PAT 204/504 (Fall 2024)

Creative Coding

Lecture 7: Lists & Data I/O

Instructor: Hao-Wen Dong

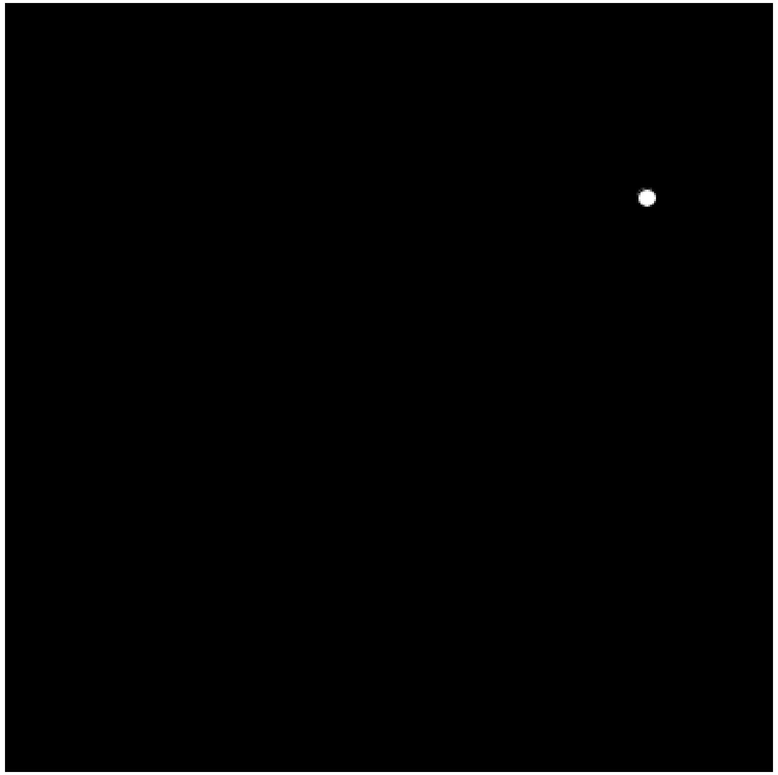
(Recap) Example: Bouncing Ball

```
class Ball {
  float size = 10;
  float speed = 5;
  float x, y, speedX, speedY;

  Ball() {
    // Constructor
  }

  void show() {
    // Show the ball
  }

  void move() {
    // Move the ball
  }

  void checkWalls() {
    // Check if the ball hit the walls
  }
}
```

Fields

Constructor

Methods



(Recap) Example: Bouncing Balls

```
Ball[] balls = new Ball[20];
```

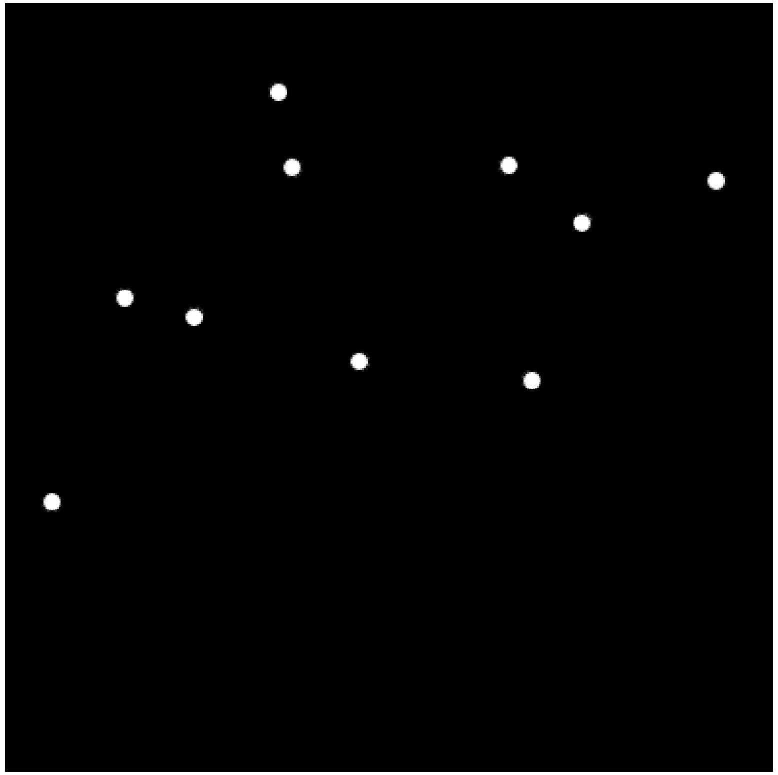
An array of objects

```
void setup() {
  size(400, 400);

  for (int i = 0; i < balls.length; i++) {
    balls[i] = new Ball();
  }
}
```

