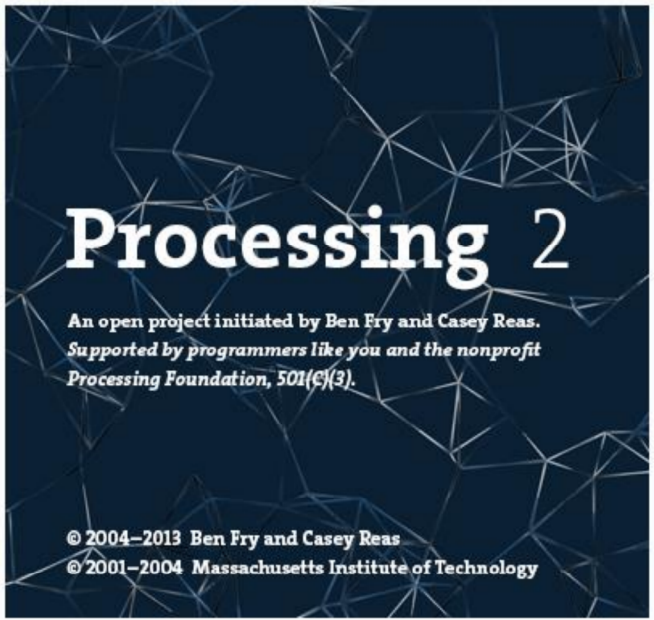
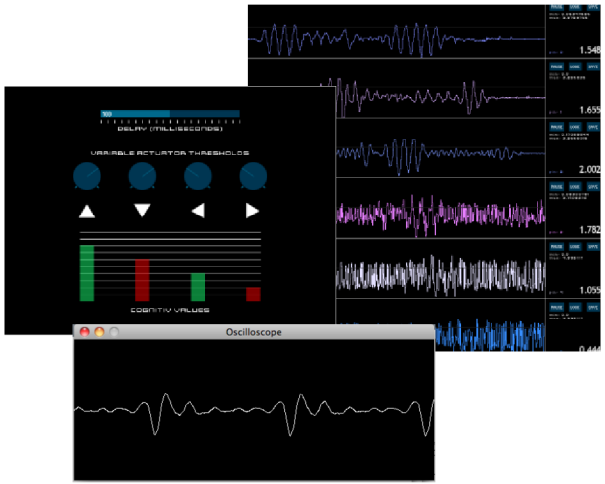
Processing: Diseño de Interfaces

Processing: Diseño de Interfaces



- Qué es Processing?
- Comenzando con Processing
- Mostrando Texto
- Eventos del Mouse y del Teclado
- Comunicando Galileo con Processing
- Comunicando Processing con Galileo



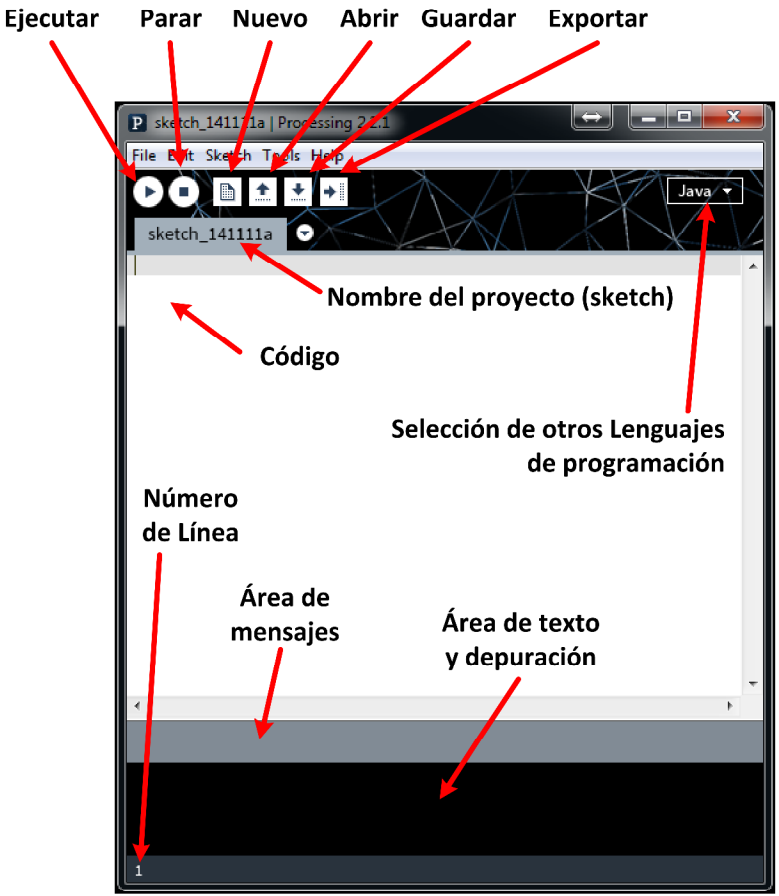
Qué es Processing?

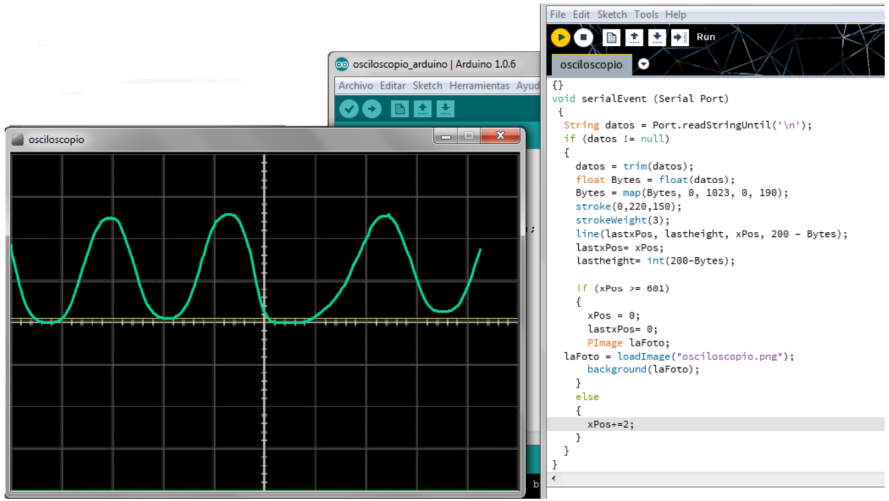
Processing:

Es un lenguaje de programación y un entorno de desarrollo de software con una curva de aprendizaje rápida, con el que se pueden diseñar interfaces gráficas para monitorear y controlar distintas tareas. Es un buen complemento para la tarjeta de desarrollo Galileo.

1. Descarga gratuita ya que es un Programa de Código Abierto.
2. Se puede ejecutar en Linux,Mac OS X y Windows.
3. Mas de 100 bibliotecas que extienden las capacidades del software.
4. Se puede combinar programación de Processing con Java puro.

IDE Processing





Comenzando con Processing

El entorno de programación de Processing permite la creación de imágenes, animaciones y medios de interacción.

Comenzaremos con algunos conceptos básicos y figuras simples que mas adelante nos permitirán diseñar interfaces mas complejas.

