Corso di Laurea Magistrale in Design, Comunicazione Visiva e Multimediale - Sapienza Università di Roma

Interaction Design
A.A. 2017/2018

8 – Loops and Arrays in Processing

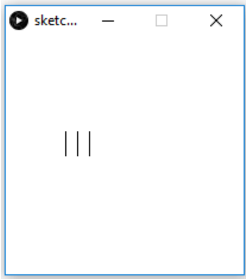
Francesco Leotta, Andrea Marrella

Last update : 19/4/2018

What is iteration?

- Iteration is the process of **repeating a set of steps over and over again**.
  - Suppose we want to draw 3 lines starting from *x coor* = 50 pixels with one line every 10 pixels.

```
void setup() {
  size(200,200);
  background(255);
}
void draw() {
  stroke(0);
  int y = 80; // Vertical location of each line
  int x = 50; // Initial horizontal location for first line
  int spacing = 10; // How far apart is each line
  int len = 20; // Length of each line
  line(x,y,x,y+len);
  x = x + spacing;
  line(x,y,x,y+len);
  x = x + spacing;
  line(x,y,x,y+len);
  x = x + spacing;
}
```



Draw the first line.

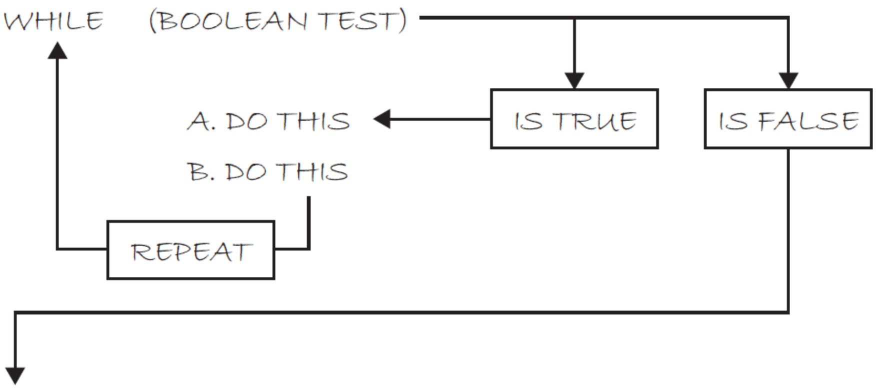
Add spacing so the next line appears 10 pixels to the right.

Continue this process for each line, repeating it over and over.
But what happens if we have to draw 100 lines?

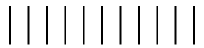
What is iteration?

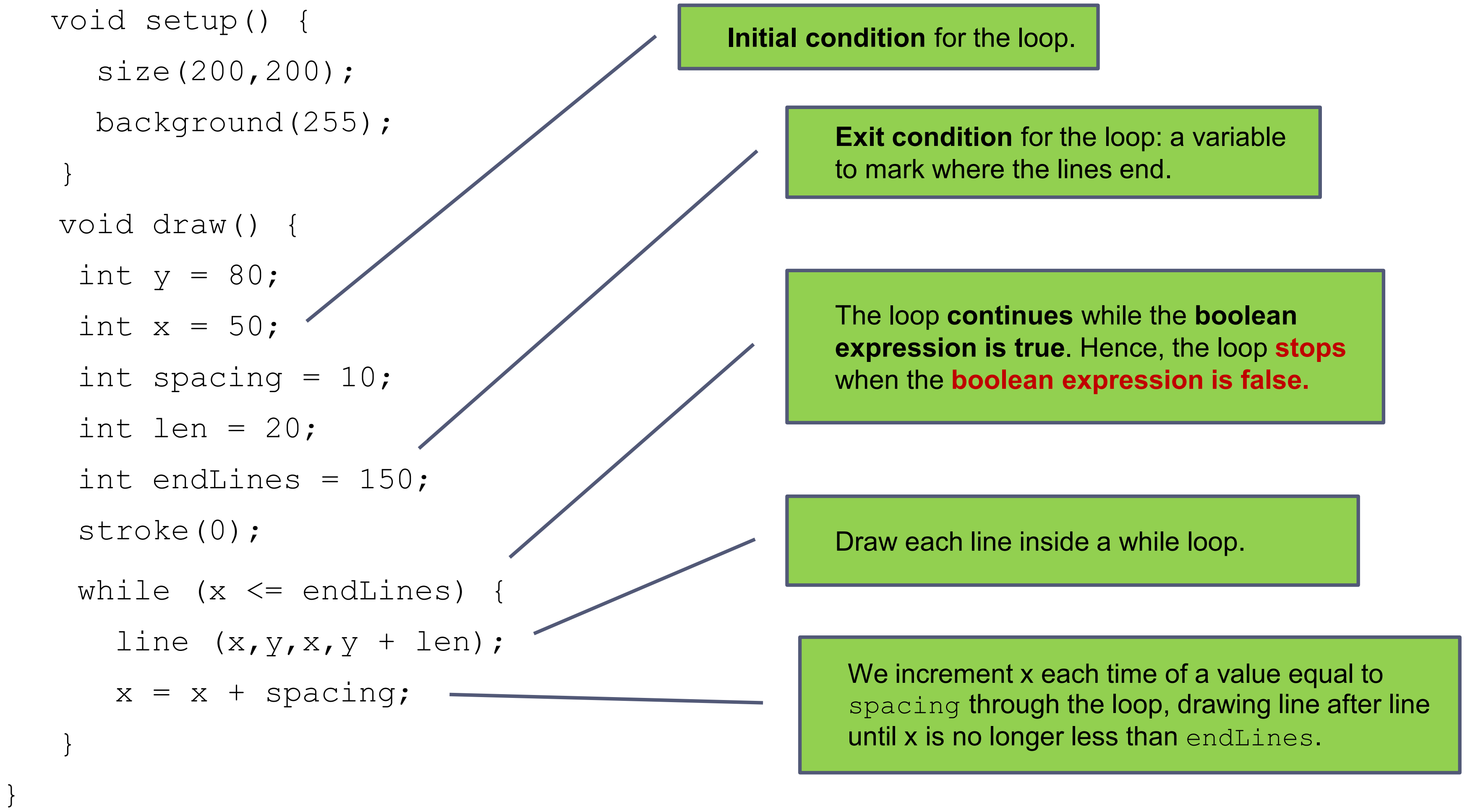
- Instead of repeating the same command over and over, we want to say something like: ***draw one line one hundred times***. This dilemma it is easily solved with a control structure called the **loop**.
- A loop structure **will ask a yes or no question** to determine ***how many times*** a block of code should be repeated. This is known as **iteration**.
- There are two main types of loops, the `while` loop and the `for` loop.
  - A `while` loop employs a **boolean test condition**. If the test evaluates to *true*, the instructions enclosed in curly brackets are executed; if it is *false*, we continue on to the next line of code.

```
while (boolean test condition) {
   // The instructions inside the while block continue to be executed
  // over and over again until the test condition becomes false.
   }
```



