

# *Arrays*



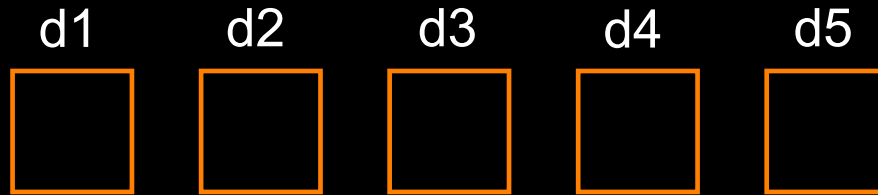
# Objectives

- After reading, you should be able to:
  - Understand how arrays are structured.
  - Create and initialize arrays.
  - Understand array indexing and bound checking
  - Use `for` loops to iterate over array elements.
  - Understand some basic examples of array processing:
    - Computing the sum of all elements
    - Finding the max or min element
    - **Shifting** elements to the right or left.
  - Use arrays in Processing to solve problems.

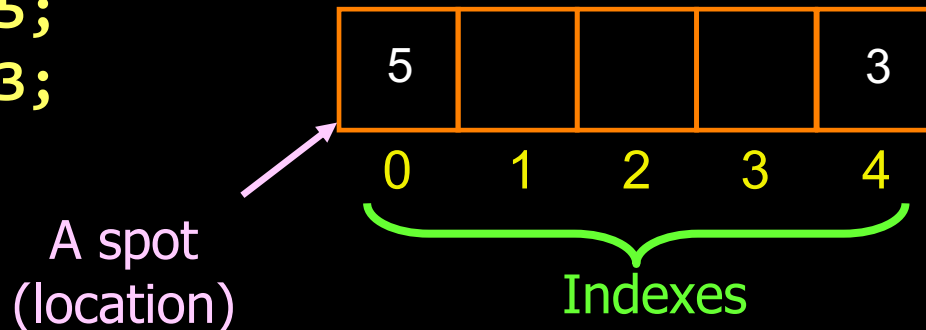


# Arrays Overview

- Suppose you need many variables in your program.
- You could either create a separate name for each variable:  
`double d1, d2, d3, d4, d5;`



- Or you could create an array that has multiple spots (indexes):  
`double[] myArray = new double[5];`  
`myArray[0] = 5;`  
`myArray[4] = 3;`



# Arrays

- An **array** is a collection of data items of the same type.
- An array reference is denoted using the open and close square brackets “ [ ] ” during declaration.
  - You can have an array of any data type including the base types (int, double) and object-types (Ball).

- Examples:

```
int[] integers;  
float[] numbers;  
Ball[] balls;
```

- Similar to an object, when you declare an array you are creating a **reference** to an array. Until you actually create the space for the array using **new**, no array exists in memory.  
**float[] numbers = new float[10];**

# Array Indexing

- When creating an array using `new`, the number in square brackets is the number of elements in the array:  
`float[] numbers = new float[20];` // 20 items
- Note the **first element** of the array has **index 0** instead of 1.
  - In the previous example, the first index is 0 and the last is 19.
- When an array is created, its values are **initialized to defaults**:
  - 0 for numbers, **false** for boolean, **null** for object references
- To access or set a value in an array, use its subscript:  
`numbers[0] = 10;` // Sets first element to 10  
`numbers[19] = numbers[0];` // Sets last element same as 1st

# Array Details

- To get the length of an array in your program:

```
float[] numbers = new float[25];  
float size = numbers.length; // Returns 25
```
- You can initialize an array with values when you first declare it:

```
int[] primes = {2, 3, 5, 7, 11};
```

  - **new** is not used with an initializer list. Initializers can only be used during declaration.
- Processing (Java) performs automatic **bound checking** whenever an array element is referenced.
  - If index is in the valid range, the reference is carried out.
  - If index is not valid, an error called, `ArrayIndexOutOfBoundsException`, is displayed.

# Practice Questions

- 1) Create an `int` array with name `myArray` with 20 elements.
  - Set the value of the 1<sup>st</sup> element to 10.
  - Set the value of the last element to 1.
  
- 2) Create an array that has 10 elements. Put the numbers from 1 to 10 in the array.
  
- 3) How do you know how many elements are in an array?

# Solution

1) `int[] myArray = new int[20];`

`myArray[0] = 10;`

`myArray[19] = 1;`

OR

`myArray[myArray.length-1] = 1;`

2) `int[] arr = new int[10];`  
`for (int i=0; i < 10; i++)`  
`arr[i] = i+1;`

3) You can use the `.length` property of the Array object.

# Working with Array Elements

- **1) Using a loop:** it is preferred to use a loop to process array elements when applying *the same action* to several elements.

```
for (int i=0; i<arr.length; i++){  
    //same actions applied to all elements  
}
```

- **Example:** this code Initializes an array ar with random values :

```
for (int i = 0; i < ar.length; i++)  
    ar[i] = random(100);
```

- **2) Without a for loop:** if you are applying different actions to the array elements, you should probably avoid using a for loop.

# Working with Array Elements: Examples

- **Initializing** an array with random values:

```
for(int i = 0; i < arr.length; i++)  
    arr[i] = random(100);
```

- **Summing** all elements

```
int sum = 0;  
for(int i = 0; i < arr.length; i++)  
    sum += arr[i];
```

- **Finding** the largest element

```
float max = arr[0];  
for (int i = 1; i < arr.length; i++)  
    if (arr[i] > max)  
        max = arr[i];
```

# Working with Array Elements: Examples

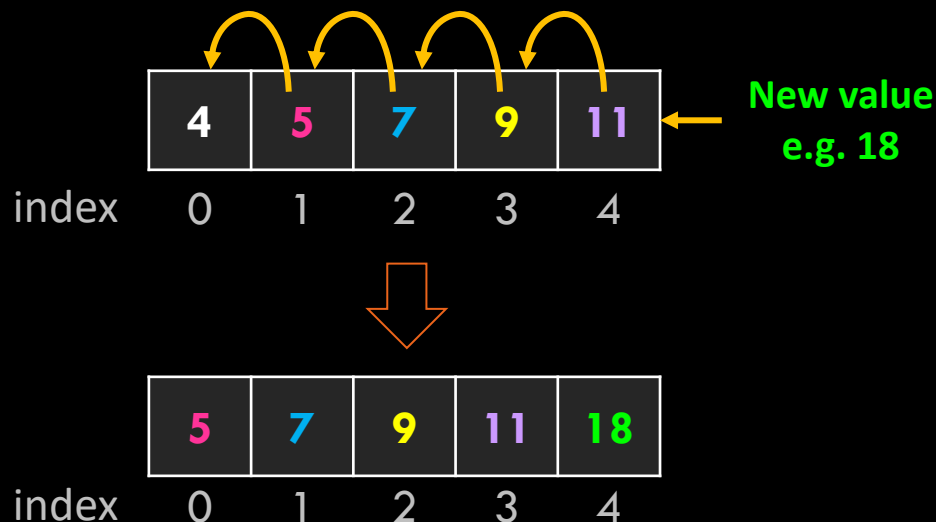
- **Shifting left:** shifting the elements one position to the left:

```
// Shift elements left
```

```
for (int i = 0; i < arr.length-1; i++)  
    arr[i] = arr[i+1];
```

```
// insert a new elements at the right
```

```
arr[arr.length-1] = 18;           //18 is any new value
```



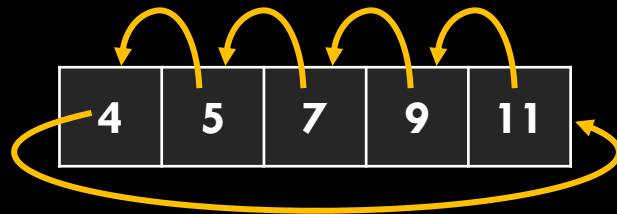
# Working with Array Elements: Examples

- **Rotating:** Shifting the elements one position to the left *and* filling the last element with the first element:

```
// Retain the first element
float temp = arr[0];

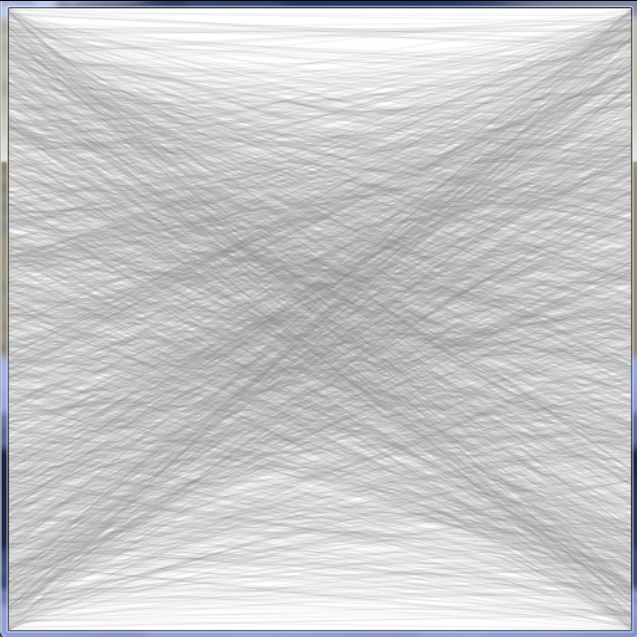
// Shift elements left
for (int i = 0; i < arr.length-1; i++)
    arr[i] = arr[i+1];

// Move the first element to fill in the last position
arr[arr.length - 1] = temp;
```



# Array of Numbers

- This example uses arrays to display lines randomly drawn between the side edges of a window.



```
size(500, 500);  
background(255);  
stroke(128, 30);  
int N = 2000;
```

```
// 1) Create two arrays: y1 and y2
```

```
float[] y1 = new float[N];  
float[] y2 = new float[N];
```

```
// 2) Initialize arrays with random y values
```

```
for (int i = 0; i < N; i++) {  
    y1[i] = random(0, height);  
    y2[i] = random(0, height);  
}
```

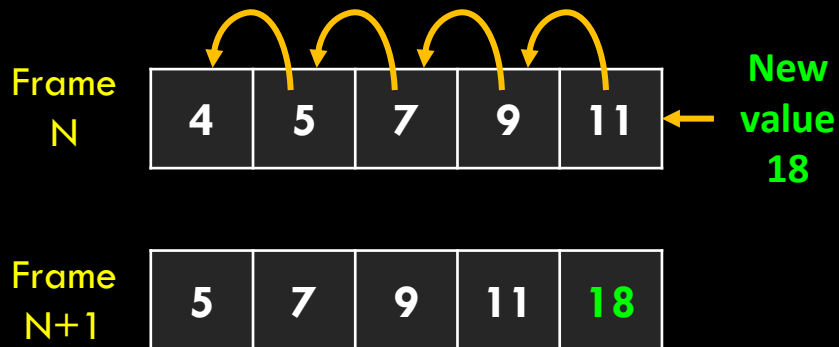
```
// 3) Draw lines based on array values
```

```
for (int i = 0; i < N; i++)  
    line(0, y1[i], width, y2[i]);
```

## The Snake!

source: textbook

- A snake is an **array** of 50 ellipses centered at 50 locations (x,y) saved in two arrays: x and y.
- Every time the mouse moves, two things happen:
  - 1) all elements in the snake array are shifted to the left.
  - 2) the new position of the mouse is inserted at the last array element.
- All ellipses identified by the array are then drawn.



```
int[] x = new int[50];      //initialized to 0's
int[] y = new int[50];      //initialized to 0's

void setup() {
  size(200, 200);  noStroke(); noCursor();
}

void draw() {
  background(255);
  // Shift array values to left
  for (int i=0; i<x.length-1;i++){
    x[i] = x[i+1];
    y[i] = y[i+1];
  }
  // New mouse location in last spot
  x[x.length-1] = mouseX;
  y[y.length-1] = mouseY;
  // Draw everything
  for (int i=0;i<x.length;i++){
    fill(255-i*5);
    ellipse(x[i], y[i], i, i);
  }
}
```



# Overview

The notes covered the following:

- Arrays structure, array indexing, and bound checking.
- Creating and initializing arrays.
- Using `for` loops to process array elements.
- Basic examples :
  - Computing the sum of all elements
  - Finding the max or min element
  - Shifting elements to the right or left.
- Two Processing sketches:
  - Many random lines (initializing arrays with random values)
  - Snake that follows the mouse (shifting)

# Arrays

**Question:** What is the size of this array?

```
int[] myArray = new int[10];
```

- A. error
- B. 10
- C. 9
- D. 11
- E. None of the above

## Arrays

**Question:** What are the contents of this array?

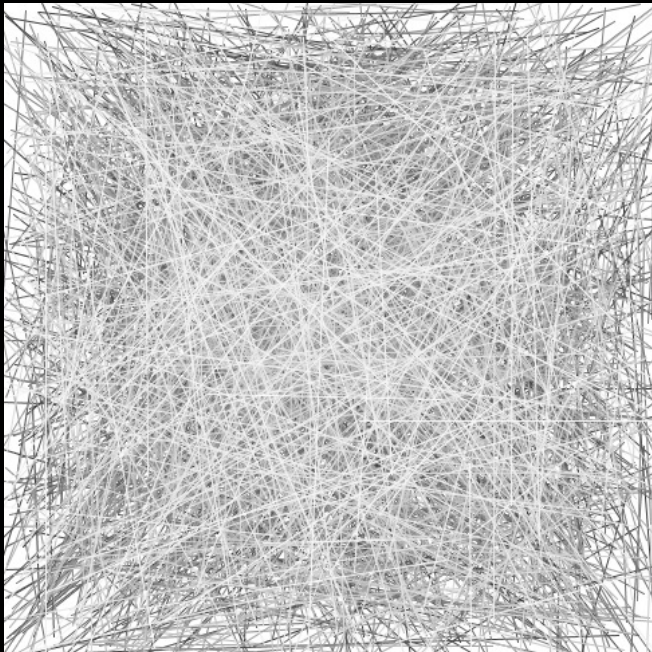
```
int[] myArray = new int[4];  
myArray[3] = 1;  
myArray[2] = 2;  
myArray[1] = 3;  
myArray[0] = 4;
```

- A. error
- B. 0, 1, 2, 3
- C. 1, 2, 3, 4
- D. 4, 3, 2, 1
- E. None of the above

# The Haystack!

This program is similar to the other example except:

- A)** it uses random values for both x and y coordinates for each line end.
- B)** the line color starts dark and gets lighter as we draw more lines.



```
size(500, 500); background(255);

int N = 2000;

// create arrays
float[] x1 = new float[N];
float[] x2 = new float[N];
float[] y1 = new float[N];
float[] y2 = new float[N];

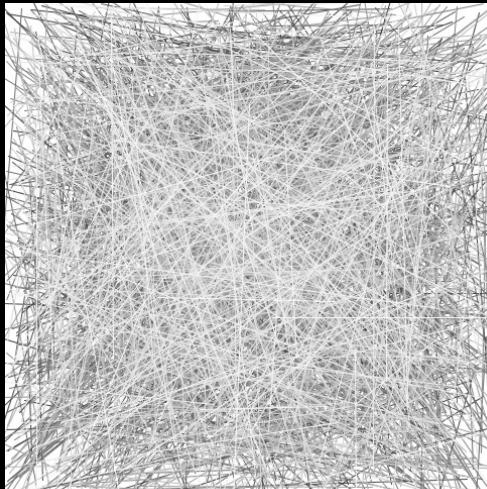
// initialize arrays
for (int i = 0; i<N; i++) {
  x1[i] = random(0, width);
  x2[i] = random(0, width);
  y1[i] = random(0, height);
  y2[i] = random(0, height);
}

// draw many lines
for (int i = 0; i<N; i++) {
  stroke(map(i,0,N,50,250));
  line(x1[i], y1[i], x2[i], y2[i]);
}
```

# Discussion

**Q1.** Are all lines' ends located on the edges (i.e. each line is extending from one edge to another)?

**Q2.** Can this sketch be produced without using arrays? If Yes, why would we need arrays then?



```
size(500, 500); background(255);

int N = 2000;

// create arrays
float[] x1 = new float[N];
float[] x2 = new float[N];
float[] y1 = new float[N];
float[] y2 = new float[N];

// initialize arrays
for (int i = 0; i<N; i++) {
  x1[i] = random(0, width);
  x2[i] = random(0, width);
  y1[i] = random(0, height);
  y2[i] = random(0, height);
}

// draw many lines
for (int i = 0; i<N; i++) {
  stroke(map(i,0,N,50,250));
  line(x1[i], y1[i], x2[i], y2[i]);
}
```

# *Objects with Arrays as Attributes*

## The Snake Class

- Let's repeat "The Snake" exercise using OOP.
  - A Snake is represented by a class that has:
    - x and y arrays as attributes
    - len attribute representing the # of ellipses (i.e. the length)
    - a one-arg constructor that initializes len and creates the arrays.
    - two functions:
      - **update()** that: a) shift-left the arrays, and (b) insert a mouse location in last element.



- **display()** to draw the snake.

- In **setup()**: we create one snake object
- In **draw()**: we call **update()** and **display()** of the snake object.



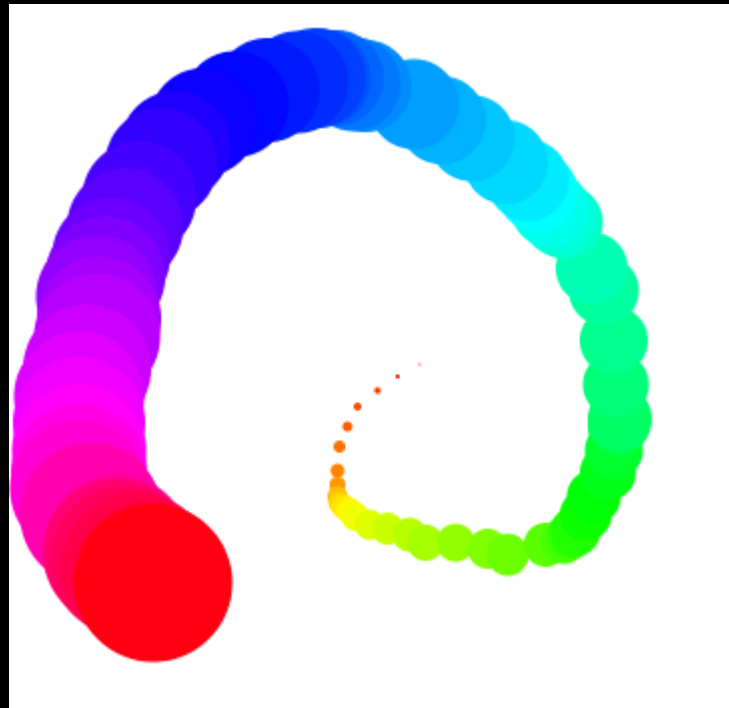
# The Snake Class – the code

```
Snake s;  
void setup() {  
  size(400, 400);  
  noStroke();  
  s = new Snake(80);  
}  
void draw() {  
  background(255);  
  s.update();  
  s.display();  
}
```

```
class Snake {  
  int[] x, y; // declare arrays as attributes  
  int len;  
  Snake(int L) {  
    len = L;  
    x = new int[len]; // create arrays in constructor  
    y = new int[len];  
  }  
  void update() {  
    // Shift array values to left  
    for (int i=0; i<x.length-1; i++) {  
      x[i] = x[i+1];  
      y[i] = y[i+1];  
    }  
    // New mouse location in last spot  
    x[x.length-1] = mouseX;  
    y[y.length-1] = mouseY;  
  }  
  void display() {  
    for (int i=0; i<x.length; i++) {  
      fill(255-i*250/len);  
      ellipse(x[i], y[i], i, i);  
    }  
  }  
}
```

# *Can you produce the output below?*

- i.e. the snake should be colorful instead of B/W



## The Haystack Class

```
Haystack hay1, hay2;
void setup(){
    size(500, 500);
    hay1 = new Haystack(400,100,100);
    hay2 = new Haystack(400,300,300);
}
void draw(){
    background(255);
    hay1.display(); hay1.wiggle(2);
    hay2.display();
}
```

```
class Haystack{
    int N; float x,y; //center
    float[] x1, y1, x2, y2;
    Haystack(int num, float xc, float yc){
        N = num; x = xc; y = yc;
        y1 = new float[N];
        y2 = new float[N];
        x1 = new float[N];
        x2 = new float[N];
        for (int i = 0; i<N; i++) {
            y1[i] = random(90, height-90);
            y2[i] = random(90, height-90);
            x1[i] = random(90, width-90);
            x2[i] = random(90, width-90);
        }
    }
    void wiggle(int sp){
        for (int i = 0; i<N; i++) {
            y1[i] += random(-sp, sp);
            y2[i] += random(-sp, sp);
            x1[i] += random(-sp, sp);
            x2[i] += random(-sp, sp);
        }
    }
    void display(){
        pushMatrix(); translate(width/2-x,height/2-y);
        for (int i = 0; i<N; i++) {
            stroke(map(i,0,N,50,250));
            line(x1[i], y1[i], x2[i], y2[i]);
        }
        popMatrix();
    }
    // add more methods here to move, scale, etc.
}
```

# *Arrays of Objects*

# Arrays of Primitives

- An array of primitive types stores the values in the slots in the array.

```
int[] numbers = new int[5];  
numbers[0] = 10;  
numbers[1] = 40;  
numbers[2] = numbers[0]+10;
```

numbers



|    |
|----|
| 10 |
| 40 |
| 20 |
| 0  |
| 0  |

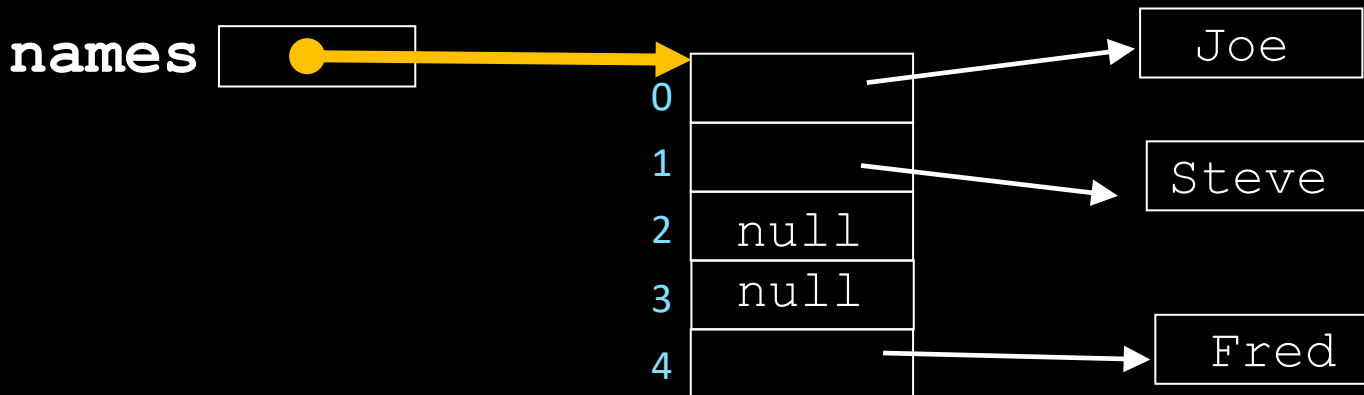
array indices

0  
1  
2  
3  
4

# Arrays of Objects

- An object array is an array of object references. Example:

```
String[] names = new String[5];  
names[0] = "Joe";  
names[1] = "Steve";  
names[4] = "Fred";
```



- When allocating an object array, Processing does not create the objects for you. That is, each object reference is initialized to null.

