Week 5: Arrays

DCR2284

Announcement

- Assignment #1 : Progress Status Good?
- Individual Work.

Midterm Examination

- THREE Groups (Refer the list in OL)
- Group –A [8:30AM -10:00AM]
- Group – B [8:30AM -10:00AM]
- Group - C [8:30AM -10:00AM]
- Venue : OPEN LEARNING/MS TEAMS
- ----------

Submission:

1. Screen capture of code and outputs, paste in *.pdf file and submit in OL/Share Drive
2. Transfer your code to the answer booklet.

Arrays...

- Arrays
  - Why?
  - What is an Array?
  - Declaring and Creating an Array
  - Initializing and Array
  - Array Operations
  - Array Examples
  - Arrays of Objects

Why do we care (about arrays)?

- What if you have a whole bunch of cars (or aliens or balls or ???) bouncing around the screen?
  - How do we keep track of where each one is?
    - int car1X, car1Y, car2X, car2Y, car3X, car3Y……
- How about keeping them in a ‘table’ or a ‘grid’
- CarX table:

1	2	3	4	5	6	7	8	9	10

  - Each ‘column’ has a number
  - You can refer to CarX table, column 3 for Car3…
- Note that all of the values in the row must be the same data type

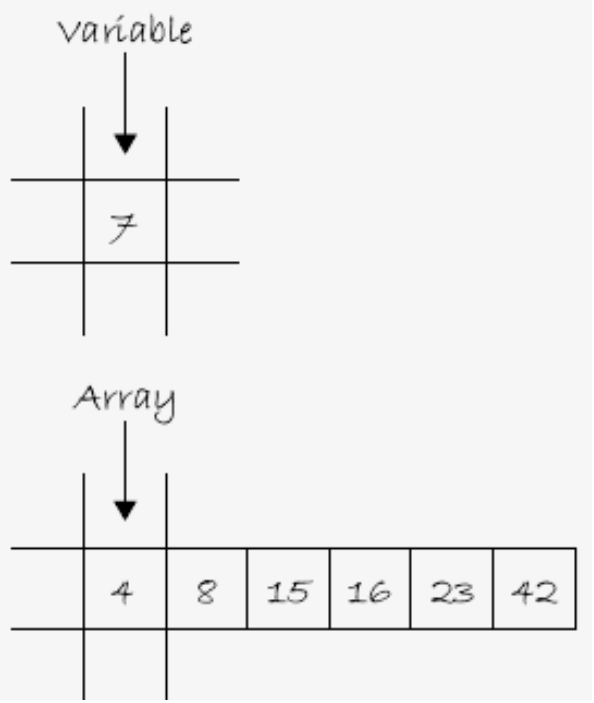
When to use an Array

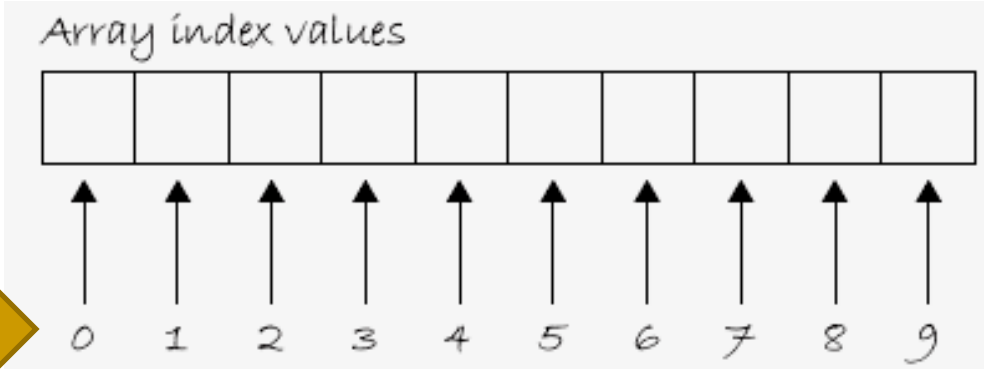
- Any time a program requires multiple instances of similar data, it might be time to use an array.
- For example, an array can be used to store:
  - The scores of four players in a game
  - A selection of 10 colors in a design program
  - A list of fish objects in an aquarium simulation
  - The days per month in a scheduling program

1	2	3	4	5	6	7	8	9	10	11	12
31	28	31	30	31	30	31	31	30	31	30	31

What is an array?