- Funciones Básicas
- Figuras Básicas
- Agregando color

Descarga: http://processing.org/

Funciones Básicas

Processing, al igual que Arduino tiene dos funciones básicas:

- **setup( )**
- **draw( )**

La función «**setup( )**» se ejecuta una sola vez al inicio del programa.

La función «**draw( )**» hace las veces de la función «**loop( )**» de Arduino, la cual se ejecuta repetitivamente mientras el programa se encuentre ejecutándose.

Otras dos funciones básicas que necesitaremos son:

- **size( )**
- **background( )**

size( )
background( )

size( )

Define la dimensión de la ventana de la interfaz en pixeles. Esta función debe ser la primer línea de código dentro de la función **setup( )**

Sintaxis:

size(w, h)

w -> ancho de la ventana (int)
h -> alto de la ventana (int)

background( )

Establece el color usado para el fondo de la ventana de la interfaz. El fondo predeterminado es de color gris claro.

Sintaxis:

background(grayscale)
background(R,G,B)
background(rgb)

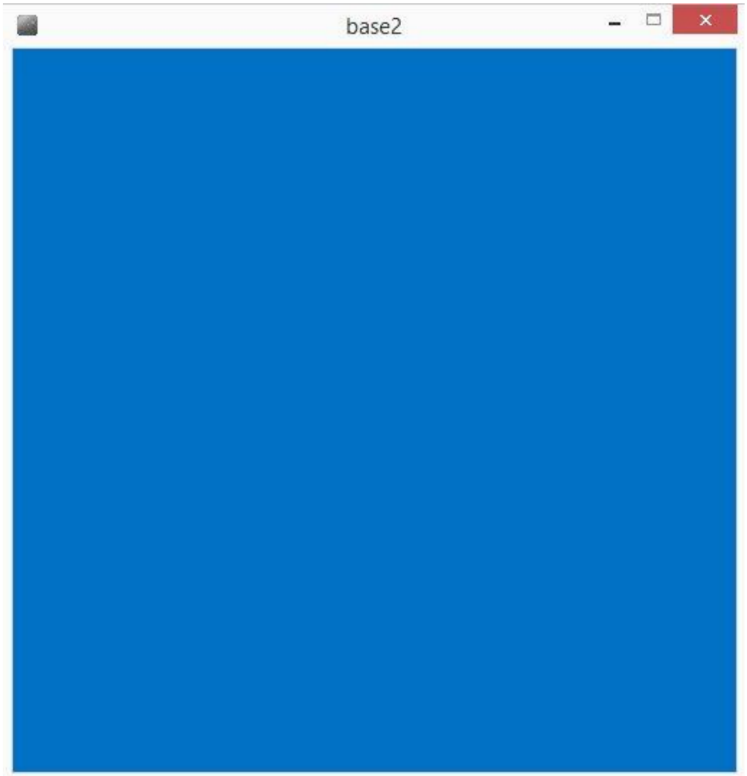
rgb -> Codificación hexadecimal

#FFFFFF -> Blanco

Código
base.pde

```
void setup(){
  size(500,500);
}

void draw(){
}
```

Crea una ventana de
500x500 pixeles



Código
base2.pde

Crea una ventana de

500x500 pixeles

```
void setup(){
  size(500,500);
  background(#0071c5);
}

void draw(){
}
```

Fondo de la ventana

Color: Azul “**Intel**”

RGB: 0, 113, 197

Figuras Básicas

En Processing podemos comenzar nuestros dibujos con formas muy básicas como líneas, elipses y cuadriláteros con las siguientes funciones::

line( )

Dibuja una línea en la pantalla entre dos puntos indicados. La línea dibujada tiene un grosor de **1 pixel** y color **negro** por defecto

Sintaxis:

line(x1,y1,x2,y2)

rect( )

Dibuja un rectángulo en la pantalla. Las primeras dos coordenadas corresponden a la esquina superior izquierda, la tercera es el ancho y la cuarta la altura.

Sintaxis:

rect(x1,y1,w,h)

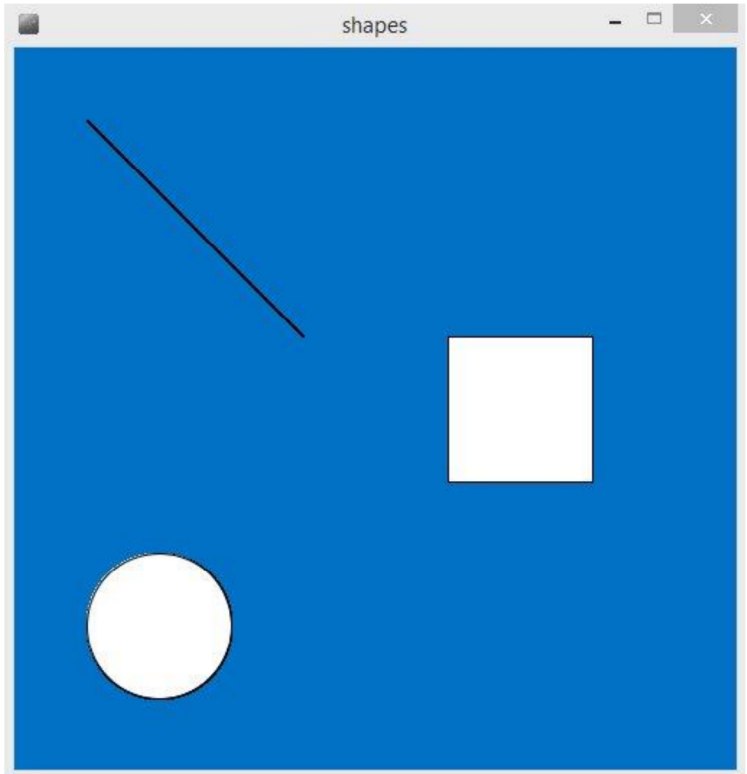
ellipse( )

Dibuja una elipse en la pantalla. Las primeras dos coordenadas corresponden al centro de la elipse, la tercera y la cuarta al ancho y la altura respectivamente.

