Do we need Wolframese for education?

Some people advocate the use of Wolframese in education and one needs little experience to see why. Based on Lisp−like Mathematica 9 Wolframese has a succinct notation, supports virtually every programming style, has fascinating facilities for programming over the Web, is applicable for a wide range of problem domains, is well implemented and offers much more at a reasonable price.

Wolframese supports 'computational thinking' and includes a wealth of curated data on many fields, some far outside of Mathematics. An excellent introduction is in Wolfram's An elementary introduction to the Wolfram language on http://www.wolfram.com/language/elementary. It can be tried out for free using a Programming Lab: www.wolfram.com/programming−lab. So one might wonder: why not use this in education to teach computation, design and what not?

This question is harder to answer than it seems. Even experienced programmers must work long hours to learn Wolframese well, need to read a few thousand pages in the manuals and will need to talk about subtleties of their own programs before they can teach programming Wolframese. Full documentation as is now available approaches 50.000 pages of text and a typical installation needs several Gigabytes of diskspace. Therefor full mastery seems out of the question even for many computer scientists. The average high school student will be hard put to learn it by him− or by herself, and so will their teachers. To introduce this type of thing in highschool requires answers to questions like how many pupils will be glad with it in less than forty hours of work, how much service they will need and how much time their teachers need to, say, master *some* details.

Programming languages can not tell us if and why we should use computers in the first place, to what purpose(s) they should be programmed or how programmers should be trained. These questions will not be discussed below. However, it is clear that some people do want to use computers and want to teach programming interactively. Therefor some people want to compare languages to do this well. Though rapid results and interactivity are adamant, and are provided by Wolframese, serious programs need serious documentation. As this requires didactical skills too, programming should interest the average science teacher if only for didactical reasons.

In Wolframese documentation takes much more space then the programs themselves, as will be illustrated below.

Joris Verrips teached mathematics for a while and uses Wolframese for his research in speaking software. Mail: j.verrips@planet.nl

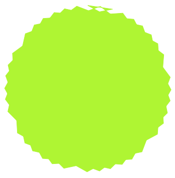
Psychedelic wallpaper designs.

In[18]:=

For fascinating short Wolfram programs see http : // wolframtap.tumblr.com. To read them requires expertise.

```
In[18]:= J = NestList[# / .86 &, Table[{Sin @ t, Cos @ t}, {t, 0, 38}], 24];
         "" Graphics[{Riffle[RandomColor[Length[J]], Polygon /@ J]}]
```

Out[19]=



We will try to create psychedelic wallpaper, try to document as well as show some related details that should be understandable to the average science teacher.

Polygon is a function in Wolframese that needs a list of points. For a regular n-faced polygon these points have coordinates like {r Cos[α], r Sin[α]} with n $\alpha = 2\pi$. This may be done with a Table command in steps of $2\pi/10$ and as follows:

```
In[20]:= Table[ {Cos[x], Sin[x]}, {x, 0, 2 π, 2 π/10}] //N (* // N rounds off *)
Out[20]= {{1., 0.}, {0.809017, 0.587785}, {0.309017, 0.951057}, {-0.309017, 0.951057},
          {-0.809017, 0.587785}, {-1., 0.}, {-0.809017, -0.587785},
          {-0.309017, -0.951057}, {0.309017, -0.951057}, {0.809017, -0.587785}, {1., 0.}}
```

To use with different values for n we define our own function polygon, in lowercase.

