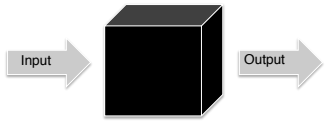
arc497i

Week 4

Review:
Functions (Modularity, Abstraction)
New
Objects (OOP)
Arrays (in more depth)
Library Example

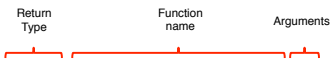
Functions



Defining functions.
3 things to remember:

- Return type
- Function name
- Arguments

Functions




```
void draw_red_square(){
   // code for drawing red square
}
```

Remember: Functions won't work unless called!

Functions
Return Types

```
void sum(int a, int b){
  println(a + b);
}

int sum(int a, int b){
  int total = a + b;
  return total;
}
```

Now we can do things like

```
int x = sum(40, 30);
int z = sum(40, 30)/3;
line(10, 10, 20, sum(14, 12));
```

Object Oriented Programming (OOP)

Object Oriented Programming (OOP)

```
int my_int();

someobject my_object();
```

When we create classes, we define new object types. They work the same as Processing's primitive data types (int, float, etc.)

Classes are also blocks of code.

Object Oriented Programming (OOP)
Anatomy of a Class

```
class Spot(){
  float x, y, diameter;
  Spot(float xpos, float ypos, float dia){
    x = xpos;
    y = ypos;
    diameter = dia;
  }
  void display(){
    ellipse(x, y, diameter, diameter);
  }
}
```

Class name

Data

Constructor

Functionality

All classes must contain name, data, constructor, and functions!

Object Oriented Programming (OOP)
How to use classes: Create Objects

```
Spot my_spot;

void setup(){
  size(100, 100);
  smooth();
  noStroke();
  sp = new Spot(33, 50, 30);
}

void draw(){
  background(0);
  sp.display();
}
```

Declaring the object

Object initialization

Call to object's function
Notice '.' syntax.

Object Oriented Programming (OOP)
How to use classes: Create Several Objects

```
Spot mySpotA, mySpotB, mySpotC;

void setup(){

  size(200, 200);
  smooth();
  noStroke();
  mySpotA = Spot(20, 50, 40);
  mySpotB = Spot(30, 10, 50);
  mySpotC = Spot(80, 50, 30);

}

void draw(){
  fill(0, 15);
  rect(0,0,width, height);
  fill(255);
  mySpotA.display();
  mySpotB.display();
  mySpotC.display();
}
```

Declaring objects

Initializing objects (using constructor)

Calling object's functions

OOP is just a way of organizing functionality around entities we define.

Object Oriented Programming (OOP)
Add Functionality

```
class Spot(){
  float x, y, diameter, speed;
  Spot(float xpos, float ypos, float dia, float sp){
    x = xpos;
    y = ypos;
    diameter = dia;
    speed = sp;
  }
  void move(){
    y+= speed*direction;
    if ((y>(height-diameter/2))||(y<diameter/2)){
      direction*=-1;
   }
  }
  void display(){
    ellipse(x, y, diameter, diameter);
  }
}
```

How would we have to change the main code?

Object Oriented Programming (OOP)
Using Arrays to Store Multiple Objects

```
int numSpots = 10;
Spot[] spots = new Spot[numSpots];

Spot my_spot;

void setup(){
  size(200, 200);
  smooth();
  noStroke();

  for (int i = 0; i<spots.length; i++){
     float x = 10 + i*16;
     spots[i] = new Spot(x, 50, 16, random(0.3, 2.0));
  }
}

void draw(){
  background(0);
  for (int i = 0; i < spots.length; i++){
    spots[i].move();
    spots[i].display();
  }
}
```

Declaring array size

Declaring array

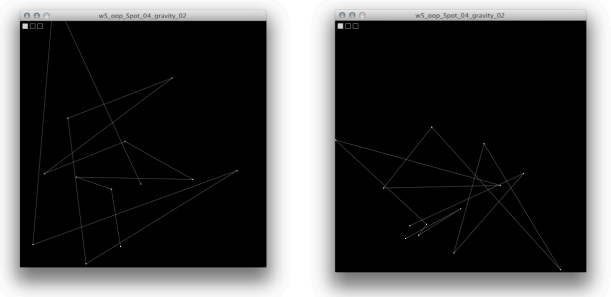
Arrays are compound data types. They extend a collection Java class.

