# Treballant amb dades. Tipus bàsics

PID_00268289

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

Tercer edició: febrer 2021

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

# Continguts


## Treballant amb dades

**Tipus bàsics de dades**

- Variables
  - Concepte clau
- Sintaxi per a declarar variables i constants
- Regles per escollir el nom d'una variable
- Com podem assignar un valor a una variable?
  - Concepte clau
- Tipus bàsics de dades
- Barrejant tipus
- Funcions de conversió de tipus
- Expressions
- Regles per calcular expressions
- Exemples d'expressions
  - Concepte clau
- Comunicació amb l'exterior: entrada/sortida
- Com podem obtenir les dades d'entrada de l'algorisme?
- Com mostrar els resultats?
  - Concepte clau

**Com treballem amb els tipus bàsics de dades en JS?**

- Sintaxi per declarar els tipus bàsics de dades en JS
- Com assignem valors a variables i constants?
- Tipus bàsics de dades
- Què passaria si…?

# Treballant amb dades

## Tipus bàsics de dades

Tot i que encara no sabem com dissenyar algorismes, el que sí sabem és que hauran de tractar amb dades. De fet, han de processar i gestionar tota la informació necessària per resoldre el problema plantejat. Per tant, hem de saber representar i tractar les dades de partida del problema (dades d'entrada) i les que demana la solució del problema (dades de sortida). També necessitem saber com podem llegir les dades d'entrada i com podem retornar les dades de sortida.

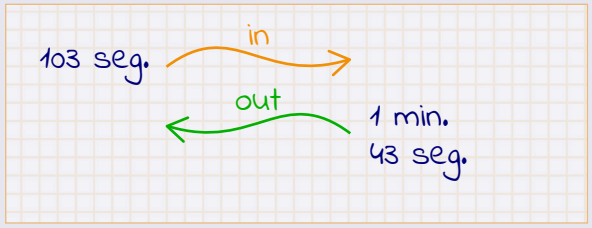
A aquestes alçades ja hem entès el problema a resoldre (pas 1), així que ja tenim clar què ha de fer l'algorisme, quines dades d'entrada tindrà i quines de sortida haurà de generar per resoldre el que ens demanen. També hem pensat com resoldre'l així que tenim decidida una estratègia de passos per solucionar el problema (pas 2). Ja podem **dissenyar l'algorisme** (pas 3), explicitar la solució que hem pensat en un algorisme tot utilitzant la notació algorísmica bàsica. Recordeu que també ho podríem fer amb un diagrama de flux.

### Exemple

Imaginem que ens truca un amic que fa temps que no veiem. Ens passem molta estona al telèfon, ens hem posat al dia. Just abans de penjar llegim a la pantalla "**103 SEC**". Què estrany, ens indica el temps de la trucada en segons? Quin mòbil més curiós. I això quants minuts són? Farem un algorisme que converteixi els segons en minuts i segur que així ho entendrem millor.

#### 1 Entenem el problema

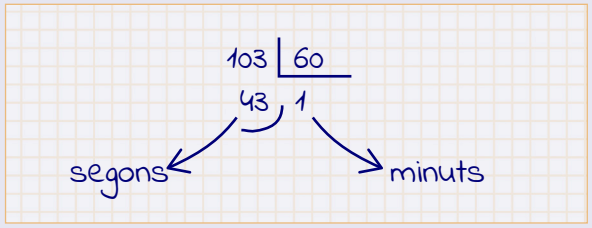
D'entrada, tindrem un número que seran els segons de la trucada i volem calcular i donar com a resultat altres dos números que siguin aquest mateix temps, però expressat en minuts i segons:




Ja tenim clar el problema a resoldre.

#### 2 Pensem com resoldre'l

Per calcular els minuts i segons que hi ha en 103 segons, ens cal agrupar els segons de 60 en 60 i així sabrem quants minuts exactes són, i la resta seran els segons que sobren. Això ho calculem amb una divisió:




Ja tenim la manera (l'estratègia) de trobar la solució al nostre problema.

#### 3 Dissenyem l'algorisme

Ara hem de plasmar aquesta estratègia en un algorisme. Dissenyar (escriure) un algorisme que indiqui clarament els pasos a seguir per resoldre el problema, tot utilitzant els elements algorísmics bàsics tant per representar les dades que calguin com per indicar les instruccions a seguir a cada pas.

Primera qüestió a resoldre: com representem els minuts? I els segons? Necessitem saber com representar les dades algorísmicament.

## Variables

Quan dissenyem l'algorisme, necessitem referenciar les dades que tracta d'alguna manera, representar-les algorísmicament.

### Concepte clau

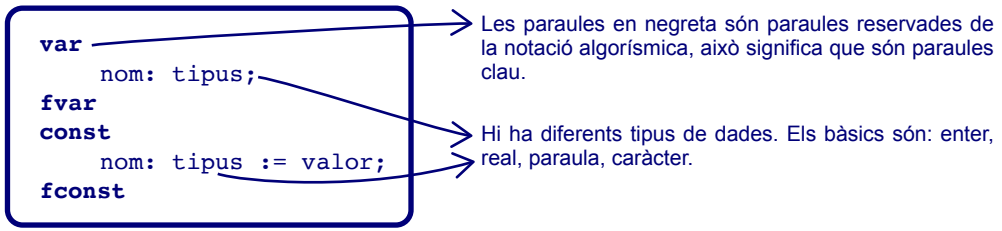
Recordeu que l'algorísmica gestiona tres elements de tractament de dades:
**ENTRADA:** indica a l'usuari que es pot introduir una dada, habitualment des del teclat.
**SORTIDA:** indica que es mostrarà una dada, habitualment a la pantalla.
**GUARDAR/RECORDAR:** enregistrar una dada perquè la pugui tractar l'algorisme.

Un algorisme necessita recordar la informació d'entrada, com el número 103 de l'exemple, per poder-la tractar (per utilitzar-la en la divisió, en aquest cas) i també la que va obtenint per solucionar el problema, les dades intermèdies (el quocient i la resta de la divisió, si seguim amb l'exemple). Les variables s'utilitzen per emmagatzemar aquesta informació. Una **variable** es pot veure com una caixa on es deixa cada una de les dades que va utilitzant l'algorisme.

Perquè un algorisme pugui treballar amb una variable necessita tenir un **nom** per poder-s'hi referir, saber quin **tipus** de dades pot contenir i el **valor** de la dada. Una variable sempre està definida per aquests tres elements. El valor guardat en una variable pot canviar ("variar") perquè l'algorisme el modifiqui amb els tractaments que faci.

A vegades, però, l'algorisme necessita treballar amb valors constants, que sabem del cert que no canviaran amb les operacions que faci l'algorisme, llavors utilitzarem **constants** per emmagatzemar aquest valor. Una constant és igualment una caixa on guardar una dada d'un tipus i que s'identifica amb un nom, però aquesta caixa "guarda" un **valor predeterminat i fix** durant tot l'algorisme, d'aquí el seu nom.

## Sintaxi per a declarar variables i constants




## Regles per escollir el nom d'una variable:

- El nom ha de ser simple, clar i significatiu, ha de tenir sentit i fer referència al valor que hi haurem de guardar. Això facilita la llegibilitat de l'algorisme a qualsevol persona que l'hagi de llegir i a nosaltres mateixos si passat un temps l'hem de modificar.
- Les variables s'han de declarar abans de fer-les servir, així l'algorisme sap a què ens estem referint.
- En el nom podem fer servir lletres i números.
- El nom no pot començar amb un número.
- El nom no pot tenir espais en blanc.
- No podem utilitzar com a nom paraules reservades de la notació (**var**, **const**, **fvar**, etc.).
- Les mateixes regles es fan servir per escollir els noms de les constants, les funcions, els algorismes, etc.

Aquesta convenció de nomenclatura és adoptada generalment per tots els llenguatges de programació com un costum que els bons programadors segueixen.

Serien noms correctes: *segonsInicials* o *segons_inicials*. I incorrectes o no recomanables: *s* (no s'entén el significat), *s_ini* (poc concret), *segons inicials* (incorrecte, conté espais en blanc).

## Com podem assignar un valor a una variable?

En un algorisme ja sabem com indicar quines variables utilitzarem i de quin tipus són. Per poder utilitzar una variable en un algorisme, ens cal poder donar-li (assignar-li) un valor, es a dir, saber, saber com guardar un valor a la caixa per poder-lo utilitzar en l'algorisme quan ens calgui recordar-lo.

Mireu si és important poder fer això que l'acció d'assignar un valor a una variable és una operació bàsica i fonamental sense la qual no es pot pensar cap algorisme. L'assignació es representa amb el símbol **:=** que es llegeix com "pren per valor".

Evidentment, a una variable només se li pot assignar un valor que sigui del mateix tipus que el seu.

## Concepte clau

Una **variable** és de fet un espai de la memòria de l'ordinador que emmagatzema valors. Ara, però, no ens preocupa com s'implementa físicament una variable a la memòria de l'ordinador, sinó com es pot declarar i fer servir en un algorisme per guardar les dades que s'hagin de tractar.

Les variables són un element clau en l'algorísmica, perquè ens permeten tractar amb les dades: podem guardar-hi valors, fer-hi operacions, intervenen en la presa de decisions quan hem de decidir entre una alternativa o altra, i ens ajuden a determinar quan ha de parar una iteració, com veurem més endavant.
Una variable es declara indicant els seus tres elements:

1. **Nom**
2. **Tipus** de dada que pot guardar
3. **Valor** que té en un moment donat

Recordeu que quan un valor no ha de variar durant les operacions de l'algorisme es fa servir una **constant** que s'identifica amb el nom i el valor concret. Per assignar un valor determinat a una variable s'utitliza l'assignació **:=**
que és l'acció més bàsica que es pot fer en un algorisme. Ja hem vist que el símbol d'assignació s'utilitza també per a indicar (assignar) el valor d'una constant.

Veiem en aquest vídeo ("The box variable activity" 7 min. aprox.) les variables en "acció" El vídeo utilitza un exemple en llenguatge Python, que té una sintaxi una mica diferent de la notació algorísmica que acabem de descriure; per exemple, per assignar enlloc d'utilitzar el símbol **:=** fa servir el símbol **=** però il·lustra molt bé com s'opera amb variables.

```
color1 := "vermell"
color2 := "verd"

temp := color1
color1 := color2
color2 := temp;
```

## Tipus bàsics de dades

Els algorismes han de manipular dades de tipus molt diversos que haurem de poder representar algorísmicament d'alguna manera. Per fer-ho, l'algorísmica té predefinits uns tipus bàsics de dades amb els que podem representar les dades amb les que calgui treballar per a solucionar el problema (dades d'entrada i dades de sortida, i també dades intermèdies que calgui utilitzar per fer els càlculs).

Els tipus bàsics ens permeten representar números (**enter** i **real**) i textos (**caràcter** i text o **cadena de caràcters**).

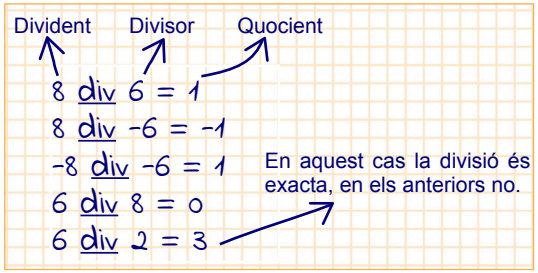
## Sintaxi per a declarar els tipus bàsics

```
var
   nom1 : enter;
   nom2 : real;
   nom3 : caracter;
   nom4 : cadena;
fvar
```

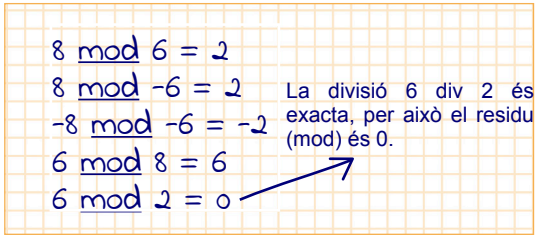
El tipus **ENTER** representa un número positiu o negatiu sense part decimal, per exemple, el 103.

Els enters els podem **sumar, restar, multiplicar i dividir**, també els podem **canviar el signe**. La notació algorísmica té operadors que permeten fer aquestes operacions. Però atenció, algorísmicament, aquestes operacions que podem fer amb els enters (que són les que després, en un llenguatge de programació, una màquina les sabrà interpretar) són **operacions internes**, és a dir, que el resultat ha de ser del mateix tipus que els operands, en definitiva, que donen com resultat un enter.

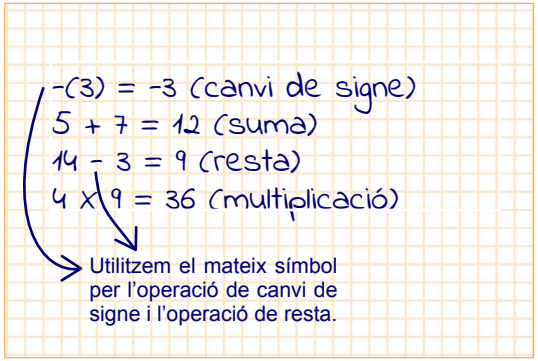
Així doncs, la divisió que podem fer amb els enters és la divisió entera, que es representa amb l'operador **div,** el qual ens retorna el quocient de la divisió entre dos enters, sense decimals. La divisió no és exacta en alguns casos.




I, com podem saber si la divisió és exacta o no? Coneixent el residu: si és 0, la divisió és exacta. A més, tenim un operador per calcular el mòdul (residu) d'una divisió entera: .

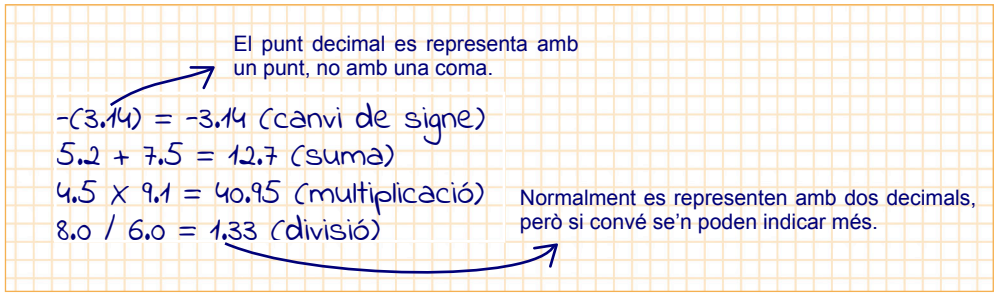



La resta d'operacions aritmètiques no tenen gaire misteri algorísmicament parlant, ja que funcionen tal com les usem matemàticament (tal com ho fem amb una calculadora).




El tipus **REAL** representa un número positiu o negatiu amb punt decimal, per exemple, 2.5.

Els reals també els podem sumar, restar, multiplicar i dividir, i també els podem canviar el signe. La notació algorísmica té operadors que permeten fer aquestes operacions ( **+**, **-**, **X**, **/** ). En aquest cas, algorísmicament, aquestes operacions algèbriques són tal com les entenem matemàticament.



## Exemple

Ara sí que ja podem resoldre el nostre problema. Recordeu? Passar els 103 segons a minuts i segons.

**Com representem els minuts i segons en algorismica?**

```
var
  segons_inicials : enter;
  minuts : enter;
  segons_restants : enter;
fvar

segons_inicials := 103;
minuts := (segons_inicials div 60);
segons_restants := segons_inicials mod 60;
```

La variable *segons_inicials* pren com a valor 103.

Les paraules que anomenen els operadors també són paraules reservades del llenguatge, per això s'indiquen en negreta en els algorismes (o subratllades si escrivim a mà).

O també, més sintèticament:

```
var
  segons_inicials, minuts, segons_restants : enter
fvar
```

Altres possibles noms correctes: *segonsInicials*, *s_ini*, encara que aquest és un xic imprecís i podria portar confusió. I seria poc recomanable el nom "*s*" perquè no aporta significat sobre el sentit de la variable.

Seria incorrecte: *segons inicials* perquè conté espais.

I aquí tenim l'algorisme que resol el nostre problema!

El tipus **CARÀCTER** representa qualsevol caràcter alfanumèric: una lletra, un dígit (xifra), un espai o un símbol, per exemple: la lletra 'a' o el dígit '3' o el signe d'interrogació '?'. Per distingir-los dels números, s'escriuen normalment entre cometes simples: 'a', '3', o '@'.

Amb els **caràcters** no podem fer operacions aritmètiques, òbviament; no els podem sumar, ni restar, etc., però sí que els podem concatenar.

No és el mateix el número enter 3 que el caràcter '3', encara que per nosaltres puguin tenir el mateix significat.

El tipus **CADENA DE CARÀCTERS** representa un conjunt de caràcters, un text. També s'escriuen entre cometes: "Això és un text", "una cadena de caràcters".

L'operació de **concatenació** la representem amb el símbol **+**:
"Això és un text" + ", " + "una cadena de caràcters" + "." generaria una frase molt més llarga: "Això és un text, una cadena de caràcters."

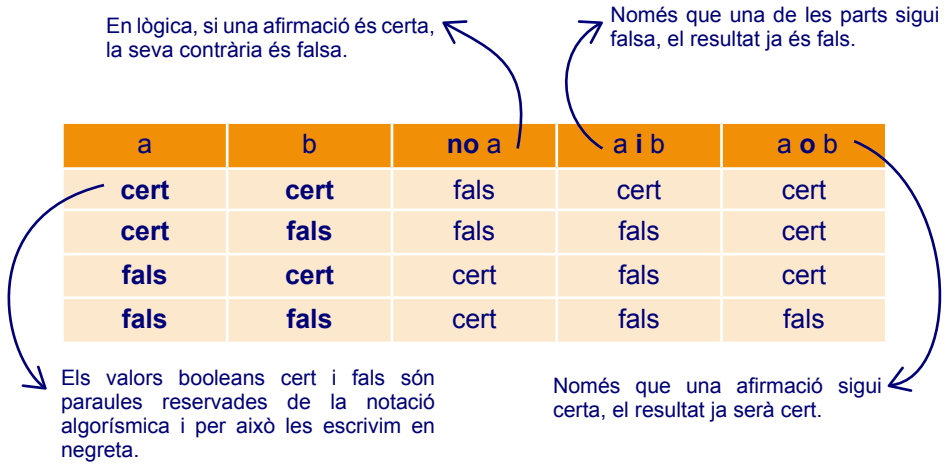
Amb les **cadenes de caràcters** tampoc no podem fer operacions aritmètiques però sí que les podem concatenar, unir, per formar frases més llargues.

Afegirem un darrer tipus bàsics importantíssim per l'algorísmica: el tipus **BOOLEÀ**. Aparentment, és un tipus que pot semblar molt poca cosa perquè és un tipus que només pot tenir dos valors: **cert** i **fals**. I ja està. Però malgrat ser un tipus peculiar, és imprescindible, per exemple, per poder prendre decisions en un algorisme.

En lògica, aquest tipus s'utilitza per indicar la certesa o falsedat d'un fet (d'un enunciat, diríem en lògica).

Amb els **valors booleans** es poden fer operacions, això és, combinar diferents condicions certes o falses entre si i veure si el resultat és cert o fals. Els valors booleans es combinen entre ells d'acord amb la lògica binària i amb tres operadors lògics: **no** (negació), **i** (es llegeix com a "i lògic") i **o** ("o lògic").

Podem veure totes les combinacions dels valors **cert** i **fals** amb aquestes operacions en una única taula que es coneix amb el nom de **taula de veritat**:




| a | b | **no** a | a **i** b | a **o** b |
|---|---|---|---|---|
| **cert** | **cert** | fals | cert | cert |
| **cert** | **fals** | fals | fals | cert |
| **fals** | **cert** | cert | fals | cert |
| **fals** | **fals** | cert | fals | fals |

Veiem en aquest vídeo ("Boolean: comnining keywords" 4 min. aprox.) una aplicació pràctica d'aquestes operacions.

El tipus booleà té el seu origen en l'àlgebra de Boole, ideada pel matemàtic Georges Boole al segle XIX. Treballa amb aquest dos valors lògics binaris, **cert** i **fals**, que sovint estan representats com a **1** i **0**, respectivament, i que són la base de la computació.

## Barrejant tipus

Segurament, per resoldre problemes caldrà en alguns casos comparar valors o saber si un número és més gran que l'altre, per exemple. El tipus booleà permet d'ampliar el ventall d'operacions dels altres tipus elementals amb **operacions externes** (en diem externes perquè donen com a resultat un valor d'un tipus diferent, en aquest cas booleà).

5 > 3 és cert
5 < 3 és fals
5 = 3 és fals
5 ≠ 3 és cert
5 ≥ 3 és cert
5 ≤ 3 és fals

Encara que sembli estrany, aquestes operacions també es poden aplicar als caràcters i fins i tot als booleans. Això passa perquè els caràcters estan ordenats alfabèticament: la 'a' està abans que la 'b' i així successivament. Per tant, quan diuen que 'a' és més petit que 'b', en realitat volem dir que la lletra 'a' va abans que la 'b' a l'alfabet.

'a' = 'b' és fals
'a' < 'b' és cert
'a' > 'b' és fals però 'A' < 'b' és cert
**cert** > **fals**

Per convenció, les minúscules són més grans que les majúscules.

Per convenció, també el valor **cert** es considera major que el valor **fals**.

I, per extensió, també podem fer el mateix amb les cadenes (tot avaluant caràcter a caràcter des del primer a l'últim i en el mateix ordre en què estan escrites:

"abcd" > "abcz" és fals
"Hola" < "adéu" és cert

La 'd' és més petita que la 'z'.

Les minúscules es consideren "més grans" que les majúscules.

Però compte! Quan fem càlculs i comparacions amb els tipus elementals, cal de recordar sempre que "No es poden sumar peres amb pomes", en definitiva, que podem sumar dos enters, multiplicar dos reals, comparar dues lletres, però no podem barrejar en una mateixa operació valors de tipus diferents.

| Incorrecte | Correcte |
|---|---|
| 5 + 7.3 | 5.0 + 7.3 |
| 5.0 + 4 | 5 + 4 |
| "el resultat és el número:" + 1 | "el resultat és el número:" + "1" |

## Funcions de conversió de tipus

A vegades, però, cal poder barrejar dades de diferents tipus. Per això, l'algorísmica disposa d'una llista de funcions predefinides de conversió de tipus, que canvien el tipus d'una dada a un altre tipus.

L'ASCII és un joc de caràcters que assigna valors numèrics (del 0 al 127, 7 bits de longitud) a les lletres, les xifres i els signes de puntuació. Gairebé tots els sistemes informàtics uttlitzen el codi ASCII per representar textos.

| | |
|---|---|
| **realAEnter** | Converteix un valor real a enter, deixant només la part entera del valor real.<br>**realAEnter** (3.5) retorna el valor enter 3. |
| **enterAReal** | Converteix un valor enter a real, afegint 0 a la part decimal.<br>**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.<br>**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.<br>**enterACaracter** (65) retorna el caracter 'A'. |
| **enterACadena** | Converteix un enter en una cadena de caràcters.<br>**enterACadena** (4) dona com a resultat '4'. |

Marquem les funcions de conversió en negreta perquè són paraules reservades de la notació. Això vol dir que no podem utilitzar-les per a noms de variables o funcions que inventem nosaltres perquè l'ordinador es confondria en interpretar-les.

Fixeu-vos que per a un ordinador no és el mateix el número enter 4 que el caràcter '4'. Nosaltres, en veure un 4, sabem interpretar que és el número 4, però per a un ordinador és un número només si no porta cometes.

## Expressions

A les instruccions per fer càlculs les anomenem expressions. Normalment es tracta d'operacions amb números (aritmètiques). I les podem indicar amb els operadors associats a cada tipus: **+**, **-**, **div**, etc. Però també podem fer expressions amb booleans, caràcters o cadenes.

Podem construir expressions amb valors (per exemple, 3 + 5) o amb variables (per exemple, *plugesGener + plugesFebrer + plugesMarç = plugesPrimerTrimestre* en què les quatre variables fossin de tipus enter).

Això sí, és important vigilar que els tipus de valors o de variables que fem servir a l'expressió siguin correctes, per exemple, per sumar calen dos enters o dos reals, perquè la suma necessita com a mínim dos sumands. I no podríem sumar dos booleans, seria una operació incorrecta: no(7), 5+**cert**, 6/2.

## Regles per calcular expressions

- Quan en una expressió hi ha diversos operadors, es comença a avaluar-los tot seguint la següent taula, on l'operador més prioritari és el de més amunt:

| canvi de signe | no | | | | |
|---|---|---|---|---|---|
| x | / | div | mod | | |
| + | - | | | | |
| = | ≠ | < | > | ≤ | ≥ |
| i | | | | | |
| o | | | | | |

- Les operacions que hi ha entre parèntesi tenen prioritat respecte a les de fora.
- Els operadors de la mateix fila, tenen la mateixa procedència. La precedència determina l'ordre en que s'apliquen quan avaluem una expressió, quina operació fem primer i quina després. S'avalua primer el que es troba més a l'esquerra de l'expressió concreta que estem avaluant:

2.0 + 8.0 / 2.0 +10.0 x 5.0
2.0 + 4.0 + 10.0 x 5.0
2.0 + 4.0 + 50.0
6.0 + 50.0
56.0

## Exemples d'expressions

Aquests tres expressions NO seran correctes:

```
3 x 5 cert
```
→ No té cap sentit, podríem calcular 3x5 i donaria 15 però 15 cert no té cap sentit, no hi ha cap operació que lligui l'enter 15 i el booleà **cert.**

```
const
  const1 : real := 1.0;
fconst
var
 var1: enter;
fvar
const1 x enter1
```
→ És incorrecte perquè intentem multiplicar un real amb un enter.

```
var
  r1, r2: real;
fvar
(r1/r2) 4.5
```
→ 1r: (r1/r2) dona com a resultat un real

2n: real 4.5 no significa res, falta l'operador! L'ordinador no sap iinterpretar que si no hi ha cap operador és una multiplicació.

Aquestes quatre expressions serien correctes:

```
var
 c, d, e : enter;
fvar
c div d x e
```
→ Primer calculem *c* div *d* i dona com a resultat un enter, després el multipliquem per *e* i dona un altre enter com a resultat final.

```
5 = 4 - -(1)
```
→ Calculem:
1r: -(1) dona -1
2n: - -1 dona +1
3r: 4 +1 dona 5
4t: 5 = 5 dona **cert**

```
3.5 > 2.8
```
→ Donaria com a resultat **cert**

```
'a' < 'A'
```
→ Donaria **fals** perquè, segons la codificació ASCII, les minúscules són més grans que les majúscules.

## Concepte clau

Els **tipus bàsics de dades** permeten representar dades simples. També s'anomenen tipus elementals.

En algorísmica, els tipus bàsics per representar números són el tipus **enter** i el tipus **real,** i per representar textos el tipus **caràcter** i el tipus **cadena de caràcters** (*string* en anglès).

Amb els tipus numèrics es poden fer les **operacions** habituals de l'aritmètica matemàtica: **suma, resta, multiplicació** i **divisió,** i també se'ls pot **canviar el signe** (de positiu a negatiu o viceversa), sempre tenint en compte que no es poden barrejar valors de diferents tipus en una mateixa operació (són **operacions internes**, això és que donen com a resultat un element del mateix tipus). Per això, en el cas dels enters, utitlitzem la divisió entera (**div**) i el mòdul (**mod**) que ens retorna el residu d'una divisió.

Aquests operadors són elements predefinits de la notació algorísmica que hem definit i, per tant, els podem utilitzar en els algorismes. Per això s'escriuen en negreta: són paraules reservades de la notació algorísmica.

Atès que no es poden barrejar valors de diferents tipus en una mateixa operació, a vegades necessitem utilitzar les **funcions de conversió de tipus,** que ens permeten passar un valor d'un tipus a un altre: d'enter a caràcter, a cadena o a real, de caràcter a enter quan això tingui sentit, i de real a enter. Són funcions predefinides en la notació algorísmica, així que no cal preocupar-se de com funcionen i les podem utilitzar directament. Per això també les escrivim en negreta.

Un tipus imprescindible i dels més utilitzats en l'algorísmica és el **booleà**, que només té dos valors possibles: **cert i fals**. El seu origen està en l'àlgebra de Boole.També se l'anomena tipus lògic. Amb els booleans podem fer tres operacions lògiques: **no** (negació), **i** (conjunció) i **o** (disjunció). La taula de veritat detalla el resultat d'aquestes operacions.

El tipus booleà és imprescindible en algorísmica perquè ens cal per poder decidir si executar unes instruccions o unes altres, i també per saber fins quan cal estar repetint unes instruccions concretes, per exemple.

També ens permet d'ampliar el ventall d'operacions dels altres tipus elementals amb **operacions externes** (en diem externes perquè donen com a resultat un valor d'un tipus diferent, en aquest cas booleà): podem comparar números, saber si un és més gran que l'altre, si és igual, si és menor o si és diferent.

A totes aquestes instruccions amb les quals fem operacions diverses, en algorísmica les anomenem **expressions**.

## Comunicació amb l'exterior: entrada/sortida

Un algorisme és un conjunt d'instruccions que resol un problema. Per fer-ho li calen dades, són les dades d'entrada. I, un cop resolt el problema, ha de comunicar el resultat, que són les dades de sortida.

## Com podem obtenir les dades d'entrada de l'algorisme?

L'algorisme pot aconseguir les dades que necessita per resoldre el problema de diferents maneres:

- Demanar-les a l'usuari i llegir-les des del teclat a través d'instruccions de l'algorisme.
- Escriure-les directament en l'algorisme com a constants (d'això, en anglès se'n diu *hard code*).
- Llegir-les d'un fitxer on estiguin emmagatzemades.

Per demanar dades a l'usuari, ens calen funcions d'entrada/sortida que ens permetin escriure per la pantalla el que li vulguem demanar i llegir del teclat el que l'usuari ens escriu. Per això, els llenguatges disposen de funcions predefinides. Algorísmicament les indiquem així:

```
escriurePantalla (valor)
v:= llegirTeclat();
```

Visualitza a la pantalla el valor que li passem com a paràmetre.

Llegeix del teclat un valor i el guarda a la variable.

L'opció de llegir dades des de fitxers no la veurem, de moment.

## Com mostrar els resultats?

La funció d'escriptura per pantalla ens servirà també per mostrar a l'usuari els resultats del nostre algorisme. Igualment, podríem deixar els resultats en un fitxer, però aquesta opció no la veurem, per ara.

Habitualment, hi ha diferents funcions de lectura i escriptura per a cada tipus de dada, algorísmicament ho simplifiquem en una de sola per a tots els tipus. Quan programem en un llenguatge concret cal estar atents a les seves especificitats i triar la que convingui segons el cas.

### Exemple

Reprenem ara l'algorisme que transformava els segons d'una trucada telefònica a minuts i segons. El recordeu?

```
var
  segons_inicials : enter;
  minuts : enter;
  segons_restants : enter;
fvar

segons_inicials := 123;
minuts := (segons_inicials div 60);
segons_restants := segons_inicials mod 60;
```

Hem fet servir la tècnica de *hard code* per assignar la dada d'entrada a l'algorisme.

O també, més sintèticament:

```
var
  segons_inicials, minuts, segons_restants : enter
fvar
```

## Exemple

Ara generalitzem-ho per a qualsevol valor d'entrada, que demanarem a l'usuari per pantalla:

```
var
  segons_inicials : enter;
  minuts : enter;
  segons_restants : enter;
fvar

escriurePantalla("Indica els segons a convertir si us plau:");
segons_inicials := llegirTeclat();
minuts := (segons_inicials div 60);
segons_restants := segons_inicials mod 60;
escriurePantalla("El resultat és: "+ enterACadena(minuts) + "minuts i " + enterACadena(segons_restants) + "segons");
```

Un altre possible nom correcte: "*segonsInicials*".

Serien poc recomanables *s_ini* perquè és un xic imprecís i portaria a confusió i *s* perquè no aporta significat sobre el sentit de la variable".

Seria incorrecte *segons inicials* perquè conté espais.

Escribim per pantalla aquesta frase.

Llegim del teclat la dada d'entrada.

Variables que contenen els segons restants.

Concatenem els valors i es mostrarà per pantalla el següent text:
"El resultat és: 1 minuts i 43 segons".

L'operador d'assignació també és un símbol reservat de la notació algorísmica.

Variable que conté els minuts resultants.

Aquí caldria escriure "minuts" o "minut" segons si el valor de la variable *minuts* és 1 o més. Ho aprendrem a fer més endavant amb les estructures algorísmiques. El mateix passa amb la variable *segons_restants*.

O també, més sintèticament:

```
var
  segons_inicials, minuts, segons_restants : enter;
fvar
```

## Concepte clau

Les **funcions predefinides d'entrada/sortida** permeten que l'algorisme rebi les dades que necessita per resoldre el problema i també que retorni els resultats. Habitualment els llenguatges de programació disposen de diverses funcions per a cada tipus de dada bàsica: per llegir enters, per llegir reals, per escriure un caràcter, etc.

Altres maneres d'aconseguir les dades d'entrada són definir-les dins de l'algorisme o el programa (*hard code*) o llegir-les d'un fitxer extern. Igualment, el resultat es podria emmagatzemar en un fitxer en comptes de mostrar-lo per pantalla. Segons el cas, tindrà sentit fer-ho d'una manera o una altra.

# Com treballem amb els tipus bàsics de dades en JS?

La gramàtica del llenguatge JS té les seves peculiaritats en relació amb els tipus de dades. Per exemple, hi ha dues maneres de declarar variables (**let** i **var**). Per començar a programar, utilitzarem les opcions més senzilles i més recomanables tot atenent al que hem vist algorísmicament.

## Sintaxi per declarar els tipus bàsics de dades en JS

```
var nomVariable;
```

En JS no s'indica el tipus de la variable a la seva delcaració.

A diferència de la definició de variables que hem vist en la notació algorísmica, una característica del JS és que és un llenguatge "feblement tipat", això vol dir que no cal declarar el tipus d'una variable quan es defineix. El tipus s'assumeix dinàmicament en el moment que se li assigna un valor, és a dir, que quan li donem un valor a la variable el programa sap de quin tipus és la variable.

De fet, en JS ni tan sols és obligatori declarar les variables abans de fer-les servir, tot i que és una pràctica gens recomanable.

### Exemple

```
1 //Variables
2 var segons_inicials = 103;
3 var minuts;
4 var segons_restants;
5 //Constant
6 const segonsMinut = 60;
7 //Línia de comentari. No s'executarà.
8
```

JS és *case-sensitive*, això vol dir que distingeix entre minúscules i majúscules, per tant, el nom "segons_inicials" serà diferent de "segons_Inicials".

Així es defineixen les constants en JS.

Es recomanable que les instruccions acabin amb punt i coma. Encara que des de la consola ens funcionarà habitualment sense punt i coma, penseu que no és l'únic lloc des d'on s'executa JS, així que val més posar-lo sempre.

És un molt bon costum comentar el codi, en facilita la llegibilitat i les posteriors modificacions. Utilitzem la doble barra per indicar que és una línea de comentaris i que, per tant, no s'ha d'executar.

## Com assignem valors a variables i constants?

Per assignar valors a variables o constants utilitzarem el símbol **=**.

```
nomVariable = valor;
```

Llavors, no podrem utilitzar el símbol **=** per comparar valors perquè seria confús. Per comparar valors s'utilitza el símbol **==**.

## Tipus bàsics de dades

- El JS treballa només amb tres tipus bàsics de dades: **Number**, **String** i **Boolean**. Per tant, no distingeix entre enter i real ni entre caràcter i cadena de caràcters. Això sovint ens simplifica els càlculs.

- En el cas del **tipus Number**, els símbols dels operadors aritmètics són: **+**, **-**, **x**, **/** i **%,** que correspon a l'operador **mod** algorísmic (calcula el mòdul d'una divisió). La coma decimal es representa amb un "**.**". Els operadors de comparació son: **==** (igual), **!=** (diferent), **>** (major), **<** (menor), **<=** (menor o igual) i **>=** (major o igual).

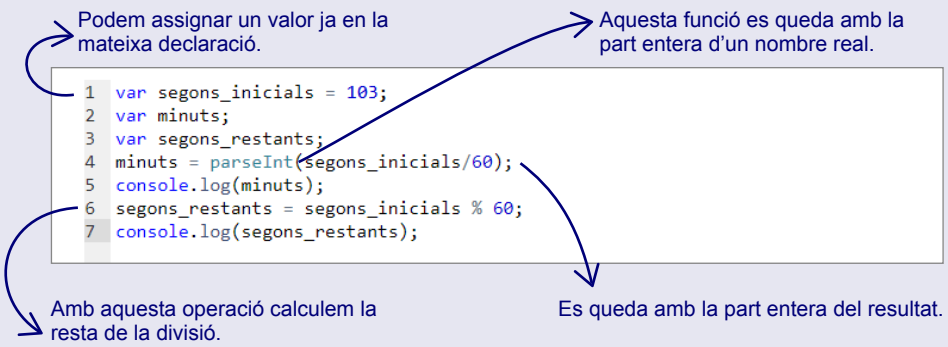
  I comptem també amb les següents funcions de conversió de tipus: **parseInt()** i **parseFloat(),** que converteixen un valor en un enter o en un real, respectivament.

- En el cas del **tipus Boolean**, els símbols dels operadors són: **&&** (la I lògica), **II** (la O lògica) i **!** per a la negació. I els valors **true** i **false** s'escriuen en minúscula.

- El **tipus String** es representa entre cometes dobles, per exemple: "Joan". Si concatenem un Number amb un String amb l'operador de concatenació **+**, JS fa la conversió automàtica de l'enter a String, per això no cal tenir funcions predefinides de número a caràcter.

### Exemple

Seguint amb l'exemple anterior, aquí tenim la seva codificació en JS.



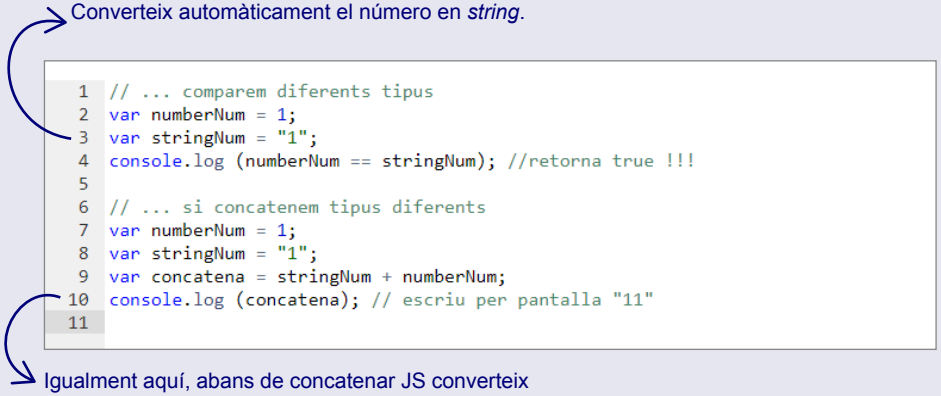

```
1 var segons_inicials = 103;
2 var minuts;
3 var segons_restants;
4 minuts = parseInt(segons_inicials/60);
5 console.log(minuts);
6 segons_restants = segons_inicials % 60;
7 console.log(segons_restants);
```

En JS s'utilitza la funció **console.log()** per mostrar per pantalla el valor d'una variable o un literal. I la funció **window.prompt()** per demanar i llegir un valor des de la pantalla.

## Què passaria si...?

Combinant els tres tipus, podem trobar-nos amb resultats una mica sorprenents.

### Exemple




```
1  // ... comparem diferents tipus
2  var numberNum = 1;
3  var stringNum = "1";
4  console.log (numberNum == stringNum); //retorna true !!!
5
6  // ... si concatenem tipus diferents
7  var numberNum = 1;
8  var stringNum = "1";
9  var concatena = stringNum + numberNum;
10 console.log (concatena); // escriu per pantalla "11"
11
```

La pràctica i el coneixement d'un llenguatge de programació concret ens faràn conèixer aquests comportaments automàtics del llenguatge que inicialment poden resultar estranys. Recordeu que cada llenguatge té les seves particularitats i implementa de manera diferent els principis algorísmics que són més universals. Una cosa semblant passa amb les llengües.

La fase 4 d'implementar consisteix precisament en trobar les instruccions, operadors i funcions concretes de llenguatge més adients per expressar l'algoritme de resolució que hem pensat,tenint en compte les particularitats concretes de cada llenguatge. No es tracta, en absolut, d'una simple traducció de la notació algorítmica al llenguatge de programació, sinó més aviat, de la concreció de les idees algorítmiques de resolució del problema en el llenguatge de programació escollit. Hauria de ser un pas no massa complicat donat que tots els llenguatges estructurats comparteixen unes característiques comuns. Més endavant, la destresa en un llenguatge de programació ens permetrà treure'n el millor partit del llenguatge per fer programes més eficients.

Trobareu una explicació detallada del funcionament dels tipus bàsics en JavaScript a la guia de JavaScript de Mozilla.

Podeu provar aquests codis a la web de PythonTutor.