Chapitre 1

Script Smalltalk Dr. Geo

Dr. Geo est une application dynamique écrite en Smalltalk. Cela signifie qu'il est possible de modifier Dr. Geo alors qu'il est en cours de fonctionnement. Nous avons exploité cette possibilité pour définir dans Dr. Geo des items de figure qui sont en fait des scripts Smalltalk – des bouts de codes – pour étendre dynamiquement, à l'infini, les possibilités de Dr. Geo. Mais qu'est-ce que Smalltalk ?

> *Smalltalk est un langage de programmation orienté objet, réflexif et dynamiquement typé. Il fut l'un des premiers langages de programmation à disposer d'un environnement de développement intégré complètement graphique. Il a été créé en 1972. Il est inspiré par Lisp et Simula. Il a été conçu par Alan Kay, Dan Ingals, Ted Kaehler, Adele Goldberg au Palo Alto Research Center de Xerox. Le langage a été formalisé en tant que Smalltalk-80 et est depuis utilisé par un grand nombre de personnes. Smalltalk est toujours activement développé.*
>
> *Smalltalk a été d'une grande influence dans le développement de nombreux langages de programmation, dont : Objective-C, Actor, Java et Ruby.*
>
> *Un grand nombre des innovations de l'ingénierie logicielle des années 1990 viennent de la communauté des programmeurs Smalltalk, tels que les Design Patterns (appliquées au logiciel), l'Extreme Programming (XP) et le refactoring. Ward Cunningham, l'inventeur du concept du Wiki, est également un programmeur Smalltalk.* [1]

Cet extrait de la préface du livre *Pharo By Example* [2] – `http://pharobyexample.org` – décrit précisément la plate forme Smalltalk utilisée pour Dr. Geo :

> *Pharo est une implémentation moderne, libre et complète de l'environnement et langage de programmation Smalltalk.*
>
> *Pharo s'attache à offrir une plate forme robuste et stable pour du développement professionnel en langages et environnement dynamiques.*

Dr. Geo exploite l'environnement Smalltalk pour proposer, d'une part, un environnement convivial d'écriture de scripts et, d'autre part, pour donner accès à l'interface des items géométriques ou numériques constitutifs d'une figure. L'interface est en fait l'ensemble des méthodes d'instance – fonctions de classe – de ces items.

Ainsi l'utilisateur peut écrire des scripts pour manipuler les items des figures ; et puisque ces scripts sont des items de figure au même titre que d'autres, ils n'ont pas besoin d'être dans un fichier séparé, ils sont enregistrés dans le fichier de la figure.

L'autre grande force des script est de s'appuyer sur l'environnement de développement de Pharo Smalltalk ; l'utilisateur bénéficie ainsi d'outils évolués pour mettre au point ses

1. Article Smalltalk de Wikipédia en français (`http://fr.wikipedia.org/wiki/Smalltalk`). Page consultée le : 2 janvier 2011 15 :47 UTC. Contenu soumis à la licence CC-BY-SA 3.0 (`http://creativecommons.org/licenses/by-sa/3.0/deed.fr`).
2. Une traduction en français est en cours de finalisation.

scripts : navigateur de classes, inspecteur, débogueur, etc. L'utilisateur souhaitant exploiter au mieux la puissance des scripts est donc invité à étudier le livre *Pharo By Example* – `http://pharobyexample.org`, il y apprendra le langage Smalltalk et son environnement.

1.1 Script par l'exemple

Les scripts ont deux facettes :
- l'écriture du script lui même ;
- l'utilisation du script dans la figure, un même script est utilisable plusieurs fois, avec des paramètres différents.

L'outil pour créer ou éditer un script est disponible depuis le menu `Numériques>Éditer un script` , la fonction est aussi disponible depuis la barre d'outils.

L'outil pour utiliser un script est disponible depuis le menu `Numériques>Utiliser un script` , la fonction est aussi disponible depuis la barre d'outils.