The `while` loop





```
  void setup() {
     size(200,200);
     background(255);
   }
   void draw() {
    int y = 80;
    int x = 50;
    int spacing = 10;
    int len = 20;
    int endLines = 150;
    stroke(0);
    while (x <= endLines) {
       line (x,y,x,y + len);
       x = x + spacing;
   }
}
```

Initial condition for the loop.

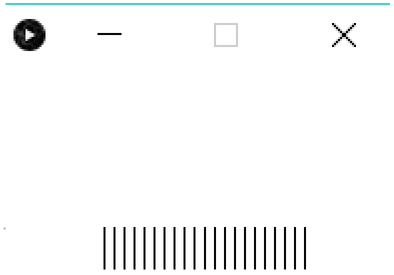
Exit condition for the loop: a variable to mark where the lines end.

The loop **continues** while the **boolean expression is true**. Hence, the loop **stops** when the **boolean expression is false.**

Draw each line inside a while loop.

We increment x each time of a value equal to `spacing` through the loop, drawing line after line until x is no longer less than `endLines`.

The `while` loop



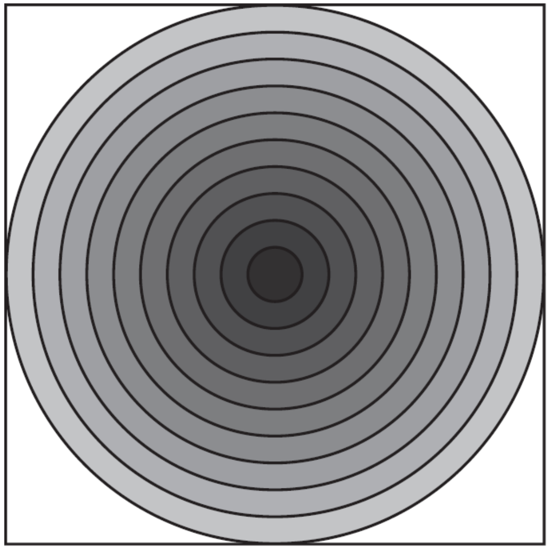
```
   void setup() {
     size(200,200);
     background(255);
    }
    void draw() {
     int y = 80;
     int x = 50;
     int spacing = 5;
     int len = 20;
     int endLines = 150;
     stroke(0);
     while (x <= endLines) {
       line (x,y,x,y + len);
       x = x + spacing;
    }
}
```

A smaller spacing value results in **more lines** that are **closer together**.

Exercise 1 - *Concentric Circles*

- Complete the following code to recreate the below screenshot:

```
float x = 100;
float y = 100;
float w = 200;
float h = 200;
void setup() {
  size(200,200);
  background(255);
}
void draw() {
 while (________) {
   stroke(0);
   fill(______);
   ellipse(___,___,___,____);
   __ = __-20;
   __ = __-20;
}
}
```

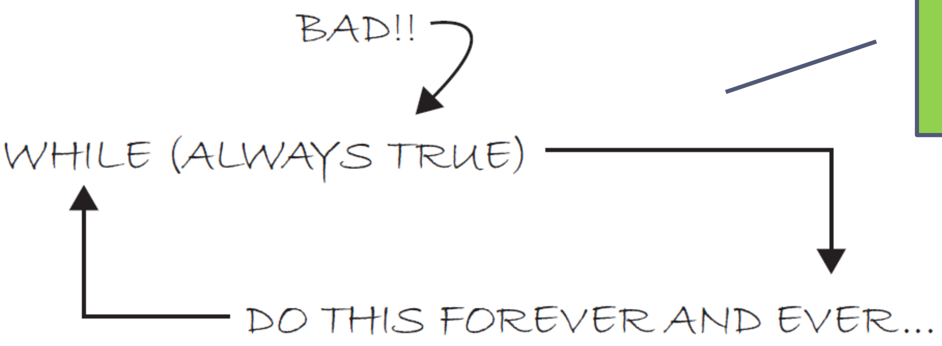


Solution of Exercise 1

```
float x = 100;
float y = 100;
float w = 200;
float h = 200;
void setup() {
  size(200,200);
  background(255);
}
void draw() {
 while (w>=0) {
   stroke(0);
   fill(w);
   ellipse(x,y,w,h);
   w = w-20;
   h = h-20;
}
}
```

Exit conditions

- When we use a loop, we **must make sure** that the **exit condition for the loop will eventually be met**!



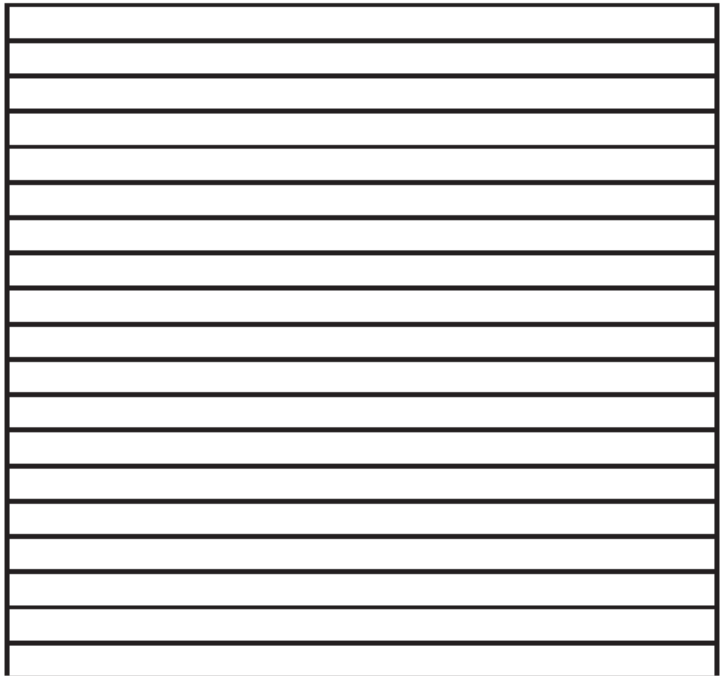
Processing **will not give you an error** should your exit condition never occur.

```
int x = 0;
while (x < 10) {
 println(x);
 x = x - 1;
}
```