In essence, they are collections of objects we can refer to individually.

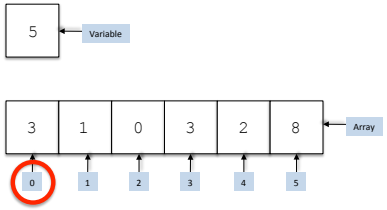
"populating" the array

Iterating through the contents of the array

Object Oriented Programming (OOP)
Using Arrays to Store Multiple Objects

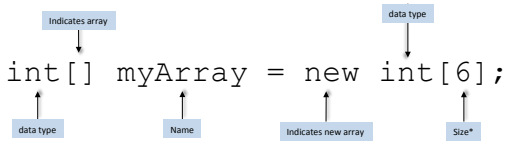


More on Arrays



Arrays are ***zero-based*** **indexed** lists of variables

More on Arrays



* must be an integer (hard-coded, variable, or expression)

Exercise 1

Declare and create arrays for the following data

1,000 floating point numbers
23 Spot objects
3 integers

Initializing Arrays

```
int [] numbers = new int[10];
----
float[] numbers = new float [5+6];
----
int num = 5;
float[] numbers = new int[num];
----
float num = 5.2;
Car[] cars = new Car [num];
----
int num = (5*6)/2;
float[] numbers = new float[num = 5];
----
int num = 5;
Spot[] spots = new Spot[num*10];
```

Initializing Arrays

```
int[] myArray = new int[3]; // array declaration

myArray[0] = 20;
myArray[1] = 12;
myArray[2] = 3;
```

arrayName[INDEX] = value

Initializing Arrays

```
int[] myArrayInt = {20, 12, 3};
float[] myArrayFloat = {1.2, 10.0, 302.12};
```

Initializing Arrays

How to create and initialize an array of 10,000 random numbers?

Initializing Arrays

How to create and initialize an array of 10,000 random numbers?

```
float[] values = {random(0, 10),
random(0,10), random(0,10),
random(0,10), random(0,10),
random(0,10), random(0,10),
random(0,10), random(0,10),
random(0,10), random(0,10),
random(0,10), etc., etc…}
```

Initializing Arrays

How to create and initialize an array of 10,000 random numbers?

```
float[] values = new float[10000];

values[0] = random(0, 10);
values[1] = random(0, 10);
values[2] = random(0, 10);
etc., etc.
…
```

Initializing Arrays

How to create and initialize an array of 10,000 random numbers?

```
float[] values = new float[10000];

values [n] = random(0, 10);
values [n+1] = random(0, 10);
values [n+2] = random(0, 10);
…
```

Initializing Arrays: Using Loops

How to create and initialize an array of 10,000 random numbers?

```
float[] values = new float[10000];

int counter = 0;
while (counter < 10000){
  values[counter] = random(0, 10);
  counter ++;
}
```

Initializing Arrays: Using Loops

How to create and initialize an array of 10,000 random numbers?

```
float[] values = new float[10000];

int counter = 0;
while (counter < 10000){
  values[counter] = random(0, 10);
  counter ++;
}
```

Creates a variable for keeping track of progress

Sets the loop and its end condition

Assigns the random value to each position in the array.

Adds to the counter. Without this line, the loop would go on forever (not good).

Initializing Arrays: Using Loops

How to create and initialize an array of 10,000 random numbers?

```
float[] values = new float[10000];

for (int i = 0; i < 10000; i++){
  values[i] = random(0, 10);
}
```

Initializing Arrays: Using Loops

How to create and initialize an array of 10,000 random numbers?

```
float[] values = new float[10000];

for (int i = 0; i < 10000; i++){
  values[i] = random(0, 10);
}
```

Sets the loop, the counter variable, the increment, and the end condition all in one line!

Assigns the value to each position in the array.

Note: Random numbers are just an example.

Initializing Arrays: Using Loops

How to create and initialize an array of 10,000 random numbers?

```
float[] values = new float[10000];

for (int i = 0; i < values.length; i++){
  values[i] = random(0, 10);
}
```

Assigns the value to each position in the array.

Note: *arrayName.length* is much better than a hard-coded number.

