# Reutilizando código. Funciones

PID_00275851

Autores que han participado colectivamente en esta obra
Lluís Beltrà
Kenneth Capseta
Maria Jesús Marco Galindo
Antonio Ponce

Idea y dirección de la obra
Maria Jesús Marco Galindo

Diseño y edición gráfica
Asunción Muñoz

**Material docente de la UOC**

Universitat Oberta de Catalunya

Tercera edición: febrero 2021

Av. Tibidabo, 39-43, 08035 Barcelona
Realización editorial: FUOC

# Contenidos

# Reutilizando código
## Funciones

Llegados a este punto ya sabemos diseñar algoritmos, empezando con una abstracción de la realidad del problema para definir los datos de entrada y salida, y luego decidiendo las acciones necesarias para resolverlo que se organizan y combinan, según sea necesario, utilizando las estructuras algorítmicas (secuencial, condicional y repetitiva) para determinar el flujo que deben seguir hasta llegar a la solución final.

En el código, hemos utilizado también funciones ya existentes que nos han permitido hacer algunas acciones sin tener que preocuparnos de cómo están diseñadas e implementadas. Todos los lenguajes de programación disponen de una serie de funciones predefinidas que podemos utilizar sin siquiera conocer el código que contienen. Son de hecho, programas que podemos utilizar dentro de nuestros programas. Esto da mucha potencia porque podemos programar a partir de código ya existente que se puede reutilizar tantas veces como se necesite.

### Funciones predefinidas

Entre las funciones habituales que todos los lenguajes de programación proporcionan se encuentran las que permiten intercambiar información entre el programa y el usuario. Son funciones imprescindibles, dado que controlan el intercambio de información con dispositivos como la pantalla o el teclado, y que ya tenemos predefinidas y se pueden utilizar sin saber cómo están implementadas.

Eso sí, cada lenguaje tiene sus propias funciones, pero algunas de ellas son muy similares o tienen la misma funcionalidad. En notación algorítmica, para simplificar, hemos definido las funciones que permiten gestionar la comunicación con el usuario como las funciones de entrada y salida de datos que ya conocemos:

```
variable := leerTeclado();
escribirPantalla(valor);
```

A partir de estas funciones conocidas se puede ver que hay dos tipos diferentes de funciones:

- Las que hacen alguna acción o cálculo y devuelven un valor: sería el caso de *leerTeclado (),* que devuelve el valor que se ha introducido a través del teclado.
- Las que modifican alguna cosa pero no devuelven ningún valor: sería el caso de *escribirPantalla (valor)*, que escribe un valor que le damos a la pantalla.

Conocemos también otras funciones predefinidas de los lenguajes de programación: las funciones de conversión de tipo. Aunque en cada lenguaje se concretan de una manera determinada, algorítmicamente, recordémoslo, las hemos definido así:

| | |
|---|---|
| **realAEntero** | Convierte un valor real a entero, dejando solo la parte entera del valor real.<br>**realAEntero** (3.5) devuelve el valor entero 3. |
| **enteroAReal** | Convierte un valor entero a real, agregando 0 a la parte decimal.<br>**enteroAReal** (3) devuelve 3.0. |
| **caracterAEntero** | Convierte un carácter a entero. Este entero es el que le corresponde al carácter según el sistema de codificación ASCII.<br>**caracterAEntero** (‘A’) da como resultado 65, que es el número con el que en codificación ASCII se representa la letra A mayúscula. |
| **enteroACaracter** | Convierte un entero en un carácter aplicando la tabla de codificación ASCII.<br>**enteroACaracter** (65) devuelve el carácter ‘A’. |
| **enteroACadena** | Convierte un entero en una cadena de caracteres.<br>**enteroACadena** (4) da como resultado ‘4’. |

Estas funciones reciben un valor y lo convierten en otro de un tipo diferente, que será el que retornen. Por ejemplo: *valor2: = enteroAReal (valor1),* que recibe un valor entero y lo convierte en un valor real que retorna. Son funciones, pues, similares a la función de *leerTeclado()*, que devuelven un valor. En este caso, sin embargo, necesitan recibir un valor inicial, con el que harán la conversión. Este valor se pasa como parámetro de la función, entre paréntesis justo después del nombre, de manera similar a cómo se hace en las funciones matemáticas..

