Algoritmos: 5

Funciones

El concepto de función en programación se fundamenta en el concepto de función matemática.

Una función, desde el punto de vista de la programación, se define como un proceso que recibe valores de entrada (llamados parámetros) y el cual retorna un resultado (valor).

Comparación entre función matemática y función en programación

- **Entrada (argumento)**: Ambas funciones, tanto en JavaScript como la cuadrática en matemáticas, toman una entrada x.

- **Proceso (cálculo)**: En JavaScript, realizamos operaciones dentro de la función como x*x. En la función cuadrática, realizamos una serie de operaciones algebraicas sobre x.

- **Salida (resultado)**: Ambas devuelven un valor basado en el cálculo. En el ejemplo de JavaScript, el valor es 9 para x = 3, mientras que en la función cuadrática, el valor es 28 para x = 3

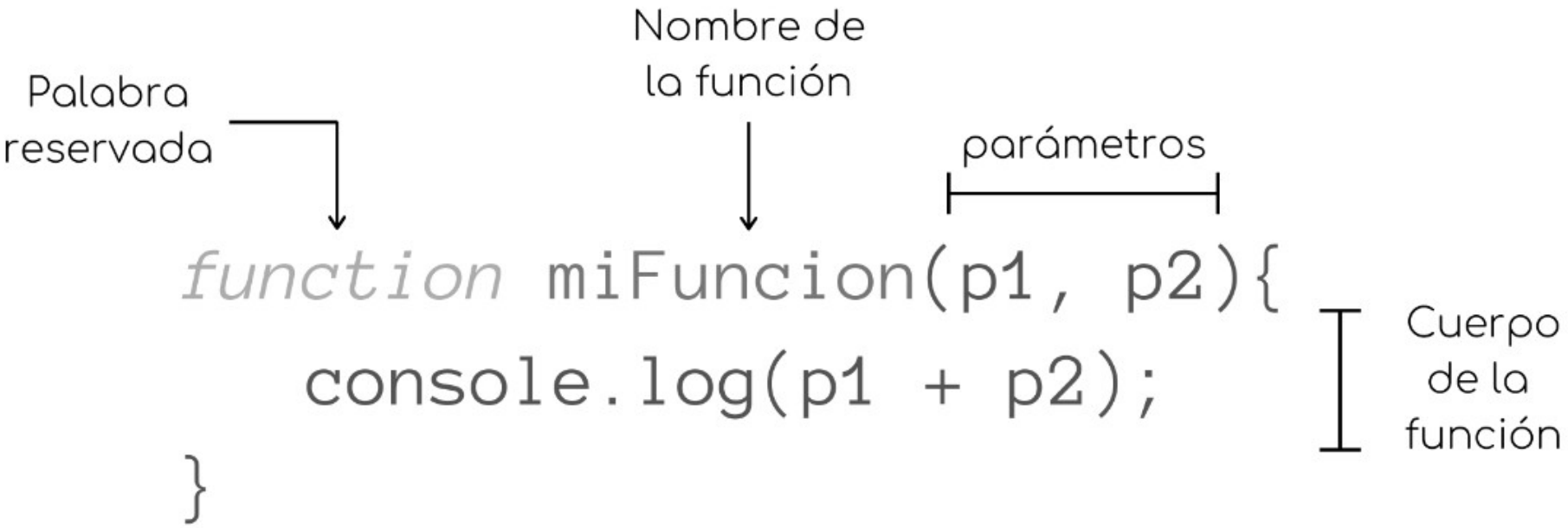
```
function cuadrado(x) {
 return x * x;
}
console.log(cuadrado(3)); // Salida: 9
```