Un script peut recevoir de 0 à n paramètres d'entrée. Après avoir choisi un script à insérer dans la figure, il suffit de choisir, dans la construction, les paramètres d'entrée puis de cliquer quelque part sur le fond de la figure pour y placer le résultat du script.

Dans la suite nous vous proposons de travailler sur quelques exemples de scripts, leurs fonctionnalités et leur puissance seront plus facilement analysées. Les scripts, comme les macro-constructions, donnent une dimension particulière à Dr. Geo, ils permettent – chacun avec un positionnement différent [3] – d'aller là où les auteurs du logiciel ne sont pas allés ou ne souhaitent pas aller.

Il est aussi important de comprendre que la plupart des fonctionnalités de Pharo Smalltalk sont disponibles depuis les scripts. C'est particulièrement vrai pour ses bibliothèques de fonctions [4], nous allons bien sûr les utiliser intensément.

1.1.1 Script sans paramètre d'entrée

Premier exemple : La procédure pour créer un script sans paramètre d'entrée est la suivante :

1. **Édition du script**

   (a) Choisir `Numériques>Éditer un script` dans le menu. L'éditeur de script s'affiche alors :

   L'éditeur de script comprend trois parties :
   - En haut à gauche les catégories de scripts, il s'agit simplement de les y ranger : `examples`, `private` et `scripts`. C'est dans cette dernière catégorie que l'utilisateur doit placer ses scripts.
   - En haut à droite les scripts de la catégorie sélectionnée sont présentés à l'utilisateur. En cliquant sur un de ceux-ci, son code source est affiché dans l'éditeur de texte en bas.
   - En bas sur toute la largeur, la zone texte avec le code source du script sélectionné. C'est ici que nous créons ou modifions les scripts. Pour valider une modification, il suffit de presser les touches Ctrl-s.

   (b) Dans l'éditeur de script, cliquer sur la catégorie `scripts` jusqu'à ce que la zone texte en bas contienne le modèle de script tel qu'affiché dans la figure 1.1.

   (c) Saisir le code source ci-dessous et comme dans la figure 1.2 :

3. Les macro-constructions ont une approche géométrique tandis que les scripts ont une approche numérique mais aussi et surtout nous pouvons les utiliser dans un esprit de bidouillage ("hacking" en anglais).
4. En particulier, les fonctions mathématiques.

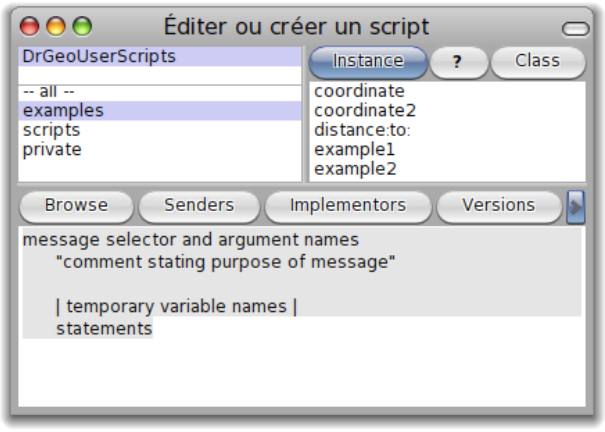

FIGURE 1.1 – L'éditeur de script

```
MonPremierScript
"Le commentaire de mon premier script"
   ^ 'hello !'.
```

Sauvegarder le script par la combinaison de touches CTRL-S, DR. GEO vous demandera vos nom et prénom, vous pouvez entrer vos initiales, c'est sans grande importance pour le moment. La premier ligne du script, `MonPremierScript` désigne le nom du script, la suite est le code source du script. La première ligne de celui-ci, entre guillemets, est un commentaire de ce que fait le script, nous conseillons de bien le documenter car c'est utile pour la suite.

