Reutilitzant el codi. Funcions

PID_00275850

Autors que han participat col·lectivament en aquesta obra
Lluís Beltrà
Kenneth Capseta
Maria Jesús Marco Galindo
Antonio Ponce

Idea i direcció de l'obra
Maria Jesús Marco Galindo

Disseny i edició gráfica
Asunción Muñoz

Material docent de la UOC

Universitat Oberta
de Catalunya

Tercera edició: febrer 2021
© Luis Beltrà Valenzuela, Kenneth Capeseta Nieto, Antonio Ponce Tarela, Asunción Muñoz Fernández, Maria Jesús Marco Galindo
Tots els drets reservats
© d'aquesta edició, FUOC, 2020
Av. Tibidabo, 39-43, 08035 Barcelona
Realització editorial: FUOC

Cap part d'aquesta publicació, incloent-hi el disseny general i la coberta, no pot ser copiada, reproduïda, emmagatzemada o transmesa de cap manera ni per cap mitjà, tant si és elèctric com químic, mecànic, òptic, de gravació, de fotocòpia o per altres mètodes, sense l'autorització prèvia per escrit dels titulars dels drets.

Continguts

Reutilitzant el codi

Funcions

- Funcions predefinides
- Com podem crear funcions?
- Intercanvi de dades entre un algorisme i una funció: paràmetres
- Sintaxi per a declarar una funció
- Conceptes clau

Com definim i utilitzem funcions en JavaScript?

- Sintaxi per declarar funcions en JS
- Què passaria si…

Reutilitzant el codi
Funcions

Arribats a aquest punt ja sabem dissenyar algorismes, començant per fer una abstracció de la realitat del problema per definir les dades d'entrada i sortida, i després decidint les accions necessàries per resoldre'l, que s'organitzen i es combinen segons calgui tot utilitzant les estructures algorísmiques (seqüencial, condicional i repetitiva) per determinar el flux que han de seguir fins a arribar a la solució final.

En el codi, hem fet servir també funcions ja existents que ens han permès de fer algunes accions sense haver-nos de preocupar de com estan dissenyades i programades. Tots els llenguatges de programació disposen d'un seguit de funcions predefinides que podem utilitzar sense ni tan sols conèixer el codi que contenen. Són, de fet, programes que podem utilitzar dins els nostres programes. Això ens dona molta potència perquè podem programar a partir de codi ja existent que es pot reutilitzar tantes vegades com calgui.

Funcions predefinides

Entre les funcions habituals que tots els llenguatges de programació proporcionen es troben les que permeten intercanviar informació entre el programa i l'usuari. Són funcions imprescindibles, atès que controlen l'intercanvi d'informació amb dispositius com la pantalla o el teclat, i que ja tenim predefinides i es poden utilitzar sense saber com estan implementades.

Això sí, cada llenguatge té les seves pròpies funcions, però algunes d'elles són molt similars o tenen la mateixa funcionalitat. En notació algorísmica, per simplificar, hem definit les funcions que permeten gestionar la comunicació amb l'usuari com a les funcions d'entrada i sortida de dades que ja coneixem:

```
variable := llegirTeclat()
escriurePantalla(valor)
```

A partir d'aquestes funcions conegudes es pot veure que hi ha dos tipus diferents de funcions:

- Les que fan alguna acció o càlcul i retornen un valor, seria el cas de *llegirTeclat()*, que retorna el valor que s'ha introduït a través del teclat.
- Les que modifiquen alguna cosa però no retornen cap valor, seria el cas de *escriurePantalla(valor)*, que escriu un valor que li donem a la pantalla.

Coneixem també altres funcions predefinides dels llenguatges de programació: les funcions de conversió de tipus. Tot i que en cada llenguatge es concreten d'una manera determinada, algorísmicament, recordem-ho, les hem definit així:

realAEnter	Converteix un valor real a enter, deixant només la part entera del valor real. **realAEnter** (3.5) retorna el valor enter 3.
enterAReal	Converteix un valor enter a real, afegint 0 a la part decimal. **enterAReal** (3) retorna 3.0.
caracterAEnter	Converteix un caràcter a enter. Aquest enter és el que li correspon al caràcter segons el sistema de codificació ASCII. **caracterAEnter** ('A') dona com a resultat 65, que és el número amb què en codificació ASCII es representa la lletra A majúscula.
enterACaracter	Converteix un enter en un caràcter tot aplicant la taula de codificació ASCII. **enterACaracter** (65) retorna el caràcter 'A'.
enterACadena	Converteix un enter en una cadena de caràcters. **enterACadena** (4) dona com a resultat '4'.

