

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

# ***Interaction Design***

## ***A.A. 2017/2018***

6 – Variables and Images in Processing

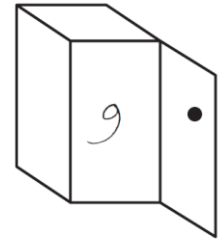
Francesco Leotta, Andrea Marrella

Last update : 12/4/2018

# Variables

---

- ▶ A **variable** is like a *storage locker*.
  - ▶ You put something in the locker where it can live safely, and retrieve it whenever you need it.
- ▶ Technically speaking, a variable is a **named pointer to a location in the computer's memory** (“memory address”) where data is stored.
- ▶ Since computers only **process information one instruction at a time**, a variable allows a programmer to **save information from one point in the program and refer back to it at a later time**.
- ▶ Variables can keep track of information related to shapes, color, size...
- ▶ The power of a variable does not simply rest with the ability to remember a value, but on the fact that its value **varies** over time, and a programmer periodically alters that value.



variable  
locker

# Variables in Scrabble game

- ▶ Consider a game of Scrabble between Billy and Jane.
- ▶ To keep track of the score, Jane takes out paper and pencil, and scrawls down two column names: “Billy’s Score” and “Jane’s Score”.
- ▶ If we imagine this game to be **virtual Scrabble** programmed on a computer, we suddenly can see the concept of a *variable that varies* emerge.
- ▶ That piece of paper is the computer’s memory and on that paper, information is written “Billy’s Score” and “Jane’s Score” are **variables**, *locations in memory where each player’s total points are stored and that change over time*.

| Jane's Score  | Billy's Score |
|---------------|---------------|
| <del>5</del>  | <del>10</del> |
| <del>30</del> | <del>25</del> |
| <del>53</del> | <del>47</del> |
| <del>65</del> | <del>68</del> |
| <del>87</del> | <del>91</del> |
| 101           | 98            |

name

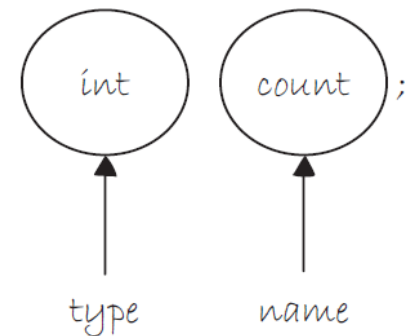
value

type of the information:  
an integer

# Variable Declaration in Processing

- ▶ Variables are **declared** by first stating the **type**, followed by the **name**.
- ▶ A **type** is the **kind of data stored in that variable**. This could be:
  - ▶ Integer: ... -2, -1, 0, 1, 2, 3 ...
  - ▶ Decimal: ... 3.14159, 2.5, -9.95 ...
  - ▶ Character: ... 'a', 'b', 'c' ...
  - ▶ and many others...
- ▶ Variable **names**:
  - ▶ must be **one word** (no spaces)
  - ▶ must **start with a letter**
  - ▶ can include numbers, but **cannot start with a number**
  - ▶ can not include any punctuation or special characters, with the exception of the underscore: “\_”

Example of a variable named **count** of type **int**, which stands for *integer*.



# Primitive types and variable names

---

## *All Primitive Types*

- *boolean*: true or false
- *char*: a character, 'a', 'b', 'c', etc.
- *byte*: a small number, -128 to 127
- *short*: a larger number, -32768 to 32767
- *int*: a big number, -2147483648 to 2147483647
- *long*: a really huge number
- *float*: a decimal number, such as 3.14159
- *double*: a decimal number with a lot more decimal places (only necessary for advanced programs requiring mathematical precision).

## **Tips for choosing good variable names**

- Avoid using words that appear elsewhere in the *Processing* language. In other words, do not call your variable *mouseX*, there already is one!
- Use names that mean something. This may seem obvious, but it is an important point. For example, if you are using a variable to keep track of score, call it "score" and not, say, "cat."
- Start your variable with a lowercase letter and join together words with capitals. Words that start with capitals are reserved for classes (Chapter 8). For example: "frogColor" is good, "Frogcolor" is not. this canTake some gettingUsedTo but it will comeNaturally soonEnough.

# Variable Initialization

---

- ▶ Once a variable is declared, we can then **assign it a value** by setting it equal to something. This procedure is called **initialization**.
  - ▶ If we **forget** to initialize a variable, Processing will give it a default value, such as 0 for integers, 0.0 for floating points, and so on.
  - ▶ However, it is good to **always initialize variables** in order to avoid confusion.

```
int count;  
count = 50;
```

Declare and initialize a variable named **count** of type **int** with the value **50**;

- ▶ We can combine the above two statements into one.

```
int count = 50;
```

Declare and initialize a variable in one line of code.

- ▶ A variable can also be initialized by another variable ( *x equals y* ), or by evaluating a mathematical expression ( *x equals y plus z* , etc.).

