CLASE 3
INTERRUPCIONES Y PROCESSING

ARQUITECTURA DE COMPUTADORES Y ENSAMBLADORES 2

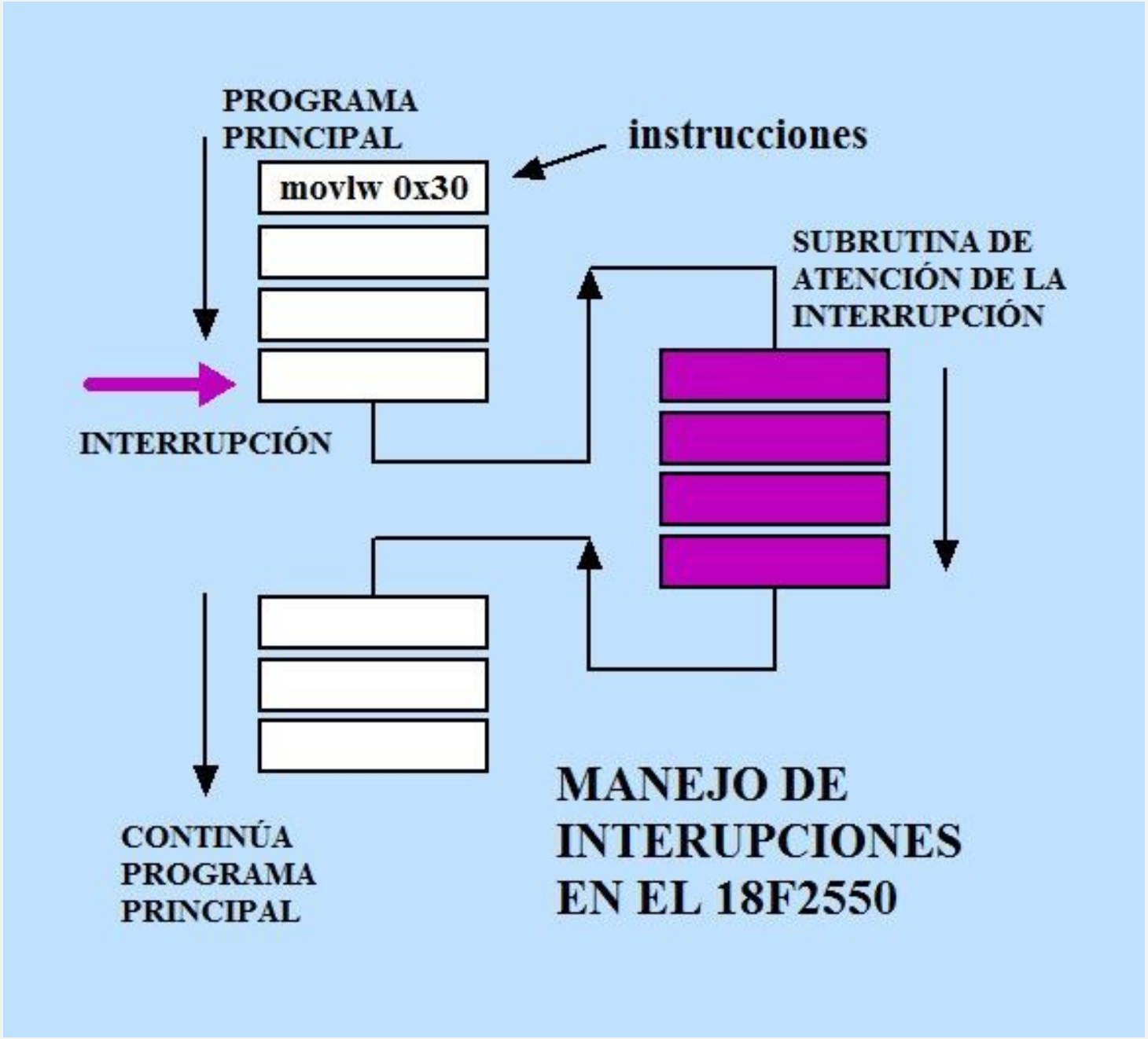
INTERRUPCIÓN

¿QUÉ ES?

