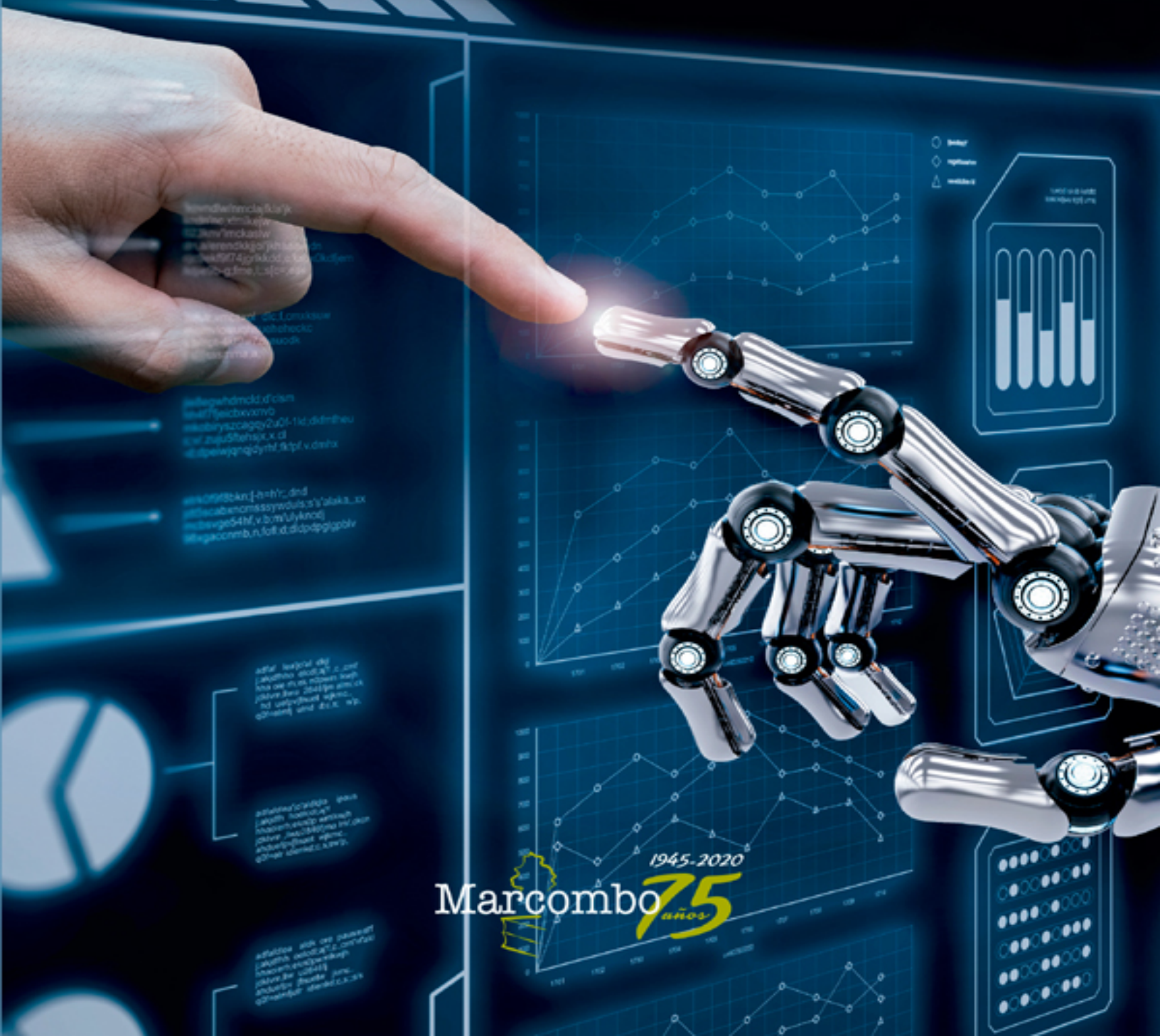
Processing

Desarrollo de interfaces de usuario, aplicaciones de visión artificial e IoT para Arduino y ESP8266

Tomás Domínguez Mínguez



Marcombo 1945-2020 75 años

PROCESSING

Desarrollo de interfaces de usuario, aplicaciones de visión artificial e IoT para Arduino y ESP8266

TOMÁS DOMÍNGUEZ MÍNGUEZ

PROCESSING

Desarrollo de interfaces de usuario, aplicaciones de visión artificial e IoT para Arduino y ESP8266

Marcombo 1945-2020 75 años

Processing
Desarrollo de interfaces de usuario, aplicaciones de visión artificial e IoT para Arduino y ESP8266

Primera edición, 2020

www.marcombo.com

Cubierta: ENEDENÚ DISEÑO GRÁFICO
Maquetación: Giancarlo Salinas
Correctores: Nuria Barroso y Genís Monraba
Directora de producción: Mª Rosa Castillo
Producción del ebook: booqlab

ISBN: 978-84-267-3202-6

A mi mujer, que es
el amor de mi vida

ÍNDICE

1

QUÉ ES PROCESSING

Processing se puede describir como un *lenguaje de programación*, pero también es un entorno de desarrollo *de código abierto* basado en *Java,* orientado al diseño gráfico. Fue desarrollado por Ben Fry y Casey Reas, ambos miembros del departamento Media Lab del MIT (*Massachusetts Institute of Technology*), a partir de 2001.