Example of **infinite loop**! This program never ends! **Don't do this!**

Exercise 2 – *Multiple Lines*

- Write the Processing code to recreate the below screenshot:



Solution of Exercise 2

```
float x1 = 0;
float x2 = 200;
float y = 10;

void setup() {
  size(200,200);
  background(255);
}
void draw() {
 while (y<=height) {
   stroke(0);
   line(x1,y,x2,y);
   y = y+10;
}
```

The `for` loop

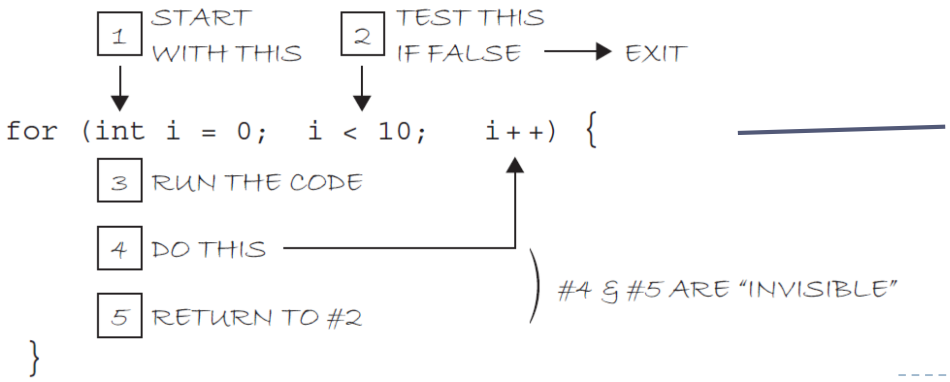
- A useful shortcut of `while` loop, to be used where one value is incremented repeatedly, is the `for` loop.

Initialization: a variable is *declared* and *initialized* for use within the body of the loop. It often acts as a **counter**.

Boolean test: any Boolean expression that evaluates to true or false

Iteration expression: instruction that happens at the end of any loop cycle. It usually increments the variable used as a counter

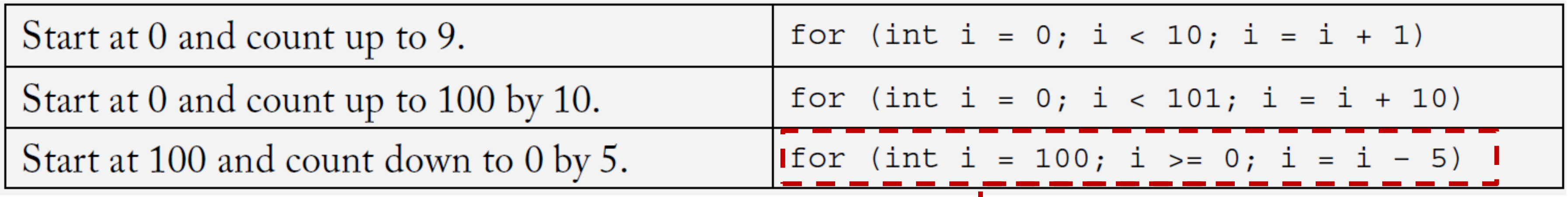
```
for(initialization; boolean test; iteration expression) {
 // The instructions inside the for block continue to be executed
    over and over again until the test condition becomes false.
}
```



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

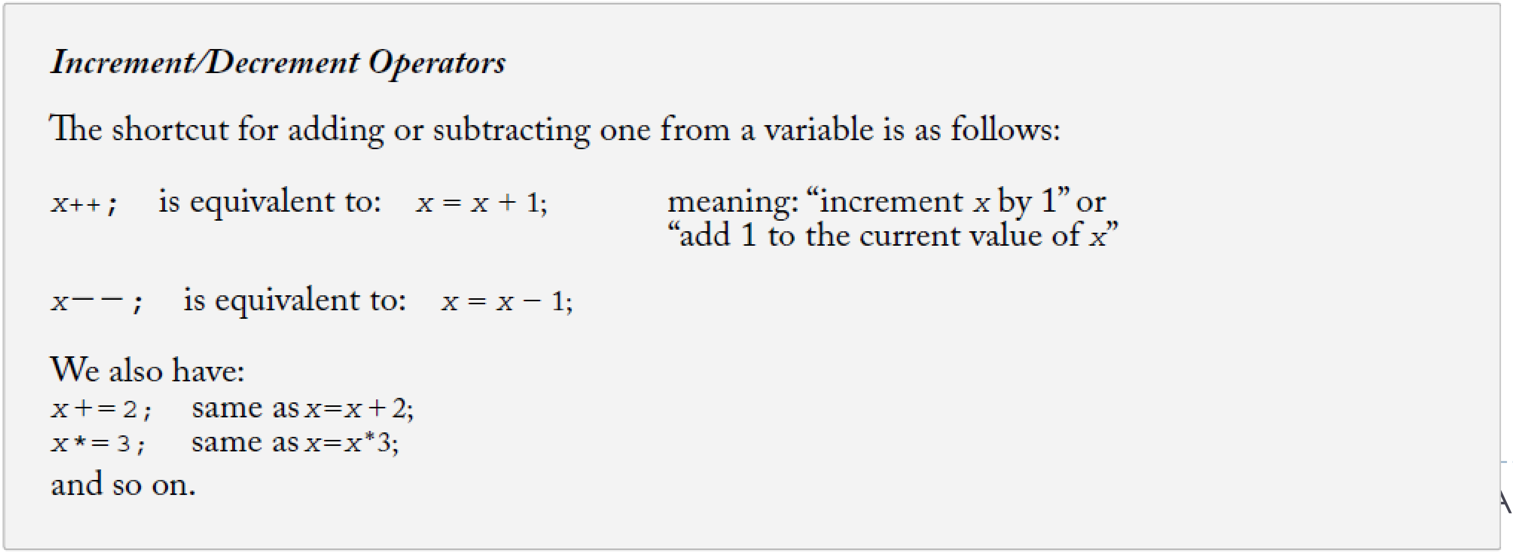
A `for` loop can have its own **local variable** just for the purpose of counting.

