Programació:

La programació és el procés de dissenyar, codificar, depurar i mantenir el codi font de programes computacionals.

El codi font és escrit en un **llenguatge de programació**.

El propòsit de la programació és crear programes que exhibeixin un comportament desitjat.

El procés d'escriure codi requereix freqüentment coneixements a diverses àrees diferents, a més del domini del llenguatge a utilitzar, algorismes especialitzats i lògica formal.

Programar no involucra necessàriament altres tasques tals com l'anàlisi i disseny de l'aplicació (però sí el disseny del codi), encara que sí solen estar fusionades en el desenvolupament de petites aplicacions.

Llenguatges de programació:

La programació es regeix per regles i un conjunt més o menys reduït d'ordres, expressions, instruccions i comandes que tendeixen a assemblar-se a una llengua natural fitada (en anglès); i que a més tenen la particularitat d'una reduïda ambigüitat.

Com menys ambigu és un llenguatge de programació, es diu, és més potent.

Sota aquesta premissa, i en l'extrem, el llenguatge més potent existent és el binari, amb ambigüitat nul·la (la qual cosa porta a pensar així del llenguatge assemblador).

En els llenguatges de programació d'alt nivell es distingeixen diversos elements entre els quals s'inclouen el lèxic propi del llenguatge i les regles semàntiques i sintàctiques.

Classificació segons nivell jeràrquic:

Llenguatges de Baix nivell:

Primera Generació: Llenguatge de Màquina, és el més bàsic però en realitat l'únic que comprèn l'ordinador.
se centra en codi binari, la resta de llenguatges actuen traduint el codi.

Segona Generació: Llenguatge d'assemblador, conté la mateixa estructura i variables que el llenguatge màquina però utilitza noms i variables en lloc de nombres, que un programador pot manipular i definir.
El llenguatge es limita al tipus de màquina per la qual es desenvolupa i ha de ser modificat per a altres ordinadors.

Classificació segons nivell jeràrquic:

Alt nivell (o **Tercera Generació**), que permeten al programador crear programes més o menys independents del tipus d'ordinador.

Són llenguatges més fàcils d'administrar, d'entendre i actualitzar perquè segueixen unes ordres més similars al llenguatge humà.

Exemples:

Basic, C, C++, Delphi, Java, Lisp, Pascal, Perl, Pyton, Ruby….

Quarta Generació: llenguatges de quarta generació. són els més propers a la sintaxi de la llengua humana, i s'acostumen a utilitzar en les creacions de bases de dades o com a llenguatges de programació dels llenguatges o sistemes d'autor.

Classificació segons l'ús d'instruccions:

Llenguatges imperatius, en els que els conceptes es tradueixen de forma natural dins la màquina, com FORTRAN,C, PASCAL...

Llenguatges Funcionals, en els que es construeixen funcions a partir d'altres ja existents, i permeten resoldre problemes matemàtics i simbòlics, com LISP, ML, Haskell, Mercury...

Llenguatges lògics, centrats en el concepte matemàtic de lògica de primer ordre, en què es defineixen una sèrie de declaracions a les que assignem valors per obtenir resultats, com Prolog, Mercury.

Llenguatges Orientats a Objectes, orientats a solucionar problemes del món real, defineixen objectes i les respectives classes per simbolitzar situacions donades, aprofitant que les classes són reutilitzables.
Un dels primers va ser SIMULA, i més actuals són ADA95, C++ i JAVA .

Compilació o interpretació del codi font:

Una vegada realitzat un programa amb un llenguatge de programació, l'hem de **traduïr** de tal manera que les diferents màquines el puguin executar.

A aquest procés se li anomena **compilació,** i s'obté com a resultat un arxiu executable (. exe a windows).

En alguns casos, no cal compilar el nostre codi, sino que farem servir programes que **l'interpretaran** i ens donaran el resultat per pantalla, com els navegadors web (interpreten HTML i javascript), o la màquina virtual de java, que interpreta els programes fets amb java (.jar).

