

## UF07. - FUNCIONS

### - *Teoria* -

PROGRAMACIÓ  
CFGs DAW

José Manuel Martí Fenollosa  
[josemanuel.marti@ceedcv.es](mailto:josemanuel.marti@ceedcv.es)

Revisat per:

Guillermo Garrido Portes  
[g.garridoportes@edu.gva.es](mailto:g.garridoportes@edu.gva.es)

2023/2024 1

1. INTRODUCCIÓ
2. DECLARACIÓ D'UNA FUNCIO
3. ANOMENA A UNA FUNCIO
4. ÀMBIT DE LES VARIABLES
5. PARÀMETRES: PAS PER VALOR I PER REFERÈNCIA
6. DEVOLUCIÓ D'UN VALOR

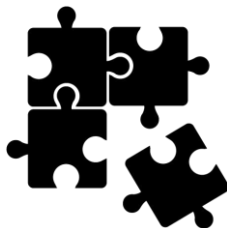
# 1. INTRODUCCIÓ

## INTRODUCCIÓ



GENERALITAT  
VALENCIANA

ceedcv  
CENTRE ESPECÍFIC  
D'EDUCACIÓ A DISTÀNCIA DE  
LA COMUNITAT VALENCIANA



La millor manera de crear i mantindre un **programa** gran és construir-lo **a partir de peces més xicotetes o mòduls**. Cadascun dels quals és més **maneja**ble que el programa íntegrament.

Les **FUNCIONS** (o **SUBPROGRAMES**) són utilitzades per a **evitar la repetició de codi en un programa** en poder executar-les des de diversos punts d'un programa amb només invocar-les.

Permeten la **descomposició funcional i la diferenciació de tasques**.

# 1. INTRODUCCIÓ

## INTRODUCCIÓ



GENERALITAT  
VALENCIANA



### Per a què les utilitzem?

- Agrupar codi que **forma una entitat pròpia** o una idea concreta.
- Agrupar codi que es necessitarà diverses vegades en un programa, amb al missió de **no repetir codi**.
- **Dividir** el codi d'un **programa gran un subprogrames (funcions)**, cadascun d'ells especialitzats a resoldre una part del problema.

### Característiques

- Es defineixen mitjançant un **nom únic** que representa el bloc de codi.
- Poden ser **anomenades (executades) des de qualsevol part del codi**.
- **Se'ls pot passar valors** perquè els processen d'alguna forma.
- **Poden retornar un resultat** per a ser usat des d'on se'ls haja anomenat.

## 2. DECLARACIÓ D'UNA FUNCIO

### DEFINICIÓ



La **capçalera** es declara en una sola línia i es compon de:

- **Modificadors de funció:** Existeixen molts però els veurem en futures unitats. (Per ara només utilitzarem public static).
- **Tipus retornat:** El tipus de dada que retornarà la funció, com per exemple int, double, char, boolean, String, etc. Si la funció no retorna res s'indica mitjançant void.
- **Nom de la funció:** Identificador únic per a cridar a la funció.
- **Llista de paràmetres:** Indica els tipus i noms de les dades que se li passaran a la funció quan siga anomenada. Poden ser varis o cap.

El **cos** és un bloc de codi entre claus { ... } que s'executarà quan des d'una altra part del codi utilitzem la funció.

## 2. DECLARACIÓ D'UNA FUNCIO

### EXEMPLES

**Exemple 1:** Aquest és un exemple molt senzill d'una funció anomenada 'imprimeixHolaMon', que no té paràmetres d'entrada (no hi ha res entre els parèntesis) i no retorna cap valor (indicat per void). Quan la diguem l'única cosa que farà serà escriure per pantalla el missatge "Hola mon".

```
public static void imprimeixHolaMon() {  
    System.out.println("Hola mon");  
}
```

**Exemple 2:** Aquesta funció es diu 'imprimeixHolaNom', té com a paràmetre d'entrada una dada String anomenat 'nom' i no retorna res. Quan la cridem ens imprimirà per pantalla el text "Hola " seguit del String nom que li'l passarem com a paràmetre .

```
public static void imprimeixHolaNom(String nom) {  
    System.out.println("Hola " + nom);  
}
```

## 2. DECLARACIÓ D'UNA FUNCIO

### EXEMPLES



GENERALITAT  
VALENCIANA



**Exemple 3:** Aquesta funció es diu 'doble', té com a paràmetre d'entrada una dada int anomenat 'a' i retorna una dada de tipus int. Quan la cridem calcularà el doble de 'a' i el retornarà (amb el return).

```
public static int doble(int a) {  
    int resultat = a * 2;  
    return resultat;  
}
```