Operate on an Array's Members

Square each number

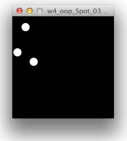
```
int[] nums = {5, 4, 2, 3, 7, 2, 8, 14};

for (_____________________________){
  ____________________;
}
```

Operate on an Array's Members

Add a random number between 1 and 10 to each number

```
int[] nums = {5, 4, 2, 3, 7, 2, 8, 14};

for (_____________________________){
  ____________________;
}
```

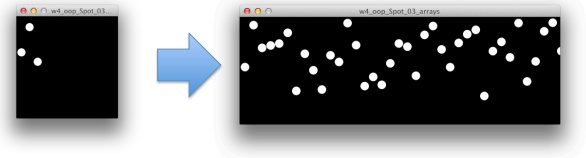
Arrays and OOP



```
class Spot{

  float x, y, diameter;
  float speed;
  int direction = 1;
  float gravity = 0.1;

  Spot(float xpos, float ypos, float dia, float sp){
    x = xpos;
    y = ypos;
    diameter = dia;
    speed = sp;
  }

  void move(){
    y += (speed*direction*0.98);
    if (( y> (height-diameter/2)) || ( y < diameter/2)){
     direction *=-1;
    }
  }

  void display(){
    ellipse(x, y, diameter, diameter);
  }

}
```

Arrays and OOP

Class definitions can be used to create an unlimited number of objects.



Remember our Spot code?

Arrays and OOP

Class definitions can be used to create an unlimited number of objects.

```
class Spot{
  float x, y, diameter;
  float speed;
  int direction = 1;
  float gravity = 0.1;
  Spot(float xpos, float ypos, float dia, float sp){
    x = xpos;
    y = ypos;
    diameter = dia;
    speed = sp;
  }
  void move(){
    y += (speed*direction*0.98);
    if (( y> (height-diameter/2)) || ( y < diameter/2)){
     direction *=-1;
    }
  }
  void display(){
    ellipse(x, y, diameter, diameter);
  }
}
```

Single Instance

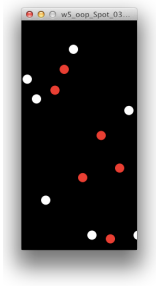
```
// declare, initialize
Spot spotA = new Spot(10, 10, 4, 1);

// activate methods
spotA.move();
spotA.display();
```

Array of Objects

```
// declare
Spot[] spots = new Spot[100];

// initialize
for (int i = 0; i<spots.length; i++){
  spots[i] = Spot(10*i, 10, 4, 1);
}

// activate methods
for (int i = 0; i<spots.length; i++){
  spots[i].move();
  spots[i].display();
}
```

Arrays and OOP

Adding Interactivity



```
class Spot{
  float x, y, diameter;
  float speed;
  int direction = 1;
  float gravity = 0.1;
  boolean mouse = false;
  Spot(float xpos, float ypos, float dia, float sp){
    x = xpos;
    y = ypos;
    diameter = dia;
    speed = sp;
  }
  void move(){
    y += (speed*direction*0.98);
    if (( y> (height-diameter/2)) || ( y < diameter/2)){
     direction *=-1;
    }
  }

 void display(){
    if (mouse){
      fill(255, 0, 0);
      ellipse(x, y, diameter, diameter);
    }
    else{
      fill(255);
      ellipse(x, y, diameter, diameter);
    }
  }
void rollover(){
    if ((mouseX > x-(diameter/2)) && (mouseX < x + (diameter/2))
&& (mouseY > y - (diameter/2)) && (mouseY < y + (diameter/2))){
      mouse = true;
    }
  }
  }
}
```

Arrays and OOP

Adding Interactivity

```
void mousePressed(){
 Spot s = new Spot(mouseX, mouseY, 5, random(0.2, 3));
 spots = (Spot[]) append(spots, s);
}
```

Append adds an element to the (last position of the) array.

Making interactive objects: making a button class

```
class Button {
 // data


 // constructor




 // methods




}
```

Making interactive objects: making a button class

In pairs: take one of the programs you wrote for assignment 4. Make it interactive. Use the button class we just wrote.

