

Llenguatges de Programació

Introducció

Albert Rubio, Jordi Petit, Fernando Orejas



Universitat Politècnica de Catalunya, 2019

Introducció

Un **llenguatge de programació** (LP) és un llenguatge formal utilitzat per controlar el comportament d'un computador tot implementant un algorisme.

Cada llenguatge té una sèrie de regles estrictes:

- **Regles sintàctiques:** descriuen l'estructura dels programes vàlids.
- **Regles semàntiques:** descriuen el seu significat.

Cada llenguatge té (hauria de tenir) una especificació:

- un document estàndard (Ansi C) o
- una implementació de referència (CPython).

Característiques bàsiques d'un LP

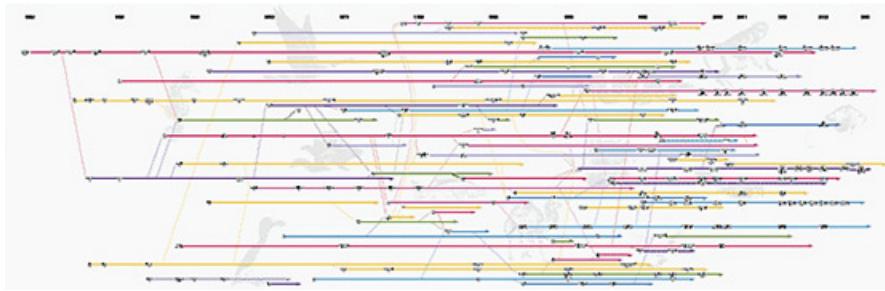
- Tipus de dades: amb quins dades i objectes treballem.
- Sistema de tipus.
- Control de seqüència: en quin ordre s'executen les operacions.
- Control de dades. Com s'accedeix a les dades i als objectes.
- Entrada / Sortida.

Qualitats dels llenguatges de programació

- Llegibilitat
- Eficiència
- Fiabilitat
- Expressivitat
- Simplicitat
- Nivell d'abstracció
- Adequació als problemes a tractar
- Facilitat d'ús
- Ortogonalitat

Història

O'Reilly History of Programming Languages



Font: O'Reilly

Història: Orígens

Tauletes babilòniques (2000ac)

A [rectangular] cistern.

The height is 3, 20, and a volume of 27, 46, 40 has been excavated. The length exceeds the width by 50. You should take the reciprocal of the height, 3, 20, obtaining 18. Multiply this by the volume, 27, 46, 40, obtaining 8, 20. Take half of 50 and square it, obtaining 10, 25. Add 8, 20, and you get 8, 30, 25. The square root is 2, 55. Make two copies of this, adding [25] to the one and subtracting from the other. You find that 3, 20 [i.e., 3 1/3] is the length and 2, 30 [i.e., 2 1/2] is the width.

This is the procedure.



Vídeo: Math whizzes of ancient Babylon figured out forerunner of calculus



. D.E. Knuth, Communications ACM 1972.

Història: Orígens

Teler de Jacquard (1804)



Fotos: <http://www.revolutionfabrics.com/blog/2018/9/26/the-jacquard-loom-and-the-binary-code>

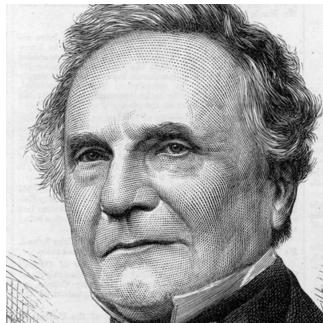
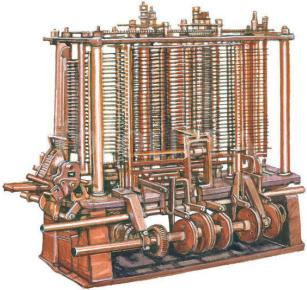


Vídeo: How an 1803 Jacquard Loom Lead to Computer Technology

Història: Orígens

Màquina analítica de Charles Babbage (1842)

Es considera que Ada Lovelace és la primera programadora.



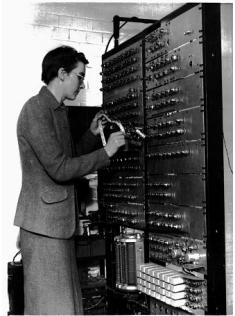
Fotos: Domini públic



Vídeo: Ada Lovelace, The World's First Computer Nerd?

