

variables & functions

vicky isley & paul smith
www.boredomresearch.net
aoc@boredomresearch.net

introduction

This session will cover the following:

AM

- Students present their week three assignment.
- Exercise to illustrate variable declaration, assignment and manipulation.
- Detailed description of variables and data types.
- Presentation on writing, declaring and assigning variables.
- Presentation on creating functions.
- Examples of using functions to build shape primitives.
- Presentation "building tools for creativity, artist as technologist"

PM

- Practical workshop "writing functions to build software tools"
- Present next week's assignment.

Present findings from week three assignment

Present results from week three assignment.

Look at differences in approach to problem.

Question lack of response (too intimidating ? confusion? lack of interest?)

Exercise: making a box....

here is a number... put it in box32. What box?

This is an illustration of what happens when we refer to something that does not yet exist.

[1] make four boxes and name them x_box, y_box, w_box, h_box (boxes will be represented by people a,b,c,d)

[2] put 0 into x_box and y_box and 10 into w_box and 100 into h_box

[3] give person e the job of calculator, note they can only perform one calculation at a time.

[4] give person f the job of draw_function, they will take the given data and draw a rectangle to the screen (table)

[5] give person g the job of executing the program and manipulating the data before passing it to the draw function

[6] take a photo at each iteration of the program.

[7] run animation and discuss what happened.

[*note] We may need to debug this program.

Variables and data types in Processing

What is a variable

A variable is a named pointer to a location in the computer's memory. You can think of it as a storage locker for data. Computers only process information one instruction at a time - using a variable allows a programmer to save information calculated at one point in the program and refer back to it at a later time.

Data types

Variables in Processing (and other strictly typed languages) must be defined or declared with their type.

The different (primitive) types that Processing/Java support are as follows:

int - An integer. Whole numbers only.

boolean - True or False. Often the result of an expression.

float - A number with a decimal point. 1.5, 3.987 and so on.

byte - An 8 bit value. Ranges from -127 to 128 in value.

char - A single character.

Declaring, assigning and manipulating variables

Declaring

Variables can hold different types of information and in Processing, we have to explicitly define what kind of information we want to hold in a variable before we begin to use the variable as seen in the above group exercises. In order to use a variable, you must first declare your intentions. Variables are declared by first stating the type, followed by the name. Note that your variable names must be one word (no spaces) and must start with a letter (they can include numbers, but cannot start with a number).

For example: `int count;`

The variable can also be initialized with a value, another variable, or by evaluating an expression. Here are some examples:

```
int count = 0; // declare an int named count, assigned the value 0
```

```
char letter = 'a'; // declare a char named letter, assigned the value 'a'
```

```
double d = 132.32; // declare a double named d, assigned the value 132.32
```

```
boolean happy = false; // declare a boolean named happy, assigned the value false
```

```
float x = 4.0f; // declare a float named x, assigned the value 4.0 // (note the 'f'; required since default floating point type is double)
```

```
float y; // declare a float named y (no assignment)
```

```
y = 5.2f; // assigned the value 5.2 to the previously declared y
```

```
float z = x*y + 15.0f; // declare a variable named z, assign it the value which is x times y + 15.0.
```

Variables must be declared with their type as shown here:

```
int myinteger;  
float yourfloat;  
boolean mybool;
```

Assigning:

Once they are declared, you are free to assign values to them and subsequently use them in operations:

```
myinteger = 5;  
myinteger = 5 + 5;  
myinteger = myinteger - 5;  
yourfloat = 5.5;
```

Of course, as shown in the examples above, you can take a shortcut and do both declaration and assignement in one step:

```
int someint = 89;
```

Procedural programming, creating functions.

Creating your own functions

In processing, we've been using functions all along. When we say "line(0,0,200,200)" we are calling the function "line", a "built-in" function of the processing environment. When we say "void setup()", we are declaring the "setup" function, which is a special processing function that is called when the program first starts.

The function declaration/definition has three parts:

1.) return type 2.) function name 3.) parameters

It looks like this:

```
ReturnType FunctionName ( Parameters ) { code body of function }
```

You can create your own functions that you can call anywhere else in your code.

Here's a simple example:

```
void cornerToCorner() {  
  line(0,0,width,height);  
}
```

This is a simple function that performs one basic task – drawing a line from corner to corner. It has no return type, and therefore we say "void". It has a function name: cornerToCorner. It has no parameters (and therefore nothing is placed in between the parentheses). We can call the function (and therefore have the instructions of the function executed) like this:

```
cornerToCorner();
```

Local/Global variables:

If you declare a variable outside of a block of code, in the main processing code section it is a global variable. If you declare it inside of a block of code, it is a local variable. Local to that block.

