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

Interaction Design

A.A. 2017/2018

7 – Conditionals in Processing

Francesco Leotta, Andrea Marrella

Last update : 12/4/2018

Conditional Statements

* Boolean expression

- **Conditional statements**: How a program produces different results based on varying circumstances.
- In the world of computer programming, we only take one kind of test: the **boolean test**: *true* or *false*.
- A **boolean expression** is an expression that evaluates to either *true* or *false*. Let's look at some common language examples:
  - I am hungry* → true
  - I am afraid of computer programming* → false
- In the formal logic of computer science, we test relationships between numbers.
  - 15 is greater than 20* → false
  - 5 equals 5* → true
  - 32 is less than or equal to 33* → true

Conditionals: if, else

- Boolean expressions (often referred to as *conditionals*) operate within the sketch as **questions**.
  - *Is 15 greater than 20?*
  - If the answer is yes (true), we can choose to execute certain instructions (such as draw a rectangle); if the answer is no (false), those instructions are ignored.
- This introduces the idea of **branching**; depending on various conditions, the sketch can follow different paths.

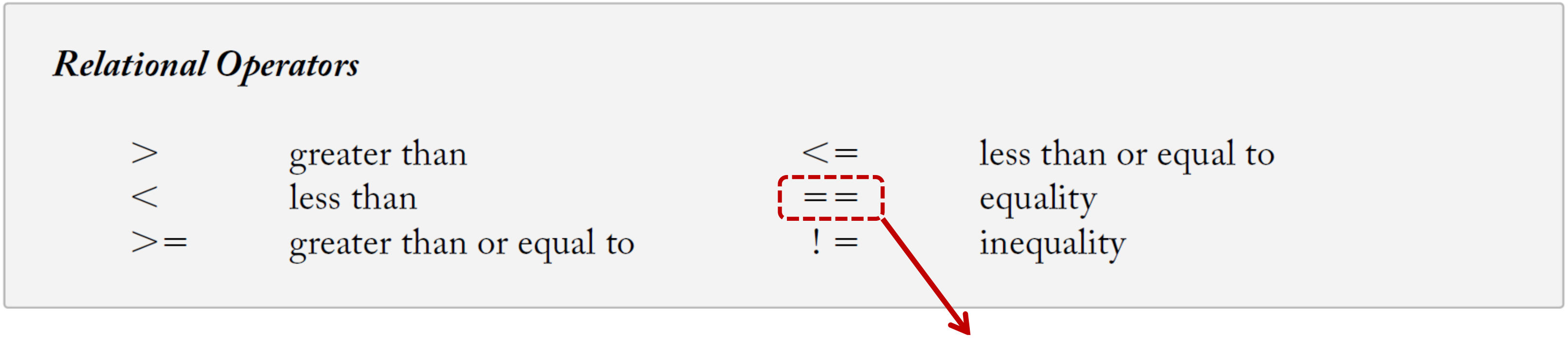
```
if (boolean expression) {
 // code to execute if boolean expression is true
}
```

- The structure can be expanded with the keyword `else` to include code that is executed if the boolean expression is false.

```
if (boolean expression) {
 // code to execute if boolean expression is true
} else {
 // code to execute if boolean expression is false
}
```

Boolean Expressions as Conditionals

- In Processing, boolean expressions have the following form:
  - x > 20 → depends on current value of x
  - y == 5 → depends on current value of y
  - z <= 33 → depends on current value of z
  - z >= k → depends on current values of z and k
  - x != k + z – y → depends on current values of x, z, k and y
  - k < z + 2 → depends on current values of k and z
- The following operators can be used in a boolean expression:



