Tutoriel CarScripts

Alain Busser

August 7, 2011

Contents

Les exercices décrits dans ce document sont conçus pour servir d'introduction au paradigme des CarScripts (programmes en JavaScript sous CaRMetal). Ils sont souvent très brefs, et découpés en trois étapes:

1. Écriture du programme;
2. Test du programme et obtention d'un résultat;
3. Analyse du résultat obtenu.

Seules les étapes 1 et 3 seront données dans le document pdf. Donc, à moins d'être très champion en JavaScript (et dans ce cas le document est inutile) la deuxième étape ne peut être accessible qu'en copiant le script donné dans le document dans l'éditeur JavaScript de CaRMetal, et en cliquant sur l'icône ressemblant à un feu vert. En bref, ce document n'est pas une lecture de chevet mais est destiné à être lu devant un ordinateur allumé sur lequel CaRMetal (version supérieure à 3.0) a été préalablement installé, et sur lequel on manipule vraiment...

Variables numériques

Affectation

Création

On va commencer par du très court: Afficher le contenu de x et aller à la ligne ("Print and go to the line" s'abrège en "Println"):

```
Println(x);
```

Que s'est-il passé? Si si, il s'est passé quelque chose, mais c'est en bas de la fenêtre de l'éditeur de scripts. Somme toute c'est logique: Comment peut-on afficher le contenu de quelque chose qui n'existe pas?

Pour faire mieux, on rajoute une ligne où on crée x (par l'instruction "var" qui est une abréviation de "variable"; en effet x est une variable):

```
var x;
Println(x);
```

Là il s'est passé quelque chose, dans une fenêtre créée pour les sorties du programme (penser de temps en temps à les refermer; également cliquer systématiquement

sur l'icône d'annulation juste à gauche du feu vert: C'est plus prudent...). L'affichage n'est peut-être pas non plus celui qu'on espérait. En effet, *JavaScript* n'initialise pas les variables lors de leur création. x a été créé mais pour l'instant ce n'est ni un nombre, ni un tableau de nombres, ni autre chose non plus. Pour que x soit un nombre, il faut l'***affecter***, c'est-à-dire lui donner une valeur numérique:

```
var x=2;
Println(x);
```

On peut aussi séparer la création et l'affectation:

```
var x;
x=2;
Println(x);
```

Essayer maintenant:

```
var x="2";
Println(x);
```

Bien que l'effet produit soit le même, il ne s'est pas passé la même chose: Dans le cas présent, x n'est pas un nombre, mais une chaîne de caractères (voir le chapitre suivant) que "Println" affiche. Alors que précédemment, x était un nombre, que "Println" convertissait en chaîne de caractères afin de l'afficher: C'est donc la chaîne "2" et non le nombre 2 qui est affiché dans les deux cas.

Modification

J'ai créé une variable x égale à 2 mais comme je suis versatile, j'ai changé d'avis et me dis que finalement, j'eûsse préféré que x soit après tout égal à 3. Pour cela, il suffit que je remplace x par 3:

```
var x=2;
x=3;
Println(x);
```

La valeur de x est bien celle voulue.
Variante importante: Au lieu de purement et simplement remplacer x par 3, on peut l'***incrémenter***, en le remplaçant par "un de plus que lui-même":

```
var x=2;
x=x+1;
Println(x);
```

L'écriture d'une égalité entre deux nombres qui ne sont visiblement pas égaux entre eux étant un peu choquante pour un logicien, au lieu d'écrire l'affectation avec un autre symbole que l'égalité (par exemple le "$\leftarrow$" qui se lit "prend"), les concepteurs du langage JavaScript ont préféré écrire l'incrémentation avec la notation $x+=1$ (pour "on rajoute un pas de 1 à x"):

```
var x=2;
x+=1;
Println(x);
```

Une variante propre à l'incrémentation proprement dite (c'est-à-dire avec un pas de 1) est le célèbre "$x++$" hérité du langage "C":

```
var x=2;
x++;
Println(x);
```

Cette dernière variante est particulièrement commode pour des boucles (par exemple, si on veut construire un polygone à 3096 côtés: On peut!)