$$f(x) = 2x^2 + 3x + 1$$

$$f(3) = 2(3)^2 + 3(3) + 1$$

$$f(3) = 28$$

Estructura de una función



Tipos de funciones

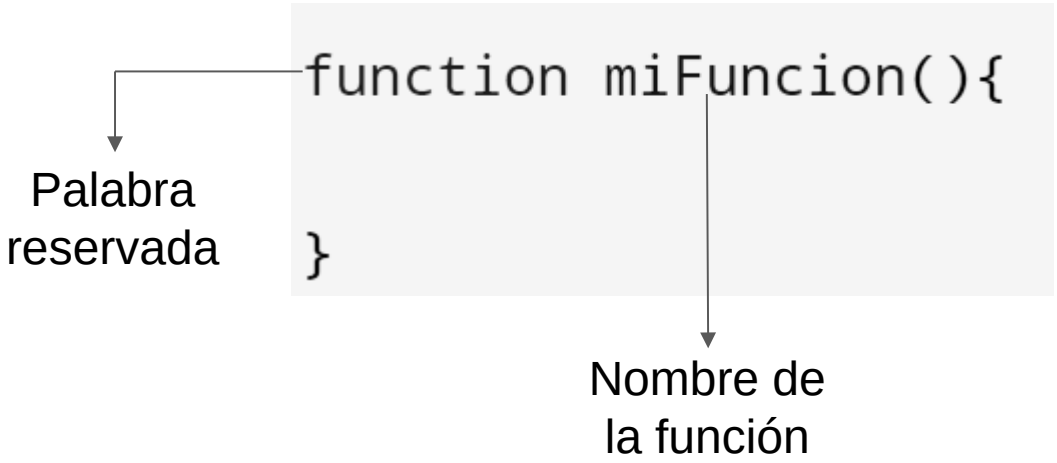
En javascript al momento de crear una función hay partes de esta las cuales no son obligatorias para que la dicha funcione de manera correcta.

¿Cuales son dichas partes?

Las partes que podemos omitir son dos (2), estas son los parámetros de entrada y el retorno de la función.

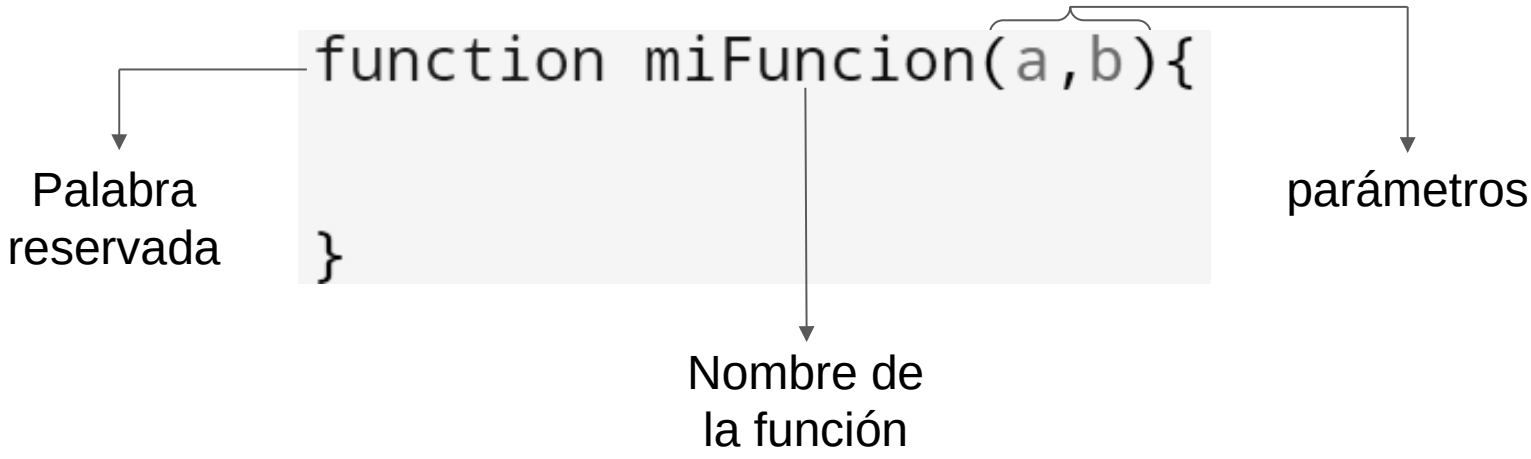
Para más claridad podemos ver la siguiente sintaxis.

