# Del código al programa



En esta sección vamos a ver el proceso que hay desde la escritura de código en un lenguaje de programación (código fuente) hasta la ejecución del mismo en el sistema operativo (código objeto).

## Código compilado

En primer lugar veremos varios ejemplos de lenguajes cuyo código se compila: C, Go y Rust. Los ejemplos los haremos en línea de comandos y el código simplemente imprimirá los números del 1 al 10. No es la intención de esta sección enseñar a programar en estos lenguajes si no mostrar el proceso de compilación y ejecución de un programa por lo que no te preocupes si no entiendes el código.

### C

Para realizar el proceso de codificación y compilación en C (y en el resto de los lenguajes) lo primero que hemos de hacer es crear un fichero de texto plano (sin formato) con la extensión `.c`. En este ejemplo lo llamaré `ejemplo_c.c`. El contenido del fichero será el siguiente:

```
#include <stdio.h>

funcion main() {
    for (int i = 1; i <= 10; i++) {
        printf("%d\n", i);
    }
}
```

A continuación hemos de compilar el fichero. Para ello abrimos una terminal y nos movemos a la carpeta donde se encuentra el fichero. Una vez allí ejecutamos el siguiente comando:

```
cc cuenta.c -o cuenta
```

Este proceso compilará el fichero `cuenta.c` y generará un ejecutable llamado `cuenta` (en GNU Linux y variantes) o `cuenta.exe` (en sistemas Windows). Para ejecutar el programa hemos de ejecutar el siguiente comando:

```
./cuenta
```

O bien en Windows:

```
cuenta.exe
```

### Nota: Instalación de un compilador de C

Es muy probable que el comando `gcc` o `cc` no funcione ya que no es común que dispongamos de un compilador de C en nuestro sistema. En el caso de que no tengamos un compilador de C podemos instalarlo ejecutando el siguiente comando:

**Ubuntu Linux**

```
sudo apt install build-essential
```

**Windows**

```
winget install --id=GNU.GCC
```

O usando el sistema de paquetes scoop:

```
scoop install gcc
```

## Go

El caso de Go es muy similar al de C pero con la diferencia de que Go es un lenguaje más moderno y con ciertas peculiaridades. Para crear una aplicación Go hemos de crear un *módulo* de Go (Go no permite que nuestro código ande suelto por el monte, ha de pertenecer a un módulo). Para ello hemos de crear un directorio y crear un módulo dentro del mismo:

```
mkdir ejemplo_go
cd ejemplo_go
go mod init cuenta
```

Con estos comandos hemos creado un módulo de Go llamado `cuenta` dentro de la carpeta `ejemplo_go`.

Para compilar el código Go y obtener un ejecutable hemos de crear un fichero de con extensión `.go` dentro de dicho directorio y con el siguiente contenido:

```go
func main() {
    for i := 1; i <= 10; i++ {
        fmt.Println(i)
    }
}
```

Finalmente para compilar el fichero ejecutamos el siguiente comando:

```
go build .
```

Y como resultado de este proceso obtendremos un ejecutable llamado `cuenta.exe` que podemos ejecutar.

## Rust

## Caso especial: Lenguajes que compilan a *bytecode*

Antes de hablar de el caso especial de estos lenguajes necesitamos entender qué es *bytecode*.

*Bytecode*, **código intermedio** o **código portable** es un *conjunto de instrucciones* diseñado para ser ejecutado por un **intérprete software**. Es decir, cuando un programa compila a *bytecode* lo que hace generar un código escrito mediante un *conjunto especial de instrucciones* que **no serán para el procesador del ordenador** si no para ese **intérprete software**. Será ese intérprete el que haga de intermediario entre el *bytecode* y el sistema operativo para ejecutar las instrucciones.

Tres ejemplos de lenguajes que *compilan a bytecode* son Java, Kotlin y C#. Los dos primeros generan *bytecode* para la máquina virtual de java (JVM) mientras que C# lo hace para CLR (Common Language Runtime) de .NET.

A continuación veremos ejemplos de estos lenguajes. Una vez más lo haremos en línea de comandos para que se vea el proceso de compilación y ejecución de los programas y no se vea oculto por un IDE.