Història: Ensabladors

Es considera que Kathleen Booth va escriure el primer llenguatge ensamblador per un ordinador ARC al 1947.



```
Multp: .equ 32           ; multiplicand
Multd: .equ 64           ; multiplier
.org 4096          ; 0x1000
lar r30, Done        ; Load address of Done for branch
lar r31, Loop         ; Load address of Loop for branch
la r1, Multd         ; Load multiplier
la r2, Multp         ; Load multiplicand
andi r3, r3, #0       ; clear r3
andi r4, r4, #0       ; clear r4
addi r5, r3, #1       ; place 1 in r5
neg r3, r5            ; place -1 in r3
Loop: add r4, r4, r2   ; add multiplicand to running sum
      add r1, r1, r3   ; start loop, decrement multiplier
      brzr r30,r1        ; jump to Done if multiplier = 0
      br r31             ; jump back to Loop
Done: st r4, Result    ; store result
      stop
.Result: .org 8192        ; 0x2000
.Result: .dw 1             ; storage for result
```

Fotos: Domini públic

Història: Plankalkül

Primer LP d'alt nivell dissenyat per Konrad Zuse (1942-1945) pel seu ordinador a relés Z4. Implementat al 1990.

Fig. 3.

y	$\textcircled{1} R(V) \succ R$
s	$\begin{matrix} o & o \\ m & o & o \end{matrix}$
y	$\textcircled{2} As(V) \succ \& R$
K	$\begin{matrix} o & o \\ o & o \\ s & o \end{matrix}$
S	$\begin{matrix} o & o \\ o & o \\ \sigma & \sigma \end{matrix}$
y	$\textcircled{3} \mu x \left[x \in V \wedge x + y \right] \succ Z$
K	$\begin{matrix} o & o \\ o & o \\ \sigma & m \sigma \end{matrix}$
S	$\begin{matrix} o & o \\ o & o \\ \sigma & \sigma \end{matrix}$
y	$\textcircled{4} Kt(Z) \rightarrow (e + 1 \succ e)$
S	$\begin{matrix} 1 & \\ \sigma & \sigma \end{matrix}$
y	$\textcircled{5} e \geq o \succ \& R$
S	$\begin{matrix} o & o \\ o & o \\ \sigma & \sigma \end{matrix}$
y	$\textcircled{6} Sx(Z) \succ \& R$
S	$\begin{matrix} o & o \\ o & o \end{matrix}$



Fotos: Domini públic

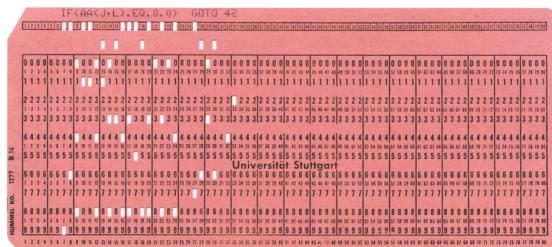
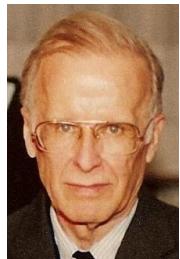
Història: Fortran

FORmula TRANslator (1954-1957). Desenvolupat per John Bakus a IBM per a computació científica.

Es volia generar codi comparable al programat en ensamblador.

Idees principals:

- Variables amb noms (6 caràcters)
- Bucles i condicionals aritmètics
- E/S amb format
- Subrutines
- Taules



Fotos: Domini públic i Wikipedia

Història: Fortran



Història: COBOL

Common Business Oriented Language (1959). Desenvolupat per Grace Hopper pel DoD i fabricants per a aplicacions de gestió.