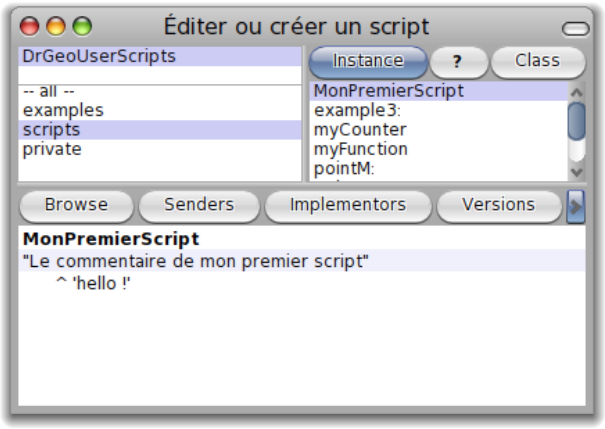

FIGURE 1.2 – Mon premier script

L'éditeur de script peut maintenant être fermé.

2. **Utilisation du script dans la figure**
   Choisir `Numériques>Utiliser un script` dans le menu. Dans la boîte de dialogue qui s'affiche alors, choisir le script `MonPremierScript` que nous avons créé précédemment. Noter qu'à chaque fois qu'un script est choisi, son commentaire descriptif est affiché en bas de la boîte de dialogue.
   Une fois le script sélectionné, cliquer dans la figure à l'emplacement souhaité, le script y sera inséré ; dans cet exemple le script ne fait que retourner le message « hello ! ».
   La valeur retournée par un script est celle affichée dans la figure.

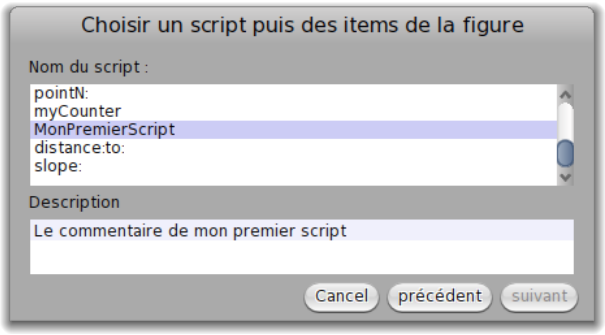

FIGURE 1.3 – Choisir un script

Dans les exemples qui suivent, nous donnons simplement le code source des scripts Smalltalk, il suffira de reprendre les étapes expliquées précédemment pour créer le script correspondant et l'utiliser dans une figure, nous ne revenons pas sur ces étapes.

Un générateur de nombres aléatoires et autres : Si vous souhaitez un générateur de nombres entiers aléatoires entre 0 et 10, rien de plus simple, c'est ce que fait le script suivant :

```
aléatoire
"Je retourne un nombre continuellement modifié aléatoirement"
   ^ 10 atRandom.
```

À chaque mise à jour de la figure, il génère un nombre aléatoire entier dans l'intervalle [0 ; 10].

Si vous préférez un nombre flottant dans l'intervalle [0 ; 1], utilisez ce script :

```
aléatoire2
"Je retourne un nombre décimal aléatoire entre 0 et 1"
   ^ 100 atRandom / 100.
```

(!) Quelques précisions :
- La valeur retournée par le script est le résultat de l'expression après le symbole ^ ;
- La valeur retournée peut être de n'importe quel type [5], DR. GEO en affiche une représentation sous forme de chaîne de caractères ;
- Si l'on souhaite retourner la valeur d'une variable, il suffit de mettre son nom après le symbole ^ .

Calculer des valeurs usuelles : Pour calculer une valeur approchée de π :

```
pi
   ^ Float pi
```

ou de e :

```
e
   ^ Float e
```

Les valeurs retournées par ces scripts sont ensuite utilisables comme toutes les autres valeurs numériques que peut générer DR. GEO. Pour toutes ces petites choses les scripts sont donc vos amis. Mais ils peuvent faire bien plus de choses intéressantes lorsqu'ils reçoivent des paramètres en entrée.

En effet ici les scripts n'avaient aucun argument, il n'y avait donc pas lieu de sélectionner des items de la figure lors de l'insertion des scripts dans la construction. Bien sûr leur intérêt réside dans les traitements numériques qu'ils permettent sur des items et la restitution de ce résultat dans la figure, sous la forme d'un objet qui lui même peut-être utilisé par d'autres scripts. Dans les sections suivantes nous montrerons de tels enchaînements de scripts.

