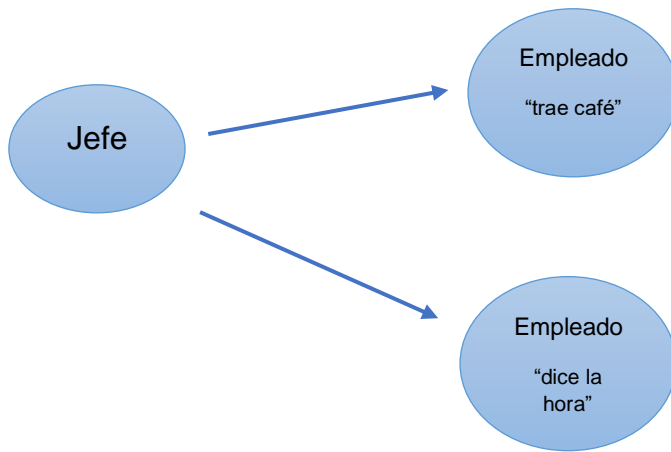


Funciones

Introducción



Imaginemos el jefe de una empresa.

Este jefe tiene dos empleados que cumplen funciones distintas. Por un lado, hay un empleado que le **trae el café** al jefe y hay otro empleado que le **dice que hora es**.

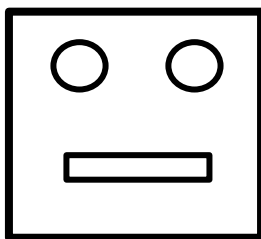
Estos empleados no cumplen con su tarea cuando quieran. Solamente las cumplen cuando el jefe da la orden de ejecutar su tarea.

Esto es lo que llamamos delegar funciones.

Estas funciones son ejecutadas solo cuando se dé la orden.

¿qué es una función en programación?

La definición que se suele dar es: un bloque de código que cumple una **tarea concreta**.



Ese **bloque** de código lo podemos ver como un **robot** que hace lo que le digamos.

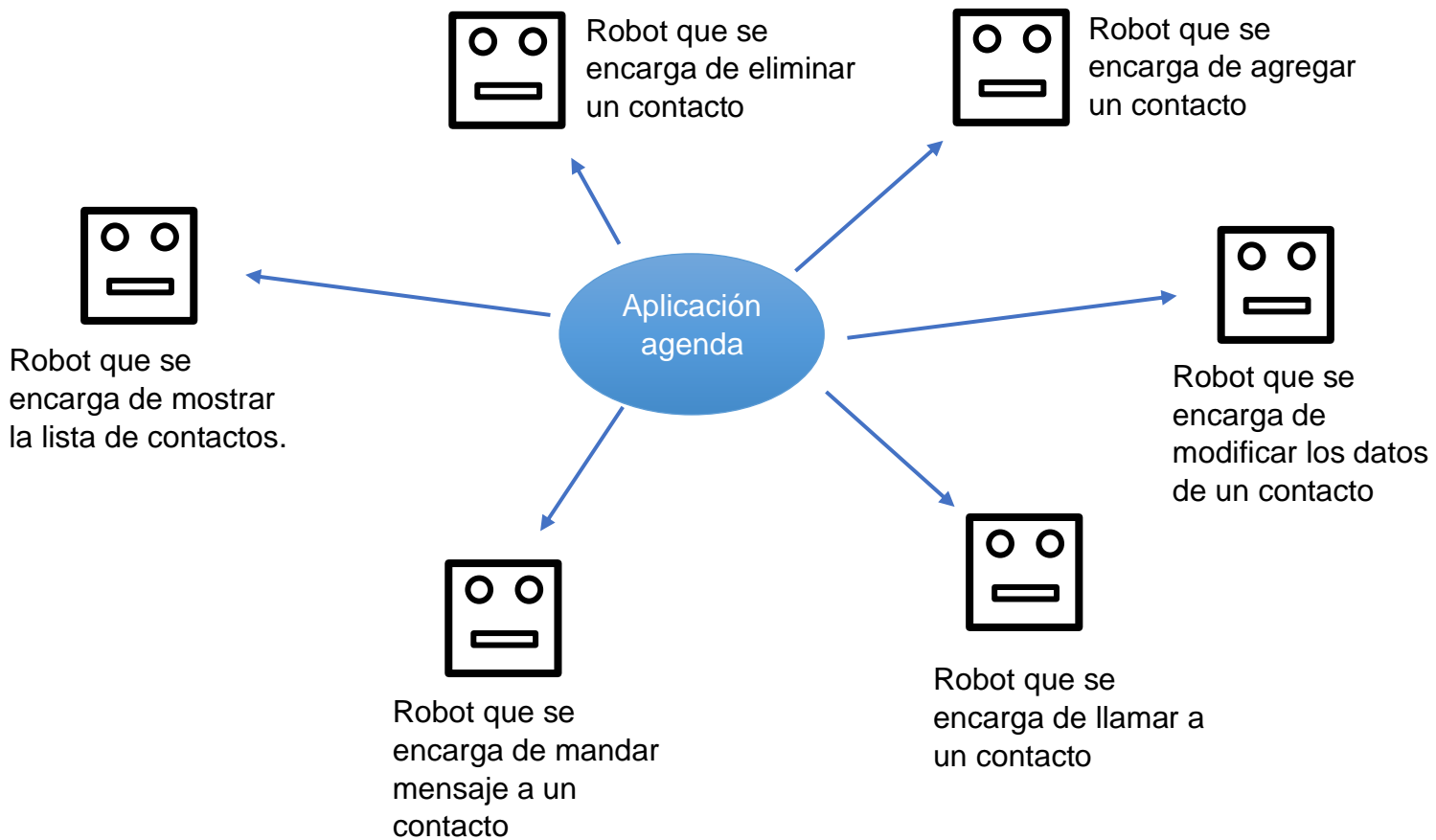
Es como un robot que **tiene código dentro** y que ejecuta ese código cuando le demos la orden.