Initialization

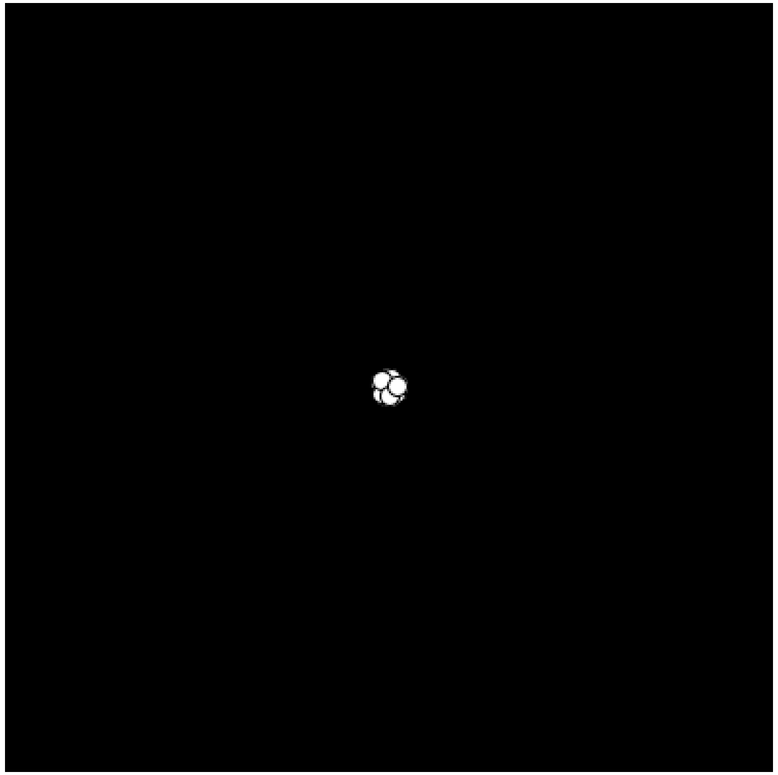
```
void draw() {
  background(0);

  for (int i = 0; i < balls.length; i++) {
    balls[i].move();
    balls[i].checkWalls();
    balls[i].show();
  }
}
```

Call the methods!



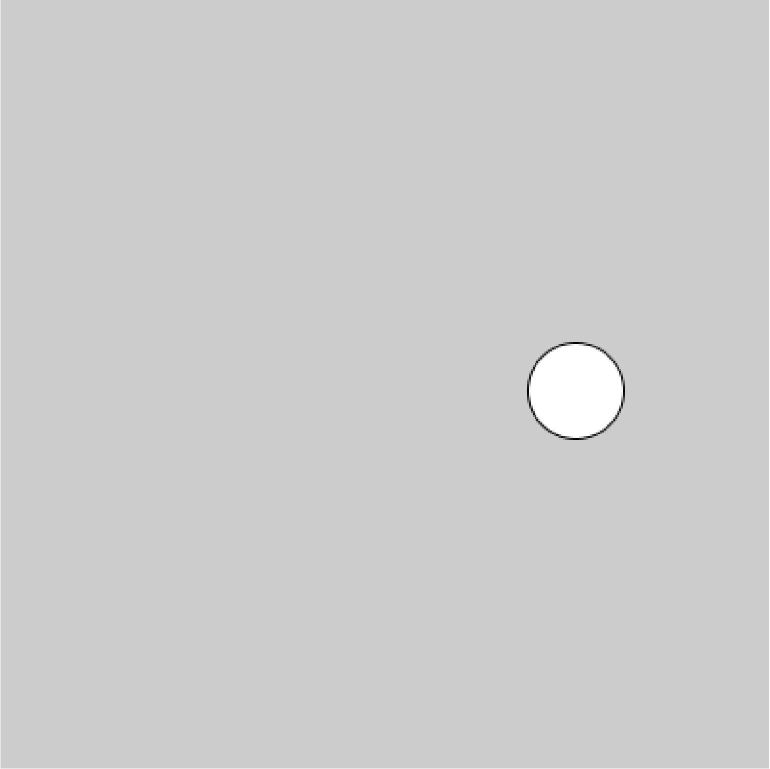
(Recap) Signature Polymorphism

```
class Ball {
  float size = 10;
  float speed = 5;
  float x, y, speedX, speedY;

  Ball() {
    x = random(width);
    y = random(height);

    float theta = random(0, TWO_PI);
    speedX = speed * cos(theta);
    speedY = speed * sin(theta);
  }

  Ball(float x, float y) {
    this.x = x;
    this.y = y;

    float theta = random(0, TWO_PI);
    speedX = speed * cos(theta);
    speedY = speed * sin(theta);
  }
}
```



(Recap) Example: Rotating Ball

```
PVector vec = new PVector(100, 0);

void setup() {
  size(400, 400);
}

void draw() {
  // Rotate the vector by a fixed angle
  vec.rotate(PI * 0.01);

  // Draw the circle
  circle(200 + vec.x, 200 + vec.y, 50);
}
```