# ¿Cómo podemos crear funciones?

## Ejemplo

Estamos en plena campaña de recogida de alimentos. Por suerte se han recogido muchas toneladas. Ahora hay que embalarlo todo en unos contenedores grandes, aprovechando al máximo el espacio. Nos ha tocado embalar los tetrabriks de leche. En cada contenedor caben 500. Para no estar calculando cada vez cuántos contenedores se necesitan, hemos pensado hacer un programa que nos facilite el trabajo.

La estrategia es la siguiente: a partir del número de tetrabriks de leche recogidos, para saber cuántos contenedores necesitamos, hay que hacer una división entre 500:

```
const
    capacidadContenedor : entero := 500;
fconst
var
    numTetrabriks : entero;
    numContenedores : entero;
fvar

numTetrabriks := leerTeclado();

numContenedores := numTetrabriks div capacidadContenedor;

si numTetrabriks mod capacidadContenedor ≠ 0 entonces
    numContenedores := numContenedores + 1;
fsi
```

De este modo, si por ejemplo tenemos 21.256 tetrabriks, se necesitan 43 contenedores, 42 de los cuales estarán llenos y el último solo contendrá 256 tetrabriks.

Ahora bien, imaginemos que no siempre nos llegan contenedores del mismo tamaño, a veces tienen capacidad para 500 tetrabriks, a veces para 300, etc. ¿Se pueden generalizar las acciones anteriores? Sí, efectivamente, solo habría que ajustar el valor de la variable *capacidadContenedor* a cada caso.

Ahora generalizamos el embalaje de los productos de recogida de alimentos teniendo en cuenta que tenemos que embalar leche, arroz y aceite, y que lo tenemos que hacer en los contenedores que envían. Además, en un mismo contenedor no se pueden mezclar productos diferentes.

```
const
    capacidadLeche : entero := 500;
    capacidadArroz : entero := 750:
    capacidadAceite : entero := 800
fconst

var
    numTetrabriksLeche : entero;
    numPaquetesArroz : entero;
    numLitrosAceite : entero;

    numContenedores, numContenedoresLeche, numContenedoresArroz,
    numContenedoresAceite : entero;
fvar

numTetrabriksLeche := leerTeclado();
numPaquetesArroz := leerTeclado();
numLitrosAceite := leerTeclado();

numContenedoresLeche := numTetrabriksLeche div capacidadLeche;
si numTetrabriksLeche mod capacidadLeche ≠ 0 entonces
    numContenedoresLeche := numContenedoresLeche + 1;
fsi

numContenedoresArroz := numPaquetesArroz div capacidadArroz;
si numPaquetesArroz mod capacidadArroz ≠ 0 entonces
    numContenedoresArroz := umContenedoresArroz + 1:
fsi

numContenedoresAceite := numLitrosAceite div capacidadAceite;
si numLitrosAceite mod capacidadAceite ≠ 0 entonces
    numContenedoresAceite := numContenedoresAceite + 1:
fsi

numContenedores := numContenedoresLeche +
numContenedoresArroz + numConetenedoresAceite.
```