1.1.2 Script avec un paramètre d'entrée

La procédure pour créer un script avec un paramètre d'entrée est sensiblement la même :

1. Lors de l'édition du script (`Numériques>Éditer un script`), l'argument s'intercale dans le nom du script.
   Par exemple pour calculer la distance à l'origine d'un point, nous écrivons le script suivant :

   ```
   distanceToOrigin: item
   "Retourne la distance à l'origine d'un point"
      ^ item point dist: 0@0
   ```

   Quelques explications :
   - `item` est l'argument de notre script, cela doit être un point. C'est en fait une instance de la classe `DrGPointItem` [6].
   - `DrGPointItem` a une méthode d'instance `#oint` qui retourne ses coordonnées. Ainsi de l'argument nous pouvons extraire des informations, ici ses coordonnées et faire un calcul à partir de celles-ci.
   - `0@0` est une instance de la classe `Point` de coordonnées (0,0).
   - `#dist:` est un message à mot clé [7] de la classe `Point` qui attend comme unique argument une instance d'un autre point, elle calcule la distance entre ces deux instances. Elle peut se comprendre comme : "distance entre item point et (0,0)". Les messages à mot clé sont une spécificité de Smalltalk : les arguments sont intercalés dans le nom du message.
2. Lors de l'utilisation du script (`Numériques>Utiliser un script`), Dr. Geo attend que l'utilisateur clique sur un point, puis sur un emplacement de la figure.
   Attention, si un autre objet qu'un point est choisi, Dr. Geo génère une erreur, il suffit de fermer la fenêtre du débogueur et de sélectionner à nouveau le script pour recommencer.

Selon le type d'objet en référence, diverses méthodes sont disponibles ; qui pour obtenir sa valeur, qui pour obtenir ses coordonnées, etc. Le répertoire des méthodes est disponible depuis la section Méthodes de référence des scripts 1.2, p. 8.

1.1.3 Script avec deux paramètres d'entrée :

Supposons que nous souhaitions calculer la distance entre deux points, nous créons alors un script avec deux paramètres d'entrée, ils sont intercalés dans le nom du script. Celui-ci peut s'appeler `distance:to:` ; chaque " : " designe l'emplacement d'un paramètre. Ainsi dans l'éditeur de script nous écrivons le code source suivant :

```
distance: item1 to: item2
"Calcule la distance entre deux points"
   ^ item1 point dist: item2 point
```

6. Pour découvrir le protocole de cette classe, écrire son nom n'importe où dans l'environnement Dr. Geo, le sélectionner à la souris puis presser les touches Ctrl-b, un navigateur de classes s'affiche alors sur cette classe, il permet de naviguer et d'étudier son code source.
7. Comme le script ici son nom se termine par " :"

`item1` et `item2` sont les deux noms de nos paramètres, nous sommes libres de leur nommage. Pour utiliser ce script, procéder comment dans les exemples précédents : choisir deux points de la construction et un emplacement de la figure pour y placer le résultat du script.

1.1.4 Example détaillé de figure avec plusieurs scripts

Dans la section suivante, nous présentons une figure plus complexe intégrant un enchaînement de scripts pour la construction d'une portion de courbe représentative d'une fonction et la tangente en un point mobile de cette portion de courbe.

La figure finale est disponible dans le dossier `examples` de DR. GEO, elle s'appelle `Curve and slope.fgeo`.

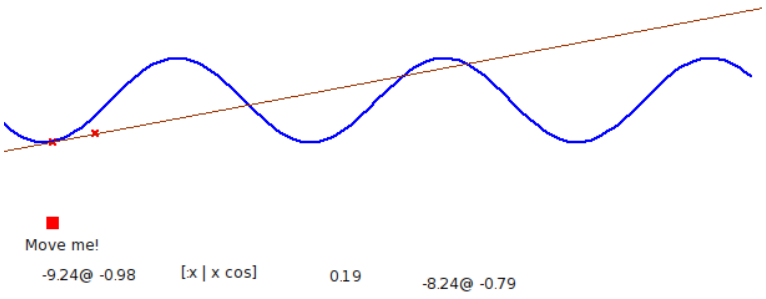

FIGURE 1.4 – Courbe et tangente en un point

Dans une nouvelle figure, nous commençons par construire un segment horizontal, nous y plaçons un point libre appelé "Move me!". Ce point servira de base à la construction de la courbe comme lieu d'un point.