- A variable is a named location in memory
- An array is a named group of variables (of the same kind)
  - A ‘list’ of variables
- Each element of the array has a number called an ‘index’
- Starts at 0!





Why start at 0? Why not 1?

- Which of the following is why arrays start at 0?
  - A) Because programmers think that way
  - B) Because I say so
  - C) So that programmers can make 'the big bucks'
  - D) So you can use loops easily to 'walk' the array
- Here's a preview of adding up all the 'votes':

0	1	2	3
127	99	53	101

```
int sumArray(int size) {
  int sum = 0;
  for (int c = 0; c < size; c++) {
    sum = sum + votes[c];   // huh? [ ]?
  }
  return sum;
}
```

How to Declare an array (Step 1)

- Create the array variable (with the name and type)
  - Make a named 'list' with the following parts:

Type	Square Braces	Array name	semicolon
`int`	`[ ]`	`votes`	`;`

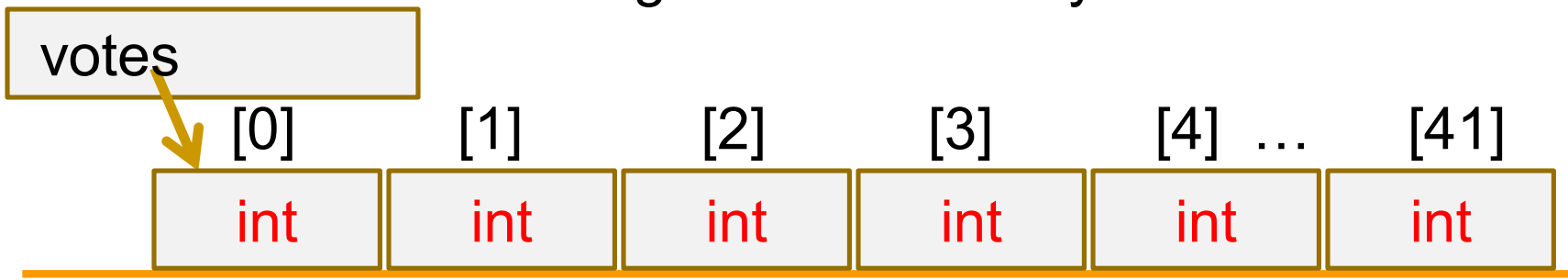
  - You are 'declaring' that
    - There is an array named `votes`
    - The elements inside are of type `int`
    - You have not (YET) declared how many are in inside
- Other Rules:
  - Arrays can be declared anywhere you can declare a variable
  - Same naming rules as variable names

How to Create an array (Step 2)

- Reserve memory for all of the elements in the list:

Array name		Keyword	Type	Size	semicolon
`votes`	`=`	`new`	`int`	`[42]`	`;`

  - You are reserving memory for:
    - The array named `votes`
    - Needs storage for `[42]` elements
    - Each element is of type `int`
    - Now you (and the compiler) know how many are in inside
    - You cannot change the size after you declare it!



Two Steps on one line:

- Declare and Create at the same time:

Type	Braces	Array name		Keyword	Type	Size	semi
`int`	`[]`	`votes`	`=`	`new`	`int`	`[42]`	`;`

  - You are 'declaring' that
    - There is an array named `votes`
    - The elements inside are of type `int`
  - You are reserving memory for the array
    - Needs storage for `[42]` elements
    - Each element is of type `int`
- Note that the size cannot be changed, so you should make enough room when you start!