Conceptes de programació.

Comentaris:

En tots els llenguatges de programació tenim la possibilitat de insertar comentaris dintre del codi font per poder explicar parts del codi.

Normalment els comentaris d'una linea es defineixen amb dues barres.

```
//comentari d'una linea
```

Quan volem introduïr comentaris multilinea es fa entre /* */

Alguns llenguatges tenen formes peculiars de comentar el codi, com HTML que ho fa entre els tags <!-- i -->

Variables:

En programació, les **variables** són espais reservats en la memòria que, com el nom indica, poden canviar el valor contingut al llarg de l'execució d'un programa.

A l'hora de fer servir les variables, hem de definir el **tipus de dades** que contindrá, i el nom pel qual la coneixerem dins del programa.

La manera genérica de **definir** una variable es:

```
tipus_de_dades nom_variable;
```

Ex:

```
int numero;
char lletra;
```

Variables:

La menera de representar la definició de les variables pot variar en funció del llenguatge que estem fent servir. Així, per exemple:

Javascript:	var qualsevol_tipus_dades;
PHP:	$nom_de_variable = ‘valor’;
Python (entre d’altres):	nom_de_variable = valor (es defineix quan se li dona un valor)

Constants:

En prácticament tots els llenguatges de programació disposem de Constants, per representar valors que a diferéncia de les variables, No seran modificats en cap moment de la execució del programa.

Les podem definir nossaltres mateixos, o poden venir definides de serie pel mateix programa.

Ex:

En C++: **#define** PI 3.1415926
En Java: **final** int DIAS_SETMANA = 7;

Típus de dades:

El tipus d'una dada és el conjunt de valors que pot prendre la variable durant el programa.

Si se li intenta donar un valor fora del conjunt es produirà un error.

L'assignació de tipus a les dades té dos objectius principals:

D'una banda, detectar errors en les operacions
Per l'altre, determinar com executar aquestes operacions

Tipus de dades primitius:

CARÀCTER:

El Tipus de dada char és un dígit individual el qual es pot representar com numèrics (0 al 9), lletres (az) i símbol ($, _).

char -Rang:0-65.536 (mida = 16 bits)

Ex: char lletra = ‘a’;

Tipus de dades primitius:

NUMÈRICS:

Aquest Tipus de dada pot ser enter o real, depenent del tipus de dada que s'hagi d'utilitzar.

Enters són els valors que no tenen punt decimal, poden ser positius o negatius i el zero.

byte (mida = 8 bits)
short (mida = 16 bits)
int (mida = 32 bits)
long (mida = 64 bits)

Reals aquests caràcters s'emmagatzemen nombres molt grans que posseeixen part entera i part decimal.

float = 32 bits
double = 64 bits

Típus de dades primitius:

Booleans

Aquest Tipus de dada s'empra per a valors lògics, els podem definir com dades comparatives aquesta comparació retorna sempre resultats lògics (cert o fals).

boolean -Rang: true - false

Tipus de dades complexes:

Estan formats per varis elements de tipus simple.

Cada llenguatge de programació té els seus tipus complexes Particulars, però pràcticament tots comparteixen:

String: Cadena d'elements de tipus char.

Ex: String nom= 'Lluis';

Array: Estructura ordenada n-dimensional complexa on s'enmagatzemen diversos elements.

Ex: char[] i = new char[5];

Clases: Tipus de dades personalitzats, creats pel programador

Operadors i relacionals:

Operadors aritmètics

Els operadors aritmètics són aquells que "manipulen" dades numèriques, tant senceres com a reals.

\+ suma
-resta

* multiplicació

/ divisió.

% mòdul (resta de la divisió)

Operadors i relacionals:

Operadors aritmètics

L'operador aritmètic + també es pot fer servir amb dades de tipus *string.