Este robot solo va a ejecutar su función cuando yo se lo indique.

Ejemplos de tareas concretas

Supongamos que estamos programando una agenda telefónica. Hay muchas tareas concretas que podemos delegar a algunos robots.



- **Agregar contacto:** el robot encargado de esa tarea pedirá primero los datos del contacto (nombre, apellido, num_tel). Luego va a recorrer el array de contactos para verificar si ya existe un contacto con esos datos. De ser así, el robot notificará al usuario diciéndole que ya existe un contacto con esos datos. De lo contrario, simplemente va a agregar el contacto nuevo al array.
- **Eliminar contacto:** el robot con esa tarea pedirá el nombre y apellido del contacto a eliminar. Luego de eso, recorre el array buscando al contacto que tenga los datos ingresados. Si no lo encuentra, se le avisa al usuario que no existe un contacto con los datos que ingresó. De lo contrario, si se encuentra, simplemente se eliminará del array.

- **Modificar contacto:** pide nombre y apellido del contacto a modificar, luego lo busca en el array. Si no lo encuentra, avisará al usuario que no existe un contacto con esos datos. De lo contrario, si lo encuentra, pedirá los nuevos datos del usuario (nuevo nombre, nuevo apellido, nuevo num_tel).
- **Mostrar lista:** Cuando se de la orden, el robot primero lo que hace es preguntar al usuario si quiere la lista ordenada alfabéticamente por nombre. Si el usuario quiere la lista ordenada, el robot ordenará la lista alfabéticamente por los nombres, luego de eso mostrará la lista con los datos de todos los contactos. Si el usuario no quiere la lista ordenada, el programa mostrará la lista con los contactos por orden de llegada.
- **Mandar mensaje:** Lo primero que hará es preguntar nombre y apellido del contacto. Luego pedirá que mensaje enviarle. Por último, buscará al contacto en el array y si lo encuentra, le enviará el mensaje. Si no se encuentra, se le notifica al usuario que el contacto no existe.
- **Llamar:** El robot pedirá nombre y apellido del contacto, luego lo buscará en el array y lo llama. Si el contacto no existe, se le notifica al usuario.