Función sin parámetros ni retorno



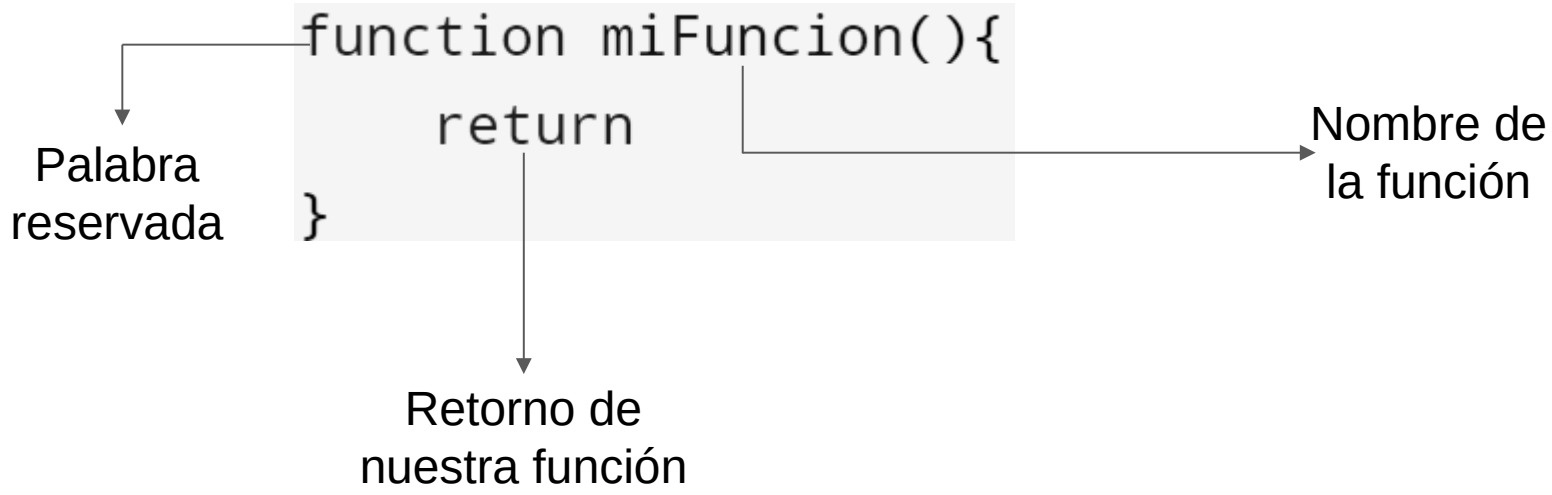
En este ejemplo declaramos una función la cual no posee parámetros y tampoco nos genera algún retorno, hay que tener en cuenta que a pesar que nuestra función no recibe parámetros, esta debe de tener () al final de su nombre para que se reconozca como una función, de lo contrario nos generará un error.

Función con parámetros sin retorno



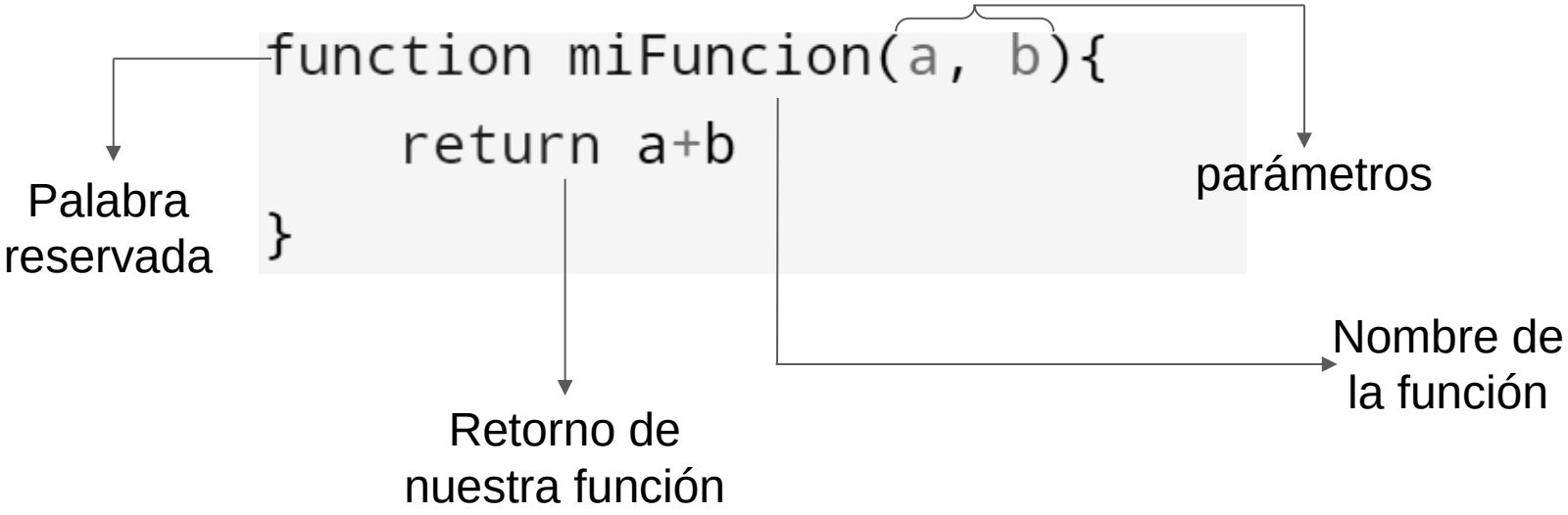
En este ejemplo creamos una función la cual tiene como parámetros a y b, dichos parámetros pueden ser usados dentro de nuestra función a nuestro gusto.

Función sin parámetros con retorno



En este ejemplo creamos una función la cual no tiene parametros pero si genera un retorno, este retorno puede ser cualquier tipo.(String, int, boolean char, etc…)

Función con parámetros y retorno



En este ejemplo creamos una función la cual tiene como parámetros a y b y genera un retorno que es la suma de a y b.

Llamado de funciones

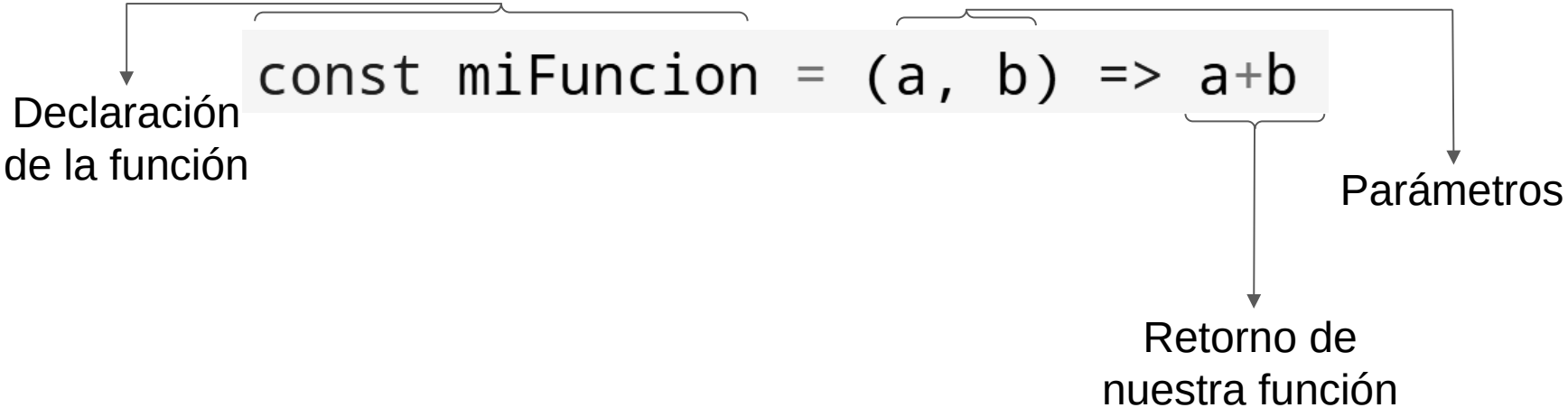
Las funciones son partes de código dentro de un programa, que se pueden reciclar o invocar (ejecutar) desde cualquier parte del algoritmo, es decir, desde otra función, desde la misma función, cuantas veces sea necesario.
Para invocar nuestra función debemos escribir el nombre de la función que queremos invocar y proporcionar los argumentos(los valores que reemplazarán a los parámetros en el cuerpo de la función) que esta requiere.

```
function miFuncion(a, b){
    return a+b
}

let result = miFuncion(2, 5)
```