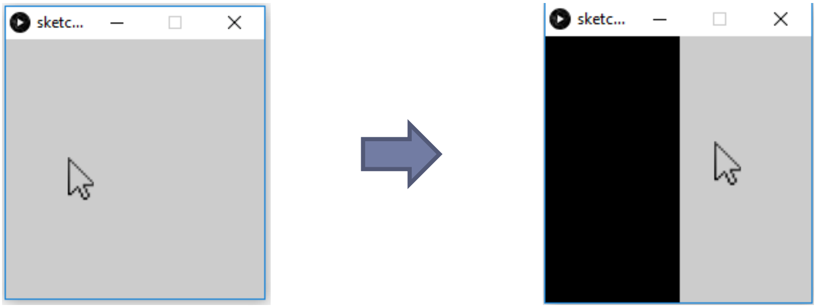
Relational Operators

>	greater than	<=	less than or equal to
<	less than	==	equality
>=	greater than or equal to	! =	inequality

*Equality is done with **double equals.** Single equal is used for variables assignment operations.*

Example

- *If the mouse is on the right side of the screen, draw a black rectangle on the left side of the screen .*

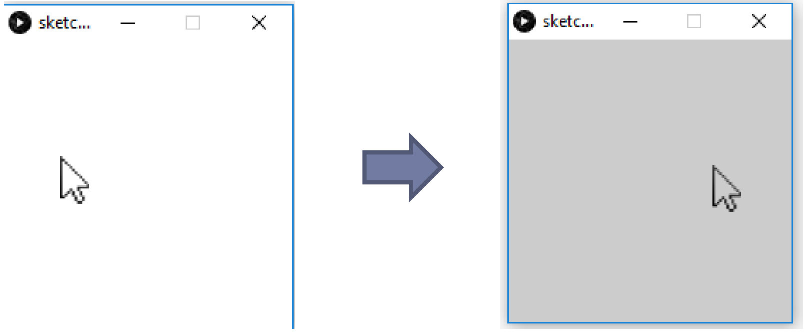


```
void setup() {
  size(200,200);
}

void draw() {
  if (mouseX > width/2) {
     fill(0);
     rect(0,0,width/2,height);
    }
}
```

Example

- *If the mouse is on the left side of the screen, draw a white background, otherwise draw a grey background .*

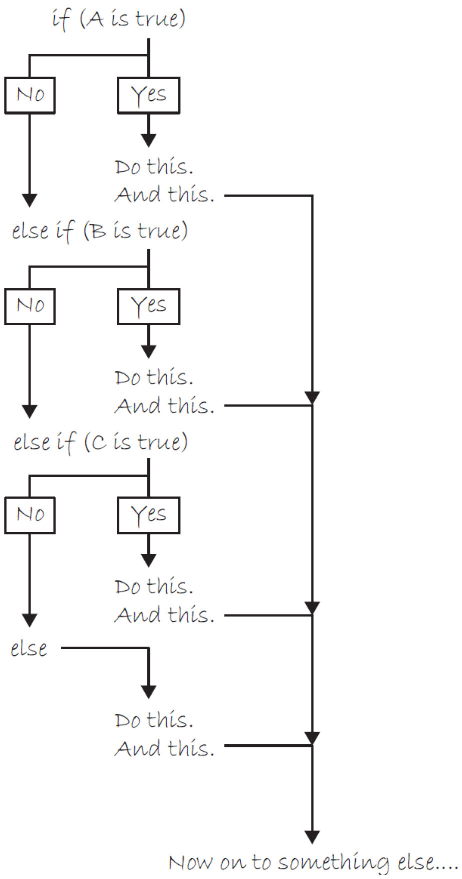
```
void setup() {
  size(200,200);
}

void draw() {
  if (mouseX < width/2) {
     background(255);
     } else {
     background(150);
  }
}
```