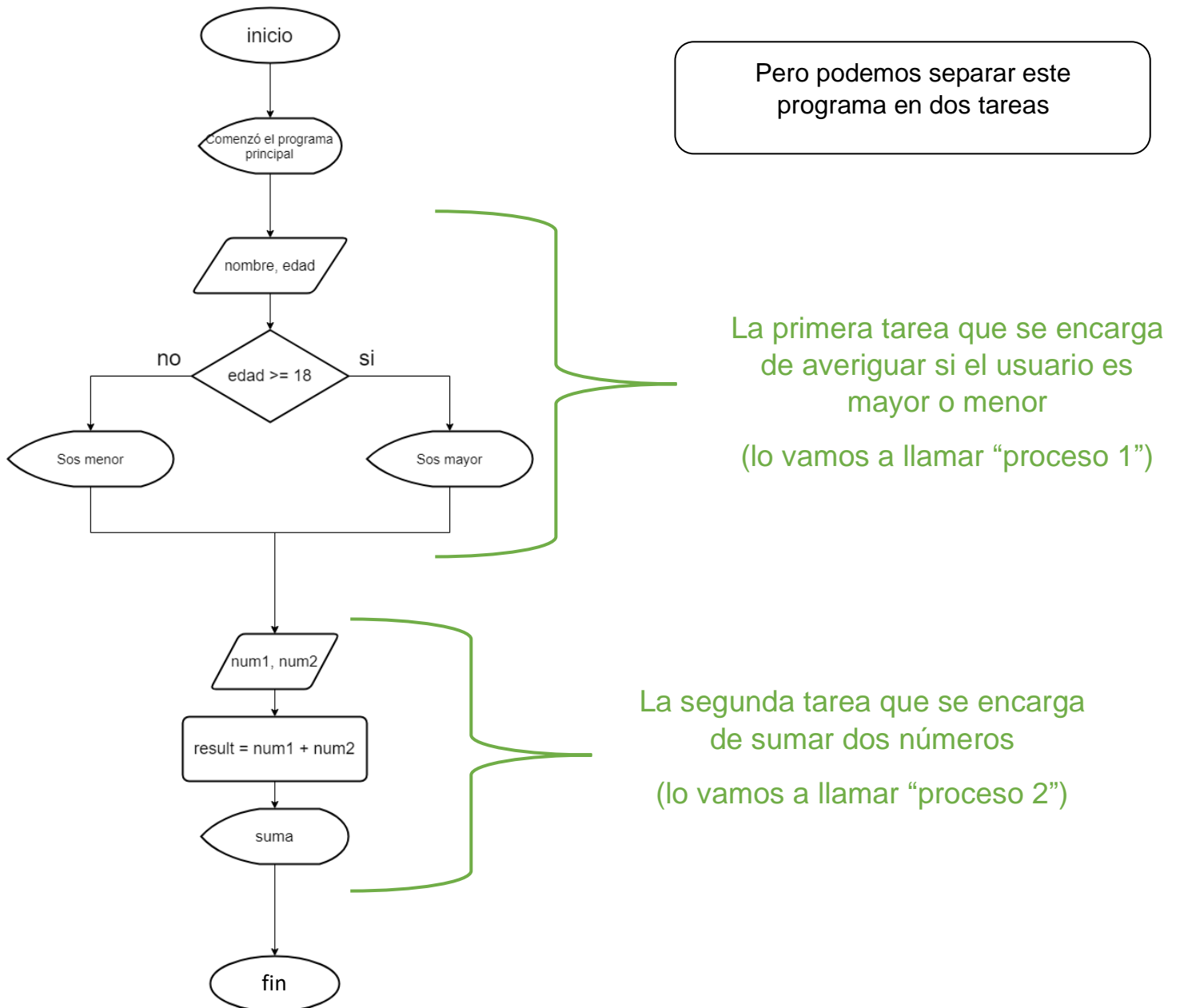
Divide y vencerás

La idea de las funciones es de **dividir** un problema grande (la agenda) en subproblemas pequeños (agregar contacto, modificar, eliminar, etc).

Ejemplo con diagrama de flujo

Ejercicio: Hacer un programa que haga dos cosas. Primero pedirá la edad del usuario y le dirá si es mayor o menor. Lo segundo que hace es pedir dos números y mostrar la suma.

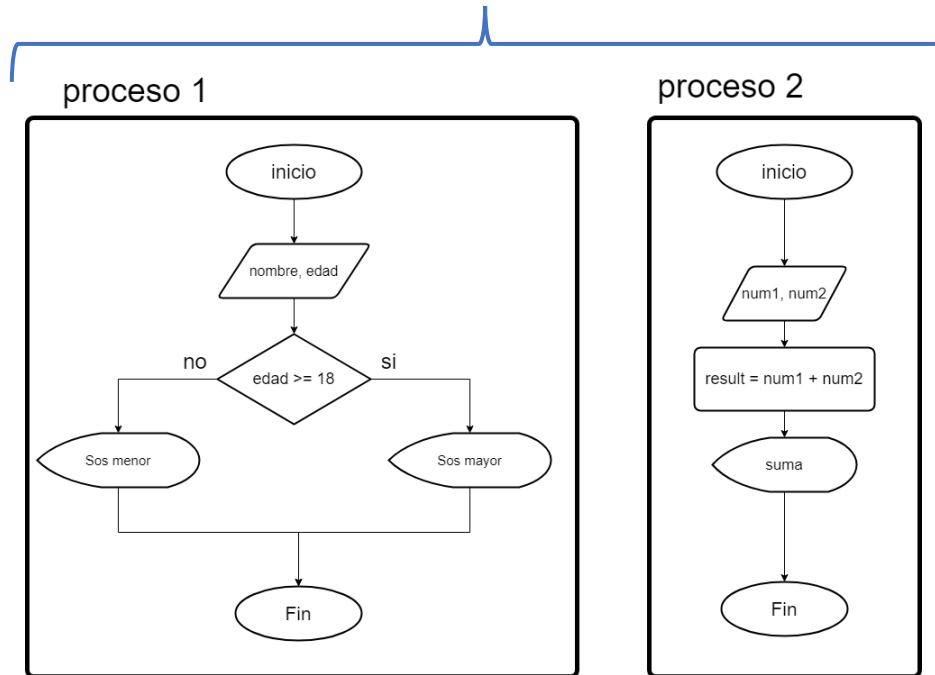
Vemos que, si programamos esto, el flujo del programa es uno solo (y es grande)