Aquestes funcions reben un valor i el converteixen en un altre d'un tipus diferent que serà el que retornin. Per exemple: *valor2 := enterAReal (valor1)* que rep un valor enter i el converteix en un valor real que és el que retorna. Són funcions doncs, similars a la funció de *llegirTeclat()*, que retornen un valor. En aquest cas, però, necessiten rebre un valor inicial, amb el que faran la conversió. Aquest valor es passa com a paràmetre de la funció, entre parèntesi just després del nom, de manera similar a com es fa a les funcions matemàtiques.

Com podem crear funcions?

Exemple

Estem en plena campanya de recapte d'aliments. Per sort, s'han recollit moltes tones. Ara cal embalar-ho tot en uns contenidors grans, tot aprofitant al màxim l'espai. Ens ha tocat embalar els tetra brics de llet. A cada contenidor n'hi caben 500. Per no estar calculant cada vegada quants contenidors ens calen, farem un programa que ens facilitarà la feina.

L'estratègia és la següent: a partir del nombre de tetra brics de llet recaptats, per saber quants contenidors necessitem, cal fer una divisió entre 500:

```
const
    capacitatContenidor : enter := 500;
fconst
var
    numTetrabrics : enter;
    numContenidors : enter;
fvar

numTetrabrics := llegirTeclat();

numContenidors := numTetrabrics div capacitatContenidor;

si numTetrabrics mod capacitatContenidor ≠ 0 llavors
    numContenidors := numContenidors + 1;
fsi
```

D'aquesta manera, si per exemple, tenim 21.256 tetra brics, es necessiten 43 contenidors, 42 dels quals estaran plens, i el darrer només contindrà 256 tetra brics.

Ara bé, imaginem que no sempre ens arriben contenidors de la mateixa mida, a vegades tenen capacitat per a 500 tetra brics, a vegades per a 300, etc. Podem generalitzar les accions anteriors? Sí, efectivament, només caldria ajustar el valor de la variable *capacitatContenidor* a cada cas.

Ara generalitzem l'embalatge dels productes de recapte d'aliments tenint en compte que hem d'embalar llet, arròs i oli, i que ho hem de fer en els contenidors que ens envien, i que en un mateix contenidor no es poden barrejar productes diferents.

```
const
    capacitatLlet : enter := 500;
    capacitatArròs : enter := 750:
    capacitatOli : enter := 800
fconst

var
    numTetrabricsLlet : enter;
    numPaquetsArròs : enter;
    numLitresOli : enter;

    numContenidors, numContenidorsLlet, numContenidorsArròs,
    numContenidorsOli : enter;
fvar

numTetrabricsLlet := llegirTeclat();
numPaquetsArròs := llegirTeclat();
numLitresOli := llegirTeclat();

numContenidorsLlet  := numTetrabricsLlet div capacitatLlet;
si numTetrabrikcsLlet mod capacitatLlet ≠ 0 llavors
    numContenidorsLlet := numContenidorsLlet + 1;
fsi

numContenidorsArròs := numPaquetsArròs div capacitatArròs;
si numPaquetsArròs mod capacitatArròs ≠ 0 llavors
    numContenidorsArròs := numContenidorsArròs + 1:
fsi

numContenidorsOli := numLitresOli div capacitatOli;
si numLitresOli mod capacitatOli ≠ 0 llavors
    numContenidorsOli := numContenidorsOli + 1:
fsi

numContenidors := numContenidorsLlet + numContenidorsArròs
+ numContenidorsOli.
```

Al contenidor hi cabem 500 tetra brics de llet o bé 750 paquets d'arròs, o bé 800 litres d'oli.

Calculem el número de contenidors de llet que calen.

Calculem el número de contenidors d'arròs que calen.

Calculem el número de contenidors d'oli que calen.

Si ens hi fixem, hem repetit les mateixes accions per calcular els contenidors necessaris de llet, oli i arròs. Imagineu que ho hem de fer per desenes de productes, l'algorisme s'allarga molt i molt, tot repetint les mateixes accions. Podem definir les nostres pròpies funcions per evitar haver de fer servir les mateixes línies de codi una vegada i una altra.

Amb la paraula clau **funció** indiquem l'inici de la funció.

