Començant a programar

PID_00268291

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

Segona edició: juliol 2020

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

Continguts

Començant a programar

Què és programar?

Abans d'entrar en matèria cal saber de què parlem.

Un **programa** és un conjunt d'instruccions que un ordinador sap executar per fer una tasca. Un ordinador pot semblar una màquina molt intel·ligent però, de fet, no sap "pensar" en el sentit que ho fem les persones, només sap executar un conjunt concret d'instruccions simples, això sí, de manera increïblement ràpida i amb molta precisió. Som els programadors els que pensem quines són aquestes instruccions que cal seguir per fer una tasca concreta.

I, de fet, pensar aquestes instruccions a seguir és **pensar computacionalment** i, escriure-les de manera que l'ordinador les entengui és **programar**.

Així doncs, pensar computacionalment no és el mateix que programar. Tampoc no és pensar com un ordinador, atès que els ordinadors no saben pensar, saben executar instruccions. El **pensament computacional** és el que ens permet a les persones determinar de quina manera, amb quins passos es pot fer una tasca o resoldre un problema.

Exemple

Imaginem que se'ns ha espatllat el forn i ens truca un tècnic per venir a casa a arreglar-lo. Ens pregunta com arribar-hi des del seu taller. Hem de pensar els camins possibles i decidir quin és el millor per arribar a casa. El millor pot ser el més ràpid, el més curt o el més fàcil, al capdavall, el camí amb el qual té menys possibilitats de perdre's. I aquestes tres condicions poden no coincidir en la mateixa opció. Cal tenir clar en funció de quins criteris escollirem una o altra opció. Un cop decidit el millor camí, has de donar-li les indicacions per arribar de la manera més clara i concisa possible.

Quan penses en les rutes que hi ha i quina seria la millor estàs utilitzant el pensament computacional, i quan dones al tècnic les indicacions per arribar a casa teva, estàs dissenyant l'algorisme de la solució. Si, en comptes de donar les instruccions per arribar a una persona, les haguéssim de donar a un robot, per exemple, llavors hauríem de fer un programa en algun llenguatge de programació que pogués entendre el robot. Fer el programa no seria més que codificar l'algorisme amb les indicacions escrivint-lo en el llenguatge concret.

Un **algorisme** és, doncs, un conjunt de passos per fer una tasca i un **programa** és un algorisme que ha estat escrit en un llenguatge que pot entendre un ordinador.

> "An algorithm is a set of rules for getting a specific output from a specific input. Each step must be so precisely defined that it can be translated into computer language and executed by machine." — Donald Knuth, un dels referents de l'algorítimica (1977)

Un algorisme seria, per exemple una recepta de cuina o les instruccions per a muntar un moble pas a pas. Si la seqüència d'instruccions és correcta i les segueixes fil per randa, arribaràs al resultat esperat, fer una truita de patates o muntar una estanteria. Vivim, doncs, envoltats d'algorismes.

Exemple

Per exemple, aquest és possiblement l'algorisme que executa un programa de la rentadora.

- Agafar l'aigua.
- Agafar el sabó.
- Escalfar l'aigua a 40 graus.
- Repetir durant 20 minuts. → Rentar.
  - Fer voltes al tambor.
- Expulsar l'aigua.
- Repetir quatre cops. → Cada volta és una esbandida de roba.
  - Agafar aigua.
  - Repetir durant 10 minuts.
    - Fer voltes al tambor.
  - Expulsar l'aigua.
- Repetir durant 1 minut. → Centrifugar.
  - Fer voltes al tambor molt ràpidament.

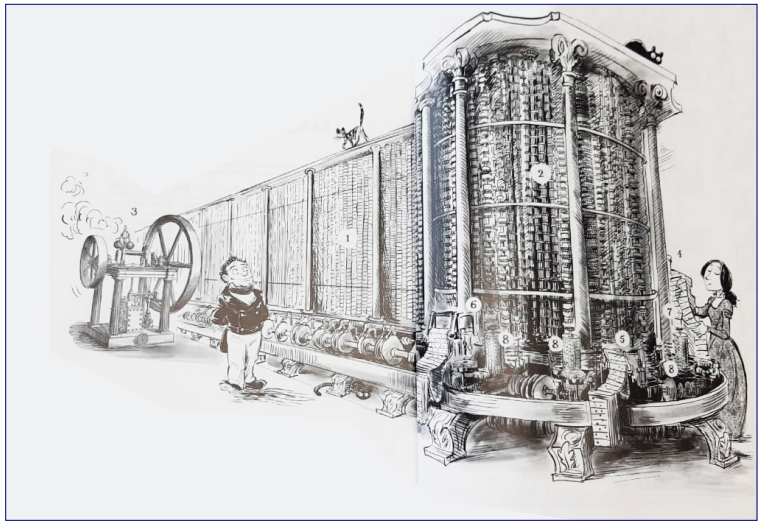
Això sí, les instruccions de l'algorisme han de ser precises, clares i completes. També correctes i eficients.

Va molt bé treballar amb el concepte d'algorisme perquè tot el que expliquem relacionat amb l'algorísmica és vàlid i, en general, ho podrem aplicar després a qualsevol llenguatge de programació. També, perquè expressar les instruccions en notació algorísmica és més senzill i no cal estar pendents de la sintaxi més estricta dels llenguatges de programació. Ens permet primer centrar-nos en com resoldre el problema sense haver-nos de preocupar de com expressar-ho exactament en un llenguatge de programació concret.