Sintaxis:

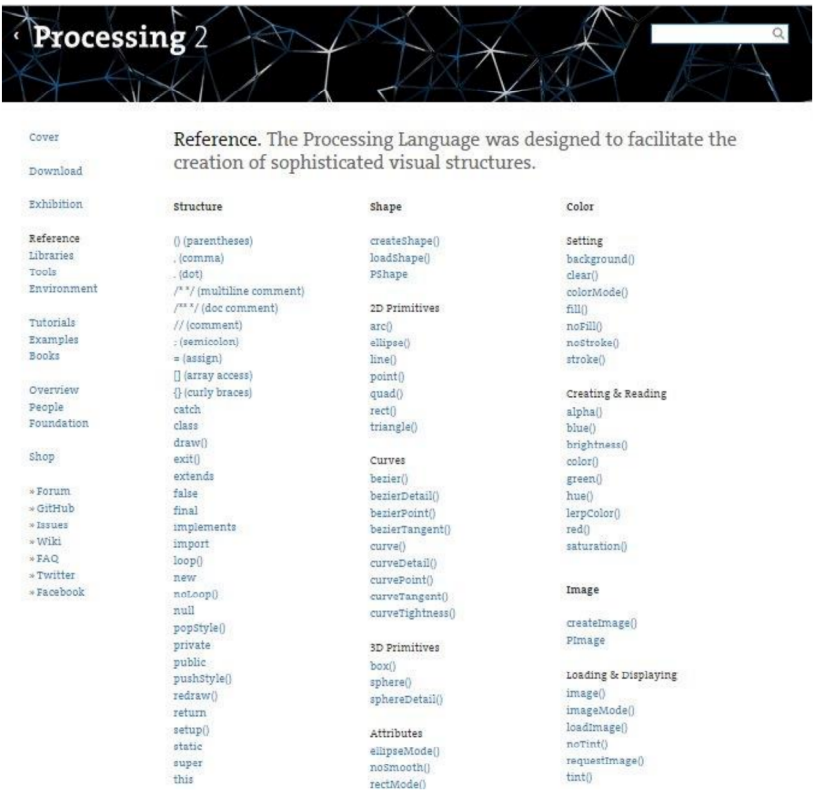
ellipse(x,y,w,h)

Código
shapes.pde




```
void setup(){
  size(500,500);
  background(#0071c5);
}

void draw(){
  line(50,50,200,200);
  rect(300,200,100,100);
  ellipse(100,400,100,100);
}
```

Processing Language Reference



El lenguaje Processing es diseñado con la finalidad de proporcionar una forma rápida para el diseño de prototipos de software, con el cual podemos crear desde formas muy básicas, hasta estructuras visuales muy sofisticadas.

Al igual que Arduino, Processing nos proporciona un amplio compendio de Referencia de su lenguaje, donde podemos encontrar la explicación, sintaxis y ejemplos de todas las funciones implementadas nativamente en su IDE y de algunas bibliotecas que también vienen incorporadas en él.

https://www.processing.org/reference/

noFill()

noStroke()

Agregando Color

Podemos especificar el color del relleno de las figuras y de su contorno, así como el grosor de éste:

fill( )

Configura el color del relleno de las figuras. El color por default de las figuras es el **blanco**.

Sintaxis:

```
fill(grayscale)
fill(R,G,B)
fill(rgb)
```

stroke( )

Configura el color de las líneas y de los contornos de las figuras.

Sintaxis:

```
stroke(grayscale)
stroke(R,G,B)
stroke(rgb)
```

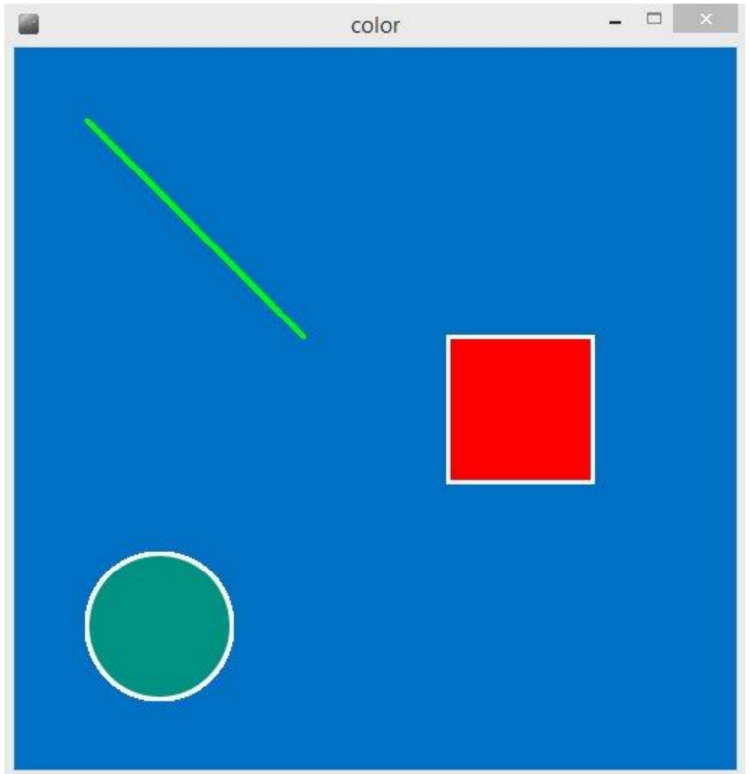
strokeWeight( )

Configura el grosor de las líneas y de los contornos de las figuras. El valor es expresado en pixeles.

Sintaxis:

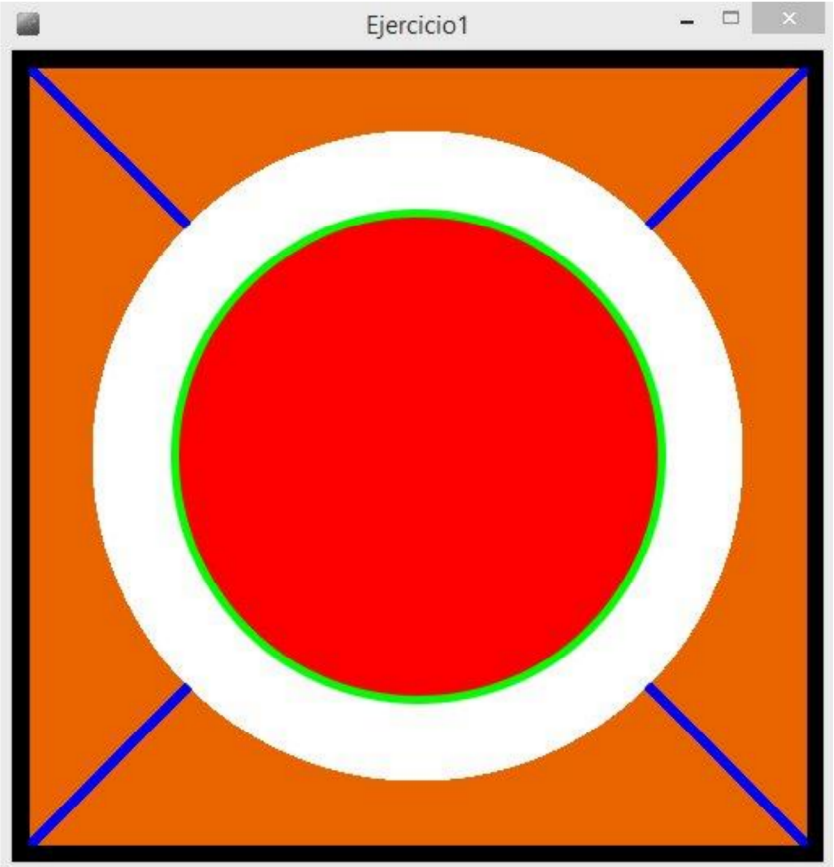
```
strokeWeight(weight)
```

Código
color.pde




```
void setup(){
  size(500,500);
  background(#0071c5);
}

void draw(){
  fill(255,0,0);
  stroke(0,255,0);
  strokeWeight(3);
  line(50,50,200,200);
  stroke(255,255,255);
  rect(300,200,100,100);
  fill(#009383);
  ellipse(100,400,100,100);
}
```

Reto:

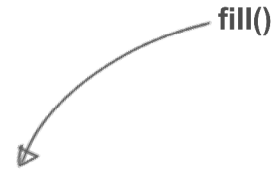


Mostrando Texto

Además de crear figuras, también podemos agregar texto a nuestras interfaces, lo cual nos permitirá agregar leyendas, instrucciones, sugerencias, comentarios, etc.