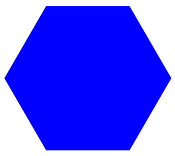
```
polygon[vertices_: 6, size_: 1, rotation_: 0] :=
 Polygon[size Table[{Cos[rotation + i 2 π / vertices],
     Sin[rotation + i 2 π / vertices]}, {i, 0, vertices}]]
(* The parameters have default values 6, 1 and 0,
we should therefor see six points of sizes 1 because Cos^2 + Sin^2 = 1  *)
polygon[] // N
```

```
Out[22]= Polygon[{{1., 0.}, {0.5, 0.866025}, {-0.5, 0.866025},
  {-1., 0.}, {-0.5, -0.866025}, {0.5, -0.866025}, {1., 0.}}]
```

This Polygon with a list of points as parameter can be used by Graphics to display a colored form.

```
In[23]:= (* The empty string influences the size that is displayed *)
"" Graphics[{Blue, polygon[]}]
```

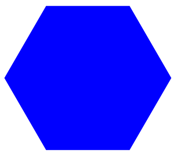
Out[23]=



An alternative is to use the built in functions
RegularPolygon[6] and Style, assuming of course
one knows that they exist : there are 5000 of them.

```
"" Graphics[Style[RegularPolygon[6], Blue]]
```

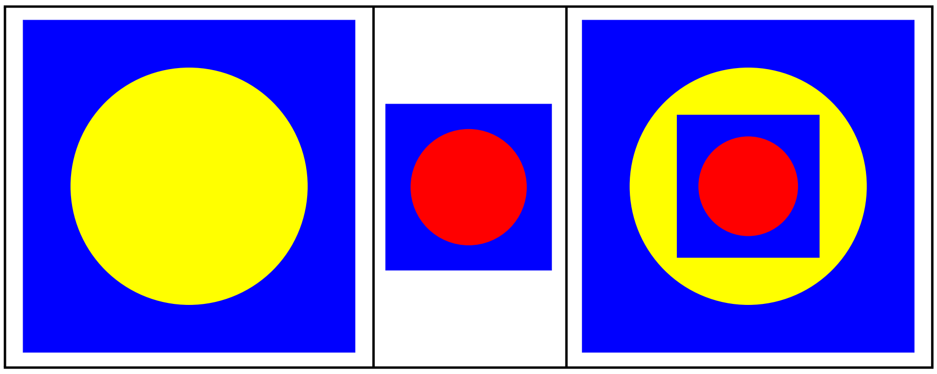
Out[24]=



Combining several colored forms.

```
(* Rectangle and Disk are other predefined functions *)
pic1 = Graphics [ {Blue, Rectangle [.7 {-1., -1.}, .7 {1., 1.}],
     Yellow, Disk[{0, 0}, .5 ]}, ImageSize → Small ] ;
pic2 = Graphics[ {Blue, Rectangle [.3 {-1., -1.}, .3 {1., 1.}],
     Red, Disk[{0, 0}, .3  .7]}] ;
(* They are shown in a grid that is displayed below because
 it does NOT end with ; as do pic1 and pic2.
     Show combines several pictures.  *)
threepictures = "" Grid[{{pic1, pic2, Show[{pic1, pic2}]}}, Frame → All]
```

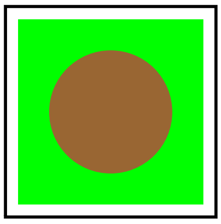
Out[27]=



Shorter and more abstract.

```
In[28]:= colors = {Green, Brown} ;
forms = {Rectangle[-.9 {x, x}, .9 {x, x}], Disk[{0, 0}, .6 x]} ;
(* The function /. x → 1 below means Replace x by 1 *)
pic3 = Framed[Graphics[Riffle[colors, forms], ImageSize → Tiny ] /. x → 1]
```