A global and a local variable:

```
int myint = 90; // Global
```

```
void setup()  
{  
  int myotherint = 100; // Local to the setup function  
}
```

Using functions to extend the creative possibilities of software

[1] building primitives for reuse

Last week we were using Processing's built in 'line' function to execute simple shapes and compositions. Today we are going to use the **beginShape()** and **endShape()** functions to create more complex forms.

beginShape() begins recording vertices for a shape and **endShape()** stops recording.

The value of the **MODE** parameter tells it which types of shapes to create from the provided vertices. The parameters available for **beginShape()** are **LINES**, **LINE_STRIP**, **LINE_LOOP**, **TRIANGLES**, **TRIANGLE_FAN**, **TRIANGLE_STRIP**, **QUADS**, **QUAD_STRIP**, and **POLYGON**.

If there is no **MODE** specified, **POLYGON** is used.

After calling the **beginShape()** function, a series of **vertex()** commands must follow. To stop drawing the shape, call **endShape()**. The **vertex()** function with two parameters specifies a position in 2D and the **vertex()** function with three parameters specifies a position in 3D. Each shape will be outlined with the current stroke color and filled with the fill color. Transformations such as **translate()**, **rotate()**, and **scale()** do not work within **beginShape()**.

Example to draw a filled square:

```
beginShape();  
vertex(30, 20);  
vertex(85, 20);  
vertex(85, 75);  
vertex(30, 75);  
endShape();
```

Variables must be declared with their type as shown here:

```
int myinteger;  
float yourfloat;  
boolean mybool;
```

Assigning:

Once they are declared, you are free to assign values to them and subsequently use them in operations:

```
myinteger = 5;  
myinteger = 5 + 5;  
myinteger = myinteger - 5;  
yourfloat = 5.5;
```

Of course, as shown in the examples above, you can take a shortcut and do both declaration and assignement in one step:

```
int someint = 89;
```

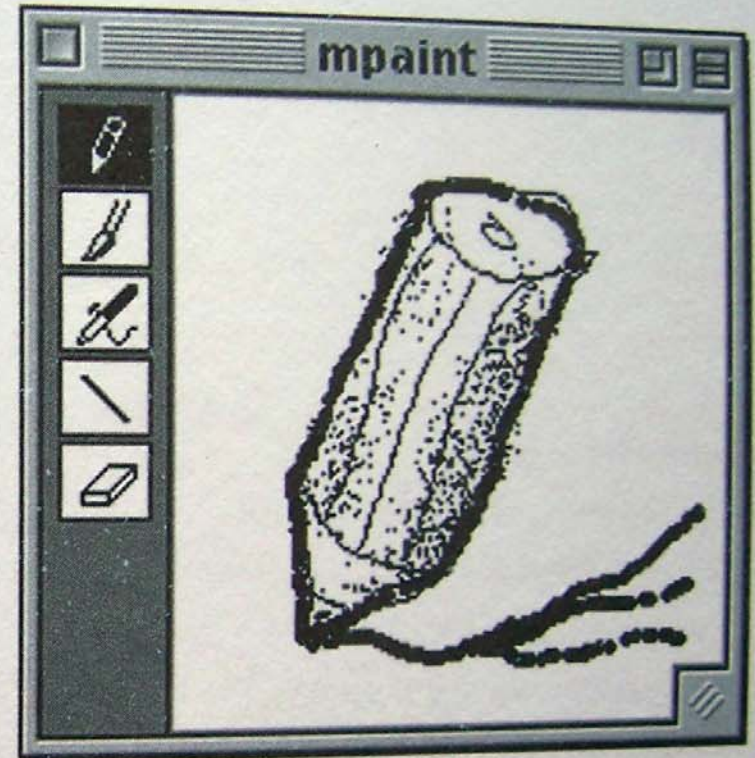
Presentation: Building tools for creativity, artist as technologist

'With each new era, there is a technical process that people copy from one another verbatim, allowing it to accidentally unify all the resulting craft. In painting, brush strokes have always rippled with the texture left by the brush. The musical style of classical music composers were governed by the makers of musical instruments (and vice versa). If it is not Adobe, then it is Apple. If it is not Apple, then it is who ever made the hardware. If it's not computers, then we will find some other standardizing factor to blame our emerging uniformity on. The truth is that today's leading designers are also very good technologists, and that is what keeps them original in a creative and human way. The cage is removed and their hearts shine through much clearer.' SetPixel essay, Josh Nimoy Oct 2004

Josh Nimoy is a designer, artist, technologist, and convicted narcissist. Nimoy holds a B.A. from UCLA and a masters from NYU ITP - having also had stints at Calarts and the MIT Media Lab.