Class Activity: Write the declarations for:

- Declare the array variable (step 1) for these:

30 integers	
100 floating point numbers	
56 Zoog objects	

Class Activity: Write the declarations for:

- Declare the array variable (step 1) for these:

30 integers	
100 floating point numbers	
56 Zoog objects	

```
// type [] arrayName ;
int [] arrayOfInts;
float [] hundredFloats;
Zoog [] zoogArmy;
```

A: Write the declarations for:

- Declare new items (step 2) for the following:

30 integers	
100 floating point numbers	
56 Zoog objects	

B: Write the declarations for:

- Declare new items (step 2) for the following:

30 integers	
100 floating point numbers	
56 Zoog objects	

```
// arrayName = new type[size];
arrayOfInts = new int[30];
hundredFloats = new float[100];
zoogArmy = new Zoog[56];
```

C: Write the declarations for:

- Both Steps 1 and 2 on the same line:

30 integers	
100 floating point numbers	
56 Zoog objects	

Exercise 1-3: Write the declarations for:

- Both Steps 1 and 2 on the same line:

30 integers	
100 floating point numbers	
56 Zoog objects	

```
// type [] arrayName = new type[size];
int [] arrayOfInts = new int[30];
float [] hundredFloats = new float[100];
Zoog [] zoogArmy = new Zoog[56];
```

D: What's wrong with these?

- May be valid, maybe not!

`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;` `Zoog[] zoogs = new Zoog[num * 10];`	

Initializing an Array

- The 'long' way

```
int [] stuff = new int[3];
stuff[0] = 8;
stuff[1] = 3;
stuff[2] = 1;
```

- The 'shortcut' way:

```
int [] stuff = { 8, 3, 1 };
```

  - The compiler counts the number of elements (3)
    - Sets the size automatically!
  - Fills the contents with the values inside the `{ };`
  - Values are separated by commas

Initializing with Iteration

- Rolling 5 dice (Example)
- The 'hard' way

```
int[] die = new int[5];
die[0] = (int)random(1,7);
die[1] = (int)random(1,7);
die[2] = (int)random(1,7);
die[3] = (int)random(1,7);
die[4] = (int)random(1,7);
```

- Using a while loop:

```
int n = 0;
while (n < 5) {
  die[n] = random(1,7);
  n = n + 1;
}
```

Initializing with Iteration

- Rolling 5 dice (Yahtzee) Example
- The 'hard' way

```
int[] die = new int[5];
die[0] = (int)random(1,7);
die[1] = (int)random(1,7);
die[2] = (int)random(1,7);
die[3] = (int)random(1,7);
die[4] = (int)random(1,7);
```

- Using a for loop:

```
for (int n = 0; n < 5; n++ ) {
  die[n] = (int)random(1,7);
}
```

Using a Constant for the array size

- Programming Tip:
  - Use a constant (**`final int`**) value for the size if it will not change during the program
  - If you re-write the game to add more dice, just change the constant in one place!

```
final int NUM_DICE = 5;
int[] die = new int[NUM_DICE];
for (int n = 0; n < NUM_DICE; n++ ) {
  die[n] = (int)random(1,7);
}
```

Ask the array how long it is!

- Programming Tip:
  - Use a the array `.length` value to determine the actual number of elements in an array
  - If you use this trick, you will never have to remember!

```
final int NUM_DICE = 5;
int[] die = new int[NUM_DICE];
for (int n = 0; n < die.length; n++ ) {
  die[n] = (int)random(1,7);
}
```

E: Loop/Array example:

- Using a for loop to initialize an array:

```
float[] values = new float[6];
for (int i = 0; i < values.length; i++)
   values[i] = 0;
```

  - Note the use of **`values.length`**

- Now write code to square each element:

```
int [] nums = { 5, 4, 2, 7, 6, 8, 5, 2, 6, 14 };
```

Square each number (i.e., multiply each by itself)	`for (int i ___; i < _____; i++) {`
	`  ____[i] = _____*_____;`
	`}`

F : Loop/Array continued:

```
int [] nums = { 5, 4, 2, 7, 6, 8, 5, 2, 6, 14 };
```

Add a random number between zero and 10 to each number.	`_____ += int(_____);`
Add to each number the number that follows in the array. Skip the last value in the array.	`for (int i = 0; i < _____; i++) { _____ += _____ [____]; }`
Calculate the sum of all the numbers.	`_____ _____ = ____; for (int i = 0; i < nums.length; i++) { _____ += _____; }`