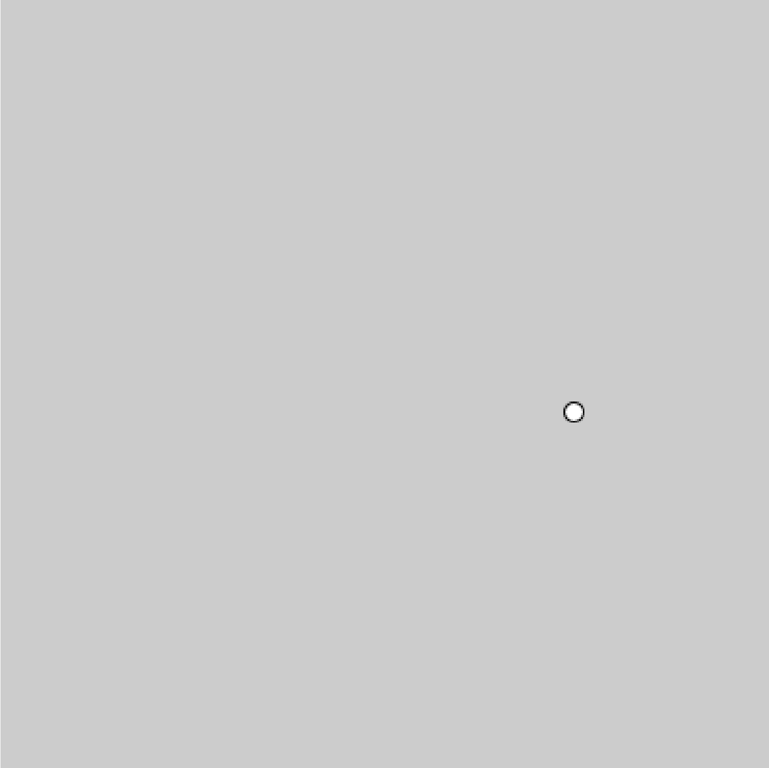


(Recap) Example: PVector.random2d

```
PVector pos = new PVector();

void setup() {
  size(400, 400);
  frameRate(30);
}

void draw() {
  pos = PVector.random2D().mult(100);
  circle(200 + pos.x, 200 + pos.y, 10);
}
```

Get a random 2D unit vector

Scale the vector by 100



Homework 3: Spectrum Visualizer

- Modify the template code to implement a spectrum visualizer
- Instructions will be released on Gradescope
- Due at **11:59pm ET** on **September 23**
- Late submissions: **1 point deducted per day**

FFT Class

```
import processing.sound.*;

int bands = 512;
FFT fft = new FFT(this, bands);
AudioIn in = new AudioIn(this, 0);
float[] spectrum = new float[bands];

void setup() {
  size(512, 360);

  in.start();
  fft.input(in);
}

void draw() {
  background(255);

  fft.analyze(spectrum);

  for(int i = 0; i < bands; i++){
    line(i, height, i, height - spectrum[i] * height * 5);
  }
}
```

Import the Sound library

Initialize an FFT object

Initialize an AudioIn object

Initialize an array to store the spectrum

Start taking audio input

Route the audio input to the FFT analyzer

Specify the array to store the outputs

Run Fast Fourier Transform

Normalized to [0, 1]

Amplitude Class

```
import processing.sound.*;

Amplitude amp = new Amplitude(this);
AudioIn in = new AudioIn(this, 0);
float a;

void setup() {
  size(400, 400);

  in.start();
  amp.input(in);
}

void draw() {
  background(0);
  a = amp.analyze();
  circle(200, 200, a * 400);
}
```

Initialize an Amplitude object

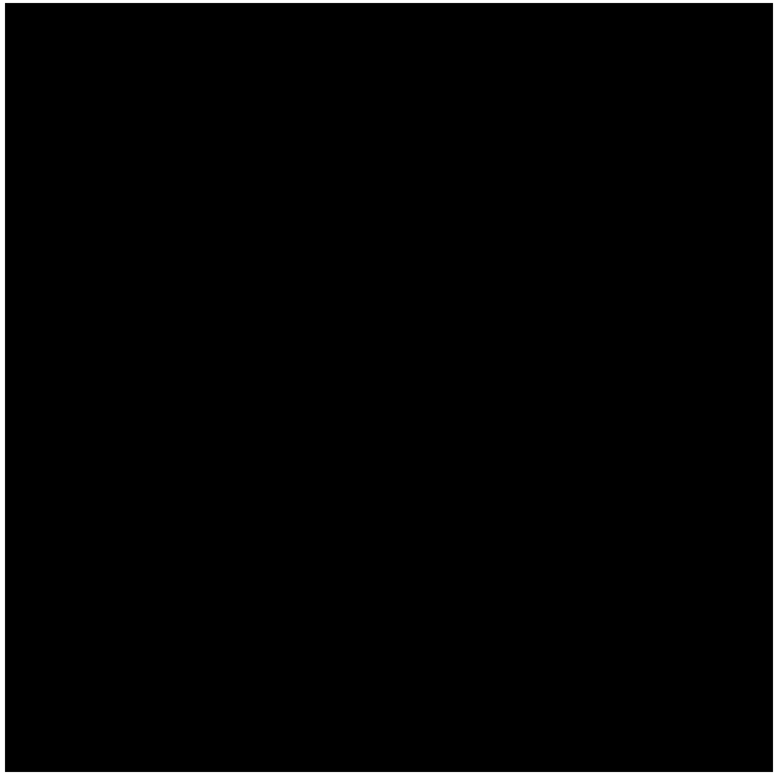
Initialize an AudioIn object

Start taking audio input

Route the audio input to the amplitude meter

Measure the amplitude

Normalized to [0, 1]