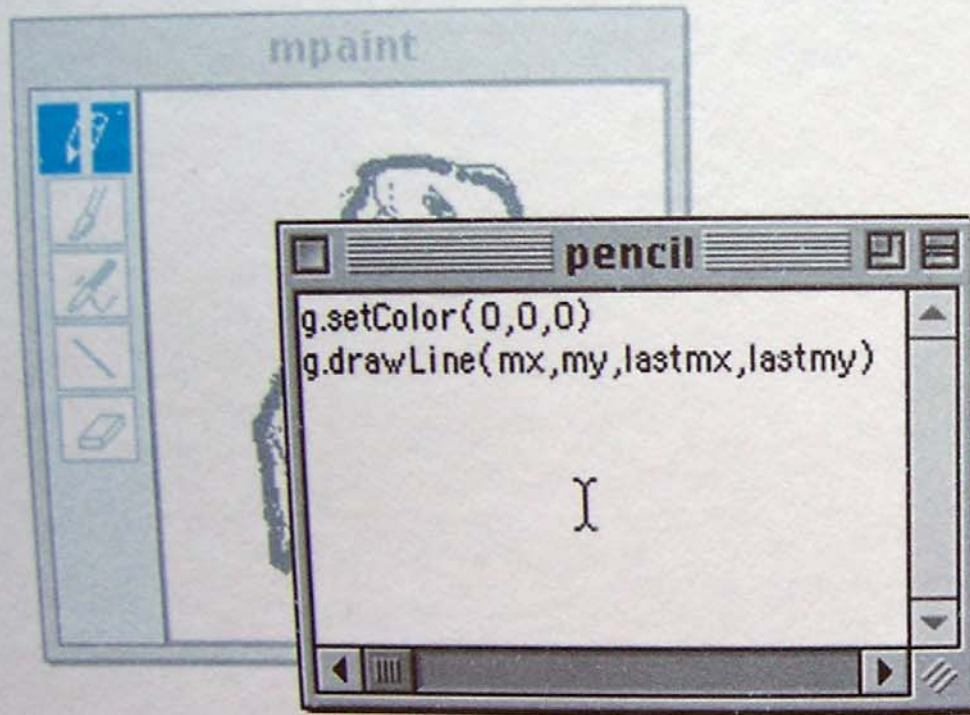
John Maeda's Macpaint

A moment of clarity presented itself five years later. I created a simple painting tool in the image of one of my first inspirations, the software system originally bundled with the Macintosh computer called MacPaint, credited to Bili Atkinson. I wondered, "What if in 1984 MacPaint were designed in such a way that it was incomplete—much like a piece of unfinished furniture or a pen-making kit."