**Exemple 4:** Aquesta funció es diu 'multiplica', té dos paràmetres d'entrada de tipus int anomenats 'a' i 'b' i retorna una dada de tipus int. Quan la cridem calcularà  $a*b$  i ho retornarà (amb el return).

```
public static int multiplica(int a, int b) {  
    int resultat = a * b;  
    return resultat;  
}
```

## 2. DECLARACIÓ D'UNA FUNCIO

### EXEMPLES

**Exemple 5:** Aquesta funció es diu 'maxim', té dos paràmetres d'entrada de tipus double anomenats 'valor1' i 'valor2' i retorna una dada de tipus double. Quan la cridem calcularà el màxim entre 'valor1' i 'valor2' i ho retornarà.

```
public static double maxim(double valor1, double valor2) {  
    double max;  
    if (valor1 > valor2)  
        max = valor1;  
    else  
        max = valor2;  
    return max;  
}
```

**Exemple 6:** Aquesta funció es diu 'sumaVector', té un paràmetre d'entrada tipus int[] (un vector de int) anomenat 'v' i retorna una dada tipus int. Quan la cridem recorrerà el vector 'v', calcularà la suma de tots els seus elements i la retornarà.

```
public static int sumaVector(int v[]) {  
    int suma = 0;  
    for (int i = 0; i < v.length; i++)  
        suma += v[i];  
    return suma;  
}
```



## 2. DECLARACIÓ D'UNA FUNCIO

### IMPORTANT

És important saber que les funcions es declaren dins de 'class' **PERÒ** fora del 'main'.

```
1 package unidad7;
2
3 public class programadeprueba {
4
5     public static void imprimeHolaMundo() {
6         System.out.println("Hola mundo");
7     }
8
9     public static int doble(int a) {
10         int resultado = a * 2;
11         return resultado;
12     }
13
14     public static int multiplica(int a, int b) {
15         int resultado = a * b;
16         return resultado;
17     }
18
19     public static void main(String[] args) {
20         // Un programa siempre empieza ejecutándose por la función main.
21         // Aquí va el código principal de nuestro programa.
22
23     }
24
25 }
26
27
```

En aquest programa tenim 4 funcions: *imprimeixHolaMon*, *doble*, *multiplica* i *main*. Sí, el 'main' on sempre has programant fins ara és en efecte una funció, però una mica especial: 'main' es la funció principal, el punt d'inici d'un programa.

És obligatori que tot programa Java tinga una funció **main**. Si et fixes, és una funció que rep com a paràmetre un `String[]` (vector de `String`) i no retorna res (encara que podria retornar un `int`). El per què d'això ho veurem més endavant.

Les 3 funcions que hem declarat a dalt del main per si soles no fan res, simplement estan ací esperant que siguin anomenades (*utilitzades*), normalment des del propi main.

### 3. ANOMENAR A UNA FUNCIO

#### DEFINICIO



GENERALITAT  
VALENCIANA

ceedcv  
CENTRE ESPECÍFIC  
D'EDUCACIÓ A DISTÀNCIA DE  
LA COMUNITAT VALENCIANA

Les funcions poden ser invocades o anomenades **des de qualsevol altra funció, inclosa ella mateixa.**

*(Ara com ara anomenarem funcions només des de la funció principal 'main')*

Flux d'execució:

S'invoca  
una funció



S'executen les  
instruccions de  
la funció



La funció retorna un  
valor i l'assignem a  
una variable.



Es torna al punt que es va  
invocar la funció i  
continua executant-se

Se li passen els paràmetres  
si n'hi haguera entre ()



### 3. ANOMENAR A UNA FUNCIO

#### EXEMPLES



Exemples: utilitzant les funcions de l'apartat anterior.

```
public static void main(String[] args) {  
    // No té paràmetres ni retorna valor. Simplemente imprimeix "Hola Mon"  
    imprimeixHolaMon();  
  
    // És habitual cridar a una funció i guardar el valor retornat en una variable  
    int a = doble(10); // a = 20    (10*2)  
    int b = multiplica(3, 5); // b = 15    (3*5)  
  
    // Poden passar-se variables com a paràmetres  
    int c = doble(a); // c = 40    (20*2)  
    int d = multiplica(a, b); // d = 300    (20*15)  
  
    // Poden combinar-se funcions i expressions  
    int e = doble(4) + multiplica(2,10); // e = 8 + 20  
    System.out.println("El doble de 35 és " + doble(35) ); // "El doble de 35 és 70"  
    System.out.println("12 per 12 és " + multiplica(12,12) ); // "12 per 12 és 144"  
}
```

### 3. ANOMENAR A UNA FUNCIO

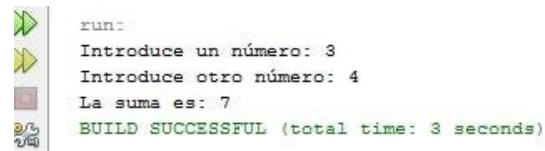
#### EXEMPLES

Exemples: Programa amb una funció que suma dos números.