PitchDetector Class

```
import processing.sound.*;

PitchDetector pd = new PitchDetector(this);
AudioIn in = new AudioIn(this, 0);
float pitch;

void setup() {
  size(400, 400);

  in.start();
  pd.input(in);

  stroke(#ff0000);
  strokeWeight(5);
}

void draw() {
  background(0);
  pitch = pd.analyze(0.5);
  if (pitch > 0) {
    line(0, height - pitch, 400, height - pitch);
  }
}
```

Initialize a PitchDetector object

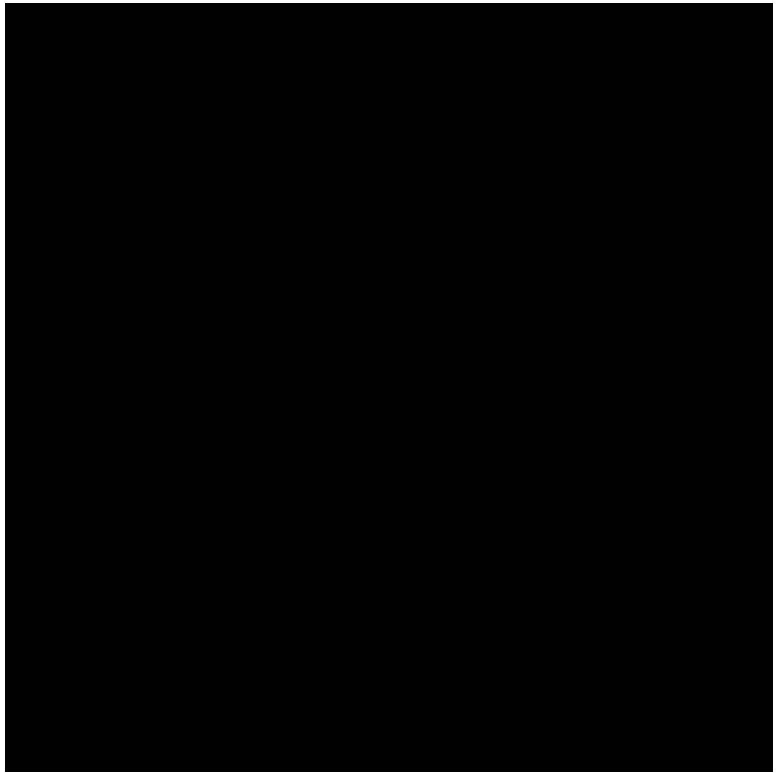
Initialize an AudioIn object

Start taking audio input

Route the audio input to the pitch detector

Set the sensitivity

Detect the pitch



Midterm Assignment: Build Your Own Music Visualizer

- **Open-ended** assignment
- Use everything you’ve learned from the class (and beyond!)
- Instructions will be released on Gradescope
- Due at **11:59pm ET** on **October 7**
- Late submissions: **NOT Accepted** **(Submit early and update later!)**

Midterm Assignment – Rubrics

- Use **two of the following three concepts** (**10pt**)
  - **Loops and recursion**
  - **Data structures** (e.g., arrays, lists, dictionaries, etc.)
  - **Objects**
- **Clear documentation** in code (**5pt**)
- Live demo in class on **October 7** (**5pt**)

SoundFile Class

```
import processing.sound.*;

SoundFile file;

void setup() {
  size(400, 400);

  file = new SoundFile(this, "emil-telmanyi_bwv1006.mp3");
  file.play();
}

void draw() {
}
```

```
fft.input(file);
```

```
amp.input(file);
```

```
pd.input(file);
```

Exercise: Spectrum Visualizer with `SoundFile`

- Modify the code so that it takes **`SoundFile`** rather than `AudioIn` as input
- Note that you need to put the audio file in the **`data/`** directory

```
fft.input(file);
```

Code

Example music

Lists

Lists

- Similar to arrays, lists hold several items of the same data type
- However, lists are built to have a **dynamic size**
- The simplest ones are **`IntList`** and **`FloatList`**

