Chapter 1

Dr. Geo Smalltalk script

Dr. Geo is a dynamic application written in Pharo Smalltalk. It means it is possible to modify it from itself as it is running. We have exploited this feature to define in Dr. Geo sketch items which are in fact Smalltalk scripts – code snippets – to extend dynamically, at the infinite, the possibility of Dr. Geo. But what is exactly Smalltalk?

> *Smalltalk is an object-oriented, dynamically typed, reflective programming language. Smalltalk was created as the language to underpin the "new world" of computing exemplified by "human–computer symbiosis." It was designed and created in part for educational use, more so for constructionist learning, at the Learning Research Group (LRG) of Xerox PARC by Alan Kay, Dan Ingalls, Adele Goldberg, Ted Kaehler, Scott Wallace, and others during the 1970s. The language was first generally released as Smalltalk-80. Smalltalk-like languages are in continuing active development, and have gathered loyal communities of users around them. ANSI Smalltalk was ratified in 1998 and represents the standard version of Smalltalk. .*[1]

This abstract from the book preface *Pharo By Example*[2] describes exactly the Smalltalk environment used by Dr. Geo:

> *Pharo is a modern open-source development environment for the classic Smalltalk-80 programming language.*
>
> *Pharo strives to offer a lean, open-source platform for professional software development, and a robust and stable platform for research and development into dynamic languages and environments.*

Dr. Geo uses the Smalltalk environment to propose a comfortable set to write scripts and to gain access to the geometric items programming interfaces. These last ones are the set of methods part of the items' definition.

Thus the user can write scripts to manipulate the sketch items; and as these scripts are also sketch items, it does not need to be in a separate file, it is saved in the sketch file.

Smalltalk script is based on the Pharo Smalltalk environment, therefore the user benefits the great developer tools of the environment: class browser, inspector, debugger, etc. The user wishing to explore the power of the script is invited to study the book *Pharo By Example*, he will learn the Smalltalk language and its environment.

[1] Wikipedia Smalltalk article (`http://en.wikipedia.org/wiki/Smalltalk`). Page view the 24 Jully 2013. Contents under the license CC-BY-SA 3.0 (`http://creativecommons.org/licenses/by-sa/3.0/deed.fr`).

[2] `http://pharobyexample.org`

1.1 Script by the example

There are two phases to use script:

- writing the script itself,
- inserting the script in a sketch, a same script can be used several times, with different parameters.

The tool to create or edit a script is on the menu `Numeric>Edit a script` , it is also reachable from the toolbar.

The tool to use a script is on the menu `Numeric>Use a script` , it is also reachable from the toolbar.

A script can receive 0 to n arguments (input parameters). After selecting a script to insert in the sketch, the user clicks on the items used as arguments then a final click somewhere in the background to pin the returned value.

In the following section, we present a few script examples, its function and power will be more easily understood. The script and the macro-construction give a special dimension to Dr. Geo – with a different positioning[3] – to let the user goes where the software was not initially planed for.

It is important to understand most Pharo Smalltlak resources are available from the scripts. It is particularly true for the class and method libraries[4], and we will use it a lot.

1.1.1 Script without parameter

First example: The procedure to create a script without input parameter is as follow:

1. **Edit the script**
   (a) Select `Numeric>Edit a script` in the menu to open the script editor:

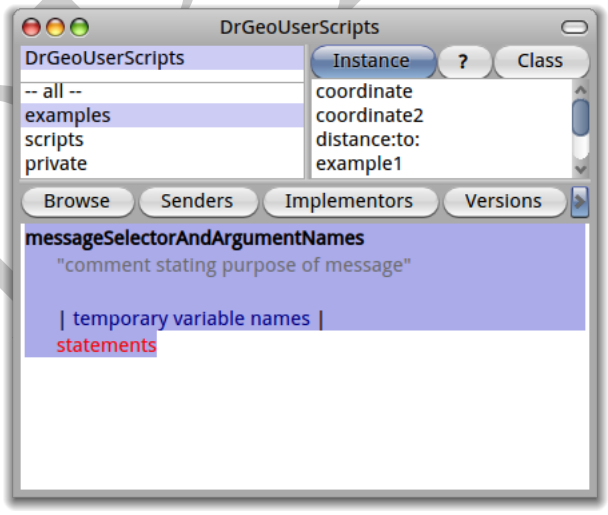

Figure 1.1: Script editor

Within the script editor there are three parts:

- In the top left, the script categories to keep in good order: `examples`, `private` and `scripts`. It is in this last one you put your scripts.