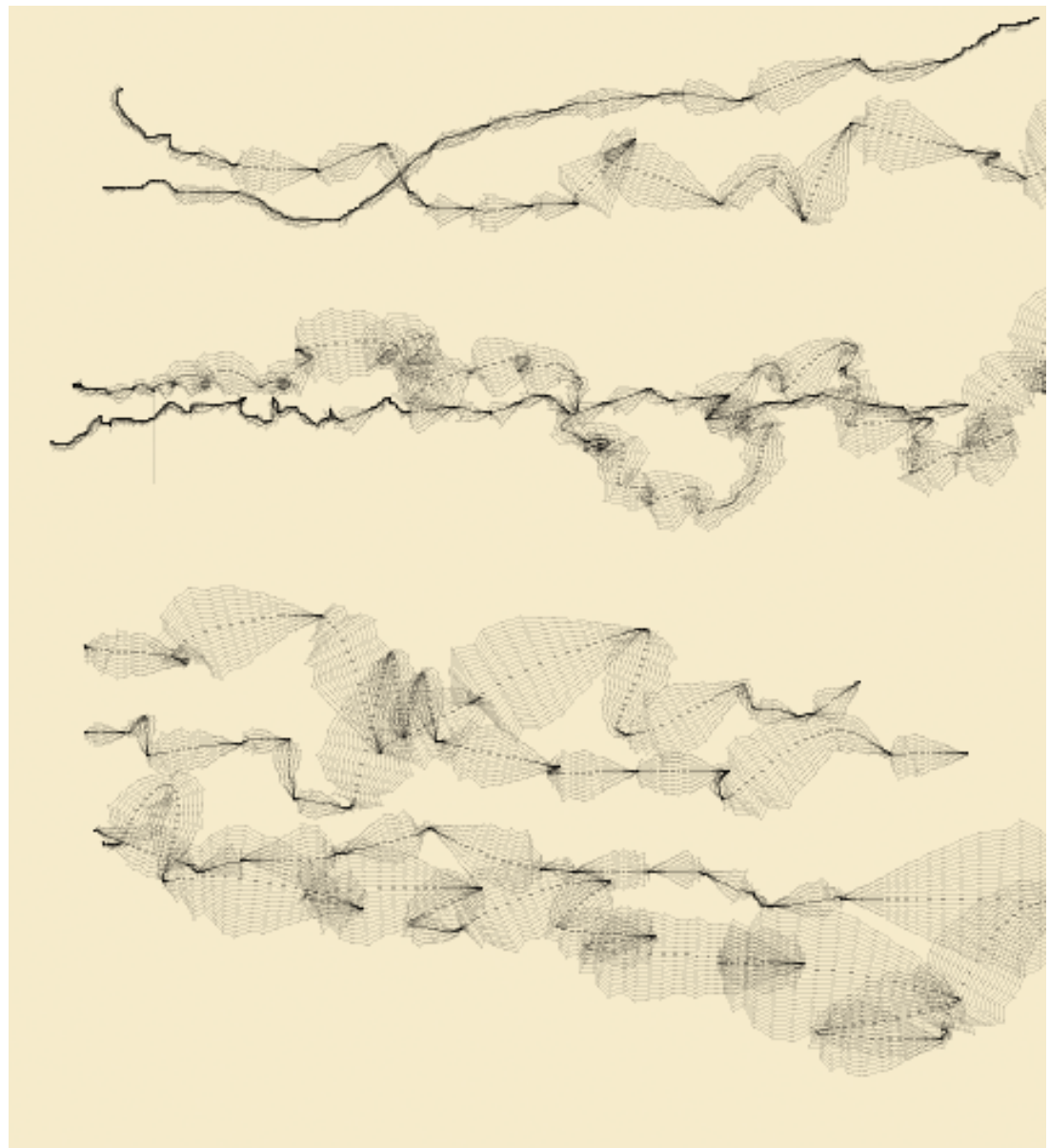
'The essence of what exists i.e the entire set of possibilities generated from a core idea – is rendered into reality as computational structures.'
John Maeda.