El objetivo de Processing es facilitar el aprendizaje de la programación en el contexto de las artes visuales, promoviendo la introducción del software dentro del diseño artístico. Por eso fue construido como una versión simplificada de Java dirigida a artistas y diseñadores, ofreciéndoles un medio sencillo que les permitiera esbozar ideas en código.

Aunque el origen de Processing está orientado a diseñadores gráficos, no hace falta ser uno de ellos para beneficiarse de las ventajas que supone su empleo. Todo lo contrario, asegura simplicidad y facilidad de uso. Y precisamente porque su orientación es gráfica, sacará un gran provecho de esa faceta visual durante el desarrollo de interfaces de usuario. Por otra parte, Arduino está especializado en la interacción con el mundo físico, siendo capaz de

recibir datos a través de sensores o actuar sobre su entorno mediante servos, motores, etc. Pero no tiene capacidades gráficas, por lo que su complemento natural para la interacción con el usuario es Processing. No en vano, Arduino nació bajo el paraguas de Processing, por lo que se le puede considerar una rama escindida de dicho proyecto, orientada al desarrollo HW. De ahí todos los aspectos en común que presentan ambos entornos de desarrollo.

Según sus creadores, Processing consiste en:

* Un entorno de desarrollo
* Una colección de funciones
* Una sintaxis de lenguaje

Un entorno de desarrollo que se usa para la implementación de código, orientado a diseñadores gráficos que no tengan necesariamente conocimientos de programación. Por ese motivo, se evita el empleo de herramientas Java cuya complejidad de instalación y configuración harían desistir de su utilización a este tipo de público.

Una colección de funciones básicas que conforman el núcleo de la interfaz de programación o API. También se podrán importar diversas librerías que permitan utilizar funciones más avanzadas como la lectura y procesamiento de imágenes y vídeo, la recepción y transmisión de datos en red, etc. Se irán introduciendo y explicando poco a poco a lo largo de libro.

Una sintaxis de lenguaje similar a java, sobre la que se han realizado algunos cambios que simplifican su uso, modificando la forma de desarrollar para orientarla hacia un modelo de programación basada en scripts, lo que permite escribir código de forma más rápida y fácil.

Al estar basado en este extendido lenguaje de programación, los desarrollos que realice con Processing heredarán muchas de sus ventajas, especialmente la de ser multiplataforma, por lo que los programas que codifique podrán ejecutarse en Mac OS, Windows o LINUX, independientemente de dónde se hayan desarrollado. Incluso

podrá generar código que pueda ejecutarse en navegadores Web o Android.

Para finalizar, es muy destacable que Processing sea código abierto. Eso quiere decir, entre otras cosas, que se distribuye como herramienta alternativa al software propietario, ya que tiene licencia *GNU GPL (General Public License)*, lo que garantiza que podrá descargarla y usarla de forma totalmente gratuita.

