Center for Visual Cultures
Computer Science Department
English Department

Expressive Data

Ira Greenberg

As a painter, artist Ira Greenberg studied nature, visually searching for patterns and forms that he translated into expressive strokes of paint. As a creative coder, he continues to search, but within motifs that extend beyond the physical, natural world. In this talk, Ira will present an overview of his creative journey, visually telling the story of how his medium transmuted from paint to code. He will also feature some of his recent work, including Protobytes (studies in artificial life) and visualization in the digital humanities.

Time: Wednesday 9/28 12:30-2pm
Place: Thomas Library 224 (refreshments served)

http://www.iragreenberg.com

Review

- Only one thing executes at any time
- Scope
  - Global
  - Function
  - Block
- Variable Access Rules
  - Outer scope variables can be accessed from an inner scope
  - Inner scope variables cannot be accessed from an outer scope
- Lifetime
  - Variables come in to existence at declaration
  - Variables go out of existence when the block exits
- Shadowing
  - An inner scope variable with the same name as an outer scope variable, shadows (or hides) the outer scope variable from the inner scope.
  - Nevertheless, both variables exist, and are distinct.

Arrays

- A special kind of variable that holds not one, by many data items of a given type.
- Declared like variables, only type is followed by a pair of brackets.

```
float x;                // Can hold one float
float[] ax;             // May hold many floats
```

- Can be initialized using a special syntax involving the `new` keyword, the type, and a *size* in brackets.

```
float x = 12.3;                              // Initialized to 12.3

float[] ax = new float[3];                   // 3 floats, all 0.0
```

$$\underbrace{\texttt{float[] ax}}_{\text{Step 1}}\ \underbrace{\texttt{= new float[]}}_{\text{Step 2}}\ \underbrace{\texttt{\{ 1.2, 2.3, 3.4 \};}}_{\text{Step 3}}\quad \texttt{// Initialized}$$

Arrays

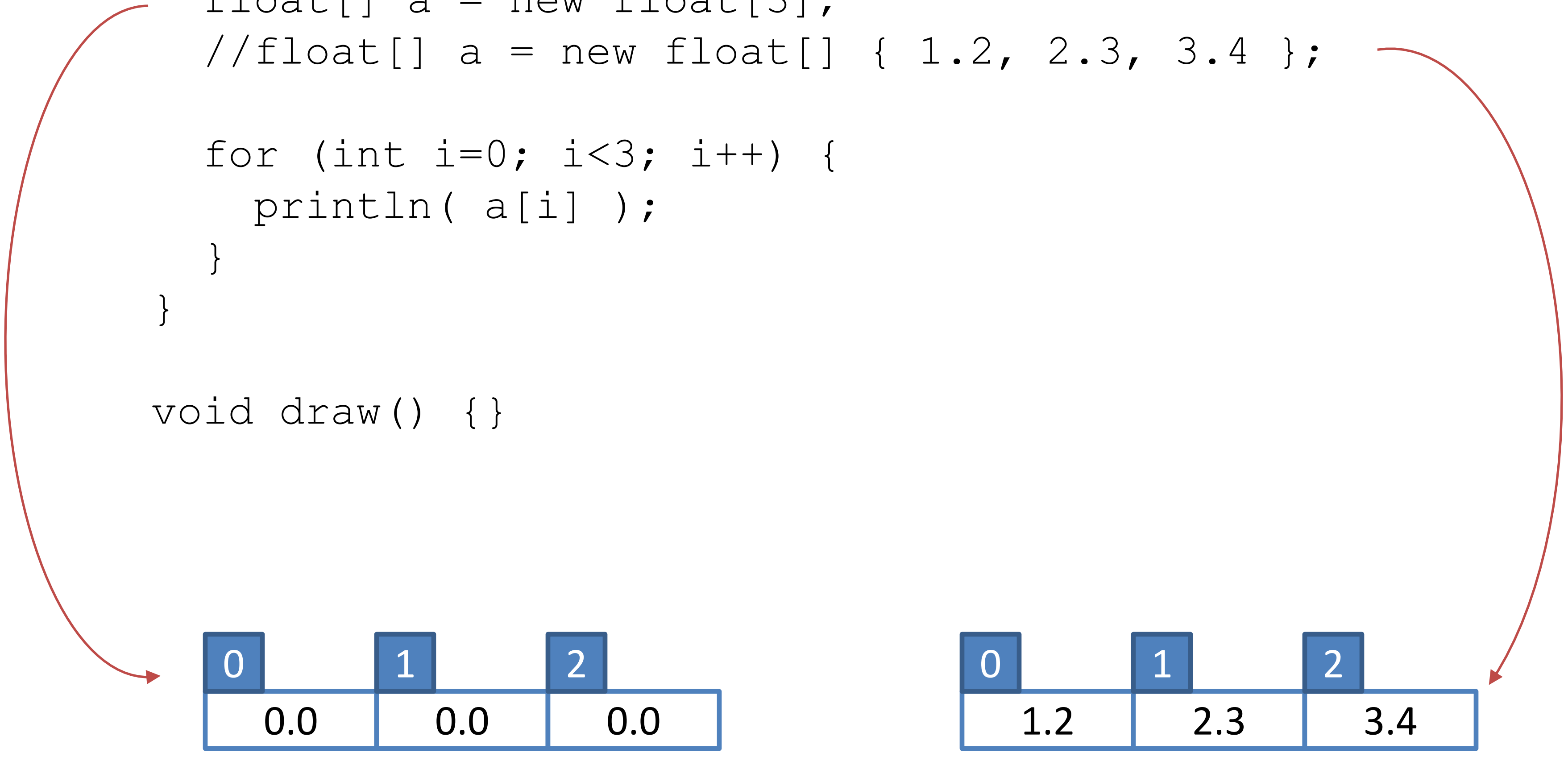
- Individual data items are accessed with an index and square brackets.
    - **Indexes start at 0!**
- The length of an array can be determined using its `length` property.
    - The length of an array is one greater than the last valid index.
- Arrays can be passed to, and returned from functions.
    - … just like other data types

```
void setup() {

  float[] a = new float[3];
  //float[] a = new float[] { 1.2, 2.3, 3.4 };

  for (int i=0; i<3; i++) {
    println( a[i] );
  }
}

void draw() {}
```