# Arrays of Objects, cont'd

To create an array of objects, you need to follow two steps:

## 1) Declare the array.

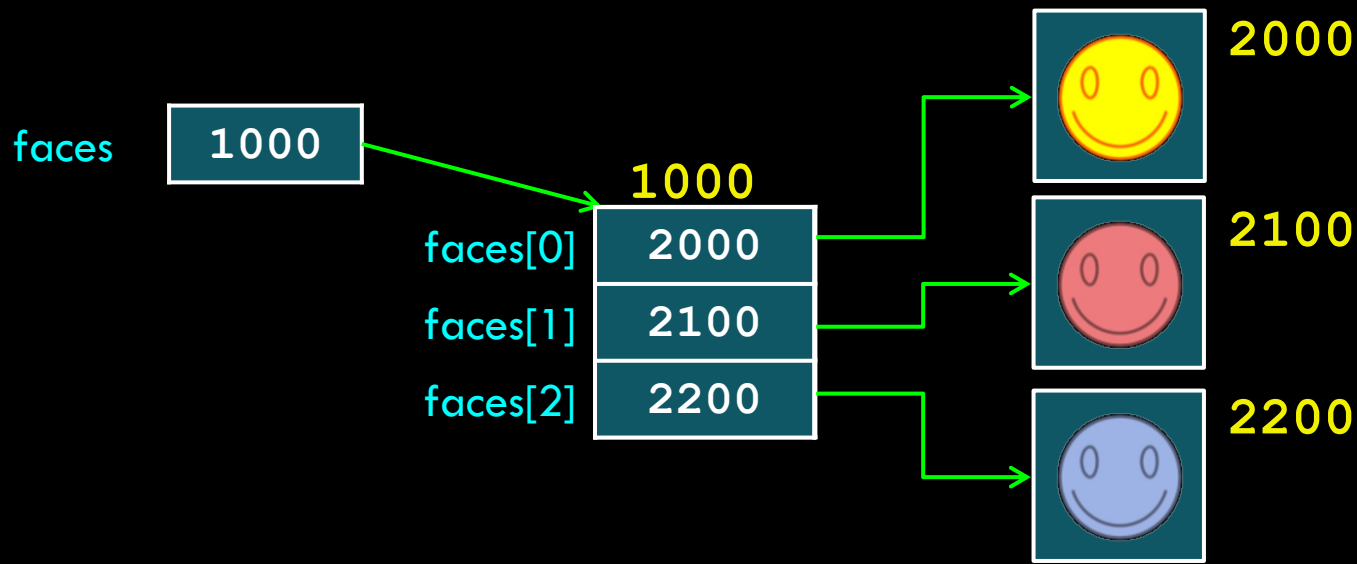
```
HappyFace[] faces = new HappyFace[3];
```

- This array doesn't have any objects yet (do you see any object constructor here?). This array currently holds **reference variables**.

## 2) Create an object for each reference variable in the array

- To create the objects, you may use a for loop like this one:

```
for(int i=0; i<faces.length; i++)  
    faces[i] = new HappyFace();
```



# Arrays of Objects, cont'd

- You may then invoke any method of the Circle objects using a syntax similar to this:

```
faces[1].display();
```

, which involves two levels of referencing:

- **faces** references to the entire array, and
- **faces[1]** references to the second HappyFace object.

- To run a function for all objects in the array, use a for loop :

```
for(int i=0; i<faces.length; i++)  
    faces[i].display();
```

The background of the slide is a solid dark blue. It is populated with numerous small, light blue smiley faces (:) scattered across the surface. Some of these faces are slightly larger and more prominent than others, giving a sense of depth or movement, as if they are floating or bouncing. The smiley faces are simple, with two dots for eyes and a curved line for a mouth.

# ***Project: Bouncing Happy Faces***

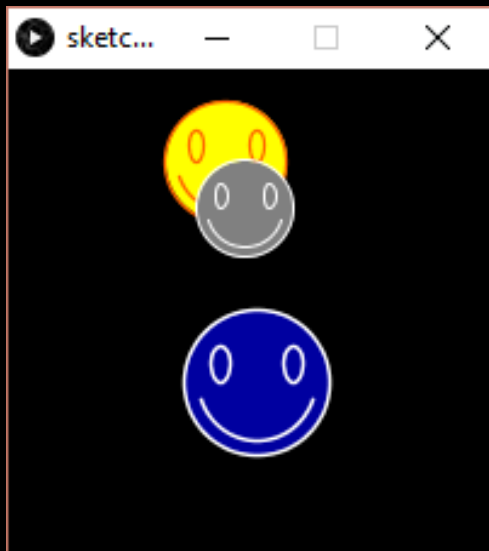
# Bouncing Happy-Faces

- Let's say we want to create many bouncing faces.
- The basic idea is to:
  - Create an array of many face objects.
  - Run for loops to move and display all faces in the array.
- To do this, we will have to create the HappyFace class first.
  - We will reuse the code from last lecture



# Bouncing Happy-Faces

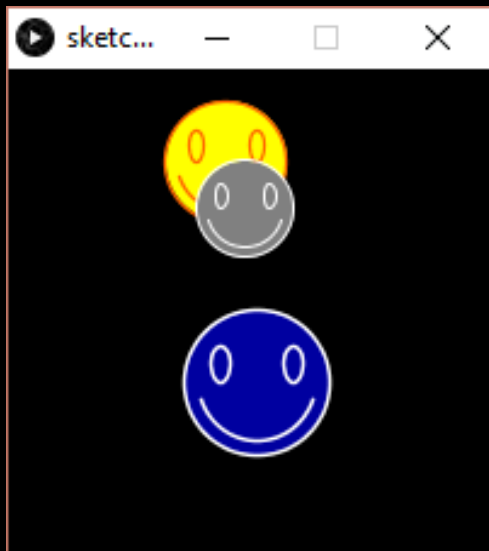
- Let's reuse the code from last lecture: we wrote code for creating three bouncing faces
- The HappyFace class is on the right.



```
class HappyFace {
  float x,y,r,speedX,speedY;
  color fillColor,outlineColor;
  HappyFace() {
    r=random(20,40);x=random(r,width-r);y=random(r,height-r);
    speedX=random(-3,3); speedY=random(-3,3);
    fillColor=color(255,255,0);
    outlineColor=color(255,100,0);
  }
  HappyFace(float a,float b,float c,float sx,float sy,color c1,color c2){
    x=a; y=b; r=c;
    speedX=sx; speedY=sy;
    fillColor=c1; outlineColor=c2;
  }
  void moveBounce() {
    x+=speedX; y+=speedY;
    if (x>width-r || x<r) speedX= -speedX;
    if (y>height-r || y<r) speedY= -speedY;
  }
  void display() {
    fill(fillColor);
    stroke(outlineColor); strokeWeight(r/20);
    ellipse(x, y, 2*r, 2*r);           //face
    arc(x,y,1.6*r,1.6*r,.1*PI,.9*PI); //mouth
    ellipse(x+r/2, y-r/4, r/4, r/2);  //right eye
    ellipse(x-r/2, y-r/4, r/4, r/2);  //left eye
  }
}
```

# Bouncing Happy-Faces

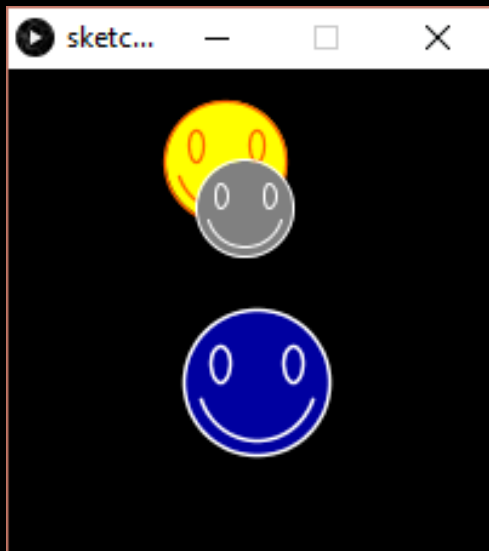
- This code shows how to create three HappyFace objects then move and display them



```
HappyFace f1, f2, f3;
void setup(){
  size(200,200);
  f1=new HappyFace();
  f2=new HappyFace(100,100,20,-1.2,1.4,color(128),color(255));
  f3=new HappyFace(150,150,30,1.4,1.3,color(0,0,160),color(255));
}
void draw(){
  background(0);
  f1.moveBounce(); f1.display();
  f2.moveBounce(); f2.display();
  f3.moveBounce(); f3.display();
}
```

# Bouncing Happy-Faces *using arrays*

- This is the same code with an array of 3 faces



```
HappyFace[] faces = new HappyFace[3];
void setup(){
  size(200,200);
  faces[0] = new HappyFace();
  faces[1] = new HappyFace(100,100,20,-1.2,1.4,color(128),color(255));
  faces[2] = new HappyFace(150,150,30,1.4,1.3,color(0,0,160),color(255));
}
void draw(){
  background(0);
  for(int i = 0; i<3; i++){
    faces[i].moveBounce(); faces[i].display();
  }
}
```

# Bouncing Happy-Faces *using arrays*

- This is the same code except we create random face objects in a loop
- We use a variable  $N = 3$  to set the number of faces



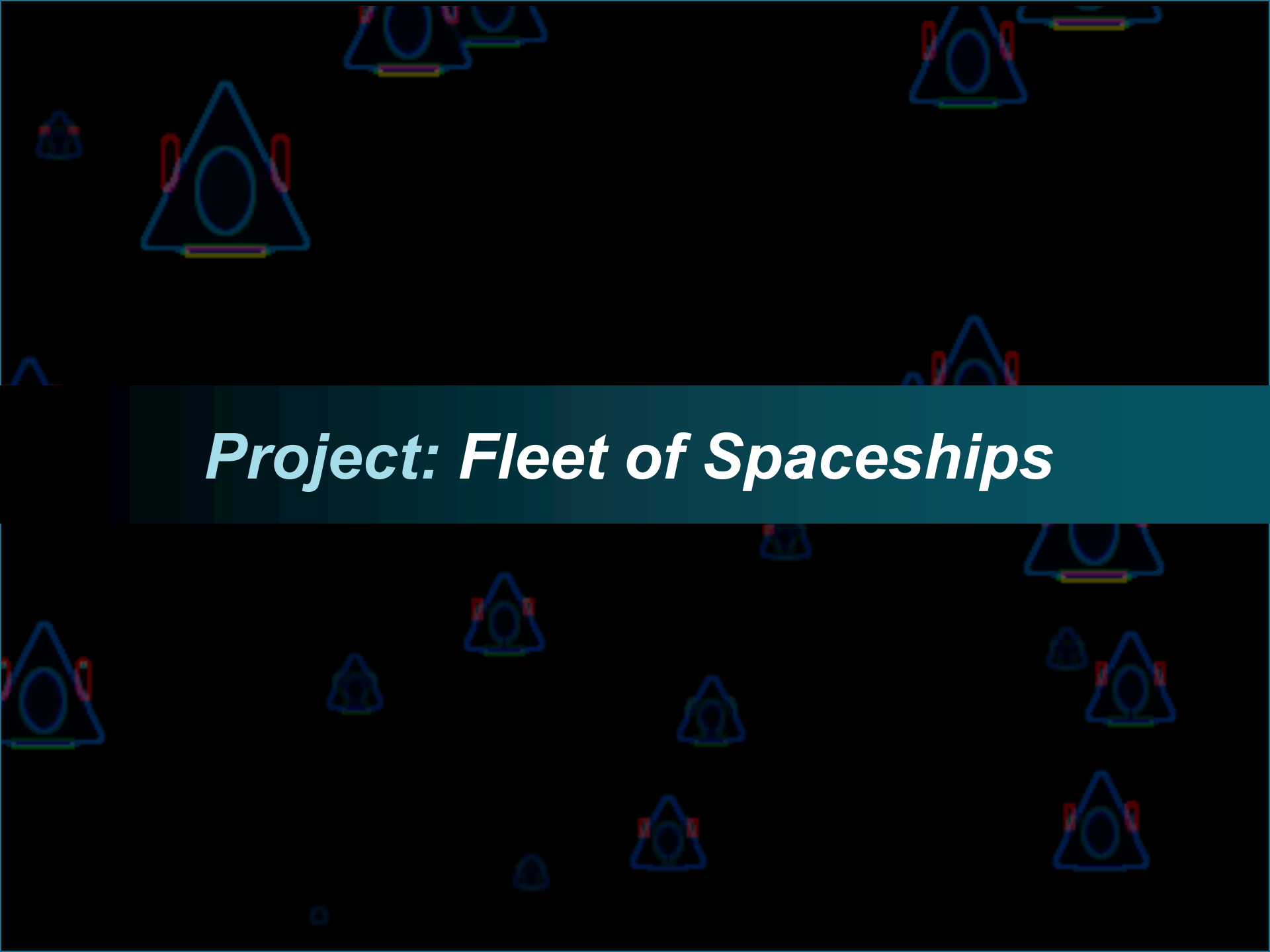
```
int N = 3;
HappyFace[] faces = new HappyFace[N];
void setup(){
  size(200,200);
  for(int i = 0; i<N; i++){
    faces[i] = new HappyFace();
  }
void draw(){
  background(0);
  for(int i = 0; i<N; i++){
    faces[i].moveBounce();
    faces[i].display();
  }
}
```

# Bouncing Happy-Faces *using arrays*

- Now lets set N to larger number.
- ENJOY!