Algorithms

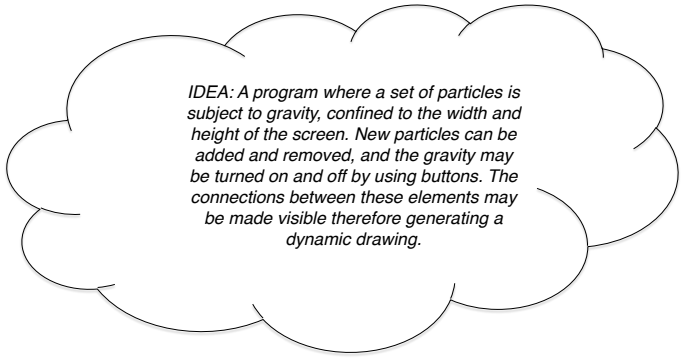
(creative) programming workflow: developing an idea, working out an algorithm, then writing the code that implements that algorithm

1.IDEA: What do you want to do?

2.ALGORITHM: Pseudo-code

3.CODE: Implementation

Algorithms



Algorithms: From Idea to Pseudocode

Setup:
- Initialize canvas
- Create array of particles
- Initialize particle objects with random directions and speeds
- Create and initialize controls (buttons)
- Start system

Draw:
- Draw background
- Check for connectedness, gravity, and other control variables
- Display particles accordingly

Objects

Particle
- Data: position, speedX, speedY, gravity
- Constructor
- Functions: Move, display, rollover

Button
-Data: position X, Y; width, height, state
-Constructor
-Functions: display, click

Algorithms: From Pseudocode to Implementation

CODE: Implementation

```
//declare collection and global variables

void setup(){

  // initialize canvas

  // initialize controls

  // Initialize particle objects with
  // random directions and speeds

}


void draw(){
 // Draw background

 // Check for connectedness, gravity, and

 // other control variables

 // Display particles accordingly

}
```

Algorithms: From Pseudocode to Implementation

```
int numSpots = 200;
int dia = 1;

Spot[] spots = new Spot[numSpots];
Button[] buttons = new Button[3];

void setup(){

  size(500, 500);
  smooth();
  stroke(200);
  noFill();
  frameRate(30);

  buttons[0] = new Button(10, 10, 10, 10);
  buttons[1] = new Button(25, 10, 10, 10);
  buttons[2] = new Button(40, 10, 10, 10);

  initiate();
}
```

```
void draw(){
  background(0);
  controls();
  for (int i = 0; i < spots.length; i++){
    spots[i].move();
    spots[i].display();
    spots[i].rollover();
  }
}
```

Algorithms: From Pseudocode to Implementation

```
class Button {

 // data
 // posx, posy, width and height, state

 // constructor

 // methods
 // display

 // click

}
```

Algorithms: From Pseudocode to Implementation

```
class Button {

 // data
 float posx = 0;
 float posy = 0;
 float w = 0;
 float h = 0;
 boolean on = false;

 // constructor
 Button(float x, float y, float tempW, float tempH){
  posx = x;
  posy = y;
  w = tempW;
  h = tempH;
  on = false;
 }

 // methods
 void display(){
   rectMode(CENTER);
   if (on){
     fill(255, 0, 0);
   }else{
     fill(255);
   }
   rect(posx, posy, w, h);
 }

 void click(int mx, int my){
    if ((mx > (posx-(w/2))) && (mx < (posx+(w/2))) && (my > (posy-(h/2))) && (my < (posy+(h/2)))){
       on = !on;
    }
  }
}
```

Algorithms: From Pseudocode to Implementation

CODE: Implementation

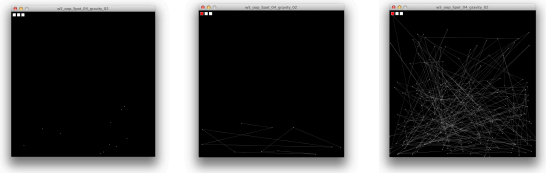
```
class Spot{

  // data
  // posx, posy, diameter, speedX, speedY

 // constructor

 // methods

 //   move

 //   rollover

 //   display
}
```

Algorithms: From Pseudocode to Implementation

```
class Spot{

  float x, y, diameter;
  float spX, spY;
  float gravity = 0.3;
  boolean halo = false;
  boolean mouse = false;

  Spot(float xpos, float ypos, float dia, float speedX, float speedY){
    x = xpos;
    y = ypos;
    diameter = dia;
    spX = speedX;
    spY = speedY;
  }

  void move(){
    x += spX;
    y += spY;
    x = constrain(x, (diameter/2)-2, width-(diameter/2)+2);
    y = constrain(y, -1000, height-(diameter/2));

    if (y < (height-(diameter/2))){
      spY += gravity;
    }
    if (y >= (height-(diameter/2))){
      spY *= -0.95;
    }
    if ((x > (width-(diameter/2)) || (x < (diameter/2)))){
      spX *= -0.95;
    } else{
    }
  }
```