- En el contexto de la informática se le llama interrupción a una señal que es recibida por el procesador la cual le indica que debe de **“interrumpir”** lo que esta realizando para atender otro asunto y ejecutar otra porción de código especifico para la situación y al terminar seguir con sus tareas

TIPOS

- **Interrupciones por hardware:** Estas son asíncronas a la ejecución del procesador, es decir, se pueden producir en cualquier momento independientemente de lo que esté haciendo el CPU en ese momento. Estas son externas y suelen venir de dispositivos de entrada o salida.
- **Excepciones:** Son aquellas que se producen de forma síncrona a la ejecución del procesador y por tanto podrían predecirse si se analiza con detenimiento la traza del programa que en ese momento estaba siendo ejecutado en la CPU. Normalmente son causadas al realizarse operaciones no permitidas tales como la división entre 0, el desbordamiento, el acceso a una posición de memoria no permitida, etc.
- **Interrupciones por software:** Las interrupciones por software son aquellas generadas por un programa en ejecución.


PROGRAMA
PRINCIPAL
instrucciones
movlw 0x30
SUBRUTINA DE
ATENCIÓN DE LA
INTERRUPCIÓN
INTERRUPCIÓN
CONTINÚA
PROGRAMA
PRINCIPAL
MANEJO DE
INTERUPCIONES
EN EL 18F2550