Testing Multiple Conditions

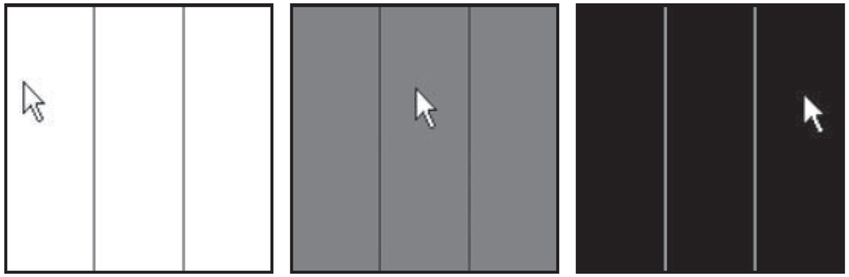
- For testing multiple conditions, we use an `else if`.
  - When an `else if` is used, the conditional statements are evaluated in the order presented.
  - As soon as one boolean expression is found to be *true*, the corresponding code is executed and the remaining boolean expressions are ignored.

```
if (boolean expression #1) {
   // code to execute if boolean expression #1 is true
   } else if (boolean expression #2) {
    // code to execute if boolean expression #2 is true
   } else if (boolean expression #n) {
    // code to execute if boolean expression #n is true
   } else {
     // code to execute if none of the above
     // boolean expressions are true
   }
```



Example

- *If the mouse is on the left third of the window, draw a white background, if it is in the middle third, draw a gray background, otherwise, draw a black background.*



```
void setup() {
  size(400,400);
}
void draw() {
  if (mouseX < width/3) {
    background(255);
  } else if (mouseX < 2*width/3) {
    background(127);
  } else {
    background(0);
  }
}
```

Boolean Variables

- A **boolean variable** (or a variable of type boolean) is a variable that **can only be** *true* or *false* (think of it as a switch. It is either on or off).
- In Processing there are several system (boolean) variables, such as `mousePressed` and `keyPressed`

```
boolean switched = false;

   void draw() {

     if (switched) {

       background(255);

     } else {

       background(0);

     }

     if(mousePressed) {

       switched = false;

     } else if(keyPressed) {

     switched = true;
   }}
```

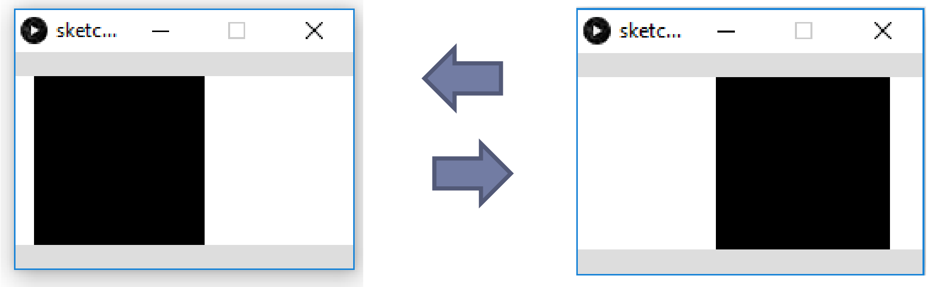
For including a boolean variable in a sketch, we should initialize it with a starting value, being it true or false.

A boolean variable can be used *alone* in a IF statement, as it already records a true/false value.

When the mouse is pressed, the variable `switched` turned to false and the background becomes black. Otherwise, when a key is pressed, the variable `switched` turned to true and the background becomes white.

Exercise 1 – *Moving Rectangle*

- Move a rectangle across a window by incrementing a variable each time of a unity.
- Start the shape at x coordinate 0 (y coordinate is fixed at 0) and use an IF statement to have it stop at x coordinate 200.
- Then, decrement the same variable each time of a unity to have it stop at coordinate 0…and so on.



Solution of Exercise 1

```
int x = 0;
int y = 0;
boolean increment = true;

void setup() {
size(200,100);
}

void draw() {
  background(255);
  fill(0);

  if(x==100) {
    increment = false;
  }
  else if(x==0) {
   increment = true;
  }
  if(increment) {
      x = x+1;
     } else {
      x = x-1;
     }

    rect(x,y,100,100);
  }
```

Exercise 2 – *Keys, Clicks and Colors*

- Write a sketch in a way that the background color is changed depending on the following rules:
  - At the beginning, use `background(255,255,255).`
  - If the left button of the mouse is pressed, use `background(100,100,100).`
  - If the right button of the mouse is pressed, use `background(10,100,200).`
  - If the letter ‘w’ of the keyboard is pressed, use `background(200,10,100).`
  - If the letter ‘x’ of the keyboard is pressed, use `background(100,200,10).`
  - In all the other cases, use `background(0,0,0).`

