TEMA 1: Conceptes bàsics

1. Introducció

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

Tradicionalment, els LPs es consideren des de tres angles (anàlegs a la lingüística):

- **Sintaxi:** la forma dels programes correctes.
- **Semàntica:** el significat de les construccions dels programes.
- **Pragmàtica:** com s'usa el LP.

La majoria d'LPs venen definits per

- una especificació estàndard, o
- una implementació de referència.

2. Perquè estudiar els LPs?

- **Visió enginyeril:** Necessitem LPs per programar computadors per fer coses xules.
- **Visió científica:** Necessitem LPs per descriure i raonar sobre aquestes computacions.

Els LPs, en tant que llenguatges, ens permeten representar conceptes i són una eina de pensament.

Els llenguatges que usem:

- influencien les nostres percepcions,
- guien i donen suport al nostre raonament,
- permeten i faciliten la nostra comunicació.

Exemple: Programació imperativa *vs* declarativa:

```
let xs = [1, 2, 3, 4, 5]

let ys = []
for (var i = 0; i < xs.length; i++) {
    ys.push(xs[i] * 2);
}
```

```
let xs = [1, 2, 3, 4, 5]

let ys = xs.map(function (x,i) {
    return 2*x
})
```

3. Com estudiar els LPs?

Estudiar-los tots. ❌

Estudiar les seves característiques, els seus usos, avaluar les seves qualitats, classificar-los en famílies, conèixer la seva història, estudiar els seus fonaments… ✅

4. 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.

5. Qualitats dels llenguatges de programació

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

6. Història

6.1 Orígens

- Tauletes babilòniques (2000 ac)
- Teler de Jacquard (1804)
- Màquina analítica de Charles Babbage (1842) (Es considera que Ada Lovelace és la primera programadora).

6.2 Ensambladors

Kathleen Booth va escriure el primer llenguatge ensamblador (per un ordinador ARC el **1947**).

6.3 Plankalkül

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

6.4 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

```
C AREA OF A TRIANGLE WITH A STANDARD SQUARE ROOT FUNCTION
      READ INPUT TAPE 5, 501, IA, IB, IC
  501 FORMAT (I5)
C IA, IB, AND IC MAY NOT BE NEGATIVE OR ZERO
C FURTHERMORE, THE SUM OF TWO SIDES OF A TRIANGLE
C MUST BE GREATER THAN THE THIRD SIDE, SO WE CHECK FOR THAT, TOO
      IF (IA) 777, 777, 701
  701 IF (IB) 777, 777, 702
  702 IF (IC) 777, 777, 703
  703 IF (IA+IB-IC) 777, 777, 704
  704 IF (IA+IC-IB) 777, 777, 705
  705 IF (IB+IC-IA) 777, 777, 799
  777 STOP 1
C USING HERON'S FORMULA WE CALCULATE THE
C AREA OF THE TRIANGLE
  799 S = FLOATF (IA + IB + IC) / 2.0
      AREA = SQRTF( S * (S - FLOATF(IA)) * (S - FLOATF(IB)) *
     +     (S - FLOATF(IC)))
      WRITE OUTPUT TAPE 6, 601, IA, IB, IC, AREA
  601 FORMAT (4H A= ,I5,5H  B= ,I5,5H  C= ,I5,8H  AREA= ,F10.2,
     +        13H SQUARE UNITS)
      STOP
      END
```

6.5 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)

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  Multiplier.
AUTHOR.  Michael Coughlan.
  Example program using ACCEPT, DISPLAY and MULTIPLY to
  get two single digit numbers from the user and multiply them together

DATA DIVISION.

WORKING-STORAGE SECTION.
01  Num1                               PIC 9  VALUE ZEROS.
01  Num2                               PIC 9  VALUE ZEROS.
01  Result                             PIC 99 VALUE ZEROS.

PROCEDURE DIVISION.
    DISPLAY "Enter first number  (1 digit) : " WITH NO ADVANCING.
    ACCEPT Num1.
    DISPLAY "Enter second number (1 digit) : " WITH NO ADVANCING.
    ACCEPT Num2.
    MULTIPLY Num1 BY Num2 GIVING Result.
    DISPLAY "Result is = ", Result.
    STOP RUN.
```

6.6 LISP

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

Idees principals:

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

```
(defun factorial (N)
    "Compute the factorial of N."
    (if (= N 1)
        1
        (* N (factorial (- N 1)))))
```

```
(defun first-name (name)
    "Select the first name from a name represented as a list."
    (first name))