Algorithms: From Pseudocode to Implementation

```
void rollover(){
    if ((mouseX > x-(diameter*5)) && (mouseX < x +
(diameter*5)) && (mouseY > y - (diameter*5)) &&
(mouseY < y + (diameter*5))){
      //println("rollover");
      spX = 0;
      spY = 0;
      gravity = 0;
      mouse = true;
    }
  }

  void display(){
    if (halo){
      strokeWeight(0.5);
      noFill();
      ellipse(x, y, diameter*10, diameter*10);
    } else if (!halo) {
        strokeWeight(1);
        ellipse(x, y, diameter, diameter);
    }
  }
}
```



Debugging

-Read the error messages in the console (*if* there are error messages)
-Clean/simplify your code.
-Use comments ("//") to isolate different parts of the code.
-Print messages with println();. The more specific the better. See concatenation.

Concatenation example:

```
println("x: " + x + ", y: " + y);
```

-Also, use println() statements to see if particular parts of the code are being used. For instance, conditionals, functions or loops. For example:

```
for (int i = 0; i < 10;i++) {
  println("Inside the loop, iteration: " + i);
}
```

Libraries

- *So far we've been using the Processing .core library.*
- *There are many libraries (additional code packages) that enhance Processing's functionality.*
- *Sound, networking, serial communication, GUIs, physics simulations, etc.*

-*Built-in example:* ***DXF Export***
- *Contributed libraries example:* ***OBJLoader***, ***controlP5***

Importing 3-D geometry using OBJ loader library

```
import saito.objloader.*;

OBJModel model;

float rotX, rotY;

void setup()
{
    size(800, 600, P3D);
    frameRate(30);

    model = new OBJModel(this, "prism+sphere.obj", "absolute", POLYGON);
    model.enableDebug();

    model.scale(20);
    model.translateToCenter();

    stroke(255);
    noStroke();
}
```

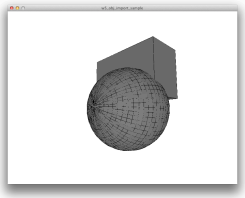
this points to the "data" folder in the sketch.

Importing 3-D geometry using OBJ loader library

```
void draw()
{
    background(255);
    lights();
    pushMatrix();
    translate(width/2, height/2, 0);
    rotateX(rotY);
    rotateY(rotX);
    model.draw();
    popMatrix();
}
```

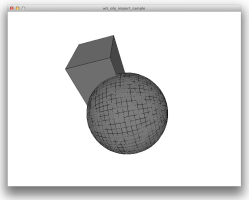
Creates a new transformation matrix for the current view

'Sheds' the transformation matrix

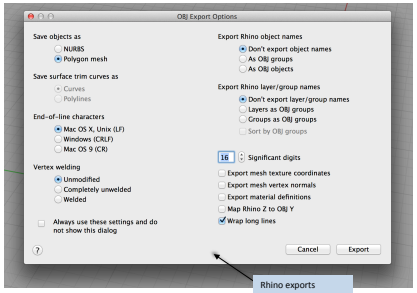


```
void mouseDragged()
{
    rotX += (mouseX - pmouseX) * 0.01;
    rotY -= (mouseY - pmouseY) * 0.01;
}
```

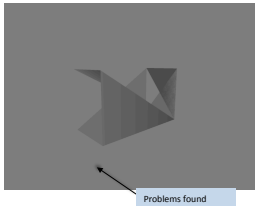
Camera (world) rotations



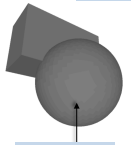
Importing 3-D geometry using OBJ loader library



Rhino exports settings



Problems found when exporting as NURBS



Successful import

Review

Arrays in depth: iteration and OOP

Code structure: from idea to algorithm, and then to implementation (code)

Debugging: debug, don't panic.

Libraries: built-in and contributed.

Review

Functions: modularity, abstraction

- Creating our own functions
- Return types
- Parametric forms

Object Oriented Programming: another kind of modularity. Functionality grouped around entity types.

-Classes
  -Data
  -Constructor
  -Functionality

-Objects
  - Initialization
  - Activation

-Arrays: collections of objects