A la funció li hem de donar un nom per després poder-la utilitzar. El nom ha de ser significatiu i convé aplicar els mateixos criteris que per nomenar les variables i els algorismes.

```
funció calcularNumContenidors()

var
    numObjectes : enter;
    numContenidors : enter;
    capacitatContenidor : enter;
fvar

numContenidors := numObjectes div capacitatContenidor;
si numObjectes mod capacitatContenidor ≠ 0 llavors
    numContenidors := numContenidors + 1;
fsi

ffunció
```

S'indica el final de la funció.

Ara tenim les instruccions per calcular el nombre de contenidors, "encapsulada" en una funció que hem creat i a la qual hem donat un nom concret. Ja podríem utilitzar-la des de l'algorisme per simplificar-lo. Però ens cal, abans, saber com podem intercanviar dades entre l'algorisme i la funció, en aquest cas *numObjectes* i *capacitatContenidor*. Perquè, si no, com farem per indicar quan estem calculant els contenidors de llet i quan els d'oli?

Intercanvi de dades entre un algorisme i una funció: paràmetres

A vegades una funció necessita treballar amb dades que provenen de l'algorisme que la utilitza. Per exemple, la funció *escriurePantalla("Hola")* mostra per pantalla el valor que se li passa, "Hola" en aquest cas. Altres vegades, la funció necessita tornar alguna dada a l'algoritme. Per exemple, la funció enterAReal (4.0), converteix el valor real que rep i retorna el valor enter equivalent, en aquest cas retorna 4.

Aquests valors que es passen a les funcions es diuen **paràmetres** i s'indiquen entre parèntesi just després del nom de la funció.

Els paràmetres que rep la funció es diuen **paràmetres d'entrada** i el que la funció retorna es diu **paràmetre de sortida**. Tal com es fa amb les funcions predefinides, els paràmetres de les funcions que definim nosaltres s'indiquen just després del nom de la funció entre parèntesi.

Exemple

Vegem com podem simplificar l'algorisme per calcular els contenidors necessaris per emmagatzemar una remesa de recapte d'aliments amb l'ús de la funció que hem creat.

Primer de tot, cal que indiquem els paràmetres a través dels quals la funció intercanviarà dades amb l'algorisme que la cridi:

- **Dades d'entrada.** Cal que li passem (dada d'entrada) el nombre d'objectes a embalar i també la capacitat d'objectes que caben al contenidor que serà diferent segons el producte del qual es tracti.
- **Dades de sortida.** La funció retornarà el nombre de contenidors necessari.

Fixeu-vos que aquest pas de pensar en les dades que hem de manipular és el mateix que es fa quan comencem a dissenyar un algorisme.

```
funció calcularNumContenidors (numObjectes : enter, capacitatContenidor : enter) : enter
var
   numContenidors : enter;
fvar

numContenidors := numObjectes div capacitatContenidor;
si numObjectes mod capacitatContenidor ≠ 0 llavors
   numContenidors := numContenidors + 1;
fsi

retorna numContenidors
ffunció
```

Amb la paraula clau **funció** indiquem l'inici de la funció.

A la funció li hem de donar un nom significatiu per després poder-la utilitzar.

Els paràmetres d'entrada, ja no cal que els declarem com a variables dins la funció.

S'indica el tipus de valor que retorna la funció si és el cas.

Només cal definir com a variables les que siguin "internes" a la funció, les que es necessitin per fer els càlculs.

En el cas que la funció necessiti paràmetres d'entrada, s'indica el seu tipus.

Amb la paraula clau **retorna**, s'indica el valor que la funció retorna a l'algorisme principal, si és el cas que retorni algun valor.

Per utilitzar funcions des del codi del programa o algorisme, s'han de "cridar". La crida es fa simplement amb el nom de la funció seguida de parèntesis. Dins els parèntesi cal posar també els paràmetres d'entrada, si calen. En aquest cas, la funció executa el seu codi i, un cop s'acaba, retorna el flux d'execució al programa principal, just a la instrucció següent de la crida a la funció.

```
algorisme

const
    capacitatLlet : enter := 500;
    capacitatArròs : enter := 750;
    capacitatOli : enter := 800;
fconst

var
    numTetrabricsLlet : enter;
    numPaquetsArròs : enter;
    numLitresOli : enter;

    numTotalContenidors, numContenidorsLlet, numContenidorsArròs, numContenidorsOli : enter;
fvar

numTetrabricsLlet := llegirTeclat();
numPaquetsArròs := llegirTeclat();
numLitresOli := llegirTeclat();

numContenidorsLlet := calcularNumContenidors (numTetrabricsLlet, capacitatLlet);
numContenidorsArròs := calcularNumContenidors (numPaquetsArròs, capacitatArròs);
numContenidorsOli := calcularNumContenidors (numLitresOli, capacitatOli);

numTotalContenidors := numContenidorsLlet + numContenidorsArròs + numContenidorsOli.

falgorisme
```

Al contenidor hi caben 500 tetra brics de llet, o bé 750 paquets d'arròs, o bé 800 litres d'oli.

A la funció li hem de donar un nom per després poder-la utilitzar. El nom ha de ser significatiu i convé aplicar els mateixos criteris que per nomenar les variables i els algorismes.

Els paràmetres d'entrada, ja NO cal que els declarem com a variables dins la funció.

Calculem el número de contenidors d'oli que calen.

Calculem el número de contenidors d'arròs que calen.

Calcular el número de contenidors de llet que calen.

Així, l'algorisme és més curt i fàcilment comprensible. I ens estalviem d'anar repetint el mateix codi.

La funció defineix el seu propi àmbit, això vol dir que tot el que es defineix dins la funció (variables i constants), es queda en la funció, no es pot utilitzar fora, per exemple en l'algorisme que la crida. Qualsevol comunicació entre funció i algorisme ha de passar pels paràmetres. I, les variables intermèdies que necessita la funció per fer les accions que hagi de fer, s'han de definir dins de la pròpia funció

En el cas d'aquest exemple *numContenidors* és una variable local de la funció *calcularNumContenidors,* per tant, és una variable que l'algorisme no pot fer servir, diguem que no la "veu". Igualment, les úniques variables que pot veure la funció de l'algorisme són els paràmetres que se li passen, en aquest cas *numObjectes*, *capacitatContenidor*.

Un altre avantatge important de les funcions és que permeten la resolució de problemes més complexos dels que hem vist fins ara. Un problema complex és aquell que no es pot resoldre directament, de cop, sinó que cal descompondre'l en subproblemes més petits i resoldre cadascun d'ells per separat.

Per exemple, amb l'ús de les funcions hem resolt el problema anterior, tot dividint-lo en subproblemes:

1) Obtenir el número de paquets de cada tipus d'aliment que tenim.
2) Calcular el número de contenidors necessaris per emmagatzemar cada tipus d'aliment.
3) Mostrar els resultats.

Per al primer subproblema, utilitzem la funció predefinida d'entrada; per al segon subproblema, la funció que hem creat; i per al tercer subproblema, de nou la funció predefinida de sortida.

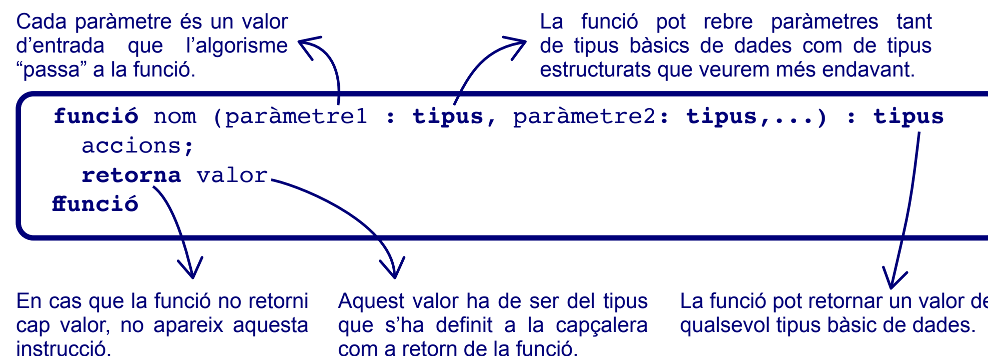
Una funció és només un fragment de codi, igual que un algorisme. L'única diferència és que pot rebre i retornar valors fora del seu àmbit. D'aquesta manera es comunica amb el programa que la crida.

Amb les funcions tenim la possibilitat de definir els nostres propis blocs de codi per executar-los quan i des d'on volguem i tantes vegades com ho necessitem. Per això necessitem un nom per definir la nostra funció, determinar el codi a executar (seguint la mateixa estratègia apresa per dissenyar algorismes) i definir els paràmetres que utilitzarem quan fem la crida si és que es necessitem passar valors a la funció (paràmetres d'entrada) i el valor que retornarà després d'executar-se (paràmetre de sortida).

Una funció encara que estigui definida mai no s'executarà si no la cridem des d'algun lloc del codi de l'algorisme o des d'una altra funció.

És força habitual provar de centralitzar totes les funcions que declarem en un lloc concret del codi, al començament, com fem amb les variables i les constants.

Sintaxi per a declarar una funció



Conceptes clau

Una **funció** és un fragment de codi que té una funcionalitat concreta i que es pot fer servir (cridar) des de qualsevol línia d'un programa o funció. És molt útil en primer lloc per estalviar haver de repetir el mateix codi una vegada i una altra. En segon lloc, per treballar amb problemes complexos que no es poden resoldre de cop sinó que requereixen dividir-los en subproblemes més petits que els van resolent parcialment. Cadascun d'aquests subproblemes es pot resoldre amb una funció específica.

Aquesta tècnica de dividir un programa en funcions amb la finalitat de fer-lo més clar, entenedor i senzill de resoldre es diu **programació modular**.

És una bona pràctica que la comunicació de dades entre el programa i la funció es faci a través dels **paràmetres** i que cada funció defineixi les variables específiques que necessiti per resoldre el problema. Hi ha, però, llenguatges de programació que són més laxes en aquest sentit i permeten altres mecanismes de compartició de dades entre un programa i una funció.

Els llenguatges de programació, a més, disposen de **funcions predefinides** que es poden fer servir directament sense necessitat de saber com estan implementades. Normalment aquestes funcions s'organitzen en llibreries. Seria el cas de les funcions d'entrada/sortida o les de conversió de tipus, per exemple.

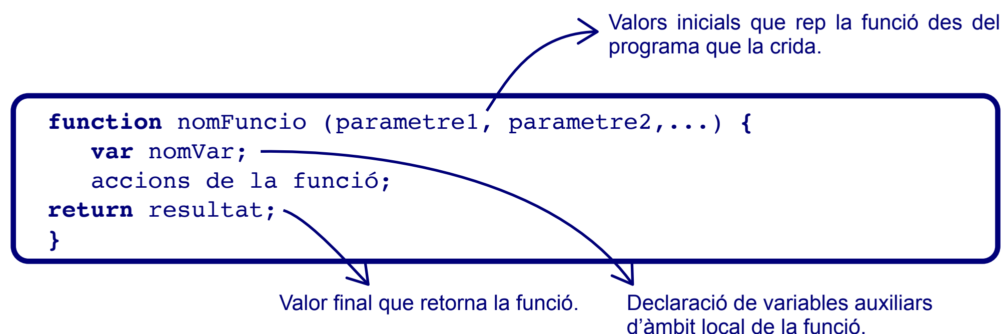
Com definim i utilitzem funcions en JavaScript?

Fixeu-vos que, a aquestes alçades, encara que potser no ens n'hàgim adonat, ja hem utilitzat alguna funció en JavaScript. Segurament la que més hem utilitzat és la funció **console.log**, que ens permet escriure un valor per la pantalla. Habitualment es fa servir per mostrar el resultat d'un algorisme, les dades de sortida.

És una de les funcions predefinides que el mateix llenguatge JavaScript proporciona. No coneixem el codi que contenen, però tant se val, les poden utilitzar cridant-les amb el seu nom i passant els paràmetres que necessiten, si és el cas.

En JavaScript també podem definir les nostres pròpies funcions, per allò que dèiem de dividir un problema complex en subproblemes més petits, cadascun resolt a partir d'una funció o bé per estalviar-nos de repetir el mateix codi una i una altra vegada en un algorisme.

Sintaxi per declarar funcions en JS



Fixeu-vos que en JS no s'indica el tipus dels paràmetres. És el mateix que passa quan es defineix una variable, JS dedueix el tipus del paràmetre en el moment que se li dona un valor.

Exemple

Suposem que som un concessionari que ven cotxes de segona mà. Estem fent plaques per a cotxes de diferents marques. Les plaques es posen en el lloc de la matrícula mentre el cotxe està esperant a ser venut.

Per poder calcular la llargada màxima de les plaques de cada remesa, ens cal un programa que identifiqui el nom de marca més llarg d'entre els cotxes que tenim en cada moment.