Funciones flechas

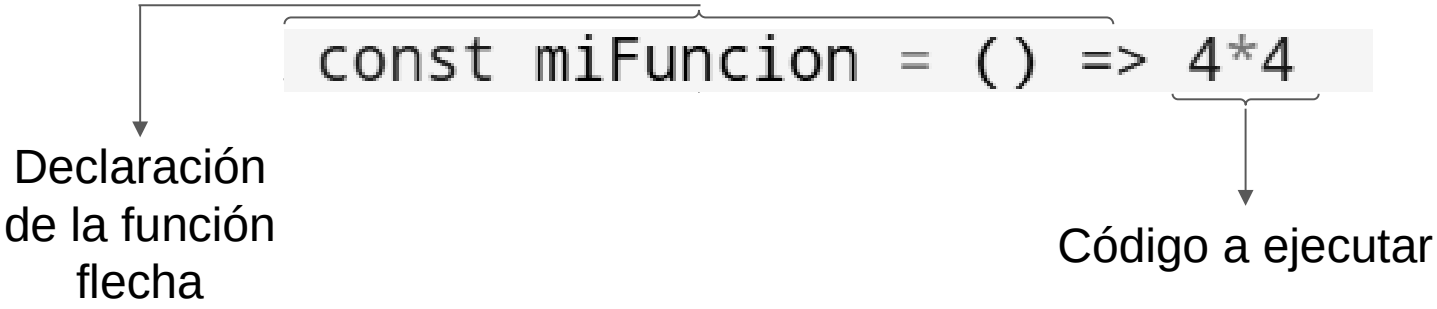
Una función flecha es una forma más concisa de escribir funciones y tiene varias ventajas, como una sintaxis más simple. Si el código que contiene nuestra función no supera más de una instrucción lo podemos dejar expresado sin necesidad de las llaves ni retorno explícito.



Tipos de funciones flecha

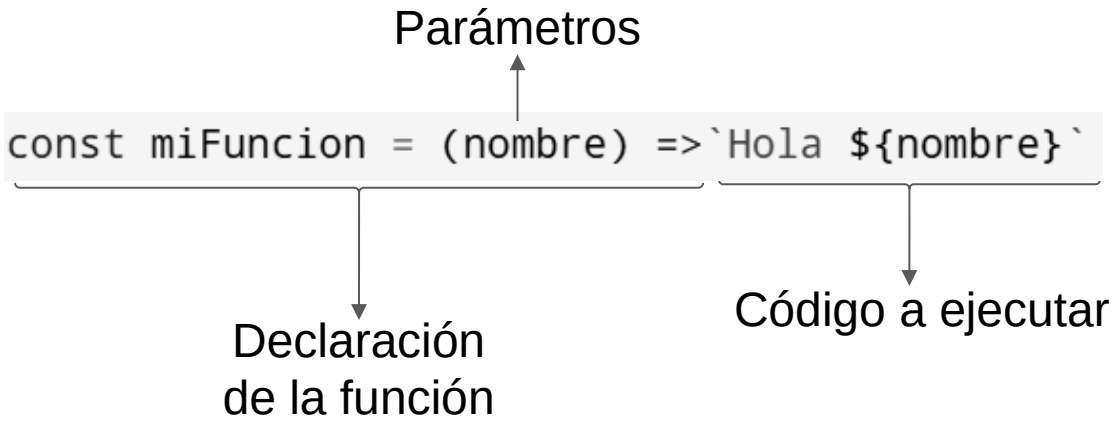
Las funciones flecha al igual que las funciones convencionales pueden ser creadas sin necesidad de que contengan parámetros o un retorno, Adelante ejemplificamos cómo es su sintaxis.