# Example of Variable Initialization

---

```
// Declare an int named count, and assign the value 0
int count = 0;

// Declare a char named letter, and assign the value 'a'
char letter = 'a';

// Declare a double named d, and assign the value 132.32
double d = 132.32;

// Declare a boolean named happy, and assign the value false
boolean happy = false;

// Declare a float named x, and assign the value 4.0
float x = 4.0;

// Declare a float named y (no assignment)
float y;

// Assign the value of x plus 5.2 to the previously declared y
y = x + 5.2;

// Declare a float named z, assign a value which is x times y plus 15.0.
float z = x * y + 15.0;
```

# Using a variable

- ▶ Let's take a simple example of a program that draws a circle onscreen.

```
int circleX = 100;  
int circleY = 100;
```

Declare and initialize two integer variables at the top of the code.

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

```
void draw() {  
  background(255);  
  fill(175);  
  ellipse(circleX, circleY, 50, 50);  
}
```

Use the variables to specify the location of an ellipse. This corresponds to:

```
ellipse(100, 100, 50, 50);
```





# Assignment Operation

---

- ▶ A variable is not simply a placeholder for one constant value. We call it a variable because it **varies**.
- ▶ To change its value, we write an **assignment operation**, which assigns a new value to the variable.

*variable name = expression;*

```
int a = 1;
int b = 2;
int x = 5;
x = a + b;
int y = 300;
x = y - 10 * 20;
x = x * 5;
```

Examples of assigning a new value to variables.

Notice that assignment is the same as how we initialize a variable, **only the variable does not need to be declared**.

Variable declaration is performed just the first time we define a variable in the source code.

At the end, x will be equal to **500**.

# Varying variables

- ▶ Let's take a simple example of a program that draws a circle onscreen.

```
int circleX = 0;  
int circleY = 100;
```

Declare and initialize two integer variables at the top of the code.

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

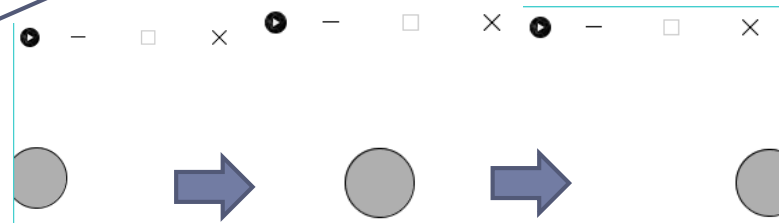
To increment the variable `circleX` by one, we say: `circleX` equals itself plus one.

In code, this amounts to the following assignment operation:

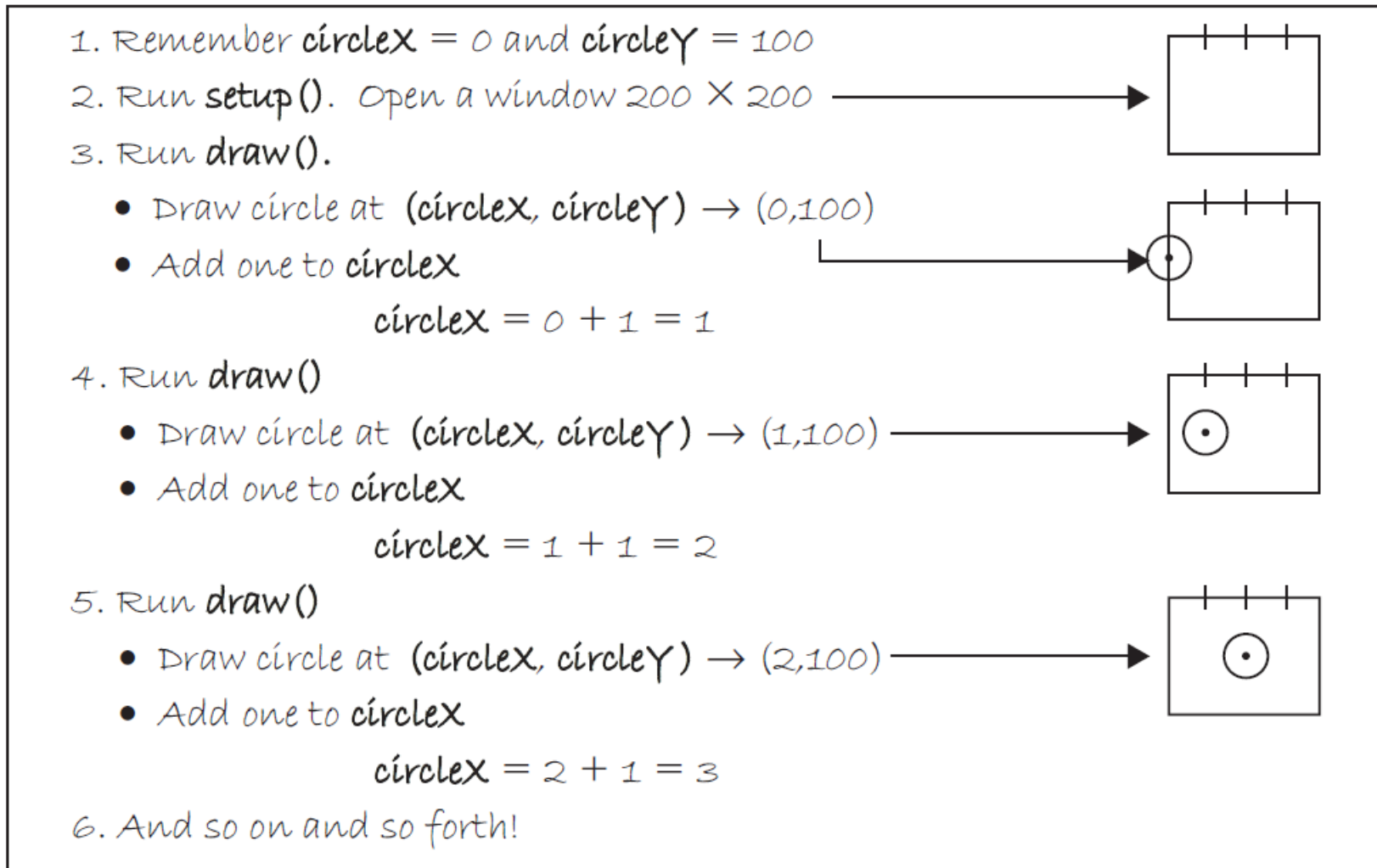
```
circleX = circleX + 1;
```

```
void draw() {  
  background(255);  
  fill(175);  
  ellipse(circleX, circleY, 50, 50);  
  circleX = circleX + 1;  
}
```