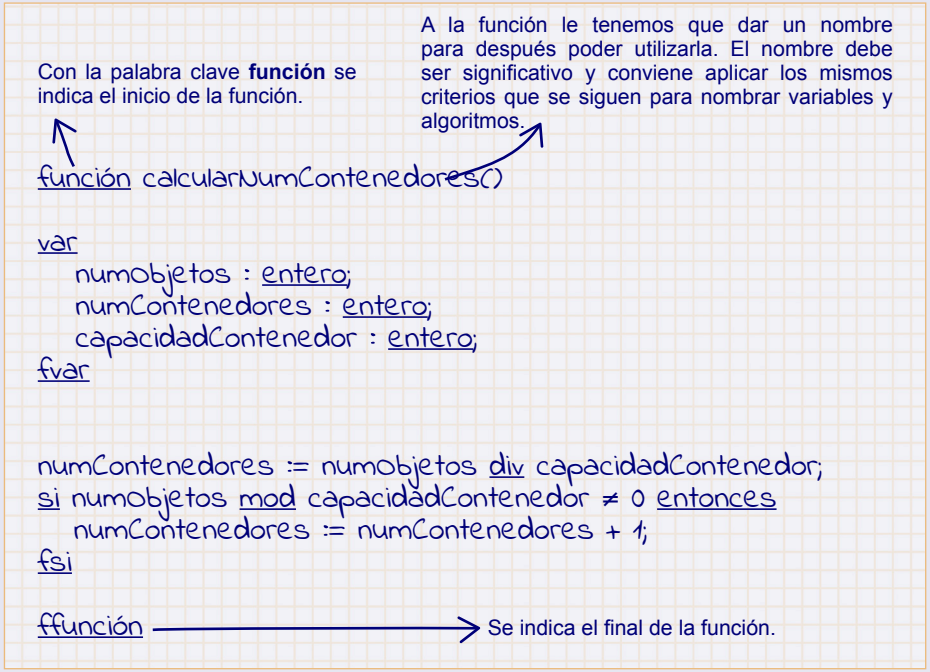
En el contenedor caben 500 tetrabrikcs de leche, o bien 750 paquetes de arroz, o bien 800 litros de aceite.

Calculamos el número de contenedores de leche que se necesitan.

Calculamos el número de contenedores de arroz que se necesitan.

Calculamos el número de contenedores de aceite que se necesitan.

Si nos fijamos, hemos repetido las mismas acciones para calcular los contenedores necesarios de leche, aceite y arroz. Imaginad que lo tenemos que hacer para decenas de productos, el algoritmo se alargará mucho y, repitiendo las mismas acciones. Podemos definir nuestras propias funciones para evitar tener que usar las mismas líneas de código una y otra vez.




Ahora tenemos las instrucciones para calcular el número de contenedores, "encapsulada" en una función que hemos creado y a la que hemos dado un nombre concreto. Ya podríamos utilizarla desde el algoritmo para simplificarlo. Pero antes necesitamos saber cómo podemos intercambiar datos entre el algoritmo y la función, en este caso *numObjetos* y *capacidadContenedor*. Porque, si no, ¿cómo lo haremos para indicar cuándo estamos calculando los contenedores de leche y cuándo los de aceite?

## Intercambio de datos entre un algoritmo y una función: parámetros

A veces una función necesita trabajar con datos que provienen del algoritmo que la utiliza. Por ejemplo, la función *escribirPantalla ("Hola")*, muestra por pantalla el valor que se le pasa, "Hola" en este caso. Otras veces, la función necesita devolver algún dato al algoritmo que la llama. Por ejemplo, la función enteroAReal(4.0), convierte el valor real que recibe y devuelve el valor entero equivalente, en este caso retorna 4.

Estos valores que se pasan a las funciones se llaman **parámetros** y se indican entre paréntesis justo después del nombre de la función.

Los parámetros que recibe la función se llaman **parámetros de entrada** y el que la función devuelve se llama **parámetro de salida**. Al igual que se hace con las funciones predefinidas, los parámetros de las funciones que definimos nosotros se indican justo después del nombre de la función entre paréntesis.