[3] A macro-construction is geometry oriented whereas a script is numerically/computation oriented and it brings in Dr. Geo the hacking spirit.

[4] For example, mathematical functions.

- In the top right, the scripts name of the selected category. When selecting one there, its source code is printed in the editor at the bottom.
- In the bottom, the source code editor for the selected script. It is there you create or modify the scripts. To validate a modification, press the keyboard keys Ctrl-s.

(b) In the script editor, select the category `scripts` until the text editor in the bottom contains a script source code model as the one shown in figure 1.1.

(c) Input the source code as bellow and as shown in figure 1.2 :

```
myFirstScript
"The comment of my first script"
   ^ 'hello !'.
```

Save the script with the keys shortcut Ctrl-s. Dr. Geo will ask your last name and first name, it is just to track the history of the script modifications. The first script line, `myFirstScript` is the name of the script, next follow the source code itself. Its first line is an optional comment, between quotation marks, it explains the script and what are the expected parameters, we strongly encourage you to document carefully because it is useful when using the script.

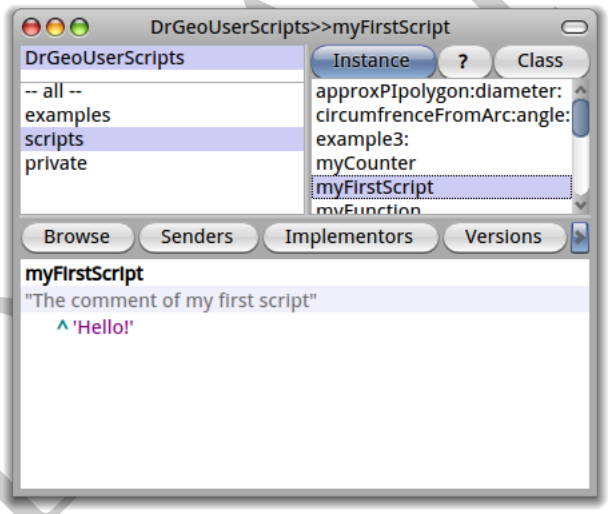

Figure 1.2: My first script

The script editor can now be closed.

2. **Using the script in the sketch**
   Select `Numeric>Use a script` in the menu. In the displayed dialogue box, choose the script `myFirstScript`. Note: each time a script is selected, its descriptive comment is printed in the bottom of the dialogue.

   Once the script selected, click on the sketch at the place to pin it. In this example the script only returns the message " hello ! ". The returned value is always printed in the sketch, if necessary it is converted as a text representation.

In the following examples, we only give the source code of the Smalltalk scripts. To use them in Dr. Geo, just follow the steps explained previously.

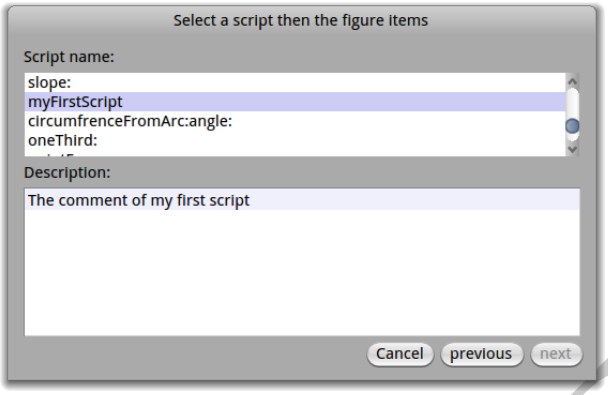

Figure 1.3: Select a script

A random number generator and more: If you need a random generator for numbers between 0 and 10, very simple, the following script exactly does that:

```
random
"I return a random number continuously updated"
   ^ 10 atRandom.
```

At each sketch update, it generates a different random number in the interval [0 ; 10]. In case you prefer a random decimal number in the interval [0 ; 1], use this script:

```
random2
"I return a random decimal number continuously updated in 0 and 1"
   ^ 100 atRandom / 100.
```

(!) Some precisions:

- the returned value by the script is the result of the expression after the ^ ,
- the returned value can be of any type[5], Dr. Geo prints its text representation (a string),
- if you want to return the value of a variable, it is enough to put its name after the symbol ^ .