Affectations multiples

Un exercice typique de contrôle de Seconde est celui-ci:

- $x \leftarrow 3$
- $x \leftarrow x + 1$
- $x \leftarrow x * x$

Dans lequel la question porte évidemment sur la valeur finale de x...
En JavaScript cela donne

```
var x=3;
x=x+1;
x=x*x;
Println(x);
```

Certains élèves de Seconde ont vraiment du mal à s'y retrouver dans ces circonvolutions temporelles, l'utilisation de la même lettre x désignant plusieurs nombres différents (la même variable mais à des instant différents. Il est possible de cacher l'aspect temporel du problème en utilisant les cases successives d'un tableau, ou simplement des lettres différentes:

```
var a=3;
b=a+1;
c=b*b;
Println(c);
```

Des chaînes qui ont du caractère

Définitions

Chaînes attachantes

Un mot, n'importe quel champion de *Scrabble* le confirmera, c'est des lettres (ou des caractères) mises bout à bout, un peu comme des maillons qui se suivent sur une chaîne. D'où le nom de chaîne de caractères donné à un texte (d'éventuellement plusieurs mots, l'espace entre les mots étant lui-même considéré comme un caractère). En JavaScript, comme on l'a vu dans le chapitre précédent, les chaînes sont juste placées entre guillemets. La traduction anglaise de "chaîne de caractères" est "string", l'image anglo-saxonne étant des perles sur une corde (un collier) au lieu des maillons d'une chaîne...

Concaténation

Définition

L'opération consistant à juxtaposer deux mots pour en obtenir un nouveau s'appelle donc ***concaténation*** (en latin, ça consiste à attacher deux chaînes courtes pour en obtenir une plus longue, en mettant le maillon terminal de la première dans le maillon initial de la seconde). C'est une opération importante en linguistique puisque les langues dites agglutinantes fabriquent des néologismes par ce biais. Par exemple, à partir des deux mots malgaches *vato* (pierre) et *kely* (petit) a été formé le mot malgache *vatokely* qui désigne un caillou.
En JavaScript, la concaténation se note par un "+" comme pour l'addition des nombres. En effet il n'y a pas de risque de confusion puisque les objets ne sont pas de même nature.

```
Println("CaR"+"Metal");
```

À noter que la concaténation se note de la même manière en CaRMetal, que ce soit avec l'objet "texte" ou dans les propriétés des objets (alias en particulier).

Propriétés

- Par convention, la concaténation est associative. Sinon il faudrait trop de parenthèses pour écrire des mots longs.
- La chaîne vide "" (deux fois le symbole de guillemets avec rien dedans) est élément neutre pour la concaténation.
- Par contre, la concaténation n'est pas commutative, sinon les notions d'anagramme et de palindrome n'existeraient pas.

Un ensemble muni d'une loi ayant ces propriétés s'appelle un ***monoïde associatif***. Si de surcroît chaque lettre possédait un inverse, l'ensemble des mots serait un groupe, présenté par *générateurs et relations*...

Échappement

La chaîne de caractères formée d'un "backslash" concaténé à un "n" a pour effet d'aller à la ligne. Donc

```
Println("Je ne suis pas un numéro"+"\n"+"Je suis un Homme");
```

a le même effet que

```
Println("Je ne suis pas un numéro");
Println("Je suis un Homme");
```

On appelle ce genre de chaînes des ***séquences d'échappement***. Lorsque le "backslash" est suivi de la lettre "u", les 4 caractères suivants sont considérés comme des caractères unicode. Par exemple, $\mathbb{R}$ peut s'écrire

```
Println("]-"+"\u221E"+";+"+"\u221E"+"[");
```

En effet la chaîne de caractères unicode "221E" code le symbole de l'infini.