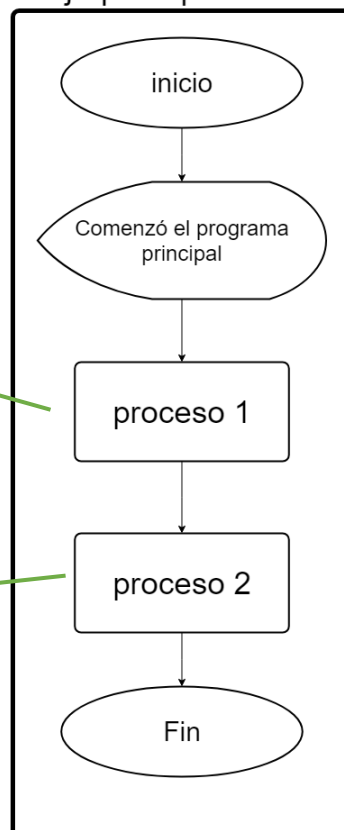
Vemos que son dos tareas distintas. No tienen nada en común.

Solución con funciones

Vemos que tenemos las tareas separadas en funciones (o robots) aparte



Flujo principal



Con las funciones podemos dejar el flujo principal más pequeño

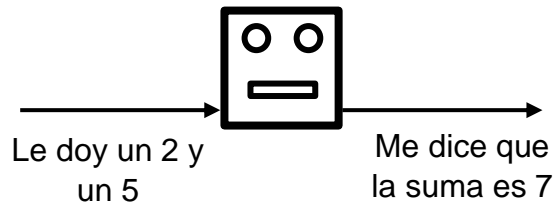
Acá simplemente doy la orden al proceso 1 de ejecutar su código interior

Lo mismo ocurre con el proceso 2

Más ejemplos de tareas

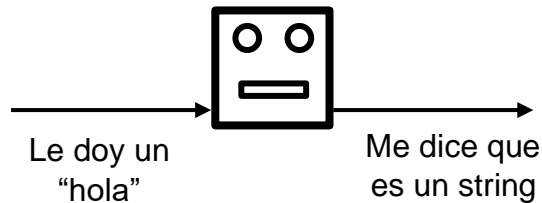
1

Robot que me dice la
suma de dos números



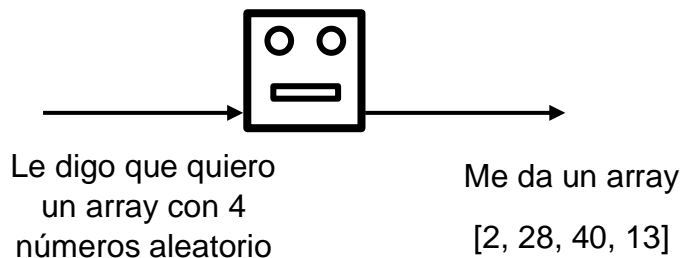
2

Robot que me dice de
qué tipo es un dato



3

Robot que se encarga
de crear un array con
números aleatorios



Punto importante: No me importa como hagan su trabajo. No me importa si el tercer robot pone números enteros o flotantes (por ejemplo). Lo que me importa es que cumplan con su trabajo.

Si tengo un robot que se encarga de ordenar los array de menor a mayor, no me importa como lo haga, no me importa qué métodos usa, me importa que me ordene el array.

Funciones predefinidas

Nosotros ya venimos trabajando con funciones desde que empezamos.

<code>alert(mensaje)</code>	→	Función que tiene código dentro. Esa función se encarga de abrir una ventana emergente y mostrar el mensaje que nosotros le demos
<code>parseInt(variable)</code>	→	Función que agarra el valor que le pasemos por los paréntesis y nos da la misma variable convertida a número int
<code>push(elem)</code>	→	Función que agarra el valor que le pasemos por los paréntesis y lo agrega en la última posición del array
<code>pop()</code>	→	Función que no espera ningún valor dentro de los paréntesis. Solamente agarra el último elemento del array, lo elimina y me lo da.

Estas son **funciones ya programadas** por los creadores de javascript y las venimos usando desde que empezamos. De no existir el `alert()`, nosotros programaríamos nuestro propio `alert()`. En este caso, como ya existen estas funciones, no nos interesa saber qué código tienen, o que método usan, no nos interesa la cantidad de líneas que tengan dentro, solo nos interesa usarlas.

A nosotros como programadores nos interesa crear nuestras propias funciones por sus beneficios:

- Reutilización de código (si tengo una función de 8 líneas y quiero que ejecute su función 4 veces, en lugar de hacer 32 líneas de código, lo que hacemos es dar la orden 4 veces y listo).
- Código más ordenado. El flujo principal queda más chico.
- Cuando haya un error, podemos detectarlo fácil yendo a la función.

Funciones en javascript

Estructura de una función

Indicamos que es una función (es como el let de las variables)

Nombre que le ponemos a la función

Dentro de los paréntesis se definen los parámetros que recibe (luego se explica que es)

```
function nombre(){  
  //Código que ejecuta  
  //Código  
  //Código  
  //Código  
}
```

Cuerpo de la función

Es el bloque de código que va a ejecutar. La tarea que ejecuta el robot.