Una mica d'història

De fet, els primers algorismes van ser desenvolupats fa milers d'anys, molt abans que existissin els ordinadors. Un dels algorismes més famosos és l'**algorisme d'Euclides,** per calcular els divisors d'un número, que és del 300 a.C.

Els algorismes ens han acompanyat sempre en forma de seqüències d'instruccions que les persones segueixen per fer una tasca. Fins que el 1837, **Charles Babbage** va imaginar una màquina capaç d'executar algorismes (de seguir instruccions) mecànicament. Ell pensava en una màquina capaç de fer càlculs matemàtics. Més tard, **Ada Lovelace**, filla de Lord Byron i matemàtica brillant, va quedar meravellada per la idea de la **màquina analítica** de Babbage i va escriure el 1842 un programa per la màquina de Babbage, i també va pensar de donar-li les instruccions a través de targetes perforades. Per això, se la considera la primera programadora de la història. Ja va imaginar en el seu dia que els ordinadors (que encara ni existien) s'utilitzarien un dia per a crear música i art. Una visionària, sens dubte! I, amb ella, arriba la programació.



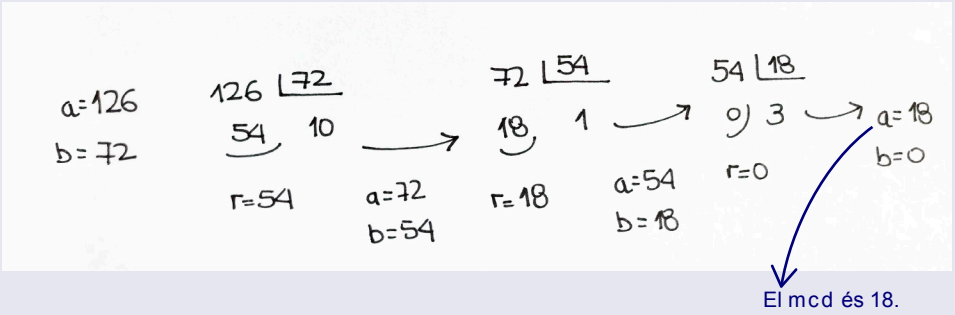
Exemple

L'algorisme d'Euclides és el mètode que es fa servir per a calcular el màxim comú divisor (mcd) entre dos números diferents de zero.

- Dades d'entrada **a** i **b.**
- L'algorisme:
  - si a < b, intercanviar a i b (a ↔ b)
  - mentre b ≠ 0 repetiu les tres instruccions següents:
    - r ← residu (la resta de la divisió entera) de **a** per **b** (doneu a **r** el valor del residu de **a** per **b**).
    - a ← b (el nou valor de **a** és l'antic valor de **b**).
    - b ← r (el nou valor de **b** és el valor de **r**).
- El resultat és **a** (el seu últim valor).

Veiem per a què ens serveix calcular el màxim comú divisor entre dos nombres amb un exemple extret d'un llibre de matemàtiques de 1r de l'ESO. Imaginem que tenim una col.lecció de vehicles en miniatura (72 cotxes i 126 motos) i acabem de comprar unes prestatgeries per guardar-los. Però volem col·locar-los tots de tal manera que hi hagi la mateixa quantitat de vehicles a cada prestatgeria i, a més, per no utilitzar massa espai, col·locar el màxim nombre de vehicles a cadascuna i sense barrejar motos i cotxes en una mateixa estanteria. Els col·leccionistes som així, tenim les nostres "manies".

Calculem el mcd (126,72):



El mcd és 18.

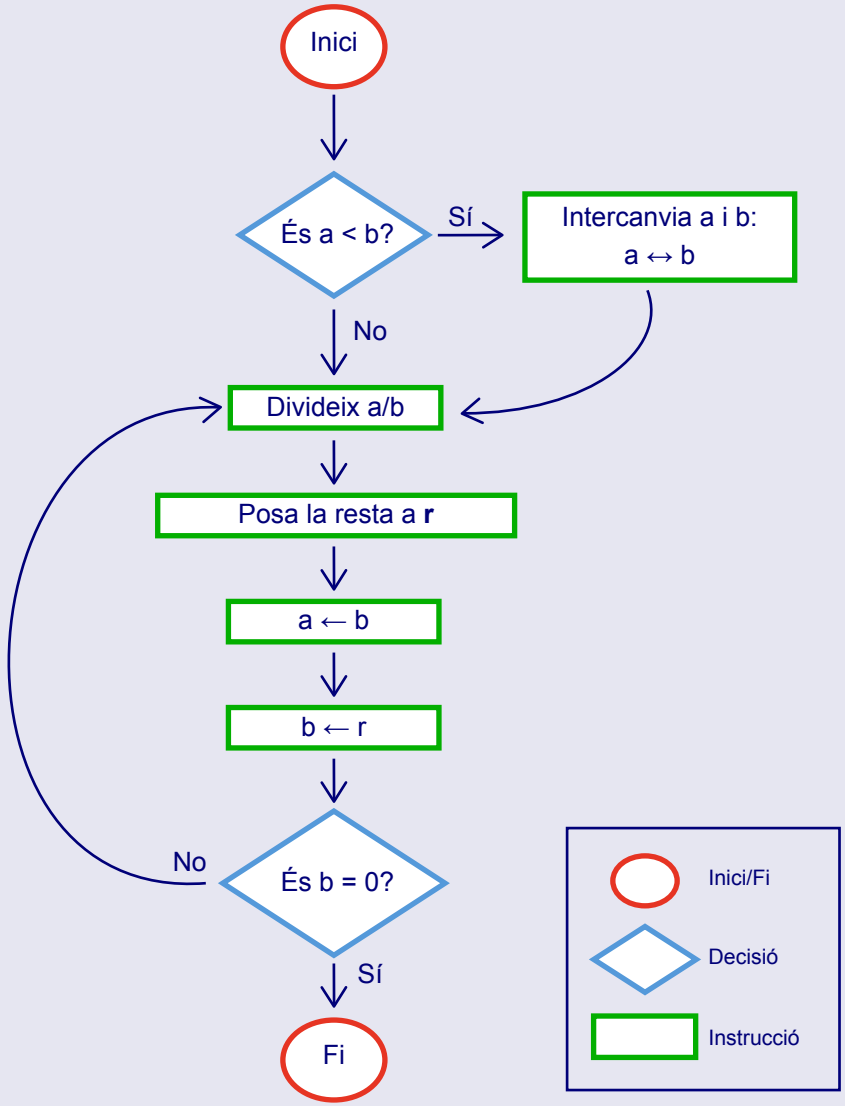
Per tant, el mcd (126, 72) = 18.