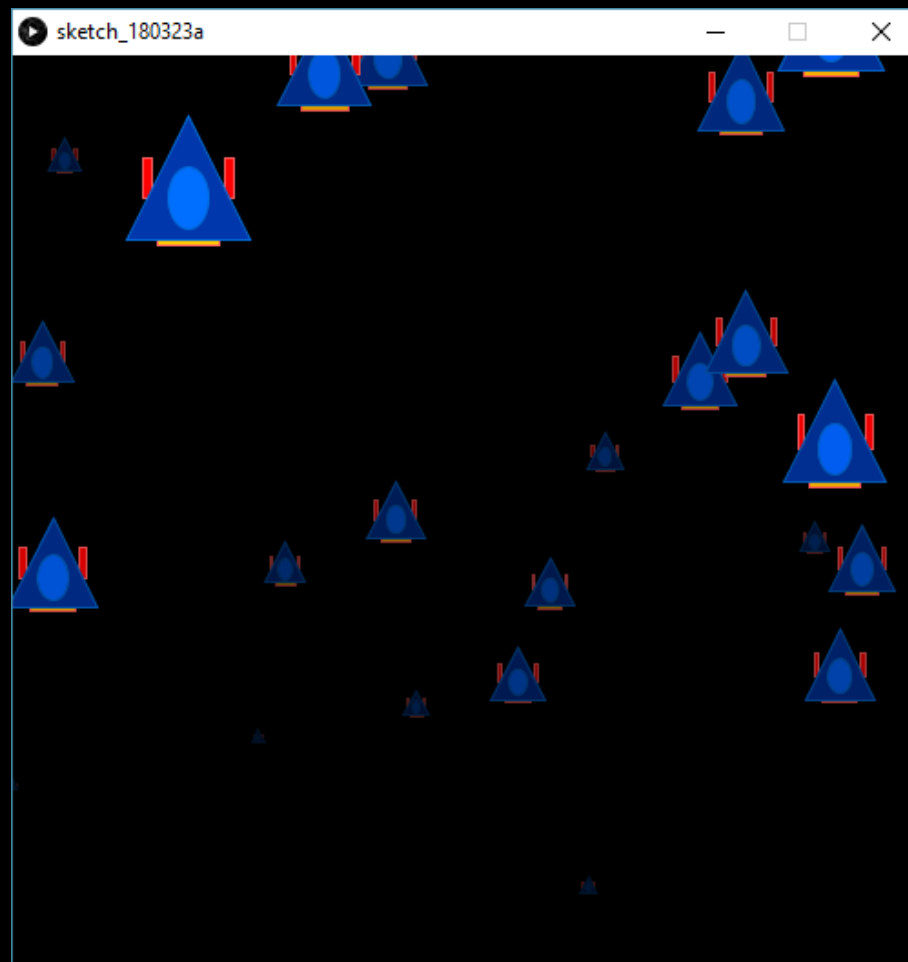
```
int N = 15;
HappyFace[] faces = new HappyFace[N];
void setup(){
    size(200,200);
    for(int i = 0; i<N; i++){
        faces[i] = new HappyFace();
    }
    void draw(){
        background(0);
        for(int i = 0; i<N; i++){
            faces[i].moveBounce();
            faces[i].display();
        }
    }
}
```

The background of the slide is dark blue with a repeating pattern of stylized, light blue spaceship icons. Each icon consists of a triangle with a circle inside, and small rectangular details on the sides and bottom. The icons are scattered across the entire slide, with a higher density in the top and bottom sections.

# ***Project: Fleet of Spaceships***

# Spaceship Fleet

- Let's create a fleet of spaceships

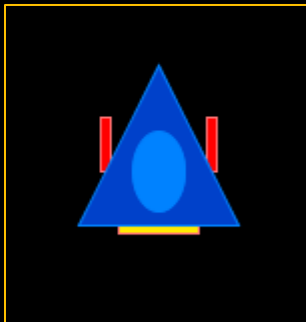


# One Spaceship

```
Spaceship ship;
void setup(){
  size(500,500);

  ship = new Spaceship();
  ship.speed = 2.5;
}
void draw(){
  background(0);
  //move & display one spaceship

  ship.move();
  ship.display();
}
```

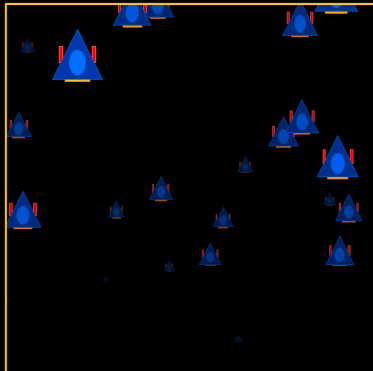


```
class Spaceship {
  float x, y, speed;
  Spaceship() { x=random(width); y=random(height); }
  void move(){
    y -= speed;
    //wrap spaceships once outside sketch
    if(y<-100)
      {x=random(width); y=random(height,height+300);}
  }
  void display() {
    // larger ships are closer, faster, and brighter
    float scale = speed*32;
    float tint = map(scale,1,60,0.1,1);
    // draw side guns
    rectMode(CENTER); fill(255*tint,0,0);
    strokeWeight(1);stroke(255*tint,90*tint,90*tint);
    rect(x-scale/3,y+scale/2,scale/15,scale/3);
    rect(x+scale/3,y+scale/2,scale/15,scale/3);
    // draw jet engine
    fill(255*tint, 180*tint, 0);
    rect(x, y+scale, scale/2, scale/10);
    // draw main body
    stroke(0,100*tint,200*tint);
    fill(0, 50*tint, 155*tint);
    triangle(x,y,x+scale/2,y+scale,x-scale/2,y+scale);
    fill(0, 100*tint, 255*tint);
    ellipse(x, y+2*scale/3, scale/3, scale/2);
    rectMode(CORNER);
  }
}
```

## Example , cont'd

# Fleet of Spaceships

```
int N = 30;
Spaceship[] ships=new Spaceship[N];
void setup(){
    size(500,500);
    for(int i = 0; i<N; i++){
        ships[i] = new Spaceship();
        ships[i].speed =(i+1)*0.07;
    }
}
void draw(){
    background(0);
    //move & display ALL spaceships
    for(int i = 0; i<N; i++){
        ships[i].move();
        ships[i].display();
    }
}
```



```
class Spaceship {
    float x, y, speed;
    Spaceship() { speed=random(1,2); x=random(width);
        y=random(height); }
    void move(){
        y -= speed;
        //wrap spaceships once outside sketch
        if(y<-100)
            {x=random(width); y=random(height,height+300);}
    }
    void display() {
        // larger ships are closer, faster, and brighter
        float scale = speed*32;
        float tint=map(scale,1,60,0.1,1);
        // draw side guns
        rectMode(CENTER); fill(255*tint,0,0);
        strokeWeight(1);stroke(255*tint,90*tint,90*tint);
        rect(x-scale/3,y+scale/2,scale/15,scale/3);
        rect(x+scale/3,y+scale/2,scale/15,scale/3);
        // draw jet engine
        fill(255*tint, 180*tint, 0);
        rect(x, y+scale, scale/2, scale/10);
        // draw main body
        stroke(0,100*tint,200*tint);
        fill(0, 50*tint, 155*tint);
        triangle(x,y,x+scale/2,y+scale,x-scale/2,y+scale);
        fill(0, 100*tint, 255*tint);
        ellipse(x, y+2*scale/3, scale/3, scale/2);
        rectMode(CORNER);
    }
}
```

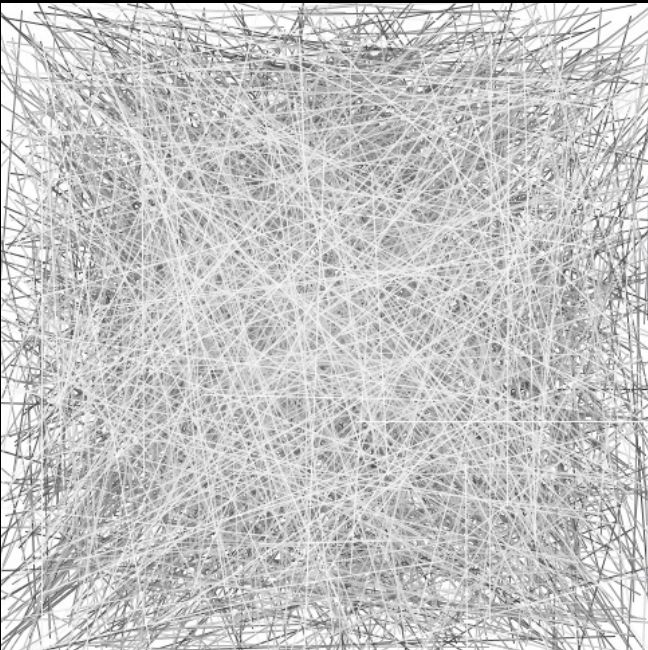


# ***Project: Wiggly Haystack***

## Example

# The OO Haystack!

This is the same program as before except that we use an array of line objects.



```
void setup(){
  size(500, 500);
  background(255);
  int N = 500;

  // create array of object references
  MyLine[] lines = new MyLine[N];

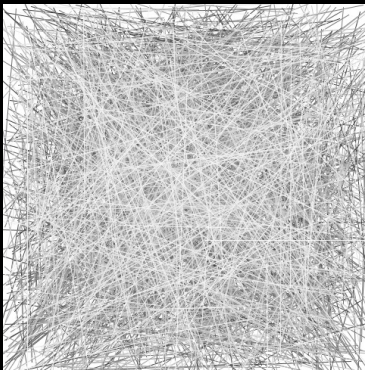
  // create the objects in the array
  for (int i = 0; i<N; i++) {
    lines[i] = new MyLine();
  }
  // display all MyLine objects
  for (int i = 0; i<N; i++) {
    stroke(map(i,0,N,50,250));
    lines[i].display();
  }
}
```

```
class MyLine{
  float x1, y1, x2, y2;
  MyLine(){
    x1=random(0,width); y1=random(0,height);
    x2=random(0,width); y2=random(0,height);
  }
  void display(){ line(x1,y1,x2,y2); }
}
```

# OO Haystack in Dynamic Mode

This is the same program as before but we create an array of objects which are the lines

- The array is created in global scope so that it can be accessed in both setup and draw functions.
- The objects in the array are created in the setup (we need to do this only once)
- Every frame, we draw all lines in the array.



```
int N = 500;
// create array in global scope
MyLine[] lines = new MyLine[N];

void setup(){
  size(500, 500);
  //create the objects in the array
  for (int i = 0; i<N; i++) {
    lines[i] = new MyLine();
  }
}

void draw(){
  background(255);
  // display all MyLine objects every frame
  for (int i = 0; i<N; i++) {
    stroke(map(i,0,N,50,250));
    lines[i].display();
  }
}

class MyLine{ /* same as before */ }
```