Igualmente podemos darle formato a este texto, como por ejemplo, modificar el tamaño, el color, su posición, la fuente tipográfica, entre otros atributos.

text()
textSize()
textAlign()



text()

Despliega texto en la pantalla.

Sintaxis:

text(texto,x,y)
text(num,x,y)

x,y -> coordenadas de la esquina inferior izquierda del texto.
num -> int o float: número a mostrar

textSize()

Configura el tamaño de la fuente. Se expresa en pixeles.

Sintaxis:

textSize(tamaño)

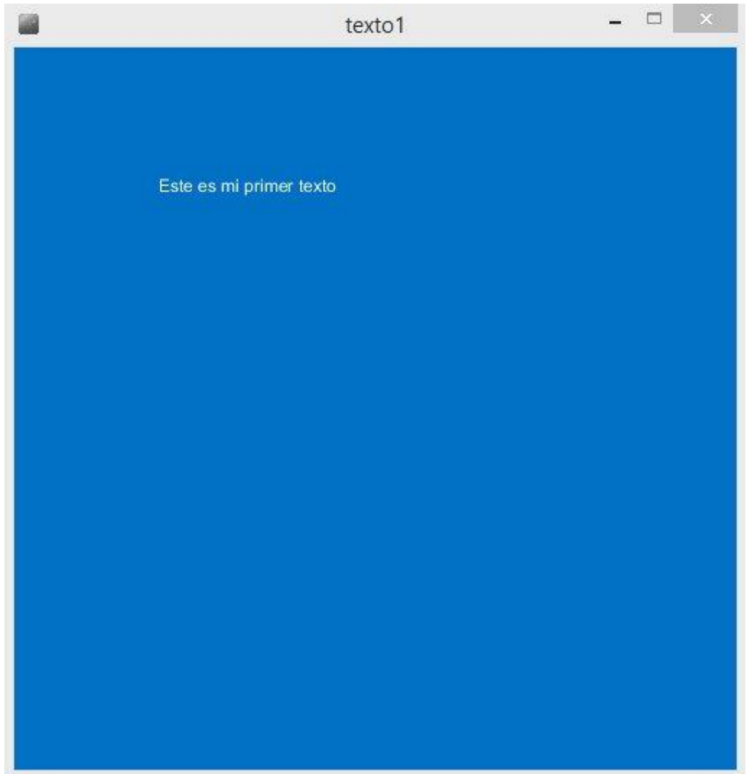
textAlign()

Configura la alineación del texto mostrado en relación con los valores de las coordenadas indicadas en la función text(): x, y.

Sintaxis:

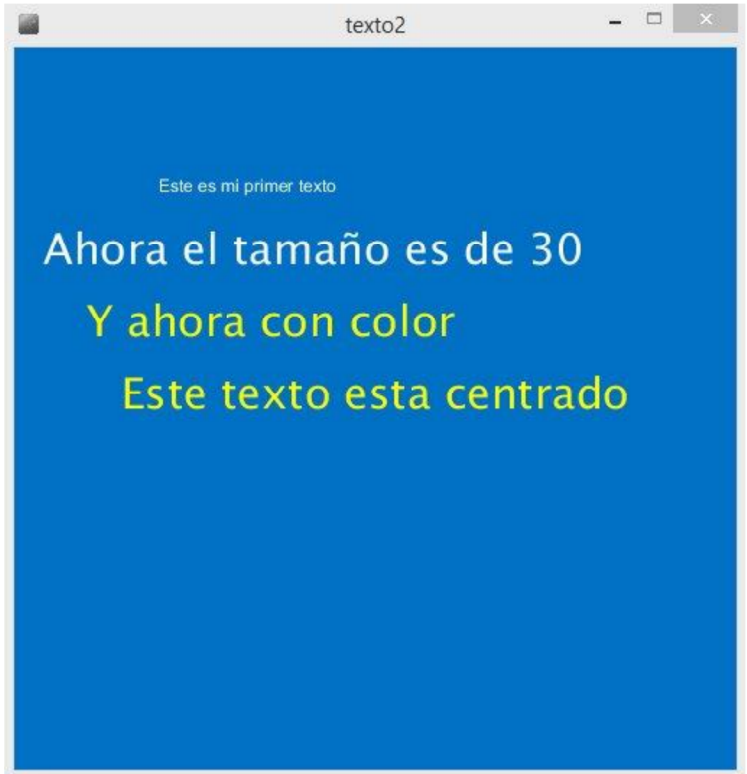
textAlign(alignX)

alignX -> LEFT, CENTER, RIGHT



Código
texto1.pde

```
void setup(){
  size(500,500);
  background(0,113,197);
  text("Este es mi primer texto",0,500);
}

void draw(){
}
```



Código
texto2.pde

```
void setup(){
  size(500,500);
  background(0,113,197);
  text("Este es mi primer texto",100,100);
  textSize(30);
  text("Ahora el tamaño es de 30",20,150);
  fill(255,255,0);
  text("Y ahora con color",50,200);
  textAlign(CENTER);
  text("Este texto esta centrado",width/2,width/2);
}

void draw(){
}
```

Eventos del Mouse

Processing nos proporciona algunas funciones para poder acceder a distintos eventos del mouse, además de proporcionarnos información sobre la posición del puntero como su posición en X y Y en ciertas variables predefinidas. Con estas funciones y variables podemos controlar un sinfín de aplicaciones, como por ejemplo, dibujar algo cuando demos un click en cierto lugar de la ventana.

Acciones del Mouse

Es importante definir que acciones del mouse puede detectar Processing:

Acción	Descripción
Clicked	Un botón del mouse es presionado y luego soltado
Pressed	Un botón del mouse es presionado y se mantiene así
Released	Un botón del mouse fue presionado, pero ahora se suelta
Moved	El mouse es movido sin que ningún botón sea presionado
Dragged	El mouse es movido con un botón presionado

Funciones del Mouse

Las acciones del mouse puede llamar las siguientes funciones y ejecutar el código que nosotros definamos dentro de ellas:

Variables Relacionadas

Tenemos a nuestra disposición dos variables que Processing nos proporciona en cualquier momento que las necesitemos a lo largo de nuestro programa y dentro de cualquier función, las cuales nos proporcionan mas información acerca de que es lo que el mouse está haciendo dentro de nuestro programa.