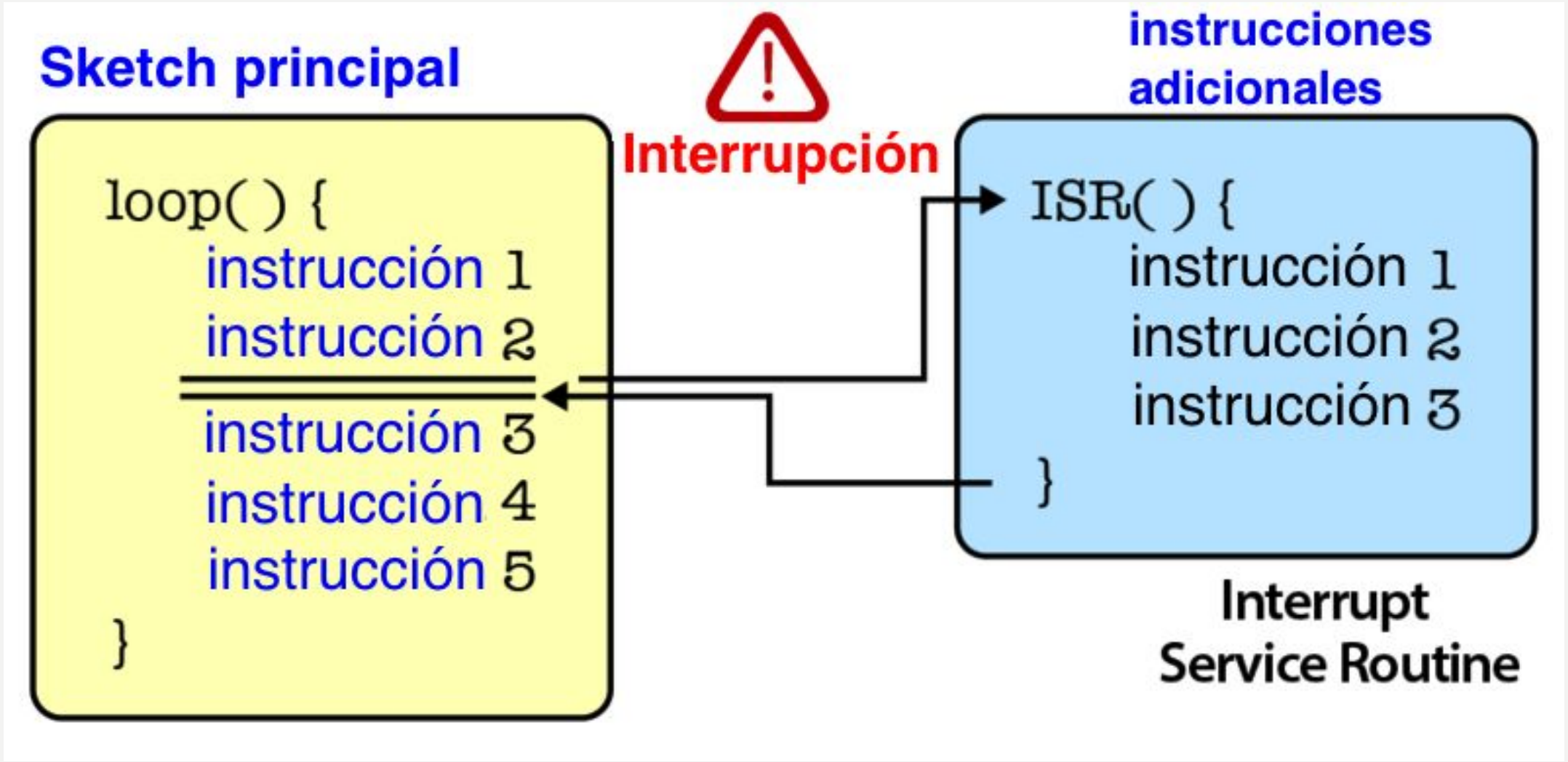
SISTEMAS DE PRIORIDAD

- **Interrupciones simultáneas:** No tienen por qué ocurrir de manera simultánea sino que se refiere a que en un momento dado pueden haber varias interrupciones activas.
- **Interrupciones anidadas:** Mientras se está procesando una determinada rutina de servicio de interrupción sucede otra señal de interrupción.
- **Inhibición de interrupciones:** Se deshabilitan las demás interrupciones mientras se está tratando una.

INTERRUPCIONES CON ARDUINO

INTERRUPCIONES EN ARDUINO

- Arduino posee dos eventos a los en los que definir interrupciones:
  - **Timers**
  - **Interrupciones por hardware**
- Dentro de las interrupciones de hardware Arduino es capaz de detectar los siguientes eventos.
  - RISING, ocurre en el flanco de subida de LOW a HIGH.
  - FALLING, ocurre en el flanco de bajada de HIGH a LOW.
  - CHANGING, ocurre cuando el pin cambia de estado (rising + falling).
  - LOW, se ejecuta continuamente mientras está en estado LOW.


Sketch principal
loop( ) {
instrucción 1
instrucción 2
instrucción 3
instrucción 4
instrucción 5
}
Interrupción
instrucciones adicionales
ISR( ) {
instrucción 1
instrucción 2
instrucción 3
}
Interrupt Service Routine

- Las interrupciones son muy útiles para hacer que las cosas ocurran automáticamente en los programas del microcontrolador y pueden resolver problemas de temporización.
- Para definir una interrupción necesitamos tres cosas:
  - Un pin de Arduino que recibirá la señal de disparo
  - Una condición de disparo
  - Una función que se ejecutará, cuando se dispara la interrupción (Llamada call back function).

INTERRUPCIONES EXTERNAS E INTERNAS

INTERRUPCIONES EXTERNAS

- Estas interrupciones hardware, se diseñaron por la necesidad de reaccionar a suficiente velocidad en tiempos inimaginablemente cortos a los que la electrónica trabaja habitualmente y a los que ni siquiera el software era capaz de reaccionar.