Built-in Array Functions

append(*array, item*)
- returns a new array expanded by one and add item to end

expand(*array, newSize*)
- returns a new array with size increased to newSize

shorten(*array*)
- returns a new array shortened by one

concat(*array1, array2*)
- returns a new array that is the concatenation of array1 and array2

subset(*array, offset [, length]*)
- returns a subset of array starting at offset and proceeding for length (or end)

splice(*array, value|array2, index*) or
- returns a new array with value or array2 inserted at index

sort(*array*)
- returns a new array sorted numerically or alphabetically

reverse(*array*)
- returns a new array with all elements reversed in order

```
// arrays1
String[] names = new String[5];

void setup() {
  size(500, 500);
  background(200);
  names[0] = "Chococat";
  names[1] = "Cinnamoroll";
  names[2] = "Landry";
  names[3] = "Pekkle";
  names[4] = "Purin";
}

void draw() {
  fill(0);
  int n = int( random(-0.1, names.length-0.1) );
  float x = random(0, width);
  float y = random(0, height);
  text( names[n], x, y );
}

void mousePressed() {
  names = shorten(names);
  background(200);
}
```

Example: Functions + Arrays + Loops

```
// gaussian
float[] bins = new float[500];

void setup() {
  size(500, 500);
  stroke(255);
  frameRate(15);
}

void draw() {

  for (int i=0; i<50; i++) {         // Add 50 new numbers
    int idx = int( map( gaussian(), -2.0, 2.0, 0, width ));
    bins[idx]++;
  }

  background(0);                     // Redraw all
  for (int i=0; i<bins.length; i++) {
    line(i, height, i, height-bins[i]);
  }
}

// Gaussian distribution sampler
float gaussian() {
  // Polar form of Box-Muller transformation
  // Converts uniform in [-1,1] to gaussian w/ mean=1.0, sd=1.0
  float x1, x2, w, y1, y2;

  while (true) {
    x1 = random(-1.0, 1.0);
    x2 = random(-1.0, 1.0);
    w = x1 * x1 + x2 * x2;
    if (w < 1.0) break;    // Try again if w >= 1.0
  }

  // Generate two samples
  w = sqrt( (-2.0 * log( w ) ) / w );
  y1 = x1 * w;
  y2 = x2 * w;

  return y1;
}
```

```
// bounce1
float ay = 0.2;      // y acceleration (gravity)
float sx;            // x position
float sy;            // y position
float vx;            // x velocity
float vy;            // y velocity

void setup() {
  size(500, 500);
  fill(255, 0, 0);
  smooth();
  ellipseMode(CENTER);

  sx = random(0.0, width);
  sy = random(0.0, 10.0);
  vx = random(-3.0, 3.0);
  vy = random(0.0, 5.0);
}
```

```
void draw() {
  background(255);

  // Move ball
  sx += vx;
  sy += vy;
  vy += ay;

  // Bounce off walls and floor
  if (sx <= 10.0 || sx >= (width-10.0)) vx = -vx;
  if (sy >= (height-10.0) && vy > 0.0) vy = -0.9*vy;

  // Draw ball
  ellipse( sx, sy, 20, 20);
}
```

```
// bounce2
