```

(e) (1 point) Que retourne l'exécution de: `quiz2("a")` ?

(f) (3 points) Que retourne l'exécution de: `quiz2("abcdef");` ?

(g) (2 points) Écrire une fonction récursive qui implémente la fonction d'Ackerman définie pour tous entiers n et m positifs ou nuls par:

$$A(m,n) = \begin{cases} n+1 & \text{si} \quad m = 0 \\ A(m-1,1) & \text{si} \quad m > 0 \text{ et } n = 0 \\ A(m-1,A(m,n-1)) & \text{si} \quad m > 0 \text{ et } n > 0 \end{cases}$$



5. (10 points) — Algorithmique

   (a) (5 points) Écrire une fonction `isSC(sc,s)` qui retourne `true` si la chaîne `sc` est une sous-chaîne de la chaîne `s` et `false` sinon. Une sous-chaîne de `s` est toute chaîne obtenue en lisant un nombre quelconque de caractères de la gauche vers la droite en autorisant des sauts de caractères. Par exemple `ift` est une sous-chaîne de **`i`**`n`**`f`**`orma`**`t`**`ique`, tout comme le sont les chaînes `info`,

`fort`, `raie`, `forme`, `nrm`, `informatique`, etc.

(b) (5 points) Écrire une méthode `magic(t)` qui retourne `true` si le tableau `t` à une dimension passé en argument représente un carré magique. Un carré magique est une matrice carrée dont toutes les lignes, toutes les colonnes et les deux diagonales somment à la même valeur. Par exemple, l'appel `magic([16,3,2,13,5,10,11,8,9,6,7,12,4,15,14,1])` devrait retourner `true` car la somme de toute colonne, ligne ou diagonale du carré correspondant est `34`, tandis que l'appel `magic([1,2,3,4,5,6,7,8,9])` devrait retourner `false`. On supposera que le tableau passé en argument représentera toujours une matrice carrée (nombre de lignes = nombre de colonnes).

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

6. (20 points) — POO

   Vous allez dans ce problème simuler une version simplifiée du jeu de pendu. Dans ce jeu, un joueur tente de trouver un mot secret choisi par la machine en le devinant lettre après lettre. Le joueur gagne s'il réussit à trouver le mot en un nombre maximum de coups fixé au début de la partie. Voici une trace d'une partie en (maximum) 6 coups où la machine a choisi aléatoirement le mot `arbre` et où le joueur propose tour à tour les lettres `e`, `b`, `r`, `c`, `t` et `u`. À chaque lettre proposée, la machine indique au joueur le nombre d'occurrences de cette lettre dans le mot secret et dévoile l'ensemble des lettres trouvées depuis le début de la partie. Dans la partie illustrée, le joueur échoue en ayant réussi à identifier en 6 coups seulement 4 des 5 lettres du mot secret (`arbre`).

```
secret: _ _ _ _ _
guess: e found:  1
coups: 1/6 found: 1
secret: _ _ _ _ e
guess: b found:  1
coups: 2/6 found: 2
secret: _ _ b _ e
guess: r found:  2
coups: 3/6 found: 4
secret: _ r b r e
guess: c found:  0
coups: 4/6 found: 4
secret: _ r b r e
guess: t found:  0
coups: 5/6 found: 4
secret: _ r b r e
guess: u found:  0
vous avez perdu: _ r b r e
le mot etait: arbre
vous avez trouve seulement 4 lettres sur 5
```

   Vous ferez l'hypothèse dans cette question de l'existence d'un objet dictionnaire `dico` qui possède deux propriétés: `size` et `get`. La première vaut le nombre de mots connus du dictionnaire, la seconde est une méthode qui prend en argument une valeur entière positive ou nulle `i` et qui retourne le `i+1`ème mot du dictionnaire (`i` est inférieur à `dico.size`). C'est à l'aide de ce dictionnaire que le programme sélectionnera le mot secret au début de chaque partie.

```
var dico = new Dictionnaire();
print("nb de mots dans dico: ", dico.size);
print("3e mot dans dico:", dico.get(2));
```

   La suite de l'énoncé spécifie un ensemble de fonctions et méthodes que vous devez coder. Voici un programme qui devrait fonctionner avec votre code et qui a été utilisé pour générer la trace vue plus haut (ceci vous impose donc un certain nombre de contraintes sur votre code):

```
var p = new Pendu(dico);
var a,guess = "ebrctuaghciujk";

print(p);
for (var i=0; i<guess.length; ++i) {
    if (!p.hasLost()) {
        a = p.guess(guess[i]);
        print("guess:", guess[i], "found: ", a);
        print(p);
    }
}

if (p.hasWon())
    print("vous avez trouve en", p.getCoupsJoues());
else
    print("vous avez trouve seulement",  p.getNbLetterFound(),
          "lettres sur",  p.getSize());
```

(a) (5 points) Identifiez l'**état** qui définit un objet `Pendu` qui permet de représenter une partie de pendu. Écrivez ensuite le constructeur `Pendu` qui doit prendre en argument une référence sur un dictionnaire, ainsi que le nombre maximum de coups à jouer pour trouver le mot secret. Si cette valeur n'est pas spécifiée, alors le nombre maximum de coups est de 6.

(b) (4 points) Écrire la méthode `guess(c)` qui prend en argument une lettre et qui retourne le nombre d'occurrences de cette lettre dans le mot secret.

(c) (4 points) Écrire les prédicats `hasWon` et `hasLost` qui retourne `true` selon que le joueur a gagné ou perdu et `false` sinon.

(d) (3 points) Écrire une méthode `toString` qui permet au programme illustré plus haut de s'exécuter comme indiqué.

(e) (4 points) Écrivez le code des méthodes que vous pensez pertinentes pour ce problème ou que le code vu plus haut requiert.