Définir un fonction : Comme un script est capable de retourner n'importe quel type d'objet, le premier de notre construction définira simplement la fonction utilisée. Pour ce faire nous utilisons des objets Smalltalk de type bloc de code – fonction anonyme en Lisp. Nous nommons ce script `myFunction`, sans paramètre :

```
myFunction
"La définition de notre fonction"
   ^ [:x | x cos]
```

Ensuite nous le plaçons dans la figure [8]. Ainsi le bloc de code retourné par `myFunction` attend un argument `:x` et retourne le cosinus de celui-ci. Nous verrons dans la suite comment manipuler ce script.

Image d'une valeur par une fonction : Maintenant nous calculons les coordonnées d'un point appartenant à la courbe. Nous utilisons notre point "Move me!" et notre fonction. Le script aura comme unique argument le point ; comme nous allons le voir nous n'avons pas besoin du script `myFunction` en argument :

8. Il est important de le référencer dans la figure afin qu'il soit inclus dans la description de celle-ci lors d'une opération de sauvegarde sur fichier.

```
pointM: item
"Given a point calculate the y value from myFunction and the x valule of the point"
   ^ item point x @(self myFunction value: item point x)
```

Le point retourné par ce script a la même abscisse que le point de départ, son ordonnée est l'image par la fonction.

Noter :

- l'accès direct au script de la fonction : `self myFunction` ;
- le passage de l'argument à la fonction doit se comprendre comme `myFunction(item point x)`.

Maintenant utilisons ce script `pointM` avec comme argument le point "Move me !" ; le résultat du script est de la forme `1.2@0.5`, cela représente un couple de coordonnées.

Avec l'outil point  , créons un point ayant ses coordonnées contraintes par le résultat de ce script.

La fonction lieu d'un point  donne la courbe en sélectionnant nos deux points.

Pente en un point d'une fonction : Pour ce faire, nous calculons une valeur approchée de la pente :

$$p = \frac{f(x + 0.001) - f(x)}{0.001}$$

Cela se traduit par un script `slope:` avec comment argument le point où calculer une approximation de la pente :

```
slope: item
"Compute the slope at the given x point position for myFunction"
| f  x |
   f := self myFunction.
   x := item point x.
   ^ (f value: x + 0.001) - (f value: x) / 0.001
```

Nous plaçons ensuite ce script dans la figure.

Noter :

- La déclaration de variables temporaires `| f x |`. Comme déjà expliqué les variables ne sont pas typées, pas de soucis de ce côté là.
- La référence du bloc de code dans une variable `f := self myFunction`. Le symbole pour assigner une valeur à une variable est "`:=`".
- Les parenthèses ! Smalltalk ne connait pas la priorité des opérateurs, en fait ils n'existent pas dans ce langage. Le lecteur est invité à étudier la section sur les messages Smalltalk du livre *Pharo By Exemple*.

Calculer et afficher la tangente à une courbe : Pour construire notre tangente nous avons besoin de deux points : un sur la courbe – nous l'avons déjà – et un deuxième. Pour ce dernier nous utiliserons le calcul précédent de notre pente. Écrivons alors un script retournant les coordonnées d'un point sur la tangente :

```
pointN: item
"Given an initial point, calculate the coordinates of a point on the tangente"
| p x |
   p := self slope: item.
   x := item point x + 1.
   ^ x @ (item point y + p)
```

En utilisant le protocole de la classe `Point`, ce script s'écrit aussi en une ligne :

```
pointN: item
"Given an initial point, calculate the coordinates of a point on the tangente"
   ^ item point + (1@ (self slope: item))
```

Utilisons ce script avec comme argument le point de notre courbe. Nous obtenons un deuxième couple de coordonnées. Avec celles-ci construisons un point, la tangente est la droite définie par ce point et celui de la courbe.