Board	Digital Pins Usable For Interrupts
Uno, Nano, Mini, other 328-based	2, 3
Mega, Mega2560, MegaADK	2, 3, 18, 19, 20, 21
Micro, Leonardo, other 32u4-based	0, 1, 2, 3, 7
Zero	all digital pins, except 4
MKR1000 Rev.1	0, 1, 4, 5, 6, 7, 8, 9, A1, A2
Due	all digital pins

INTERRUPCIONES INTERNAS

- Las interrupciones internas en Arduino son aquellas interrupciones relacionadas con los timers y que también son denominadas interrupciones de eventos programados.

ISR (INTERRUPT SERVICE ROUTINES)

- La función de callback asociada a una interrupción se denomina ISR (Interruption Service Rutine). ISR es una función especial que tiene algunas limitaciones
  - Una ISR no puede tener ningún parámetro en la llamada y no pueden devolver ninguna función.
  - Dos ISR no pueden ejecutarse de forma simultánea. En caso de dispararse otra interrupción mientras se ejecuta una ISR, la función ISR se ejecuta una a continuación de otra.
  - Una ISR debe ser tan corta y rápida como sea posible, puesto que durante su ejecución se paraliza el curso normal del programa y las interrupciones se deshabilitan.
  - La función millis() no funciona dentro del ISR puesto que usa interrupciones para su uso.
  - La función micros() funciona dentro de ISR pero después de 1-2 ms se empieza a comportar de forma extraña.

SECUENCIA CUANDO SE DISPARA UNA INTERRUPCIÓN

1. El microcontrolador completa la instrucción que está siendo ejecutada.
2. El programa de Arduino que se está ejecutando, transfiere el control a la Interrupt Service Routine (ISR). Cada interrupción tiene asociada una ISR que es una función que le dice al microcontrolador que hacer cuando ocurre una interrupción.
3. Se ejecuta la ISR mediante la carga de la dirección de comienzo de la ISR en el contador del programa.
4. La ejecución del ISR continua hasta que se encuentra el RETI (return from the interrupt instruction). Más información: http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_RETI.html
5. Cuando ha finalizado ISR, el microcontrolador continua la ejecución del programa donde lo dejó antes de que ocurriera la interrupción.

FUNCIONES DE INTERRUPCIONES EN ARDUINO

- **interrupts():** Habilita las interrupciones (antes han debido ser inhabilitadas con noInterrupts()).
- **noInterrupts():** Deshabilita las interrupciones. Las interrupciones pueden ser habilitadas de nuevo con interrupts().
- **attachInterrupt():** Me permite configurar una interrupción externa, pero no otro tipo de interrupciones. El primer parámetro es el número de interrupción que va asociado a un pin, luego la función ISR y finalmente el modo.
- **detachInterrupt():** Deshabilita la interrupción. El parámetro que se le pasa es el número de la interrupción.