2

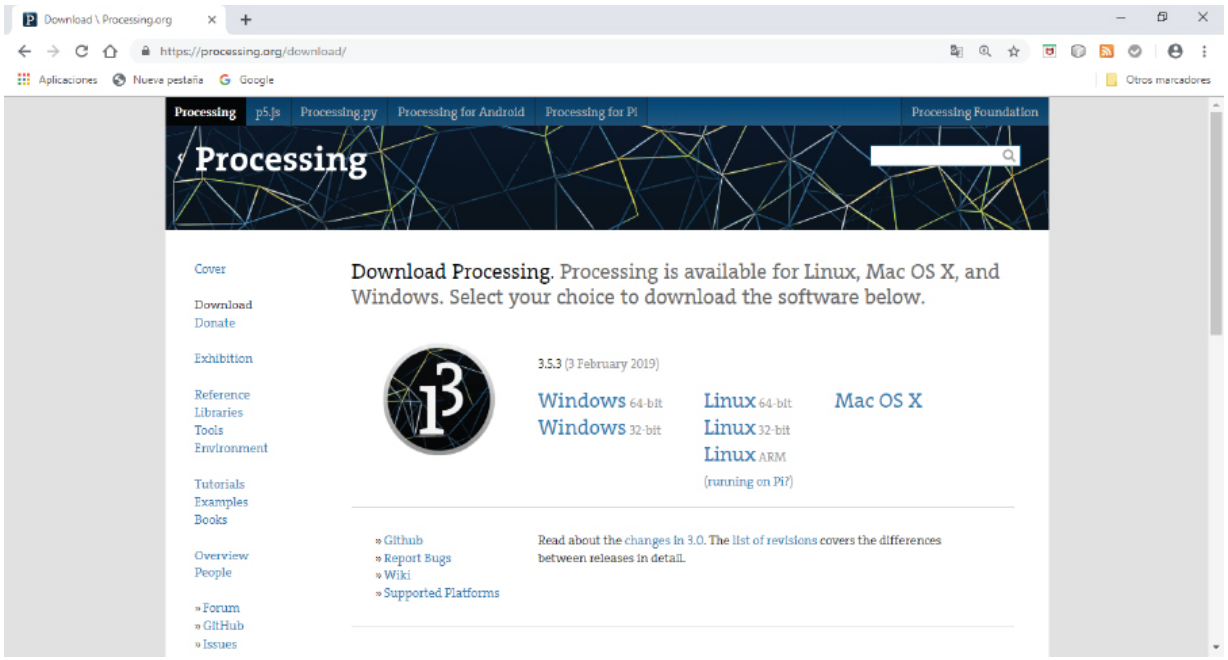
IDE PROCESSING

A los entornos de desarrollo se los conoce por el acrónimo inglés IDE (*Integrated Development Environment*). En Processing, a este IDE también se le llama PDE (*Processing Development Enviroment*), y como verá más adelante, su aspecto recuerda al de Arduino (no olvide que Arduino procede de Processing), lo que facilitará su aprendizaje.

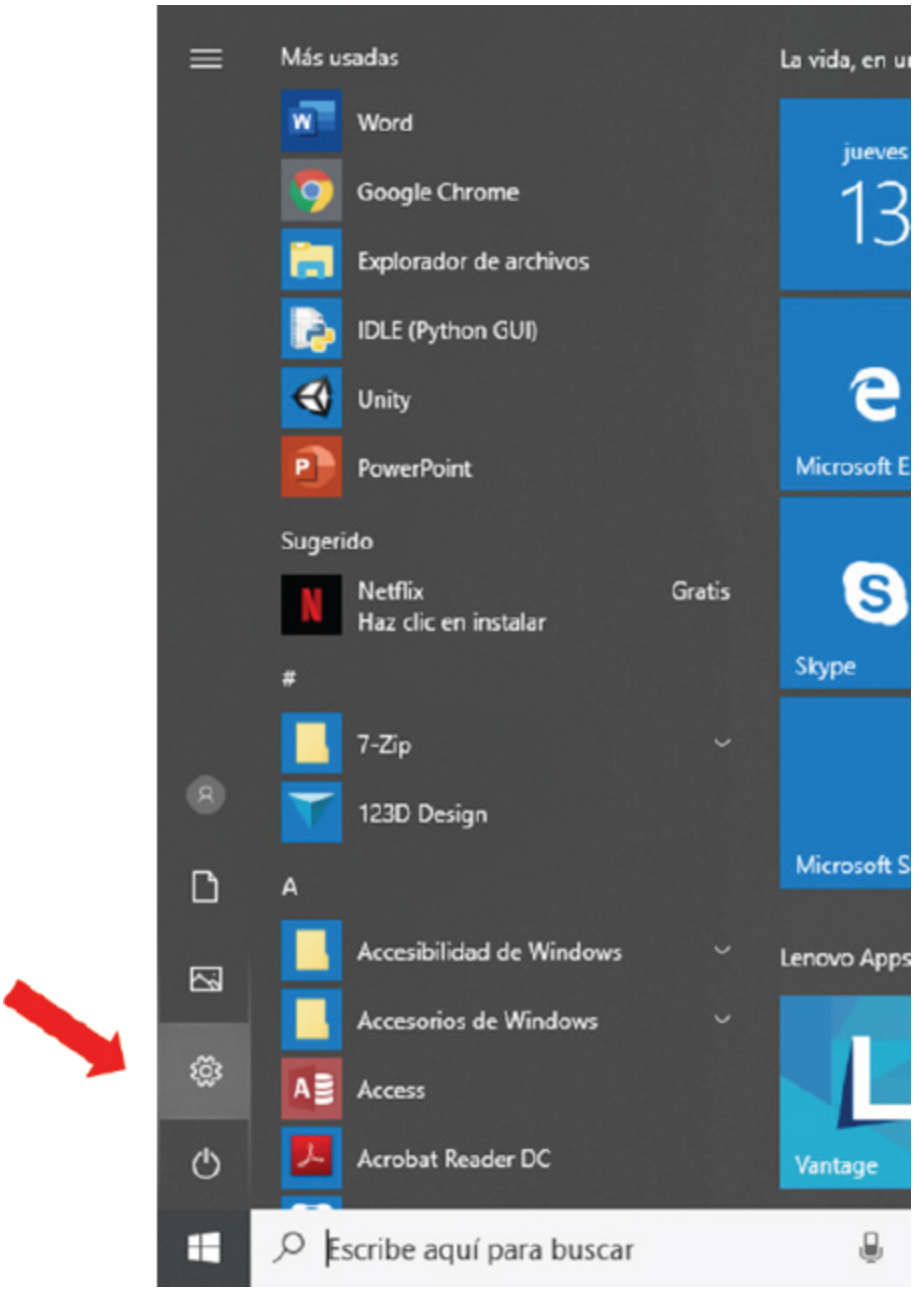
Nuestro contacto con Processing será siempre a través de su IDE. Por eso, es importante conocer las facilidades que ofrece para poder sacar el máximo provecho de todas ellas. Pero antes veamos cómo se instala.