En déplaçant le point "Move me !", la tangente est recalculée. Tout aussi intéressant : modifier le script `myFunction` actualise correctement l'ensemble de notre construction. Quelques exemples de modifications :

```
^ [:x | x *x / 10]

^ [:x | x cos + (10 * x) sin]

^ [:x | (x *5) cos + x abs]
```

1.2 Méthodes de référence pour les scripts Dr. Geo

Un argument passé à un script est toujours une référence vers une instance de classe de la hiérarchie `DrGMathItem`, elle est le modèle de base pour représenter tout objet d'une figure. Ainsi pour connaître les messages compris par un argument d'un script, il convient d'examiner la hiérarchie de `DrGMathItem`. Celle-ci comprend plus de 80 classes, mais du fait des héritages, seules quelques unes sont intéressantes pour les usages courants des scripts :

- `DrGMathItem`
- `DrGpointItem`
- `DrGDirectionItem` concerne segment, droite, demi-droite, vecteur
- `DrGArcItem`
- `DrGCircleItem`
- `DrGLocus2ptsItem`
- `DrGPolygonNptsItem`
- `DrGValueItem`

Pour explorer ces classes, ouvrir un espace de travail – Ctrl-k après un clic sur le fond de l'environnement – y saisir et sélectionner le nom de la classe puis ouvrir le navigateur de classe par Ctrl-b.

Les sections suivantes contiennent la description de quelques messages pouvant être utiles. Elles sont présentées par classe.

1.2.1 Item Math

C'est le protocole de la classe `DrGMathItem`, il concerne tout argument, c'est à dire que les messages exposés ci-dessous s'appliquent à tous les types d'objets passés en argument à un script.

`<string> item safeName`

`item` : item mathématique

⟶ une chaîne de caractères représentant le nom de l'item

Exemple :

```
nom := point1 safeName.
^ nom asUppercase.
```

`<boolean> item exist`

`item` : item mathématique

⟶ un booléen indiquant si l'item est dans un état permettant son existence
Exemple :
`line exist ifTrue: [ position := line origin ].`

`<collection> item parents`
`item` : item mathématique
⟶ une collection des items parents de item
Exemple :
`point1 := segment parents first.`

`item move: vector`
Déplace item dans une direction donnée, tout en tenant compte de ses contraintes.
`item` : item mathématique
`vector` : un vecteur (x,y) représentant le déplacement
Exemple :
`circle move: 2@1.`

`<point> item closestPointTo: aPoint`
`item` : item mathématique
`aPoint` : un couple de coordonnées
⟶ un couple de coordonnées du point de "item" le plus proche de "aPoint".
Exemple :
`position := item closestPointTo: 2@1.`

1.2.2 Point

`<point> item point`
`item` : référence d'un point
⟶ les coordonnées de ce point
Exemple :
`abscissa := pointItem point x.`

`item point: aPoint`
`item` : référence d'un point
`aPoint` : couple de coordonnées
Action : modifie les coordonnées de "item"
Exemple :
`pointItem point: 5@2.`

`<float> item abscissa`
`item` : référence d'un point libre sur une ligne
⟶ abscisse curviligne de ce point, elle appartient à l'intervalle [0 ; 1]
Exemple :
`a := pointItem abscissa.`

`item abscissa: a`
`item` : référence d'un point libre sur une ligne

`a` : nombre décimal compris dans [0 ; 1]
Action : modifie l'abscisse curviligne d'un point libre sur une ligne
Exemple :
```
pointItem abscissa: 0.5.
```

`point moveAt: aPoint`
`point` : un item représentant un point géométrique
`aPoint` : un couple de coordonnées (x,y) où déplacer "point"
Action : déplace "point" à la position donnée
Exemple :
```
point moveAt: 2@1.
```