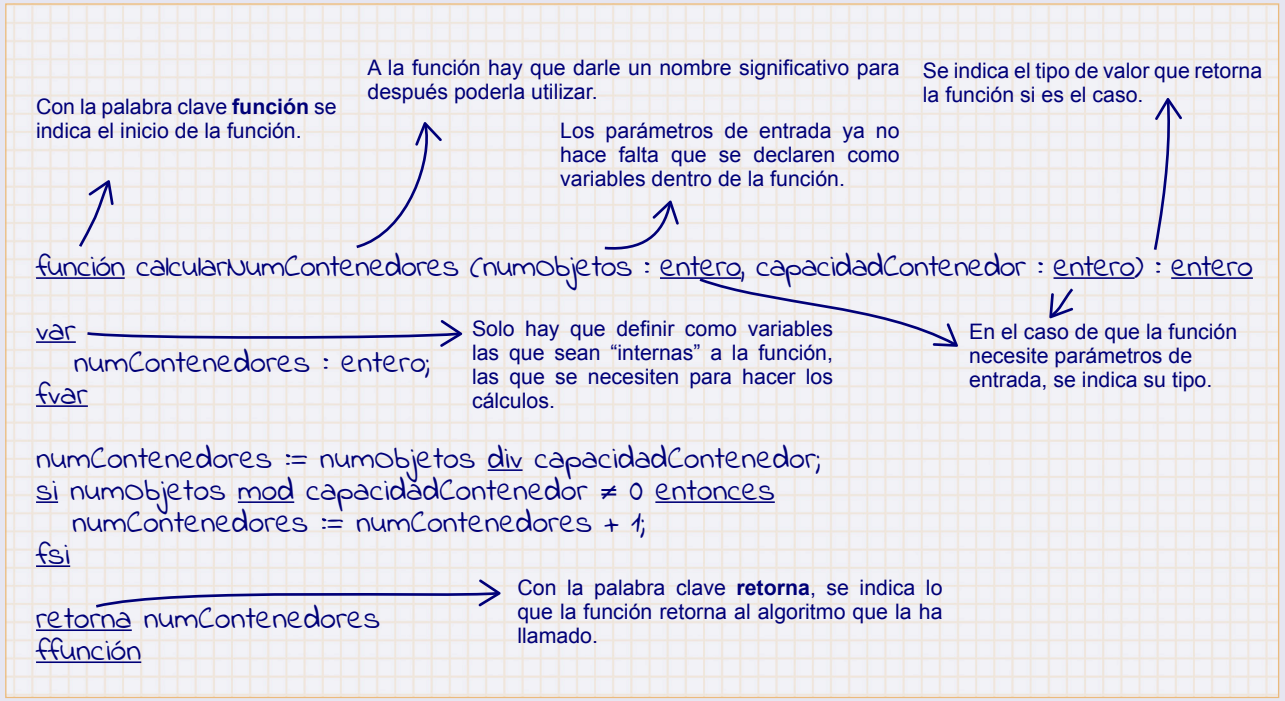
### Ejemplo

Veamos cómo podemos simplificar el algoritmo para calcular los contenedores necesarios para almacenar una remesa de recogida de alimentos con el uso de la función que hemos creado.

En primer lugar, es necesario indicar los parámetros a través de los que la función intercambiará datos con el algoritmo que la llame:

- **Datos de entrada.** Es necesario que le pasemos (dato de entrada) el número de objetos a embalar y también la capacidad de objetos que caben en el contenedor, que será diferente según el producto del que se trate.
- **Datos de salida.** La función devolverá el número de contenedores necesarios.

Fijaos que este paso de pensar en los datos que hay que manipular es el mismo que se hace cuando empezamos a diseñar un algoritmo.



Para utilizar funciones desde el código del programa o algoritmo, se deben "llamar". La llamada se hace simplemente con el nombre de la función seguida de paréntesis. Dentro del paréntesis, hay que pasarle también los parámetros de entrada si hacen falta. En este caso, la función ejecuta su código y, una vez termina, devuelve el flujo de ejecución al programa principal, justo en la instrucción siguiente a la llamada a la función.

```
algoritmo

const
    capacidadLeche : entero := 500;
    capacidadArroz : entero := 750;
    capacidadAceite : entero := 800;
fconst

var
    numTetrabriksLeche : entero;
    numPaquetesArroz : entero;
    numLitrosAceite : entero;

    numTotalContenedores, numContenedoresLeche, numContenedoresArroz, numContenedoresAceite : entero;
fvar

numTetrabriksLeche := leerTeclado();
numPaquetesArroz := leerTeclado();
numLitrosAceite := leerTeclado();

numContenedoresLeche := calcularNumContenedores (numTetrabriksLeche, capacidadLeche);
numContenedoresArroz := calcularNumContenedores (numPaquetesArroz, capacidadArroz);
numContenedoresAceite := calcularNumContenedores (numLitrosAceite, capacidadAceite);

numTotalContenedores := numContenedoresLeche + numContenedoresArroz + numContenedoresAceite;

falgoritmo
```

En el contenedor caben 500 tetrabriks de leche, o bien 750 paquetes de arroz, o bien 800 litros de aceite.