float ay = 0.2;     // y acceleration (gravity)
float sx;    // x position
float sy;    // y position
float vx;    // x velocity
float vy;    // y velocity

float sx2;    // x position
float sy2;    // y position
float vx2;    // x velocity
float vy2;    // y velocity

void setup() {
  size(500, 500);
  fill(255, 0, 0);
  smooth();
  ellipseMode(CENTER);

  sx = random(0.0, width);
  sy = random(0.0, 10.0);
  vx = random(-3.0, 3.0);
  vy = random(0.0, 5.0);

  sx2 = random(0.0, width);
  sy2 = random(0.0, 10.0);
  vx2 = random(-3.0, 3.0);
  vy2 = random(0.0, 5.0);
}
```

```
void draw() {
  background(255);

  // Move ball
  sx += vx;
  sy += vy;
  vy += ay;

  sx2 += vx2;
  sy2 += vy2;
  vy2 += ay;

  // Bounce off walls and floor
  if (sx <= 10.0 || sx >= (width-10.0)) vx = -vx;
  if (sy >= (height-10.0) && vy > 0.0) vy = -0.9*vy;

  if (sx2 <= 10.0 || sx2 >= (width-10.0)) vx2 = -vx2;
  if (sy2 >= (height-10.0) && vy2 > 0.0) vy2 = -0.9*vy2;

  // Draw ball
  ellipse( sx, sy, 20, 20);
  ellipse( sx2, sy2, 20, 20);
}
```

```
// bounce3
int nBalls = 200;

float ay = 0.2;     // y acceleration (gravity)
float[] sx = new float[nBalls];   // x position
float[] sy = new float[nBalls];   // y position
float[] vx = new float[nBalls];   // x velocity
float[] vy = new float[nBalls];   // y velocity

void setup() {
  size(500, 500);
  fill(255, 0, 0);
  smooth();
  ellipseMode(CENTER);

  for (int i=0; i<nBalls; i++) {
    sx[i] = random(0.0, width);
    sy[i] = random(0.0, 10.0);
    vx[i] = random(-3.0, 3.0);
    vy[i] = random(0.0, 5.0);
  }
}
```

```
void draw() {
  background(255);

  for (int i=0; i<nBalls; i++) {
    // Move ball
    sx[i] += vx[i];
    sy[i] += vy[i];
    vy[i] += ay;

    // Bounce off walls and floor
    if (sx[i] <= 10.0 || sx[i] >= (width-10.0))
        vx[i] = -vx[i];
    if (sy[i] >= (height-10.0) && vy[i] > 0.0)
        vy[i] = -0.9*vy[i];

    // Draw ball
    ellipse( sx[i], sy[i], 20, 20);
  }
}
```

bounce1 vs. bounce3

```
// bounce1


float ay = 0.2;      // y acceleration
float sx;            // x position
float sy;            // y position
float vx;            // x velocity
float vy;            // y velocity

void setup() {
  size(500, 500);
  fill(255, 0, 0);
  smooth();
  ellipseMode(CENTER);


  sx = random(0.0, width);
  sy = random(0.0, 10.0);
  vx = random(-3.0, 3.0);
  vy = random(0.0, 5.0);

}
```

```
// bounce3
int nBalls = 200;

float ay = 0.2;
float[] sx = new float[nBalls];
float[] sy = new float[nBalls];
float[] vx = new float[nBalls];
float[] vy = new float[nBalls];

void setup() {
  size(500, 500);
  fill(255, 0, 0);
  smooth();
  ellipseMode(CENTER);

  for (int i=0; i<nBalls; i++) {
    sx[i] = random(0.0, width);
    sy[i] = random(0.0, 10.0);
    vx[i] = random(-3.0, 3.0);
    vy[i] = random(0.0, 5.0);
  }
}
```