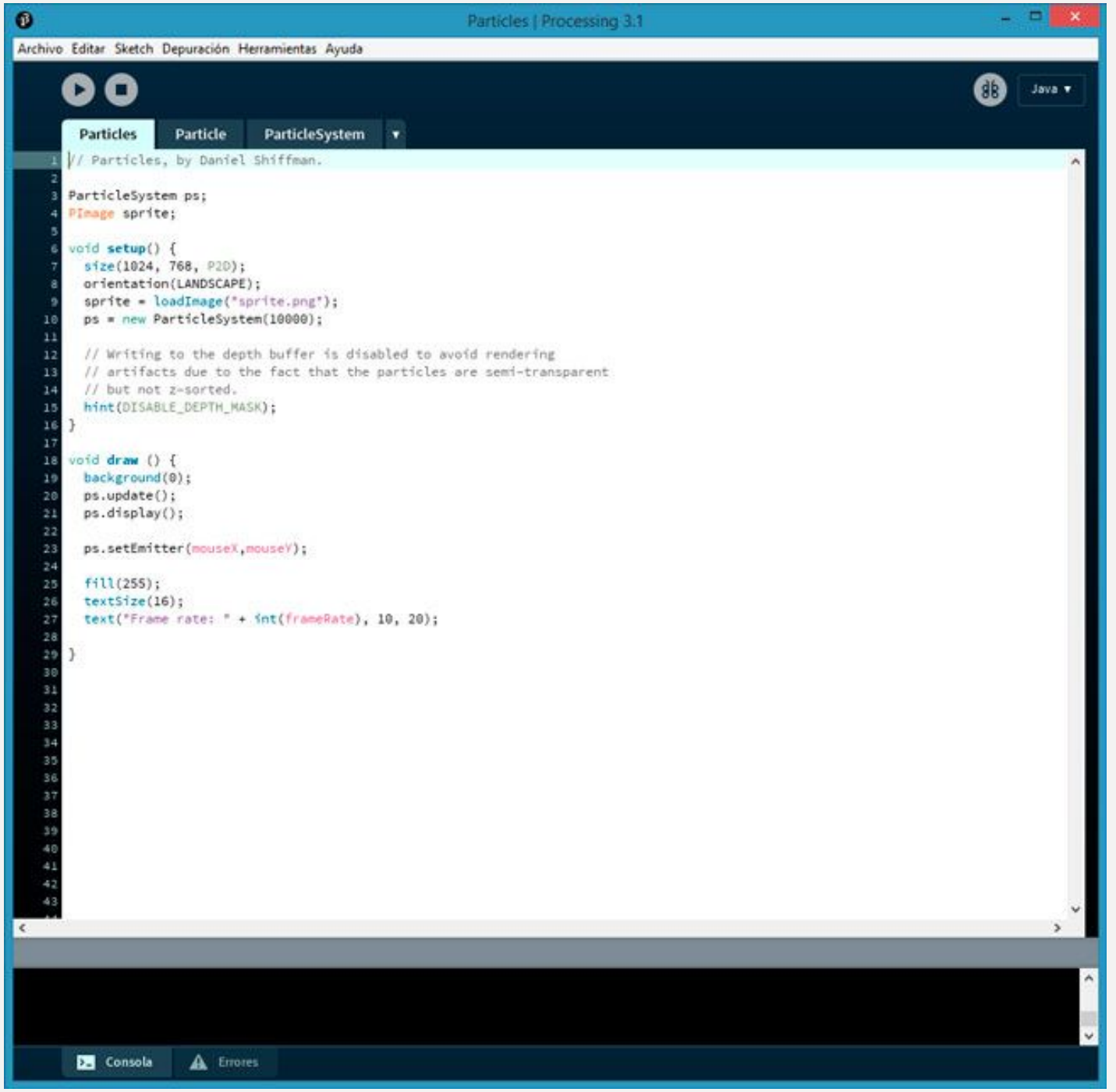
PROCESSING

¿QUÉ ES?

- Processing es un lenguaje de programación y entorno de desarrollo integrado de código abierto basado en Java, de fácil utilización, y que sirve como medio para la enseñanza y producción de proyectos multimedia e interactivos de diseño digital.

ENTORNO DE DESARROLLO

- Tiene su propio entorno de desarrollo. Cuando se ve se entiende porque se dice que la plataforma Arduino se nutre de Processing. Se llama PDE (Processing Development Enviroment) desarrollado en Java. Es muy sencillo y fácil de usar, ya que es una plataforma plug and play como Arduino.