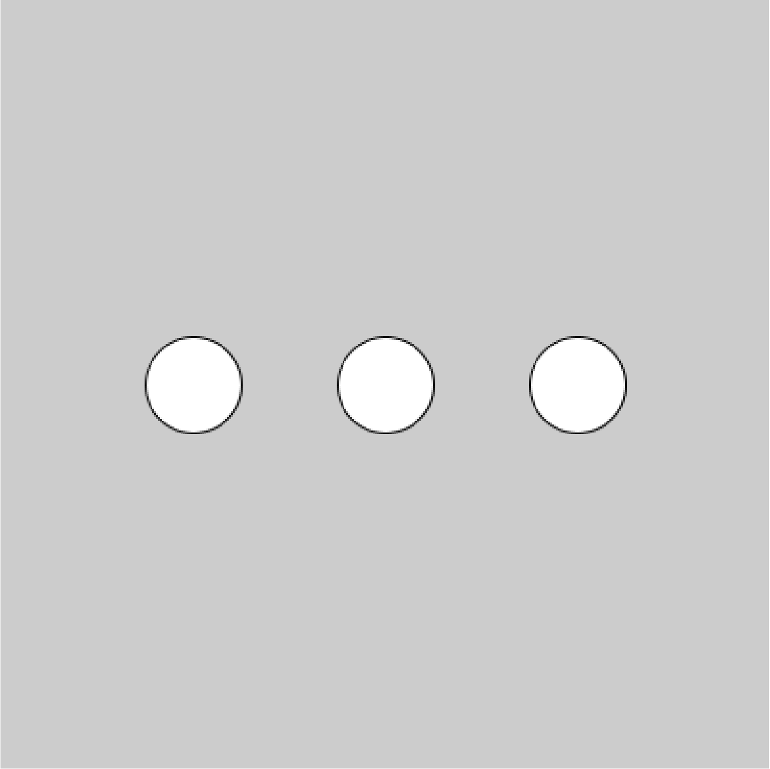
Example: Three Circles

```
IntList pos = new IntList();

void setup() {
  size(400, 400);

  pos.append(100);
  pos.append(200);
  pos.append(300);
}

void draw() {
  for (int i = 0; i < pos.size(); i++) {
    circle(pos.get(i), 200, 50);
  }
}
```

Declaration

Initialization

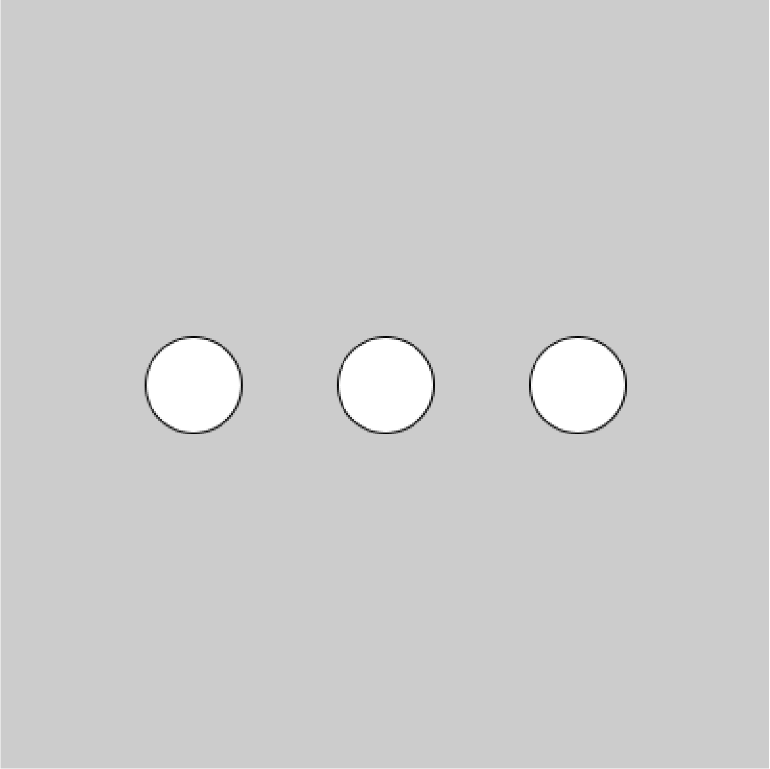
Length of the list



Example: Three Circles

```
IntList pos = new IntList();

void setup() {
  size(400, 400);

  pos.append(100);
  pos.append(200);
  pos.append(300);
}

void draw() {
  for (int x: pos) {
    circle(x, 200, 50);
  }
}
```

For-each loop



Array vs List

Array

```
float[] pos = new float[3];

void setup() {
  size(400, 400);

  pos[0] = 100;
  pos[1] = 200;
  pos[2] = 300;
}

void draw() {
  for (int i = 0; i < pos.length; i++) {
    circle(pos[i], 200, 50);
  }
}
```

Declaration

Initialization

Length of the array

List

```
FloatList pos = new IntList();

void setup() {
  size(400, 400);

  pos.append(100);
  pos.append(200);
  pos.append(300);
}

void draw() {
  for (int i = 0; i < pos.size(); i++) {
    circle(pos.get(i), 200, 50);
  }
}
```

Declaration

Initialization

Length of the list

Lists Methods

- `size()` Return the size of the list
- `get()` Return the value at the specified index
- `append()` Append an item to the end of the list
- `insert()` Insert an item to the specific index
- `set()` Set the value at the specified index
- `remove()` Remove an item at the specified index
- `clear()` Clear everything in the list
- `hasValue()` Whether the value is in the list or not

List Methods

- Unlike arrays, most list methods are **in-place**
  - `add()`
  - `sub()`
  - `mult()`
  - `div()`