The `for` loop



Start at 0 and count up to 9.	`for (int i = 0; i < 10; i = i + 1)`
Start at 0 and count up to 100 by 10.	`for (int i = 0; i < 101; i = i + 10)`
Start at 100 and count down to 0 by 5.	`for (int i = 100; i >= 0; i = i - 5)`

- To the machine, it means the following:
  - Declare a variable i.
  - Set its initial value to 100.
  - While i is greater or equal than 0, repeat the internal code of the loop.
  - At the end of each iteration, decrement i of 5.



Increment/Decrement Operators

The shortcut for adding or subtracting one from a variable is as follows:

`x++;` is equivalent to: $x = x + 1;$ meaning: "increment x by 1" or "add 1 to the current value of x"

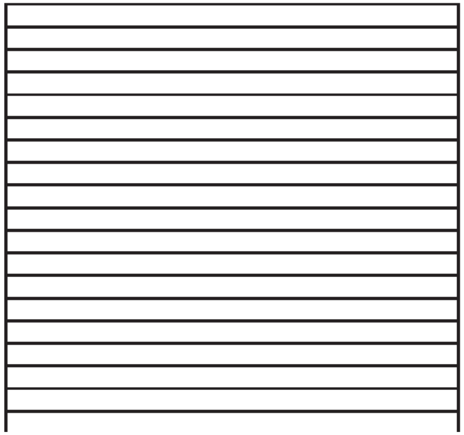
`x--;` is equivalent to: $x = x - 1;$

We also have:
`x+=2;` same as $x=x+2;$
`x*=3;` same as $x=x*3;$
and so on.

Exercise 3

- Complete the following code to recreate the below screenshot:

```
float x1 = 0;
float x2 = 200;

void setup() {
  size(200,200);
  background(255);
}
void draw() {
 for (________;_________;________) {
   stroke(0);
   line(x1,_____,x2,_____);
}
}
```

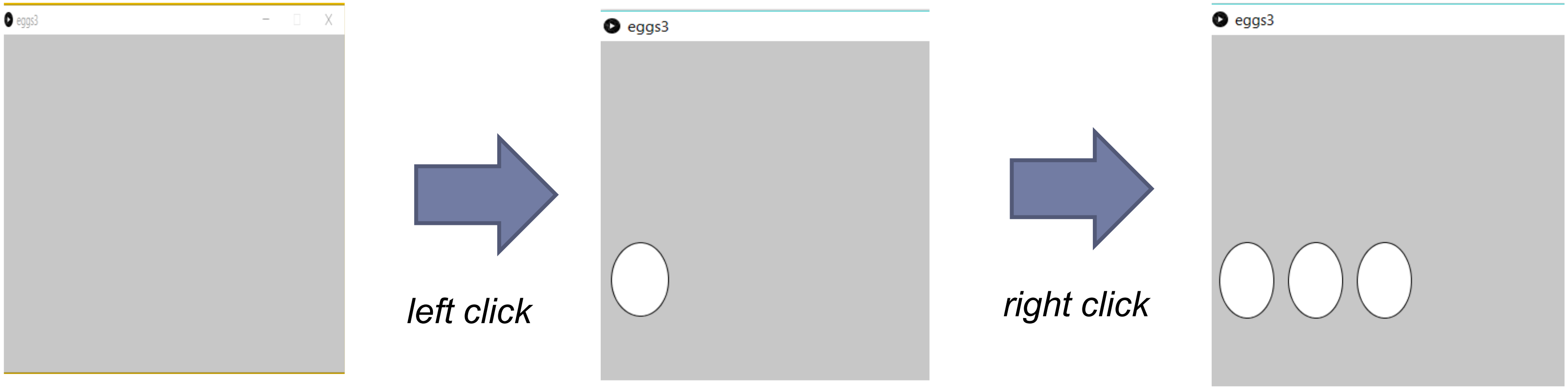


Solution of Exercise 3

```
float x1 = 0;
float x2 = 200;

void setup() {
  size(200,200);
  background(255);
}
void draw() {
 for (int y = 10; y < height; y = y+10) {
   stroke(0);
   line(x1,y,x2,y);
   }
}
```

Exercise 4 – *Drawing Eggs*

- Write a program to draw eggs in sequence using the `for` construct, by representing the following behavior:
- When the left mouse is clicked, add one egg to the sequence.
- When the right mouse is clicked, add two eggs to the sequence.



Solution of Exercise 4

```
// a variable to record the x coordinate value of any ellipse
int x;

// a variable to keep the number of eggs
int bowl;

void setup() {
  size(640, 360);
  background(199);
  fill(255);
  x = 10; // initialize the variable x
  bowl = 0; // initialize the variable bowl
}
void draw() {}
```