#### CODI

```
4 public class Suma {
5
6     public static void main(String[] args) {
7         Scanner sc = new Scanner(System.in);
8         int num1, num2, suma;
9
10        System.out.print("Introduce un número: ");
11        num1 = sc.nextInt();
12
13        System.out.print("Introduce otro número: ");
14        num2 = sc.nextInt();
15
16        suma = suma(num1, num2);
17
18        System.out.println("La suma es: " + suma);
19    }
20
21    public static int suma(int n1, int n2) {
22
23        int suma;
24
25        suma = n1 + n2;
26
27        return suma;
28    }
29
30
31 }
```

Eixida



```
run:
Introduce un número: 3
Introduce otro número: 4
La suma es: 7
BUILD SUCCESSFUL (total time: 3 seconds)
```

# 3. ANOMENAR A UNA FUNCIÓN

## EXEMPLES

Exemples: Programa amb una funció que determina si un número és parell o imparell.

### CODI

```
14 public class ParImpar {
15
16     public static void main(String[] args) {
17         Scanner in = new Scanner(System.in);
18         int num;
19
20         System.out.print("Introduce un número: ");
21         num = in.nextInt();
22
23         if(par(num) == true) // Llamada a la función desde la expresión
24             System.out.println(num + " es par.");
25         else
26             System.out.println(num + " es impar.");
27     }
28
29     public static boolean par(int numero)
30     {
31         boolean par = false;
32
33         if(numero % 2 == 0) // Si el resto es 0 par será 'true' sino 'false'
34             par = true;
35
36         return par;
37     }
38 }
```

Eixida



```
run:
Introduce un número: 9
9 es impar.
BUILD SUCCESSFUL (total time: 2 seconds)
```

## 4. ÀMBIT DE LES VARIABLES

### DEFINICIÓ



### ENCAPSULACIÓ

**Una funció només pot utilitzar les variables d'àmbit local**, és a dir, les seues pròpies variables (els paràmetres de la capçalera i les variables creades dins de la funció). Quan una funció s'executa es creen les seues variables, s'utilitzen i quan la funció acaba es destrueixen les variables.

Per tot això **una funció no pot utilitzar variables que estiguen fora d'ella**, i fora d'una funció no és possible utilitzar variables de la pròpia funció. A aquesta característica se'n diu **encapsulació** i permet que les funcions siguen independents entre si, facilitant el disseny de programes grans i complexos.

*Tècnicament sí que és possible que una funció utilitze variables que estan fora d'ella, però això ho veurem en futures unitats quan aprenguem Programació Orientada a Objectes.*

## 5. PARÀMETRES: PAS PER VALOR I PER REFERÈNCIA

### DEFINICIÓ



GENERALITAT  
VALENCIANA

oceedcv  
CENTRE ESPECÍFIC  
D'EDUCACIÓ A DISTÀNCIA DE  
LA COMUNITAT VALENCIANA

2 tipus de paràmetres



- **Paràmetres de tipus simple** (*pas per valor*)
- **Paràmetres de tipus compost** (*pas per referència*)

## 5. PARÀMETRES: PAS PER VALOR I PER REFERÈNCIA

### DEFINICIÓ



GENERALITAT  
VALENCIANA



- **Paràmetres de tipus simple (pas per valor):** Com int, double, boolean, char, etc. En aquest cas es passen per valor. És a dir, el valor es copia al paràmetre i per tant si es modifica dins de la funció això no afectarà el valor fora d'ella perquè són variables diferents.

```
public static void main(String[] args) {  
    int a = 10;  
    System.out.println("Valor inicial de a: " + a); // a val 10  
    imprimeix_doble(a); // Se li passa el 10 a la funció  
    System.out.println("Valor final de a: " + a); // a continua valent 10  
}
```

// El paràmetre 'a' és independent de la 'a' del main. Són variables diferents!

```
public static void imprimeix_doble(int a) { // Es copia el valor 10 a aquesta  
    nova 'a'
```

```
    a = 2 * a; // Es duplica el valor de la 'a' d'aquesta funció, no afecta fora  
    System.out.println("Valor d'en la funció: " + a); // 'a' val 20  
}
```