Calculate usual values To calculate an approximate value of π:

```
pi
   ^ Float pi
```

or e :

```
e
   ^ Float e
```

These returned values are next usable as any other value generated by Dr. Geo. For all these small things, the script is your friend. But it can do much more interesting things when it received input parameters.

Indeed, so far the scripts have no argument, it was not necessary to select items in the sketch when inserting the script in the construction. Of course the script interest relies on the numerical computation and its return as a value pined in the sketch for use with other constructions or scripts. In the following sections we show use of scripts in cascade.

1.1.2 Script with one input parameter

The procedure to create a script with one input parameter is mostly the same:

1. When editing the script (`Numeric>Edit a script`), the argument is inserted in the script name.
   For example to calculate the distance from the origin to a point, we write the following script:

   ```
   distanceToOrigin: item
   "Return the distance from origin to a point"
      ^ item point dist: 0@0
   ```

   A few explanations:

   - `item` is our script argument, it must be a point. It is in fact an instance of the class `DrGPointItem`[6]
   - `DrGPointItem` has an instance method `#point` returning its coordinates. Thus from the argument we can extract information, here its coordinates, and do calculus with it.
   - `0@0` is an instance of the class `Point` with coordinates (0,0).
   - `#dist:` is a keyword message [7] from the class `Point` expecting an unique argument another instance of `Point`. It calculates the distance between these two instances. It can be understand as: "distance between item point and (0,0)". The keyword message is very specific to Smalltalk: the argument are inserted in the message name.

2. To use a script (`Numeric>Use a script`), Dr. Geo expects from the user to click on a point then somewhere in the sketch.
   Attention, if an item different from a point is selected, Dr. Geo throws an error, the debugger window can be safely closed. To continue, select again the script.

Depending on the type of argument received by the script, various methods are available: to get its value, its coordinate, etc. A list of the methods you can use are presented in section 1.2, p. 8.

1.1.3 Script with two input parameters:

Let's say we want to calculate the distance between two points. To do so we create a script with two input parameters, they are inserted in the name of the script. We name it `distance:to:`; each ":" indicates the place of the parameter. Therefore in the script editor we write the following source code:

```
distance: item1 to: item2
"Calculate the distance between two points"
   ^ item1 point dist: item2 point
```

`item1` et `item2` are the name of the two parameters, nous sommes libres de leur nommage. Pour utiliser ce script, proceed like in the previous example: select two points in the construction and click somewhere in the sketch to pin the result of the script.

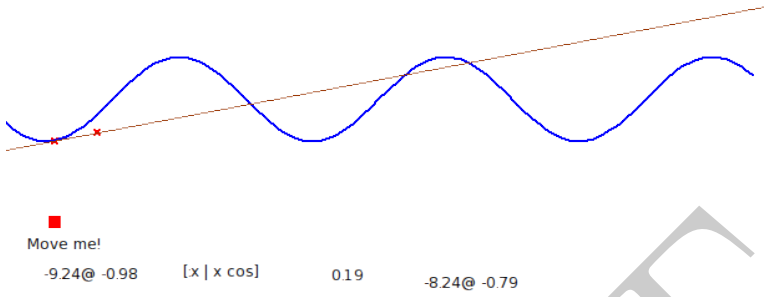

Figure 1.4: Curve and tangent to a point

1.1.4 Detailed example with several scripts

In the following section, we present a more complex sketch, integrating a cascading use of scripts to construct the curve of a function and its tangent to a mobile point of the curve.

The final sketch is on the DR. GEO folder `examples`, it is named `Curve and slope.fgeo`.

In a new empty sketch, we construct a horizontal segment, we add to this segment a free point named "Move me!". This point will be the base to construct the curve as a locus.

Define a function: As a script can return any type of object, our first construction simply defines the used function. To do so, we use a Smalltalk object named bloc of code – anonymous function in Scheme. We name this script `myFunction`, without parameter:

```
myFunction
"Definition of our function"
   ^ [:x | x cos]
```

Next we insert it in the sketch[8] So the bloc of code is returned by `myFunction` expect one argument `:x` and it returns its cosine. We see later how to manipulate this script.