# Wiggly Haystack!

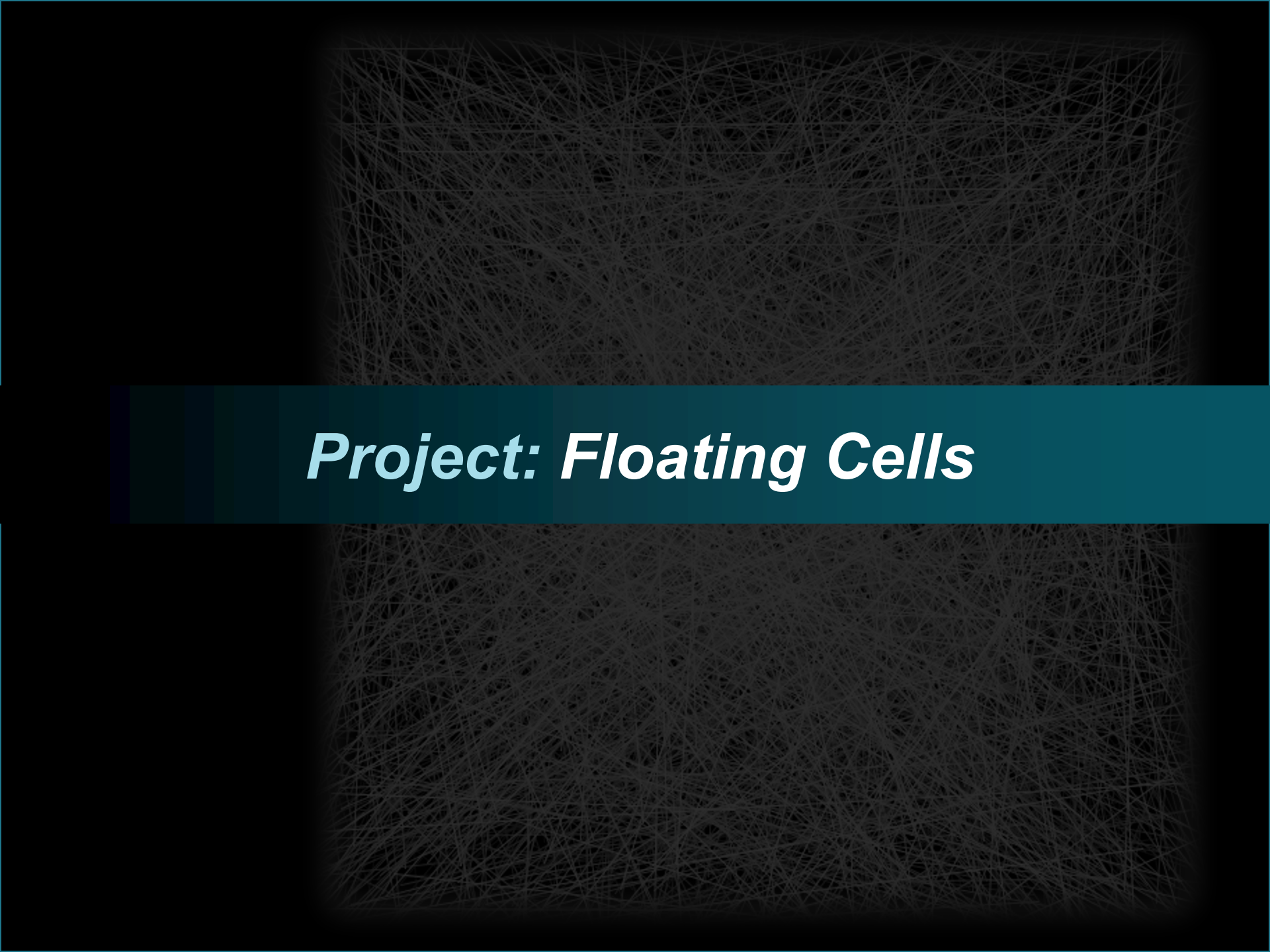
- This is the same program as before but since we added one more action (`wiggle()` function) to `MyLine` class that slightly changes the position of a line object.
- We then call the `wiggle()` function for every line in every frame before we display the line.

```
int N = 2000;
// create array in global scope
MyLine[] lines = new MyLine[N];

void setup(){
  size(500, 500);
  //create the objects in the array
  for (int i = 0; i<N; i++) {
    lines[i] = new MyLine();
  }
}

void draw(){
  background(255);
  // display and wiggle all lines
  for (int i = 0; i<N; i++) {
    stroke(map(i,0,N,50,250));
    lines[i].wiggle();
    lines[i].display();
  }
}

class MyLine{
  float x1, y1, x2, y2;
  MyLine(){
    x1=random(0,width); y1=random(0,height);
    x2=random(0,width); y2=random(0,height);
  }
  void wiggle(){
    x1+=random(-2,2); y1+=random(-2,2);
    x2+=random(-2,2); y2+=random(-2,2);
  }
  void display(){ line(x1,y1,x2,y2); }
}
```



# ***Project: Floating Cells***

## Cell Class

This class is for cells that can

- **move**
- **wrap**, i.e. once cells go beyond an edge of the sketch, they can re-appear at the opposite edge.

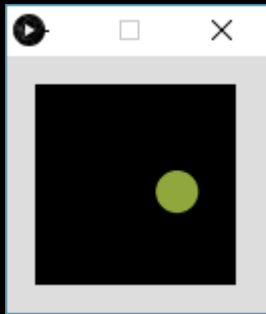
Note that this is almost the same as the Ball class we saw before except that cells can *wrap*.

**Q:** add a new method **bounce()** to the Cell class

```
class Cell{
  float x,y,speedX,speedY,r;
  color c;
  Cell(){
    x = random(0,width);
    y = random(0,height);
    speedX=random(0.3,3);
    speedY=random(0.3,3);
    r = 8/(speedX+speedY); //big circles are slow
    c = color(random(255),random(255),random(255),200);
  }
  void move(){
    x += speedX;    y += speedY;
  }
  void wrap(){
    if (x > width+r) x = -r;
    if (x < -r)      x = width + r;
    if (y > height+r) y = -r;
    if (y < -r)      y = height + r;
  }
  void display(){
    fill(c); noStroke();
    ellipse(x,y,2*r,2*r);
  }
}
```

# One Cell

- Now let's test with one cell only



```
Cell cell;

void setup(){
    size(100,100);

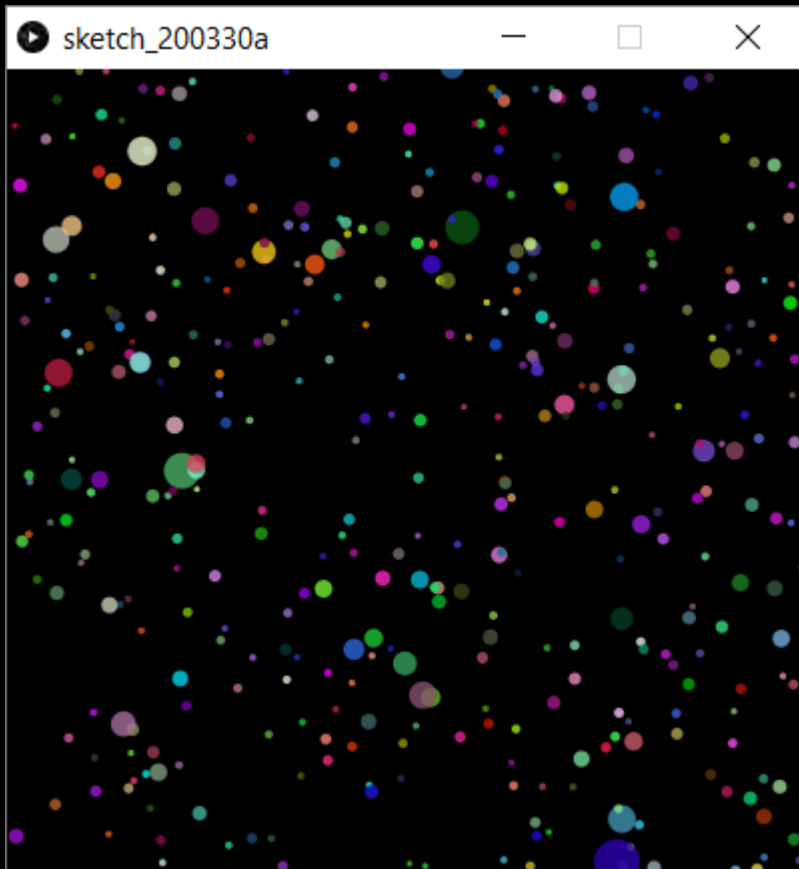
    cell = new Cell();
}

void draw(){
    background(0);

    cell.move();
    cell.wrap();
    cell.display();
}
```

# 200 Cells

- Let's try again with 200 cells



```
int N = 200;

Cell[] cells = new Cell[N];

void setup(){
  size(400,400);
  for(int i=0; i<N; i++){
    cells[i] = new Cell();
  }

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

## ***Practice: Bubbles in My Bottle***

# Bubbles in My Bottle

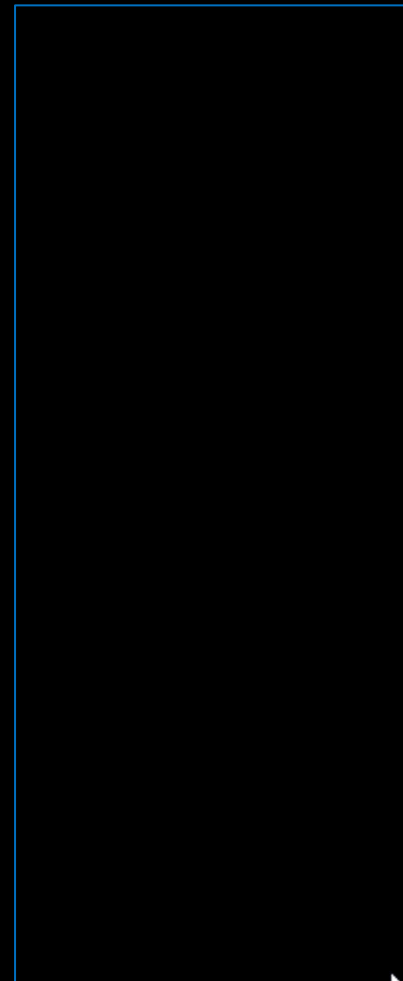
Create the animation of bubbles rising in a bottle:

- Create a bubble class
  - Attributes:  $x, y, radius, speedY$
  - Constructor:
    - $x$  = random value from 0 to the sketch width
    - $y$  = random value from height+50 to height+750
    - $radius$  = random value from 1 to 10
    - $speedY = -10/radius$
  - `move()` function:
    - $x$ -location randomly changes within -0.8 and 0.8
    - $y$ -location is incremented  $speedY$
    - Once a bubble goes above the top edge of the sketch, its attributes are assigned new set of random values (similar to how they were set in the constructor).
  - `Display()` function
    - Draw an ellipse representing the bubble.
- Create an array with 250 bubbles, then move and display them in the animation loop.

## Challenge:

- Make sure we only have a maximum of 12 big bubbles at any time (size of bubbles: small is from 1 to 3, big is 10 to 15)

size(200, 500)



A horizontal bar spanning the width of the slide, composed of a red rectangular segment on the left and a blue rectangular segment on the right.

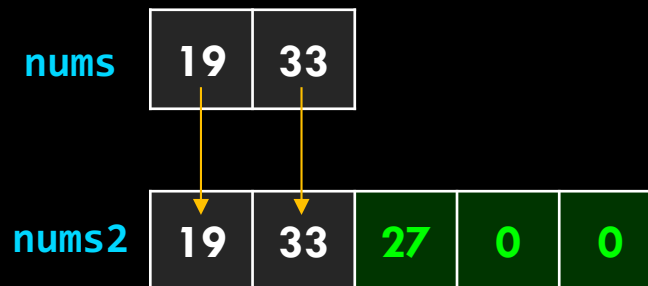
## *Built-in Array Functions*

# Array Size is Fixed!

- Arrays in Java (and Processing) have **fixed size**.
  - Once you declare an array of 10 elements, 10 spots are reserved in the memory, and you cannot store more than 10 elements in those spots.