A la función le hemos dado un nombre para después poderla utilizar. El nombre debe ser significativo y conviene aplicar los mismos criterios que se usan para nombrar variables y algoritmos.

Los parámetros de entrada ya NO hace falta que se declaren como variables de la función.

Calculamos el número de contenedores de aceite que se necesitan.

Calculamos el número de contenedores de arroz que se necesitan.

Calculamos el número de contenedores de leche que se necesitan.

Así, el algoritmo es más corto y fácil de entender. Y se evita ir repitiendo el mismo código.

La función define su propio ámbito. Esto significa que todo lo que se define dentro de la función (variables y constantes) se queda en la función y no se puede utilizar fuera, por ejemplo en el algoritmo que la llama.

Cualquier comunicación entre función y algoritmo debe pasar por los parámetros. Y las variables intermedias que necesita la función para las acciones que deba hacer se deben definir dentro de la propia función. En el caso de este ejemplo, *numContenedores* es una variable local de la función *calcularNumContenedores*, por tanto, es una variable que el algoritmo no puede usar, digamos que no la "ve". Igualmente, las únicas variables que puede ver la función del algoritmo son los parámetros que se le pasan, en este caso *numObjetos* y *capacidadContenedor*.

Otra ventaja importante de las funciones es que permiten la resolución de problemas más complejos de los que hemos visto hasta ahora. Un problema complejo es aquel que no se puede resolver directamente, así de golpe, sino que hay que descomponerlo en subproblemas más pequeños y resolver cada uno de ellos por separado.

Por ejemplo, con el uso de las funciones hemos resuelto el problema anterior, dividiéndolo en distintos subproblemas:

1) Obtener el número de paquetes de cada tipo de alimento que tenemos.
2) Calcular el número de contenedores necesarios para almacenar cada tipo de alimento.
3) Mostrar los resultados.

Para el primer subproblema, utilizamos la función predefinida de entrada; para el segundo subproblema, la función que hemos creado; y para el tercer subproblema, la función predefinida de salida.

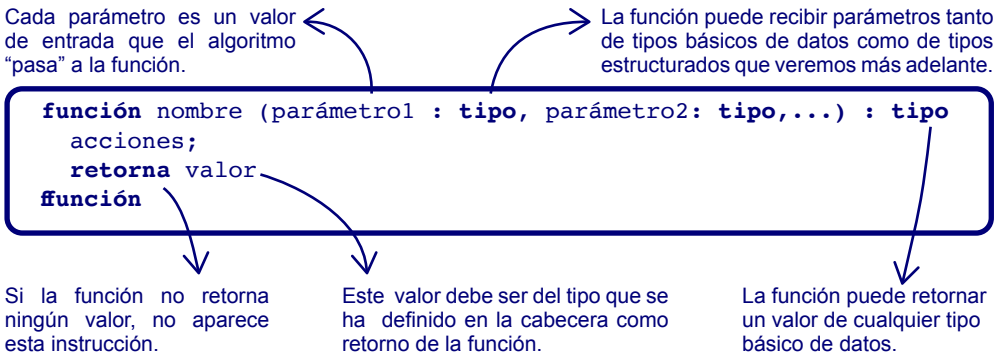
Una función es únicamente un fragmento de código, al igual que un algoritmo. La única diferencia es que puede recibir y devolver valores fuera de su ámbito. De esta manera se comunica con el programa que la llama.