2.1 INSTALACIÓN DEL IDE PROCESSING

La última versión de Processing se puede descargar de *http://processing.org/download*. Existe para entornos Mac OS, Windows o Linux. Seleccione el sistema operativo del que disponga (en este caso Windows) para descargar el fichero comprimido *.zip*.

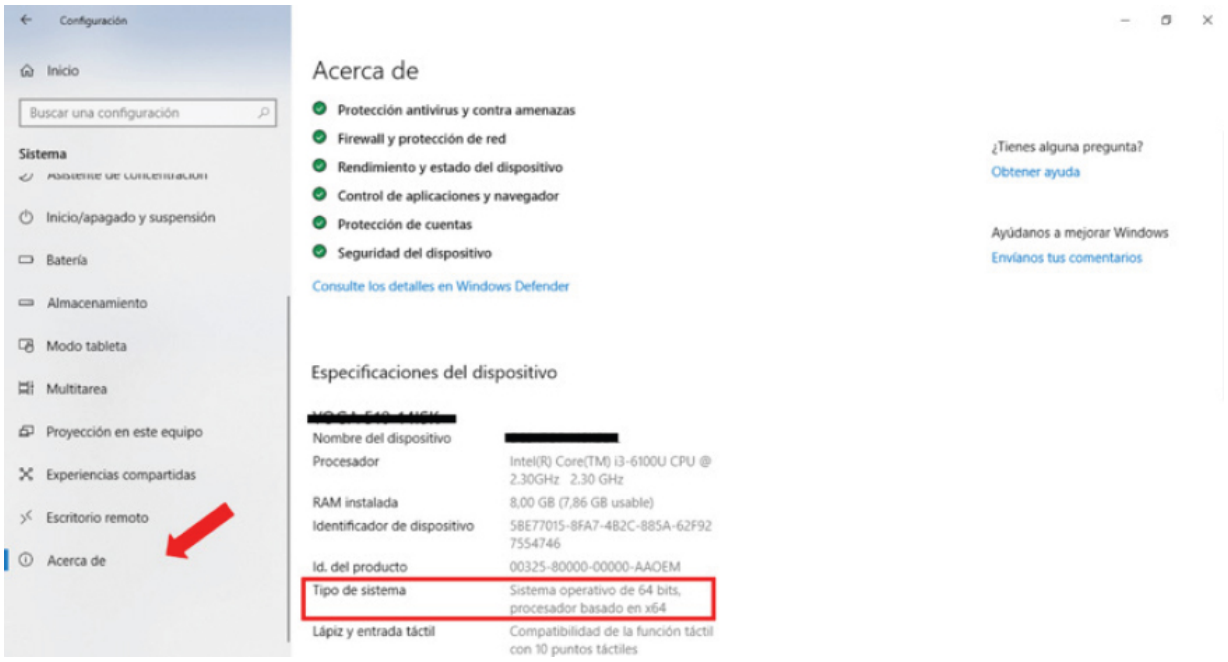


Para saber si su ordenador es Windows 32 bits o 64 bits, pulse en el botón «Inicio» y luego en «Configuración».



En la ventana que aparece pulse en la página «Sistema».

Se muestra una nueva pantalla en la que tendrá que seleccionar la pestaña «Acerca de» situada al final de todas las que aparecen en la parte izquierda. Hecho esto, en la zona derecha, en el apartado «Tipo de sistema», encontrará la información buscada.



Una vez descargado el archivo «.zip», deberá descomprimirlo en la carpeta que haya elegido, por ejemplo, en Documentos. Se encontrará con una nueva carpeta con el nombre «processing» seguido de un guion y su versión (en el momento de escribir este libro

es processing-3.5.3). No tiene que hacer nada más. En realidad, no hay ningún proceso de instalación ya que la aplicación no es más que un ejecutable situado dentro de una carpeta principal bajo la que hay una estructura de subcarpetas con todo lo necesario para su funcionamiento.

NOTA. En la ruta de instalación de la carpeta donde haya decidido instalar Processing no deberá haber acentos ni cualquier otro carácter especial.