bounce1 vs. bounce3

```
// bounce1

void draw() {
  background(255);


  // Move ball
  sx += vx;
  sy += vy;
  vy += ay;

  // Bounce off walls and floor
  if (sx <= 10.0 || sx >= (width-10.0))
        vx = -vx;
  if (sy >= (height-10.0) && vy > 0.0)
        vy = -0.9*vy;

  // Draw ball
  ellipse( sx, sy, 20, 20);

}
```

```
// bounce3

void draw() {
  background(255);

  for (int i=0; i<nBalls; i++) {
    // Move ball
    sx[i] += vx[i];
    sy[i] += vy[i];
    vy[i] += ay;

    // Bounce off walls and floor
    if (sx[i] <= 10.0 || sx[i] >= (width-10.0))
        vx[i] = -vx[i];
    if (sy[i] >= (height-10.0) && vy[i] > 0.0)
        vy[i] = -0.9*vy[i];

    // Draw ball
    ellipse( sx[i], sy[i], 20, 20);
  }
}
```

Our four arrays might look like this…

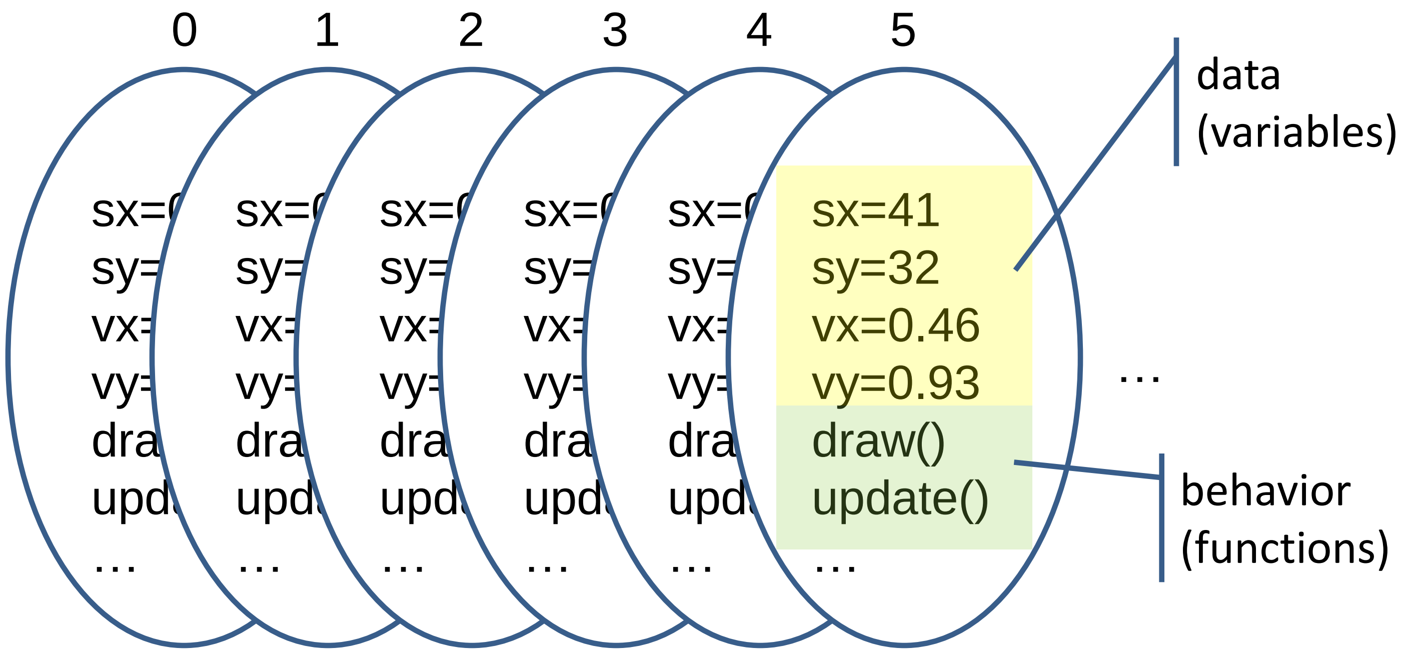
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
sx	41	68	49	3	24	5	2	38	53	72	58	11	68	82	68	28	8	5	29	42	11
sy	32	73	81	61	32	68	37	4	18	19	5	98	75	08	.6	49	23	58	65	68	63
vx	0.46	0.85	0.99	0.25	0.61	0.78	0.74	0.2	0.85	0.7	0.66	0.39	0.99	0.15	0.11	0.85	0.18	0.15	0.64	0.61	0.82
vy	0.93	0.67	0.1	0.67	0.22	0.05	0.37	0.89	0.22	0.86	0.96	0.93	0.7	0.73	0.27	0.98	0.04	0.36	0.66	0.15	0.37