Solution of Exercise 2

```
void setup() {
  background(255,255,255);
}
void draw() {}
void mousePressed() {
  if(mouseButton == LEFT) {
   background(100,100,100);
  }
  else if(mouseButton == RIGHT) {
    background(10,100,200);
  } else {
    background(0,0,0);
  }
}
```

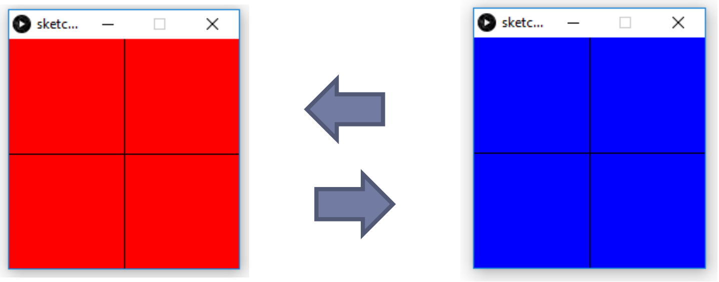
…continue…

Solution of Exercise 2

```
void keyPressed() {
  if(keyCode == ‘w’) {
   background(200,10,100);
  }
  else if(keyCode == ‘x’) {
    background(100,200,10);
  } else {
    background(0,0,0);
  }

}
```

Exercise 3 – *Dynamic Colors*

- Create a sketch that performs the following steps:
  - **Step 1**. Create variables to hold on to red, green, and blue color components. Call them r , g , and b .
  - **Step 2**. Continuously draw the background based on those colors.
  - **Step 3**. Draw lines to divide the window into quadrants.
  - **Step 4**. If the mouse is on the right-hand side of the screen, increment the value of r (increase red), if it is on the left-hand side decrement the value of r (decrease red).
  - **Step 5**. If the mouse is on the bottom of the window, increment the value of b (increase blue). Otherwise, it is on the top decrement the value of b (decrease blue).



Solution of Exercise 3

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

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

void draw() {
  background(r,g,b);
  stroke(0);
  line(width/2,0,width/2,height);
  line(0,height/2,width,height/2);
```

Step 1. Three variables for the background color.

Step 2. Draw the backgroud.

Step 3. Draw lines to divide the window into quadrants.

…continue…

Solution of Exercise 3

```
if(mouseX > width/2) {
     r = r + 1;
} else {
     r = r - 1;
}
if (mouseY > height/2) {
     b = b + 1;
} else {
     b = b - 1;
}
}
```

Step 4. If the mouse is on the right-hand side of the window, increase red. Otherwise, it is on the left-hand side and decrease red.

Step 5. If the mouse is on the bottom of the window, increase blue. Otherwise, it is on the top and decrease blue.

Constraining the value of a variable

- In the previous example, color values may increase to unreasonable extremes (less than 0 and more than 255).
- We might want to **constrain the value of a variable** (for example, a size or a location of a shape) so that it does not get too big or too small, or wander off the screen.
- For doing that, Processing offers a function entitled `constrain(var,min,max)` that takes three arguments in input:
  - the value of the variable `var` we intend to constrain
  - the minimum limit `min`
  - the maximum limit `max`
- The function returns the constrained value and is assigned back to a variable.

```
// we can add the following code to constraint the variable values
  r = constrain(r,0,255);
  g = constrain(g,0,255);
  b = constrain(b,0,255);
```

Constrain all color values to between 0 and 255.

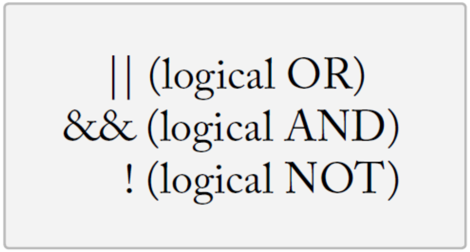
Solution of Exercise 3 (with constrained variables)

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

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

void draw() {
  background(r,g,b);
  stroke(0);
  line(width/2,0,width/2,height);
  line(0,height/2,width,height/2);
```

```
if(mouseX > width/2) {
      r = r + 1;
} else {
      r = r - 1;
}
if (mouseY > height/2) {
      b = b + 1;
} else {
      b = b - 1;
}
     r = constrain(r,0,255);
     g = constrain(g,0,255);
     b = constrain(b,0,255);
}
```