Remember that `draw()` loops over and over again, all the while retaining the value of `circleX` in memory.



# What happens by running the previous example?



# Exercise 1

---

- ▶ Change the previous example so that instead of the circle moving from left to right, the circle grows in size.

# Solution to Exercise 1

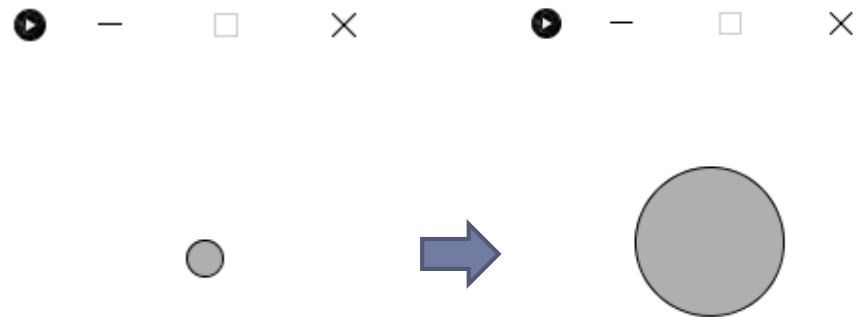
---

- ▶ Change the previous example so that instead of the circle moving from left to right, the circle grows in size.

```
int circleX = 100;  
int circleY = 100;  
int circleWidth = 0;  
int circleHeight = 0;
```

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

```
void draw() {  
  background(255);  
  fill(175);  
  ellipse(circleX,circleY,circleWidth,circleHeight);  
  circleWidth = circleWidth + 1;  
  circleHeight = circleHeight + 1;  
}
```



# Exercise 2

---

- ▶ Change the previous example so that the circle grows of a unity in size when the mouse is clicked and decreases of a unity in size when a key is pressed.

# Solution to Exercise 2

---

```
int circleX = 100;
int circleY = 100;
int circleWidth = 0;
int circleHeight = 0;

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

void draw() {
  background(255);
  fill(175);
  ellipse(circleX,circleY,circleWidth,circleHeight);
}

void mousePressed() {
  circleWidth = circleWidth + 1;
  circleHeight = circleHeight + 1;
}

void keyPressed() {
  circleWidth = circleWidth - 1;
  circleHeight = circleHeight - 1;
}
```

# Using many variables

---



- ▶ Variables can be used to express any piece of information. In the following example, 8 variables of kind *int* are used.

```
int circleX = 0;
int circleY = 0;
int circleW = 50;
int circleH = 100;
int circleStroke = 255;
int circleFill = 0;
int backgroundColor = 255;
int change = 1;

// Your basic setup
void setup() {
  size(200,200);
}

void draw() {
  // Draw the background and the ellipse
  background(backgroundColor);
  stroke(circleStroke);
  fill(circleFill);
  ellipse(circleX,circleY,circleW,circleH);

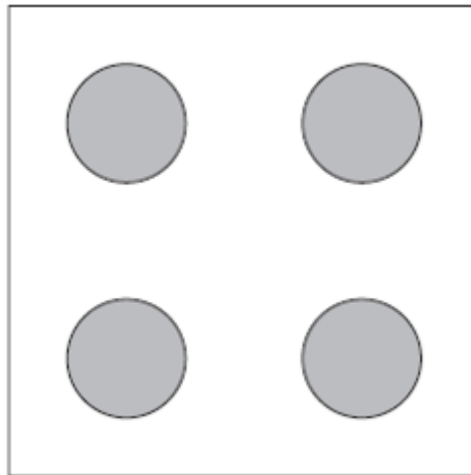
  // Change the values of all variables
  circleX = circleX + change;
  circleY = circleY + change;
  circleW = circleW + change;
  circleH = circleH - change;
  circleStroke = circleStroke - change;
  circleFill = circleFill + change;
}
```