Des chiffres et des lettres

Mélange

On a vu dans le chapitre précédent que lorsqu'on attend une chaîne de caractères et qu'à la place, on trouve un nombre, celui-ci est automatiquement converti en chaîne de caractères:

```
var x=2;
Println("Actuellement x vaut "+x);
```

Il est donc possible de bricoler des chaînes de caractères mixtes, avec des morceaux de nombres dedans. Le symbole "underscore" permet, comme on le verra au chapitre 4, de raccourcir encore le bricolage en évitant de noter la concaténation (il permet de "mettre le contenu de x" directement dans la chaîne en le convertissant en chaîne) :

```
var x=2;
Println("Actuellement x vaut _x");
```

En boucle

L'exemple suivant est extrait d'un article fondamental sur l'itération:
http://www.reunion.iufm.fr/recherche/irem/spip.php?article232
Seulement l'usage de la lettre "x" pour représenter le symbole de multiplication est quelque peu malheureux, les risques de confusion existant encore chez certains lycéens. Alors on va utiliser le symbole unicode correspondant, de code "00D7" ce qui donne ceci:

```
for(n=1;n<10;n++){
    Println(n+"\u00D7"+9+"="+n*9);
}
```

Création de points

La manipulation de chaînes de caractères permet de faire construire par JavaScript, des textes complexes ou répétitifs, comme des programmes JavaScript (tout un programme) ... ou des figures CaRMetal. En effet une figure est un fichier texte au format "xml". On va voir ce que ça donne sur une figure très simple, formée d'un ou deux points !

Un point

On peut créer le point de coordonnées (2;3) par

```
a=Point(2,3);
Println(a);
```

Comme on l'a vu au premier chapitre, la variable a a été affectée, et on voit qu'elle contient une chaîne de caractères. Que représente-t-elle ? Pour le vérifier, on peut chercher à modifier les propriétés du point créé.
En TP, les élèves (et leur prof !) utilisent des expressions comme "le point a" parce que a se manipule comme s'il était un point (on peut le bouger avec la souris ou avec "Move(a,x,y)" en JavaScript). Au lieu de dire "construire le cercle de rayon 1 dont le nom du centre est stocké dans a", on préférera dire "construire le cercle de rayon 1 et de centre a", c'est-à-dire traiter a comme si c'était un point. Ce raccourci (analogue à celui qui fait appeler hauteur la longueur du segment au lieu du segment lui-même) est très pratique. Mais des difficultés surviendront dès qu'on oublie trop que c'est un raccourci...
Si on n'a pas besoin du nom du point, on n'a évidemment pas besoin d'afficher ledit nom, ni même de le stocker dans une variable, et le script suivant

```
Point(2,3);
```

crée aussi le point, seulement on ne sait pas avec JavaScript comment il s'appelle.

Deux points

On ne peut pas bouger un point par

```
a=Point(2,3);
a=Point(4,5);
Println(a);
```

En effet on voit sur la figure que deux points ont été créés. Et la variable *a* a été affectée deux fois, avec des chaînes différentes. On reconnaît d'ailleurs le nommage automatique des points par CaRMetal. L'instruction "Point" réalise donc les opérations suivantes:

- Récupérer les coordonnées du point, entre parenthèses après "Point";
- demander à CaRMetal de créer le point, avec pour coordonnées initiales, celles qui ont été fournies;
- récupérer auprès de CaRMetal, le nom du point.

Comme à la fin du premier chapitre, on peut améliorer le script précédent en

```
a=Point(2,3);
b=Point(4,5);
Println(a);
Println(b);
```

Nommage

On peut aussi choisir de nommer soi-même le point au lieu de laisser CaRMetal s'en occuper:

```
a=Point("P1",2,3);
Println(a);
```

Par contre, si on change le nom du point sous CaRMetal, la variable a contiendra toujours "P1".
Si maintenant on essaye de donner le même nom à deux points différents:

```
a=Point("P1",2,3);
b=Point("P1",4,5);
Println(a);
Println(b);
```

On voit tout l'intérêt qu'il y a à récupérer les noms des objets ! Le rajout de "Println" à tout va est d'ailleurs le meilleur outil de débogage qui soit.