2.2 MANEJO DEL IDE PROCESSING

Realizada la operativa anterior, estará impaciente por ver el aspecto que muestra este IDE. Para ello, solo tiene que entrar en la carpeta que acaba de descomprimir y hacer doble clic sobre el archivo «processing.exe». Tras ejecutarlo, esto es lo que verá.

El entorno de desarrollo de Processing se compone de las siguientes partes:

1. Barra de menús. Contiene los menús que permiten la gestión de todas las facilidades que contiene el IDE.
2. Barra de herramientas. Engloba una serie de botones comunes, entre los que destacamos el de ejecución (símbolo *play*) y parada (símbolo *stop*) de programas. En modo java, la ejecución de un programa compila el código y abre una nueva ventana de visualización.
3. Pestañas para administrar archivos. Cuando los programas crecen, pueden llegar a ser muy difíciles de mantener si todo el código está en un único archivo. Por eso, llegados a esa situación, los programas se suelen dividir en módulos que permitan una mejor administración de las diferentes partes que lo componen. En esos casos, un programa podría estar constituido de múltiples archivos que se manejarían con pestañas en esta sección. Habrá una por cada archivo que tenga abierto.
4. Editor de texto. Es la parte principal del IDE, donde escribirá los programas.
5. Área de mensajes. En ella aparecen comentarios al guardar y exportar archivos, pero, sobre todo, mostrará los errores que vaya cometiendo durante la edición del código.
6. Consola de texto. Muestra la salida de texto resultado de la ejecución de los programas. Incluye mensajes de error del sistema, además de los mensajes informativos que haya incluido en el código con las funciones `print()` y `println()`.

NOTA. Processing tiene diferentes modos de programación para que sea posible implementar *sketchs* en diferentes plataformas y lenguajes. El modo Java es el predeterminado, que es el que usted empleará. Dicho modo puede verse seleccionado en el

menú desplegable situado a la derecha, en la barra de herramientas.

Veamos a continuación los menús que contiene.

* Archivo. Reúne, entre otras, todas las opciones necesarias para crear, abrir o guardar un programa. También contiene la importante opción «Preferencias» en la que configurará el idioma del IDE.
* Editar. Agrupa todas las opciones básicas que tiene cualquier editor de texto.
* *Sketch*. Ademas de la ejecución y parada de programas, que se muestran también como botones en la barra de herramientas, sus opciones permiten realizar la importación de librerías, con las que se amplían las capacidades de Processing más allá de las que tiene integradas. Hay infinidad de librerías generadas por la comunidad de Processing que, como verá a lo largo de este libro, le posibilitará trabajar con imágenes 3D, grabación o reproducción de audio y vídeo, visión artificial o comunicaciones web y MQTT, entre otras muchas.
* Depuración. Permite activar y configurar el depurador, que es una importante herramienta de diagnóstico de errores. Si lo activa, se podrá controlar el avance de la ejecución del código línea a línea, o detenerlo en los puntos de interrupción que determine. Una vez parado el programa, mostrará los valores de las variables en un panel separado.
* Herramientas. Contiene utilidades de diferente naturaleza necesarias para, por ejemplo, crear fuentes, encontrar el código de un determinado color, etc.
* Ayuda. Como su nombre indica, contiene todo tipo de recursos que ayudarán tanto en el aprendizaje como en el uso del IDE. Tiene enlaces a páginas web que ofrecen una explicación detallada del entorno, así como la resolución de errores, preguntas frecuentes, tutoriales, etc.

Las opciones de dichos menús se irán estudiando a lo largo del libro según se vayan utilizando. Ahora únicamente verá las correspondientes al menú Archivo, ya que serán imprescindibles para empezar a trabajar con Processing.

* Nuevo. Crea un programa en una nueva ventana, a la que nombrará con la fecha actual usando el formato «sketch_YYMMDDa».
* Abrir. Abre el archivo que contiene un programa en una nueva ventana. Los archivos Processing tienen la extensión *.pde*. Dichos programas o *sketchs* deberán estar contenidos obligatoriamente dentro de un *sketch folder*, que es una carpeta con el mismo nombre del archivo del programa (sin la extensión).
* Reciente. Muestra la lista de archivos cerrados recientemente.
* *Sketchbook*. Este término, que podría traducirse como cuaderno de bocetos (*sketchs* o programas Processing), hace referencia a la carpeta donde se almacenan los *sketch folders*.
* Ejemplos. Abre una nueva ventana para mostrar la lista de programas de ejemplos de todas las librerías que tenga instaladas, sean nativas o importadas. Os animo a que ejecutéis algunos de dichos ejemplos para daros cuenta del enorme potencial de este lenguaje.
* Cerrar. Cierra el *sketch* que está visualizando. Si este es el último que está abierto, le preguntará si desea salir. Para evitarlo, utilice directamente la opción de Salir.
* Guardar. Guarda el archivo.
* Guardar como. Guarda el programa, dándole la opción de poner un nombre diferente al archivo que lo contenga. No reemplazará la versión anterior de dicho programa.
* Exportar aplicación. Exporta una aplicación Java como un archivo ejecutable.
* Configurar página. Define las opciones de impresión del código.
* Imprimir. Imprime el código del editor de texto.