Our four arrays might look like this…

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
sx	41	68	49	3	24	5	2	38	53	72	58	11	68	82	68	28	8	5	29	42	11
sy	32	73	81	61	32	68	37	4	18	19	5	98	75	08	.6	49	23	58	65	68	63
vx	0.46	0.85	0.99	0.25	0.61	0.78	0.74	0.2	0.85	0.7	0.66	0.39	0.99	0.15	0.11	0.85	0.18	0.15	0.64	0.61	0.82
vy	0.93	0.67	0.1	0.67	0.22	0.05	0.37	0.89	0.22	0.86	0.96	0.93	0.7	0.73	0.27	0.98	0.04	0.36	0.66	0.15	0.37

But we think of them like this … all data items for the same ball

Stored like this …



For each ball …
… we want the **data** (variables),
… as well as the **behavior** (functions),
… to be grouped together into a single software unit with which we can work

OBJECTS

Object Oriented Programming

- Objects are software bundles that wrap up all semantically related variables and functions.
    - Object variables are called fields
    - Object functions are called methods
- Objects are said to Encapsulate (hide) its detail
    - How an object method is implemented is not important
    - What it does is important
- Objects can be created, named and referenced with variables
    - Very similar to standard data types
- An object's individual fields and methods are accessed using syntax called dot-notation

Recall … Images

`loadImage(`*`filename`*`);`

- Loads an image from a file in the *data* folder in sketch folder.
- Must be assigned to a variable of type PImage.

`image(`*`img, X, Y, [X2, Y2]`*`);`

- Draws the image *img* on the canvas at X, Y
- Optionally fits image into box X,Y and X2,Y2

`imageMode(CORNER);`

- X2 and Y2 define width and height.

`imageMode(CORNERS);`

- X2 and Y2 define opposite corner.

Image Example

```
imageExample
    └ imageExample.pde
    └ data
          └ natura-morta.jpg
```

```
PImage img;

void setup()
{
  size(500, 400);
  img = loadImage("natura-morta.jpg");
  image(img, 50, 40);
}
```

loadImage is a function that reads image data from a file, stores it in a new PImage object, and returns the new PImage object.

The image function takes a variable of type PImage as its first argument and renders it on your sketch.

The PImage Object

- Fields

– width	*image width*
– height	*image height*
– pixels[]	*1D array holding all image pixels*

- Methods

– loadPixels()	*fill the pixels[] array with image pixels*
– updatePixels()	*copy pixels in pixels[] array back to image*
– get(*x, y*)	*reads a pixel at position x, y*
– set(*x, y, color*)	*set the color at position x, y*
– save(*path*)	*saved an image to a file*
– …	

- Related Functions

– loadImage(*path*)	*create a new PImage and init with image file*
– createImage(*w, h, form*)	*create a new empty Pimage object*
– image(*img, x, y*)	*draw a PImage to a sketch*

http://processing.googlecode.com/svn/trunk/processing/build/javadoc/core/index.html

Image Example

```
// imageExample2

PImage img;

void setup()
{
  size(500, 400);
  img = loadImage("natura-morta.jpg");
  image(img, 50, 40);
}

void mousePressed() {
  // Print the size of the PImage
  println(img.width);
  println(img.height);
}

void draw() {}
```

Dot-notation …
To access the fields and methods within an object, join the object and field/method using a dot.

```
// imageExample3

PImage img;

void setup()
{
  size(500, 400);
  img = loadImage("natura-morta.jpg");
  image(img, 50, 40);
}

void mousePressed() {
  // Fade the image to black
  float fade = 0.95;

  // Load pixel colors from image into array
  img.loadPixels();

  // Reduce value of each color component by fade
  for (int i = 0; i < img.pixels.length; i++) {
    color p = img.pixels[i];
    img.pixels[i] = color(fade*red(p), fade*green(p), fade*blue(p));
  }

  // Copy pixel colors back to array
  img.updatePixels();

  // Draw image to sketch
  image(img, 50, 40);
}

void draw() {}
```

Nearly identical to code used in vevents that continuously faded drawing.

The String Object

- Fields
  - …
- Methods
  - equals(*anotherString*)
  - length()
  - substring()
  - toLowerCase()
  - toUpperCase()

http://download.oracle.com/javase/1.4.2/docs/api/

String Method Examples

```
String s;

s = "BrynMawr";
println(s);
println( s.length() );

println( s.substring(4) );
println( s.substring(3,7) );

println( s.toUpperCase() );
println( s.toLowerCase() );
```

```
BrynMawr
8
Mawr
nMaw
BRYNMAWR
brynmawr
```