Image of a value by function: Now we calculate the coordinates of a point on the curve. We use our point "Move me!" abscissa and our function previously defined. Our script will have one unique point argument; as we will see we do not need to pass `myFunction` as argument:

```
pointM: item
"Return a point on the curve of myFunction and driven by item point"
   ^ item point x @(self myFunction value: item point x)
```

The point returned by this script has the same abscissa of the argument, its ordinate is its abscissa image by the function.

Note:

[6] To learn about the protocol of this class, write its name in a workspace, select it with the mouse then press the keys Ctrl-B, a class browser opens on this class to navigate in its protocol and source code.

[7] As the script here because its name terminate with ":"

[8] It is important to add it to the construction to get it included in the file when the sketch is saved.

- the direct access to the function: `self myFunction`,
- the way to pass an argument to a bloc of code, it can be understood as `myFunction(item point x)`.

Now we use this script `pointM:` with the point "Move me!" as argument; the result of this script is of the form `1.2@0.5`, it represents a couple of coordinates.

With the tool  , `Points>Coordinates`, we create a point with its coordinate specified by the result of this script.

The tool locus  , `Lines>Locus`, gives the curve after selecting our two points.

Slope at a point of the function's curve: To do so, we calculate an approximation of the slope:

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

It is directly translated in the script `slope:` with unique argument the point where to approximate the slope:

```
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
```

Next we insert this script in the construction.
Note:

- The declaration of temporary variables `| f x |`. As explained variables are not typed, so no trouble here.
- A reference of a bloc of code in a variable `f := self myFunction`. The symbol to assign a value to a variable is "`:=`".
- The parenthesis! Smalltalk does not know operator priority, in fact operators does not exist in this language. The reader is encouraged to study the section about Smalltalk messages in the book *Pharo By Exemple*.

Calculate and display the tangent to a curve: To construct our tangent we need two points: one on the curve – we have it already – and a second one on the line. For this last one we use the previously calculated slope. Let's write a second script computing this second point coordinates:

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

With the protocol of the class `Point` – message `+` sent to a point – this script can be written in one line:

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

We use this script with argument the point on the curve. We get as result the coordinates of the second point on the tangent. With these ones we construct the second point[9], the tangent is the line defined by this point and the point on the curve.

When moving the point "Move me!", the tangent is recomputed. Equally interesting: modifying the script `myFunction` update the whole construction accordingly. A few modification examples to this script:

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

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

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

1.2 Reference methods for Dr. Geo scripts

An argument passed to a script is always a reference to an instance of a class from the hierarchy `DrGMathItem`, it the base class of all items used in a construction. So to know about the messages you can send to an argument passed to a script, it is wise to examine the hierarchy of `DrGMathItem`. It contains more than 80 classes, but because of the inheritance, only few classes are interesting to explore:

- `DrGMathItem`
- `DrGpointItem`
- `DrGDirectionItem` for segment, line, ray, vector
- `DrGArcItem`
- `DrGCircleItem`
- `DrGLocus2ptsItem`
- `DrGPolygonNptsItem`
- `DrGValueItem`

To explore these classes, open a workspace – Ctrl-k after a click in the background of the environment – input and mouse select the class name to explore then press the shortcut Ctrl-b to open a class browser on it.

The following sections contain the description of some useful messages. It is ordered by class.

1.2.1 Math Item

It is the protocol of the class `DrGMathItem`, it concerns all type of argument passed to a script.

`<string> item safeName`

`item` : mathematics item
⟶ text representing the item name
Example :

[9] Tool `Points>Coordinates`

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

`<boolean> item exist`
`item` : mathematics item
⟶ boolean indicating the item exist or not given the whole state of the sketch
Example :
`line exist ifTrue: [ position := line origin ].`

`<collection> item parents`
`item` : mathematics item
⟶ collection of the parents of this item
Example :
`point1 := segment parents first.`

`item move: vector`
Move an item in a given direction, while respecting its constraints.
`item` : mathematics item
`vector` : a vector (x,y) representing the displacement
Example :
`circle move: 2@1.`

`<point> curve closestPointTo: aPoint`
`curve` : mathematics curve
`aPoint` : coordinates
⟶ coordinates of the point on "curve" the closest to "aPoint".
Example :
`position := segment closestPointTo: 2@1.`
Example :
`position := arc closestPointTo: 2@1.`

1.2.2 Point

A point item – point object defined in a Dr. Geo construction – passed as argument to a script is a very complex object. It can be a free point on the plane, on a line, an intersection, etc. A few method can be usefully used in script.