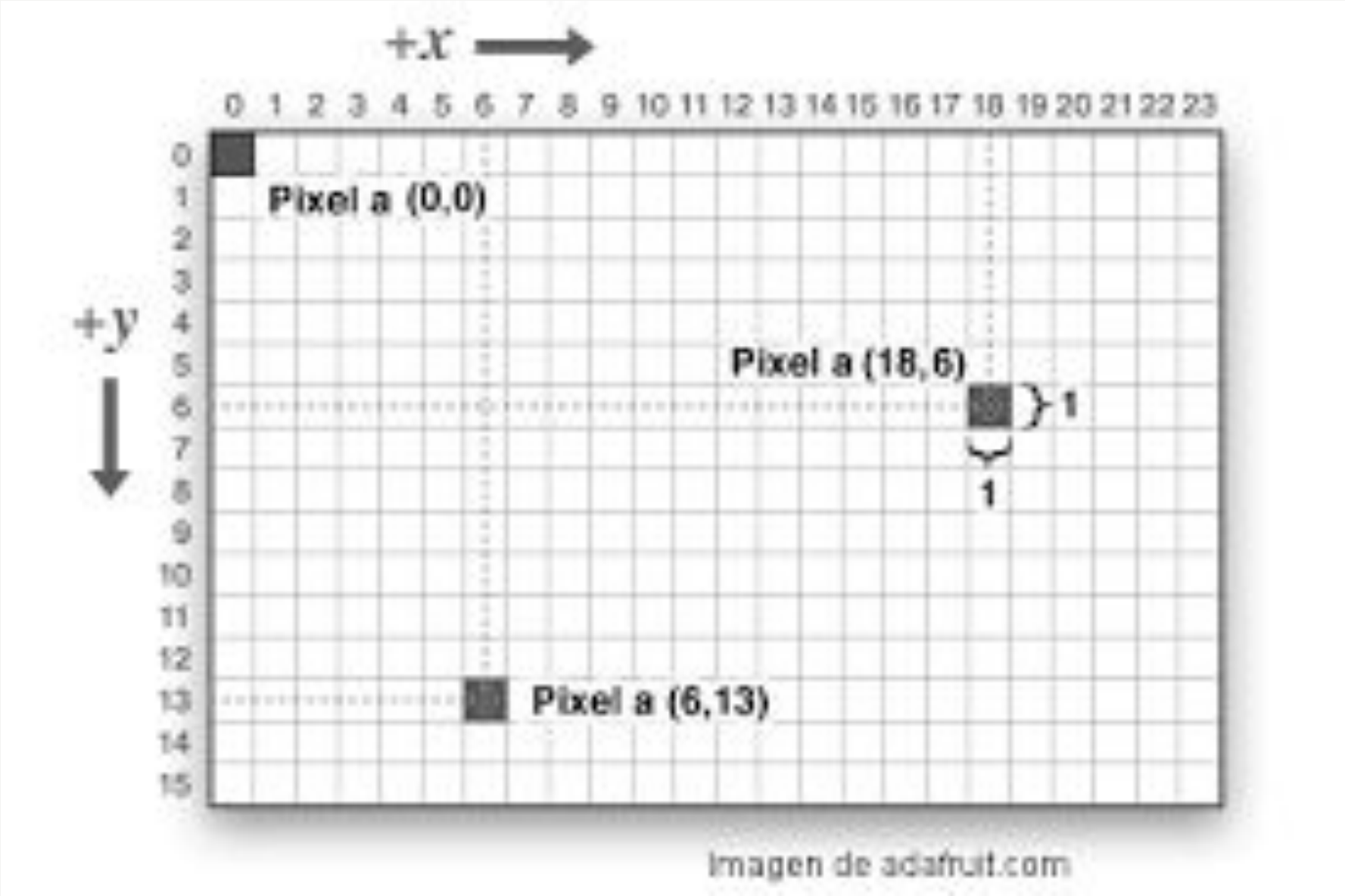

Particles | Processing 3.1
Archivo Editar Sketch Depuración Herramientas Ayuda
Java
Particles
Particle
ParticleSystem
// Particles, by Daniel Shiffman.
ParticleSystem ps;
PImage sprite;
void setup() {
size(1024, 768, P2D);
orientation(LANDSCAPE);
sprite = loadImage("sprite.png");
ps = new ParticleSystem(10000);
// Writing to the depth buffer is disabled to avoid rendering
// artifacts due to the fact that the particles are semi-transparent
// but not z-sorted.
hint(DISABLE_DEPTH_MASK);
}
void draw () {
background(0);
ps.update();
ps.display();
ps.setEmitter(mouseX,mouseY);
fill(255);
textSize(16);
text("Frame rate: " + int(frameRate), 10, 20);
}
Consola
Errores