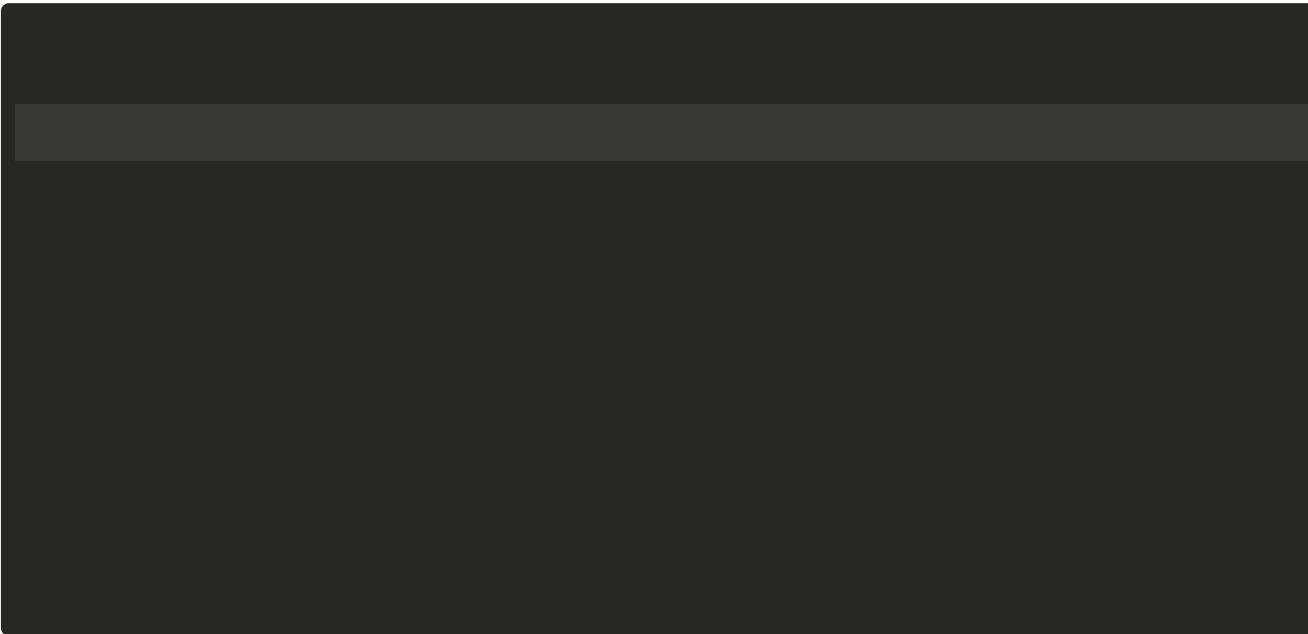
Idees principals:

- Vol semblar idioma anglès, sense símbols
 - Macros
 - Registres
 - Fitxers
 - Identificadors llargs (30 caràcters)



Foto: Domini públic

Història: COBOL



Història: LISP

LISP (LISt Processing) (1958). Desenvolupat per John McCarthy al MIT per a recerca en IA.

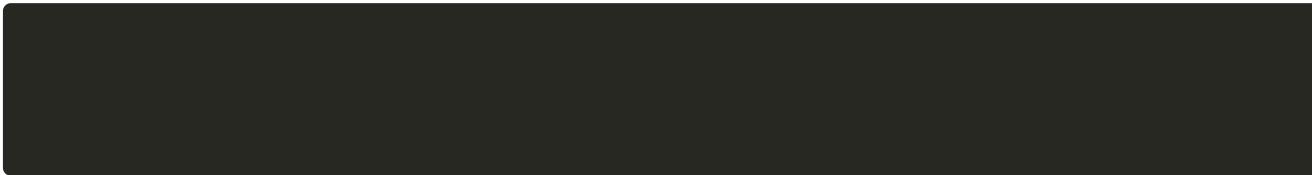
Idees principals:

- Sintàxi uniforme
- Funcions (composició i recursivitat)
- Llistes
- Expressions simbòliques
- Recol·lector de brossa



Foto: Wikipedia

Història: LISP



Història: Algol

ALGOrithmic Language (1958). Dissenyat com un llenguatge universal per computació científica. No gaire popular, però dóna lloc a LPs com Pascal, C, C++, and Java.