(setf names '((John Q Public) (Malcolm X)
              (Admiral Grace Murray Hopper) (Spot)
              (Aristotle) (A A Milne) (Z Z Top)
              (Sir Larry Olivier) (Miss Scarlet)))
```

6.7 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 (≠ pas per referència)
- Recursivitat
- Gramàtica formal (Backus-Naur Form or BNF)

Pas per nom:

```
begin
    integer i;
    integer array A[0:9];

    procedure ini (x); integer x;  (* pas per nom *)
    begin
        for i := 0 step 1 until 9 do x := 0
    end;

    ini(A[i]);
    ...
end
```

El paràmetre `x` es passa per nom (amb `value x;` es passaria per valor).

Dins de `ini()`, aparentment, s'assigna 10 cops 0 a `x`.

Però `x` és el "nom" del paràmetre real, és a dir `A[i]`.

Per tant, `ini()` realment inicialitza a zero tot l'array.

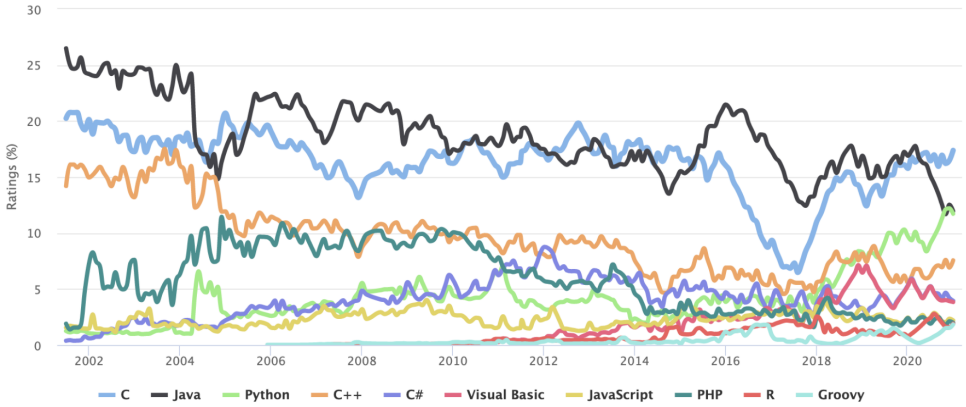
6.8 Altres llenguatges

- 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
- C++ (llenguatge dinàmic i flexible compatible amb C) - 1983
- Java (llenguatge orientat a objectes per a torradores) - 1990
- Python (llenguatge llegible d'alt nivell de propòsit general) - 1991
- …

7. Ús de LPs

Com mesurar la popularitat dels LPs?

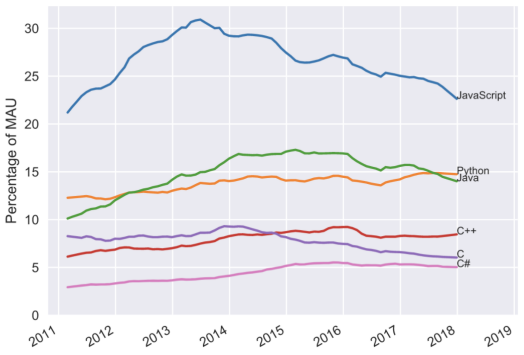
- TIOBE Programming Community Index



- IEEE Spectrum ranking: The Top Programming Languages 2018

Language Rank	Types	Spectrum Ranking
1. Python		100.0
2. C		99.7
3. Java		99.5
4. C++		97.1
5. C#		87.7
6. R		87.7
7. JavaScript		85.6
8. PHP		81.2
9. Go		75.1
10. Swift		73.7

- Estadístiques de GitHub



8. Paradigmes dels LPs

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

Paradigmes comuns:

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

```
f = 1;
while (--n) f *= n;
```

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

```
select full_name, order_date, order_amount
from customers inner join orders
on customers.customer_id = orders.customer_id
```

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

```
cincMesGrans :: [a] -> [a]
cincMesGrans = take 5 . reverse . sort
```

8.1 Paradigma imperatiu

Característiques:

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

Exemples:

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

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

Subclassificacions:

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

```
PROCEDURE swap (VAR a, b: INTEGER)
VAR c: INTEGER;
BEGIN
    c := a;  b := a;  a := c;
END;
```

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

```
Point»dist: aPoint
    dx := aPoint x - x.
    dy := aPoint y - y.
    ↑ ((dx * dx) + (dy * dy)) sqrt
```

8.2 Paradigma declaratiu

Característiques:

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

Subclasificacions:

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

```
Maximize
  x1 + 2 x2 + 3 x3 + x4
Subject To
  - x1 + x2 + x3 + 10 x4 ≤ 20
  x1 - 3 x2 + x3 ≤ 30
  x2 - 3.5 x4 = 0
```

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