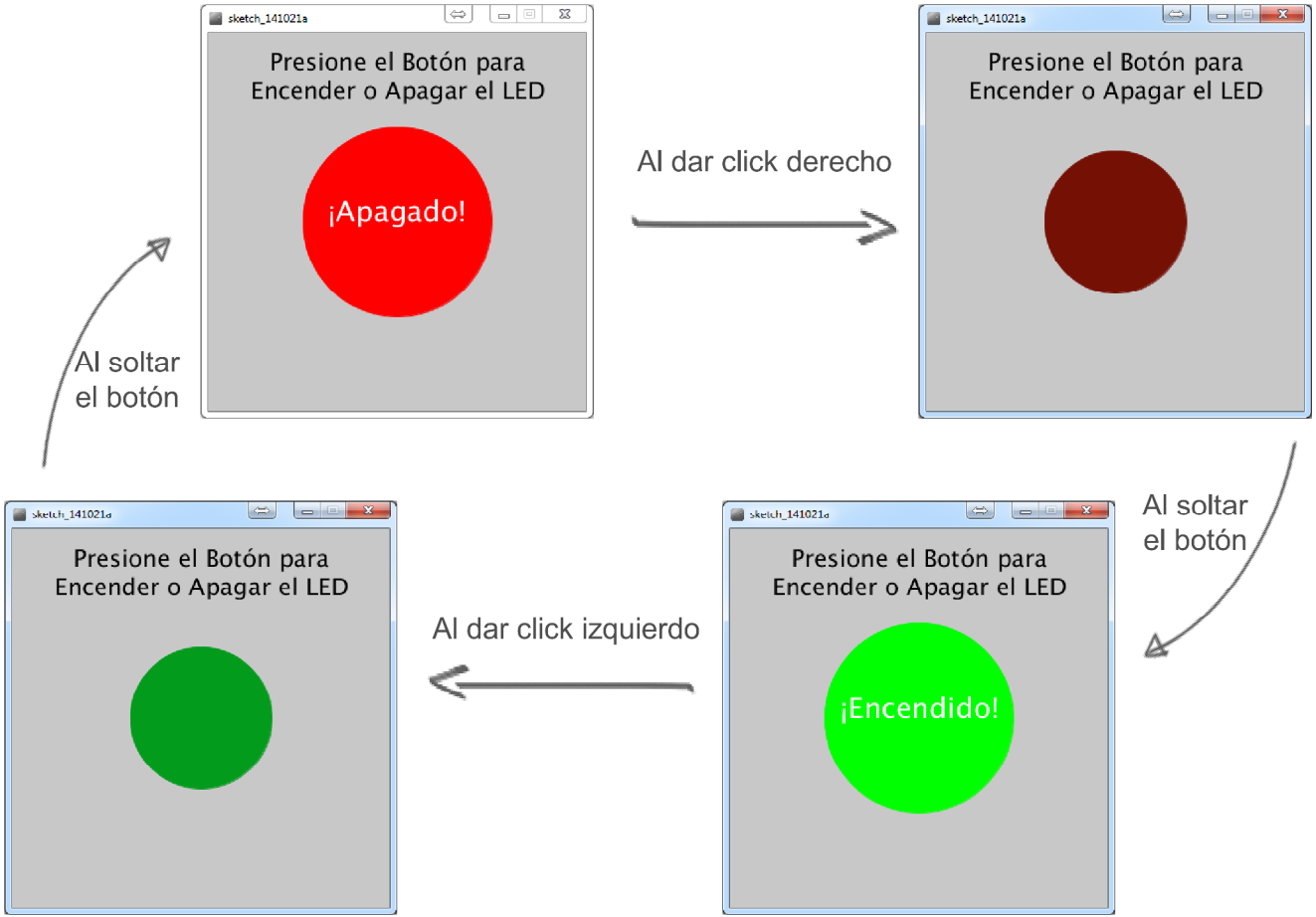
- **mouseX** Nos proporciona la posición X del mouse en la ventana
- **mouseY** Nos proporciona la posición Y del mouse en la ventana
- **mousePressed** Variable booleana, TRUE / FALSE
- **mouseButton** Botón presionado, LEFT, RIGHT, CENTER

Código
mouse.pde

```
void setup() {
  size(500, 500);
  background(0);
  textAlign(CENTER);
  textSize(24);
  fill(255);
  text("El mouse no está haciendo nada",
  width/2, height/2);
}

void draw() {
}

void mousePressed() {
  background(100, 100, 0);
  text("El mouse fue presionado",
  width/2, height/2);
  if ( mouseButton == LEFT) {
    text("y fue el botón izquierdo",
    width/2, height/2 + 40);
  }
  if (mouseButton == RIGHT) {
    text("y fue el boton derecho",
    width/2, height/2 + 40);
  }
}
```

```
void mouseReleased() {
  background(100, 0, 100);
  text("el botón fue soltado",
  width/2, height/2);
}

void mouseMoved() {
  background(150, 10, 70);
  text("El mouse fue movido",
  width/2, height/2);
}

void mouseDragged() {
  background(10, 70, 100);
  text("Se está arrastrando el mouse",
  width/2, height/2);
}
```

Reto:


sketch_141021a
Presione el Botón para
Encender o Apagar el LED
¡Apagado!
Al dar click derecho
sketch_141021a
Presione el Botón para
Encender o Apagar el LED
Al soltar
el botón
Al soltar
el botón
sketch_141021a
Presione el Botón para
Encender o Apagar el LED
¡Encendido!
Al dar click izquierdo
sketch_141021a
Presione el Botón para
Encender o Apagar el LED

Eventos del Teclado

Al igual que los eventos de mouse, es posible también detectar distintos eventos del teclado, lo que nos puede permitir tener mayor interacción con la tarjeta Galileo. Algunos de las acciones o eventos que podemos detectar es la presión de alguna tecla, determinar la tecla presionada, discriminar entre letras mayúsculas o minúsculas, entre otras cosas.

Tecla Presionada

Processing nos proporciona una función que es llamada cada vez que una tecla es presionada:

```
void keyPressed(){
}
```

Dentro de esta función podemos colocar todo el código que queremos que se ejecute cada vez cualquier tecla sea presionada, sin embargo aún no distinguimos específicamente cuál tecla se presionó.