Idees principals:

- Blocs amb àmbits de variables
- Pas per valor i pas per nom (\neq pas per referència)
- Recursivitat
- Gramàtica formal (Backus-Naur Form or BNF)

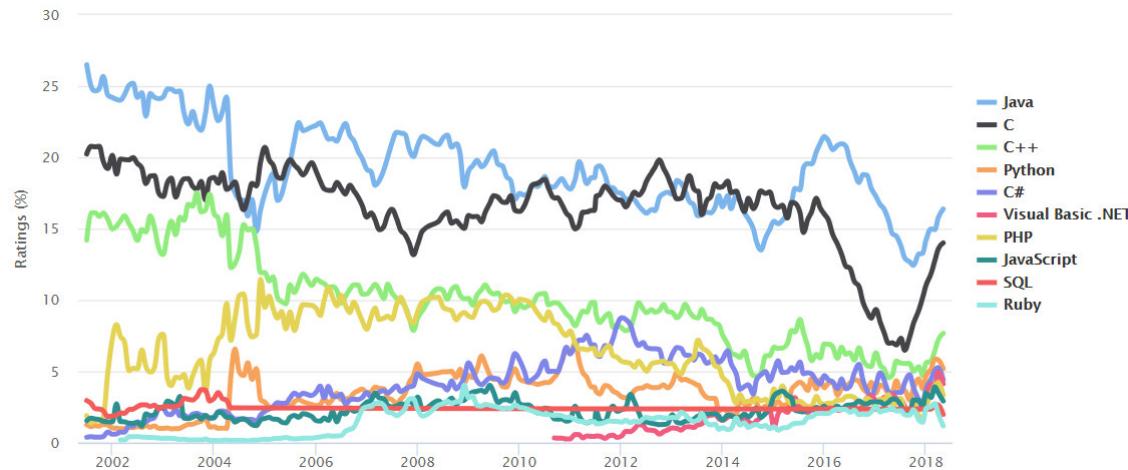
Història: altres llenguatges

- Basic (Orientat a l'ensenyament de la programació) - 1964
- Pascal/Algol 68 (els hereus directes d'Algol 60) - 1970/1968
- C (Definit per programar Unix) - 1972
- Prolog (Primer llenguatge de programació lògica) - 1972
- Simula 67/Smalltalk 80 (primers llenguatges OO)
- Ada (LP creat per ser utilitzat pel Departament de Defensa Americà) - 1980
- ML/Miranda (primers llenguatges funcionals moderns) - 1983/1986

Ús dels LPs

Com mesurar la popularitat dels LPs?

- TIOBE Programming Community Index

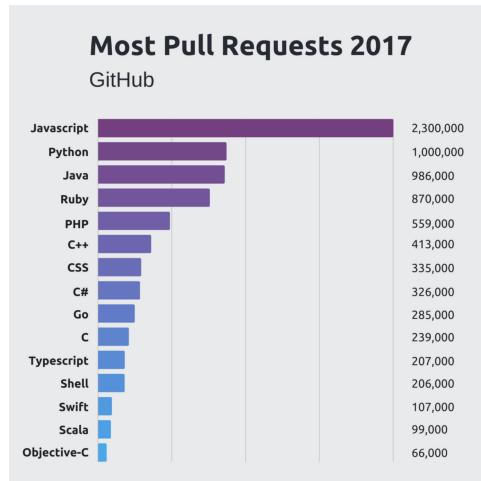


Font: <https://www.tiobe.com/tiobe-index/>

Ús dels LPs

Com mesurar la popularitat dels LPs?

- Estadístiques de GitHub



Font: <https://github.com>

Paradigmes de LPs

Els paradigmes de programació classifiquen els LPs segons les seves característiques.

The Paradigms of Programming

Robert W. Floyd
Stanford University



Paradigm (par'-adim, -dēm) ... In F. *paradigme*, ad.
L. *paradigma*, a. Or: *esquema*; pattern, example, f.
now *esquema*; or to exhibit beside, show side by side. . .]
I. A pattern, exemplar, example.

1752 J. Gill *Trinity v. 91*

The archetypal paradigm, exemplar, and idea,
according to which all things were made.

From the Oxford English Dictionary.

Today I want to talk about the paradigms of programming, how they affect our success as designers of computer programs, how they should be taught, and how they should be embodied in our programming languages.

A familiar example of a paradigm of programming is the technique of *structured programming*, which appears to be the dominant paradigm in most current treatments of programming methods. Structured programming, as formulated by Dijkstra [6], Wirth [27, 29], and Parnas [21], among others, consists of two phases:

In the first phase, that of top-down design, or stepwise refinement, the problem is decomposed into a very small number of simpler subproblems. In programming the solution of simultaneous linear equations, say, the first level of decomposition would be to identify a stage of manipulating the equations and a following stage of back-substitution in the triangulated system. This gradual decomposition is continued until the subproblems that arise are simple enough to cope with directly. In the simultaneous equation example, the back-substitution process would further decompose, say, as a backwards iteration of a process which finds and stores the value of the i th variable from the i th equation. Yet further decomposition would yield a fully detailed algorithm.