Hem de posar 18 vehicles (cotxes o motos) a cada prestatge. I, atès que tenim 126+72= 198 vehicles en total, ens caldran 198/18= 11 prestatges, 7 prestatges per a cotxes i 4 prestatges per a motos.

Els algorismes es poden descriure textualment, seguint alguna notació concreta com veurem més endavant i també gràficament, per exemple amb un **diagrama de flux.**

Exemple

L'algorisme d'Euclides representat com a diagrama de flux:

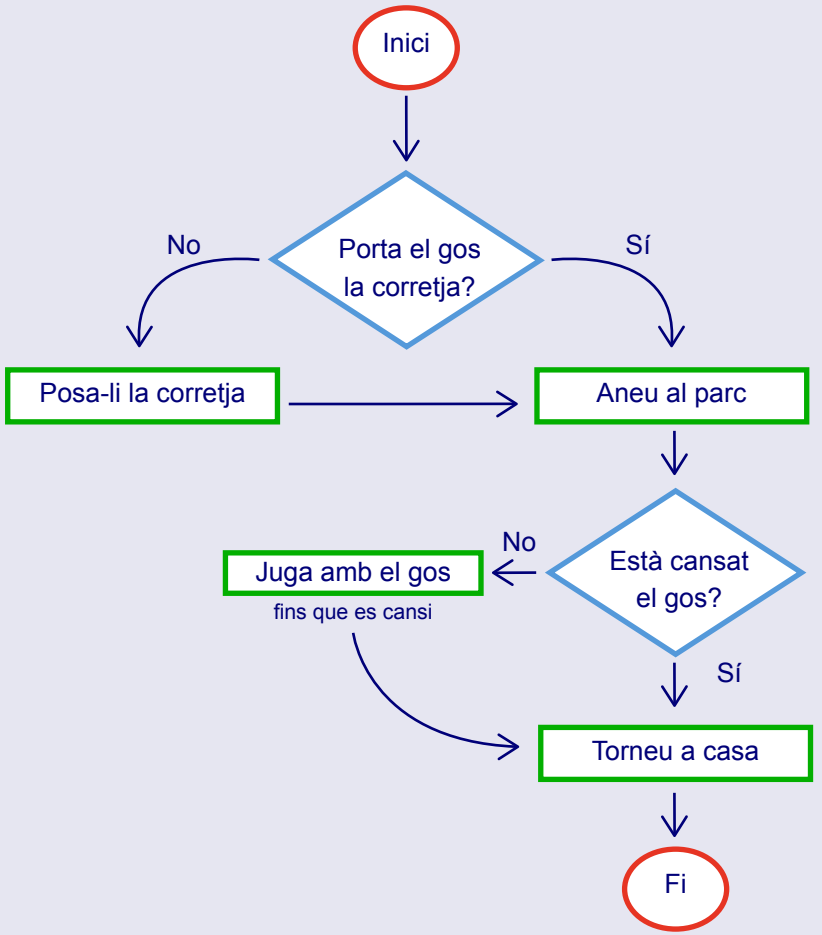


Exemple

Imagineu que heu de demanar al vostre fill que vagi a passejar el gos. És el primer cop que ho fa. Li donareu unes instruccions:

"Si el gos no porta la corretja, posa-li. Aneu al parc. Juga amb el gos fins que es cansi. Després, torneu a casa".