Walking off the end of the array…

- A very common problem is trying to access an element in the array that doesn't exist:

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

  - Remember that we start at element 0
  - What is the index of the last element? _____
  - What happens if we try to access element [10]?

```
for (int i = 0; i <= 10; i++ ) {
  xpos[i] = 0;
}
```

```
Exception in thread "Animation Thread"
java.lang.ArrayIndexOutOfBoundsException: 10
      at OutOfBounds.setup(OutOfBounds.java:21)
```

Arrays Lab: Preliminary to Snake Example

- Declare and initialize two 'global' arrays:

```
int[] xpos = new int[10];
int[] ypos = new int[10];
```

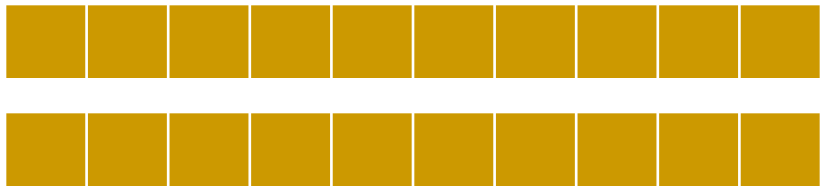
- Initialize them in setup()

```
for (int i = 0; i < xpos.length; i++ ) {
  xpos[i] = 0;
  ypos[i] = 0;
}
```

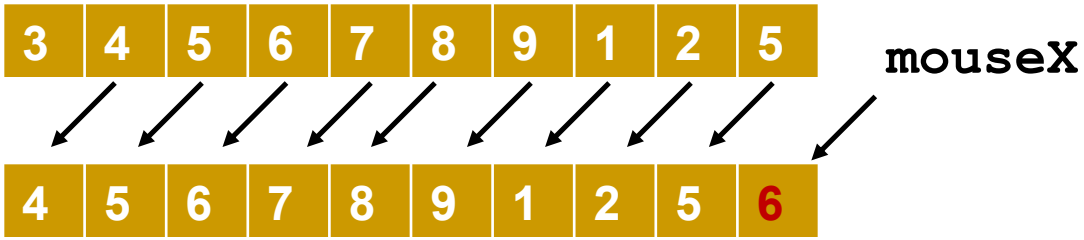
- Each time the mouse is clicked, store the mouseX and mouseY into the next available spot in the array (fill from left to right)
- Each time through draw, display a line between each location saved in the array.

Snake Example

- Track the last 50 locations of the mouse with two arrays:
  - xpos:
  - ypos:



- Each time through draw, move all elements to the left one spot, and add the newest to the last



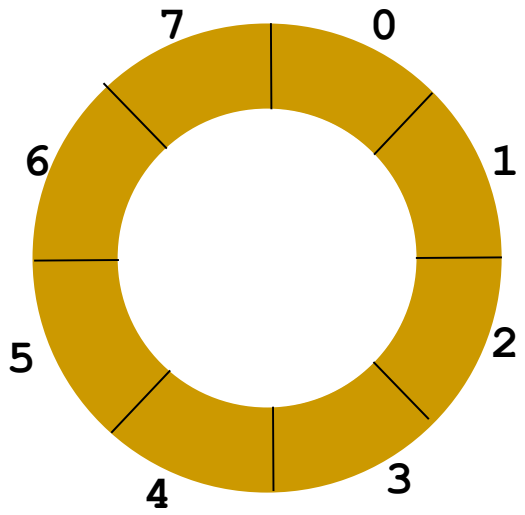
- Then draw all elements

Snake Example Code draw( ) method

```
void draw() {
  background(255);
  for (int i = 0; i < xpos.length-1; i++) {
    xpos[i] = xpos[i + 1];
    ypos[i] = ypos[i + 1];
  }
  xpos[xpos.length-1] = mouseX;
  ypos[ypos.length-1] = mouseY;
  for (int i = 0; i < xpos.length; i++) {
    noStroke();
    fill(255-i*5);
    ellipse(xpos[i],ypos[i],i,i);
  }
} // end draw
```