The second phase of the structured programming paradigm entails working upward from the concrete objects and functions of the underlying machine to the more abstract objects and functions used throughout the modules produced by the top-down design. In the linear equation example, if the coefficients of the equations are rational functions of one variable, we might first design

Permissions to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of publication and its date appear, and notice is given that copying is by permission of ACM, Inc. for Computer Monographs. To copy otherwise, or to republish, requires a fee and/or specific permission. Address reprint and permission requests to Computer Monographs, ACM, Inc., Department of Computer Science, Stanford University, Stanford CA 94305. © 1979 ACM 0001-0782/79/0800-0453 \$0.75.

R. Floyd, Communications ACM 1979.

Paradigmes de LPs

Paradigmes comuns:

- **Imperatiu:** Les instruccions precisen els canvis d'estat.



- **Declaratiu:** Es caracteritza el resultat, però no com calcular-lo.



- **Funcional:** El resultat és el valor d'una sèrie d'aplicacions de funcions.



Paradigma imperatiu

Caràcteristiques:

- Noció d'estat
- Instruccions per canviar l'estat
- Efectes laterals

Exemples:

- C/C++, Python, Java, Ensamblador, ...

Útils quan, per exemple, l'eficiència és clau.

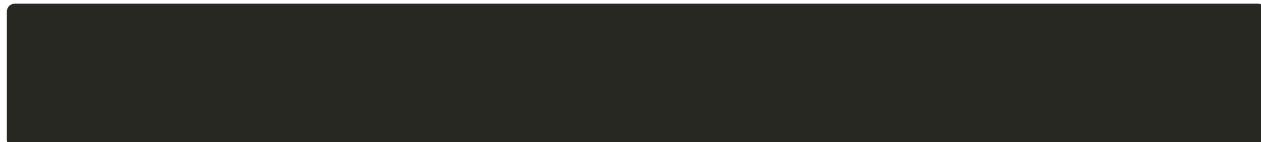
Paradigma imperatiu

Subclassificacions:

- **Procedural:** Les instruccions s'agrupen en procediments.



- **Orientat a objectes:** Les instruccions s'agrupen amb l'estat dels objectes sobre les quals operen.



Paradigma declaratiu

Caràcteristiques:

- Llenguatges descriptius.
- El programa diu què s'ha de fer, però no necessàriament com.

Utilitat:

- Prototipat d'aplicacions amb forta component simbòlica, problemes combinatoris, etc.
- Consultes en bases de dades relacionals o lògiques.
- Per especificació i raonament automàtic.

Exemples:

- SQL (consultes relacionals) / GraphQL (consultes en grafs, amb tipus) / Prolog (lògica de primer ordre) / Matemàtiques

Paradigma declaratiu

Subclasificacions:

- **Matemàtic:** El resultat es declara com a la solució d'un problema d'optimització.



- **Consultes:** Resposta a consultes a una base de dades.



- **Lògic:** Resposta a una pregunta amb fets i regles.



Paradigma funcional

Caràcterístiques:

- Procedural
- Sense noció d'estat i sense efectes laterals
- Més fàcil de raonar (sobre correctesa o sobre transformacions)

Utilitat:

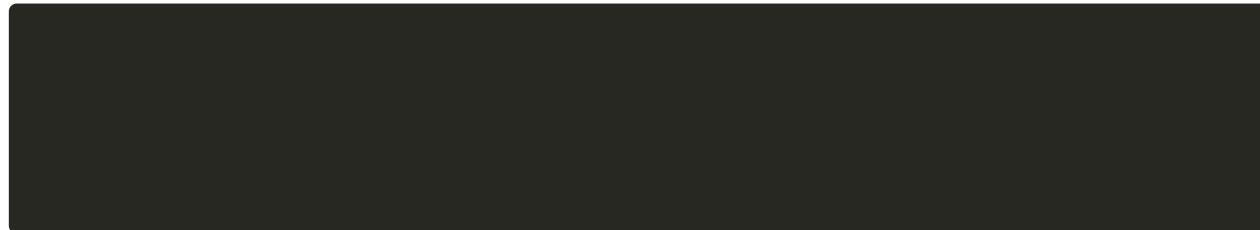
- Útils per al prototipat, fases inicials de desenvolupament (especificacions executables i transformables).
- Tractament simbòlic.
- Sistemes de tipus potents (incloent polimorfisme paramètric i inferència de tipus)

Exemples: Haskell, ML (Caml, OCaml), Erlang, XSLT (tractament XML),...