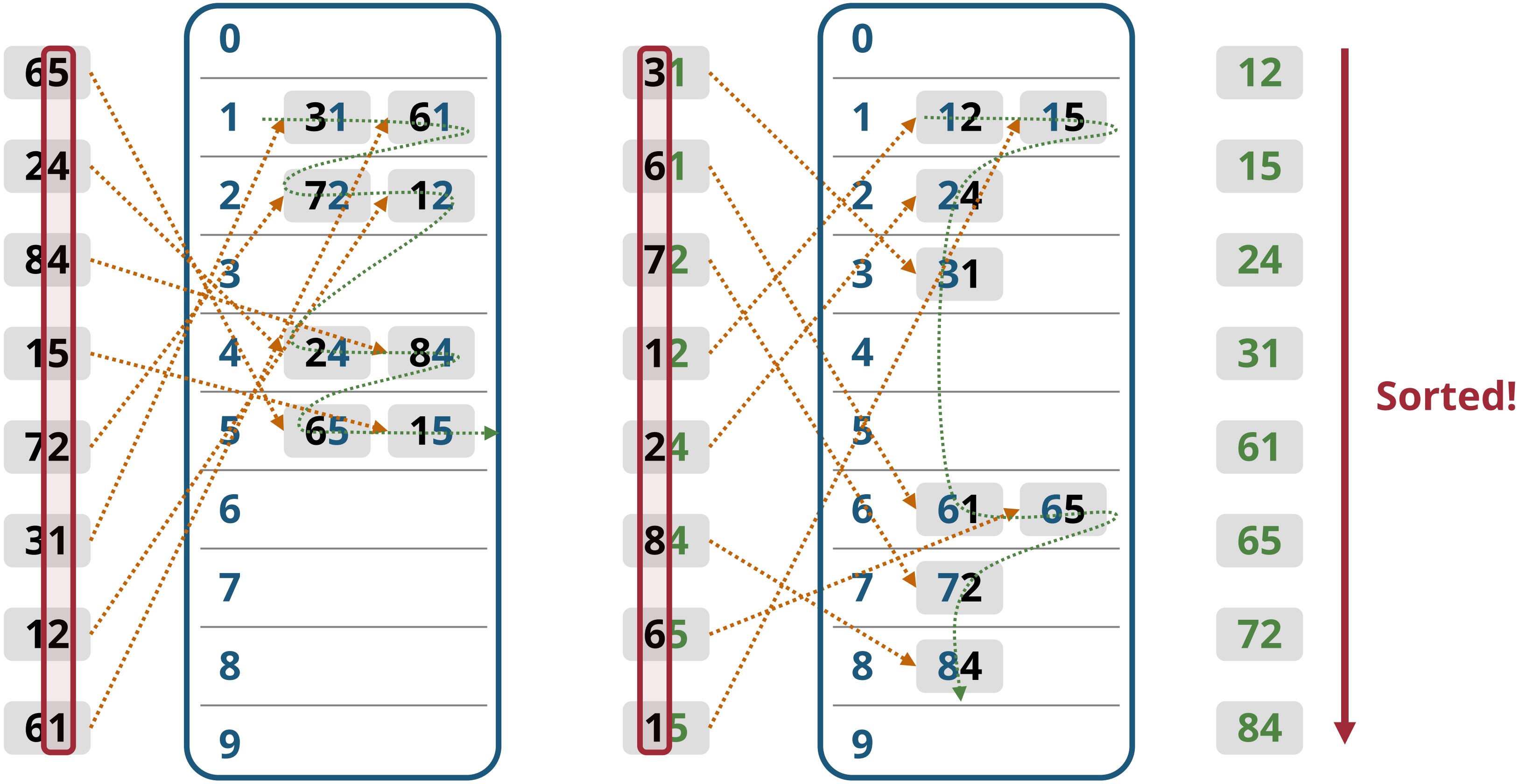
  - `sort()`
  - `sortReverse()`
  - `shuffle()`

Does not create a new copy

In-place vs Not-in-place Algorithm

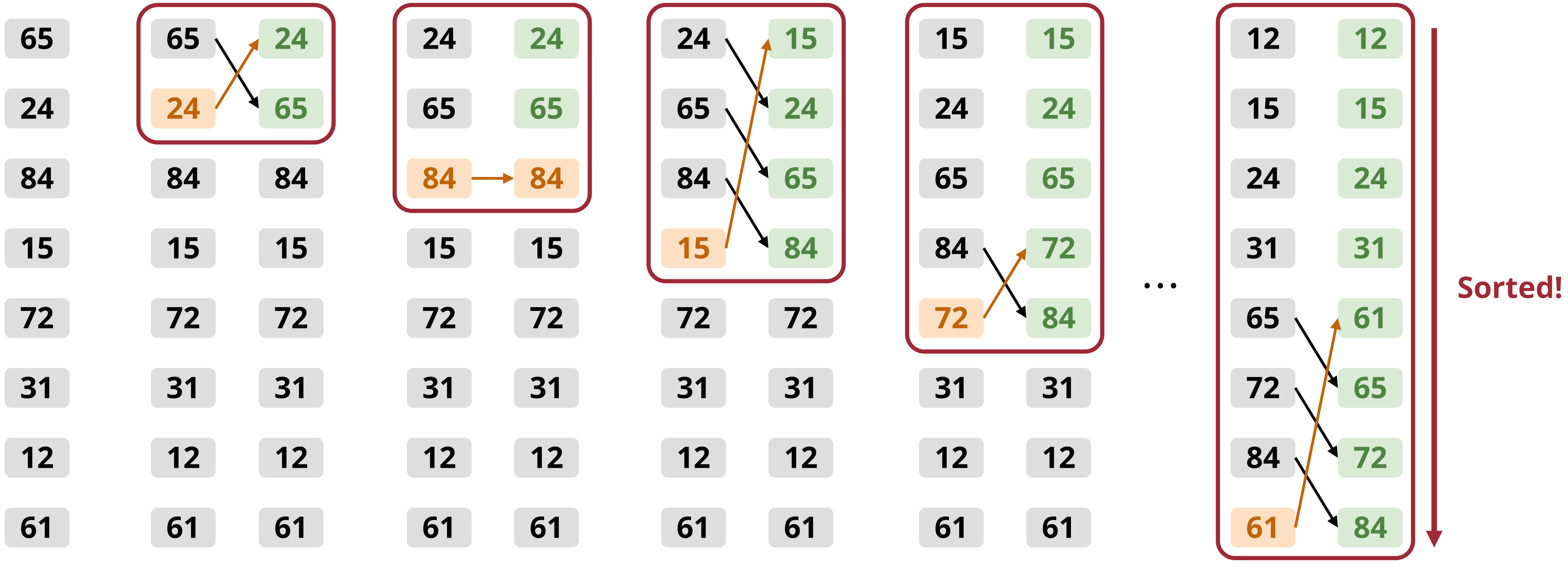
- In-place algorithm does not require additional memory
- Not-in-place algorithm needs extra memory to store intermediate results

Example of Not-in-place Algorithms: Radix Sort



Need extra memory to store intermediate results!

Example of In-place Algorithms: Insertion Sort



65	65	24	24	24	24	15	15	15		12	12
24	24	65	65	65	65	24	24	24		15	15
84	84	84	84	84	84	65	65	65		24	24
15	15	15	15	15	15	84	84	72	…	31	31
72	72	72	72	72	72	72	72	84		65	61
31	31	31	31	31	31	31	31	31		72	65
12	12	12	12	12	12	12	12	12		84	72
61	61	61	61	61	61	61	61	61		61	84

Sorted!

Don't need extra memory!

Sorting an Array vs Sorting a List

```
int[] arr = {3, 2, 1};
sort(arr);
println(arr);
```

```
[0] 3
[1] 2
[2] 1
```

sort(arr) returns a new sorted array

```
int[] arr = {3, 2, 1};
arr = sort(arr);
println(arr);
```

```
[0] 1
[1] 2
[2] 3
```

```
IntList li = new IntList();
li.append(3);
li.append(2);
li.append(1);
li.sort();
println(li);
```

```
IntList size=3 [ 1, 2, 3 ]
```

List.sort() returns the original list, sorted

List-to-Array Conversion

- **`List.toArray()`** converts a list into an array
- No easy way the other way around

Array vs List

	Array	List
Size		
Item data type		
Access speed		
Memory requirement		
Multi-dimensional		

Array vs List

	Array	**List**
Size	Fixed	Dynamic
Item data type	Same	Can be different
Access speed	Faster	Slower
Memory requirement	Low	High
Multi-dimensional	Possible	Not supported

List of Objects: ArrayList

- **ArrayList** is a list of objects

- What’s the difference?

Array of objects

```
Ball[] balls = new Ball[20];

String[] strs = new String[20];
```

ArrayList

```
ArrayList<Ball> balls = new ArrayList<Ball>()

StringList strs = new StringList()
```

Exercise: Bouncing Balls

- Add a ball when the mouse is clicked
- The new ball starts from where the mouse is
- Two approaches
  - Use an **array of objects**

```
Ball[] balls = new Ball[20];
```