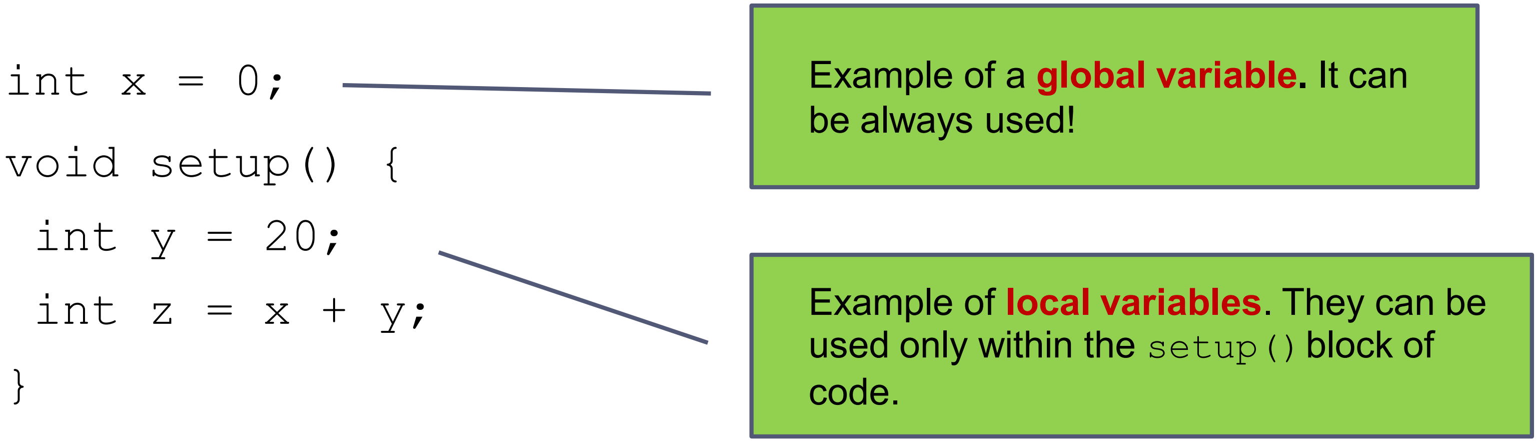
…continue…

Solution of Exercise 4

```
void mouseClicked() {
  // when the mouse is clicked, increase of one the number
  // in the variable bowl
    if(mouseButton == 37) {
       bowl = 1;
      }
    else if (mouseButton == 39) {
      bowl = 2;
      }
  // Draw as many eggs as those in the variable bowl
  for (int i = 0; i < bowl; i++) {
    ellipse(x, 250, 55, 77);
    x += 70;
  }
}
```

Local VS Global Variables

- Until now, any time that we have used a variable, we have declared it at the top of our program above `setup()`.
- Such variables are called **global variables**.
  - They can be used in *any line of code* anywhere in the program.
- **Local variables** are declared *within a block of code* (for example, in the definition of a function like `setup()` or `draw()`, or in a `if` statements, `while` and `for` loops.
  - A local variable declared within a block of code is **only available for use inside that specific block of code** where it was declared.



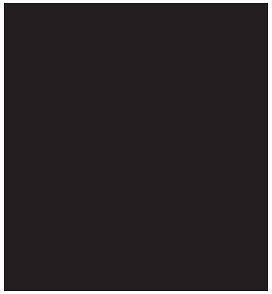
```
int x = 0;
void setup() {
 int y = 20;
 int z = x + y;
}
```

Example of a **global variable**. It can be always used!

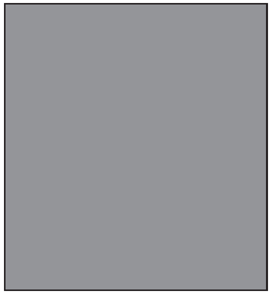
Example of **local variables**. They can be used only within the `setup()` block of code.

Exercise 5

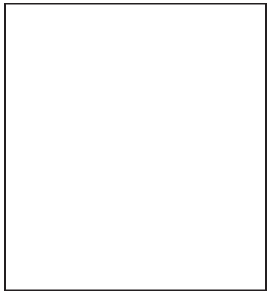
- Predict the results of the following two programs after 100 frames.



A



B



C

```
//SKETCH #1: Global
"count"
int count = 0;

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

void draw() {
  count = count + 1;
  background(count);
}
```

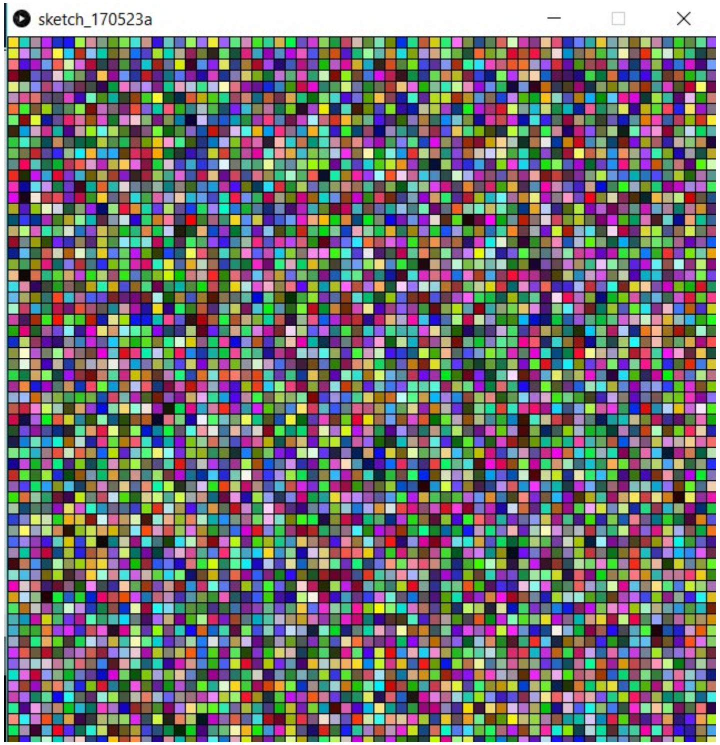
____

```
//SKETCH #2: Local
"count"
void setup() {
  size(200,200);
}

void draw() {
  int count = 0;
  count = count = 1;
  background(count);
}
```

____

Exercise 6 – *Coloured Grid*

- Create a grid of squares (each colored randomly) using a `for` loop inside the `draw()` function. Once designed, the colors should never change.



Solution of Exercise 6 (one row per frame)

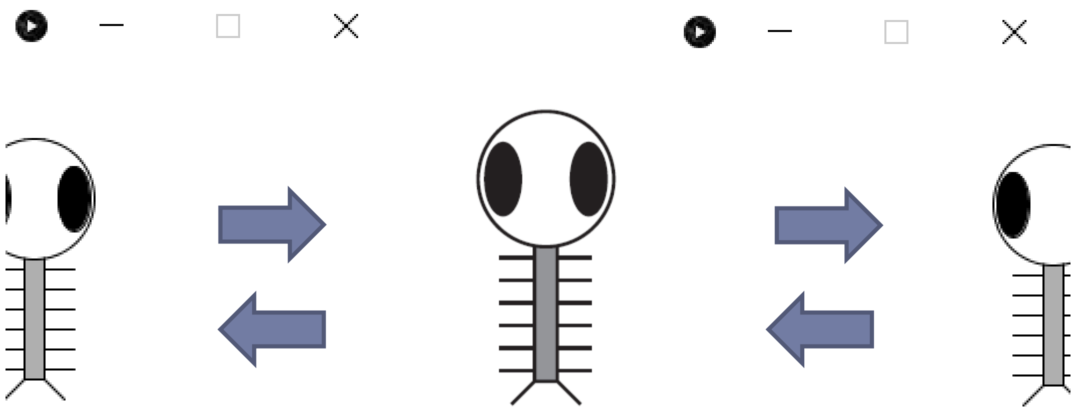
```
int y = 0;
int w = 10;
int h = 10;

void setup() {
 size(640,640);
}

void draw(){

 float r = 0;
 float g = 0;
 float b = 0;
for(int x=0;x<width;x=x+10) {
   r = random(0,255);
   g = random(0,255);
   b = random(0,255);
   fill(r,g,b);
   rect(x,y,w,h);
 }


  y+=10;
  y = constrain(y,0,height);
}
```