Código
anykey.pde

```
int r,g,b;

void setup(){
  size(600,600);
  r = 0;
  g = 0;
  b = 0;
}

void draw(){
  background(r,g,b);
}

void keyPressed(){
  r = int(random(256));
  g = int(random(256));
  b = int(random(256));
}
```

Detectando la Tecla Presionada

Como ya vimos, cada que presionamos una tecla, la función keyPressed() es llamada. Además, la información acerca de ´cuál es la tecla presionada es almacenada en una variable llamada «key».

```
void setup(){
}

void draw(){
}

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

Código
rgbkey.pde

```
int r,g,b;

void setup(){
  size(600,600);
  r = 0;
  g = 0;
  b = 0;
}

void draw(){
  background(r,g,b);
}
```

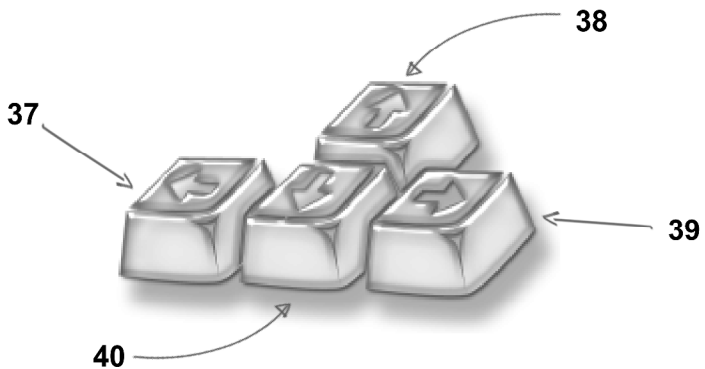
```
void keyPressed(){
  switch (key){
    case 'r':
      r=255;
      g=0;
      b=0;
      break;
    case 'g':
      r=0;
      g=255;
      b=0;
      break;
    case 'b':
      r=0;
      g=0;
      b=255;
      break;
    default:
      break;
  }
}
```

Detectando otras Teclas

En el primer ejemplo de esta sección, pudimos darnos cuenta que hay algunas teclas que no pudimos identificar, y simplemente se imprime un signo ?. Ahora, en el segundo ejemplo imaginemos que queremos detectar cuando las flechas sean presionadas, ¿cómo podemos escribir una tecla de estas dentro de las comillas?

Para esto, Processing también tiene a nuestra disposición la variable «keyCode» la cual almacena el número ASCII asociado a cada una de estas teclas:

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



Código
arrows.pde

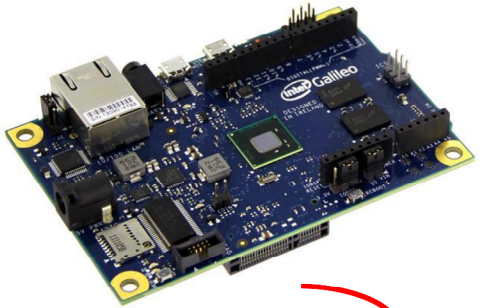
```
int r,g,b;

void setup(){
  size(600,600);
  r = 0;
  g = 0;
  b = 0;
}

void draw(){
  background(r,g,b);
}
```

```
void keyPressed(){
  switch (keyCode){
    case 37:
      r=255;
      g=0;
      b=255;
      break;
    case 38:
      r=255;
      g=255;
      b=0;
      break;
    case 39:
      r=0;
      g=255;
      b=255;
      break;
    default:
      break;
  }
}
```

Reemplaza con: LEFT

Reemplaza con UP

Reemplaza con RIGHT

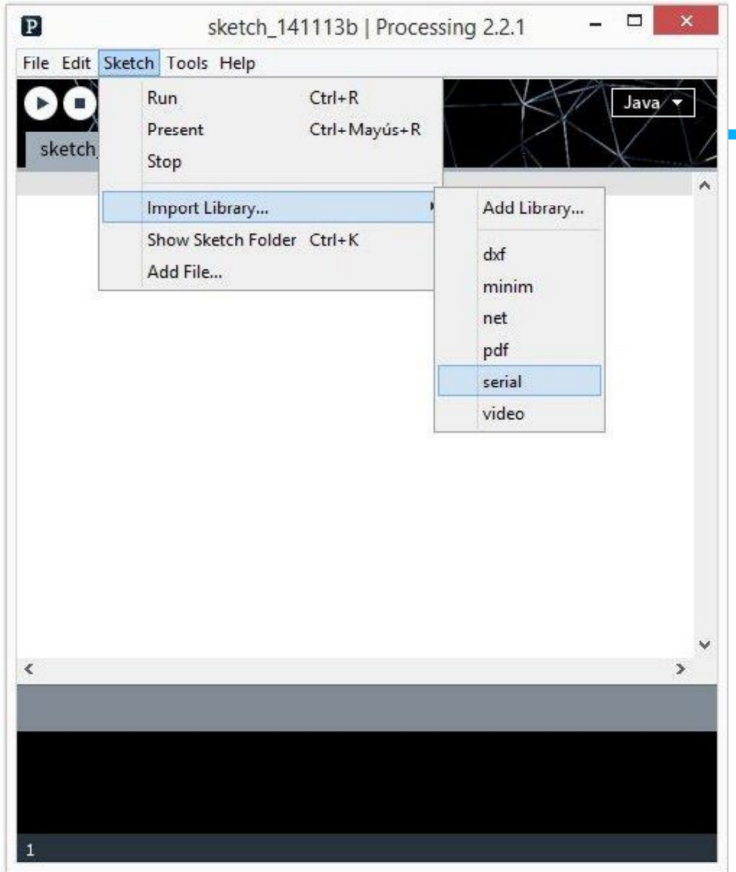


Comunicando Galileo con Processing

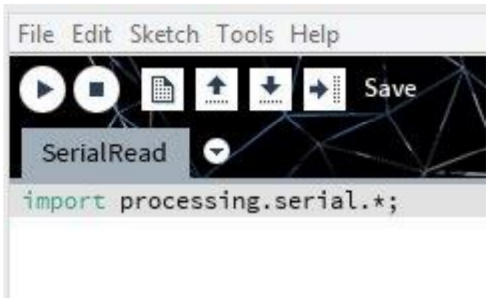
Para esta primer interacción entre ambas plataformas Hardware/Software, enviaremos datos utilizando comunicación serial, desde la tarjeta Galileo hasta la interfaz de Processing, es decir, realizaremos en tipo de “hola mundo”.

Lo primero que necesitamos sabes es cómo leer información del puerto serial.

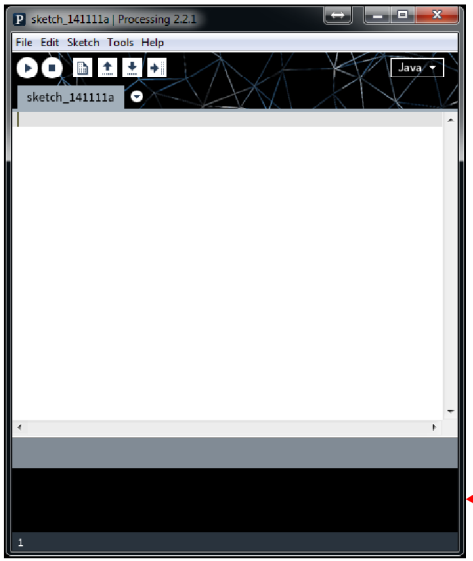
Cómo leer Datos Seriales



Lo primero es importar la biblioteca “serial” para tener acceso al puerto serial de la computadora para enviar y transmitir datos entre Processing y hardware externo:



Código
SerialRead.pde

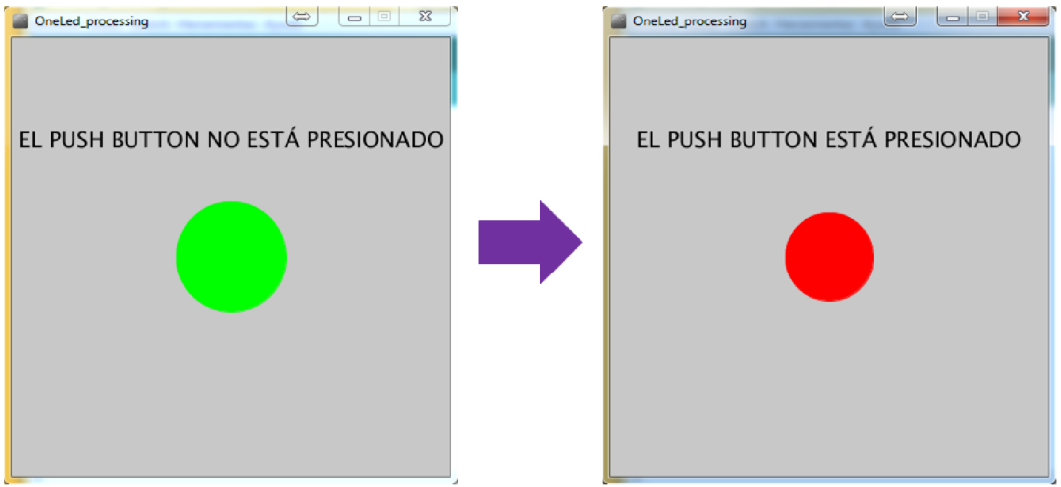