# Exercise 3

---

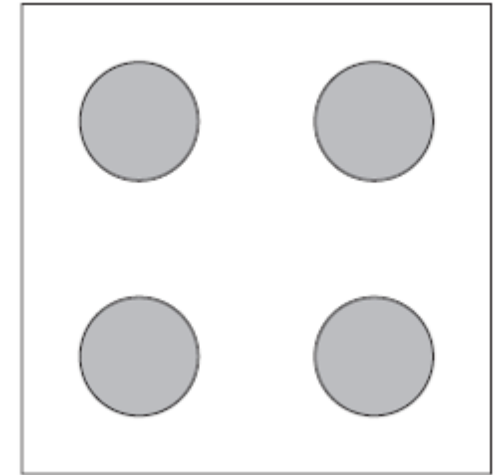
- ▶ Write code that draws the following screenshot with variables.



# Solution to Exercise 3 (1)

---

```
int firstCircleX;  
int firstCircleY;  
int secondCircleX;  
int secondCircleY;  
int thirdCircleX;  
int thirdCircleY;  
int fourthCircleX;  
int fourthCircleY;  
int circleWidth;  
int circleHeight;  
int backgroundColor;  
int fillColor;
```

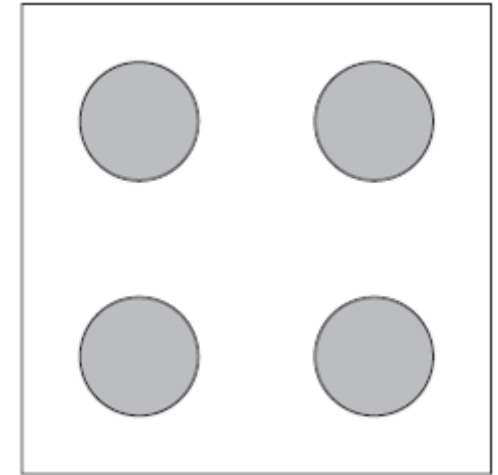


Declaration of variables.

# Solution to Exercise 3 (1)

---

```
void setup() {  
  size(200,200);  
  firstCircleX = 50;  
  firstCircleY = 50;  
  secondCircleX = 150;  
  secondCircleY = 50;  
  thirdCircleX = 50;  
  thirdCircleY = 150;  
  fourthCircleX = 150;  
  fourthCircleY = 150;  
  circleWidth = 50;  
  circleHeight = 50;  
  backgroundColor = 255;  
  fillColor = 150;  
}
```

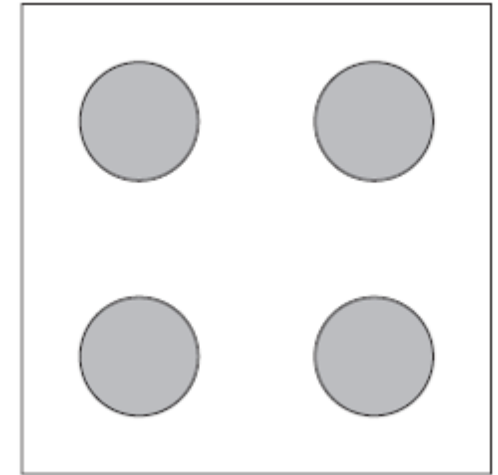


Assignment of values to variables.

# Solution to Exercise 3 (1)

---

Draw the ellipses.



```
void draw() {  
  background(backgroundColor);  
  fill(fillColor);  
  ellipse(firstCircleX, firstCircleY, circleWidth, circleHeight);  
  ellipse(secondCircleX, secondCircleY, circleWidth, circleHeight);  
  ellipse(thirdCircleX, thirdCircleY, circleWidth, circleHeight);  
  ellipse(fourthCircleX, fourthCircleY, circleWidth, circleHeight);  
}
```

# System Variables

---

- ▶ As we saw with `mouseX`, `mouseY`, `pmouseX`, `pmouseY` Processing has a set of convenient predefined **system variables** available for being used.
  - ▶ These are commonly needed pieces of data associated with all sketches (such as the width of the window, the key pressed on the keyboard, etc.).
- ▶ When naming your own variables, it is best to **avoid system variable names**.
  - ▶ however, if you inadvertently use one, your variable will become primary and override the system one.



# List of system variables

---

`width` - Width (in pixels) of sketch window.

`height` - Height (in pixels) of sketch window.

`frameCount` - Number of frames processed.

`frameRate` - Rate that frames are processed (per second).

