Autodraw Script Language

Basic Idea

A **Graph** is a combination of drawing operations. Strictly speaking, a graph is what you will see after applying the set of operations on a white clean canvas. But here, to make it easy to express, just call the set of operations Graph.

A drawing operation is expressed explicitly by exactly a line of code. The code is in form of [OPERATION] [OPTIONS]. For example

```
line 0 0 1 1
line 0 1 1 0
circle 0.5 0.5 0.707
```

The above codes simply draw a pair of crossing lines and a circle around them.

Variable

Now, we want to extend the functionality of this simple system by a tiny little bit, that is to allow a simple string replacement mechanism. So we introduce variable in this system, and `set` operation to assign values to them.

```
set x 0.5
set y 0.5
set radius 0.707
line 0 0 1 1
line 0 1 1 0
circle x y 0.707
```

You can picture the system in the following way: it maintains a dictionary from `string` to `double`, each `set` operation causes it to add or update an entry in the map and for each item in other operations it looks up the dictionary and put the result in the place. Note that mathematical expression is not supported here.

Transform

Next, we want another functionality so that we can apply some linear transform to the entire graph or part of the graph. A transform is defined by a 3x3 transform matrix.

$$\begin{pmatrix} a & b & x \\ c & d & y \\ 0 & 0 & 1 \end{pmatrix}$$

Combination of transforms corresponds to multiplication of matrices.

A 2D vector $\vec{v} = (x, y)$ is extended to $(x, y, 1)$. To apply a transform T to vector, first calculate $\vec{u} = T\vec{v}$. Then scale $\vec{u}$ such that the last entry is 1.

Transform	Matrix
Rotate	$\begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$
Flip X	$\begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
Flip Y	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
Scale X	$\begin{pmatrix} t & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
Scale Y	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & t & 0 \\ 0 & 0 & 1 \end{pmatrix}$
Translate	$\begin{pmatrix} 1 & 0 & x \\ 0 & -1 & y \\ 0 & 0 & 1 \end{pmatrix}$

To define a transform variable, we have the following operations

```
transform T a b c d x y
rotate T theta
flipx T
flipy T
flipxy T
scalex T t
scaley T t
scalexy T s t
translate T x y
combine T T1 T2
```

Here the combination of transforms means let `T` be the transform which has the same affect as applying `T1` first and then `T2`.

To apply a transform or cancel it, we use

```
push T
pop
```

Here, the `push T` operation means multiply the matrix `T` to the current matrix at the top of matrix stack, and push the result matrix into the stack.

Definition of Graph

Next, we would like to allow defining a graph and reuse it like a function in programming languages. To do this, we encircle the code by a pair of

```
begin name
end
```

One script file can contain multiple definitions of graphs.

Combine Graphs

Next, we add the functionality of combining graphs. This is like invoking a function in another one. For example, if we already have defined the way to draw a plane

```
begin plane
*** blah ... plane is finished ***
end
```

Now we want to draw a plane in an airport. We can do this by

```
*** blah ... airport is finished ***
draw plane
```

We want to draw more than one plane, in different places. We do this by applying different transformations.

```
*** blah ... airport is finished ***
draw plane
translate T 1 0
push T
draw plane
pop
translate T 2 0
push T
draw plane
pop
```

Looks messy. To simplify the repetation of pushing and poping matrices, we can allow `T` to be parameter of the operation.

```
*** blah ... airport is finished ***
draw plane
translate T 1 0
draw T plane
translate T 2 0
draw T plane
```