* Preferencias. Configura aspectos de funcionamiento del entorno, como la ubicación predeterminada del *sketchbook*, el tamaño de la fuente utilizada en el texto o, muy importante, el idioma.
* Salir. Sale del IDE Processing cerrando todas las ventanas.

3

PROGRAMACIÓN CON PROCESSING

Aunque se hable de la programación de Arduino como si de un lenguaje específico se tratara, eso realmente no es así, ya que esta se hace en C++. Lo que sucede es que Arduino incorpora dentro de su IDE un conjunto de librerías que facilitan la programación de los pines de entrada y salida asociados a sus puertos analógicos y digitales. En Processing sucede algo similar, solo que ahora se programa en Java y el foco es el diseño gráfico, no el manejo de puertos. Además, tal como apunté anteriormente, Arduino no deja de ser un proyecto escindido de Processing, por lo que su filosofía de trabajo es similar.

Aunque Java y C++ son lenguajes diferentes, el uso de variables, estructuras de control, operadores o manejo de funciones tienen una sintaxis similar, por lo que podrá reutilizar los conocimientos de programación ya adquiridos con Arduino. Lo que sea específico de Processing se irá explicando a lo largo del libro según se vaya necesitando.

Finalmente, indicar que la referencia del lenguaje Processing se localiza en *https://processing.org/reference/*. Este será el sitio al que deberá acudir para conocer los detalles de cualquiera de las funciones que lo componen.

3.1 FUNCIONES SETUP() Y DRAW()

Un programa Processing se llama *sketch*, algo que podría traducirse como boceto o borrador. Los *sketchs* se almacenan en carpetas que se llaman *sketch folders* y, estos a su vez, en lo que se conoce como *sketchbook*, otra carpeta a nivel superior que se usa como ubicación predeterminada para guardar todos sus proyectos. Se puede acceder a los *sketchs* almacenados en el cuaderno de bocetos desde Archivo → *sketchbook*. Naturalmente, desde Archivo → Abrir... se permite abrir un *sketch* ubicado en cualquier otra carpeta del sistema.

Al igual que sucedía con Arduino, los *sketchs* o programas Processing se componen de dos grandes bloques de código: `setup()` y `draw()`. Ambos bloques, en realidad, son funciones que agrupan un conjunto de instrucciones capaces de realizar una tarea específica. En general, una función toma como entrada una serie de argumentos, que son los datos que necesita para realizar dicha tarea, y devuelve un resultado. En el caso que ocupa, `setup()` y `draw()` no toman argumentos de entrada ni tampoco devuelven ninguna salida. Por eso, se declaran de esta forma:

```
void setup(){
  …
}

void draw(){
  …
}
```

La palabra reservada *void* indica que estas funciones no devuelven ningún resultado tras su ejecución. Los paréntesis vacíos indican que no tienen argumentos de entrada, es decir, no necesitan que les pase ningún dato para poder realizar su tarea. Las instrucciones que componen cada función van enmarcadas entre llaves.

Los nombres de las funciones (igual que el de las variables) son *case-sensitive*, lo que significa que un carácter en mayúsculas es diferente del mismo en minúsculas. Por ejemplo, la función `leeSensor()` es diferente de `leesensor()`. Por supuesto, un nombre nunca puede llevar espacios.

NOTA. En Java existe una serie de convenciones de nomenclatura a la hora de escribir código que, aunque no son de obligado cumplimiento, se recomienda seguir para mejorar su legibilidad. En el caso de los nombres de las funciones (también el de las variables) suele usarse una nomenclatura llamada *«camel case»* que consiste en nombrar la función empezando por minúsculas y, en vez de utilizar caracteres como «-» o «_» para separar nombres compuestos, cada componente del nombre se empieza por mayúsculas. Por ejemplo, una función que obtiene los datos de un sensor podría llamarse `obtieneDatosSensor`. Es precisamente el perfil en forma de joroba de camello que muestra una palabra al mezclar mayúsculas y minúsculas lo que da nombre a esta nomenclatura.

Las instrucciones que componen la función `setup()` se ejecutan una única vez. Por ese motivo, esta función se utilizará para definir las condiciones iniciales del entorno como, por ejemplo, el tamaño de la pantalla. Solo puede haber una función de `setup ()` en cada programa y no se debe volver a llamar después de su ejecución inicial.

Por el contrario, las instrucciones de la función `draw()` se ejecutarán una y otra vez de forma repetida. Serán las encargadas de realizar el refresco continuo de la imagen que se muestra en pantalla, al igual que los fotogramas de una película. Cada uno de dichos fotogramas se pintará en un ciclo del bucle, en el que también se atenderán los posibles eventos de teclado, ratón, comunicaciones, etc., que permitirán la interactividad del programa.