Se encierra con las llaves

Ejemplo

```
function saludar(){  
  console.log("hola, soy una función");  
}
```

Función llamada saludar()

Que ejecuta un console.log() que dice "hola".

Básicamente hace lo que dice su nombre.

Nombres de las funciones

Siempre vamos a llamarla la función con el **verbo** de la acción que hace.

Ejemplos de nombres

cerrar()

abrirCuenta()

crearUsuario()

normalizar()

obtenerID()

definirEstado()

Son todos verbos

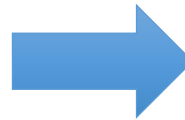
Llamar a la función

Para que la función ejecute su código, hay que darle la orden (llamarla).

Así como llamábamos a la función `push()` para agregar un elemento al array, debemos llamar a la función `saludar()` para que salude. Simplemente se escribe el nombre de la función con los paréntesis.

```
//Creación de la función
function saludar(){
  console.log("hola, soy una función");
}

//Llamada a la función (doy la orden)
saludar();
```



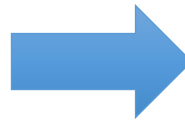
```
top
hola, soy una función
> |
```



```
//Creación de la función
function saludar(){
  console.log("hola, soy una función");
}

saludar();
saludar();
saludar();
```

Si quisiera que salude tres veces, basta con llamar a la función tres veces



```
hola, soy una función
hola, soy una función
hola, soy una función
>
```

Una función nunca se ejecutará hasta que se la llame.

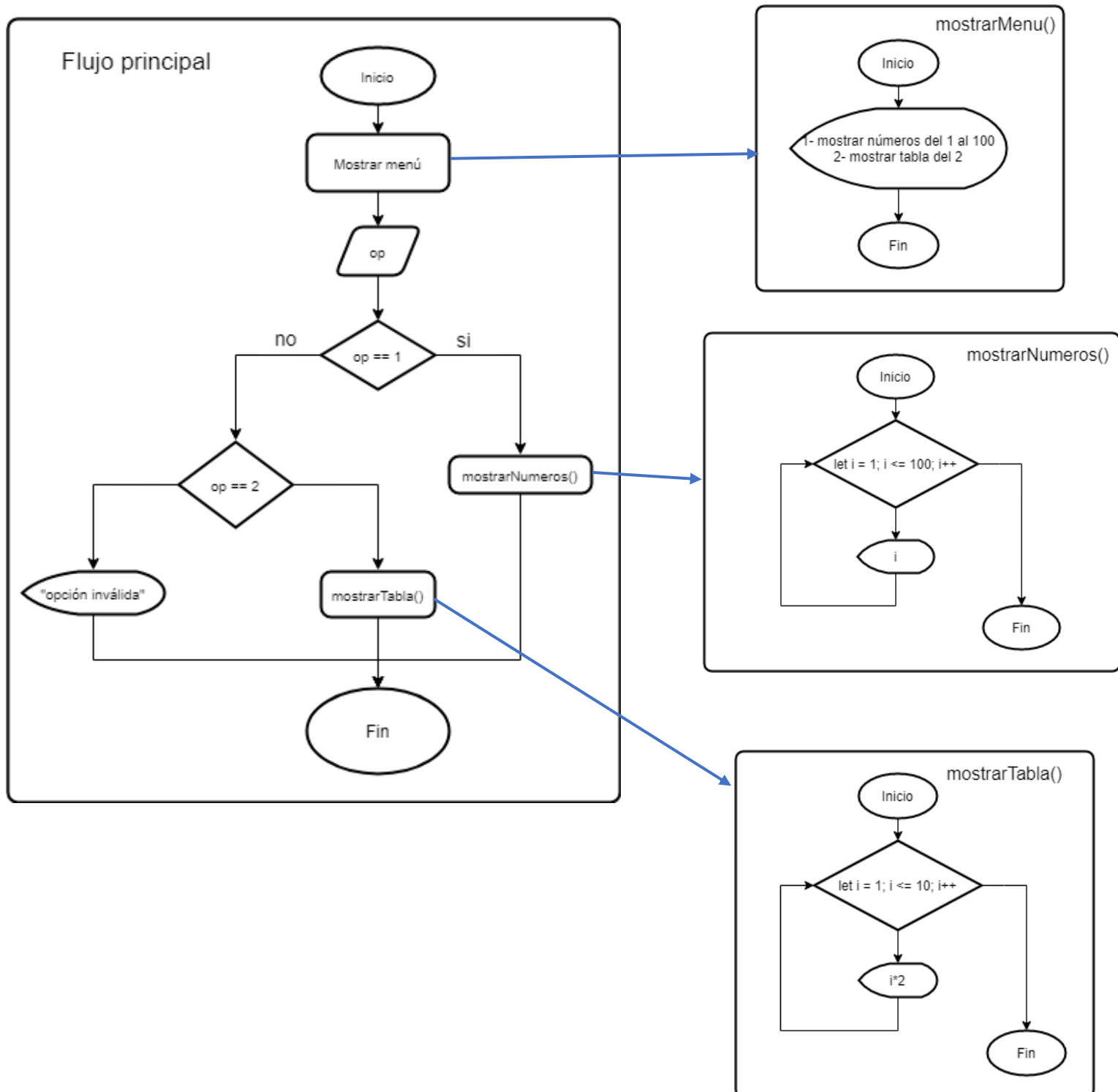
Al llamar a una función, le estoy dando la orden de que ejecute su código interno y lo va a ejecutar según las veces que la llame (el primer ejemplo saludó una sola vez porque la llame una sola vez. El segundo ejemplo ejecutó el código tres veces porque la llamé tres veces).

