Índice

Ejercicios Unidad 7

Descargar estos ejercicios

Antes de empezar

Para realizar estos ejercicios, deberás descargar los recursos del enlace de proyecto_funciones anterior. Para probar el funcionamiento correcto de los ejercicios deberás pasar los Test adjuntos a este proyecto.
Cada ejercicio estará compuesto por dos métodos:

- Uno con el mismo nombre del ejercicio que servirá para la recogida de datos y para la llamada al método que realiza la funcionalidad. Este método viene con el proyecto y tendrás que añadir la funcionalidad sustituyendo las líneas `//TODO:` por el código de solución del ejercicio.
- El o los otros métodos que deberán tener el nombre que se indica en cada uno de los ejercicios.
- Seguramente querrás pasar los Test a la vez que vayas resolviendo ejercicios, por lo que deberás comentar los test de los métodos que todavía no hayas creado, para quitar los errores usa `/* */` incluyendo dentro las líneas de código que no necesites.

Ejercicio 1. Función sin parámetros de entrada y sin retorno

Escribe una función que muestre información personal de forma estructurada. Pedirá los datos necesarios dentro del método y los mostrará en el mismo.

```
Ejercicio 1. Función sin parámetros de entrada y sin retorno
Introduce tu nombre: Juan
Introduce tu apellido: García
Introduce tu edad: 25
Introduce tu ciudad: Alicante

=== INFORMACIÓN PERSONAL ===
Nombre completo: Juan García
Edad: 25 años
Ciudad de residencia: Alicante
==========================
```

Requisitos:

- Define una función estática `void MuestraInformacion()`.
- Pide al usuario que introduzca los datos necesarios, según salida.
- La función debe formatear y mostrar la información con el diseño especificado.
- No retorna ningún valor.

Ejercicio 2. Función con parámetros de entrada y sin retorno

Crea una función que calcule el volumen de una esfera dado su radio.

```
Ejercicio 2. Función sin parámetros de entrada y sin retorno
Introduce el radio de la esfera: 3
El volumen de la esfera es: 113,10
```

Requisitos:

- Define una función estática `void MuestraVolumenEsfera(double radio)`.
- Usa la fórmula: $V = (4.0/3.0) \times \pi \times r^3$
- Usa `Math.PI` y `Math.Pow()` para los cálculos.
- Muestra el resultado con dos decimales en la propia función.

Ejercicio 3. Función con múltiples parámetros y sin retorno

Crea una función que determine el mayor de tres números enteros.

```
Ejercicio 3: Función con múltiples parámetros y sin retorno
Introduce el primer número: 15
Introduce el segundo número: 23
Introduce el tercer número: 8
El mayor de los tres números es: 23
```

Requisitos:

- Define una función estática `void Mayor(int a, int b, int c)`.
- Usa condicionales para determinar el mayor.
- La función debe mostrar el valor mayor.
- Si los tres números son iguales la función mostrará.

```
Ejercicio 3: Función con múltiples parámetros y sin retorno
Introduce el primer número: 3
Introduce el segundo número: 3
Introduce el tercer número: 3
Los números son iguales
```

Ejercicio 4. Función sin parámetros de entrada y con parámetros de retorno

Escribe una función que a partir del día actual nos devuelve una cadena formateada como se muestra en la salida.

```
Ejercicio 4. Función que retorna la fecha actual formateada
Hoy estamos a 13 de noviembre de 2025
```

Requisitos:

- Define una función estática `string FormateaFecha()`.
- Usa la clase DateTime de **System.DateTime** para conseguir la fecha actual.
- Con el método toString del objeto DateTime obtén el mes con formato largo.
- Crea la cadena con los datos extraídos del objeto DateTime y con el mes obtenido.
- El método retornará la cadena.
- Evidentemente, la fecha que se muestra en la salida no será la actual del momento de ejecución de tu programa.

Ejercicio 5. Función con parámetro de entrada y de retorno

Crea una función que valide si un año es bisiesto y úsala en el programa principal.

```
Ejercicio 5: Función de validación
Introduce un año: 2024
El año 2024 es bisiesto
```

Requisitos:

- Define una función estática `bool EsBisiesto(int año)`.
- Un año es bisiesto si es divisible por 4, excepto los años divisibles por 100, a menos que también sean divisibles por 400.

Ejercicio 6. Función generadora de contraseñas

Escribe una función que genere una contraseña aleatoria con criterios específicos.

```
Ejercicio 6. Función generadora de contraseñas
Introduce la longitud deseada: 12
¿Incluir números? (s/n): s
¿Incluir símbolos? (s/n): n
Contraseña generada: AbcDefGhiJkL
```

Requisitos:

- Define una función estática: `string GeneraContraseña(int longitud, bool incluirNumeros, bool incluirSimbolos)`
- Usa siempre letras mayúsculas y minúsculas.
- Si `incluirNumeros = true`, añade dígitos `0-9`.
- Si `incluirSimbolos = true`, añade símbolos como `!#"$%&`
- Usa `Random` para generar caracteres aleatorios. Como ya sabemos el Random nos puede proporcionar números aleatorios, pero lo que necesitamos en este ejercicio son caracteres, por lo que vamos a convertir el número generado en carácter:
  - Para ello generaremos los números que se asocian en la tabla ASCII, a los caracteres que necesitamos. Por ejemplo, para los caracteres especiales podemos coger el rango entre `33 y 38`. El número generado tendrás que convertirlo a char usando `Convert.ToChar()` o una conversión más explícita con el casting directo `(char)numero`
- Además podremos mejorar el ejercicio si para cada nuevo carácter que queremos generar, le damos aleatoriedad de forma que pueda ser mayúscula, minúscula, número o carácter especial indistintamente. Para ello podremos generar números aleatorios entre 1 y 4 y con switch de expresión con when optar por una de las opciones.

- Deberemos generar números mientras la contraseña no tenga la longitud deseada.

Ejercicio 7. Calculadora de préstamos

Implementa una función que calcule los pagos mensuales de un préstamo.

```
Ejercicio 7. Calculadora de préstamos
Introduce el monto del préstamo: 10000
Introduce la tasa de interés anual (%): 5
Introduce el plazo en años: 3

=== DETALLES DEL PRÉSTAMO ===
Monto: 10.000,00
Tasa anual: 5,00%
Plazo: 3 años (36 meses)
Pago mensual: 299,71
Total a pagar: 10.789,66
Intereses totales: 789,66
============================
```

Requisitos:

- Define una función estática:
  `(double pagoMensual, double totalPagar, double interesesTotales) CalculaPrestamo(double monto, double tasaAnual, int años)`
- Valida que todos los valores sean positivos con `Debug.Assert()`.
- Usa la fórmula de amortización, donde **PMT** es el pago mensual:

$$PMT = P \cdot \frac{r(1+r)^n}{(1+r)^n - 1}$$

- Donde **P = monto**, **r = tasa mensual**, **n = número de pagos**
- **r** es `tasaAnual/100/12`
- **n** es los años de plazo multiplicados por los 12 meses del año.
- El `totalPagar` se calcula multiplicando el pago mensual por el número de pagos.
- Los `interesesTotales` es total a pagar menos el monto.
- Retorna tupla con pago mensual, total a pagar e intereses totales.

Ejercicio 8. Función con parámetros de entrada complejos

Escribe una función que calcule la distancia entre dos puntos en un plano cartesiano usando tuplas.

```
Ejercicio 8. Función con parámetros de entrada complejos
Introduce las coordenadas del primer punto:
X1: 1
Y1: 2
Introduce las coordenadas del segundo punto:
X2: 4
Y2: 6
La distancia entre los puntos es: 5,00
```

Requisitos:

- Define una función estática: `double Distancia((double x, double y) p1, (double x, double y) p2)`
- Usa la fórmula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- Usa `Math.Sqrt()` para calcular la raíz cuadrada.
- Muestra el resultado con dos decimales.

Ejercicio 9. Sobrecarga para cálculos de tiempo

Crea múltiples versiones sobrecargadas de una función para convertir tiempo a segundos.

```
Ejercicio 9. Sobrecarga para cálculos de tiempo
Introduce los días: 1
Introduce las horas: 2
Introduce los minutos: 30
Tiempo total en segundos: 95400
```

Requisitos:

- Define tres funciones sobrecargadas llamadas `TiempoASegundos`:
  - `int TiempoASegundos(int minutos)`
  - `int TiempoASegundos(int horas, int minutos)`
  - `int TiempoASegundos(int dias, int horas, int minutos)`
- Prueba los tres métodos con distintas llamadas.

Ejercicio 10. Función con validación de entrada

Escribe una función que lea y valide un número entero dentro de un rango específico.

```
Ejercicio 10. Función con validación de entrada
Introduce un número entre 1 y 100: 150
Número fuera de rango. Introduce un número entre 1 y 100: -5
Número fuera de rango. Introduce un número entre 1 y 100: 50
Número válido introducido: 50
```

Requisitos:

- Define una función estática **`int LeeEnteroEnRango(int min, int max)`**.
- La función debe validar que el número esté en el rango especificado.
- Si no es válido, debe seguir pidiendo hasta obtener un valor correcto.
- Usa `int.TryParse()` para validar que la entrada sea un número.