Solution of Exercise 6 (all rows together)

```
boolean finished = false;
int w = 10;
int h = 10;

void setup() {
 size(640,640);
}

void draw() {

  float r = 0;
  float g = 0;
  float b = 0;

if(!finished) {
   for(int y=0;y<height;y=y+10) {
     for(int x=0;x<width;x=x+10) {
       r = random(0,255);
       g = random(0,255);
       b = random(0,255);
       fill(r,g,b);
       rect(x,y,w,h);
     }
   }
   finished=true;
 }
}
```

Exercise 7 – *Bouncing Alien with arms*

- Redesign the bouncing alien in order to add a series of line to its body, resembling arms, like in the figure.



Solution of Exercise 7

```
int x = 100;
int y = 100;
int w = 60;
int h = 60;
int eyeSize = 16;
int speed = 1;

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

void draw() {
  // Change the x location of the alien by speed
  x = x + speed;
```

…continue…

Solution of Exercise 7

```
// If we reach an edge, reverse speed (i.e. multiply it by -1)
//(Note if speed is a + number, square moves to the right,- to
the left)


if ((x > width)||(x < 0)) {
 speed = speed * -1;
}
background(255);
// Set rects to CENTER mode
rectMode(CENTER);

// Draw alien's arms with a for loop
for (int i = y + 5; i < y + h; i += 10) {
  stroke(0);
  line(x-w/3,i,x + w/3,i);
}
```

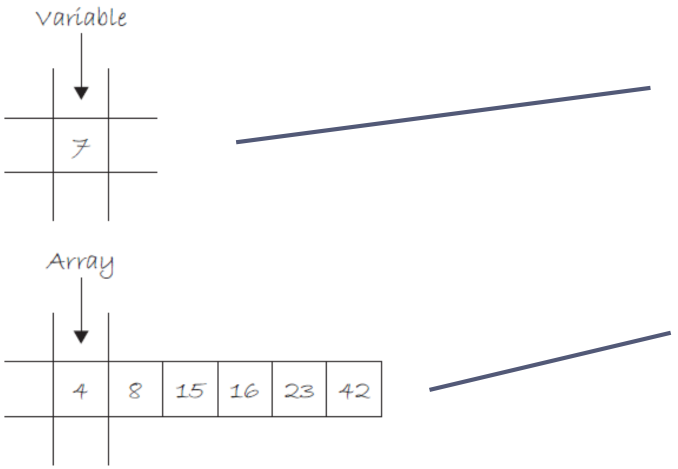
…continue…

Solution of Exercise 7

```
// Draw alien's body
stroke(0);
fill(175);
rect(x,y,w/6,h*2);

// Draw alien's head
fill(255);
ellipse(x,y-h/2,w,h);

// Draw alien's eyes
fill(0);
ellipse(x-w/3,y-h/2,eyeSize,eyeSize*2);
ellipse(x + w/3,y-h/2,eyeSize,eyeSize*2);

// Draw alien's legs
stroke(0);
line(x-w/12,y + h,x-w/4,y + h + 10);
line(x + w/12,y + h,x + w/4,y + h + 10);
}
```

Arrays

- Any time a program requires multiple instances of similar data, it might be time to use an **array**.
  - We can think to an array as a **list of variables**.



Recall that a **variable** is a **named pointer** to a location in memory where data is stored.

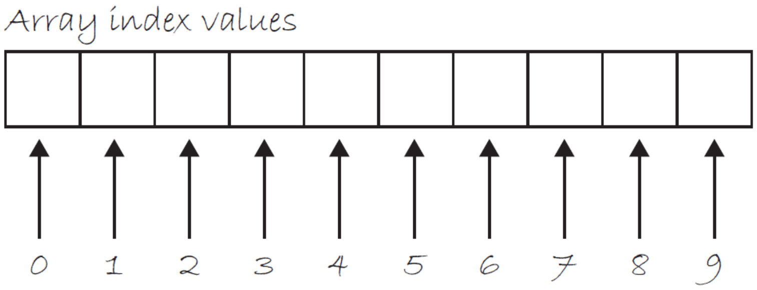
An **array**, instead of pointing to one singular piece of information, **points to multiple pieces**.

- A **list** is useful for two important reasons:

1. The list **keeps track** of the **elements** in the list themselves.
2. The list **keeps track** of the **order of those elements** (which element is the first in the list, the second, the third, etc.). This is a crucial point since in many programs, the order of information is just as important as the information itself.

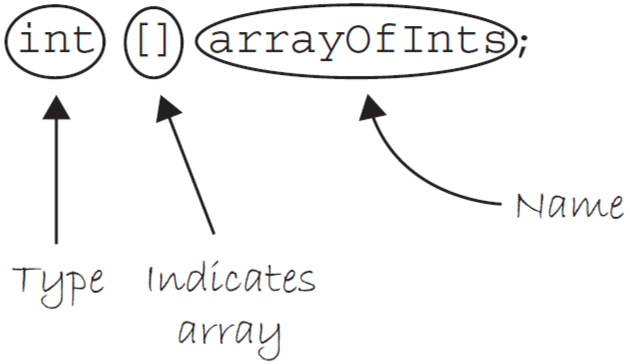
Declaration of arrays

- In an array, each element of the list has a **unique index**, an **integer value that designates its position in the list** (element #1, element #2, etc.).



Example of an array of **10 elements**. We start at **zero** because technically the first element of the array is located at *distance of zero from the beginning.*

- The **declaration** statement of an array must have a **name** and a **data type**. In addition, we denote the use of an array by placing **empty square brackets** "**[]**" after the type declaration.