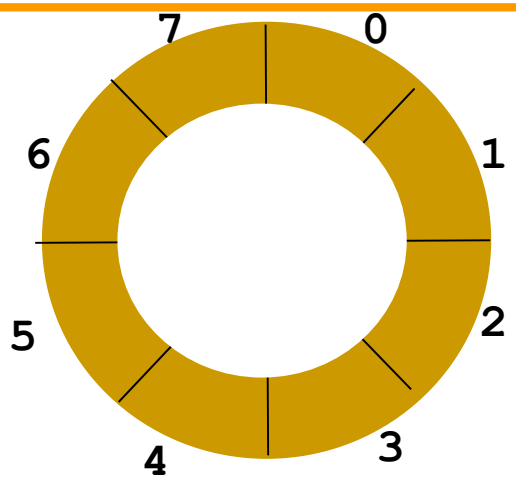
- Clear screen
- Move all left
- Save new loc.
- Draw all

Extra: Circular Arrays



- Efficiency Trick (not in text)
- Moving 50 (*2) items is work
  - Every time through draw
  - May slow your program down!
- The idea of a circular array is that the end of the array "wraps around" to the start of the array
- The `mod` operator (%) can be used to calculate the next 'spot' of the circular array
  - `next = (next + 1) % 8;`
- The result of % 8 is a number between 0 and 7
- Our counter would go from 0 to 7 then back to 0

Extra: Snake Circular Array

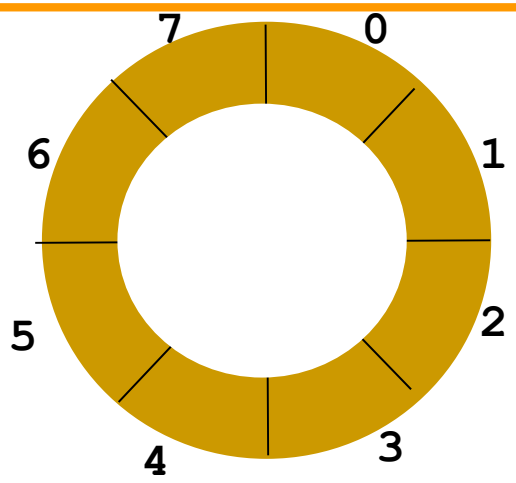


- Setup constant ‘MAX’
- Keep track of two indexes:
  - first (first to draw)
  - last (last to draw)

```
void draw() {
  background(255);
  last = (last + 1) % MAX;
  if (last == first) {
    first = (last + 1) % MAX;
  }
...
```

- Both start at 0
- In draw
  - Add 1 to last
  - Keep first and last in range
  - If we have a collision (last == first)
    - Set first to one ‘after’ last (last + 1), and keep in range

Extra: Snake Circular Array



- Example Run

first	last
0	0
0	. . .
0	7
1	0
2	1
3	2
4	3
5	4
6	5
7	6
0	7

```
void draw() {
  background(255);
  last = (last + 1) % MAX;
  if (last == first) {
    first = (last + 1) % MAX;
  }
  println("First: " + first +
          " Last: " + last);
  . . .
```

Extra: Snake Circular Array

```
void draw() {
  int next;

  xpos[last] = mouseX;
  ypos[last] = mouseY;



  for (int i = 0; i < MAX; i++) {
    noStroke();
    fill(255-i*5);
    next = (first + i) % MAX;
    ellipse(xpos[next],ypos[next],i,i);
  }
} // end draw
```

- Setup a new variable 'next'
  - Holds the next slot to display
- Update the last spot in the array with the mouse location.
  - Starts at first
- Loop 50 times
  - Same stroke and fill
  - Set next to 'wrap' around
  - Draw each ellipse

Additional Processing Array features

- Processing provides a set of functions to help manage arrays (covered):
  - `shorten()`
  - `concat()` -- Puts two arrays together
  - `subset()`
  - `append()`
  - `splice()`
  - `expand()` – Make an array larger in size!
  - `sort()` – Sort all the elements in the array
  - `reverse()` – Reverse the order of all elements

Arrays of Objects

- Using arrays to store multiple objects is a very powerful programming feature
- We will use the Car class , and create a screen full of unique cars
- We will use an array to create a numbered set of ‘parking stalls’ for a bunch of Cars.