Logical Operators

- Sometimes, simply performing some code based on one condition is not enough. For example:
  - *If the mouse is on the right side of the screen **AND** the mouse is on the bottom of the screen, draw a rectangle in the bottom right corner*.
  - *If a key is pressed **OR** the left button of the mouse is **NOT** clicked, draw a black ellipse.*
- In order to build complex conditions, some **logical operators** can be used and properly combined in a boolean expression.



|| (logical OR)
&& (logical AND)
! (logical NOT)

- *Build a rectangle if the mouse is on the right side of the screen **AND** on the bottom.*

```
if (mouseX > width/2 && mouseY > height/2) {
  fill(255);
  rect(width/2,height/2,width/2,height/2);
}
```

- *Build an ellipse if the mouse is on the right side of the screen **OR** on the bottom.*

```
if (mouseX > width/2 || mouseY > height/2) {
  fill(255);
  rect(width/2,height/2,width/2,height/2);
}
```

The NOT Logical Operator

- In addition to && and ||, there is also the logical operator **NOT** written as an exclamation point: !
  - *If the mouse is **NOT** pressed, draw a circle, otherwise draw a square.*

```
if (!mousePressed) {
  ellipse(width/2,height/2,100,100);
} else {
  rect(width/2,height/2,100,100);
}
```

  - In the previous example, `(! mousePressed)` means "NOT mousePressed". The resulting boolean expression has a value that is either true or false (depending on whether or not the mouse is currently pressed).
    - If the mouse is pressed, (! mousePressed) is equal to FALSE.
    - If the mouse is not pressed, (! mousePressed) is equal to TRUE.

Evaluate Logical Operators

- Given two boolean expressions A and B associated with the logical operator && (AND), the resulting expression **(A && B)** is *true* if and only if both A and B are *true*. Otherwise, it is *false*.
- Given two boolean expressions A and B associated with the logical operator || (OR), the resulting expression **(A || B)** is *true* if and only if at least one between A and B is *true*. Otherwise, it is *false*.
- Given a boolean expression A associated with the logical operator ! (NOT), the resulting expression **(!A)** is inverted: if A is *true*, (!A) is *false*; if A is *false*, (!A) is *true*.

A	B	A && B	A || B	!A
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

Exercise 4

- Are the following expressions *true* or *false*?
  - Assume variables `int x = 5` and `int y = 6`

```
!(x > 6)

(x == 6 && x == 5)

(x == 6 || x == 5)

(x == 3 || y == 5)

(x == 5 && y == 6)

(x == 5 && (y == 6 || y == 7))

(x > -1 && y < 10)
```

  - Although the syntax is correct, what is flawed about the following boolean expression?

```
(x > 10 & & x < 5)
```

Solution of Exercise 4

- Are the following expressions *true* or *false*?
  - Assume variables `int x = 5` and `int y = 6`

```
!(x > 6)  → TRUE
(x == 6 && x == 5)  → FALSE
(x == 6 || x == 5)  → TRUE
(x == 3 || y == 5)  → FALSE
(x == 5 && y == 6)  → TRUE
(x == 5 && (y == 6 || y == 7))  → TRUE
(x > -1 && y < 10)  → TRUE
```