```
SELECT full_name, order_date, order_amount
FROM customers INNER JOIN orders
ON customers.customer_id = orders.customer_id
```

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

```
human(socrates).
mortal(X) :- human(X).
?- mortal(socrates).
```

8.3 Paradigma funcional

Característiques:

- 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),...

Conceptes clau:

- **Recursivitat**:

```
(define (fib n)
  (cond
    ((= n 0) 0)
    ((= n 1) 1)
    (else
      (+ (fib (- n 1))
         (fib (- n 2))))))
```

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

```
map(lambda x: 2*x, [1,2,3,4])
```

- **Aplicació parcial** (currying):

```
doble = (*2)
```

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

8.4 Paradigma OO

Característiques:

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

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

9. Llenguatges multiparadigma

Mols LPs combinen diferent paradigmes. Per exemple:

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

Alguns LPs incorporen característiques d'altres paradgimes:

- Prolog: lògic (+ imperatiu + funcional).

Altres combinacions:

- Erlang: funcional + concurrent + distribuït.

9.1 Llenguatges visuals

En un llenguatge de programació visual (VPL), els programes són creats manipulant elements gràfics.

- Visual Basic: Dialecte de BASIC utilitzant una interfície gràfica per facilitar la creació d'interfícies gràfiques (1991).
- Scratch: Eina d'educació per a nens a partir de 8 anys (2003).

9.2 Llenguatges esotèrics

- Brainfuck: basat en màquines de Turing, només té 8 instruccions.
- Piet: usa imatges amb 20 colors com a codi font.
- Whitespace: només té espais en blancs i tabuladors.
- Shakespeare: amaga un programa dins d'una obra de teatre.

9.3 Turing completesa

Màquina de Turing: model matemàtic de càlcul imperatiu molt simple (Alan Turing, 1936):

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

Tesi de Church-Turing: "Tot algorisme és computable amb una màquina de Turing o amb una funció en ambda-càlcul."

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

LPs Turing complets:

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

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

LPs no Turing complets:

- expressions regulars (Perl o AWK)
- ANSI SQL

10. Sistema d'execució

Pot ser:

- Compilat: el codi es transforma en codi objecte i després es monta en un executable. Sol ser eficient. Es distribueix l'executable. Ex.: 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. Augmenten la portabilitat i l'expressivitat (es poden fer més coses en temps d'execució), però disminueix l'eficiència. Ex.: 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).

El sistema d'execució depèn més de la implementació que del LP.

11. Sistema de tipus

Un sistema de tipus és un conjunt de regles que assignen tipus 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.

```
class Persona;
class Forma;
class Rectangle :Forma;
class Triangle  :Forma;

double area (const Forma&);
```

- cridar a `area` amb un `Rectangle` o un `Triangle` és correcte,
- cridar a `area` amb una `Persona` o un enter és un error de tipus.

11.1 Error de tipus

Un error de tipus consisteix en aplicar a les dades una operació que el seu tipus no soporta.

- Type cast: usar valor d'un tipus en un altre tipus.
- Aritmètica de punters.
- Alliberament explícit de memòria (deallocate/delete): per exemple, en Pascal usar un apuntador alliberat pot donar error de tipus.
- En llenguatges OO (antics), no existència d'un mètode (degut a l'herència).

11.2 Seguretat de tipus

La seguretat de tipus (type safety) és la mesura de com de fàcil/difícil és cometre errors de tipus en un LP.

Exemples:

- C té reputació de ser un LP sense gaire seguretat de tipus (unions, enumerats, void*, ...).
- C++ hereta C, però proporciona més seguretat de tipus: els enumerats no es poden convertir implícitament amb els enters o altres enumerats, el dynamic casting pot provocar errors de tipus en temps d'execució, …
- Java és dissenyat per a proporcionar seguretat de tipus. Però pot abusar del classloader.
- Python, igual que Java, està dissenyat per donar seguretat de tipus, malgrat que el tipatge sigui dinàmic.
- Es creu que Haskell és type safe si no s'abusa d'algunes construccions com unsafePerformIO.

11.3 Tipat fort *vs* feble

Tipat fort (strong typing): L'LP imposa restriccions que eviten barrejar valors de diferents tipus (conversions explícites).

Tipat feble (weak typing): L'LP té regles màgiques per convertir tipus automàticament.

Javascript

```
4 + '7';      // '47'
4 * '7';      // '28'
```

PHP

```
4 + '7';      // '11'
4 * '7';      // '28'
```

Python

```
4 + '7'       # ❌ TypeError: unsupported operand type(s) for +: 'int' and 'str'
4 * '7'       # '7777'
```

11.4 Comprovació

Pot ser:

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