PROGRAMACIÓN DE CÓDIGOS QR (O DE BARRAS) CON

PYTHON, JAVA, JAVASCRIPT, C++

RUBÉN HIDALGO TROYANO
MARÍA MEGÍAS MOYANO

ÍNDICE

1. Introducción

¿Qué son los códigos QR y los códigos de barras?

Los códigos de barras son representaciones gráficas de datos legibles por máquinas que consisten en líneas paralelas de diferente grosor y espaciado. Se utilizan comúnmente para identificar productos en tiendas, facilitar inventarios y agilizar procesos logísticos. Los códigos QR (Quick Response) son una evolución de los códigos de barras que permiten almacenar más información en un espacio reducido, utilizando un patrón bidimensional de cuadros blancos y negros. Estos pueden contener enlaces, texto, datos de contacto, coordenadas, entre otros.

Usos comunes en la actualidad

Los códigos de barras son una tecnología ampliamente utilizada para la identificación y seguimiento de productos. Se encuentran en casi todos los productos de consumo, permitiendo un control eficiente del inventario. En el ámbito comercial, los códigos de barras se emplean en el sistema de facturación de los puntos de venta (POS), facilitando el proceso de compra y acelerando las transacciones. Además, desempeñan un papel crucial en la gestión de la logística y el transporte, ayudando a rastrear los productos a lo largo de su cadena de distribución. Por otro lado, los códigos QR han ganado una gran popularidad en aplicaciones digitales. Permiten redirigir a los usuarios a enlaces directos a sitios web o redes sociales, facilitando el acceso a información en línea. También son comúnmente utilizados para la descarga de aplicaciones móviles, proporcionando una forma rápida de acceder a contenido adicional en dispositivos móviles. Además, los códigos QR se emplean en procesos de autenticación en diversas plataformas, aumentando la seguridad en transacciones o accesos. En el ámbito turístico, se utilizan para proporcionar información adicional y señalética accesible, como es el caso de los códigos NaviLens, diseñados para personas con discapacidad visual. Finalmente, se utilizan frecuentemente en boletos electrónicos y pagos digitales, transformando la manera en que gestionamos entradas a eventos o realizamos compras en línea.

2. Estructura y funcionamiento de los códigos

Códigos de Barras:

Los códigos de barras son representaciones gráficas de datos cuya información se encuentra codificada por líneas paralelas de distinto grosor y espacio. Se les conoce como 1D debido a que solamente se pueden leer de izquierda a derecha, conociendo la dirección de lectura gracias a que todos los códigos de barras comienzan por una doble barra simple (además de que muchas veces contienen un código numérico en la zona inferior del código, ayudando esto también a conocer cuál es la dirección correcta).

Cada línea representa un número o carácter, pudiendo el lector láser interpretar las variaciones de luz y oscuridad para convertirlas en información digital.

Un uso cotidiano del código de barras se puede observar en cualquier supermercado ya que todos los productos (los más conocidos son los alimentos) deben estar catalogados según la AECOC (representante oficial de GS1 en España), usando para ello códigos GTIN 13 o EAN-13 aunque si los alimentos vienen en empaques pequeños pueden venir catalogados por un GTIN 8 o EAN-8.

De hecho, prácticamente cualquier producto comercial contiene un GTIN, incluídos los libros los cuales contienen un GTIN 13 específico llamado ISBN cuya característica distintiva es comenzar por los prefijos 978 o 979.

Los códigos de barras tienen varios inconvenientes, entre ellos, una capacidad de almacenamiento de datos muy limitada (máximo de 20 o 25 caracteres), requiere una orientación de lectura específica (además de tener que leer de izquierda a derecha) y son muy sensibles a daños como los rayores o dobleces (de ahí que también vengan acompañados con un código numérico para usar en situaciones en los que alguna barra quede ilegible).

Códigos QR:

Los códigos QR (Quick Response) surgieron tras los códigos de barras, almacenando información de forma bidimensional. Esto se debe a que, en lugar de líneas, usan módulos cuadrados en una matriz. El código QR se lee tanto en horizontal como en vertical, pudiendo almacenar mucha más información que un código de barras en menos espacio. Además, contienen algoritmos de corrección de errores (como Reed-Solomon) para que el código pueda seguir funcionando incluso si está parcialmente dañado.