`<point> item point`
`item` : point item
⟶ coordinates of this item
Example :
`abscissa := pointItem point x.`

`item point: aPoint`
`item` : point item
`aPoint` : coordinates
Action : modify the coordinate of "item"
Example :
`pointItem point: 5@2.`

`<float> item abscissa`
`item` : free point on a line
⟶ curvilinear abscissa of this point, in the interval [0 ; 1]
Example :
```
a := pointItem abscissa.
```

`item abscissa: a`
`item` : free point on a line
`a` : decimal number in the interval [0 ; 1]
Action : modify the curvilinear abscissa of a free point on a line
Example :
```
pointItem abscissa: 0.5.
```

`item moveAt: aPoint`
`item` : geometric point
`aPoint` : coordinates where to displace "item"
Action : displace "item" at the given position
Example :
```
point moveAt: 2@1.
```

1.2.3 Straight or curved line

`<float> curve abscissaOf: aPoint`
`curve` : straight or curved line
`aPoint` : coordinates (x,y) of a point
⟶ number in [0 ; 1] curvilinear abscissa of "aPoint" on "curve"
Example :
```
a := curve abscissaOf: 2@1.
```

`<point> curve pointAt: anAbscissa`
`curve` : straight or curved line
`anAbscissa` : number in [0 ; 1]
⟶ coordinates of a point on "curve" with curvilinear abscissa equal to "anAbscissa"
Example :
```
myPoint := curve pointAt: 0.5.
```

`<boolean> curve contains: aPoint`
`curve` : straight or curved line
`aPoint` : coordinates (x, y) of a point
⟶ boolean indicating if the "aPoint" is on "curve"
Example :
```
(curve contains: 0@1) ifTrue: [^ 'Yes!'].
```

1.2.4 Line, ray, segment, vector

`<point> item origin`
`item` : line, ray, segment or vector

⟶ coordinates of the origin of this item
Example :
```
item origin.
```

`<vector> item direction`
`item` : line, ray, segment or vector
⟶ vector (x, y) indicating the item direction
Example :
```
v := item direction.
slope := v y / v x.
```

`<vector> item normal`
`item` : line, ray, segment or vector
⟶ unit vector normal to the item direction
Example :
```
n := item normal.
```

1.2.5 Segment

`<float> item length`
`item` : segment
⟶ length of the segment
Example :
```
segment := canvas segment:  0@0 to:  5@5.
l := segment length
```

`<point> item extremity1`
`item` : segment
⟶ coordinates of the extremity 1
Example :
```
segment := canvas segment:  0@0 to:  5@5.
p := segment extremity1.
```

`<point> item extremity2`
`item` : segment
⟶ coordinates of the extremity 2
Example :
```
segment := canvas segment:  0@0 to:  5@5.
p := segment extremity2
```

`<point> item middle`
`item` : segment
⟶ coordinates of the middle of the segment
Example :
```
segment := canvas segment:  0@0 to:  5@5.
m := segment middle
```

1.2.6 Circle, arc, polygon

`<point> item center`
`item` : circle or arc
⟶ coordinates of the centre of the circle or arc
Example :
```
c := item center.
```

`<float> item radius`
`item` : circle or arc
⟶ radius of the circle or arc
Example :
```
r := item radius
```

`<float> item length`
`item` : circle, arc or polygon
⟶ length of the circle, arc or polygon
Example :
```
l := arc length
```

1.2.7 Value

`<float> item valueItem`
`item` : value
⟶ value of this item
Example :
```
n1 := item2 valueItem.
n2 := item2 valueItem.
n1 + n2.
```

`item valueItem: v`
`item` : free value item
`v` : decimal number
Action : modify the value of a free value item
Example :
```
item valueItem: 5.2.
```

`item position: aPoint`
`item` : value
`aPoint` : coordinates (x, y) of a point
Action : displace at the screen the position of a value item
Example :
```
item position: 0.5@2.
```

1.2.8 Angle

`<integer> angle degreeAngle`
`angle` : angle, oriented or geometric

⟶ measure of this angle item in degree
Example :
```
angle1 := a1 degreeAngle.
```

`<float> angle radianAngle`
`angle` : angle, oriented or geometric
⟶ measure of this angle item in radian
Example :
```
angle1 := a1 radianAngle.
```