Llamar  dar la orden

Ejemplo 1

Un programa que me pregunte si quiero ver los números del 1 al 100 o ver la tabla del 2. Hará lo que yo elija.

- Menú
 - Números del 1 al 100
 - Tabla del 2
- Tareas que podemos separar



Código

```
//Funciones
function mostrarMenu(){
    alert("1- mostrar números del 1 al 100\n2- mostrar tabla del 2");
}

function mostrarNumeros(){
    for(let i = 1; i <= 100; i++){
        document.write(i + "<br>");
    }
}

function mostrarTabla(){
    for(let i = 1; i <= 10; i++){
        document.write((i*2) + "<br>");
    }
}

//Comienza el flujo principal
mostrarMenu();
let op = parseInt(prompt("¿cuál opción quieres?"));

switch(op){
    case 1:
        mostrarNumeros();
        break;
    case 2:
        mostrarTabla();
        break;
    default:
        alert("opción inválida");
}
```

Con solo leer las
funciones, ya
sabemos que hace

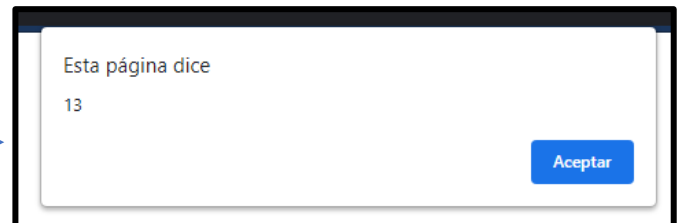
Vemos que el flujo
principal queda más
chico

Otro ejemplo

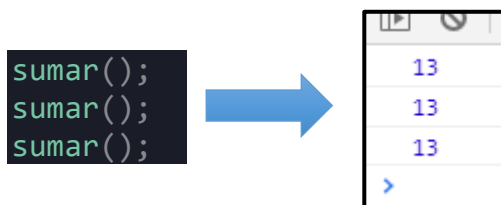
Una función que sume dos números.

```
function sumar(){  
  let num1 = 6;  
  let num2 = 7;  
  
  console.log(num1+num2);  
  
}
```

`sumar();` →



En este caso estamos haciendo una función con los números ya puestos (6 y 7), por lo que, si llamamos a la función tres veces, **siempre sumará los mismos números**.



Como se ve, está sumando siempre los mismos números. ¿Cómo hago si en cada llamada que se hace en el flujo principal quiero enviarle distintos números?

Para eso le pasamos los números como **argumentos**.

Parámetros y argumentos

- Los argumentos son los **valores que le pasamos dentro de los paréntesis** para que trabaje con ellos.

Por ejemplo, cuando usamos el `parseInt("2")` lo que hacemos es darle un valor dentro de los paréntesis y lo que hace es agarrar ese valor y usarlo dentro del código interno para convertirlo a número int.

Otro ejemplo es el `array.splice()` que recibe dos argumentos para eliminar un elemento (recordemos que el primer argumento es la posición del elemento a eliminar y el segundo argumento es la cantidad de elementos a eliminar)

`splice(3, 1)`

La función recibe estos argumentos y los usa en el código interno

- Los parámetros son las variables en donde se guardan los valores pasados como argumentos. Se define dentro de los paréntesis de la estructura.

indicamos que la función recibe dos parámetros separados por coma (puedo nombrarlas como quiera)

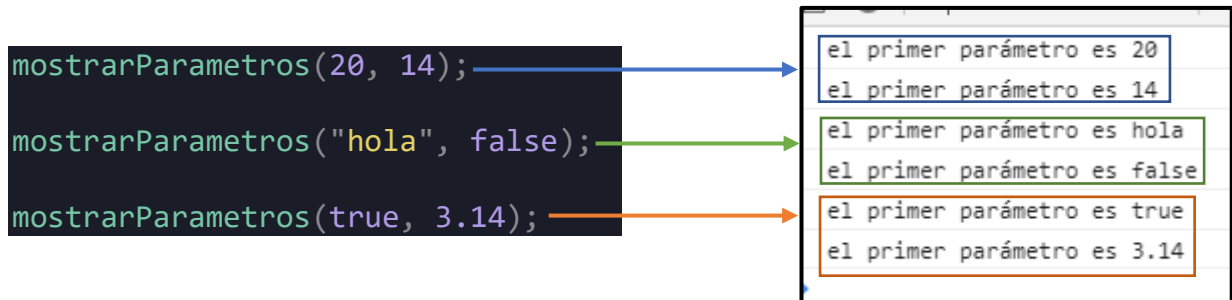
Miremos esta función

```
function mostrarParametros(param1, param2){  
  console.log("el primer parámetro es " + param1);  
  console.log("el primer parámetro es " + param2);  
}
```

Puedo usarlas como variables y mostrar su contenido

En definitiva, esta función espera que se le pasen dos parámetros al ser llamada.