John Maeda's Macpaint



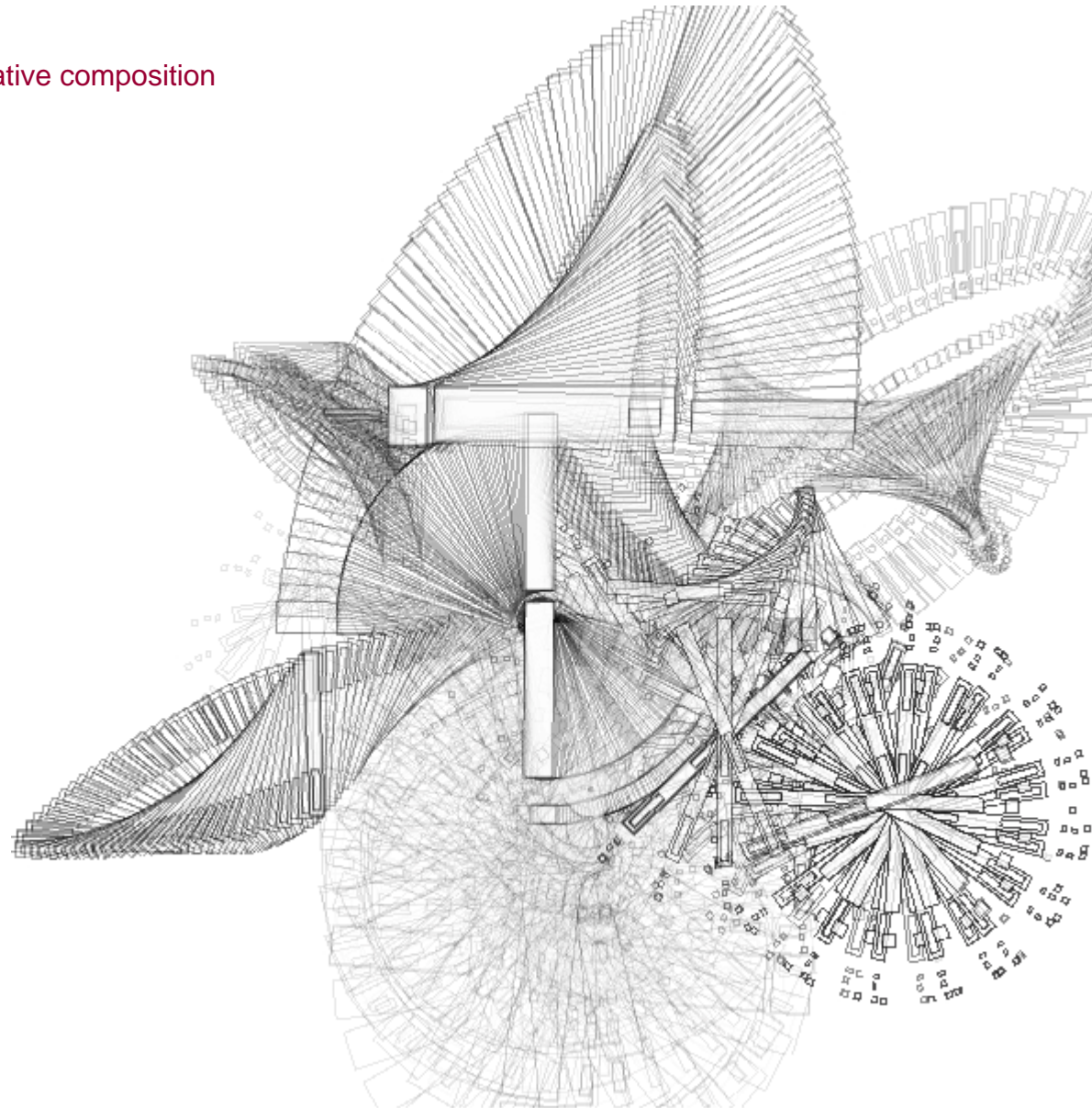
By cracking the icon for a tool open, you would reveal the programming code that realized that specific function. By changing the code, the user could change the way the pencil behaved at will, much in the same way a real pencil might be customized. I believe that such approaches are the basis for fine artistic control over a system. We must have the ability to dig into the tools and materials, whether they be pencil, paper, metal, oil paints—whatever—to reshape, manipulate, and recast them into something either new or at least relevant to our individual styles of thought.

PencilBrush (2005) – David Lu (<http://david-lu.net>) a behavioural variation on a simple drawing tool.



amorphoscapes – Stanza (<http://www.stanza.co.uk>)

An example of an accumulative composition from simple forms.



PM

Workshop Creating functions to build software tools.

[1] Plan your primitive on paper

[2] Consider what parameters could be passed to it.

[3] Write a notation to explain the functionality to another user.

[4] Start experimenting in processing use the `beginShape()` and `endShape()` functions

Week 4 Assignment: Create your own program to execute a composition

Following on from today's examples exercises and presentations make a number of simple functions that draw shapes and line in a way that extends the possibilities offered by Processing's shape primitives. The aim is to make a tool that will be later used by you and others. For this reason you will need to write an explanation of the tool and give a clear example of how it might be used.

Write the function as a .pde including an example of its use in the draw function and commented description and instruction.

Name the file with the with your initials and w4a

e.g. my file would be called psw4a and the processing code file would be called psw4a.pde.

Please complete and email your final function(s) as .pde files to us at
aoc@boredomresearch.net
by Friday 3rd March 2006

By the end of this session each student will:

[1] understand the principles of data storage in the form of variables and the different data types

[2] be able to declare and assign variables.

[3] understand the significance of variable scope.

[4] understand the structure form and usage of functions.

[5] understand the return type

[6] have experience of the useful implementation of functions

[7] understand the main shape primitives and be able to create new shape functions also using the "beginShape()" and "endShape()" functions.

[8] have a good appreciation of nested functionality and its use for developing software tools

[9] appreciate the potentials for developing customized functionality for creative expression.

[10] be inspired to create new tools to extend their creative possibilities.