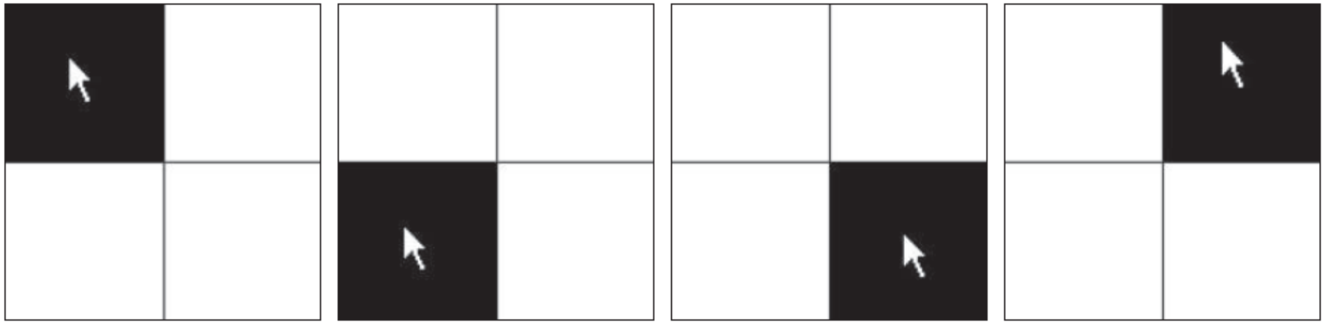
- Although the syntax is correct, what is flawed about the following boolean expression?

```
(x > 10 && x < 5)
```

It is always **false**. It is not possible that x is greater than 10 and lower than 5 at the same time!

Exercise 5 – *Multiple Rollovers*

- Write the Processing code that solves the following problem:



- ***Setup:***
- **1.** Set up a window of 200 200 pixels .
- ***Draw:***
- **2.** Draw a white background.
- **3.** Draw horizontal and vertical lines to divide the window in four quadrants .
- **4.** If the mouse is in the top left corner, draw a black rectangle in the top left corner.
- **5.** If the mouse is in the top right corner, draw a black rectangle in the top right corner.
- **6.** If the mouse is in the bottom left corner, draw a black rectangle in the bottom left corner.
- **7.** If the mouse is in the bottom right corner, draw a black rectangle in the bottom right corner.

"How do we know if the mouse is in a given corner?" To accomplish this, we would say: *"If the mouse X location is greater than 100 pixels and the mouse Y location is greater than 100 pixels, draw a black rectangle in the bottom right corner".*

Solution of Exercise 5

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

void draw() {
  background(255);
  stroke(0);
  line(100,0,100,200);
  line(0,100,200,100);
  // Fill a black color
  noStroke();
  fill(0);
```

Solution of Exercise 5

```
if (mouseX < 100 && mouseY < 100) {
  rect(0,0,100,100);
} else if (mouseX > 100 && mouseY < 100) {
  rect(100,0,100,100);
} else if (mouseX < 100 && mouseY > 100) {
  rect(0,100,100,100);
} else if (mouseX > 100 && mouseY > 100) {
  rect(100,100,100,100);
}
}
```

Depending on the mouse location,
a different rectangle is displayed!

Exercise 6 – *Perimeter Rectangle*

- Draw a rectangle that moves and follows the edges of a window.

- One way to solve this problem is to think of the rectangle's motion as having four possible states, numbered 0 through 3.
  - **State #0**: left to right.
  - **State #1**: top to bottom.
  - **State #2**: right to left.
  - **State #3**: bottom to top.
- We can use a variable to keep track of the state number and adjust the x, y coordinate of the rectangle according to the state.
- Once the rectangle reaches the endpoint for that state, we can change the state variable.

Solution of Exercise 6

```
int x = 0;
int y = 0;
int speed = 5;
int state = 0;


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

x and y locations of the square.

Speed of the square. It determines how fast the square moves.

A variable to keep track of the square’s state. Depending on the value of its state, it will either move right, down, left, or up.