Segment

```
a=Point(2,3);
b=Point(4,5);
s=Segment(a,b);
Println(s);
```

Rien de nouveau ici, à part le nom du segment (en minuscules comme il se doit puisque c'est une ligne). Par contre on constate que les deux variables fournies à "Segment" sont des chaînes de caractère:

```
a=Point(2,3);
b=Point(4,5);
s=Segment("P1","P2");
Println(s);
```

Pour peu que la figure ait été vierge au moment du lancement du script, et que les points ne soient pas renommés automatiquement "A", "B", "C" etc., le script a le même effet que le précédent.

Triangle

Un triangle est un polygone à trois côtés. Il faut donc fournir la liste (sous forme d'une chaîne de caractères) de ses sommets:

```
a=Point("A",2,3);
b=Point("B",4,5);
c=Point("C",3,0);
p=Polygon("A,B,C");
Println(p);
```

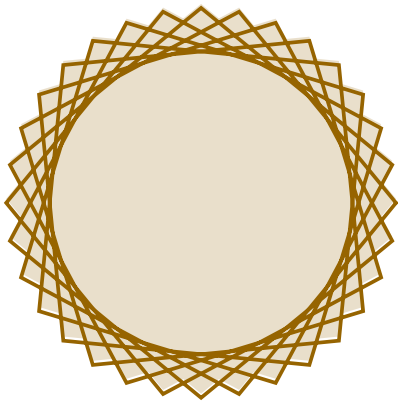
Il a donc été nécessaire de nommer les sommets pour construire la chaîne de caractères "A,B,C" dont on a eu besoin dans l'instruction "Polygon". Et bien non ! On aurait très bien pu construire la chaîne par concaténation:

```
a=Point(2,3);
sommets=a;
b=Point("B",4,5);
sommets=sommets+","+b;
c=Point("C",3,0);
sommets=sommets+","+c;
p=Polygon(sommets);
Println(p);
```

À la fin de la ligne 2, la variable *sommets* contient la même chose que a, soit "P1" (par exemple).
À la fin de la ligne 4, après concaténation d'une virgule et du contenu de b (ici "P2"), *sommets* contient maintenant "P1,P2".
De même après la ligne 6, la variable *sommets* contient "P1,P2,P3" comme on aurait pu le vérifier par un $Println(sommets)$. Et le triangle se construit correctement.
Cette méthode permet de construire des polygones ayant beaucoup de sommets comme le flocon de neige de Von Koch, ce qui est très puissant parce que CaRMetal permet d'attacher un point à l'intérieur d'un tel polygone !
Un triacontakaidigone à titre d'exemple:

```
a=Point(1,0);
SetHide(a,true);
sommets=a;
for(i=1;i<=32;i++){
    a=Point(Math.cos(Math.PI/16*7*i),Math.sin(Math.PI/16*7*i));
    SetHide(a,true);
    sommets=sommets+","+a;
}
p=Polygon(sommets);
```

Une fois rempli, le polygone ressemble à ceci, et les mouvements d'un point attaché à l'intérieur de cet objet sont indescriptibles:



Une liste de points peut aussi être utile lorsqu'on souhaite modifier leur aspect (par exemple les cacher) tous en même temps.

Manipulation de coordonnées

Des chaînes comme coordonnées

```
a=Point("A","2","3");
```

On remarque la différence par rapport au chapitre précédent: Les coordonnées de A ne sont plus des nombres mais des chaînes de caractère. On remarque que pourtant ça marche comme avant !