De esta manera yo le puedo pasar distintos valores como argumento.



En cada llamada recibe valores diferentes.

```
mostrarParametros( 20 , 14 );
```

```
mostrarParametros( param1 , param2 ){  
}
```

Se guardan en los parámetros correspondientes

```
mostrarParametros( "hola" , false );
```

```
mostrarParametros( param1 , param2 ){  
}
```

Sigamos con el ejemplo

Una función que **reciba** dos números y que los sume.

Cuando llame a la función,
va a recibir dos números

```
function sumar(num1, num2){  
  console.log(num1 + " + " + num2 + " = " + (num1+num2));  
}
```

Luego al llamarla le paso valores diferentes.

The diagram shows three function calls on the left, each with a blue arrow pointing to a corresponding result on the right. The function calls are: `sumar(5, 7);`, `sumar(1, 9);`, and `sumar(2, 2);`. The results are: `5 + 7 = 12`, `1 + 9 = 10`, and `2 + 2 = 4`.

<code>sumar(5, 7);</code>	<code>5 + 7 = 12</code>
<code>sumar(1, 9);</code>	<code>1 + 9 = 10</code>
<code>sumar(2, 2);</code>	<code>2 + 2 = 4</code>

Argumento



Es el valor que nosotros le
pasamos cuando llamamos a
la función

Parámetro



Es la variable que definimos
dentro de los paréntesis de la
estructura

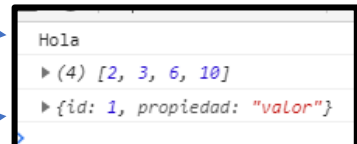
Podemos pasar distintas estructuras de datos

```
//Función que muestra en la consola lo que recibe por parámetro
function mostrar(param){
  console.log(param);
}

let variable = "Hola";
mostrar(variable);

const arr = [2, 3, 6, 10];
mostrar(arr);

const obj = {id: 1, propiedad: "valor"};
mostrar(obj);
```



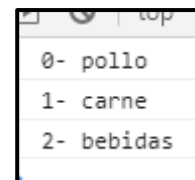
```
Hola
▶ (4) [2, 3, 6, 10]
▶ {id: 1, propiedad: "valor"}
```

Ejemplo con arrays

```
function mostrarLista(arr){
  for(let i = 0; i < arr.length; i++){
    console.log(i + " - " + arr[i]);
  }
}

const compras = ["pollo", "carne", "bebidas"];
mostrarLista(compras);
```

Recibe el array y lo recorre

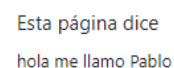


```
0- pollo
1- carne
2- bebidas
```

Ejemplo con objetos

```
function presentarse(p){
  alert("hola me llamo " + p.nombre);
}

const pablo = {nombre: "Pablo"};
presentarse(pablo);
```



Esta página dice
hola me llamo Pablo

Aceptar

Funciones con retorno de valor

Muchas veces lo que buscamos es que una función haga una tarea, pero nos devuelva un valor que podemos guardar en una variable.

Ejemplos:

```
let pos = arr.indexOf(elem);
```

Devuelve la posición en la que se encuentra el elemento

```
let num = parseInt("5");
```

Devuelve el valor que le di, pero convertido a int

```
let existe = arr.includes(elem);
```

Devuelve true si se encuentra el elemento o false si no se encuentra

```
let letra = "hola".charAt(2);
```

Devuelve la letra que se encuentra en la posición 2 (la 'l')

Sigamos con el ejercicio de la función sumar.

Para que devuelva la suma, hay que retornarla con la instrucción return.

```
function sumar(num1, num2){  
  //hago la suma  
  let resultado = num1+num2;  
  
  //devuelvo la suma  
  return resultado;  
}
```

Return significa retornar (o devolver). Básicamente le digo que devuelva el resultado

```
//llamo a la función y me devuelve los resultados
```

```
let suma1 = sumar(5, 7);  
let suma2 = sumar(1, 9);  
let suma3 = sumar(2, 2);
```

En cada llamada usa los valores, los suma y me da los resultados para guardarlos en variables

```
//muestro los resultados
```

```
document.write("recibí los resultados:<br>");  
document.write(suma1 + "<br>");  
document.write(suma2 + "<br>");  
document.write(suma3 + "<br>");
```

recibí los resultados:

12
10
4

Esto me da la posibilidad de seguir trabajando en el flujo principal.