The 'Car' Class Revisited

```
class Car { // Define a class below the rest of the program.
  color c;        // Variables.
  float xpos, ypos, xspeed;

  // Constructor with three parameters
  Car(color col, float xp, float yp, float xspd) {
    c = col;
    xpos = xp;
    ypos = yp;
    xspeed = xspd;
  }

  void display() { // Function
    // The car is just a square
    rectMode(CENTER);
    stroke(0);
    fill(c);
    rect(xpos,ypos,20,10);
  }

  void move() { // Function
    xpos = xpos + xspeed;
    if (xpos > width) {
      xpos = 0;
    }
  }
}
```

Create a parking lot with numbered stalls

- You can use the 'Car' class just like any other type
  - Goal: Declare an array of our new Cars object:

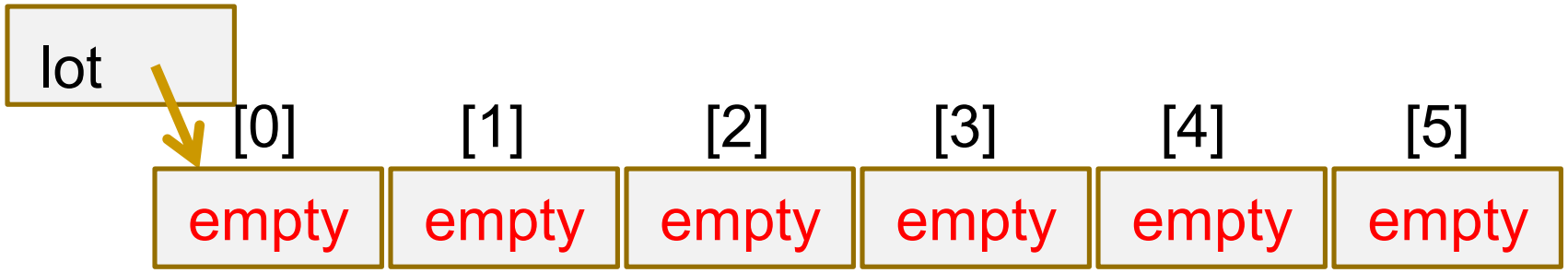
```
Car [] parkingLot;              // A sign with an arrow
setup() {
   parkingLot = new Car[10]; // 10 Car size stalls
}
```

- But wait… what just happened?
  - Did you create 10 cars? No, not yet.
  - You created a parking lot with 10 stalls (0-9) for Cars
  - So we still have to 'build' the cars, park them in stalls, and set all of their colors, locations and speeds…

```
parkingLot[0] = new Car(color, x, y..);
```

Making the parking lot…

- Step 1: Make the reference to the lot
  - Type Square Braces Array name semicolon
  - `Car [] lot ;`
- Step 2: Make the parking stalls
  - Array name Keyword Type Size semicolon
  - `lot = new Car [100] ;`
- Or the 'one-liner' version of Steps 1 and 2
  - Type [ ] Array name Keyword Type[Size] semi
  - `Car [] lot = new Car[100] ;`



Filling the parking lot the easy way!

- Once you have the parking lot created,

```
Car [ ] parkingLot;
setup() {
   parkingLot = new Car[10];
}
```