Como habrá comprobado, ambos bloques son similares a los de `setup()` y `loop()` de Arduino.

NOTA. En este libro, con el ánimo de mostrar de la forma más clara y sencilla el funcionamiento de las funciones de Processing, se van a utilizar ejemplos que, por su simplicidad, en algunos casos permitirían que todo el código pudiera situarse dentro del bloque `setup()`, dentro del bloque `draw()` o, incluso, escribirlo sin utilizar ninguno de dichos bloques. Sin embargo, siempre se usarán ambos bloques ya que, de lo contrario, no se estaría siguiendo la filosofía de programación subyacente bajo Processing.

NOTA. Con frecuencia llamaré a las funciones Processing comandos. También me referiré a las líneas de código acabada en punto y coma («;») como sentencias. En el caso de las funciones `setup()` y `draw()`, al ser la base de la estructura de un programa Processing, las identificaré generalmente como bloques.

4

FUNCIONES GRÁFICAS

En Processing, el dibujo de interfaces de usuario se realiza a través de funciones específicas que le permitirán pintar, desde los elementos gráficos más elementales (como puntos y líneas), pasando por otros más elaborados (como figuras geométricas), hasta los más complejos realizados en 3D.

Naturalmente, podrá emplear el color allí dónde lo necesite, tanto para el fondo de la ventana de trabajo, como el trazo o el relleno de las figuras exhibidas.

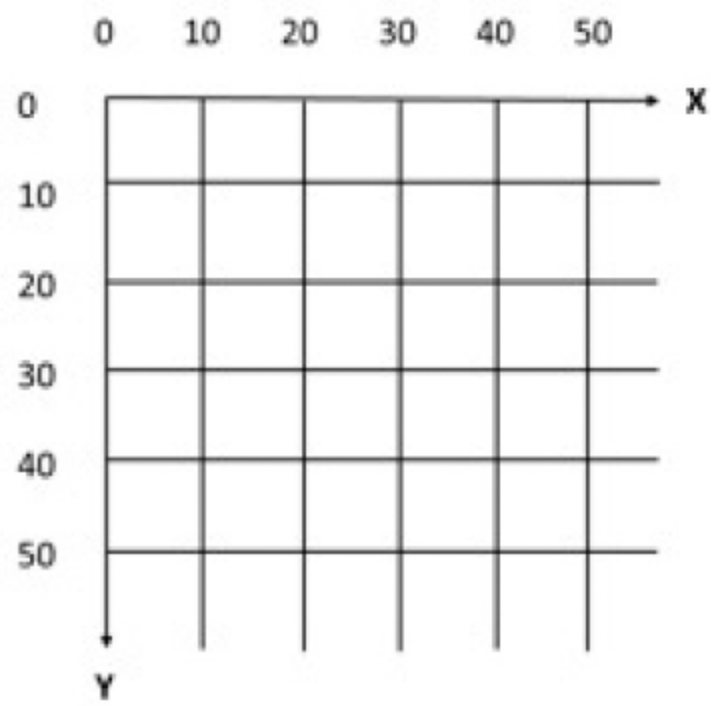
Finalmente, Processing le permitirá realizar operaciones de rotación, traslación o escalado que, junto con el uso de las matrices de transformación, posibilitarán el desarrollo de gráficos avanzados que incluyan animaciones y efectos visuales con los que conseguirá interfaces realmente impactantes.

4.1 GRÁFICOS ELEMENTALES

Los gráficos más elementales son los puntos y las líneas. Para dibujarlos, además de conocer las funciones gráficas que los muestran en pantalla, deberá saber cómo situarlos en una posición concreta. Para ello, será necesario conocer el sistema de coordenadas utilizado por Processing, concepto fundamental, previo a cualquier otro.