Otro ejemplo

Un programa que pida un número N, luego hace la sumatoria desde 1 hasta N. Si el resultado es mayor o igual a 10, avisarlo con un alert(). Si es menor, comprobar si es par o impar.

Tareas detectadas:

- pedirNumero()
- hacerSumatoria()
- comprobarMenorMayor()
- comprobarParImpar()

Veamos la solución sin usar el return:

```
function pedirNumero(){
  let n = parseInt(prompt("ingresar número"));

  hacerSumatoria(n);
}

function hacerSumatoria(n){
  let suma = 0;
  for(let i = 1; i <= n; i++){
    suma += i;
  }

  comprobarMenorMayor(suma);
}

function comprobarMenorMayor(suma){
  if(suma >= 10){
    alert("es mayor");
  }else{
    comprobarParImpar(suma);
  }
}

function comprobarParImpar(suma){
  if( (suma%2) == 0){
    alert("es menor y par");
  }else{
    alert("es menor E impar");
  }
}

pedirNumero();
```

El robot encargado de pedir el número llama al robot que hace la sumatoria y le da la orden.

El robot de pedirNumero() no debe llamar al otro robot porque esa no es su función.

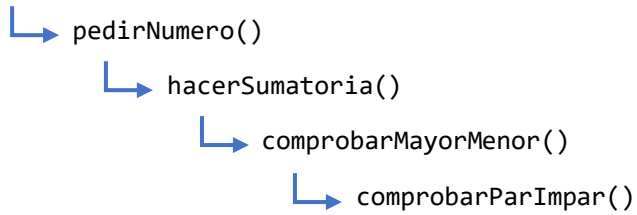
Lo mismo pasa con las demás funciones. No deben llamar a otras funciones

En este código el flujo principal da la orden al primer robot de pedir un número, luego ese robot da la orden al segundo robot de hacer la sumatoria y le da el número, después ese robot llama al tercero para que compruebe si es mayor o menor y por último, el tercer robot llama al cuarto.

Las funciones solo deben limitarse a cumplir su función. La función de `pedirNumero()` debe limitarse a pedir el número y devolverlo al flujo principal, no tiene por que llamar a la función de `hacerSumatoria()`, ya que no tiene nada que ver con su tarea principal, que es pedir un número.

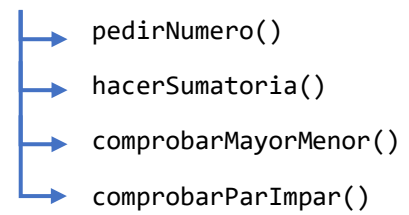
Lo que nos está pasando

flujo principal



Lo que se desea

flujo principal



En este ejercicio queremos que el flujo principal de la orden a `pedirNumero()` y que este le devuelva el número que pidio. Luego el flujo principal llamará a `hacerSumatoria()`, le da el número y este le devuelve la suma

Solución con return

```
function pedirNumero(){  
  let n = parseInt(prompt("ingresar número"));  
  
  return n;  
}
```

La primera función pide el número y luego lo devuelve (es decir, no se lo queda)

```
function hacerSumatoria(n){  
  let suma = 0;  
  for(let i = 1; i <= n; i++){  
    suma += i;  
  }  
  
  return suma;  
}
```

```
function comprobarMenorMayor(suma){  
  if(suma >= 10){  
    return true;  
  }else{  
    return false;  
  }  
}
```

Lo mismo pasa con el resto de funciones

```
function comprobarParImpar(suma){  
  if( (suma%2) == 0){  
    alert("es menor y par");  
  }else{  
    alert("es menor E impar");  
  }  
}
```

La última función no devuelve nada, solo avisará con alert()

Es una función sin retorno de valor

Flujo principal

```
//comienza el flujo principal  
let num = pedirNumero();  
let sumatoria = hacerSumatoria(num);  
let mayor = comprobarMenorMayor(sumatoria);  
  
if(mayor){  
  alert("es mayor");  
}else{  
  comprobarParImpar(sumatoria);  
}
```

Vemos que el flujo principal es el único que llama a las funciones. Primero llama a `pedirNumero()` y guarda el número devuelto en una variable. Luego llama a `hacerSumatoria()` y le da ese número y espera a que le de la sumatoria para guardarla en otra variable

Así sucesivamente

OJO

Van a haber ciertos casos en los que una función requiera de otra función para seguir con su tarea.

Por ejemplo, si estamos programando una agenda telefónica y hay una función llamada `mostrarLista(lista)`, esa función recibe como argumento la lista a mostrar y en su interior llamará a otra función llamada `ordenarLista(lista)`. Esto es válido en este caso porque la función `mostrarLista()` requiere de otra función para seguir con su tarea principal.

Esto es algo que se aprende practicando con ejercicios.