Los códigos QR pueden encontrarse en varias situaciones cotidianas. Se pueden encontrar actuando como enlace a una web, para leer menús de restaurantes, tarjetas de visita o, incluso, para realizar pagos móviles.

Además, tienen varias ventajas sobre los códigos de barras como una mayor capacidad de almacenamiento de datos (hasta unos 7000 caracteres), se puede leer

desde cualquier ángulo, se puede leer con cámaras sin necesidad de usar láser y pueden almacenar texto, enlaces, geolocalización...

Ventajas y Desventajas

Característica	Código de Barras (1D)	Código QR (2D)
Capacidad de datos	Baja	Alta
Tamaño requerido	Más largo	Más compacto
Orientación de lectura	Horizontal	Cualquier ángulo
Corrección de errores	No	Sí
Uso común	Productos físicos	Digital, pagos, menús, etc
Equipamiento necesario	Lector láser	Cámara o lector QR

3. Programación de Códigos QR y de Barras

3.1. PYTHON

Python es uno de los lenguajes más versátiles que existen, contando con varias librerías que permiten generar y leer códigos QR y códigos de barras de forma sencilla.

Algunas de estas librerías son qrcode, la cual genera códigos QR de una forma rápida y sencilla, pudiendo personalizar características del QR generado como el color, tamaño, especificar el nivel de corrección de errores...

Otra librería imprescindible para generar, en este caso, códigos de barras usando Python es python-barcode, que permite generar códigos de barras en diferentes formatos como EAN, UPC, Code128...

También instalaremos la librería pyzbar para poder leer los códigos QR o los códigos de barras desde imágenes. Para poder leer la imagen y decodificar el código, debe trabajar en conjunto con la librería OpenCV. OpenCV es una librería de Python que sirve para leer imágenes y vídeos, siendo su uso principal mostrar imágenes por pantalla y procesarlas.

Además, al instalar qrcode[pil], también estamos instalando junto a qrcode las dependencias de la librería Pillow, la cual sirve principalmente para mostrar imágenes o guardar los códigos generados.

```
import sys
!{sys.executable} -m pip install qrcode[pil]
!{sys.executable} -m pip install opencv-python
!{sys.executable} -m pip install pyzbar
!{sys.executable} -m pip install python-barcode
```

GENERAR Y LEER UN CÓDIGO QR

Para empezar, para generar el código QR usaremos la librería Pillow y qrcode, además de matplotlib (para mostrar la imagen por pantalla en Jupyter Notebook).

Generamos una variable llamada data, la cual contiene el texto que nuestro QR contendrá internamente.

A continuación, generamos el código QR, pudiendo modificar sus características. En este caso, hemos creado uno muy básico (versión 1: el más pequeño, error_correction L: low, es más bajo, box_size 10, el tamaño de cada cuadrado en píxeles, y por último border 4, el espacio mínimo recomendado que debe haber alrededor del QR).

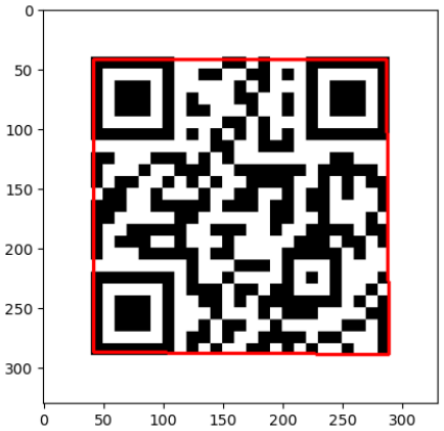
Insertamos el texto en el QR con add.data(data) y ajustamos el tamaño del QR al texto con un make.(fit=True). Este último es importante ya que, si usamos versión 1, quizá el texto no pueda contenerse en el QR, por lo que esto hace que el tamaño del QR sea del mínimo tamaño que pueda contener esa información sin tener que hacer distintas pruebas.