La propietat *length* aplicada a un nom ens indica el nombre de caràcters que té.

```
1  //Definim les dades d'entrada com a hardcode
2  const marca1 = "Ford";
3  const marca2 = "Mercedes";
4
5  //Variable del programa
6  var midaMaximaPlaca;
7
8  //Definim la funció
9  function midaNomMesLlarg (parametre1, parametre2) {
10    var nomLlarg; //variable local de la funció
11    if (parametre1.length > parametre2.length) {
12        nomLlarg = parametre1;
13     } else {
14       nomLlarg = parametre2;
15     }
16     return nomLlarg.length;
17 }
18
19 //Crida a la funció que calcula el nom més llarg
20 midaMaximaPlaca = midaNomMesLlarg(marca1, marca2);
21
22 //Retorn de les dades del programa
23 console.log ("Mida màxima de la placa: " + midaMaximaPlaca);
24
```

Trobareu una explicació detallada del funcionament de les funcions en JavaScript a la guia de JavaScript de Mozilla, com a les unitats anteriors.

Què passaria si...

...cridem a una funció que no existeix?

El programa s'atura perquè no troba cap funció amb el nom de la crida. Vegem-ho amb el PythonTutor.

JavaScript

```
1  //Definim les dades d'entrada com a hardcode
2  const marca1 = "Ford";
3  const marca2 = "Mercedes";
4
5  //Variable del programa
6  var midaMaximaPlaca;
7
8  //Crida a la funció que calcula el nom més llarg
9  midaMaximaPlaca = midaNomMesLlarg(marca1, marca2);
10
11 //Retorn de les dades del programa
12 console.log ("Mida màxima de la placa: " + midaMaximaPlaca);
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Done running (4 steps)

ReferenceError: midaNomMesLlarg is not defined

Frames

Global frame

midaMaximaPlaca	undefined
marca1	"Ford"
marca2	"Mercedes"

Què passaria si...

... no li passem els paràmetres que la funció necessita?

Com que la funció no rep les dades que necessita, en el moment de l'execució apareix un error perquè no es pot executar una de les instruccions en ser una dada indefinida. Fixeu-vos que el PythonTutor ens ajuda molt clarament a veure el flux d'execució i el valor de les variables tant de l'àmbit del programa com de la funció.

```
1  //Definim les dades d'entrada com a hardcode
2  const marca1 = "Ford";
3  const marca2 = "Mercedes";
4
5  //Variable del programa
6  var midaMaximaPlaca;
7
8  //Definim la funció
9  function midaNomMesLlarg (parametre1, parametre2) {
10    var nomLlarg; //variable local de la funció
11    if (parametre1.length > parametre2.length) {
12        nomLlarg = parametre1;
13     } else {
14       nomLlarg = parametre2;
15     }
16     return nomLlarg.length;
17 }
18
19 //Crida a la funció que calcula el nom més llarg
20 midaMaximaPlaca = midaNomMesLlarg(marca1);
21
22 //Retorn de les dades del programa
```

Edit this code

line that just executed
next line to execute

Global frame

midaMaximaPlaca	undefined
midaNomMesLlarg	
marca1	"Ford"
marca2	"Mercedes"

```
function midaNomMesLlarg(parametre1, parametre2) {
   var nomLlarg; //variable local de la funció
   if (parametre1.length > parametre2.length) {
      nomLlarg = parametre1;
    } else {
     nomLlarg = parametre2;
    }
    return nomLlarg.length;
}
```

midaNomMesLlarg

parametre1	"Ford"
parametre2	undefined
nomLlarg	undefined

Què passaria si...

… definim la funció però no la cridem des del programa principal?

Les instruccions de la funció no s’executen mai. El programa només executa 4 instruccions i, per tant, el resultat queda indefinit.

```
//Definim les dades d'entrada com a hardcode
const marca1 = "Ford";
const marca2 = "Mercedes";

//Variable del programa
var midaMaximaPlaca;

//Definim la funció
function midaNomMesLlarg (parametre1, parametre2) {
   var nomLlarg; //variable local de la funció
   if (parametre1.length > parametre2.length) {
       nomLlarg = parametre1;
    } else {
      nomLlarg = parametre2;
    }
    return nomLlarg.length;
}

//Retorn de les dades del programa
console.log ("Mida màxima de la placa: " + midaMaximaPlaca);
```

Print output: Mida màxima de la placa: undefined

Global frame	
midaMaximaPlaca	undefined
midaNomMesLlarg	→ function
marca1	"Ford"
marca2	"Mercedes"

Done running (4 steps)

Podeu provar aquests codis a la web de PythonTutor