Out[30]=



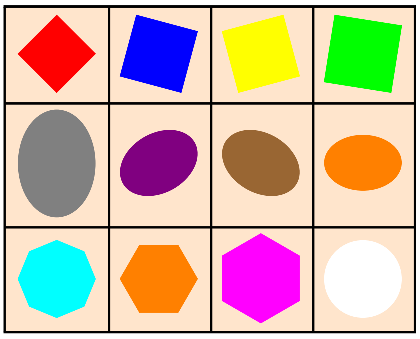
Defining more forms, most with function polygon as defined earlier.

```
In[31]:= square[x_] := polygon[4, x, 0]
         rotsquare1[x_] := polygon[4, x, π / 6]
         rotsquare2[x_] := polygon[4, x, π / 3]
         rotsquare3[x_] := polygon[4, x, π / 5]

         disk[x_] := Disk[{0, 0}, x]
         ellips1[x_] := Rotate[Disk[{0, 0}, {x, .72 x}], π / 6]
         ellips2[x_] := Rotate[Disk[{0, 0}, {.72 x, x}], π / 3]
         ellips3[x_] := Disk[{0, 0}, {.72 x, x}]
         ellips4[x_] := Disk[{0, 0}, {x, .72 x}]

         octagon[x_] := polygon[8, x, 0]
         hexagon1[x_] := polygon[6, x, 0]
         hexagon2[x_] := polygon[6, x, π / 6]

         (* We again display in a Grid, three rows,
         again with a replace command in postfix notation. *)
         "" Grid[
             {{Graphics[{Red, square[x]}], Graphics[{Blue, rotsquare1[x]}],
               Graphics[{Yellow, rotsquare2[x]}], Graphics[{Green, rotsquare3[x]}]},
               {Graphics[{Gray, ellips3[x]}], Graphics[{Purple, ellips1[x]}],
               Graphics[{Brown, ellips2[x]}], Graphics[{Orange, ellips4[x]}]},
              {Graphics[{Cyan, octagon[x]}], Graphics[{Orange, hexagon1[x]}],
               Graphics[{Magenta, hexagon2[x]}], Graphics[{White, disk[x]}]}},
                 Frame → All, ItemSize → 3, Background → LightOrange] /. x → 1
```

Out[57]=


Lists of forms, colors, sizes and pictures that depend on a number.

```
In[44]:= preparedata[number_] := (
           forms := RandomSample[{disk, square, ellips1, rotsquare1, ellips2, rotsquare2,
                                  hexagon1, rotsquare3, hexagon2, octagon}, number];
            pform = Permutations[forms];
            colorlist := RandomSample[ColorData[5, "ColorList"], number];
            pcolor = Permutations[colorlist];
            sizes = Table[0.64^i, {i, 1, number}]; )
```

```
(* Check your understanding: *)
preparedata[3];
Grid[{pform[[1]], pcolor[[1]], sizes }, Frame → All ]
```

Out[70]=

hexagon2	rotsquare2	square
🟧	🟫	🟧
0.64	0.4096	0.262144

Pictures are stored in tablepictures with Riffle and MapThread[Compose, {someforms, sizes}].

```
In[45]:= (* Explaining Riffle with dummy variables form1 and form2,
         as yet undefined functions *)
         Riffle[{Red, Blue}, {form1, form2}]

Out[45]= {🟥, form1, 🟦, form2}

In[67]:= (* Explaining Compose or @ *)
         {Compose[form1, 4], form1 @ 4}

Out[67]= {form1[4], form1[4]}

In[46]:= (* Explaining MapThread[Compose, {someforms, somesizes}} *)
          Riffle[{Red, Blue}, MapThread[Compose, {{form1, form2}, {1.0, 0.7}}]]

Out[46]= {🟥, form1[1.], 🟦, form2[0.7]}
```

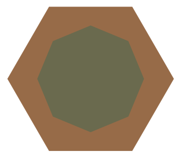
```
(* You should now be able to understand these:  *)
picture[listofcolors_, someforms_, sizes_] :=
 Riffle[listofcolors, MapThread[Compose, {someforms, sizes}]]
(* We store pictures that combine forms of different color. *)
tablepictures := Table[picture[pcolor[[i]], pform[[j]], sizes],
   {i, 1, Length[pcolor]}, {j, 1, Length[pform]}];
```

```
In[49]:= (* Explaining again. *)
         preparedata[2];
         Grid[{tablepictures[[1]][[1]]}, Frame → All ]
         Grid[
          {{pcolor , pform, sizes, "" Graphics @tablepictures[[1]][[1]]}}, Frame → All ]
```