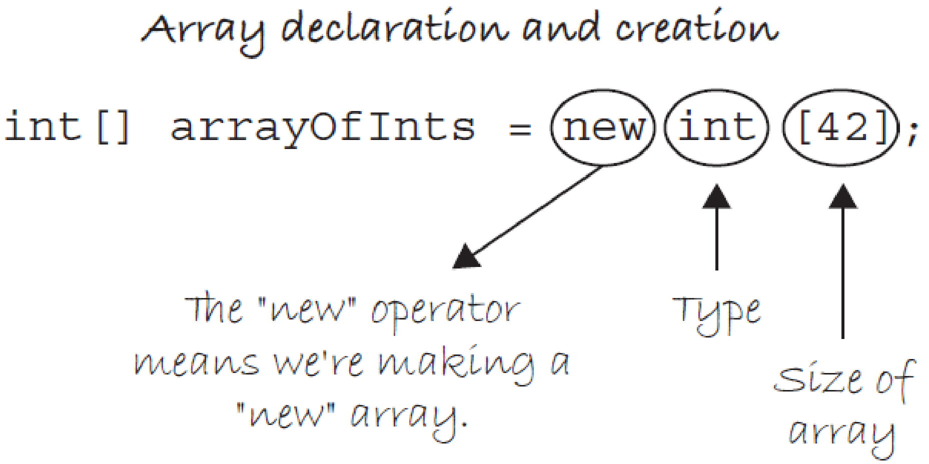


```
int [] arrayOfInts;
```

Array of primitive values (integers). The array named `arrayOfInts` will store a list of integers.

Creation of arrays

- One fundamental property of arrays is that they are of **fixed size**.
  - The **size** of an array specifies how many elements we want the array to hold.
- We define the size of an array during the **creation** stage.



```
int[] arrayOfInts = new int [42];
```

Array declaration and creation
The "new" operator means we're making a "new" array.
Type
Size of array

To create an array, we use the `new` operator, followed by the *data type*, followed by the *size* of the array enclosed in brackets.

We are defining an array that can contain **42 integer values!**

- Once we define the size for an array, **<u>its size can never change</u>**.
  - *A list of 42 integers **can never go** to 43.*

Example of array declaration and creation

```
// A list of 10 integers numbers
int[] numbers = new int[10];

// A list of 4 floating numbers
float[] scores = new float[4];

// Using a variable to specify size
// A list of 5 integers numbers
int num = 5;
int[] numbers = new int[num];

// A list of 5 float numbers
int num = 5;
float[] scores = new float[num];
```

Initializing an array

- One way to fill an array is to store the values in each spot of the array.
- The initialization happens with the **name** of the array, followed by the **index value** enclosed in brackets. → `arrayName[INDEX]`

```
int[] stuff = new int[3];

// The first element of the array equals 8
stuff [0] = 8;

// The second element of the array equals 3
stuff [1] = 3;

// The third element of the array equals 1
stuff [2] = 1;
```

- A **second option** for initializing an array is to manually type out a list of values enclosed in curly braces and separated by commas.

```
int[] arrayOfInts = {1, 5, 8, 9, 4, 5};
float[] floatArray = {1.2, 3.5, 2.0, 3.4123, 9.9};
```

Initializing huge arrays

- To initialize big arrays, it is possible to iterate through its elements.
- Using a *while* loop to initialize all elements of an array

```
float[] values = new float[1000];
int n = 0;
while (n < 1000) {
 values[n] = random(0,10);
 n = n + 1;
}
```

Assign to any element of the array a random value ranging from 0 to 10.

- Using a *for* loop to initialize all elements of an array

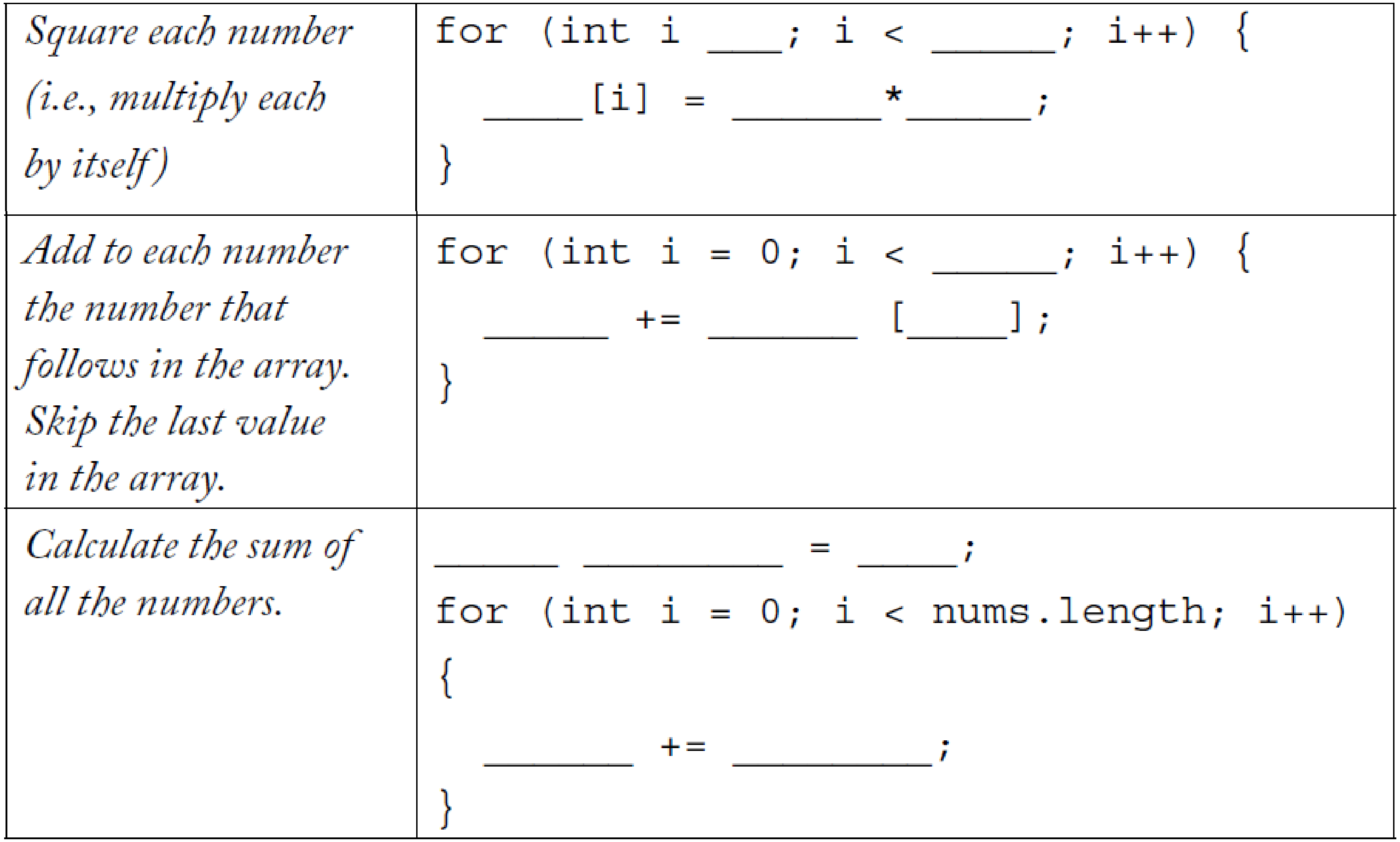
```
float[] values = new float[1000];
for (int n = 0; n < 1000; n++) {
   values[n] = random(0,10);
}
```