LENGUAJE DE PROGRAMACIÓN PROCESSING

- Processing está basado en Java, más concreto en la versión 1.4.2. Esto no quiere decir que podamos utilizarlo con las versiones más actuales.
- Aunque Processing esté basado en Java, encontramos diferencias entre estos dos lenguajes de programación.
  - La sintaxis es más relajada, plataforma plug and play y sin preparativos.
  - En el modo básico programamos sin funciones, sin clases, sólo líneas de código con variables globales. En Java no podemos.
  - En el modo procedural al estilo de la programación en C, podemos definir nuestras propias funciones a las que llamamos.
  - En el modo más avanzado, programación orientada a objetos, podemos crear clases de una forma muy sencilla que luego son transformadas e completas clases Java.

FUNCIONAMIENTO

- Primero de todo hay que saber que la pantalla de visualización que creamos con Processing es un gráfico de píxeles, cada uno registrado con una coordenada (X, Y). El origen del gráfico está situado a la esquina izquierda arriba con las coordenadas (0, 0). Si sumamos a X, nos desplazamos a la derecha. Si sumamos a Y, nos desplazamos hacia abajo.


+x
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Pixel a (0,0)
+y
Pixel a (18,6)
1
1
Pixel a (6,13)
Imagen de adafruit.com

FUNCIONES

- **setup():** Función de arranque cuando el programa empieza. Igual que el de Arduino.
- **draw():** Bucle que se ejecuta cíclicamente siempre. Como el loop() de Arduino.
- **size(x, y):** Con esta función, Processing crea una ventana de X píxeles por Y píxeles. En esta ventana rearemos nuestro entorno gráfico.
- **background(0,0,0);** Determina el color del fondo de la ventana creada en modo RGB

- **stroke(0,0,0);** Función de trazo (bordes) que permite cambiar el color del dibujo actual en modo RGB. Cada función de dibujo llamada después de stroke tendrá el color de este.
- **noStroke();** Sin bordes.
- **point(x, y);** Crea un punto en la posición x, y del gráfico.
- **line(x1, y1, x2, y2);** Crea una línea. Los dos primeros valores son la primera coordenada X, Y. Los dos últimos valores son la última coordenada X, Y.
- **triangle(x1,y1,x2,y2,x3,y3);** Dibuja un polígono de tres puntas. Tiene seis valores que son las tres coordenadas de las tres puntas.
- **quad(x1,y1,x2,y2,x3,y3,x4,y4);** Dibuja un polígono de cuatro puntas. Los ocho valores son las cuatro coordenadas de las cuatro puntas.
- **rect(x, y, width, height);** Dibuja un rectángulo. Los dos primeros valores es la posición X, Y en el gráfico. Los dos últimos valores son la anchura y la altura respectivamente.

- **ellipse(x, y, width, height):** Dibuja una circulo. Los valores funcionan de la misa manera que rect.
- **fill(0,0,0):** Llena del color indicado la figura.
- **noFill():** Sin llenar de color.
- **text("texto",x,y):** Escribe un texto en la posición X,Y deseada.
- **mousePressed():** Función llamada cada vez que se aprieta el botón del ratón.
- **mouseReleased():** Función llamada cada vez que el botón del ratón no está apretado.
- **KeyPressed():** Función llamada cada vez que se aprieta una tecla del teclado.
- **year():** Retorna el año actual del reloj del PC
- **month():** Retorna el mes actual del reloj del PC
- **day():** Retorna el día actual del reloj del PC
- **hour():** Retorna la hora actual del reloj del PC
- **minute():** Retorna el minuto actual del reloj del PC
- **second():** Retorna el segundo actual del reloj del PC