Out[50]=

🟫	Polygon[{{0.64, 0.}, {0.32, 0.554256}, {-0.32, 0.554256}, {-0.64, 0.}, {-0.32, -0.554256}, {0.32, -0.554256}, {0.64, 0.}}]	🟩	Polygon[{{0.4096, 0.}, {0.289631, 0.289631}, {0., 0.4096}, {-0.289631, 0.289631}, {-0.4096, 0.}, {-0.289631, -0.289631}, {0., -0.4096}, {0.289631, -0.289631}, {0.4096, 0.}}]

Out[51]=

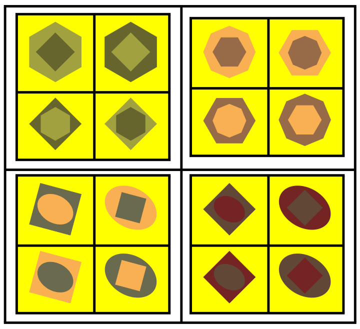
{{🟫, 🟩}, {🟩, 🟫}}	{{hexagon1, octagon}, {octagon, hexagon1}}	{0.64, 0.4096}	

The function wallpaper computes tablepictures and uses it in a Grid. Different calls give different output because of the Random command in preparedata.

```
wallpaper[n_ /; 1 < n < 5] :=
 (preparedata[n]; (* This is required for tablepictures below. *)
  Grid[
   Partition[RandomSample[Flatten[
      Table[Graphics @tablepictures[[i , j]],
       {i, 1, n!}, {j, 1, n!}]
     ], (n!)^2], n!],
   ItemSize → 2, Frame → All,  Background → Yellow ])
```

```
In[84]:= Grid[Partition[Table[wallpaper[2], {4}], 2], Frame → All]
```

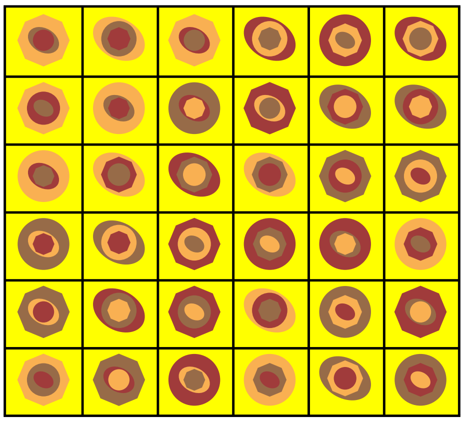
Out[84]=



The function wallpaper is rather slow :

```
In[74]:= Timing[wallpaper[3]]

Out[74]= {0.234375, 
```



```
}

         (* This is rather slow,
         at .23 for n=3 respectively at 159 seconds for n=4 and with suppressed output,
         see below. The reason is that tablepictures
          will be computed and used n! * n! times
           Therefor we also implemented wallpaper2 with
          tablepictures2 defined as a constant  *)

In[85]:= Timing[wallpaper[4];] (*  The semicolon suppresses output *)

Out[85]= {159.094, Null}

In[103]:= wallpaper2[n_ /; 1 < n < 5] :=
           (preparedata[n]; (* This is required for tablepictures2 below. *)
            tablepictures2 = tablepictures ;
            (* =
             or Set means tablepictures2 is a constant that is computed just once *)
            Grid[
             Partition[RandomSample[Flatten[
                 Table[Graphics @tablepictures2[[i , j]],
                   {i, 1, n!}, {j, 1, n!}]
               ], (n!)^2], n!],
             ItemSize → 2, Frame → All,  Background → Yellow ])
```

We now test if wallpaper2 is faster than wallpaper

```
In[101]:= Timing[wallpaper2[3];]
          Timing[wallpaper2[4];] (* Again, due to the semicolons we see no output *)

Out[101]= {0.015625, Null}

Out[102]= {0.265625, Null}
```