1.2.3 Ligne droite ou courbe

`<float> curve abscissaOf: aPoint`
`curve` : une ligne droite ou non
`aPoint` : un point (x,y)
⟶ un nombre dans [0 ; 1] abscisse curviligne de "aPoint" sur "curve"
Exemple :
```
a := curve abscissaOf: 2@1.
```

`<point> curve pointAt: anAbscissa`
`curve` : une ligne droite ou non
`anAbscissa` : un nombre dans [0 ; 1]
⟶ un point sur "curve" d'abscisse curviligne "anAbscissa"
Exemple :
```
myPoint := curve pointAt: 0.5.
```

`<boolean> curve contains: aPoint`
`curve` : une ligne droite ou non
`aPoint` : un point (x, y)
⟶ un booléen indiquant si "aPoint" est sur "curve"
Exemple :
```
(curve contains: 0@1) ifTrue: [^ 'Yes!'].
```

1.2.4 Droite, Demi-droite, Segment, Vecteur

`<point> item origin`
`item` : référence vers un objet de type droite, demi-droite, segment ou vecteur
⟶ un point origine de cet item
Exemple :
```
item origin.
```

`<vector> item direction`
`item` : référence vers un objet de type droite, demi-droite, segment ou vecteur
⟶ un vecteur (x, y) indiquant la direction
Exemple :

```
v := item direction.
slope := v y / v x.
```

`<vector> item normal`
`item` : référence vers un objet de type droite, demi-droite, segment ou vecteur
⟶ un vecteur unitaire normal à la direction de “item”
Exemple :

```
n := item normal.
```

1.2.5 Segment

`<float> item length`
`item` : référence vers un segment
⟶ longueur du segment
Exemple :

```
segment := canvas segment: 0@0 to: 5@5.
l := segment length
```

`<point> item extremity1`
`item` : référence vers un segment
⟶ coordonnées de l’extrémité 1
Exemple :

```
segment := canvas segment: 0@0 to: 5@5.
p := segment extremity1.
```

`<point> item extremity2`
`item` : référence vers un segment
⟶ coordonnées de l’extrémité 2
Exemple :

```
segment := canvas segment: 0@0 to: 5@5.
p := segment extremity2
```

`<point> item middle`
`item` : référence vers un segment
⟶ coordonnées du milieu du segment
Exemple :

```
segment := canvas segment: 0@0 to: 5@5.
m := segment middle
```

1.2.6 Cercle, Arc de cercle, polygone

`<point> item center`
`item` : référence vers un cercle ou un arc de cercle
⟶ point contenant les coordonnées du centre du cercle ou de l’arc de cercle
Exemple :

```
c := item center.
```

`<float> item radius`
`item` : référence vers un cercle ou un arc de cercle
⟶ rayon du cercle ou de l'arc de cercle
Exemple :
```
r := item radius
```

`<float> item length`
`item` : référence vers un cercle, un arc de cercle ou un polygone
⟶ longueur du cercle, de l'arc de cercle ou du polygone
Exemple :
```
l := arc length
```

1.2.7 Valeur

`<float> item valueItem`
`item` : référence vers un item de type valeur
⟶ valeur de cet item
Exemple :
```
n1 := item2 valueItem.
n2 := item2 valueItem.
n1 + n2.
```

`item valueItem: v`
`item` : référence vers un item de type valeur libre
`v` : valeur décimale
Action : modifie la valeur d'un item de type valeur libre
Exemple :
```
item valueItem: 5.2.
```

`item position: aPoint`
`item` : référence vers un item de type valeur
`aPoint` : point (x, y)
Action : déplace à l'écran la position d'un item valeur
Exemple :
```
item position: 0.5@2.
```

1.2.8 Angle

`<integer> angle degreeAngle`
`angle` : référence vers un angle, orienté ou géométrique
⟶ une mesure de cet angle en degrés
Exemple :
```
angle1 := a1 degreeAngle.
```

`<float> angle radianAngle`
`angle` : référence vers un angle, orienté ou géométrique
⟶ une mesure de cet angle en radian

Exemple :

```
angle1 := a1 radianAngle.
```