```
int[] nums = new int[2]; // an array of 2 integers
nums[0] = 19;           // first value saved
nums[1] = 33;           // second value saved
nums[2] = 27; // ERROR! - no space for a third value
```

- What if we want to save more elements than the array size?
  - A good solution is to create another array of a larger size, copy all elements from the original array, and then add more elements in the extra space



# Built-in Array Functions

- Processing provides a set of functions that manipulate the size of the array as explained in previous slide.
- Assume the following declaration for all examples:  
`int[] x, y = {5,6,7,8}, z = {-3,-4};`

Color code:

- **Green** is size
- **Orange** is value
- **Purple** is index

sizing

insert

copy  
sub / join

|                    |                                                                 |                                                                                                                          |
|--------------------|-----------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|
| <b>expand()</b>    | changes array size<br>(increasing or decreasing)                | <code>x = expand(y, 8); // x is {5,6,7,8,0,0,0,0}</code><br><code>x = expand(y, 2); // x is {5,6}</code>                 |
| <b>shorten()</b>   | Removes last element of<br>the array                            | <code>x = shorten(y); // x is {5,6,7}</code>                                                                             |
| <b>append()</b>    | Expands array by 1 spot &<br>adds element at new spot           | <code>x = append(y, -3); // x is {5,6,7,8,-3}</code>                                                                     |
| <b>splice()</b>    | Insert a <b>value</b> or <b>array</b> into<br>an existing array | <code>x = splice(y, -3, 1); // x is {5,-3,6,7,8}</code><br><code>x = splice(y, z, 1); // x is {5,-3,-4,6,7,8}</code>     |
| <b>subset()</b>    | Extracts part of existing<br>array                              | <code>x = subset(y, 1); // x is {6,7,8}</code><br><code>x = subset(y, 1, 2); // x is {6,7}</code>                        |
| <b>concat()</b>    | Concatenates two arrays                                         | <code>x = concat(y, z); // x is {5,6,7,8,-3,-4}</code>                                                                   |
| <b>arrayCopy()</b> | Copy array to another<br>array                                  | <code>//arrayCopy(src, srcPos, dst, dstPos, length)</code><br><code>arrayCopy(z, 0, y, 1, 2); // x is {5,-3,-4,8}</code> |

# *Built-in Array Functions*

- In addition, there are two more methods to change the order of elements: `reverse()` and `sort()`.
- Note: for array of objects, casting is needed.
  - Example:  
`Car[] cars = (Car[]) expand(cars,10);`

# Appending Elements to Array

- In this example, we start by empty array.
- Whenever mouse is pressed, a new ball is created and appended to array.
- *The loop in the draw() method draws all balls currently stored in the array, whatever its current size is.*

```
WigglyBall[] balls=new WigglyBall[0];
void setup(){size(300,300);}
void draw() {
    background(0);
    for(int i=0; i<balls.length; i++){
        balls[i].wiggle();
        balls[i].display();
    }
}
void mousePressed(){
    //append new ball to array when mouse is pressed
    WigglyBall b = new WigglyBall(mouseX,mouseY);
    balls = (WigglyBall[]) append(balls,b);
}
class WigglyBall{
    float x,y,r;  color c;
    WigglyBall(float mx,float my){
        x = mx;  y = my;  r = random(16,32);
        c=color(random(255),random(255),random(255),180);
    }
    void wiggle(){x+=random(-1,1); y+=random(-1,1);}
    void display(){
        noStroke();  fill(c);
        ellipse(x,y,2*r,2*r);
    }
}
```

Start with empty array

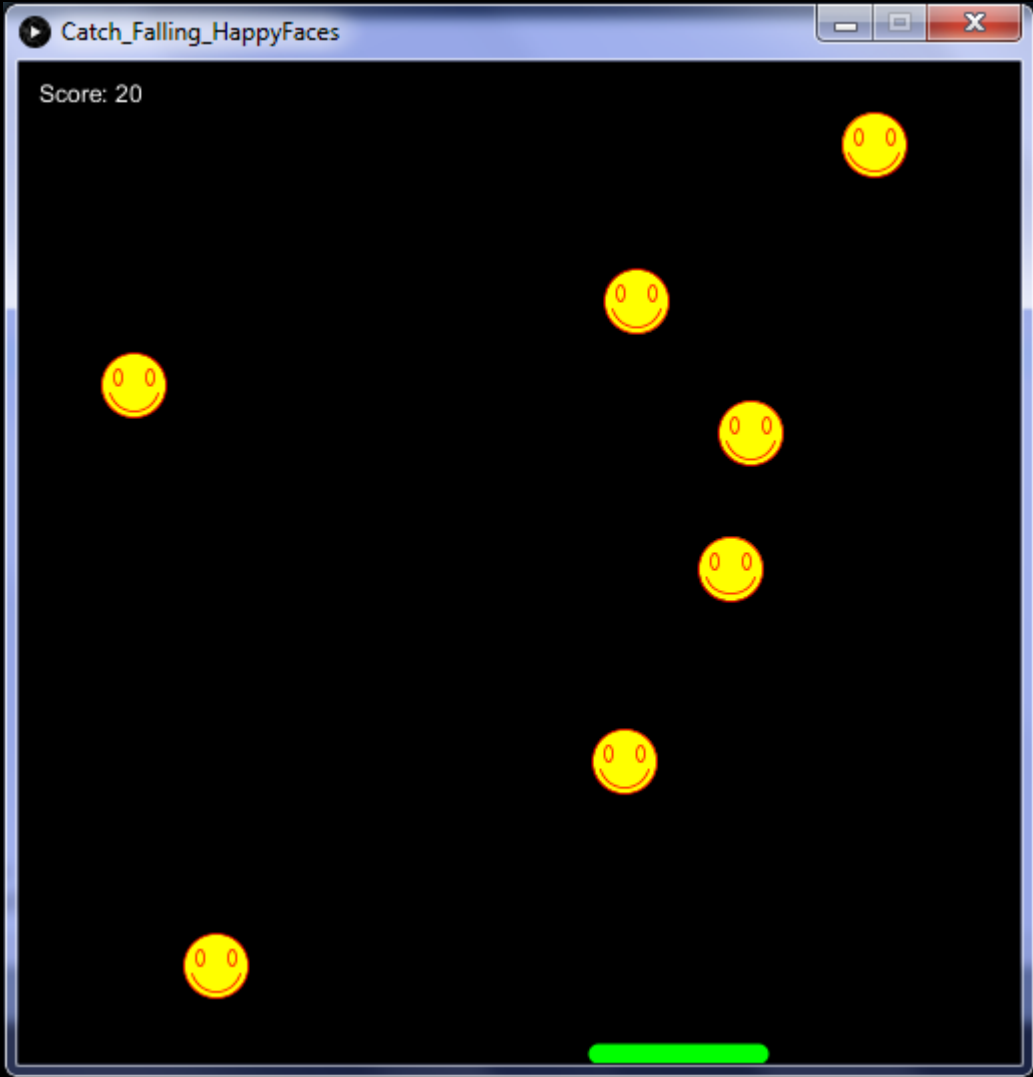
wiggle and draw all balls currently in the array.

The image features a 3D visualization of a face falling through a grid of yellow spheres. The face is positioned at the bottom center, with its head tilted upwards. It is surrounded by a grid of yellow spheres, some of which are highlighted with a red glow. The background is a dark blue gradient. A horizontal bar with a blue-to-teal gradient is located in the middle of the image, containing the text "Project: Catch Falling Face!".

## *Project: Catch Falling Face!*

# Catch the Falling Emoji's

- Design a game where the player catches falling faces.
- Use the HappyFace and Paddle class
  - HappyFace class needs modification (should be given).
- Add code to detect if a happy face is caught by the paddle



# The classes

```
class HappyFace {
  float x, y, r, speedY;
  HappyFace() {
    reset();
  }
  void move() {
    y += speedY;
    if (y>height+r) reset();
  }
  void reset() {
    r = 16;
    x = random(r,width-r);
    y = random(-500);
    speedY = random(0.5, 3);
  }
  void display() {
    fill(255,255,0);
    stroke(255,0,0); strokeWeight(r/20);
    ellipse(x, y, 2*r, 2*r);
    arc(x,y,1.4*r,1.4*r,.1*PI,.9*PI);
    fill(105,54,0);
    ellipse(x+r/2, y-r/4, r/4, r/2);
    ellipse(x-r/2, y-r/4, r/4, r/2);
  }
}
```

```
class Paddle {
  float x, y, w, h;
  Paddle(){
    this(width/2, height-5, 80, 10);
  }
  Paddle(float x1,float y1,float w1,float h1){
    x = x1;    y = y1;
    w = w1;    h = h1;
  }

  void move() {
    this.x = mouseX;
  }
  void display() {
    stroke(0, 255, 0);
    strokeCap(ROUND);
    strokeWeight(h);
    line(x-this.w/2,y,x+w/2,y);
  }
}
```

# Falling Emoji: In-Class Demonstration

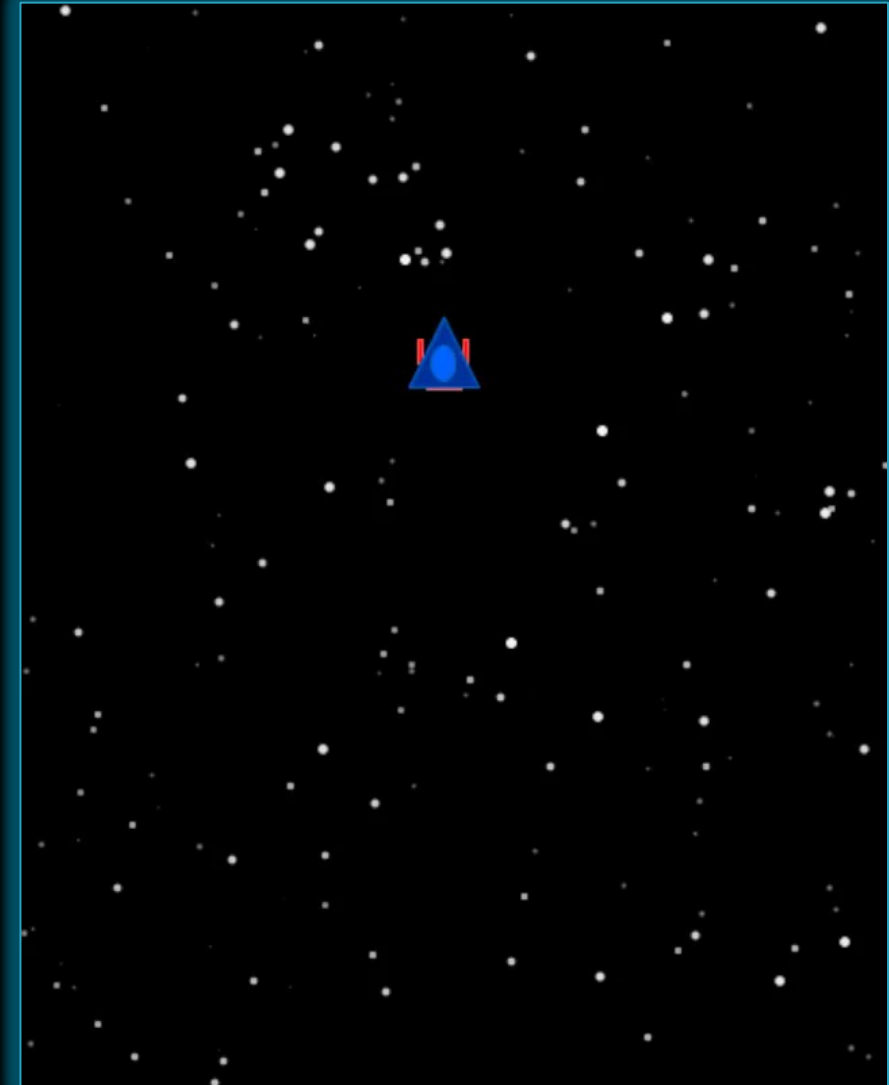
```
Paddle paddle;    HappyFace[] faces;
int numFaces = 10, score = 0;
void setup() {
    size(500, 500); textSize(24); noCursor();
    paddle = new Paddle();
    faces = new HappyFace[numFaces];
    for (int i = 0; i<faces.length; i++)
        faces[i] = new HappyFace(); //x,y,r,speedY
}
void draw() {
    background(0);
    //score
    fill(255); text("Score: " + score, 10,20);
    //Paddle
    paddle.move(); paddle.display();
    //happy faces
    for (int i = 0; i<faces.length; i++) {
        faces[i].move(); faces[i].display();
        //check every happy face if it is caught by paddle
        if (faces[i].y>=height-faces[i].r/2-paddle.h && abs(paddle.x-faces[i].x)<paddle.w/2) {
            faces[i].reset();
            score++;
        }
    }
}
```

The background of the slide is a dark space filled with numerous small white stars. Three large, solid red circles are positioned in the upper right quadrant, resembling planets or moons. A horizontal teal bar spans the middle of the slide, containing the title text. Below the bar, a small blue rocket ship with a red nose cone is visible at the bottom center, and a vertical teal line of stars is located to its left.

# ***Project: Galaxy Navigator!***

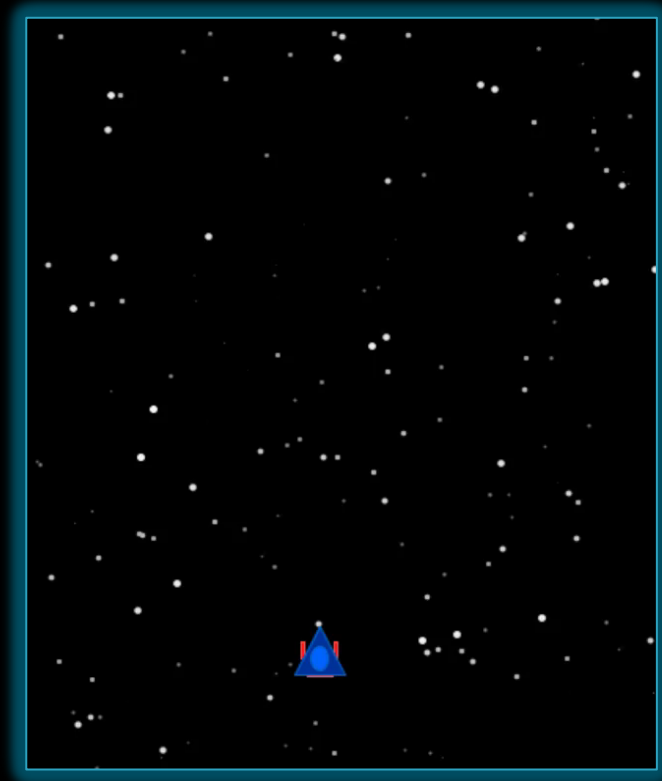
# Galaxy Navigator!

- In this example, we will build a spaceship flying in space.
- The spaceship is controlled by the mouse.
- The background is a star field with 200 stars having various sizes, speeds, and levels of brightness.
  - slower stars are smaller and darker. i.e. star speed controls its brightness and size.



# *Pseudo code for a space shooter game*