Especificamos que la imagen sea de fondo blanco (back_color) con cuadros negros (fill_color) y guardamos el código como una imagen llamada codigo_qr_generado.png, de forma que podamos usarla después para leerla.
Por último, se muestra la imagen por pantalla.

```python
import qrcode
from PIL import Image
import matplotlib.pyplot as plt


data = "https://example.com"


qr = qrcode.QRCode(
    version=1,
    error_correction=qrcode.constants.ERROR_CORRECT_L,
    box_size=10,
    border=4,
)


qr.add_data(data)
qr.make(fit=True)


img = qr.make_image(fill_color='black', back_color='white').convert('RGB')


img.save("codigo_qr_generado.png")


plt.imshow(img)
plt.axis('off')
plt.show()
```

Para empezar, se indica que se van a utilizar las librerías cv2 (OpenCV), pyzbar, numpy (para manejar matrices) y matplotlib.
A continuación, leemos el QR que anteriormente hemos guardado en una imagen y usamos la función decode() para identificar el código QR dentro de la imagen.
Continuamos, extraemos la información del QR (.data) y la pasamos a texto ya que se suele almacenar en binario (.decode()) para mostrarlo por pantalla.
Por último, dibujamos un cuadrado rojo alrededor del QR para saber que ha sido seleccionado correctamente y solamente ha tomado en cuenta el área del QR, mostrando la imagen por pantalla.

```python
import cv2
from pyzbar.pyzbar import decode
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread('codigo_qr_generado.png')


decoded_objects = decode(img)


for obj in decoded_objects:
    print("Datos del QR:", obj.data.decode('utf-8'))


for obj in decoded_objects:
    points = obj.polygon
    if len(points) == 4:
        pts = [tuple(point) for point in points]
        cv2.polylines(img, [np.array(pts, dtype=np.int32)], isClosed=True, color=(0, 0, 255), thickness=2)



plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.show()
```



GENERAR UN CÓDIGO DE BARRAS

Para empezar, generamos un código de barras usando las librerías barcode, matplotlib y cv2. Definimos un número, el cuál será el número del código de barras, en nuestro caso, un EAN-13 (el número debe ser de 12 cifras ya que la cifra de control se generará automáticamente).
Indicamos que va a ser un EAN-13 usando la función get_barcode_class.
Se crea el código de barras usando la clase ean() y la convierte a png gracias a ImageWriter(), guardándose como codigo_barras.png para su posterior lectura.
Por último, se lee la imagen al completo, sin analizarla, y se muestra por pantalla.

```python
import barcode
from barcode.writer import ImageWriter
import matplotlib.pyplot as plt
import cv2

numero = '123456789012'

ean = barcode.get_barcode_class('ean13')

barcode_img = ean(numero, writer=ImageWriter())
barcode_img.save('codigo_barras')

img = cv2.imread('codigo_barras.png')

plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

A continuación, leemos y analizamos el código de barras que hemos guardado. Usamos las librerías cv2, pyzbar y matplotlib.
Leemos la imagen y, al igual que con el código QR y, al igual que con el QR, almacenamos toda la información contenida en el código (en este caso, el código de barras). Usamos .data.decode como en el QR para tener la información del código de barras en texto, y se muestra por pantalla.


```python
import cv2
from pyzbar.pyzbar import decode
import matplotlib.pyplot as plt

img = cv2.imread('codigo_barras.png')

decoded_objects = decode(img)

for obj in decoded_objects:
    print("Datos del código de barras:", obj.data.decode('utf-8'))


plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.show()
```

Por último, Python ofrece una forma rápida de poder eliminar las imágenes almacenadas una vez utilizadas para ahorrar memoria:

```python
import os

qr_image_path = 'codigo_qr_generado.png'
barcode_image_path = 'codigo_barras.png'

if os.path.exists(qr_image_path):
    os.remove(qr_image_path)
    print(f"Archivo {qr_image_path} ha sido borrado.")
else:
    print(f"El archivo {qr_image_path} no existe.")