Función flecha sin parámetros



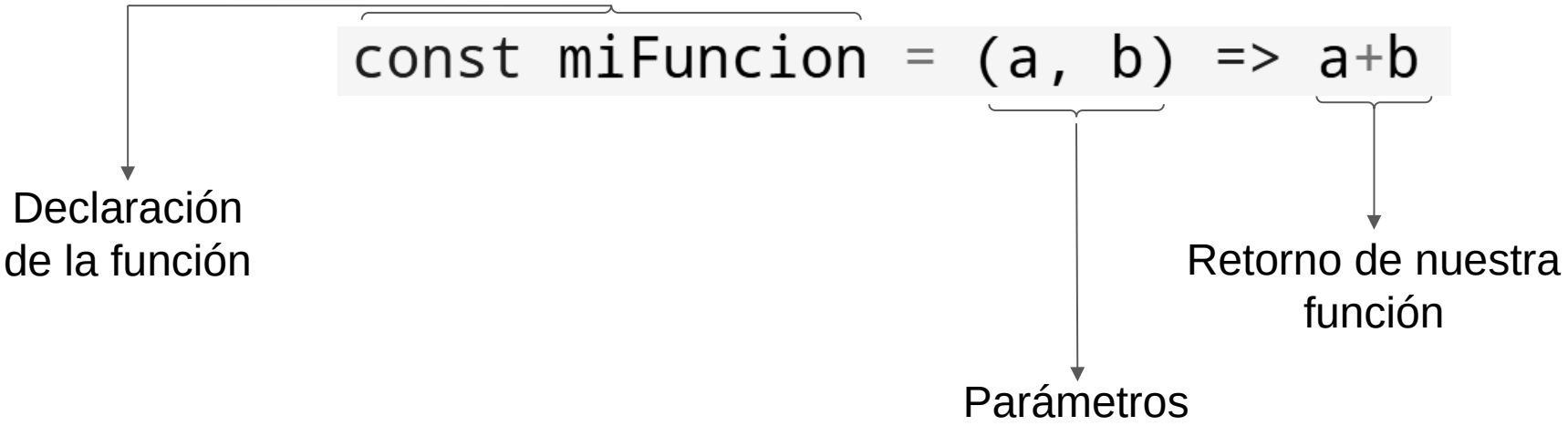
En este ejemplo creamos una función flecha la cual no tiene parámetros y tiene un retorno de 4*4

Función flecha con un parámetro



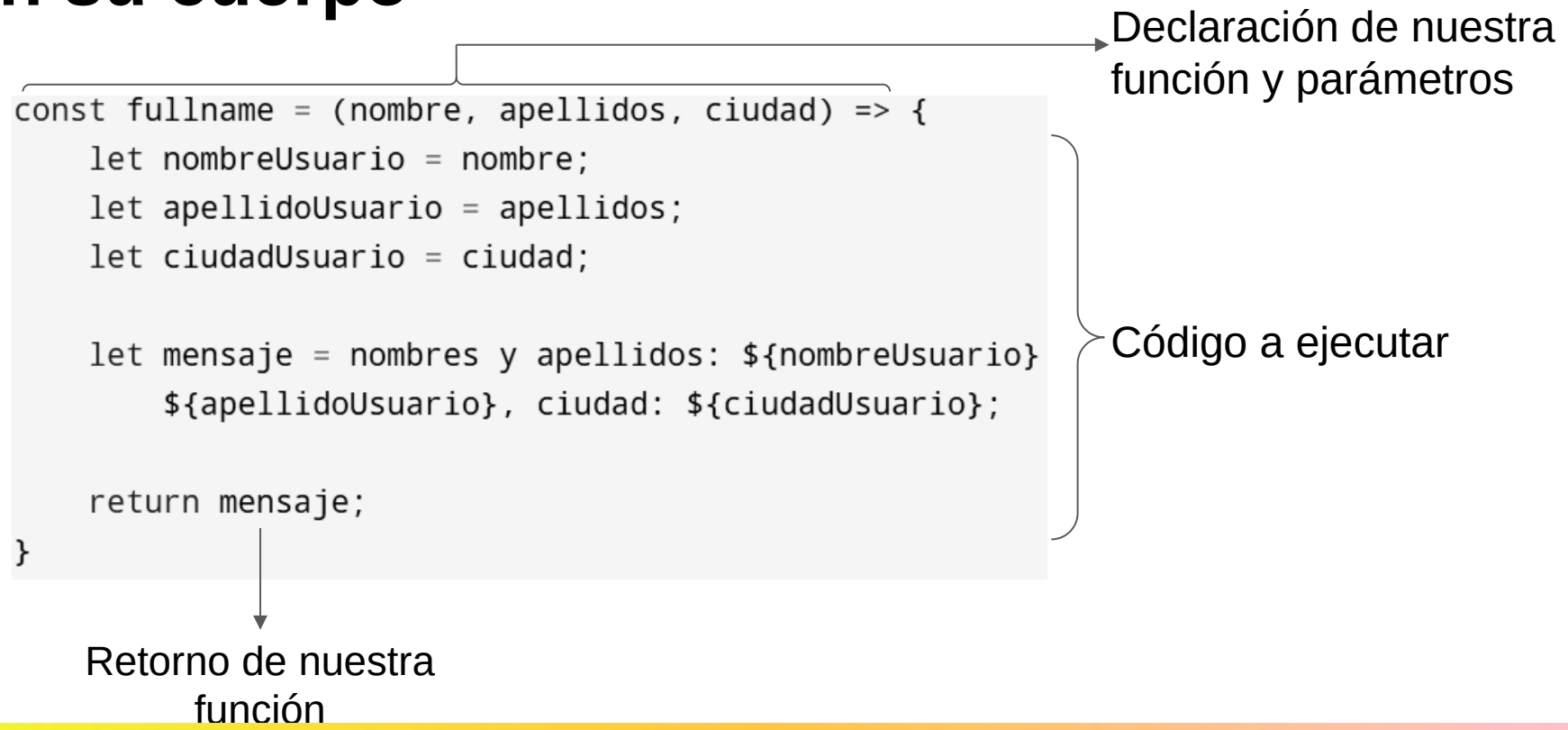
En este ejemplo creamos una función flecha la cual tiene como parámetro nombre, y nos retorna un saludo con el nombre pasado como argumento.