```
Create one spaceship;  
Create an array of stars  
Create an array of enemies  
Create an array of bullets  
void draw(){  
    spaceship.move();  
    for each star: star.move();  
    for each enemy: enemy.move();  
    for each bullet {  
        bullet.move();  
        //check collisions  
        for each enemy{  
            if (bullet.hits(enemy)){  
                bullets.remove(bullet);  
                enemies.remove(enemy);  
                score++  
            }  
        }  
    }  
    display all objects;  
}
```



# Create Star Class

- Create Star class that has:
  - 3 attributes: x, y, and speed.
  - A zero-arg constructor
    - Sets x and y to random values within the window
    - Sets the speed to random value between 0 and 2.5
  - 2 functions:
    - `move()`:
      - increment y by the speed
      - wraps around the screen when the star reaches the edge. The new location is set above the screen at a random value from -100 to 0.
    - `display()`
      - draws the star as a `point()` at (x,y).
      - To give the illusion of depth, draw the star with brightness and size that depend on speed. (slower stars are smaller and darker. i.e. star speed controls its brightness and size)
        - Stroke: color: from 50 to 250 (for speed from 0 to 2.5).  
weight:  $2 * \text{speed}$

# Create Star Class

```
class Star{
  float x,y,speed;
  Star(){
    x = random(0,width);
    y = random(0,height);
    speed = random(0,2.5);
  }
  void move(){
    y += speed;
    if(y>height) y=random(-100);
  }
  void display(){
    stroke(map(speed,0,2.5,50,255));
    strokeWeight(speed*2);
    point(x,y);
  }
}
```

# Create and Animate 200 Stars

- Create a global array of 200 Star references
- In setup()
  - Choose appropriate size for your window
  - Call noCursor() built-in function in order not to display the cursor.
  - Create star objects for your array.
- In draw():
  - move() and display() all stars in your array.
- Run the code. The output should be the background stars without the spaceship.

# Create and Animate 200 Stars

```
Star[] stars = new Star[200];
void setup(){
    size(500,500);
    for(int i=0;i<stars.length;i++){
        stars[i] = new Star();
    }
    void draw(){
        background(0);

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

```
class Star{
    float x,y,speed;
    Star(){
        x = random(0,width);
        y = random(0,height);
        speed = random(0,2.5);
    }
    void move(){
        y += speed;
        if(y>height) y=random(-100);
    }
    void display(){
        stroke(map(speed,0,2.5,50,255));
        strokeWeight(speed*2);
        point(x,y);
    }
}
```

# Add Your Favorite Spaceship

- Add code to your program so that it displays a spaceship following the mouse movement. In my example, I used the spaceship design we created earlier.
- Here is what I did:
  - I copied the Spaceship class to our program.
  - I added a move() function for the ship to follow the mouse.
  - I moved and displayed the spaceship in draw() function.
- You can either use the above spaceship, or you can **create your own class** that has either
  - your own spaceship design.
  - an image of a spaceship (e.g. from <http://kenney.nl>) that represents a drawing of a spaceship design.



# Updated Code *(new code in green)*

```
Spaceship spaceship = new Spaceship();
Star[] stars = new Star[200];
void setup(){
    size(500,500);
    for(int i=0;i<stars.length;i++){
        stars[i] = new Star();
        noCursor();
    }

    void draw(){
        background(0);

        //stars
        ... (same as before)

        //spaceship
        spaceship.move();
        spaceship.display();
    }
}

class Spaceship {
    float x, y, size;
    Spaceship() {
        x = width/2; y = height/2; size=32;
    }
    void move(){ x = mouseX; y = mouseY; }
    void display() {
        // draw side guns
        rectMode(CENTER); fill(255, 0, 0);
        strokeWeight(1); stroke(255,90,90);
        rect(x-size/3,y+size/2,size/15,size/3);
        rect(x+size/3,y+size/2,size/15,size/3);
        // draw jet engine
        fill(255,180,0); rect(x,y+size,size/2,size/10);
        // draw main body
        stroke(0,100,200); fill(0, 50, 155);
        triangle(x,y,x+size/2,y+size,x-size/2,y+size);
        fill(0, 100, 255);
        ellipse(x, y+2*size/3, size/3, size/2);
        rectMode(CORNER);
    }
}

class Star{
    ... (same as before)
}
```

# Going Warp Speed!

- In this example, we will modify your code so that the spaceship appears to be warping in space as it moves up the screen! To give warp illusion:
  - Stars should move faster
  - Stars and ship should have trails.
- Here is how to do this:
  - Replace `background(0)` in `draw()` with a black rectangle of which transparency depends on `mouseY`
  - Add another attribute to `Star` class, e.g. `speedInc`, that has a value ranging from 0 (when the mouse is at the bottom) to 2 (when the mouse is at the top). Used `speedInc` in the `move()` method to update `y` using this expression:  
 $y = y + \text{speed} + \text{speedIncr};$



## Step (4) : code

# Updated Code *(new code in green)*

```
Spaceship spaceship = new Spaceship();
Star[] stars = new Star[200];
void setup(){
  ... (same as before)
}
void draw(){
  //instead of background(0);
  fill(0,map(mouseY,0,width,10,255));
  rect(0,0,width,height);

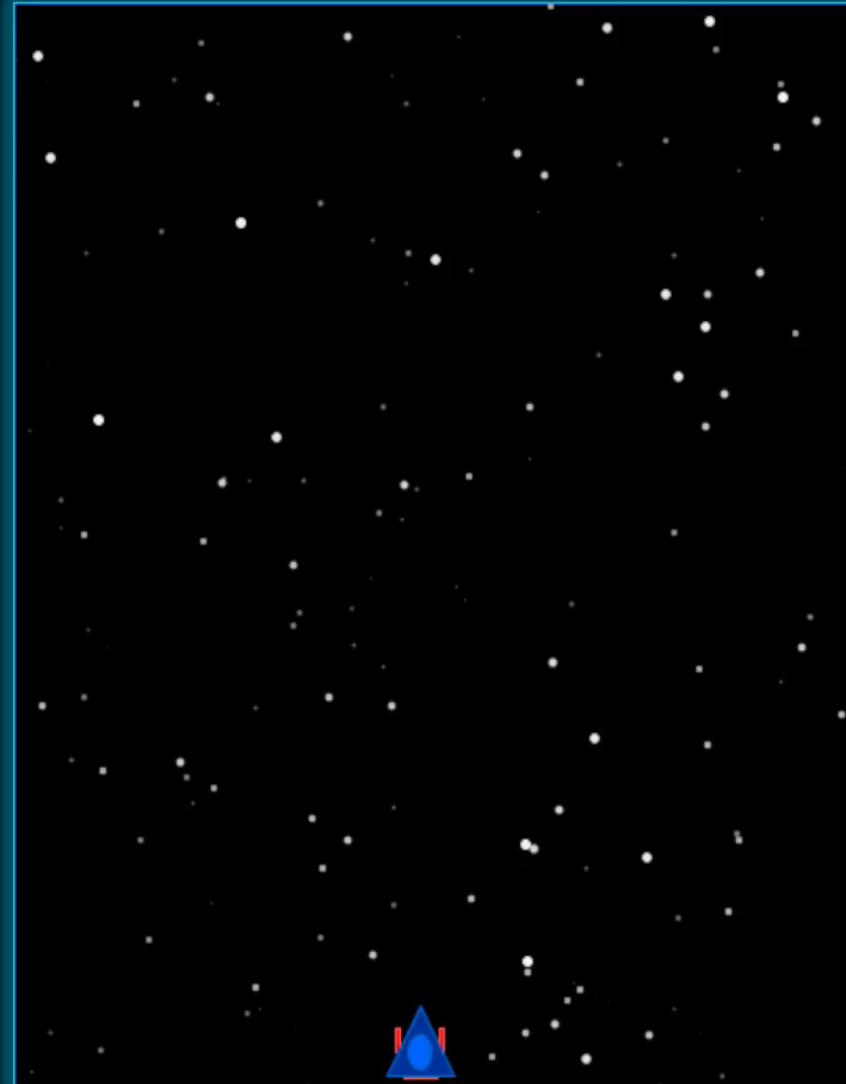
  //stars
  for(int i=0;i<stars.length;i++){
    stars[i].speedIncr =
      map(mouseY,0,height,3,0);
    stars[i].move();
    stars[i].display();
  }

  //spaceship
  ... (same as before)
}

class Spaceship {
  ... (same as before)
}
class Star{
  float x,y,speed,speedIncr;
  Star(){
    x = random(0,width);
    y = random(0,height);
    speed = random(0,2.5);
  }
  void move(){
    y += speed + speedIncr;
    if(y>height) y=random(-100);
  }
  void display(){
    stroke(map(speed,0,2.5,50,255));
    strokeWeight(speed*2);
    point(x,y);
  }
}
```

# Shooting in the Space!

- In this example, we will add more code so that we can shoot bullets from our spaceship using the Space button.
- Here is how to do this:
  - Create a class `Bullet` with `x,y` attributes, a constructor to set `x,y`, and 2 functions:
    - `move()`: decrements `y` by 5,
    - `display()`: draws a line from `(x,y)`.
  - Create an empty array of `Bullet` type, let's call it `bullets`.
  - Create `keyPressed()` function that creates a new `Bullet` at location of the spaceship, and then appends the new bullet to `bullets` array.
  - In the `draw()` function, create a loop the moves and displays all bullets in the array, regardless of the array length.
- What is the problem with this algorithm?