Alternatively, we can use the **length** property.

```
for (int n = 0; n <
values.length; n++) {
```

Exercise 8

- Given the following array: `int[] nums = {5,4,2,7,6,8,5,2,8,14};`

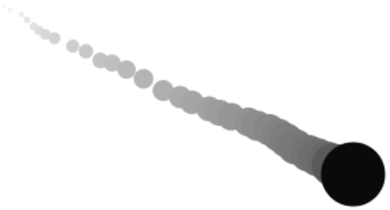


Square each number (i.e., multiply each by itself)	`for (int i ___; i < ______; i++) {`
	`   _____[i] = _______*______;`
	`}`

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++)`
	`{`
	`   _______ += __________;`
	`}`

Exercise 9 – *The Snake*



- We want to program a trail following the mouse.
  - The solution requires **two arrays**, one to store the history of horizontal mouse locations, and one for vertical.
  - Let's say, arbitrarily, that we want to store the last **50 mouse locations**.
- First, we declare the two arrays.

```
int num = 50;
int[] xpos = new int[num];
int[] ypos = new int[num];
```

- Second, in `setup()`, we initialize the arrays. Since at the beginning there has not been any mouse movement, we fill the arrays with 0.

```
void setup() {
 size(640,480);
 for (int i = 0; i<xpos.length; i++) {
  xpos[i] = 0;
  ypos[i] = 0;
 }
}
```

Exercise 9 – *The Snake*

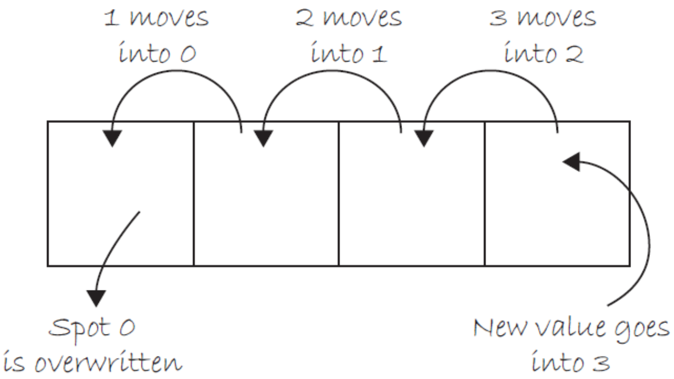
- Each time through the main `draw()` loop, we want to update the array with the current mouse location.
- Let’s choose to put the current mouse location in the last spot of the array.
  - The length of the array is 50, meaning index values range from 0–49. The the last spot is index 49, or the length of the array minus one.

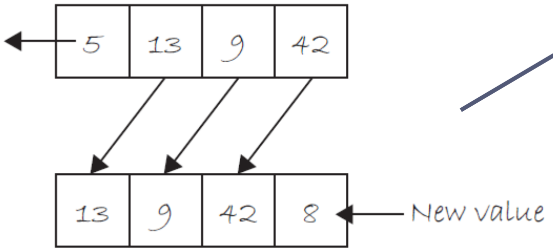
```
void draw() {
  background(255);
  .........
  xpos[xpos.length-1] = mouseX;
  ypos[ypos.length-1] = mouseY;
  .........
```

Exercise 9 – *The Snake*

- We want to keep **only the last 50 mouse locations**,
- We store the current mouse location at the end of the array; basically, we are overwriting what was previously stored there.
- The solution is to **shift all of the elements of the array** down one spot before updating the current location.

Element index 49 moves into spot 48, 48 moves into spot 47, 47 into 46, and so on.





Exercise 9 – *The Snake*

- We loop through the array and sett each element index i to the value of element i plus one.
  - Note we must stop at the second to last value since for element 49 there is no element 50 (49 plus 1).
  - In other words, instead of having an exit condition:
    - i < xpos.length;
  - we must instead say:
    - i< xpos.length – 1;
  - The full code for performing this array shift is as follows:

```
for (int i = 0; i < xpos.length-1; i++) {
  xpos[i] = xpos[i + 1];
  ypos[i] = ypos[i + 1];
}
```

Exercise 9 – *The Snake*

- Finally, we can use the history of mouse locations to draw a series of circles. For each element of the `xpos` array and `ypos` array, draw an ellipse at the corresponding values stored in the array.

```
for (int i = 0; i < xpos.length; i++) {
 noStroke();
 fill(255-i*5);
 ellipse(xpos[i],ypos[i],i,i);
}
}
```

We link the **brightness** and the **size** of the circle to the location in the array.

The earlier (and therefore older) values will be bright and small and the later (newer) values will be darker and bigger.

Exercise 9 – *The Snake (complete code)*

```
int num = 50;
int[] xpos = new int[num];
int[] ypos = new int[num];

void setup() {
 size(640,480);

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

void draw() {
 background(255);

// Shift array values
for (int i = 0; i < xpos.length-1; i++) {
 xpos [i] = xpos[i + 1];
 ypos[i] = ypos[i + 1];
}

// New location
xpos[xpos.length-1] = mouseX;
ypos[ypos.length-1] = mouseY;

// Draw everything
for (int i = 0; i < xpos.length; i++) {
 noStroke();
 fill(255-i*5);
 ellipse(xpos[i],ypos[i],i,i);
}
}
```