Que, representat com a diagrama de flux, seria:



Començant a programar

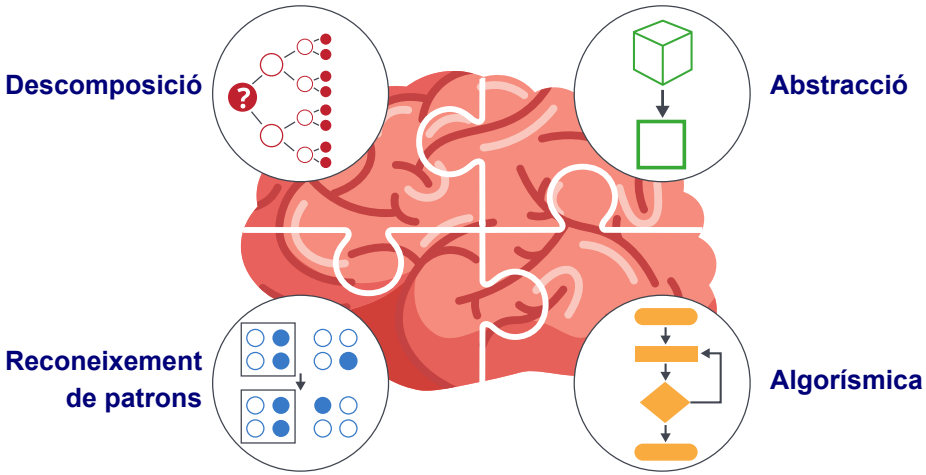
Pensament computacional

Els ordinadors ens ajuden a resoldre problemes. Però per resoldre un problema primer cal comprendre'l i pensar possibles solucions. En això ens ajuda aplicar el pensament computacional, que ens permet primer entendre un problema complex i, després, solucionar-lo pas a pas de manera suficientment clara i concisa perquè ho pugui entendre algú altre i també un ordinador.

Es fonamenta en quatre tècniques, bàsicament. Totes elles ajuden a gestionar la complexitat dels problemes:

Descomposició	Dividir un problema complicat, que no sabem resoldre directament, en parts més petites i més fàcilment solucionables.
Reconeixement de patrons	Buscar similituds entre diferents problemes per aprofitar el que se sap per solucionar-ne un per aplicar-ho a l'altre.
Abstracció	Simplificar, ignorant els detalls innecessaris per quedar-nos amb l'essència i fixar-nos només en allò és rellevant en relació al problema.
Algorísmica	Dissenyar els passos un a un per a solucionar un problema. Per molt màgic que pugui semblar, tot el que vulguem que faci un ordinador es pot escriure a partir d'unes instruccions concretes i senzilles que es combinen entre elles en forma de seqüència, repetició i selecció.

Tenir destresa en aquestes tècniques ajuda a programar perquè facilita dividir un problema complex en un seguit de problemes més senzills i gestionables (descomposció) que es poden anar resolent un a un individualment, tenint en compte com s'han solucionat problemes similars anteriorment (reconeixement de patrons) i centrant-nos només en els detalls rellevants del problema (abstracció). Tot seguit, es pot definir el pla d'acció, entès com a passos a seguir per resoldre cadascun dels problemes (fer l'algorisme). Finalment, aquests passos seran l'esquema de base per codificar el programa en un llenguatge de programació concret amb el qual un ordinador resoldrà el problema.



El **pensament computacional** es va fer popular després d'un article de Jeannette Wing de 2006, que el postulava com una habilitat fonamental per a totes les disciplines, més enllà de la programació. El terme, però, el va fer servir per primer cop Seymour Papert ja el 1980, pràcticament tres dècades abans.

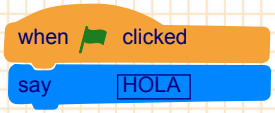
Les tècniques del pensament computacional són útils també en moltes altres disciplines i de fet en la nostra vida diària, sempre que calgui resoldre problemes complexos. D'aquí que es consideri una de les competències del segle XXI.

Començant a programar

Llenguatges de programació

A l'hora de programar, podem escollir entre una pila de llenguatges, alguns més adequats per a segons quins tipus de programes o aplicacions que d'altres. Alguns més sofisticats i complicats d'aprendre, altres més eficients per fer segons què.

C	Un dels més utilitzats. És molt potent per exemple fer programari que necesita funcionar ràpid, com els sistemes operatius. Però també és complicat d'aprendre.	`#include <stdio.h>`
		`main()`
		`{`
		`  printf("HOLA");`
		`}`

BASIC	Un dels més antics, ja no s'utilitza.	`10 PRINT "HOLA"` `20 GOTO 10`
SCRATCH	És visual, en comptes d'escriure instruccions s'han d'anar combinant diferents blocs de codi per construir-les. Per això és molt adequat per aprendre a programar, especialment per als nens.	

JAVA	És molt versàtil ja que permet que un programa funcioni amb diferents sistemes operatius i pugui funcionar en ordinadors, telèfons mòbils i tauletes sense cap canvi.	`class Hola, WorldApp{`
		`  public static void main (String{}args){`
		`  System.out.println("HOLA");`
		`  }`
		`}`

JAVASCRIPT	És molt útil per crear codi que funcioni dins un HTML, per això s'utilitza per donar interactivitat a les pàgines web i per demanar informació a l'usuari, per exemple. Actualment també es poden crear servidors, aplicacions web i d'escriptori que funcionen amb JavaScript.	`console.log('HOLA');`
PYTHON	És més recent, molt popular i versàtil. S'utilitza per a diferents propòsits. És simple i relativament senzill d'aprendre.	`print("HOLA")`

PHP	S'utilitza juntament amb l'HTML per dissenyar pàgines web.	`<?php`
		`echo "HOLA";`
		`?>`

Els llenguatges de programació, tal com passa amb les llengües, són "vius", evolucionen constantment. Així que cal saber sempre on trobar la documentació de referència, les llibreries existents, els manuals més actuals i els fòrums de dubtes per estar al corrent dels darrers canvis.