…continue…

Solution of Exercise 6

```
void draw() {
background(255);

// Display the square
noStroke();
fill(0);
rect(x,y,10,10);
```

…continue…

Solution of Exercise 6

```
if (state == 0) {
  x = x + speed;
  if (x > width-10) {
     x = width-10;
     state = 1;
  }
}
else if (state == 1) {
  y = y + speed;
  if (y > height-10) {
    y = height-10;
    state = 2;
  }
}
```

State 0: left to right

State 1: top to bottom

…continue…

Solution of Exercise 6

```
else if (state == 2) {
  x = x - speed;
  if (x < 0) {
    x = 0;
    state = 3;
  }
}
else if (state == 3) {
  y = y - speed;
  if (y < 0) {
    y = 0;
    state = 0;
  }
}
}
```

State 2: right to left

State 3: bottom to top

Let’s Play: Add Gravity!

```
float x = 100; // x location of square
float y = 0; // y location of square
float speed = 0; // speed of square
float gravity = 0.1;

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

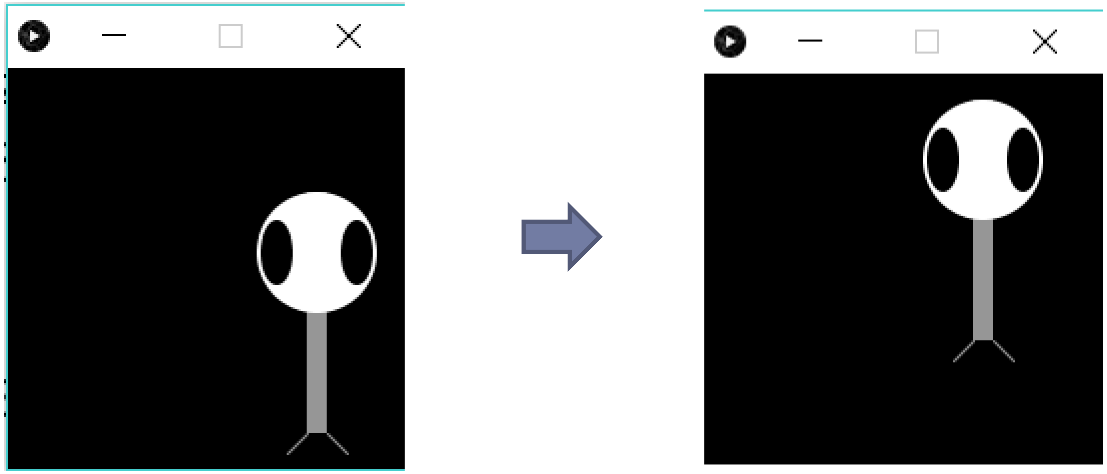
A new variable, for simulating gravity. We use a small number (0.1) that accumulates over time, increasing the speed. *Try changing this number to 2.0 and see what happens.*

```
void draw() {
  background(255);
  // Display the square
  fill(0);
  noStroke();
  rectMode(CENTER);
  rect(x,y,10,10);
  y = y + speed;
  speed = speed + gravity;
  // If square reaches the bottom
// Reverse speed
if (y > height) {
speed = speed * -0.95;
}}
```

Multiplying by -0.95 instead of 1 slows the square down each time it bounces (by decreasing speed). This is known as a “dampening” effect and is a more realistic simulation of the real world (without it, a ball would bounce forever).

Exercise 7 – *Bouncing Alien*

- Write the Processing code that allows to move the alien within the screen by bouncing on the edges of the screen.



Solution of Exercise 7

```
float x = 100;
float y = 100;
float w = 60;
float h = 60;
float eyeSize = 16;
float xspeed = 3;
float yspeed = 1;

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

The alien has variables for speed in the horizontal and vertical direction.

…continue…

Solution of Exercise 7

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

  if ((x > width) || (x < 0)) {
    xspeed = xspeed * -1;
  }

  if ((y > height) || (y < 0)) {
    yspeed = yspeed * -1;
  }
```

An IF statements with a logical OR determines if the alien has reached either the right or left edges of the screen. When this is true, we multiply the speed by -1, reversing the alien's direction!

Identical logic is applied to the y direction as well.

Solution of Exercise 7

```
   background(0);
   ellipseMode(CENTER);
   rectMode(CENTER);
   noStroke();
   // Draw alien's body
   fill(150);
   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(150);
   line(x-w/12,y + h,x-w/4,y + h + 10);
   line(x + w/12,y + h,x + w/4,y + h + 10);
}
```