# Índice



# Ejercicios Unidad 6

Descargar estos ejercicios

> **Antes de empezar**
>
> Para realizar estos ejercicios, deberás descargar los recursos del enlace de proyecto_bucles anterior. Cada ejercicio se codificará en el cuerpo del método que corresponda con el nombre del ejercicio. Es decir, sustituirás las líneas `//TODO:` por el código de solución del ejercicio. Para probar el funcionamiento correcto de los ejercicios, deberás pasar los Test adjuntos a este proyecto ( como se explica en la unidad 4 ).

## Ejercicio 1. Contador de números

Escribe un programa que pida al usuario que vaya introduciendo números hasta que introduzca un **0** y cuente cuántos de ellos son positivos y cuantos negativos.

**Ejemplo de ejecución:**

```
Ejercicio 1: Contador de números
Introduzca valor 1: 10
Introduzca valor 2: -5
Introduzca valor 3: 2
Introduzca valor 4: 4
Introduzca valor 5: -7
Introduzca valor 6: -9
Introduzca valor 7: -3
Introduzca valor 8: -6
Introduzca valor 9: 0
Números positivos introducidos: 3
Números negativos introducidos: 5
```

**Requisitos:**

- Usa un bucle `do-while` para resolverlo.
- La condición de salida del bucle será cuando el número introducido sea igual a 0.
- Usa el condicional `if - else` para incrementar el `contador` correspondiente.
- No cuentes el 0 introducido.

## Ejercicio 2. Suma y producto

Programa que calcula y muestra la **suma** y el **producto** de los **10 primeros** números naturales.

```
Ejercicio 2: Suma y producto
SUMA: 55
PRODUCTO: 3628800
```

**Requisitos:**

- Usa un bucle `for`.
- Se deben usar `acumuladores` para resolverlo.
- No olvides inicializar a **1** la variable para el producto.

## Ejercicio 3. Contador de números pares

Crea un programa que cuente cuántos números pares hay entre 1 y un número introducido por el usuario.

```
Ejercicio 3: Contador de números pares
Introduce un número: 10
Entre 1 y 10 hay 5 números pares
```

**Requisitos:**

- Usa un bucle `for` para recorrer los números.
- Usa el operador módulo `%` para verificar si un número es par.
- Usa un `acumulador` contador para llevar la cuenta.

## Ejercicio 4. Suma de números

Escribe un programa que vaya pidiendo números al usuario mientras sean positivos y al final muestre la suma total.

```
Ejercicio 4: Suma de números
Introduce un número (0 o negativo para terminar): 5
Introduce un número (0 o negativo para terminar): 3
Introduce un número (0 o negativo para terminar): 7
Introduce un número (0 o negativo para terminar): 0
La suma total es: 15
```

**Requisitos:**

- Usa un bucle `do-while`.
- Usa una variable `acumulador` para la suma.
- El bucle debe terminar cuando se introduzca 0 o un número negativo.

## Ejercicio 5. Producto mediante sumas sucesivas

Programa que obtenga el producto de dos números enteros positivos mediante sumas sucesivas. Esto es, para calcular `2 * 5` haga `2 + 2 + 2 + 2 + 2`

```
Ejercicio 5: Factorial de un número
Introduzca operador 1: -1
Introduzca operador 2: 5
ERROR: Sólo se permiten números positivos

Introduzca operador 1: 4
Introduzca operador 2: 3
Sumando....
 4 x 3 = 12
```

**Requisitos:**

- Usa un bucle `for`.
- Usa una variable tipo `double` para el `acumulador` del producto.

## Ejercicio 6. Validación de entrada

Escribe un programa que pida al usuario un número entre 1 y 10. Si el número no está en ese rango, debe seguir pidiendo hasta que introduzca un valor válido.

```
Ejercicio 6: Validación de entrada
Introduce un número entre 1 y 10: 15
Número inválido. Introduce un número entre 1 y 10: -3
Número inválido. Introduce un número entre 1 y 10: 7
Número válido: 7
```

**Requisitos:**

- Usa un bucle `do-while`.
- Usa operadores de comparación para validar el rango.
- Muestra un mensaje de error cuando el número no sea válido.

## Ejercicio 7. División mediante restas sucesivas

Programa que obtenga el cociente y el resto de dividir dos números enteros positivos utilizando restas. Por ejemplo, para calcular `5 / 2` haga `cociente -= 2` mientras `cociente >= 2` el cociente, que al principio estará inicializado a 5, será el número de veces que se ha podido restar.

```
Ejercicio 7. División mediante restas sucesivas
Introduzca dividendo: 10
Introduzca divisor: -2
ERROR: Sólo se permiten números positivos

Introduzca dividendo: 10
Introduzca divisor: 2

10 / 2 = 5
Resto: 0
```

**Requisitos:**

- Usa un bucle `while` para verificar divisibilidad.
- Usa un `acumulador` para guardar las restas sucesivas.
- El número que quede que ya no se pueda restar, será el resto de la división.

## Ejercicio 8. Adivinar número

Escribe un programa que genere un número aleatorio entre 1 y 100. El usuario debe adivinarlo y el programa le dirá si su número es mayor, menor o correcto.

```
Ejercicio 8: Adivinar número
Adivina el número entre 1 y 100
Introduce tu número: 50
El número es mayor
Introduce tu número: 75
El número es menor
Introduce tu número: 63
¡Correcto! Has adivinado el número en 3 intentos
```

**Requisitos:**

- Usa el objeto de tipo `Random` que se proporciona en el método para generar un número aleatorio entre 1 y 100.
- Usa un bucle `while` hasta que adivine.
- Usa una variable contador para los intentos.

## Ejercicio 9. Máximo y mínimo

Crea un programa que pida 5 números al usuario y muestre cuál es el mayor y cuál es el menor.

```
Ejercicio 9: Máximo y mínimo
Introduce el número 1: 23
Introduce el número 2: 45
Introduce el número 3: 12
Introduce el número 4: 67
Introduce el número 5: 34
El número mayor es: 67
El número menor es: 12
```

**Requisitos:**

- Usa un bucle `for` para pedir los 5 números.
- Inicializa el máximo y mínimo con el número menor y mayor del tipo entero (consulta el punto *Esquema algorítmico para obtener máximos y mínimos* del tema).
- Usa condicionales dentro del bucle para actualizar máximo y mínimo.

## Ejercicio 10. Secuencia Fibonacci

Escribe un programa que genere los primeros N números de la secuencia de Fibonacci, donde N es introducido por el usuario.

```
Ejercicio 10: Secuencia Fibonacci
Introduce cuántos números de Fibonacci quieres: 8
0, 1, 1, 2, 3, 5, 8, 13
```

**Fórmula:**

- F(0) = 0, F(1) = 1
- F(n) = F(n-1) + F(n-2) para n > 1

**Requisitos:**

- Usa un bucle `for`.
- Necesitarás variables para los dos números anteriores de la secuencia.