Començant a programar

Ser un bon programador

Convertir-se en un bon programador, en un expert, necessita temps, com passa en qualsevol disciplina. I necessita molta pràctica i entrenament. A programar s'aprèn programant: provant molt, equivocant-se, corregint errors, preguntant, tafanejant i analitzant altres programes, pensant noves solucions, reescrivint codi, buscant informació, aprenent nous llenguatges, etc.

Això no vol dir que la programació sigui una mena d'art que depèn de la inspiració del moment, i de provar i provar fins que la cosa funcioni. No. És cert que cal ser tafaner, tenir curiositat, ganes de provar, dedicar-hi temps. També que la creativitat i la persistència acceleren i incrementen l'aprenentatge i si, a més, es gaudeix programant i cada programa s'afronta com un repte, s'arriba a un nivell de destresa important molt més ràpid.

Però també és cert que cal aprendre i consolidar les tècniques de programació bàsiques, l'algorísmica, en definitiva. Aprendre-les bé des del principi i també consolidar-les, i això es fa amb la pràctica i amb el temps.

En concret, atès que un programa combina instruccions i treballa amb dades per solucionar un problema determinat, els elements bàsics de la programació que necessitem són dos:

Això ho treballarem a les unitats 2, 5 i 6.

1. Saber com representar les dades amb què hem de treballar.
2. Saber quines instruccions solucionaran el problema y com les hem de combinar.

Això ho aprendrem a les unitats 3, 4 i 7.

Però, a més d'aprendre els elements i les tècniques algorísmiques i saber-les utilitzar convenientment, hem d'incorporar bones pràctiques en la nostra manera de fer que ajudaran a fer que aconseguim bons programes: correctes, clars i eficients.

I amb això tampoc no en tindrem prou, caldrà que coneguem molt bé la sintaxi i els entrellats del llenguatge de programació que utilitzarem, que sapiguem consultar els manuals de referència adients en cas de dubte, i finalment, que provem bé el programa per assegurar-nos i comprovar que fa el que ha de fer.

En definitiva, hem de posar en joc a parts iguals una bona tècnica algorísmica, un seguit de bones pràctiques, la inspiració, el pensament computacional, el coneixement fi del llenguatge (en aquest cas el JS) i la persistència. La bona notícia és que tot això es pot aprendre i millorar.

Bones pràctiques

Per aconseguir un bon programa, abans cal entendre què és el que s'ha de resoldre i, a continuació, abans de llançar-se a escriure instruccions sense solta ni volta i provar si per casualitat funciona la cosa, pensar com resoldre-ho, com arribarem a calcular o obtenir el que ens demanen, i finalment programar-ho en JavaScript de manera molt precisa (ja sabeu que els ordinadors no pensen i només entenen exactament allò que escrivim, qualsevol petit lapsus no el sabran interpretar). Finalment, hem de provar el que hem fet per veure si fa el que ha de fer. Lògic, oi?

Així, un bon costum per incorporar bones dinàmiques de programació serà seguir sempre aquests pasos:

1. Entendre el problema, 2. Pensar com resoldre'l, 3. Fer un esborrany de la solució (algorisme), 4. Implementar-lo en un llenguatge de programació i 5. Provar el codi.

1 Entendre el problema | 2 Pensar com resoldre'l | 3 Fer un esborrany de la solució (algorisme) | 4 Implementar-lo en un llenguatge de programació | 5 Provar el codi

1 Entendre el problema

És imprescindible assegurar-nos que entenem l'encàrrec que ens donen (l'enunciat), en definitiva, que entenem quin problema cal que solucioni el programa. I, tot i que sembla molt de calaix, aquest pas no és tan simple. A vegades, l'enunciat no és prou clar, és ambigu o no ens dona tota la informació que necessitem. Si aquest és el cas, hem de repreguntar fins que no tinguem cap dubte. Altres vegades no l'interpretem bé, així que també és una bona estratègia comprovar que ho hem entès, contrastant-lo amb qui ens dona l'enunciat.

Com ho podem fer?

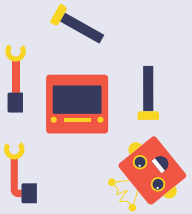
Llegint l'enunciat i preguntant-nos:

1. quines són les dades que tenim de partida?; i
2. quines dades hem d'obtenir com a solució?

En definitiva, un programa agafa unes dades inicials, les transforma i n'obté uns resultats, així que tenint clares aquestes preguntes ja tenim molt de guanyat.

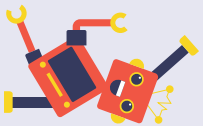
Exemple

Enunciat:
"Se'ns ha desmuntat el robot.
Heu d'enganxar totes les peces":



Aquest seria un exemple d'enunciat poc precís. Podem deduir que vol dir muntar-lo:

Però també podem no interpretar-ho bé:



Aquest resultat també respondria l'enunciat, oi? El robot està muntat però malament, ho veieu?

Un enunciat més correcte seria: "Se'ns ha desmuntat el robot. Heu de muntar cada peça al seu lloc corresponent". Aquí ja sí que la primera solució seria correcta però la segona no.

Són les dades que ens dona l'enunciat.

Dades d'entrada: cada peça del robot per separat.
Dades de sortida: un robot amb totes les peces connectades al lloc que els pertoca.

És la sortida esperada del nostre programa.