`displayWidth` - Width (in pixels) of entire screen.

`displayHeight` - Height (in pixels) of entire screen.

`key` - Most recent key pressed on the keyboard.

`keyCode` - Numeric code for key pressed on keyboard.

`keyPressed` - True or false? Is a key pressed?

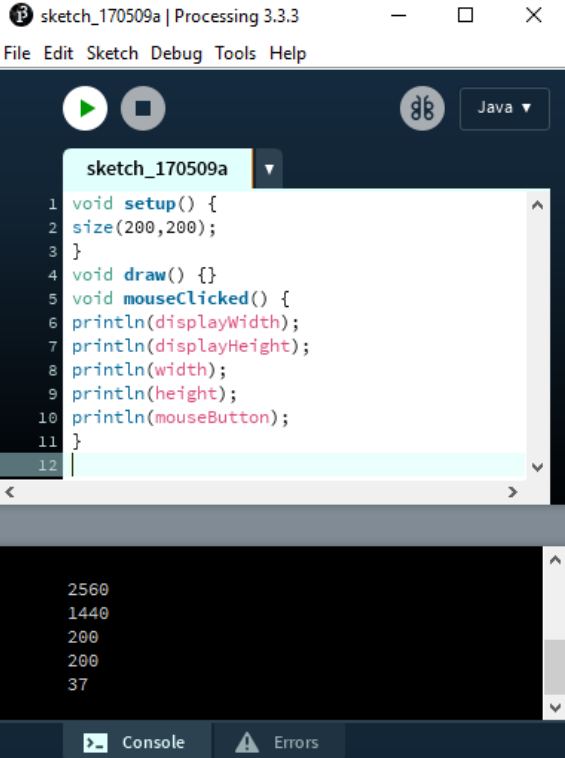
`mousePressed` - True or false? Is the mouse pressed?

`mouseButton` - Which button is pressed? Left, right, or center?

# Useful system functions: println(...)

- ▶ Sometimes, it is useful to display text information in the console of the Processing Development Environment (located at the bottom).
- ▶ This is accomplished using the `println("string")` function, which takes one argument, a String of characters enclosed in quotes.
- ▶ When the program runs, Processing displays that String in the message window. This ability to print the message window comes in handy when attempting to **debug the values of variables**.

```
void setup() {  
    size(200,200);  
}  
  
void draw() {}  
  
void mouseClicked() {  
    println(displayWidth);  
    println(displayHeight);  
    println(width);  
    println(height);  
    println(mouseButton);  
}
```



The screenshot shows the Processing IDE interface. The title bar reads "sketch\_170509a | Processing 3.3.3". The menu bar includes "File", "Edit", "Sketch", "Debug", "Tools", and "Help". The code editor displays the following code:

```
1 void setup() {  
2   size(200,200);  
3 }  
4 void draw() {}  
5 void mouseClicked() {  
6   println(displayWidth);  
7   println(displayHeight);  
8   println(width);  
9   println(height);  
10  println(mouseButton);  
11 }  
12
```

The console window at the bottom shows the output of the program:

```
2560  
1440  
200  
200  
37
```

# Useful system functions: `frameRate (...)`

---

- ▶ The function `frameRate(n)` requires an integer  $n$  between 1 and 60 and enforces the speed at which Processing will cycle through `draw()`.
- ▶ For example, `frameRate(30)` means 30 frames per second, a traditional speed for computer animation.
- ▶ If you do not include `frameRate(n)`, Processing will attempt to run the sketch at 60 frames per second.
  - ▶ Since computers run at different speeds, `frameRate(n)` is used to make sure that your sketch is consistent across multiple computers.



# Using System Variables

```
void setup() {  
  size(200,200);  
  frameRate(30);  
}
```

The function `frameRate()` is used to color a rectangle.

```
void draw() {  
  background(100);  
  stroke(255);  
  fill(frameCount/2);  
  rectMode(CENTER);  
  rect(width/2,height/2,mouseX + 10,mouseY + 10);  
}
```

`frameCount` is used to color a rectangle.

```
void keyPressed() {  
  println(key);  
}
```

The rectangle will always be in the middle of the window if it is located at:  $(width/2, height/2)$ .

Width and height change depending on the mouse location.

# Useful system functions: `random (...)`

---

- ▶ The function `random(n1, n2)` function requires two arguments and returns a **random floating number** ranging from the first argument `n1` to the second `n2`.

- ▶ The second argument must be larger than the first for it to work properly.

```
float w = random(1, 100);
```

It returns a **float** number.

- ▶ However, if you want a random integer, you can convert the result of the random function to an `int`.

```
int w = (int) random(1, 100);  
rect(100, 100, w, 50);
```

A random integer between 1 and 100. We force the float value returned by the `random(...)` function to be an integer value

- ▶ The process of converting one data type to another is referred to as **casting**.