Paradigma funcional

Conceptes clau:

- **Recursivitat:**



- **Funcions d'ordre superior:** funcions que reben o retornen funcions.



- **Aplicació parcial (currying):**



Paradigma funcional

Conceptes clau:

- **Funcions pures:** Amb els mateixos arguments, les funcions sempre retornen el mateix resultat. No hi ha efectes laterals.
- **Mecanismes d'avaluació:** Avaluació estricta avaluació mandrosa.
- Sistemes de tipus

Paradigma OO

Caràcteristiques:

- Es basa en (atributs + mètodes) i potser .
- Inclou principalment i .
- Poden tenir sistemes de tipus complexos.

Exemples: Smalltalk, Simula, C++, Java...

Llenguatges multiparadigma

Molts LPs combinen diferents paradigmes. Per exemple:

- Python, Perl: **imperatiu** + orientat a objectes + funcional
- OCaml: **funcional** + imperatiu + orientat a objectes

Alguns LPs incorporen caràcteristiques d'altres paradigmes:

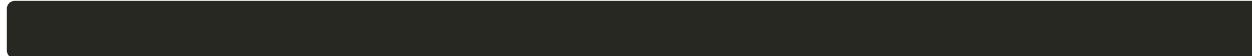
- Prolog: **lògic** (+ imperatiu + funcional)

Altres combinacions:

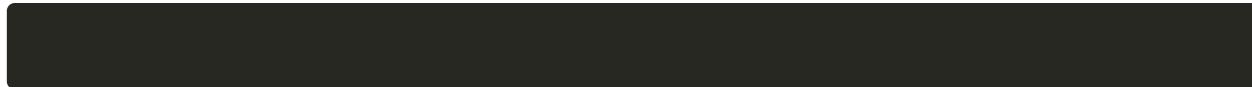
- Erlang: **funcional** + concurrent + distribuït

Llenguatges esotèrics

- **Brainfuck:** Basat en màquines de Turing, només té 8 instruccions. Exemple:



- **Whitespace:** Només té espais en blanc i tabuladors. Exemple:



(indenteu-lo bé quan el copieu 😊)

- **Shakespeare:** Amaga un programa dins d'una obra de teatre.

```
The Infamous Hello World Program.

Romeo, a young man with a remarkable patience.
Juliet, a likewise young woman of remarkable grace.
Ophelia, a remarkable woman much in dispute with Hamlet.
Hamlet, the flatterer of Andersen Insulting A/S.

Act I: Hamlet's insults and flattery.

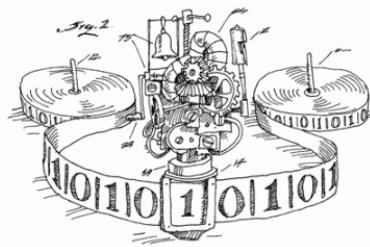
Scene I: The insulting of Romeo.

[Enter Hamlet and Romeo]

Hamlet:
  You lying stupid fatherless big smelly half-witted coward!
  You are as stupid as the difference between a handsome rich brave
```

Turing completeness

Màquina de Turing: Model matemàtic de càlcul imperatiu molt simple.
(Allan Turing, 1936)



- Cinta infinita amb un capçal mòvil per llegir/escriure símbols
- Conjunt finit d'estats
- Funció de transició (estat, símbol → estat, símbol, moviment)

λ -càcul: Model matemàtic de càlcul funcional molt simple.
(Alonzo Church, 1936).

$$(\lambda y.x(yz)) (ab) \downarrow x(abz)$$



- Sistema de reescritura
- basat en abstracció i aplicació de funcions.

Tesi de Church-Turing: "Tot algorisme és computable amb una Màquina de Turing o amb una funció en λ -càcul".