El resultat és la concatenació de les cadenes. Per exemple:

```
String cadena;

cadena = "Això";
cadena = cadena + " és una concatenació de" ;
cadena = cadena + " cadenes";
```

Operadors i relacionals:

Operadors relacionals

Els operadors relacionals ens permeten comparar expressions de tipus compatibles, retornant un resultat de tipus lògic: la comparació és veritable o falsa.

Per exemple, l'expressió (3 + 4) > 12 seria falsa, ja que 7 és menor que 12.

L'estructura d'una operació relacional seria la següent:

expressió_1 operador_relacional expressió_2

Els operadors relacionals més comuns són: **< , >, <=, >=, !=, ==**

Operadors i relacionals:

Operadors lògics:

Serveixen per fer operacions lògiques a l'hora de avaluar una operació.

OR (||) : Per obligar a que es compleixi al menys una sentencia**.**

AND (&&) : per exigir que es compleixin varies sentencies

NOT (!) : Per negar una senténcia llògica

Condicionals:

En programació, una sentència **condicional** és una instrucció o grup d'instruccions que es poden executar o no en funció del valor d'una condició.

Els tipus més coneguts de sentències condicionals són el

SI..LLAVORS (if..then),

SI..LLAVORS..SI NO (if..then..else)

SEGONS (case o switch)

Les sentències condicionals constitueixen, juntament amb els bucles, els pilars de la programació estructurada

Condicionals:

SI..LLAVORS

Executarem les sentencies dintre de la condició sii es compleix la nostra condició.

La estrucutura bàsica seria la següent:

```
SI (llista de condicions)
LLAVORS { llista d'accions}
```

Ex:

```
if (Cont >=50 && pucAccedir) {
        Cont = 0:
        accedirAFitxer("fitxer1.txt);
}
```

Condicionals:

SI..LLAVORS… SI NO

Executarem les sentencies dintre de la condició sii es compleix la nostra condició si no es compleix executarem altres sentencies.

La estrucutura bàsica seria la següent:

```
SI (llista de condicions)
LLAVORS { llista d'accions}
SI NO{ llista d'accions}
```

Ex:

```
if (Cont >=50 && pucAccedir) {
        Cont = 0:
        accedirAFitxer("fitxer1.txt);
} else
        { tancarPrograma(); };
```

Condicionals:

SEGONS

Avaluem una condició, en funció del resultat, farem una cosa o farem una altra.

La estrucutura bàsica seria la següent:

SEGONS (condició)

CAS1: accions;
…
CAS N:accions;

Iteracions:

En programació, la iteració és la repetició d'un procés dins d'un programa d'ordinador tantes vegades com volem.

Hi ha dos maneres de aconseguir aquesta repetició, mitjançant l'estrucutura de control **WHILE** o mitjançant l'estructura de control **FOR**

Iteracions:

WHILE:

El **Bucle while** o **bucle mentre** és una estructura de la majoria dels llenguatge de programació estructurat el propòsit de la qual és repetir un bloc de codi mentre una condició es mantingui veritable.

La sintaxi en pseudocodi és la següent:

MENTRE condició FER
instruccions
FI MENTRE

Ex:

```
int i = 0;
while (i < 80) {
        line(30, i, 80, i);
        i = i + 5; }
```

Iteracions:

FOR:

El **bucle for** o **cicle for** és una estructura de control en la qual es pot indicar el nombre mínim d'iteracions.
Està disponible en gairebé tots els llenguatges de programació imperatius.

PER **i** desde X fins a N, a increments de s FER

instruccions

fi PER

On **i** es la variable de control que fem servir per realitzar la iteració.

Sub-rutines o funcions:

En computació, una **subrutina, funció** o **subprograma**, com a idea general, es presenta com un subalgorisme que forma part de l'algorisme principal, el qual permet resoldre una tasca específica.

Una subrutina en ser cridada dins d'un programa fa que el codi principal es detingui i es dirigeixi a executar el codi de la subrutina.

S'ha de definir la subrutina especificant qué retornarà a la sortida, i quins paràmetres d'entrada rebrà:

retorna nomFunció (paràmetres_entrada) {

Codi de la funció

}