Con las funciones tenemos la posibilidad de definir nuestros propios bloques de código para ejecutarlos cuando y desde donde queramos, y tantas veces como lo necesitemos. Para ello, necesitamos un nombre para definir nuestra función, determinar el código a ejecutar (siguiendo la misma estrategia aprendida para diseñar un algoritmo) y definir los parámetros que utilizaremos cuando hacemos la llamada, si es que necesitamos pasar valores a la función (parámetros de entrada), y el valor que devolverá después de ejecutarse (parámetro de salida).

Una función, aunque esté definida, nunca se ejecutará si se llama desde algún lugar del código del algoritmo o desde otra función.

Es bastante habitual tratar de centralizar todas las funciones que declaramos en un lugar concreto del código, al principio, como hacemos con las variables y constantes.

## Sintaxis para la declaración de una función




### Conceptos clave

Una **función** es un fragmento de código que tiene una funcionalidad concreta y que se puede usar (llamar) desde cualquier programa o función. Es muy útil, en primer lugar, para evitar tener que repetir el mismo código una y otra vez. En segundo lugar, para trabajar con problemas complejos que no se pueden resolver de golpe sino que requieren dividirlos en subproblemas más pequeños que los resuelven parcialmente. Cada uno de estos subproblemas se puede resolver con una función específica.

Esta técnica de dividir un programa en funciones con el fin de hacerlo más claro, fácilmente abordable y comprensible se llama **programación modular**.

Es una buena práctica que la comunicación de datos entre el programa y la función se haga a través de los **parámetros** y que cada función defina las variables específicas que necesite para resolver el problema. Hay, sin embargo, lenguajes de programación que son más laxos en este sentido y permiten otros mecanismos de compartición de datos entre un programa y una función.

Los lenguajes de programación, además, disponen de **funciones predefinidas** que se pueden utilizar directamente sin necesidad de saber cómo están implementadas. Normalmente estas funciones se organizan en librerías. Sería el caso de las funciones de entrada/salida o de conversión de tipo, por ejemplo.

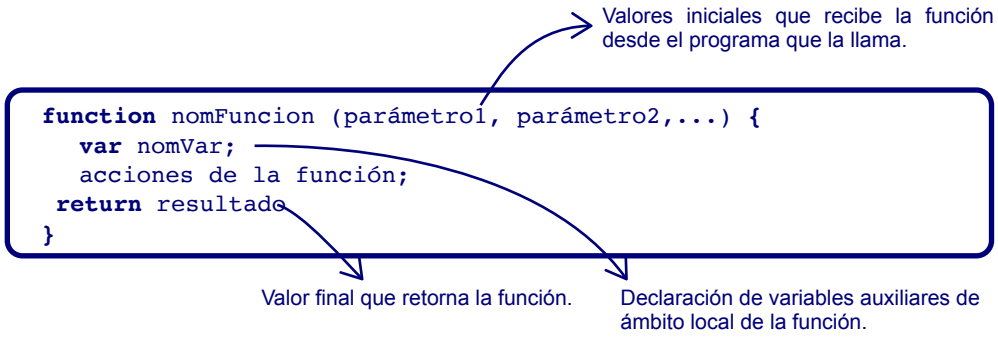
# ¿Cómo definimos y utilizamos funciones en JavaScript?

Tened en cuenta que, a estas alturas, aunque tal vez no nos hemos dado cuenta, ya hemos utilizado alguna función en JavaScript. Seguramente la que más hemos utilizado es la función **console.log** que permite escribir un valor por la pantalla. Habitualmente se utiliza para mostrar el resultado de un algoritmo, los datos de salida.

Es una de las funciones predefinidas que el propio lenguaje JavaScript proporciona. No conocemos el código que contiene, pero no importa, se puede utilizar llamándola con su nombre y pasando los parámetros que necesita, si es el caso.

En JavaScript también podemos definir nuestras propias funciones, por aquello que decíamos de dividir un problema complejo en subproblemas más pequeños, cada uno resuelto a partir de una función o bien para evitar repetir el mismo código una y otra vez en un algoritmo.