Turing completesa

Un LP és **Turing complet** si pot implementar qualsevol càlcul que un computador digital pugui realitzar.

Alguns autors consideren només com a LPs els llenguatges Turing complets.

- LPs Turing complets:

- LPs de programació de propòsit general (C/C++, Python, Haskell...)
- automàts cel·lulars (Joc de la vida, ...)
- alguns jocs (Minecraft, Buscamines...)

Per ser Turing complet només cal tenir salts condicionals (bàsicament,  i ) i memòria arbitràriament gran.

- LPs no Turing complets:

- expressions regulars (a Perl o a AWK)
- ANSI SQL

Sistemes d'execució

- **Compilat:** el codi és transforma en codi objecte i després es monta en un executable. Sol ser eficient. Es distribueix l'executable.

Exemples: C, C++, Ada, Haskell, ...

- **Interpretat:** el codi s'executa directament o el codi es transforma en codi d'una màquina virtual, que l'executa. Es distribueix el codi font.

Aumenten la portabilitat i l'expressabilitat (es poden fer més coses en temps d'execució) però disminueix l'eficiència.

Exemples: Python, JavaScript, Prolog (WAM), Java (JVM), ...

Sistemes mixtes:

- **Just in Time compilation:** Es compila (parcialment) en temps d'execució.
- Alguns interpretats, poden ser també compilats (per exemple, Prolog).
- i al revés (Haskell).

Sistemes de tipus

Un **sistema de tipus** és un conjunt de regles que assignen als elements d'un programa (com ara variables, expressions, funcions...) per evitar errors.

La comprovació de tipus verifica que les diferents parts d'un programa es comuniquin adequadament en funció dels seus tipus.

Per exemple, amb

- cridar a amb un o un és correcte,
- cridar a amb una o un enter és un error de tipus.

Sistemes de tipus: *type safety*

Un llenguatge és **type safe** si cap programa pot donar errors de tipus en temps d'execució.

Alguns errors típics en execució:

- Type Cast: usar un valor d'un tipus en un altre tipus.

Exemple: En C, un enter pot ser usat com a funció (com a adreça), però pot saltar on no hi ha una funció i pot provocar un error.

- Aritmètica de punters.

Exemple: En C, si tenim `char *p` llavors `p` té tipus `char*`, però el que hi ha a `p` pot ser qualsevol altra cosa i pot provocar un error de tipus.

- Alliberament explícit de memòria (deallocate/delete).

Exemple: En Pascal usar un apuntador alliberat pot donar errors de tipus.

- En llenguatges OO (antics), no existència d'un mètode (degut a l'herència).

Sistemes de tipus: *type safety*

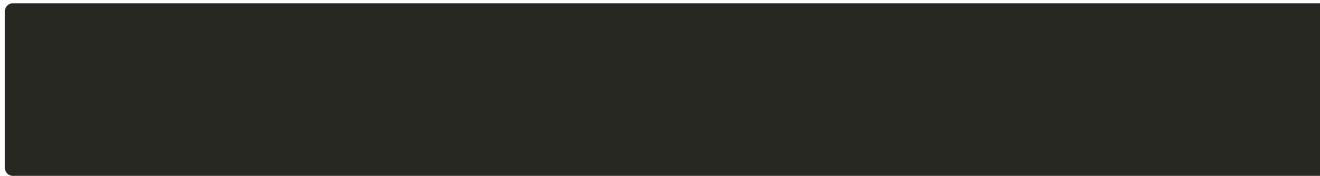
Exemples de llenguatges:

- : Haskell, Java, ...
- : C, C++, Pascal, ...

Sistemes de tipus: Tipat fort/feble

Els llenguatges amb **tipat fort** imposen restriccions que eviten barrejar valors de diferents tipus (per exemple, amb conversions implícites).

Per exemple, amb un tipat feble podem tenir:



Exemples de llenguatges:

- Tipat fort: C++, Java, Python, Haskell, ...
- Tipat feble: Basic, JavaScript, Perl, ...

Sistemes de tipus: Comprovació

La comprovació de tipus pot ser:

- **Estàtica:** en temps de compilació.
- **Dinàmica:** en temps d'execució.

Exemples de llenguatges:

- Comprovació estàtica: Haskell, C++, Java, ...
- Comprovació dinàmica: Python, Ruby, ...