```
import processing.serial.*;

Serial miPuerto;

void setup(){
  miPuerto = new Serial(this,"COM5",9600);
}

void draw(){
  if (miPuerto.available() > 0){
   int dato = miPuerto.read();
   println(dato);
  }
}
```

```
void draw(){
}

void serialEvent(Serial miPuerto){
  int dato = miPuerto.read();
   println(dato);
}
```

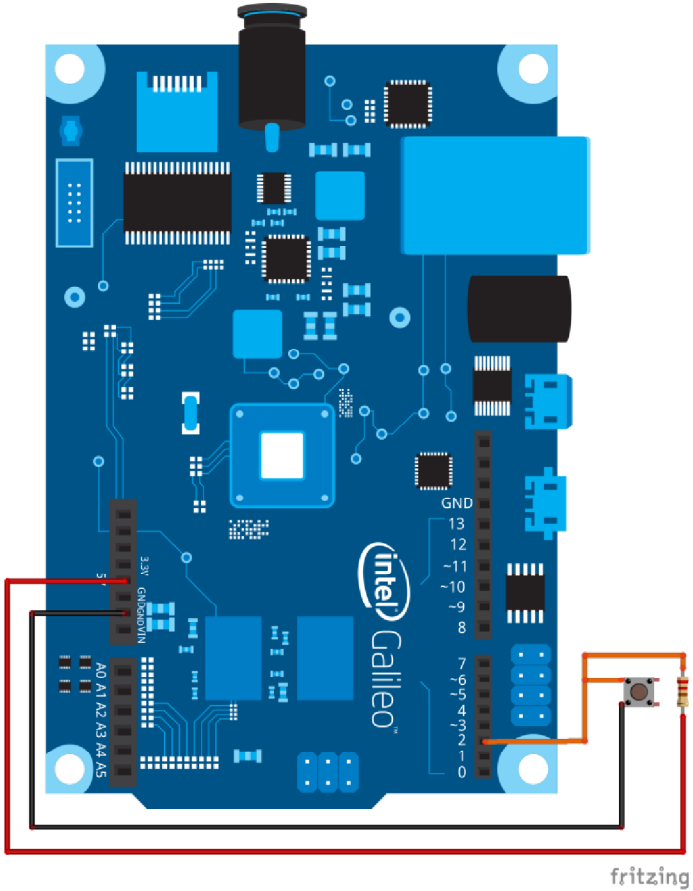
Comunicando Galileo con Processing

Para este primer ejemplo lo que haremos será enviar la información del estado de un push button conectado a la tarjeta Galileo, a una interfaz diseñada en Processing:



Esquemático

NOTA: Aunque este es el diagrama de conexión para este ejercicio, usaremos el botón A del Joystick:

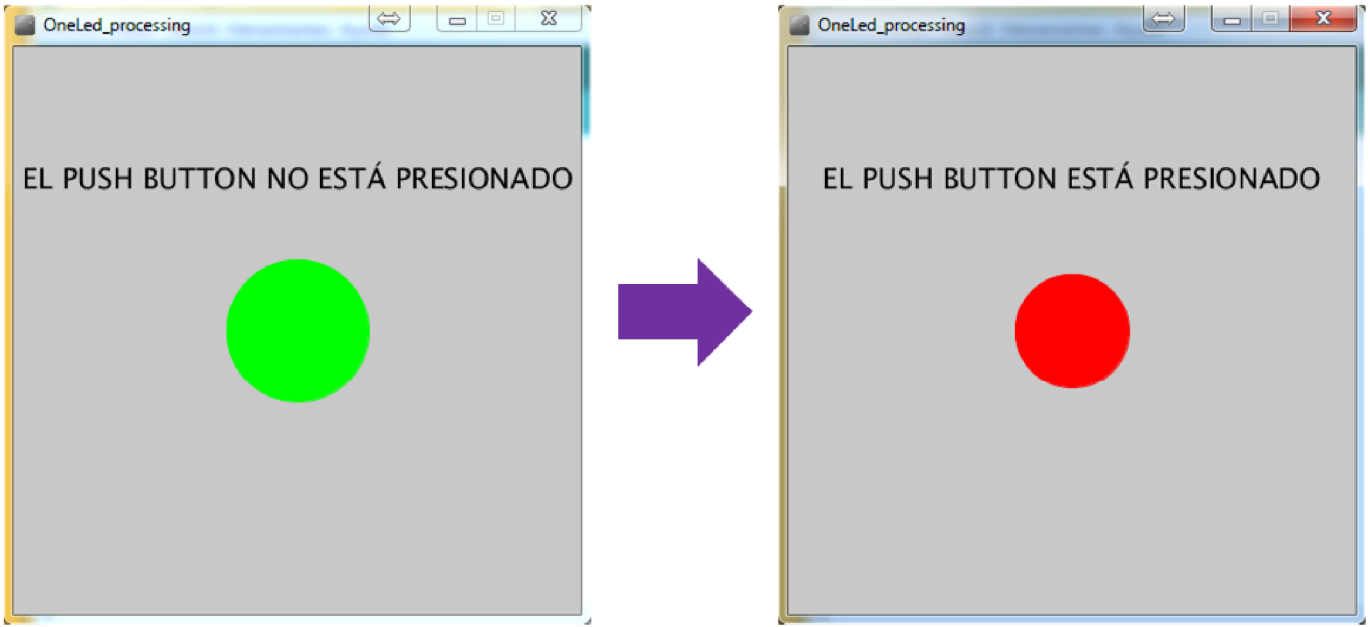


Galileo

Código
OneLed.ino