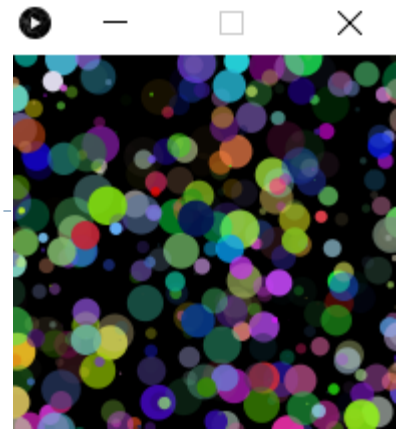
# Example with random (...)

---

```
float r;
float g;
float b;
float a;
float diam;
float x;
float y;

void setup() {
  size(200,200);
  background(0);
}

void draw() {
  // Fill all variables
  // with random values
  r = random(0,255);
  g = random(0,255);
  b = random(0,255);
  a = random(0,255);
  diam = random(0,20);
  x = random(0,width);
  y = random(0,height);
  // Use values to draw an ellipse
  noStroke();
  fill(r,g,b,a);
  ellipse(x,y,diam,diam);
}
```



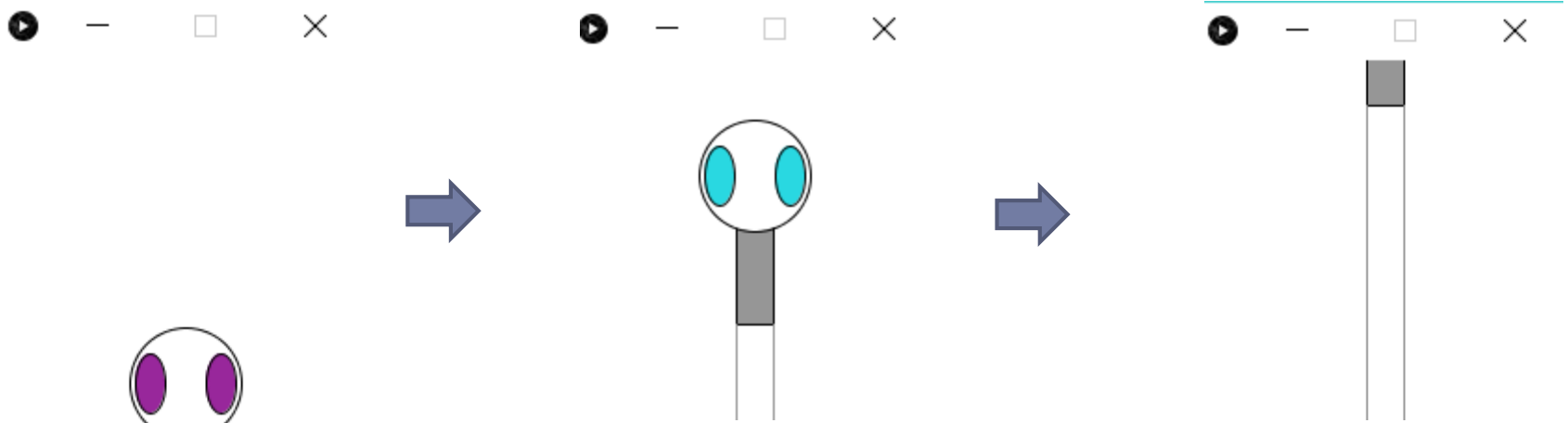
# Exercise 4

---

Redesign the alien by adding two features:

**New feature #1** The alien will rise from below the screen and fly off into space (above the screen). No mouse interaction is allowed.

**New feature #2** —Alien's eyes will be colored randomly as the alien moves.



# Solution to Exercise 4

---

```
float bodyX;  
float bodyY;  
float eyeR;  
float eyeG;  
float eyeB;
```

Declaring variables. `bodyX` and `bodyY` are for feature #1. `eyeR`, `eyeG`, `eyeB` are for feature #2.

```
void setup() {  
  size(200,200);  
  bodyX = width/2; // The alien always starts in the middle  
  bodyY = height + 100; // The alien starts below the screen  
}
```

```
void draw() {  
  background(255);  
  rectMode(CENTER);  
  
  // Draw alien's body  
  stroke(0);  
  fill(150);  
  rect(bodyX,bodyY,20,100);  
}
```

`bodyX` `bodyY` are used for the body location. They are initialized based on the size of the window. Note we can not initialize these variables before `size(...)` is called since we are using the system variables `width` and `height`. Their value is known only after the execution of the `size(...)` function.