# Updated Code *(new code in green)*

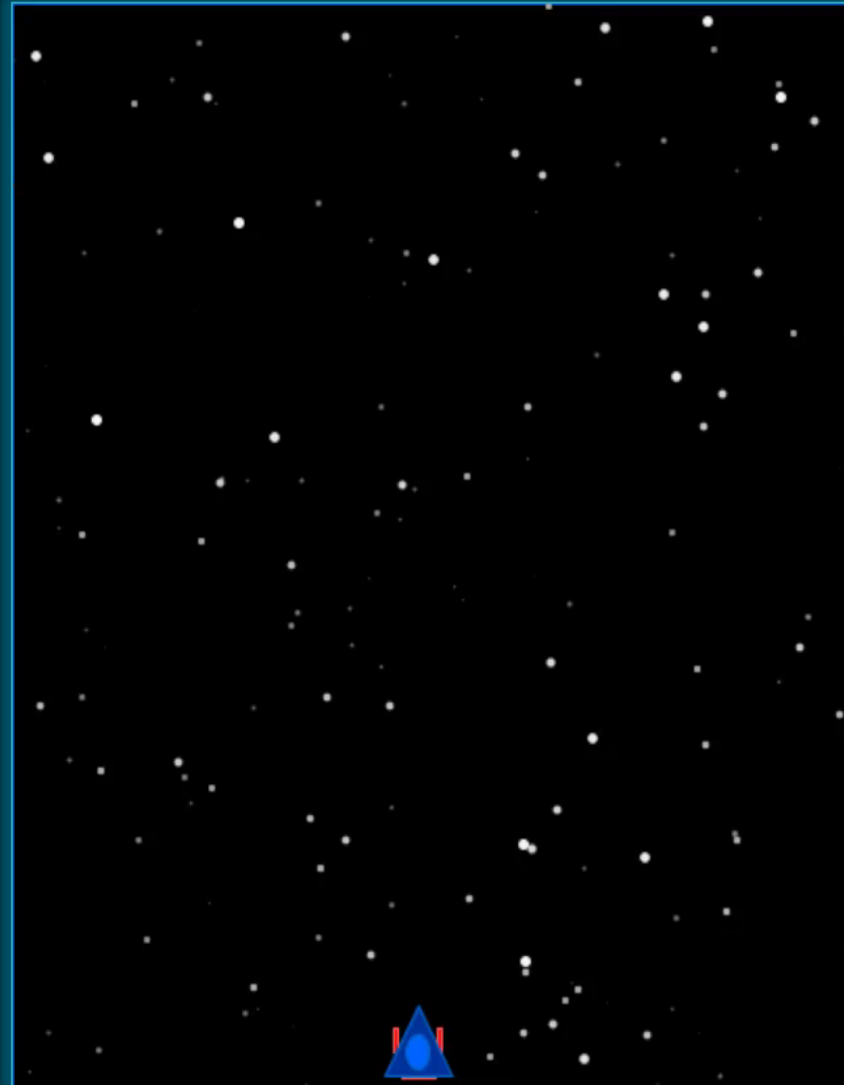
```
Bullet[] bullets = {};  
Spaceship spaceship = new Spaceship();  
Star[] stars = new Star[200];  
void setup(){  
    ... same as before then  
    for(int i = 0; i<enemies.length; i++){  
        enemies[i]=new Enemy(width/2,-300-90*i);  
    }  
void draw(){  
    //background;r  
    ...  
    //stars  
    ...  
    //spaceship  
    ...  
    //bullets  
    for(int i=0;i<bullets.length;i++){  
        bullets[i].move();  
        bullets[i].display();  
    }  
}  
void keyPressed(){  
    if(key == ' '){  
        Bullet b=  
            new Bullet(spaceship.x,spaceship.y);  
        bullets = (Bullet[])append(bullets,b);  
    }  
}
```

```
class Spaceship {  
    ... (same as before)  
}  
class Star{  
    ... (same as before)  
}  
class Bullet{  
    float x,y;  
    Bullet(float bx, float by){  
        x = bx; y = by;  
    }  
    void move(){  
        y -= 5;  
    }  
    void display(){  
        stroke(0,255,255);  
        strokeWeight(3);  
        line(x,y,x,y+25);  
    }  
}
```

**Problem: a bullet that  
is off the screen is still  
moving!**

# Shooting in the Space!

- Once a bullet goes off the screen, we need to remove it from bullets array.
- How to do this? Here is one way:
  - add a Boolean attribute, e.g. **active**, to keep track of the state of the bullet. This attribute should be set to **false** when the bullet leaves the screen (e.g. in the move function).
  - In the draw() function, check all bullets and **remove** anyone that is **not active**.
- There is no built-in function for removing an element from an array, but you can use the two functions `arrayCopy()` and `shorten()` to remove an element. HOW?



## Step (6): code

*(new code in green)*

```
...
void setup(){
  ... (same as before)
}
void draw(){
  //background;
  ...
  //stars
  ...
  //spaceship
  ...
  //bullets
  for(int i=0;i<bullets.length;i++){
    bullets[i].move();
    bullets[i].display();
    //remove inactive bullets
    if(!bullets[i].active){
      arrayCopy(bullets,i+1,bullets,
                i,bullets.length-1-i);
      bullets=(Bullet[])shorten(bullets);
    }
  }
}
void keyPressed(){ ... }
```

```
class Spaceship {
  ... (same as before)
}
class Star{
  ... (same as before)
}
class Bullet{
  float x,y;
  boolean active = true;
  Bullet(float bx, float by){
    x = bx; y = by;
  }
  void move(){
    y -= 5;
    if(y<0)
      active = false;
  }
  void display(){
    stroke(0,255,255);
    strokeWeight(3);
    line(x,y,x,y+25);
  }
}
```

# Beware of the Enemies!

- In this part, we will add 6 enemy ships that move in a sinusoidal path. Here is how to do this:
  - Create an Enemy class with (x,y) attributes, and move() and display() functions.
    - In the move() function:
      - Increment y by some amount
      - Compute x using sin() function:  
$$x = \text{width}/2 + \text{width}/3 * \sin(\text{map}(y, 0, \text{height}, 0, 2 * \text{PI}))$$
*Can you explain this expression?*
      - If enemy goes off the screen, set its y back to the top.
    - Create an Enemy array with 6 enemies.
    - In the draw() function, move() and display() all enemies using a loop.
  - Don't worry now about hitting and destroying the enemy ships!



## Step (7): code

*(new code in green)*

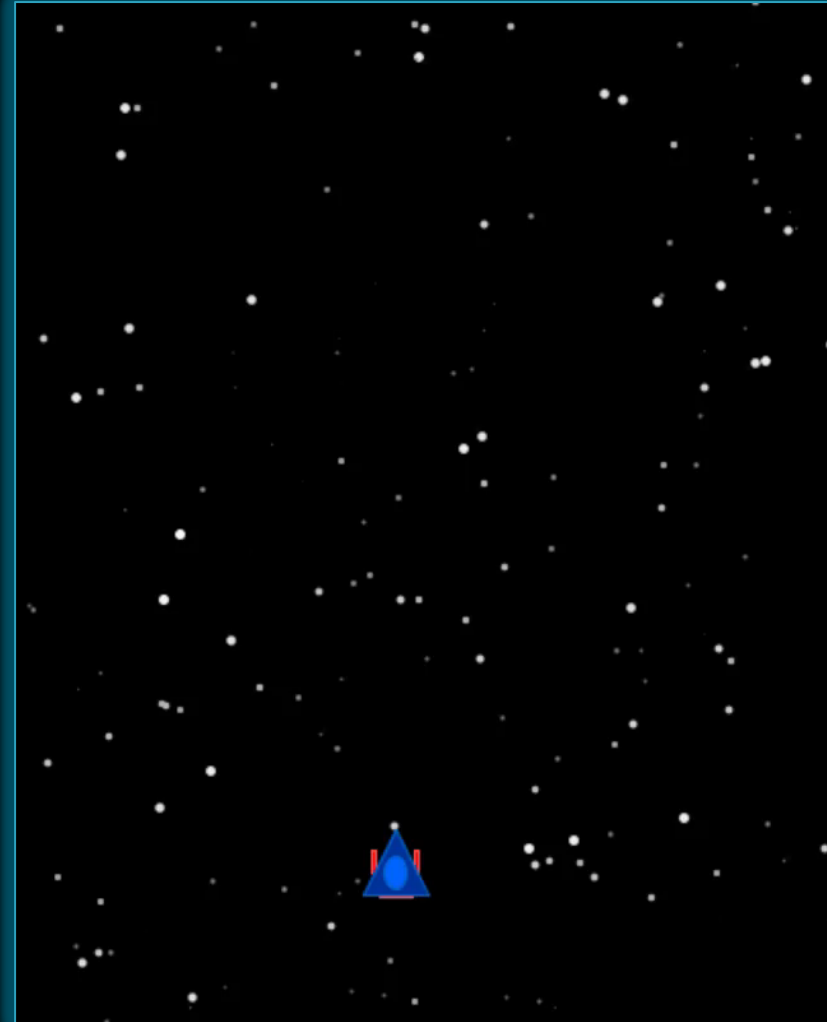
```
Spaceship spaceship = new Spaceship();
Star[] stars = new Star[200];
Enemy[] enemies = new Enemy[6];
void setup(){
    ... (same as before)
    for(int i = 0; i<enemies.length; i++)
        enemies[i]=new Enemy(width/2,-300-90*i);
}
void draw(){
    //background;
    ...
    //stars
    ...
    //spaceship
    ...
    //bullets
    ...
    //enemies
    for(int i = 0; i<enemies.length; i++){
        enemies[i].move();
        enemies[i].display();
    }
}

void keyPressed(){ ... }
```

```
class Spaceship {
    ...
}
class Star{
    ...
}
class Bullet{
    ...
}
class Enemy {
    float x, y, r;
    Enemy(float ex, float ey) {
        x = ex; y = ey; r = 16;
    }
    void move() {
        y+=2;
        x=width/2+width/3*sin(map(y,0,height,0,2*PI));
        if (y>height+r) y = -r;
    }
    void display() {
        fill(255, 0, 0); noStroke();
        float rnd = random(2*r-6, 2*r+6);
        ellipse(x, y, rnd, rnd);
    }
}
```

# Eliminate Those Evil Enemies!

- To know whether an enemy is hit, we compute the distance between each bullet and each enemy. (hint: nested for loops)
- Whenever the distance is less than a certain number, the enemy is hit.
  - We can do this using either:
    - `dist()` function within the draw function, or
    - one more method to bullets that checks the distance and returns true or false
      - e.g. `bullet.hit(enemy)`
- What happens when the enemy is hit?
  - The bullet becomes inactive (vanishes)
  - The enemy also becomes inactive (*using same technique used in bullets*).
  - Score is incremented.
- We can use a similar technique to check whenever our spaceship pumps into an enemy, but we will not do it today! You are welcome to try if you wish 😊



## Step (8): code

*(new code in green)*

```
...
void setup(){...}
void draw(){
    //background ...
    //stars ...
    //spaceship ...
    //bullets
    for(int i=0;i<bullets.length;i++){
        bullets[i].move();
        bullets[i].display();
        //check if bullet hits an enemy
        for(int j = 0; j<enemies.length; j++){
            if(bullets[i].hit(enemies[j])){
                bullets[i].active = false;
                enemies[j].active = false;
            }
        }
        //remove bullets (gone off screen or hit enemy)
    }
    //enemies
    ...
    //remove enemies that were shot
    if(!enemies[i].active){
        arrayCopy(enemies,i+1,enemies,
                  i,enemies.length-1-i);
        enemies = (Enemy[])shorten(enemies);
    }
}

void keyPressed(){ ... }
```

```
class Spaceship {
    ...
}
class Star{
    ...
}
class Bullet{
    ... Same except we add this method
    //check if bullet hits enemy
    boolean hit(Enemy enemy){
        if(dist(x,y,enemy.x,enemy.y)<enemy.r)
            return true;
        else
            return false;
    }
}
class Enemy {
    ... same except we add one attribute
    boolean active = true;
}
```