```
void setup() {
  pinMode(2, INPUT); // Configuración del pin 2 como entrada
  Serial.begin(9600); //Inicialización de comunicación serial a 9600 baudios
}

void loop() {
  // Transfiere a través del puerto serial el estado digital del pin 2
  Serial.print(digitalRead(2));
  delay(100); //retardo de 100 milisegundos
}
```

Reto:


OneLed_processing
EL PUSH BUTTON NO ESTÁ PRESIONADO
OneLed_processing
EL PUSH BUTTON ESTÁ PRESIONADO

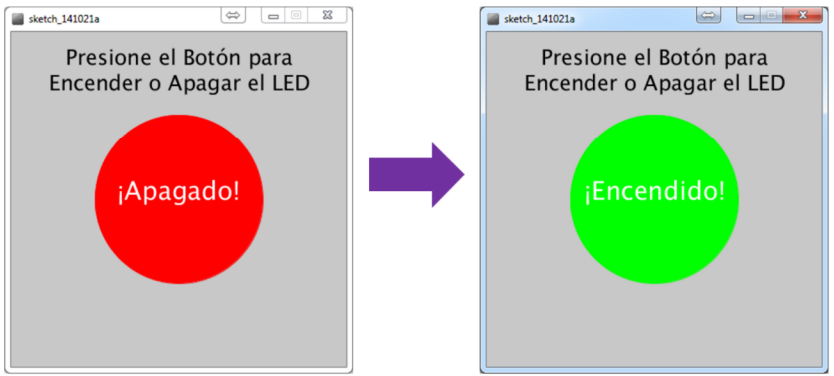
Comunicando Processing con Galileo

Ahora enviaremos datos a la tarjeta Galileo desde una interfaz diseñada en Processing para controlar una tarea, que para este caso se controlará el encendido y apagado de un LED.

Para este ejercicio necesitamos saber cómo escribir al puerto serial, para que los datos sean transmitidos hacía la Galileo.

Comunicando Processing con Galileo

La interfaz de Processing contará con un botón virtual, el cual al dar click derecho sobre éste, transmitirá serialmente un dato que le informará a la tarjeta Galileo que deberá de encender un LED y si se da click izquierdo, deberá apagarlo:

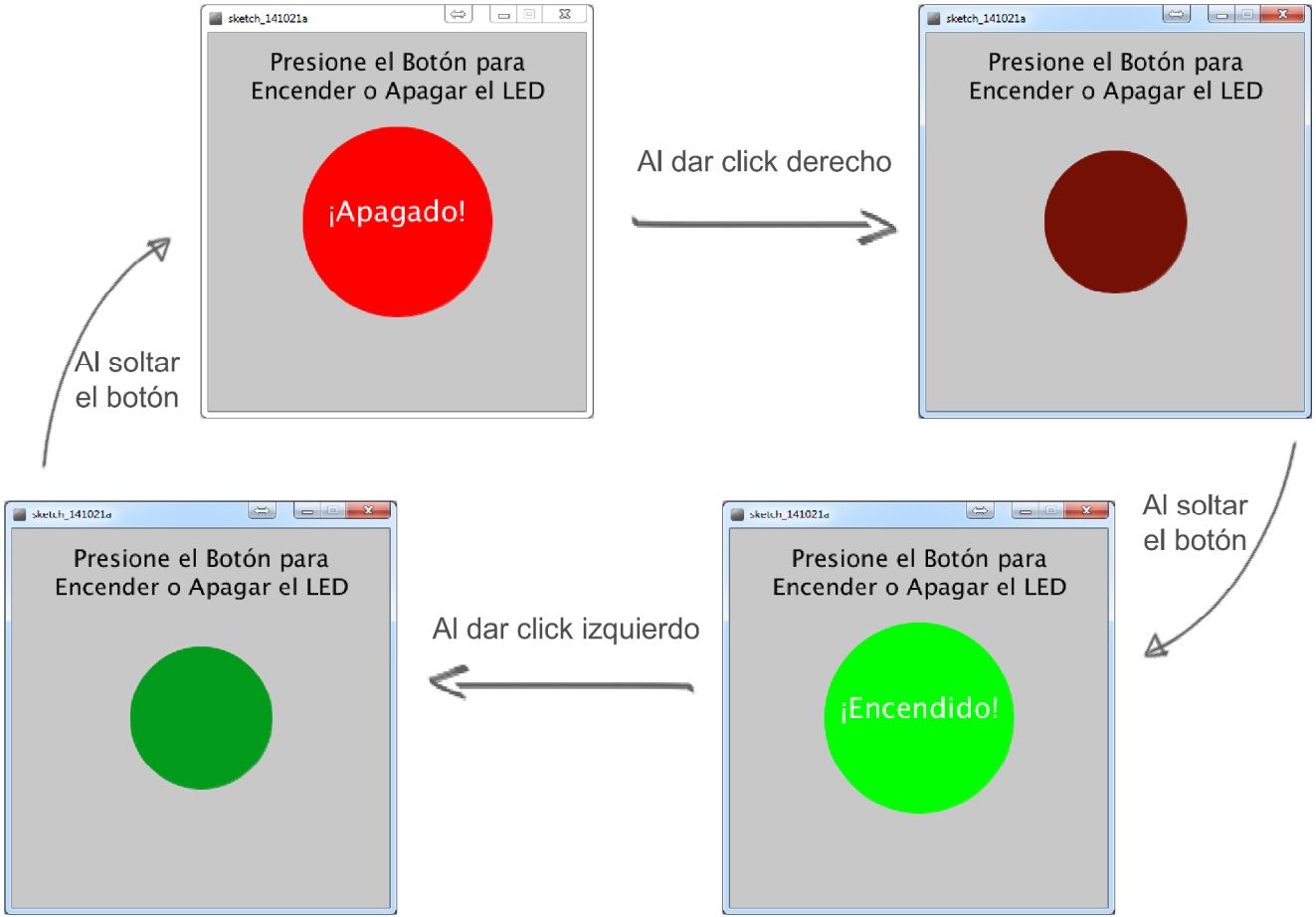


Galileo

Código
LED_on_off.ino

```
int byteRecibido=0;  //Variable que almacenará el byte recibido
const int led = 13;  //Pin al que se conectará el LED

void setup(){
  Serial.begin(9600);    //Inicialización del puerto serial a 9600 baudios
  pinMode(led, OUTPUT);  //Se declara el pin 13 como salida
  digitalWrite(led, LOW);//Apagar el LED inicialmente
}

void loop(){
  //Se comprueba si hay algún dato por leer en el buffer del puerto serial
  if (Serial.available() > 0){
    byteRecibido=Serial.read();  //Lectura del dato recibido
    if (byteRecibido=='A')       //Comprobar que el dato recibido sea una "A"
      digitalWrite(led, HIGH); //Encender el LED
    else if (byteRecibido=='B')
      digitalWrite(led, LOW);  //Apagar el LED
  }
}
```

Reto:


sketch_141021a
Presione el Botón para
Encender o Apagar el LED
¡Apagado!
Al dar click derecho
sketch_141021a
Presione el Botón para
Encender o Apagar el LED
Al soltar
el botón
Al soltar
el botón
sketch_141021a
Presione el Botón para
Encender o Apagar el LED
¡Encendido!
Al dar click izquierdo
sketch_141021a
Presione el Botón para
Encender o Apagar el LED