  - Use an **ArrayList**

```
ArrayList<Ball> balls = new ArrayList<Ball>()
```

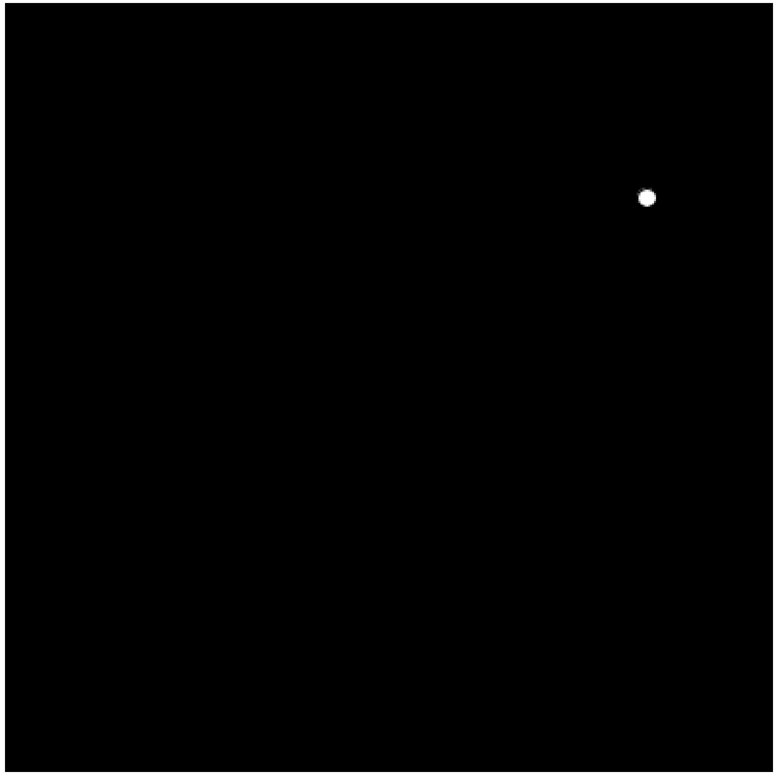
(Recap) Example: Bouncing Ball

```
class Ball {
  float size = 10;
  float speed = 5;
  float x, y, speedX, speedY;

  Ball() {
    // Constructor
  }

  void show() {
    // Show the ball
  }

  void move() {
    // Move the ball
  }

  void checkWalls() {
    // Check if the ball hit the walls
  }
}
```

Fields

Constructor

Methods



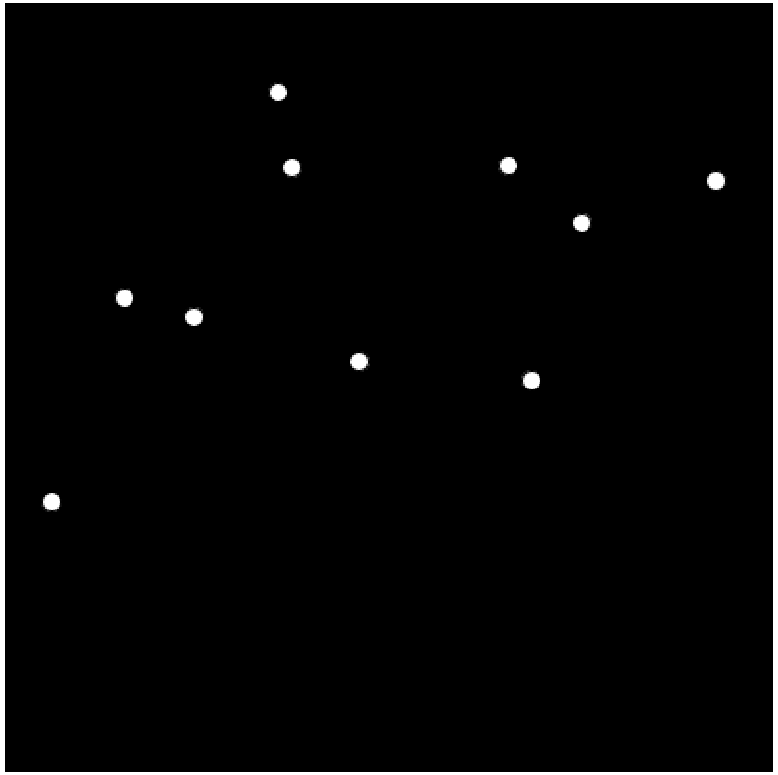
(Recap) Example: Bouncing Balls

```
Ball[] balls = new Ball[20];

void setup() {
  size(400, 400);

  for (int i = 0; i < balls.length; i++) {
    balls[i] = new Ball();
  }
}

void draw() {
  background(0);

  for (int i = 0; i < balls.length; i++) {
    balls[i].move();
    balls[i].checkWalls();
    balls[i].show();
  }
}
```

An array of objects

Initialization

Call the methods!



Array vs List

Array

```
float[] pos = new float[3];

void setup() {
  size(400, 400);

  pos[0] = 100;
  pos[1] = 200;
  pos[2] = 300;
}

void draw() {
  for (int i = 0; i < pos.length; i++) {
    circle(pos[i], 200, 50);
  }
}
```

Declaration

Initialization

Length of the array

List

```
FloatList pos = new IntList(); 

void setup() {
  size(400, 400);

  pos.append(100);
  pos.append(200);
  pos.append(300);
}

void draw() {
  for (int i = 0; i < pos.size(); i++) {
    circle(pos.get(i), 200, 50);
  }
}
```

Declaration

Initialization

Length of the list

(Recap) Array vs List

	Array	**List**
Size		
Item data type		
Access speed		
Memory requirement		
Multi-dimensional		

(Recap) **Exercise**: Bouncing Balls

- Add a ball when the mouse is clicked
- The new ball starts from where the mouse is
- Two approaches
  - Use an **array of objects**

```
Ball[] balls = new Ball[20];
```

  - Use an **ArrayList**

```
ArrayList<Ball> balls = new ArrayList<Ball>()
```