



## OBJETIVOS DE APRENDIZAJE

- **Aprender a definir y usar funciones**
  - **Funciones que devuelven un único resultado**
  - **Funciones que devuelven más de un resultado.**

## EJERCICIO 4.1 CONVERSIÓN DE TEMPERATURAS

En diferentes lugares del mundo, se usa la escala *Fahrenheit* para representar la temperatura. En Europa, utilizamos la escala *Celsius* o centígrada. Para convertir grados *Fahrenheit* a *Celsius*, debemos aplicar la siguiente fórmula:

$$c = \frac{(F - 32) * 5}{9}$$

- a) Crear una función llamada *fahrenheitToCelsius* para convertir grados *Fahrenheit* a grados *Celsius*.

```
function celsius = fahrenheitToCelsius(fahrenheit)
```

- b) Crear una función llamada *celsiusToFahrenheit* para convertir grados *Celsius* a grados *Fahrenheit*.

```
function fahrenheit = celsiusToFahrenheit(celsius)
```

- c) Probar ambas funciones en la ventana de comandos.

```
>> celsiusToFahrenheit(0)
ans =
32
>> fahrenheitToCelsius(32)
ans =
0
```

## EJERCICIO 4.2 POLINOMIO DE SEGUNDO GRADO

Implementar una función llamada *secondDegreePolynomial* que devuelva la raíz positiva y negativa de una ecuación de segundo grado. Los parámetros de entrada son los coeficientes *a*, *b* y *c*. Las dos raíces se obtienen según la siguiente fórmula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

El prototipo de la función es el siguiente:

```
function [ posRoot,negRoot ] = secondDegreePolynomial ( a, b, c )
```

Prueba la función con los siguientes ejemplos:

```
>> [posRoot,negRoot]= secondDegreePolynomial (5,2.4,1)
```

```
posRoot =
```

```
-0.4098
```

```
negRoot =
```

```
-11.5902
```

Execution in Command Window:

```
>> [posRoot,negRoot]= secondDegreePolynomial (1,2,3)
```

```
posRoot =
```

```
-1.0000 + 1.6583i
```

```
negRoot =
```

```
-1.0000 - 1.6583i
```

## EJERCICIO 4.3 MEDIR EL VOLUMEN

Una empresa de nuestro entorno está calibrando una nueva herramienta para tomar medidas y, para ello, ha medido el radio y la altura de 8 cilindros diferentes. Las medidas son las siguientes:

```
r = [5.499 5.498 5.5 5.5 5.52 5.51 5.5 5.48];
```

```
h = [11.1 11.12 11.09 11.11 11.11 11.1 11.08 11.11];
```

Para verificar la calibración, calculamos la superficie y el volumen de cada cilindro y además comprobamos que todos los valores de r y de h sean positivos. Para ello:

- Crear una función *cylinderVolume*, la cual recibe los vectores radio (*r*) y altura (*h*) y calcula el vector volumen. La función también debe comprobar que los valores sean positivos ( $>0$ ). Para calcular el volumen es necesario usar la siguiente fórmula:  $v = \pi r^2 h$ .

```
function [vol,err] = cylinderVolume(r, h)
```

- Crear un script llamado *testCylinder.m* donde se definen los vectores r y h del enunciado y además que dibuje una gráfica que muestre los volúmenes para comprobar su variabilidad. También se muestra un mensaje con la media y la desviación estándar de los volúmenes.

## EJERCICIO 4.4 CALCULAR PRECIOS TOTALES

- a) Escribir una función *calculatePrices* que nos pida dos vectores y un escalar como datos de entrada:
- en el primer vector tenemos los precios de algunos productos adquiridos en una tienda,
  - en el segundo vector tenemos el número de ejemplares de cada producto,
  - y el escalar es el impuesto (porcentaje de tipo IVA).

Esta función nos tiene que devolver un vector con el precio total por producto y la suma total a pagar. Por ejemplo:

```
>>[prices,totalPrice]=calculatePrices([10 20 3],[2 1 3], 16)
prices =
    23.2000    23.2000    10.4400
totalPrice =
    56.8400
```

- b) Escribir un script *ticket.m* que pida al usuario los precios y el número de ejemplares de 3 productos diferentes y el porcentaje del impuesto. Utilizando la función del apartado a), escribir el resultado según el siguiente formato:

```
Paid for the first product: 23.20
Paid for the second product: 23.20
Paid for the third product: 10.44
Total amount to be paid: 56.84
```