if os.path.exists(barcode_image_path):
    os.remove(barcode_image_path)
    print(f"Archivo {barcode_image_path} ha sido borrado.")
else:
    print(f"El archivo {barcode_image_path} no existe.")
```

```
✓ 0.0s
Archivo codigo_qr_generado.png ha sido borrado.
Archivo codigo_barras.png ha sido borrado.
```

3.2. JAVA

Java es un lenguaje de programación robusto y multiplataforma que, aunque no tiene tantas librerías específicas como Python para tareas visuales, permite generar y leer códigos QR y códigos de barras de forma eficaz gracias al uso de bibliotecas externas como ZXing (abreviatura de *Zebra Crossing*), una de las más utilizadas para este propósito.

La librería ZXing permite tanto la generación como la lectura de códigos QR y de barras. Es compatible con múltiples formatos como QR Code, EAN-13, Code 128, UPC-A, entre otros.

Para usarla en Java, es necesario descargar los archivos .jar correspondientes (por ejemplo, core-3.5.2.jar y javase-3.5.2.jar) y agregarlos manualmente al proyecto. En este caso, se ha utilizado Visual Studio Code como entorno de desarrollo, trabajando con archivos .java en la carpeta src, compilando en bin y cargando las librerías desde una carpeta lib.

GENERAR Y LEER UN CÓDIGO QR

Para la generación de un código QR, se utiliza QRCodeWriter de ZXing, que permite introducir un texto personalizado y convertirlo en una imagen PNG. Se define una cadena de texto que será el contenido del QR, se especifican dimensiones y luego se genera una imagen que se guarda como codigoQR.png.

```java
import com.google.zxing.BarcodeFormat;
import com.google.zxing.WriterException;
import com.google.zxing.qrcode.QRCodeWriter;
import com.google.zxing.common.BitMatrix;
import com.google.zxing.client.j2se.MatrixToImageWriter;

import java.io.IOException;
import java.nio.file.FileSystems;
import java.nio.file.Path;

public class GenerarQR {
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        String texto = "https://etsiit.ugr.es/";
        int ancho = 300;
        int alto = 300;
        String nombreArchivo = "codigoQR.png";

        QRCodeWriter qrWriter = new QRCodeWriter();
        try {
            BitMatrix matriz = qrWriter.encode(texto, BarcodeFormat.QR_CODE, ancho, alto);
            Path ruta = FileSystems.getDefault().getPath(nombreArchivo);
            MatrixToImageWriter.writeToPath(matriz, format:"PNG", ruta);
            System.out.println("Código QR generado correctamente.");
        } catch (WriterException | IOException e) {
            System.out.println("Error al generar el código QR:");
            e.printStackTrace();
        }
    }
}
```

El código Java crea un objeto BitMatrix con el contenido del QR, el cual luego se convierte a imagen mediante MatrixToImageWriter.writeToPath(). La imagen queda lista para ser leída posteriormente.

Para la lectura de un código QR, se utiliza BufferedImage junto con BufferedImageLuminanceSource para transformar la imagen en un formato que ZXing pueda interpretar. Luego, usando MultiFormatReader, se decodifica el contenido del QR.

```java
import com.google.zxing.*;
import com.google.zxing.client.j2se.BufferedImageLuminanceSource;
import com.google.zxing.client.j2se.MatrixToImageWriter;
import com.google.zxing.common.HybridBinarizer;

import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

public class LeerCodigoQR {

    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        String archivoImagen = "codigoQR.png";

        try {
            // Leer la imagen del QR
            BufferedImage imagen = ImageIO.read(new File(archivoImagen));

            // Preparar la imagen para decodificación
            LuminanceSource fuente = new BufferedImageLuminanceSource(imagen);
            BinaryBitmap bitmap = new BinaryBitmap(new HybridBinarizer(fuente));

            // Leer el QR con ZXing
            Result resultado = new MultiFormatReader().decode(bitmap);

            // Mostrar el contenido del QR
            System.out.println("Contenido del código QR: " + resultado.getText());