## Sintaxis para declarar funciones en JS




Fijaos que en JS no se indica el tipo de los parámetros. Es lo mismo que pasa cuando se define una variable, JS deduce el tipo del parámetro en el momento que se le da un valor.

### Ejemplo

Supongamos que somos un concesionario que vende coches de segunda mano. Estamos fabricando placas para coches de diferentes marcas. Las placas se ponen en el lugar de la matrícula mientras el coche está esperando para ser vendido.

Para poder calcular la longitud máxima de las placas de cada remesa, necesitamos un programa que identifique el nombre de marca más largo entre los coches que hay en el concesionario en cada momento.

La propiedad *length* aplicada a un nombre indica el número de caracteres que tiene.

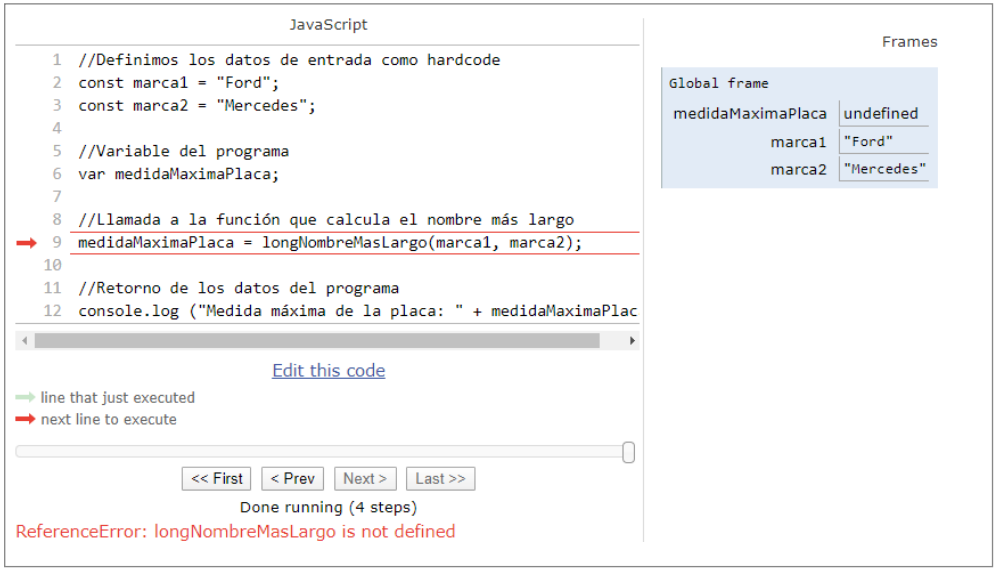
```
1  //Definimos los datos de entrada como hardcode
2  const marca1 = "Ford";
3  const marca2 = "Mercedes";
4
5  //Variable del programa
6  var medidaMaximaPlaca;
7
8  //Definimos la función
9  function longNombreMasLargo (nombre1, nombre2) {
10    var nombreLargo; //variable local de la función
11    if (nombre1.length > nombre2.length) {
12       nombreLargo = nombre1;
13    } else {
14       nombreLargo = nombre2;
15    }
16    return nombreLargo.length;
17 }
18
19 //Llamada a la función que calcula el nombre más largo
20 medidaMaximaPlaca = longNombreMasLargo(marca1, marca2);
21
22 //Retorno de los datos del programa
23 console.log ("Medida máxima de la placa: " + medidaMaximaPlaca);
24
```

Igual que en las unidades anteriores, encontraréis una explicación detallada del funcionamiento de las funciones en JavaScript en la guia de JavaScript de Mozilla.

## ¿Qué pasaría si...

... llamamos a una función que no existe?