Exemple

Enunciat:
Calcula la suma dels cinc primers números enters.

Aquest enunciat és clar i concís. No admet dubtes ni confusions, oi?
Dades entrada: 1, 2, 3, 4, 5 (els cinc primers números enters).
Dada de sortida: la suma d'aquests cinc nombres.

En definitiva, si l'enunciat no és clar, o no l'entenem bé, s'ha d'aclarir abans de continuar.

2 Pensar com resoldre'l

Un cop tenim clar què cal resoldre, hem de pensar com fer-ho. Buscar la solució, en definitiva, pensar l'estratègia (passos a seguir) per arribar a obtenir les dades que ens demana l'enunciat a partir de les que ens dona. I d'estratègies n'hi pot haver més d'una. Si es dona el cas, triem la més senzilla i clara.

Exemple

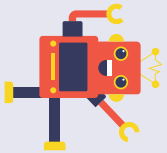
Com muntarem el robot?

Una estratègia possible seria:
"Primer enganxar el cap al cos, després enganxar-hi els braços i per últim enganxar-hi les cames".

Una altra manera seria:
"Primer enganxar les cames al cos, després enganxar-hi els braços i finalment enganxar-hi el cap."

De les dues maneres muntarem el robot, el resultat serà el mateix. Ara bé, el muntaré correctament? Potser no.

Les dues solucions són poc precises i poden donar lloc a:



Precisem una mica més:
"Enganxar el cap a sobre del cos, després enganxar els braços, un a cada costat del cos i per últim enganxar les cames una a cada costat, a sota del cos."

Ara sí que tenim clar com muntar el robot correctament.

Exemple

Enunciat:
Calcula la suma dels cinc primers números enters.

Dades entrada: 1, 2, 3, 4, 5 (els cinc primers números enters).
Dada de sortida: la suma d'aquests cinc nombres.

Una estratègia seria calcular la suma de 1+2+3+4+5.

Una altra estratègia possible seria sumar 5+4+3+2+1. Ja sabem que la suma es commutativa així l'ordre dels sumands no té importància. Però és més clara la primera opció, no?

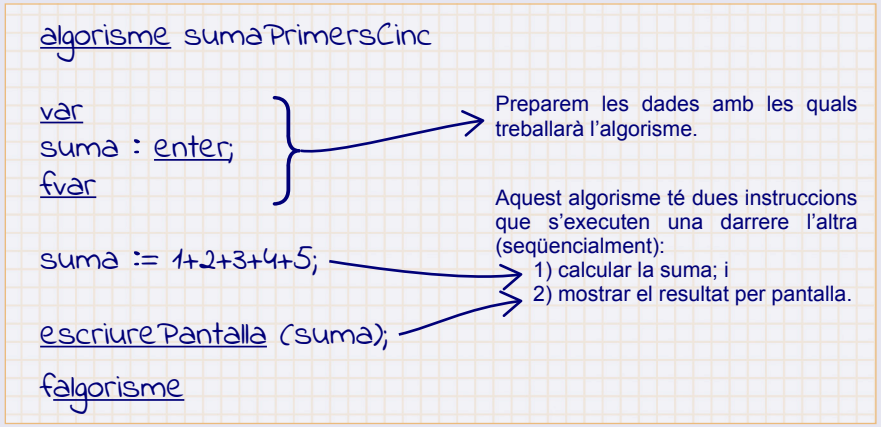
Així que, si tenim diverses maneres de solucionar el problema, escollirem sempre la més senzilla i clara. A més, convé recordar que els ordinadors no "pensen", segueixen instruccions específiques, que han de ser precises i simples. Així que el programa només farà allò que s'indiqui explícitament a les instruccions.

3 Fer un esborrany de la solució (un algorisme)

Un cop hem pensat com trobar la solució, hem de fer un esborrany de les instruccions detallades que ens portaran a la sortida esperada partint de les dades d'entrada. I per això només podem organitzar instruccions de manera que una màquina les pugui seguir (recordeu que estem fent un programa), això vol dir que només podem utilitzar els elements de l'algorísmica. I d'aquí la importància de conèixer l'algorísmica a fons.

Exemple

Tot i que encara no coneixeu els elements de l'algorísmica, vegem com seria l'algorisme que resol la suma dels cinc primers nombres:



S'entén què fa, oi? Un algorisme és, doncs, un conjunt d'instruccions expressades de manera que es puguin seguir clarament per fer una tasca, en aquest cas sumar els cinc primers nombres enters.

Aquesta seria la solució al problema ja expressada, en notació algorísmica.

Per què hem de fer aquest pas? Amb l'anterior no n'hi hauria prou? No, hem de trobar una solució al problema, però ha de ser una **solució algorísmica**, sinó, no la podrem codificar posteriorment en un llenguatge de programació. És per això que cal poder expressar aquesta solució en termes algorísmics, fer l'algorisme, en definitiva. Això sí, la solució s'expressa en una notació senzilla que permet escriure les instruccions de manera simple i molt natural, en català en aquest cas.

Escriure l'algorisme és una bona manera de planificar la solució abans de codificar-la en un o altre llenguatge de programació concret.