Eixida

```
run:  
Valor inicial de a: 10  
Valor de a en la funció: 20  
Valor final de a: 10
```



## 5. PARÀMETRES: PAS PER VALOR I PER REFERÈNCIA

### DEFINICIÓ



GENERALITAT  
VALENCIANA

ceedcv  
CENTRE ESPECÍFIC  
D'EDUCACIÓ A DISTÀNCIA DE  
LA COMUNITAT VALENCIANA

- **Paràmetres de tipus compost (pas per referències)** : Com a les variables de tipus String (están compostes de variables de tipus simple char), els Arrays (compostos de múltiples valors del mateix tipus), etc. En aquest cas no es copia el valor de la variable, sinó que **se li passa a la funció una referència a la variable original (un punter)**. Per això **des de la funció s'accedeix directament a la variable** que es troba fora. Els canvis que fem dins de la funció afectaran a la variable externa.

```
// Summa x a tots els elements del vector v
public static void summa_x_al_vector(int v[], int x) {
    for (int i = 0; i < v.length; i++)
        v[i] = v[i] + x;
}

public static void main(String[] args) {
    int v[] = {0, 1, 2, 3};
    System.out.println("Vector abans: " + Arrays.toString(v));
    summa_x_al_vector(v, 10);
    System.out.println("Vector després: " + Arrays.toString(v));
}
```

Eixida

```
run:
Vector antes: [0, 1, 2, 3]
Vector después: [10, 11, 12, 13]
```

## 5. PARÀMETRES: PAS PER VALOR I PER REFERÈNCIA

### DEFINICIÓ



**IMPORTANT:** Com un paràmetre de tipus compost és una referència a la variable que està fora d'ella, si se li assigna un altre valor es perd la referència i ja no es pot accedir al valor original fora de la funció. **Encara que Java permet fer-ho, no s'aconsella fer-ho.**

```
// Assignem a x un nou vector, per la qual cosa x deixarà d'apuntar al vector original.  
// El vector original no canvia! Simplement ja no podem accedir a ell des de x  
// perquè s'ha perdut la referència a aquesta variable.
```

```
public static void funcion1(int x[]) {  
    x = new int[10];           // x apuntarà a un nou vector, l'original queda intacte  
                                // el que fem ací amb x no afectarà el vector original  
}
```

```
// El mateix succeeix en aquest exemple, perdem la referència al String original
```

```
public static void funcion2(String x) {  
    x = "Hola";                // x apuntarà a un nou String, l'original queda intacte  
                                // el que fem ací amb x no afectarà el String original  
}
```

**NO S'ACONSELLA FER ESTE TIPUS DE COSES.**

## 6. DEVOLUCIÓ D'UN VALOR

### DEFINICIÓ



GENERALITAT  
VALENCIANA

ceedcv  
CENTRE ESPECÍFIC  
D'EDUCACIÓ A DISTÀNCIA DE  
LA COMUNITAT VALENCIANA

Comando *return* ➡ Devolució d'un valor

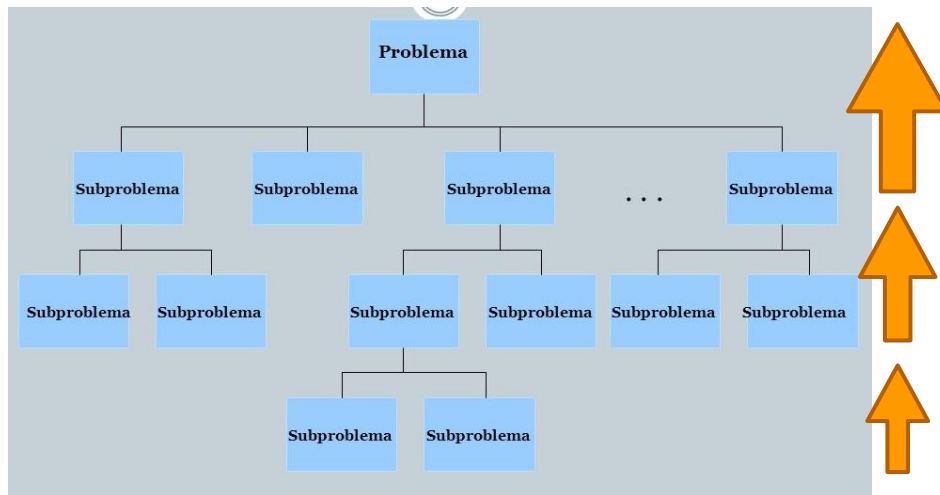
Els mètodes poden **retornar valors de tipus bàsic o primitiu** (int, double, boolean, etc.) i **també una referencia en el cas de tipus compost** (Strings, arrays, etc.).