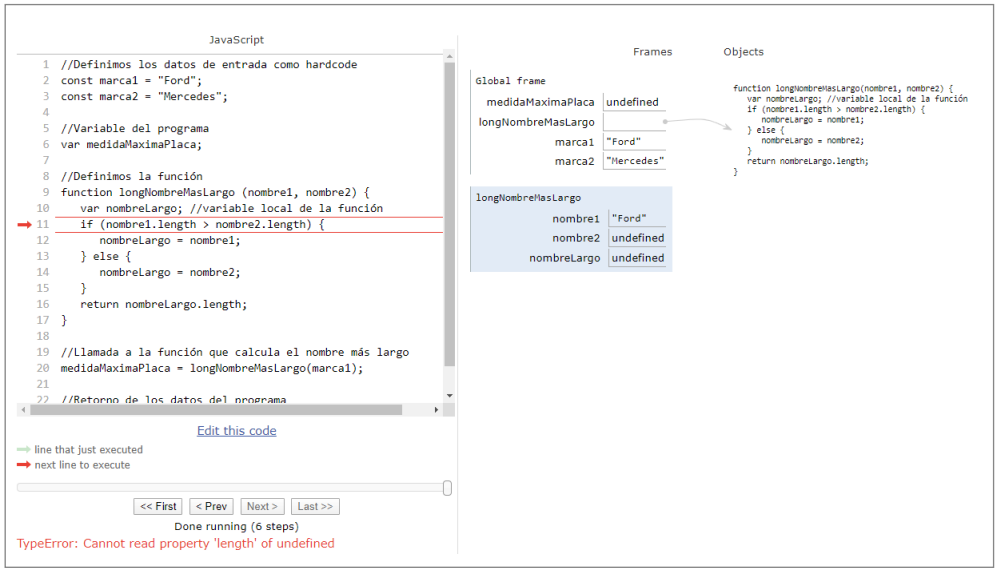
El programa se detiene porque no encuentra ninguna función con el nombre de la llamada. Veámoslo con el PythonTutor.




## ¿Qué pasaría si...

... no le pasamos los parámetros que la función necesita?

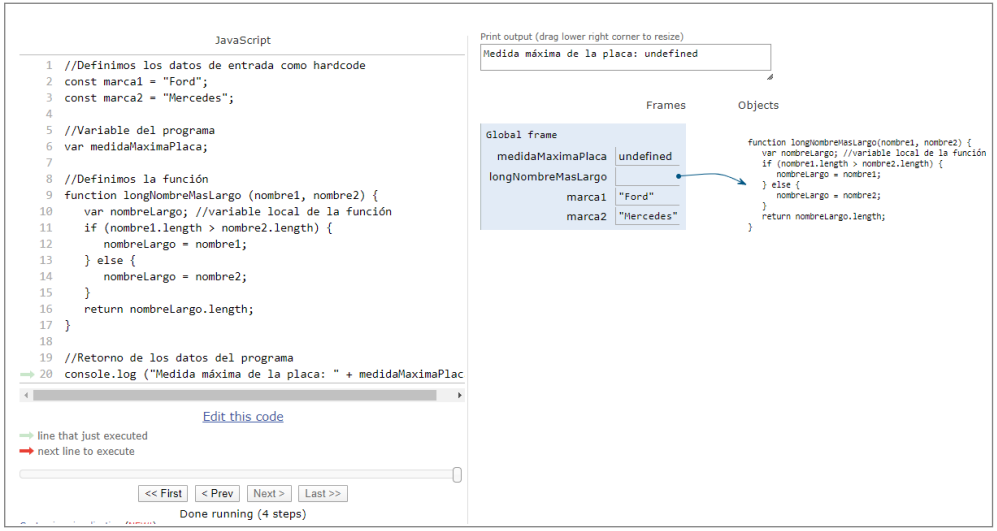
Dado que la función no recibe los datos que necesita, en el momento de la ejecución aparece un error ya que no se puede ejecutar una de las instrucciones al estar el dato indefinido. Tened en cuenta que el PythonTutor nos ayuda muy claramente a ver el flujo de ejecución y el valor de las variables tanto del ámbito del programa como de la función.



# ¿Qué pasaría si...

... definimos la función pero no la llamamos desde el programa principal?

Las instrucciones de la función no se ejecutan nunca. El programa solo ejecuta 4 instrucciones y, por tanto, el resultado queda indefinido.




Podéis probar estos códigos en la web de PythonTutor.