# Solution to Exercise 4

---

```
// Draw alien's head
stroke(0);
fill(255);
ellipse(bodyX, bodyY-30, 60, 60);
```

```
// Draw alien's eyes
eyeR = random(255);
eyeG = random(255);
eyeB = random(255);
fill(eyeR, eyeG, eyeB);
ellipse(bodyX - 19, bodyY - 30, 16, 32);
ellipse(bodyX + 19, bodyY - 30, 16, 32);
```

**eyeR, eyeG, eyeB are given random values and used in the fill(...) function.**

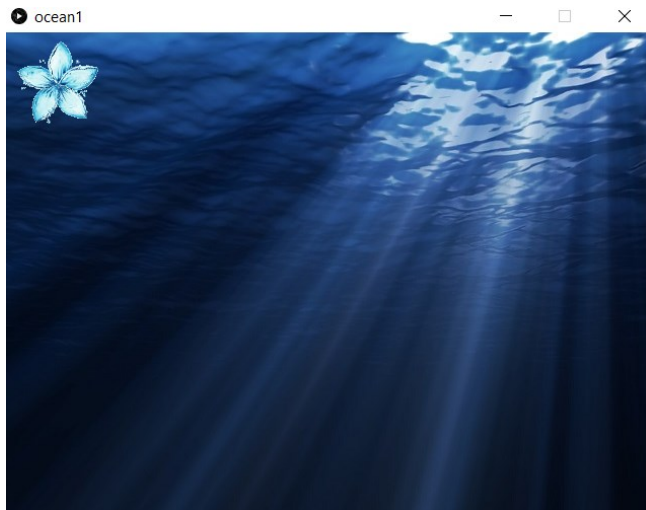
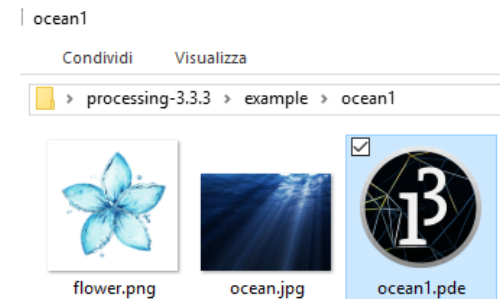
```
// Draw alien's legs
stroke(150);
line(bodyX - 10, bodyY + 50, bodyX - 10, height);
line(bodyX + 10, bodyY + 50, bodyX + 10, height);
```

```
// The alien moves up
bodyY = bodyY - 1;
}
```

**bodyY is decreased by one so that alien moves upward on the screen.**

# Images in Processing

- ▶ A digital image is nothing more than numbers indicating variations of red, green, and blue at a particular location on a grid of pixels.
- ▶ The basic way to **load** and **display images** is `PImage`.
- ▶ Download file `ocean1.zip` from the following link:  
<http://www.dis.uniroma1.it/~leotta/interactiondesign/2017-2018/ocean1.zip>
- ▶ Unzip the file and open `ocean1.pde` from folder `ocean1`.
  - ▶ The code allows to place two images on the window.



```
ocean1 | Processing 3.3
File Edit Sketch Debug Tools Help

ocean1
PImage ocean;
PImage flower;

void setup() {
  size(640, 480);
  ocean = loadImage("ocean.jpg");
  flower = loadImage("flower.png");
  image(ocean, 0, 0, 640, 480);
}

void draw() {
  image(flower, 0, 0, 100, 100);
}
```

# Images

---

```
PImage ocean;  
PImage flower;
```

Declaration of two variables of kind `PImage`.

```
void setup() {  
  size(640, 480);  
  ocean = loadImage("ocean.jpg");  
  flower = loadImage("flower.png");  
  image(ocean, 0, 0, 640, 480);  
}
```

Initialize the variables to load the image files we want to use with the function: `loadImage("nomeFile")`

Use the variable to draw the image with the function:

`image(imgName, xPosition, yPosition, width, height)`

```
void draw() {  
  image(flower, 0, 0, 100, 100);  
}
```

The `image(...)` function works also with 3 arguments only. In fact, it is not mandatory to specify the width and the height of the image. If the last two arguments are missing, the image will be displayed in its original size.

`image(imgName, xPosition, yPosition)`



# The function `tint (...)`

Function used to change the color and the transparency of an image.

```
tint (red, green, blue, transparency)
```

```
PImage ocean;  
PImage flower;
```

Declaration of two variables of kind `PImage`.

```
void setup() {  
  size(640, 480);  
  ocean = loadImage("ocean.jpg");  
  flower = loadImage("flower.png");  
  tint(0, 255, 0, 30);  
  image(ocean, 0, 0, 640, 480);  
}
```