## 7. RECURSIVITAT

### *Disseny descendent*

Disseny descendent, consisteix a dividir un programa en diferents subprogrames, i aquests subprogrames en altres subprogrames fins a arribar a un nivell de subprogrames que siguen fàcils de resoldre directament (també anomenats cas base).

La suma de totes les solucions dels subproblemes interns serà la solució del problema del qual provenen.



## 7. RECURSIVITAT

### DEFINICIÓ I TIPUS



GENERALITAT  
VALENCIANA



Un tipus d'exemple de solució utilitzant el disseny descendent és la recursivitat.

La recursivitat és la capacitat d'una funció o mètode per a cridar-se a si mateixa. Això permet dividir un problema gran en problemes més xicotets similars i resoldre'ls directament.

Hi ha diferents tipus de recursivitat:

- **Recursivitat directa:** Una funció es crida directament a si mateixa.
- **Recursivitat indirecta:** Una funció crida a una altra que al seu torn crida a la primera. Es produeix un cicle cridaes recursives entre ambdues.
- **Recursivitat lineal:** Cada cridada recursiva es fa sobre un subproblema diferent. Per exemple, en recórrer un arbre binari.
- **Recursivitat en cua:** La cridada recursiva és l'última instrucció de la funció. Això permet optimitzar l'ús de la pila.

## 7. RECURSIVITAT

### CASOS



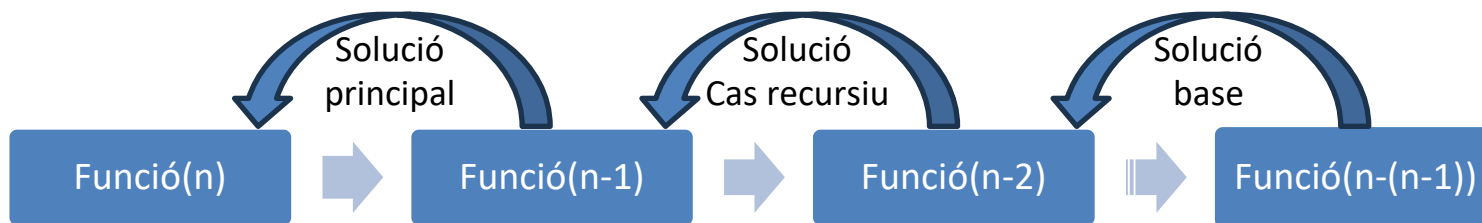
GENERALITAT  
VALENCIANA

ceedcv  
CENTRE ESPECÍFIC  
D'EDUCACIÓ A DISTÀNCIA DE  
LA COMUNITAT VALENCIANA

Cas base: Cas més simple que es pot resoldre directament i permet parar les recursions.

Casos recursius: Crida a si mateixa per a reduir el problema en una instància més menuda i simple.

Imaginem una funció que hauria de processar un conjunt de dades. Mitjançant la recursivitat, aquesta funció es crida repetidament amb subconjunts més petits de dades fins que s'arriba a casos base, que es poden resoldre directament:



## 7. RECURSIVITAT

### RECURSIVITAT VS ITERACIÓ



GENERALITAT  
VALENCIANA



La recursivitat i la iteració (usant bucles com while o for) són dues formes diferents de resoldre el mateix problema.

Exemple: Suma dels números de l'1 al n iterativa

```
public static int suma(int n) {  
    int suma = 0;  
    for (int i = 1; i <= n; i++) {  
        suma += i;  
    }  
}
```

La versió iterativa utilitza un bucle i una variable acumuladora (suma) per a anar guardant el resultat parcial.

La iteració sol ser més eficient quant a ús de memòria i velocitat d'execució, però hi ha casos que la seua implementació pot ser tan complexa que faça que el codi siga poc clar i eficient.

## 7. RECURSIVITAT

### RECURSIVITAT VS ITERACIÓ



GENERALITAT  
VALENCIANA



Exemple: Suma dels números de l'1 al n iterativa

```
public static int suma(int n) {  
    if (n == 1) { //Cas base  
        return 1;  
    } else { //Casos recursius  
        return n + suma(n-1);  
    }  
}
```

La versió recursiva es basa en la descomposició del problema en problemes més xicotets, fins a arribar a un cas base senzill de resoldre (quan n és 1).

La recursivitat pot ser més elegant i fàcil d'entendre, però consumeix més recursos ja que es van acumulant anomenades en la pila.





# EXEMPLES DE RECURSIVITAT





# EXERCICIS PROPOSATS