        } catch (IOException e) {
            System.out.println("Error al leer la imagen.");
            e.printStackTrace();
        } catch (NotFoundException e) {
            System.out.println("No se encontró ningún código QR en la imagen.");
        }
    }
}
```

Si se encuentra el código, se imprime por pantalla el texto contenido. En caso contrario, se notifica que no se encontró ningún QR. Este proceso permite validar la funcionalidad del generador y reutilizar los archivos generados.

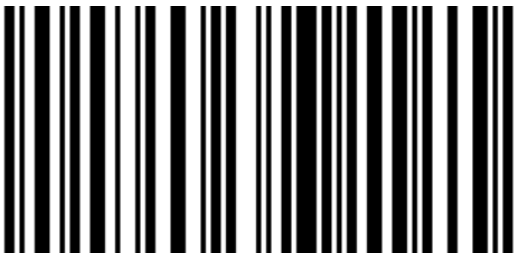
```
maria@maria-Lenovo-IdeaPad-S340-15IIL:~/Escritorio/PDIH/Prácticas/Proyecto$ javac -cp "lib/*" src/LeerCodigoQR.java -d bin
maria@maria-Lenovo-IdeaPad-S340-15IIL:~/Escritorio/PDIH/Prácticas/Proyecto$ java -cp "lib/*:bin" LeerCodigoQR
Contenido del código QR: https://etsiit.ugr.es/
```

GENERAR UN CÓDIGO DE BARRAS

La generación de códigos de barras también se realiza con ZXing, utilizando Code128Writer o EAN13Writer, dependiendo del tipo de código deseado. En este caso, se define un número (por ejemplo, uno de 12 cifras para EAN-13) como contenido del código de barras.

```java
import com.google.zxing.BarcodeFormat;
import com.google.zxing.WriterException;
import com.google.zxing.common.BitMatrix;
import com.google.zxing.client.j2se.MatrixToImageWriter;
import com.google.zxing.oned.Code128Writer;

import java.io.IOException;
import java.nio.file.FileSystems;
import java.nio.file.Path;

public class GenerarCodigoBarras {

    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        String texto = "123456789012";
        int ancho = 400;  // Ancho de la imagen
        int alto = 150;   // Alto de la imagen
        String nombreArchivo = "codigoBarras.png";

        Code128Writer barcodeWriter = new Code128Writer();
        try {
            BitMatrix matriz = barcodeWriter.encode(texto, BarcodeFormat.CODE_128, ancho, alto);
            Path ruta = FileSystems.getDefault().getPath(nombreArchivo);
            MatrixToImageWriter.writeToPath(matriz, format:"PNG", ruta);
            System.out.println("Código de barras generado correctamente.");
        } catch (IOException e) {
            System.out.println("Error al generar el código de barras:");
            e.printStackTrace();
        }
    }
}
```

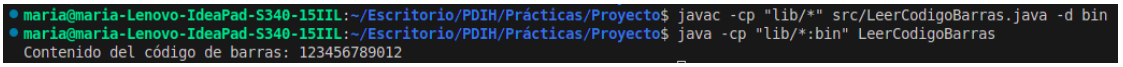
Se genera un objeto BitMatrix que representa visualmente el código, y se convierte a imagen PNG igual que en el caso del QR, mediante MatrixToImageWriter. El archivo se guarda como codigoBarras.png.



La lectura de un código de barras se realiza igual que con el código QR. Se carga la imagen con ImageIO.read(), se prepara para análisis con HybridBinarizer, y luego se utiliza MultiFormatReader().decode() para obtener la información contenida en el código.