Per escriure algorismes, definirem una **notació algorísmica**, que serà una manera simple de descriure conjunts d'instruccions sense una sintaxi específica de cap llenguatge de programació. És una notació absolutament arbitrària però que és similar a la d'un llenguatge de programació. Aquesta notació ens permetrà "entendre" entre nosaltres els algorismes que fem, ja que tots els expressarem de la mateixa manera. Revisant la bibliografia trobareu que hi ha tantes notacions algorísmiques com referències, però totes són molt similars perquè tenen els mateixos components, els elements fonamentals de l'algorísmica.

Conèixer la notació algorísmica ens permet, a més, representar la solució al problema en pseudocodi sense preocupar-nos de la sintaxi ni de les particularitats del llenguatge en el qual haurem de fer el programa després i, el que és més important, separar l'etapa de pensar en la solució de la de codificar-la.

4 Implementar-lo en un llenguatge de programació

Ja sabeu que hi ha molts llenguatges de programació. Així que, finalment, per poder executar el programa caldrà que el traduïm a un llenguatge de programació en concret. Haurem triat un o un altre depenent de diversos motius. Hi ha llenguatges més adients per a segons quines tasques que d'altres. Moltes vegades, però, el llenguatge ja ens vindrà determinat. En el nostre cas, treballarem amb el llenguatge **JavaScript,** que és un llenguatge que s'utilitza per donar interactivitat a les pàgines web, per exemple.

Així que haurem de conèixer molt bé la sintaxi i la semàntica del llenguatge per fer els nostres codis **JS**.

Exemple

Aquest seria un programa JS que soluciona el problema de la suma.

```
1  var suma;
2  suma = 1+2+3+4+5;
3  console.log(suma);
```

És similar a l'algorisme, oi?

I aquest seria el mateix algorisme en llenguatge C:

```
#include <stdio.h>

int main()

{
 int suma;

 suma = 1+2+3+4+5;
 printf(suma);
}
```

Són similars, però tenen una sintaxi una mica diferent. A més, cada llenguatge té les seves pròpies característiques per poder fer els codis més eficients en termes de ús de memòria i temps d'execució. Ara això no és rellevant perquè els programes que escriurem són molt senzills.

Però tots els llenguatges comparteixen, això sí, els mateixos principis algorísmics.

Per això és una bona pràctica pensar la solució independentment del llenguatge. Així separem el problema en dues parts: primer, **hem de pensar en la solució** i, després, preocupar-nos de les **qüestions tècniques del llenguatge** en concret.

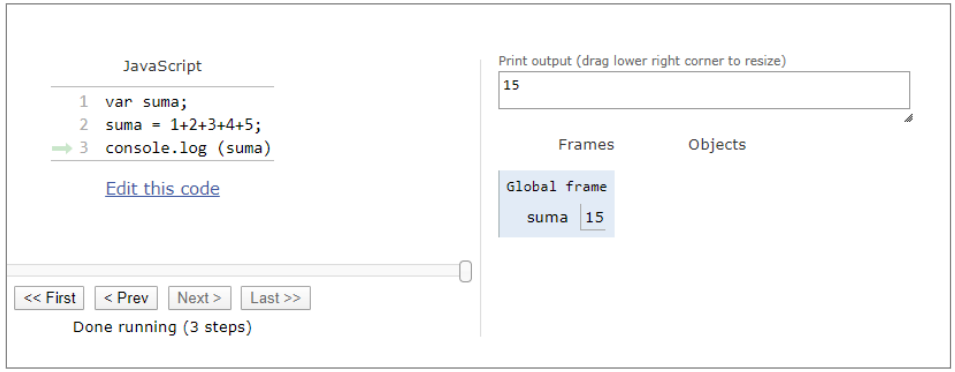
És similar al que passa quan escrivim un text: primer pensem què volem dir, en quins apartats el distribuirem i fem un esborrany tot posant-hi les idees. Després, en fem una relectura per fixar-nos en l'ortografia, la gramàtica, etc. A mesura que guanyem pràctica com a escriptors, la segona volta es va reduint perquè ja directament en la primera passada sabem escriure sense faltes i sense haver de pensar en les normes d'ortografia. Amb la programació passa una cosa similar: a mesura que guanyem experiència en un determinat llenguatge, ja no ens costa escriure directament el codi en el llenguatge sense que això interfereixi en el seu disseny.

5 Provar el codi

Finalment, un cop el programa funciona, cal comprovar que la solució resol el problema. Això és **provar el programa**. Aquest pot ser un pas complicat, especialment en programes grans i sofisticats. Per això existeixen diverses tècniques de prova.

En el cas del nostre codi JS serà pràctic veure què passa si l'executem des del PythonTutor.

Ho podeu provar a la web de PythonTutor:



En aquest cas és senzill perquè només hi ha una solució possible, així que, amb una vegada que ho l'executem i veiem que el resultat és 15, ja ho tenim!

Normalment, però, cal pensar en diferents "jocs de proves" per validar que un programa és correcte, i aquest pas esdevé més complicat. Però és importantíssim, això sí, per validar la nostra solució. En termes professionals, les proves són imprescindibles i representen un pas delicat que permet decidir quan una aplicació (o una nova versió d'una ja existent) es distribueix als usuaris.

Què passaria si...

...la temptació (i la mala praxis) ens fan anar al pas 4: **a codificar directament**. I si, a base de prova i error, anem polint el codi fins que funcioni més o menys? **Això, a més de clarament ineficient és poc segur.**

I si resulta que no hem entès bé què se'ns demanava perquè no hem pensat prou en l'encàrrec o l'enunciat (**pas 1**)? Haurem treballat en va perquè, quan li ensenyem a l'usuari, ens dirà que no era allò el que necessitava.