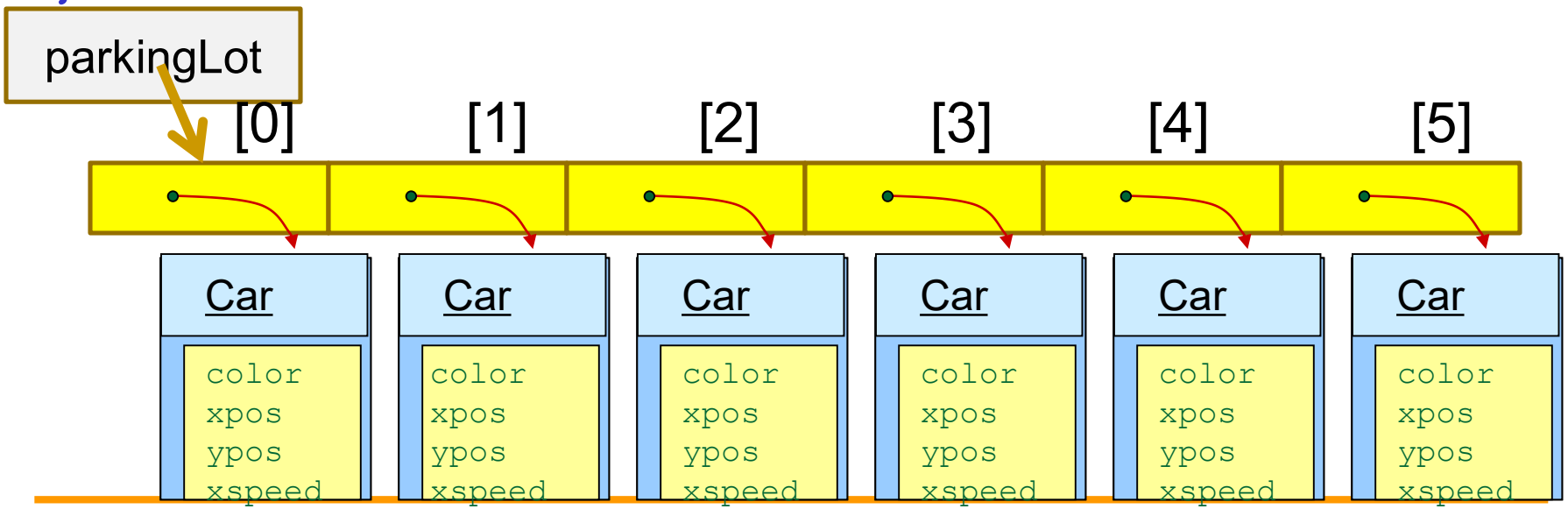
- Use a for loop to make a bunch of cars!

```
for (int i; i < 10; i++) {
  parkingLot[i] = new Car(color, x, y..);
}
```

  - Arrays and loops work wonderfully together!
  - And they are even more fun with objects!

Now Let's play General Motors!

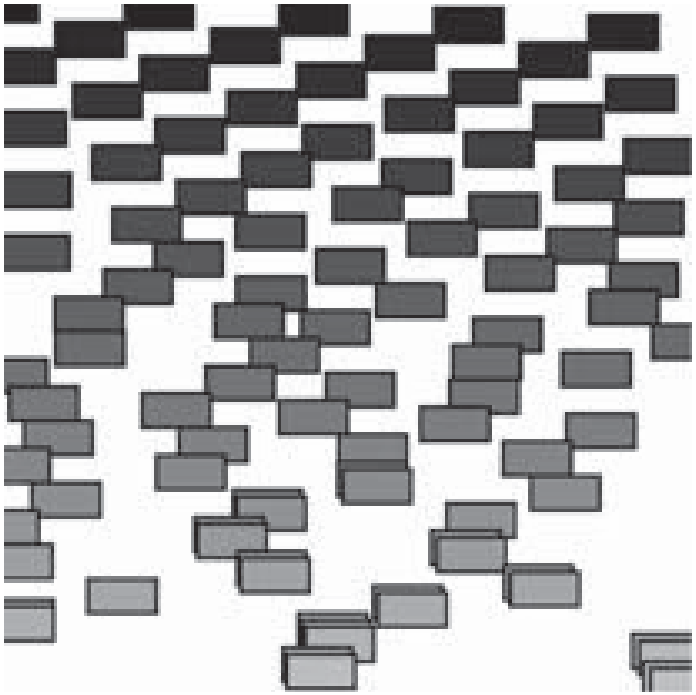
```
Car [] parkingLot;
void setup() {
  size(200,200);
  parkingLot = new Car[100]; // Car size stalls
  for (int i = 0; i < parkingLot.length; i++) {
    parkingLot[i] = new Car(color(i*2),0,i*2,i/20.0);
  }
}
```