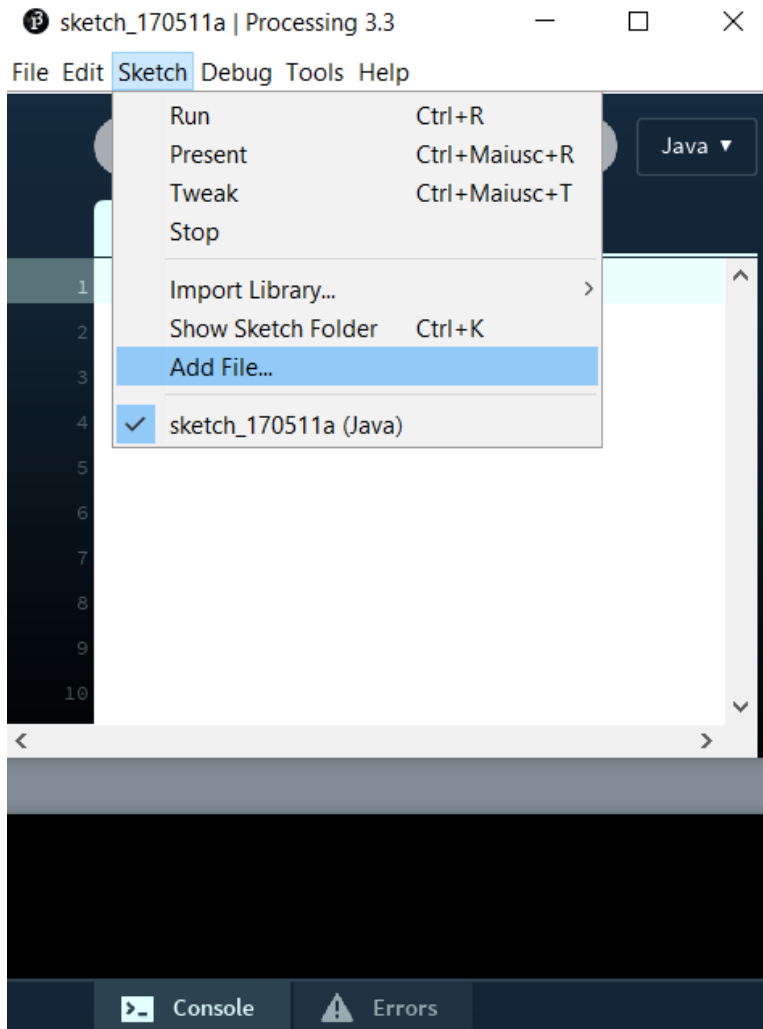
```
void draw() {  
  tint(255, 0, 0, 255);  
  image(flower, 0, 0, 100, 100);  
}
```



The image of the ocean is recolored in **green**.

The image of the flower is recolored in **red**.

# Inserting images via the Processing Development Environment



Images can be added to the data folder automatically via:

*Sketch* → *Add File...*

or manually:

*Sketch* → *Show Sketch Folder*

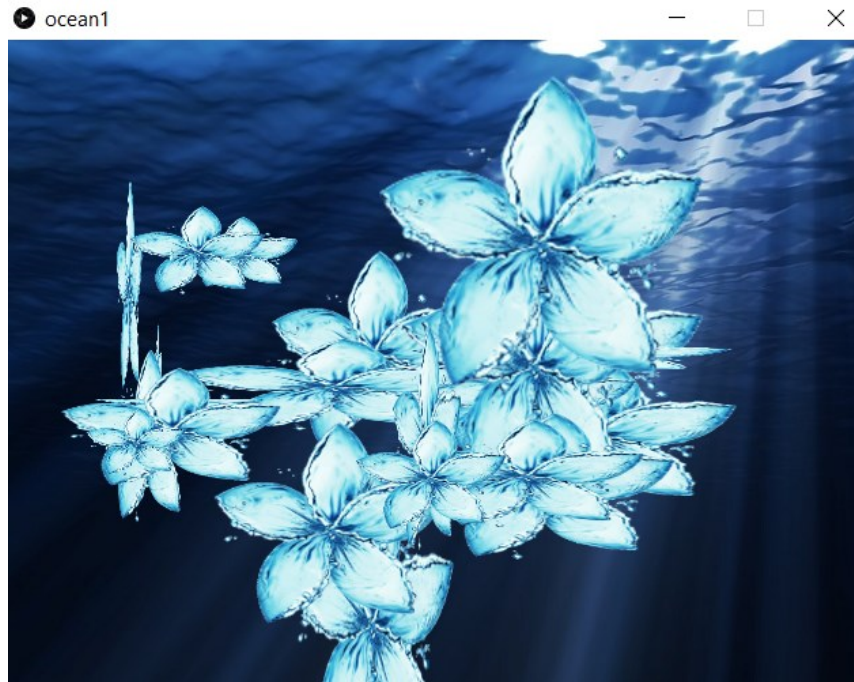
This will open up the sketch folder. If there is no data directory, create one. Otherwise, place your image files inside.

*Processing* accepts the following file formats for images: GIF, JPG, TGA, and PNG.

# Exercise 5

---

Starting from the previous example, change the code in order to generate a new flower image in random positions and of random size when the mouse is clicked.



# Solution to Exercise 5

---

```
PImage ocean;
PImage flower;
float x;
float y;
float w;
float h;

void setup() {
  size(640, 480);
  ocean = loadImage("ocean.jpg");
  flower = loadImage("flower.png");
  image(ocean, 0, 0, 640, 480);
  x = 0;
  y = 0;
  w = 0;
  h = 0;
}
```

# Solution to Exercise 5

---

```
void draw() {}

void mouseClicked() {
  x = random(0, 300);
  y = random(0, 300);
  w = random(0, 300);
  h = random(0, 300);
  image(flower, x, y, w, h);
}
```