I si ens hem posat a escriure instruccions sense pensar gaire com fer-ho (**pas 2 i pas 3**) i acabem amb un embolic de programa que no hi ha qui l'entengui? Potser sí que funciona, però algú el podrà modificar més tard? Algú més l'entendrà? Té una estructura clara? Si els programadors treballen en equip, no val més que feu programes clars i inteligibles? I què passaria si el professor que us ha de corregir no entén què heu fet? Ja pot funcionar, que la cosa no acabaria bé.

I si desconeixem la sintaxi i les característiques del llenguatge (**pas 4**), costarà que funcioni i gastarem moltes hores depurant errades i més errades. Serem poc eficients i se'ns farà pesat. Abans de codificar la solució, revisem la sintaxi, preguntem, busquem en manuals. Quan escrivim un text i tenim un dubte ortogràfic, oi que consultem el diccionari?

I si finalment no provem a fons el programa? (**pas 5**). Ves a saber si fa el que s'esperava, potser sí en alguns casos però no en tots. Abans de posar en circulació un nou codi, farem bé de provar-lo a fons. A aquest pas sovint no se li dedica el temps que cal, malauradament.

I si, de passada, el comentem bé (incloent-hi explicacions i aclariments al mateix codi), molt millor. Serà senzill d'entendre i, si cal, de modificar-lo després. Aquesta és una molt bona pràctica que sovint s'obvia per falta de temps, per les presses, tot i que està àmpliament documentada i és de sobres conegut l'alt cost que té més tard quan, per exemple, ja ningú no és capaç d'entendre aquell codi i s'ha de reescriure de nou.

Com més sistemàticament ho fem, menys coses deixarem a l'atzar i aconseguirem fer bons programes, i a ser, en definitiva, bons programadors.

Començant a programar

Concepte clau

Un **algorisme** és una seqüència de passos per a fer una tasca. Un **programa** és la codificació d'un algorisme expressat en un llenguatge de programació concret que un ordinador pot interpretar.

La programació estructurada es fonamenta en l'**algorísmica,** que presenta els elements fonamentals de la programació per a qualsevol llenguatge i que són: els tipus de dades per representar la realitat i els tres principis de combinació (seqüència, iteració o repetició, i selecció). Juntament amb les funcions per agrupar codi i els esquemes per aprofitar idees que ja funcionen, i guanyar eficiència. Pot semblar sorprenent, màgic fins i tot, però combinant aquests elements es pot solucionar qualsevol problema que es pugui resoldre mitjançant un procediment (seguint unes instruccions). El que se'n diu, en termes informàtics, un problema computable. Dominar l'algorísmica obre les portes al poderós món de la computació.

Hi ha altres paradigmes: programació funcional, lògica, per exemple, que no són els que s'utilitzen en la pràctica diària, habitualment.

El **pensament computacional** aglutina un seguit de tècniques que faciliten la programació: ajuden a dividir un problema complex en problemes més senzills i tractables (**descomposició**), que es poden anar resolent un a un individualment tenint en compte com s'han solucionat problemes similars anteriorment (**reconeixement de patrons**), i parant atenció només en els detalls rellevants del problema (**abstracció**). Tot seguit, permet definir el pla d'acció, un conjunt de passos a seguir per resoldre cadascun dels problemes (**algorísmica**) i acabar codificant aquests passos en un llenguatge de programació com a instruccions d'un programa que l'ordinador entendrà per resoldre el problema.

De **llenguatges de programació** n'hi ha de molts tipus, entre els textuals més habituals en l'àmbit professional trobem el C, el Java i el Python.També el JavaScript (JS), que és molt adient per programar la interacció en les pàgines web. Cadascun dels llenguatges té una sintaxi i una semàntica específica i diferent (tal com passa amb els llenguatges que parlem), i és més adient que d'altres per a usos determinats, així que se n'escull un o altre segons el que s'hagi de programar. Però tots comparteixen els elements fonamentals de l'algorísmica i del pensament computacional: la representació de les dades, les instruccions elementals i les possibles maneres de combinar-les (en seqüència, repetint o seleccionant).

Que que un programa "funcioni", és a dir, que faci el que es demana (això és, que resolgui el problema o que faci la tasca descrita a l'enunciat o a l'encàrrec) no és suficient perquè es pugui considerar un bon programa; cal, a més, que sigui eficient (que utilitzi els mínims recursos de temps i memòria) i clar (que s'entengui què fa i que es pugui modificar fàcilment).

Un bon programador té destresa en pensament computacional i també coneix a fons la sintaxi i la semàntica del llenguatge en la qual programa i posa en joc un conjunt de bones pràctiques: comença per tenir clar quin és l'encàrrec (la funcionalitat que ha de tenir el programa), rumia quina és la millor estratègia d'aconseguir-ho, defineix clarament els passos o les instruccions per arribar-hi, ho codifica traient el màxim partit del llenguatge de programació en el qual està programant, i comprova que el funcionament final respon al que es demanava amb un bon joc de proves. Aquesta manera de fer es sintetitza en els següents passos:

1. Entendre el problema.
2. Pensar com resoldre'l.
3. Fer un esborrany de la solució (algorisme).
4. Implementar-lo en un llenguatge de programació.
5. Provar el programa.