Función flecha con dos parámetros



En este ejemplo creamos una función flecha la cual tiene como parámetros a y b y nos retorna la suma de a y b.

Función flecha con más de una instrucción en su cuerpo



¿Como guardar el retorno de una función?

Como estuvimos viendo a lo largo de la explicación de funciones y funciones flecha, vimos que estas nos pueden generar una respuesta la cual la conocemos como retorno. Aquí vamos a aprender a como guardar esa respuesta en una variable.

```
const miFuncion = (nombre, edad) =>{
    let mensaje = `Hola soy ${nombre} y tengo ${edad} años`
    return mensaje
}

let saludo = miFuncion("Carlos", 21)
console.log(saludo)
```

En este ejemplo tenemos una función la cual nos genera un saludo personalizado, Para guardar el resultado de nuestra función creamos una variable por fuera de la función llamada saludo y su contenido va a ser igual al retorno de nuestra función, posteriormente imprimimos la variable saludo en la consola.

Pruebas de escritorio

La prueba de escritorio es una técnica utilizada en programación para simular manualmente la ejecución de un programa o fragmento de código con el objetivo de verificar su correcto funcionamiento. Durante este proceso, el programador sigue paso a paso las instrucciones del código, utilizando valores específicos para las variables, y anota los resultados de cada paso en una tabla o una hoja. Esto permite detectar errores lógicos, verificar el flujo de ejecución, y entender mejor cómo se comporta el programa antes de ejecutarlo en la computadora.

Ejemplo de prueba de escritorio

Se desea crear un algoritmo que determine el dinero total que dos hermanos tienen en sus respectivas alcancias.

Todos los datos que se necesitan para el algoritmo se pueden tratar como flotantes

Valores conocidos

- Dinero contenido en la primera alcancía
- Dinero contenido en la segunda alcancía

Valor desconocido

- Dinero total de las alcancías

Ejemplo de prueba de escritorio

Suponiendo que al ejecutar el algoritmo los hermanos tienen 150.000 en la primera alcancía y 230.000 en la segunda, la prueba de escritorio sería la mostrada en la ejecución 1 de la siguiente tabla.

Realizando la operación

total = alcancia1 + alcancia2

ejecución	Valores conocidos		Valores desconocidos
	alcancia1	alcancia2	total
1	150.000	230.000	380.000
2	75.000	63.000	138.000
3	120.000	2.500	122.500
4	20.700	80.900	101.600

Algoritmos: 5

VankVersity