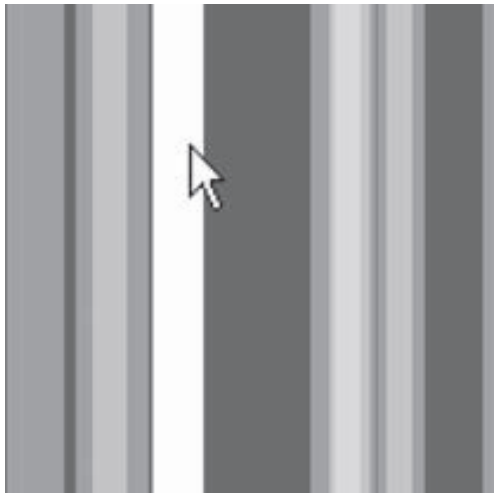
And let's get them on the road!

- Use a loop to move and display them!

```
void draw() {
  background(255);
  for (int i = 0; i < parkingLot.length; i++) {
    parkingLot[i].move();
    parkingLot [i].display();
  }
}
```



Interactive Objects



- : Interactive Stripes
  - Array of Vertical Stripe Objects

```
Stripe[] stripes = new Stripe[10];

void setup() {
  size(200,200);
  for (int i = 0; i < stripes.length; i++) {
    stripes[i] = new Stripe();
  }
}
```

- Each stripe moves from left to right at varying speeds
  - If it arrives at the right,
- Stripes are transparent (if they overlap, they are lighter)
- If the mouse is over the stripe, turn it opaque white

Example : Stripe Object

```
class Stripe {
  float x, speed, w;
  boolean mouse;

  Stripe() {
    x = 0;
    speed = random(1);
    w = random(10,30);
    mouse = false;
  }
  void display() {
    if (mouse) {
      fill(255);       // opaque white
    } else {
      fill(255,100);   // transparent
    }
    noStroke();
    rect(x,0,w,height);
  }
  void move() {
    x += speed;
    if (x > (width + 20)) x = -20;
  }
```

- Instance variables
  - Location (x)
  - speed
  - Width (w)
- Setup in constructor
- Methods
  - display()
  - move()
  - rollover()
    - Sets mouse true if over stripe

```
void rollover(int mx, int my) {
  if ( mx > x && mx < (x + w) ){
    mouse = true;
  } else {
    mouse = false;
  }
}
```

Exercise: Button Over

- Write a Button class that changes color when the mouse is over it
  - Similar to Week 2b (non-object version) with rollover
  - Assume the button is not on when created
- Create many button objects and store in an array
  - Different sizes and locations (pass to constructor)

```
class Button {
  float x, y, w, h;
  boolean on;

  Button(float tX, float tY, float tW, float tH) {
    x = tX;
    y = tY;
    w = tW;
    h = tH;
    on = __________;
  }
. . .
```

Processing's Array Functions

- Java arrays are fixed size
  - Cannot grow beyond size at run time
  - length (balls.length) returns the maximum size
- Processing provides a handy set of functions to manage arrays
  - shorten()
  - concat()
  - append()
  - splice()
  - expand()
  - sort()
  - reverse()

```
void mousePressed() {
  // Add a new ball object
  Ball b = new Ball(mouseX,mouseY,10);
  // Append to array
  balls = (Ball[]) append(balls,b);
}
```

See processing reference for more details and examples

Summary

- Arrays make it easier to store ‘groups’ or ‘lists’
  - 1) Must be exactly the same type inside
  - 2) Can be any length, but they do have an end!
  - 3) Each item has a index number (starting from 0)
- Setting up arrays is a two step process:
  - 1) Declare the array (set name, type inside)
  - 2) Create the array (sets length, uses ‘`new`’)
  - Can be done on one line though!
- Arrays can be initialized: `{ 1, 17, 28 }`
- For loops and arrays are natural partners
  - Initializing all elements
  - ‘Walking’ through the array
- Access array elements with `[index]`