Symétrique

```
a=Point("A","2","3");
b=Point("B","-1","2");
c=Point("C",2*X(b)-X(a),2*Y(b)-Y(a));
Println(X(c));
```

C est bien le symétrique de A par rapport à B comme on le vérifie avec son abscisse. Mais si on bouge A ou B avec la souris (on peut), on voit que C ne bouge pas, et ne ***reste*** pas le symétrique de A par rapport à B. Donc $X(a)$

(ou $X(\text{"}A\text{"})$ ce qui revient au même puisque a contient le texte "A") désigne l'abscisse de A au moment où on invoque la fonction X. Si après ça on bouge A, son abscisse change et il faudrait actualiser les coordonnées de C à chaque mouvement de A ou de B.

```
a=Point("A","2","3");
b=Point("B","-1","2");
c=Point("C","2*x(B)-x(A)","2*y(B)-y(A)");
Println(X(c));
```

A priori l'effet est le même, mais en fait les chaînes de caractère qui définissent les coordonnées de C sont maintenant écrites dans le langage de CaRMetal comme en témoignent les "x" et les "y" écrits en minuscule (X("A") désigne l'abscisse du point A en JavaScript, et x(A) en CaRMetal). Si on vérifie (avec l'outil modification) l'abscisse de C, on voit qu'elle n'est pas affichée sous forme numérique mais sous forme du texte "2*x(B)-x(A)", ce qui la fait dépendre de celles de A et B: Le point C est ***dynamiquement*** le symétrique de A par rapport à B et le reste si on bouge A ou B.

```
a=Point(2,3);
b=Point(-1,2);
c=Point("2*x(b)-x(a)","2*y(b)-y(a)");
Println(c);
```

Quelle est l'erreur de syntaxe dans les coordonnées de c? Déjà l'abscisse est syntaxiquement incorrecte puisque pour la calculer, CaRMetal utilise $x(a)$ qui est l'abscisse de a. Or ***il n'existe aucun point de nom a dans la figure***. Il n'existe pour l'instant que deux points, et leurs noms sont $P1$ et $P2$ (en majuscule), les noms a et b étant ceux de deux variables JavaScript qui font partie du script, mais pas de la figure.
Si on veut fabriquer sous forme d'une chaîne de caractères l'abscisse de c en utilisant les noms des points stockés dans a et dans b, on doit concaténer des chaînes:

```
a=Point(2,3);
b=Point(-1,2);
c=Point("2*x("+b+")-x("+a+")","2*y("+b+")-y("+a+")");
Println(c);
```

(on constate le raccourci de pensée "l'abscisse de c" au lieu de "l'abscisse du point dont le nom est stocké dans c). Cette fabrication de chaîne complexe et hybride par concaténation de morceaux [1], est nécessaire pour produire des figures dynamiques. Mais le caractère "underscore" permet un raccourci:

```
a=Point(2,3);
b=Point(-1,2);
c=Point("2*x(_b)-x(_a)","2*y(_b)-y(_a)");
Println(c);
```

La chaîne $\_b$ représente le contenu de b (et pas le nom b lui-même), inséré dans la chaîne à la place du nom b. On peut même encore abréger sans parenthèses (autre raccourci):

```
a=Point(2,3);
b=Point(-1,2);
c=Point("2*x_b-x_a","2*y_b-y_a");
Println(c);
```

[1] on voit mieux ce qui est produit, en rajoutant des "Println()" qui affichent les coordonnées en cours de calcul. Dans l'exemple ci-dessus $Println("2*x("+b+")-x("+a+")")$ produit "2*x(P2)-x(P1)"

Nommage automatique

On souhaite créer des noms de points automatiquement comme dans CaRMetal, égaux respectivement à "P1", "P2", "P3" etc. Et bien sûr on va le faire automatiquement dans une boucle:

```
for(i=1;i<=9;i++){
    Println("P"+i);
}
```

Peut-on faire la même chose sans le signe "+" de la concaténation?

```
for(i=1;i<=9;i++){
    Println("Pi");
}
```

C'est de mal en π, pardon, en pis ! Là encore la solution vient de l'underscore:

```
for(i=1;i<=9;i++){
    Println("P_i");
}
```

Table de multiplication

```
for(n=1;n<10;n++){
    p=n*9;
    Println("_n \u00D7 9 = _p");
}
```

Cette version "améliorée" de la table de multiplication par 9 clôt ce tutoriel.