### Ejemplo de Java

Para crear un programa en Java hemos de crear un fichero con extensión `.java` y con el siguiente contenido:

```java
public static void main(String[] args) {
    for (int i = 1; i <= 10; i++) {
        System.out.println(i);
    }
}
```

Una vez creado el fichero hemos de compilarlo. Para ello abrimos una terminal y nos movemos a la carpeta donde se encuentra el fichero. Una vez allí ejecutamos el siguiente comando:

```
javac cuenta.java
```

Esto creará un fichero `cuenta.class` que es el *bytecode* de nuestro programa. Para ejecutarlo; es decir, para enviárselo a la JVM, hemos de ejecutar el siguiente comando:

```
java cuenta
```

#### Instalación de Java

De nuevo, para que estos programas (compilador y máquina virtual) funcionen hemos de tener instalado Java en nuestro sistema. Para instalar Java en Ubuntu Linux ejecutamos el siguiente comando:

```
apt install default-jdk
```

En Windows podemos instalarlo con:

```
winget install --id=AdoptOpenJDK.OpenJDK
```

O bien descargando cualquier JDK, por ejemplo, de AdoptOpenJDK.

***Nota:*** *No hay ningún motivo por el que haya puesto dicho JDK. El que prefieras es el que has de instalar.*

## Ejemplo de Kotlin

#### Instalación de Kotlin

La instalación del compilador de línea de comandos de Kotlin hemos de seguir las instrucciones de su página web oficial Kotlin command line compiler. Aunque la forma más simple de instalar Kotlin es mediante el IDE de IntelliJ IDEA o Android Studio.

Podremos comprobar que Kotlin se ha instalado correctamente ejecutando el siguiente comando:

```
kotlin -version
Kotlin version 2.0.0-release-341 (JRE 21.0.4+7-LTS)
```

Para escribir el código en Kotlin hemos de crear un fichero con extensión `.kt` y con el siguiente contenido:

```
fun main() {
    for (i in 1..10) {
        println(i)
    }
}
```

Y para compilarlo ejecutamos el siguiente comando:

```
kotlinc cuenta.kt -include-runtime -d cuenta.jar
```

- La opción `-d` indica el nombre del fichero de salida (en este caso `cuenta.jar`).
- La opción `-include-runtime` incluye la biblioteca de tiempo de ejecución de Kotlin en el archivo JAR. Esto hace que se pueda ejecutar el archivo JAR sin contar con nada más instalado en el sistema.

Para ejecutar el programa hemos de pasar el *bytecode* a la JVM. Para ello ejecutamos el siguiente comando:

```
java -jar cuenta.jar
```

# Código interpretado

Como ejemplo de lenguajes interpretados veremos Python  y JavaScript.

## Python

### Instalación de Python

Alguna de las versiones de Python suele venir instalada por defecto con el sistema operativo. Para comprobar si tenemos Python instalado ejecutamos el siguiente comando:

```
python --version
```

Si nos devuelve algo como `Python 3.12.2` es que tenemos Python instalado. Si no es así podemos instalarlo con el siguiente comando:

```
winget install python
```

Una vez instalado, para escribir un programa en Python hemos de crear un fichero con extensión `.py` y con el siguiente contenido:

```
for i in range(1, 11):
    print(i)
```

Y para ejecutarlo hemos de abrir una terminal y ejecutar el siguiente comando:

```
python cuenta.py
```

***Nota:*** *En algunos sistemas operativos el comando `python` puede estar asociado a Python 2. Para ejecutar el programa con Python 3 hemos de usar el comando `python3`.*

## JavaScript

### Instalación de Node.js

Node.js es un entorno de ejecución para JavaScript que nos permite ejecutar código JavaScript en el sistema operativo (no en el navegador). Para instalar Node.js hemos de ir a su página web oficial Node.js y descargar el instalador correspondiente a nuestro sistema operativo. Otra opción es utilizar el sistema de paquetes de sistema operativo que usemos. Por ejemplo, para instalar Node.js en Windows 11 podemos ejecutar el siguiente comando:

```
winget install --id OpenJS.NodeJS
```

O bien seguir las instrucciones de formas alternativas de instalación de la página de Node.js.

Una vez instalado hemos de crear un fichero con el código fuente con la extensión `.js` y con el siguiente contenido:

```
for (let i = 1; i <= 10; i++) {
    console.log(i);
}
```

En este caso no hemos de compilar el código. Para ejecutarlo hemos de abrir una terminal y ejecutar el siguiente comando:

```
node cuenta.js
```

De este modo le pasamos las instrucciones escritas en JavaScript al intérprete de Node.js y este las ejecuta. Como podemos ver no se genera ningún archivo intermedio como en el caso de los lenguajes que compilan a *bytecode* o un ejecutable como en el caso de los lenguajes compilados.