```java
import com.google.zxing.*;
import com.google.zxing.client.j2se.BufferedImageLuminanceSource;
import com.google.zxing.common.HybridBinarizer;

import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

public class LeerCodigoBarras {

    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        String archivoImagen = "codigoBarras.png";

        try {
            // Cargar la imagen del código de barras
            BufferedImage imagen = ImageIO.read(new File(archivoImagen));

            // Preparar la imagen para ser procesada
            LuminanceSource fuente = new BufferedImageLuminanceSource(imagen);
            BinaryBitmap bitmap = new BinaryBitmap(new HybridBinarizer(fuente));

            // Intentar decodificar el código de barras
            Result resultado = new MultiFormatReader().decode(bitmap);

            // Mostrar resultado
            System.out.println("Contenido del código de barras: " + resultado.getText());

        } catch (IOException e) {
            System.out.println("Error al leer la imagen.");
            e.printStackTrace();
        } catch (NotFoundException e) {
            System.out.println("No se encontró ningún código de barras en la imagen.");
        }
    }
}
```

Al igual que con el QR, se imprime el resultado si se encuentra un código válido. Este método es compatible con los principales formatos de código de barras como EAN, UPC y Code128, lo que permite validar fácilmente la información visual codificada.



3.3. JAVASCRIPT

JavaScript, especialmente en entornos web, permite generar y leer códigos QR y códigos de barras de forma muy práctica gracias a diversas librerías de código abierto. Estas funciones se pueden integrar fácilmente en cualquier página web sin necesidad de software adicional.

GENERAR Y LEER UN CÓDIGO QR

Para generar códigos QR, JavaScript ofrece librerías como qrcode.js o qr-code-styling, que permiten crear códigos QR con texto personalizado y mostrarlos directamente en una página web como imagen, <canvas> o <svg>. Además, se pueden personalizar aspectos como el tamaño, color y diseño del QR.

Para leer códigos QR, se utilizan librerías como html5-qrcode o jsQR. Estas herramientas permiten escanear un código QR desde la cámara del dispositivo (PC o móvil) o desde una imagen previamente subida. Una vez leído el código, se muestra automáticamente el texto o información que contiene.

Todo esto se puede hacer directamente desde el navegador, sin necesidad de backend o software instalado, lo que lo hace ideal para aplicaciones web interactivas y móviles.

GENERAR Y LEER UN CÓDIGO DE BARRAS

Para generar códigos de barras, se puede utilizar la librería JsBarcode, que permite crear distintos tipos de códigos (como EAN-13, Code128, UPC...) y mostrarlos visualmente en la web usando etiquetas <canvas> o <svg>. Solo es necesario introducir el número o texto que se quiere codificar, y la librería genera la imagen correspondiente.

Para leer códigos de barras, se puede utilizar QuaggaJS, una librería que permite escanear códigos de barras en tiempo real desde la cámara del dispositivo o desde imágenes. Una vez detectado el código, se devuelve el contenido para poder usarlo dentro de la aplicación.

VENTAJAS DE JAVASCRIPT

Una de las principales ventajas de usar JavaScript para generar y leer códigos QR y de barras es que todo funciona directamente en el navegador, sin necesidad de instalar programas adicionales. Además, es totalmente compatible con dispositivos móviles, lo que permite escanear o visualizar códigos desde cualquier lugar. Esta facilidad lo convierte en una opción ideal para desarrollar aplicaciones web modernas, interactivas y accesibles. Por último, las librerías utilizadas se integran fácilmente con HTML y CSS, lo que facilita su incorporación en cualquier proyecto web.

3.4. C++

Aunque C++ no es tan directo como otros lenguajes como Python o JavaScript para trabajar con códigos QR o de barras, también es posible generar y leer este tipo de códigos utilizando librerías especializadas de terceros y herramientas como OpenCV.

GENERAR Y LEER UN CÓDIGO QR

Para generar códigos QR en C++, una opción es utilizar la biblioteca libqrencode, una herramienta de código abierto que permite codificar texto en un código QR y exportarlo como imagen (por ejemplo, en formato PNG). Esta biblioteca se puede compilar junto a un proyecto en C++ y permite personalizar el tamaño del QR, los márgenes y el nivel de corrección de errores.

Para leer códigos QR, se puede emplear OpenCV junto con bibliotecas como ZXing-C++, que permite detectar y decodificar códigos QR desde imágenes o desde el flujo de vídeo de una cámara. Esta combinación permite capturar una imagen, procesarla y extraer el contenido del QR de forma automatizada.

GENERAR Y LEER UN CÓDIGO DE BARRAS

Para generar códigos de barras, se puede utilizar la biblioteca Barcode Writer in Pure C++ (BWIPP) junto con herramientas como Ghostscript o incluso generar directamente imágenes desde la terminal usando scripts que interactúan con C++. También se pueden generar códigos con herramientas externas (como ZXing) y luego integrarlos en la aplicación C++.

Para leer códigos de barras, al igual que con los códigos QR, ZXing-C++ es una de las librerías más utilizadas. Esta biblioteca permite leer distintos tipos de códigos de barras (EAN, UPC, Code128, etc.) desde imágenes o secuencias de vídeo, siendo compatible con OpenCV para capturar las imágenes y detectar los códigos automáticamente.

4. Comparación entre los Lenguajes

Podemos comparar los lenguajes desde distintos puntos de vista:

Facilidad de Implementación

En Python es muy fácil programar ya que las librerías se descargan fácilmente y solamente es imprescindible usar qrcode, python-barcode y opencv-python. Además, el código generado es corto y legible, por lo que es una muy buena opción para generar los códigos QR y de barras de forma rápida.

Java, a pesar de ser más robusto, necesita una mayor preparación previa ya que es necesario configurar dependencias, compilar, vigilar que esté todo perfectamente estructurado... Por lo que es recomendable únicamente si el código que vas a generar necesita ser manipulado por muchas personas como en aplicaciones empresariales.

JavaScript está más orientado para entornos web, integrándose en cualquier página web con qrcode.js o jsbarcode. Además, no necesita instalación ya que se ejecuta en el navegador. Por tanto, está más orientado al uso de usuarios finales.

C++, por su parte, necesita bibliotecas como OpenCV o ZBar. A simple vista no parece muy complejo pero, al ser un lenguaje tan simple, es necesario tener un mayor conocimiento técnico y manejo de dependencias. Sin embargo, es potente y rápido, por lo que es ideal para integraciones con software.

Casos de Uso

Como se ha mencionado, cada lenguaje tiene unas características propias, por lo que son más indicados para su uso dependiendo del contexto en el que se deba generar el código:

Python, por ejemplo, es ideal para prototipos o scripts automáticos debido a su facilidad de programación.

Java, por su parte, es ideal para sistemas robustos, back-end o apps móviles (como las de Android).

JavaScript es el más eficiente para webs interactivas, formularios, QR online...

Y, por último, C++ está enfocado para aplicaciones a bajo nivel o, incluso, hardware.

Rendimiento y Soporte de Bibliotecas

Python es un buen soporte de bibliotecas y tiene en general un buen rendimiento. Sin embargo, puede volverse lento si no se optimiza en proyectos grandes.

Java, por su parte, tiene igualmente un buen rendimiento y, además, una buena gestión de memorias, por lo que es recomendable para los proyectos grandes que puedan saturar a Python.

JavaScript es rápido para los usuarios, pero posee un soporte limitado en

funcionalidades avanzadas de imágenes si no cuenta con ayuda de APIs Web. Sin embargo, sigue siendo el idóneo para la experiencia de usuario.
Por último, C++ tiene un mayor rendimiento y control hardware pero, al necesitar un mayor conocimiento técnico, se suele dejar como última opción si se puede usar Python o Java a pesar de tener un menor rendimiento, además de que Python y, en menor medida, Java son más sencillos para el soporte de bibliotecas.

Criterio	Python	Java	JavaScript	C++
Facilidad de uso	Muy sencillo	Mejorable	Muy sencillo	Mejorable
Casos de uso	Scripts, backend	Android, backend	Web, formularios	Aplicaciones de bajo nivel
Rendimiento	Medio	Alto	Medio-Alto	Muy Alto
Soporte de bibliotecas	Excelente	Excelente	Bueno	Bueno

5. Códigos NaviLens

¿Qué son los códigos NaviLens?

NaviLens es un sistema de códigos visuales similar a los códigos QR, pero diseñado específicamente para personas con discapacidad visual. A diferencia de los códigos tradicionales, NaviLens utiliza un patrón de colores brillantes como negro, rosa, azul, verde y amarillo. Esta combinación permite que los códigos sean detectados rápidamente desde largas distancias y en movimiento, facilitando su uso en entornos públicos y dinámicos.

¿Por qué son más accesibles?

Una de las principales ventajas de NaviLens es su capacidad de lectura a distancia. Estos códigos pueden ser detectados a más de 10 metros sin necesidad de enfocar o encuadrar la cámara, lo que los hace ideales para personas con discapacidad visual. Además, pueden leerse mientras el usuario está caminando, incluso a velocidad normal, gracias a la capacidad de la app de escanear de forma continua. Otra característica clave es que no es necesario apuntar la cámara directamente, ya que la aplicación NaviLens detecta automáticamente los códigos y proporciona la información mediante voz, sin necesidad de tocar la pantalla. Esta interfaz inclusiva mejora significativamente la autonomía de las personas ciegas o con baja visión.

Diferencias clave respecto a los códigos QR

A diferencia de los códigos QR, que son en blanco y negro, los códigos NaviLens utilizan múltiples colores que facilitan su detección. Mientras que los QR requieren que la cámara esté enfocada y fija, NaviLens puede ser leído en movimiento y sin necesidad de precisión en el encuadre. Además, su alcance es mucho mayor, pudiendo ser leídos a distancias de hasta 15 metros o más. En cuanto a accesibilidad, los códigos QR ofrecen opciones muy limitadas para personas con discapacidad visual, mientras que NaviLens ha sido diseñado específicamente para este grupo. Aunque requiere una aplicación dedicada, su funcionalidad es altamente inclusiva.

Aplicaciones reales de NaviLens

NaviLens ya se está utilizando en diversos entornos del mundo real. En el transporte público, por ejemplo, el metro de Barcelona ha incorporado señalética con estos códigos para facilitar la orientación. Museos como el Museo del Prado o el Museo Nacional de Ciencias Naturales también han implementado NaviLens para ofrecer descripciones accesibles de sus exposiciones. En edificios públicos y universidades, los códigos se usan para indicar la ubicación de salas, aulas o servicios accesibles. Incluso algunas marcas comerciales, como Kellogg's, han comenzado a incluir NaviLens en sus embalajes, permitiendo que las personas con discapacidad visual accedan a la información del producto de manera autónoma.

6. Conclusión

Resumen de lo aprendido

A lo largo de este trabajo, hemos comprendido el funcionamiento de los códigos de barras y los códigos QR, analizando sus diferencias tanto en estructura como en capacidad tecnológica. También aprendimos a generarlos y leerlos utilizando distintos lenguajes de programación como Python, Java, JavaScript y C++, lo que nos permitió evaluar su implementación y rendimiento en distintos entornos. Además, exploramos los usos más comunes de estos códigos en la actualidad, desde el comercio hasta el acceso a contenidos digitales. Por último, descubrimos los códigos NaviLens, un avance significativo en accesibilidad, especialmente diseñados para personas con discapacidad visual.

Reflexión sobre la evolución de los códigos visuales

Los códigos visuales han evolucionado notablemente desde su creación. Inicialmente concebidos para leer precios en tiendas, hoy en día actúan como puentes entre el entorno físico y el mundo digital. Su aplicación se ha expandido a múltiples áreas como el marketing, la salud, el transporte y los pagos móviles. Esta evolución no solo ha significado mejoras técnicas, sino también avances hacia una mayor inclusión. El desarrollo de herramientas como NaviLens demuestra que la tecnología puede adaptarse para garantizar la accesibilidad a toda la población.

Opinión sobre el futuro de NaviLens

Más allá de una innovación tecnológica, NaviLens representa un cambio en la forma de concebir el acceso a la información. Este sistema permite que cualquier persona, independientemente de sus capacidades visuales, pueda interactuar con el entorno de forma autónoma y segura. Si continúa recibiendo el respaldo de instituciones y empresas, NaviLens tiene el potencial de convertirse en un estándar global de accesibilidad visual. Creemos que su integración en espacios públicos, productos y servicios será cada vez más común, contribuyendo a una sociedad más equitativa e inclusiva.

7. Kahoot

Kahoot