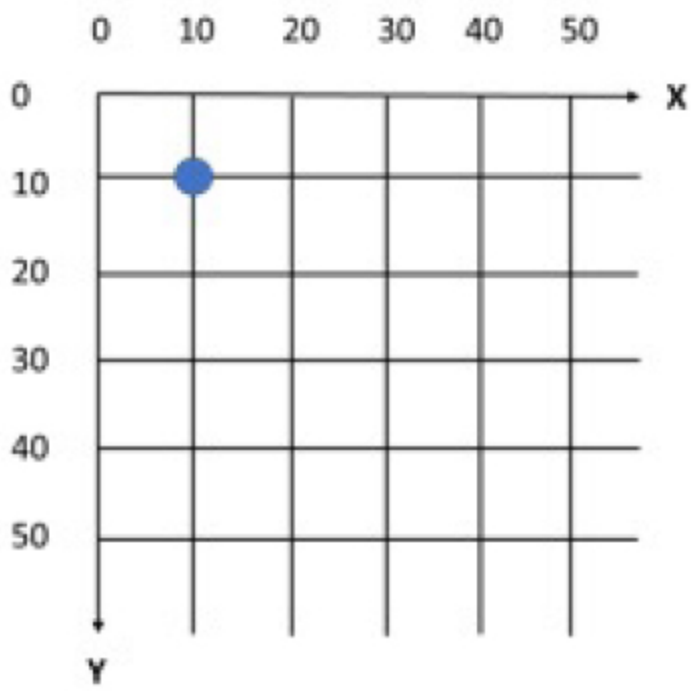
4.1.1 PUNTOS

El elemento gráfico más elemental es el punto. Pero antes de dibujarlo debe saber cómo indicar a Processing dónde situarlo. Para ello, tiene que ver la ventana de su programa como una hoja de papel cuadriculado. En dicho papel, la intersección de cada línea horizontal y vertical especificaría las coordenadas de cada punto, tal como se muestra en la siguiente figura.



Por lo tanto, las coordenadas de un punto forman una tupla de dos números x (horizontal) e y (vertical) que determina su ubicación en la ventana. Es de destacar que el origen de coordenadas (0, 0) no está situado en el centro de la imagen sino en la parte superior izquierda.

Así, por ejemplo, el punto con coordenadas (10, 10) sería:



Conociendo el sistema de coordenadas utilizado, para indicar a Processing que dibuje un punto utilizaría el siguiente comando:

```
point(x, y)
```

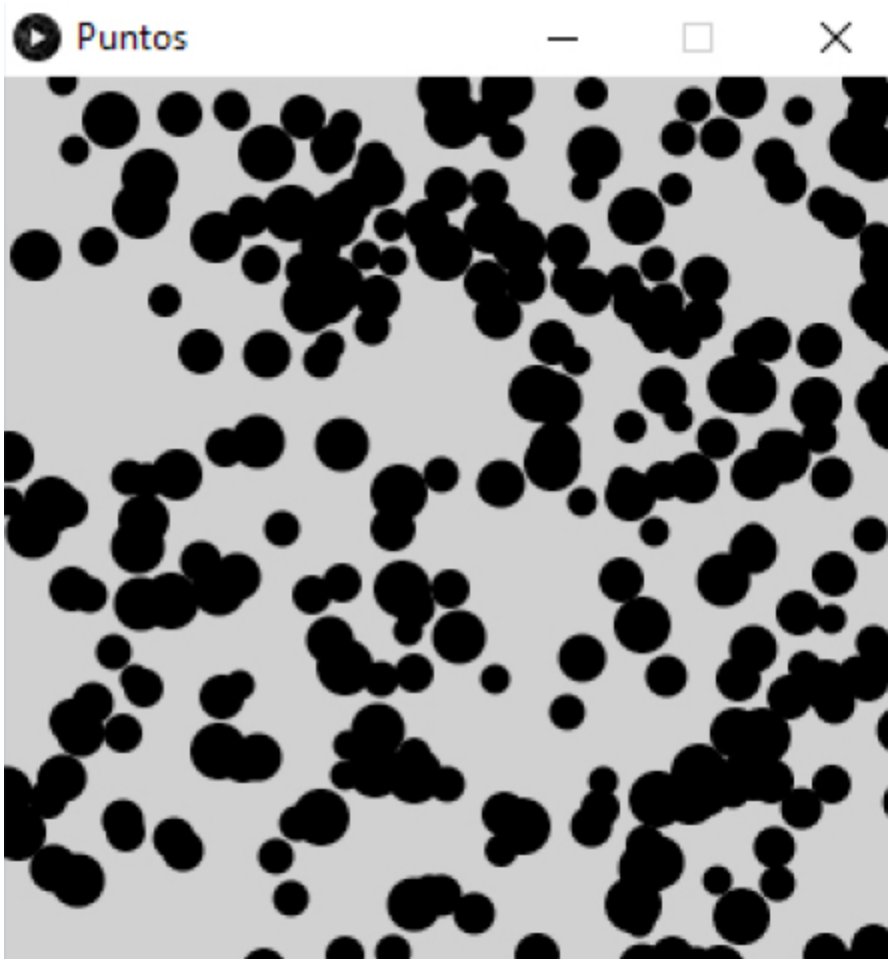
Es importante destacar que este comando hace referencia a un punto, no a un píxel. Un punto es un concepto cuya mínima expresión correspondería a un píxel. Como los puntos pueden ser más o menos grandes, Processing proporciona otro comando para especificar su grosor:

```
strokeWeight(grosor)
```

Donde el grosor viene dado en píxeles.

NOTA. Este comando también es válido para indicar el grosor de las líneas y los contornos de cualquier figura geométrica.

A continuación, se muestra un ejercicio en el que experimentará cómo se va rellenando de negro una ventana utilizando puntos en una posición y grosor aleatorio.



El código que se utilizará será el siguiente: