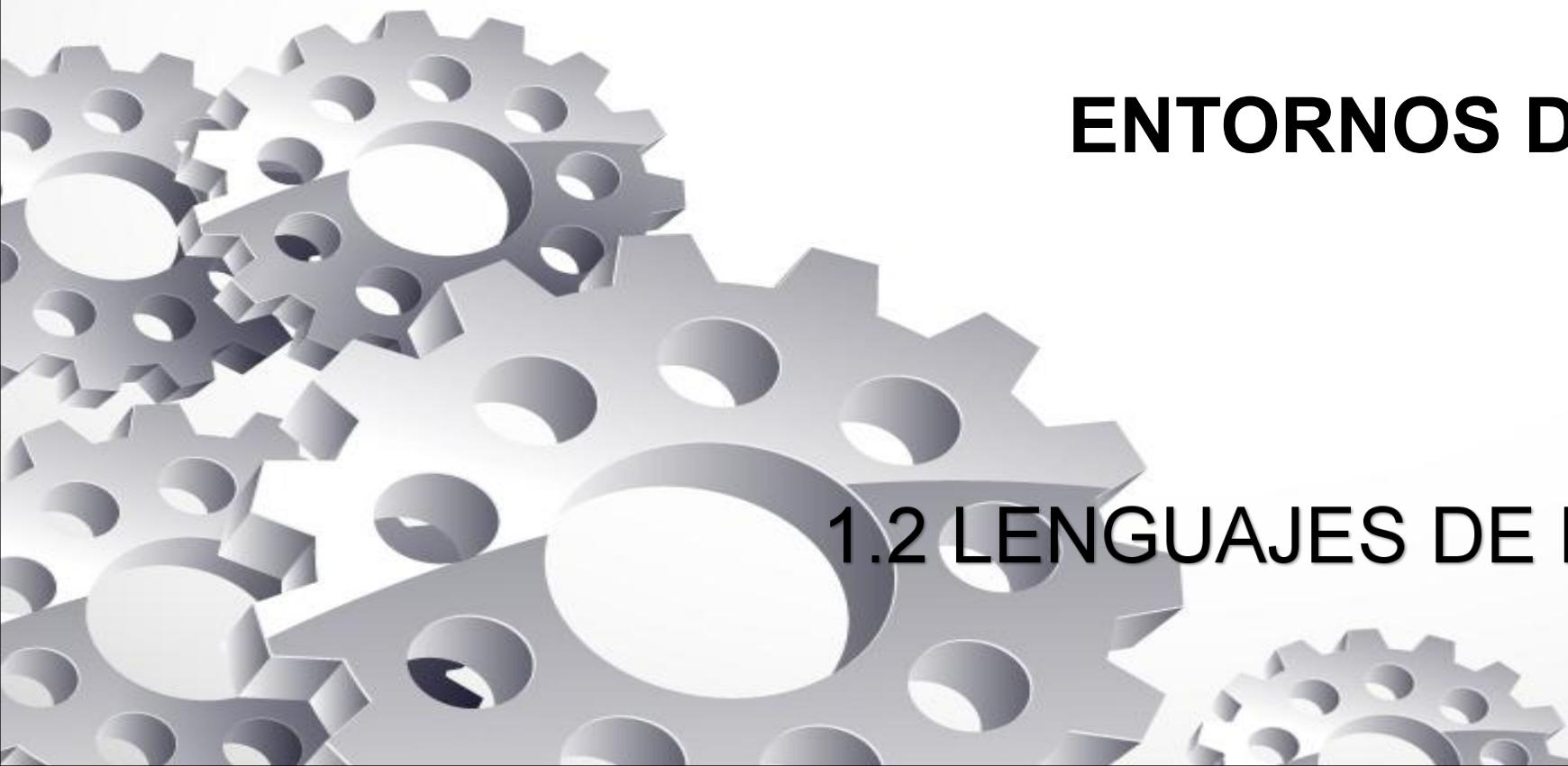


UT 1: Desarrollo de software



ENTORNOS DE DESARROLLO

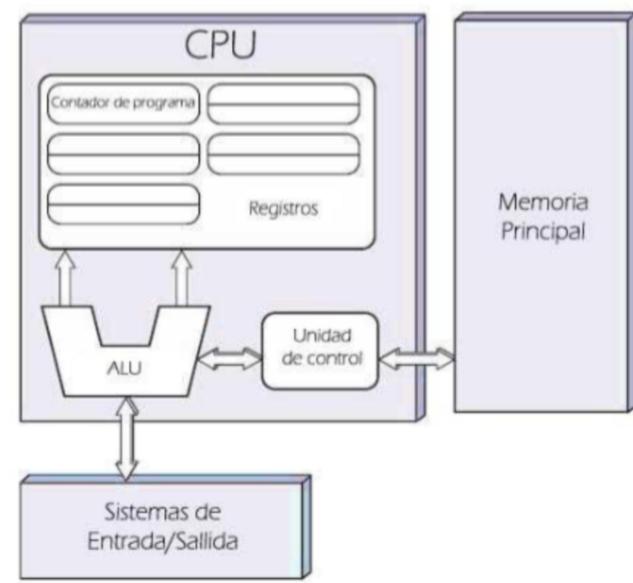
1.2 LENGUAJES DE PROGRAMACIÓN

1. Introducción



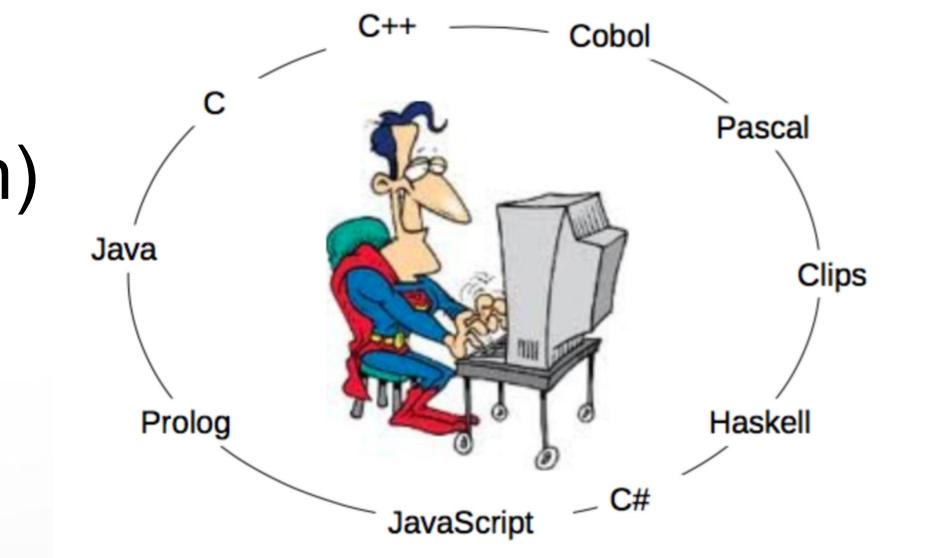
- Las computadoras están formadas por componentes de naturaleza electrónica → solo entienden 0's y 1's.
- Las tareas que pueden llevar a cabo un procesador se codifican como una combinación de los mismos en forma de instrucciones (lenguaje máquina).

1011 00000010 00000001
| | |
| | +++++++ Operando 1
| | +++++++ Operando 2
|++++++ Sumar



1. Introducción

- Lenguaje de programación: idioma artificial diseñado para expresar procesos que controlan el comportamiento lógico y físico de una máquina.
- Como todo lenguaje:
 - Alfabeto (símbolos)
 - Sintaxis (reglas para su combinación)
 - Semántica (asociadas a las construcciones sintácticas)



1. Introducción

- ## Programación

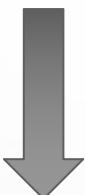
“Proceso por el cual se escribe, se prueba, se depura, se compila (de ser necesario) y se mantiene el código fuente de un programa informático.”



2. Historia y generaciones



Automatizar trabajos



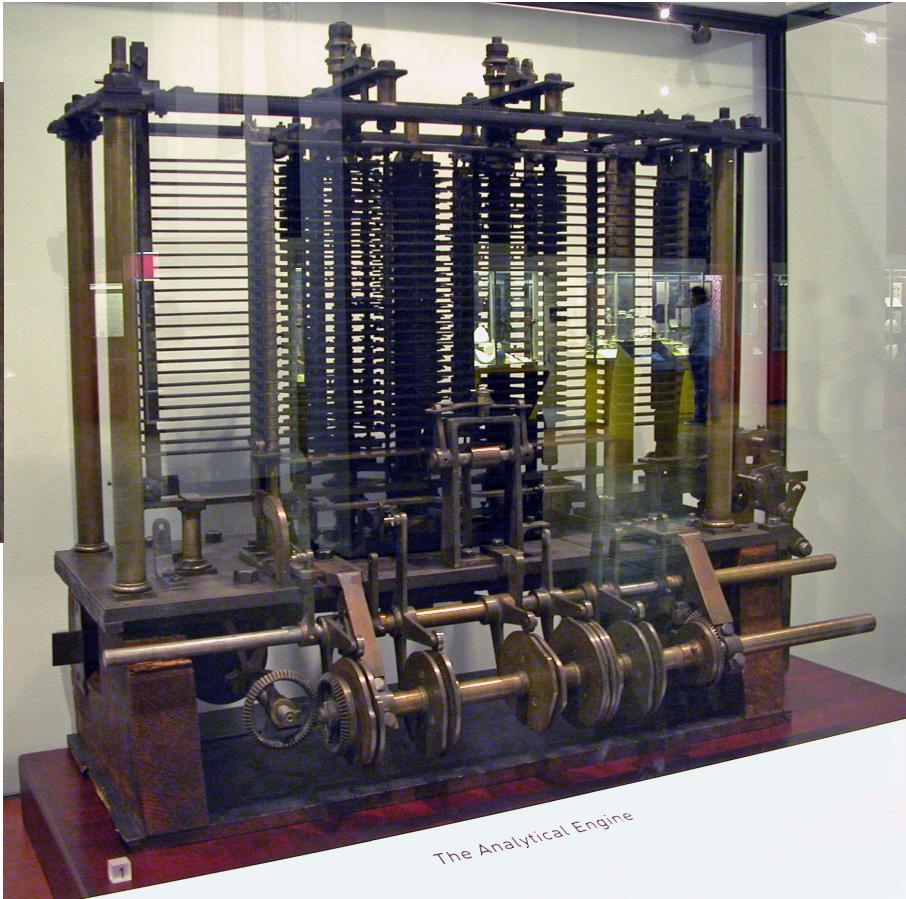
Comunicación con máquinas



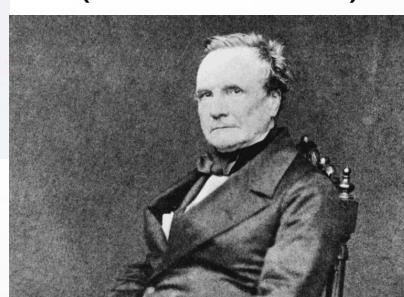
2. Historia y generaciones



Ada Lovelace
(1815 - 1852)



De Bruno Barral (ByB), CC BY-SA 2.5,
<https://commons.wikimedia.org/w/index.php?curid=6839854>



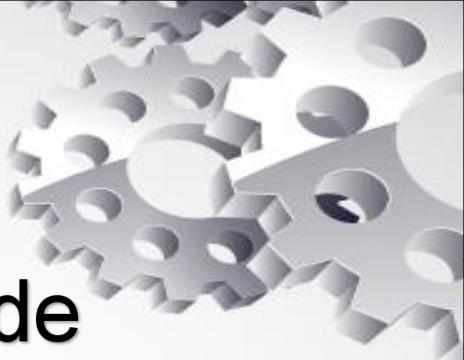
Charles Babbage
(1792 - 1871)

2. Historia y generaciones

- 1a Generación (1940-1950)
 - Códigos (instrucciones) interpretables por circuitos microprogramables.
 - Sistemas digitales (dos únicos niveles de tensión)
 - Lenguajes máquina



2. Historia y generaciones

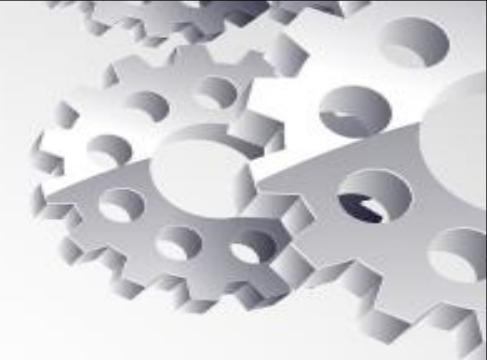


```
.file  "hola.c"
.section .rodata
.LC0:
.string "%d"
.text
.globl main
.type  main, @function
main:
.LFB0:
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset %rbp, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
subq $16, %rsp
movl $3, %edx
movl $2, %eax
addl %eax, %edx
;; movl $.LC0, %eax
;; movl %edx, %esi
```

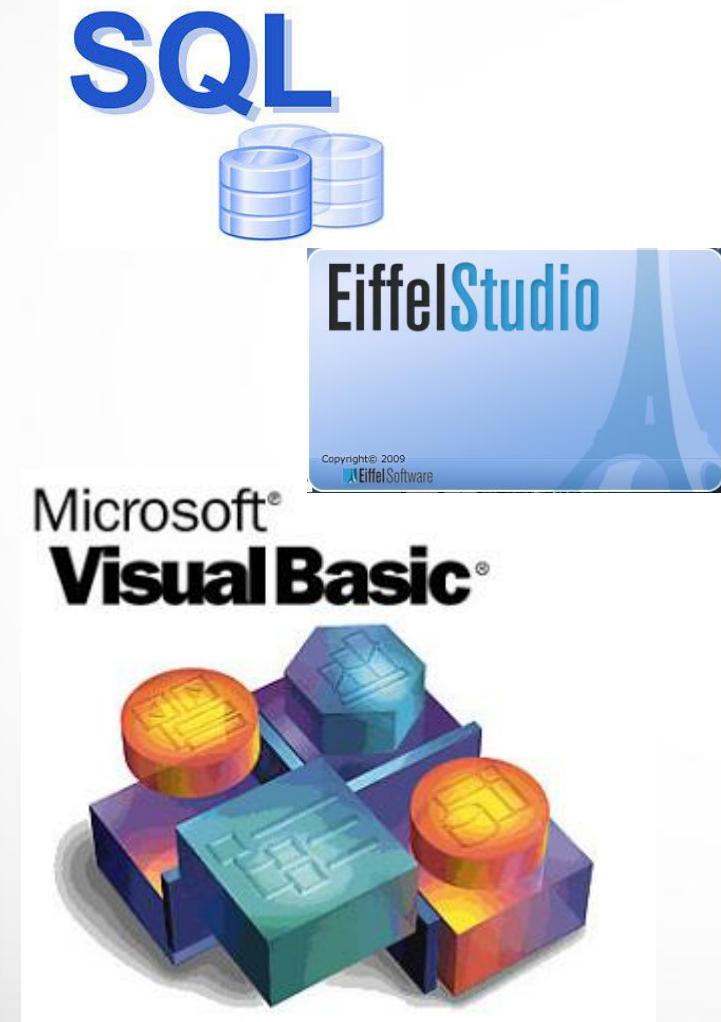
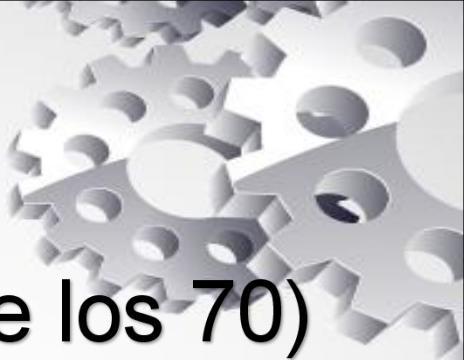
- 2a generación (finales de los 50)
 - Lenguajes simbólicos
 - Específico también de cada máquina
 - Instrucciones pueden ser cargadas en memoria (de sistema basado en microprocesador)
 - Ensamblador (mnemotécnicos)

2. Historia y generaciones

- 3a generación (a partir de los 60)
 - Lenguajes de alto nivel
 - Próximos al lenguaje natural
 - Traductor a lenguaje máquina
 - Fáciles de aprender y usar
 - Estructurados y modulares



2. Historia y generaciones



- 4a generación (a partir de los 70)
 - Se especifica “qué” se debe hacer, en lugar del “cómo”.
 - Más alto nivel que 3a gen.
 - Ajenos a procedimientos
 - Acceso a bases de datos
 - Lenguajes declarativos: SQL
 - Lenguajes visuales: Visual Basic, Visual C
 - Orientados a objetos: C++, Java, Eiffel

2. Historia y generaciones

- 5a generación (principios de los 80)
 - Qué se ha de resolver y en qué condiciones
 - Inteligencia artificial (imitar a la mente humana)
 - Mayor legibilidad, portabilidad, facilidad de aprendizaje y de modificación.



SWI Prolog



2. Historia y generaciones

- **Posteriormente**
 - Era de Internet (los 90): Python, HTML, Ruby....
 - Programación funcional: Haskell...
 - Programación concurrente y distribuida: Ada....
 - Mayor movilidad y distribución
 -



3. Clasificación

- No existe estipulada ninguna clasificación estricta
- Trataremos de esquematizar los tipos y características atendiendo a:
 - Nivel de Abstracción
 - Cercanía de la máquina en la que se ejecutan (bajo / alto nivel)
 - Implementación
 - Tipo de programa utilizado para la traducción al lenguaje máquina (compilado/interpretado/mixto)



3. Clasificación



3.1 Nivel de Abstracción

- Lenguajes de bajo nivel:

- Alta dependencia de la máquina en la que se ejecuta.
- El programa ha de ser reescrito para poder ser ejecutado en distintos procesadores.
- Lenguaje ensamblador
- Se trabaja con etiquetas nemotécnicas y direcciones simbólicas.



Add rax, 25 // rax = rax+25
| | |
| | +++++++ número 25
| | ++++++ Registro rax
| ++++++ Operación suma

Ej. Instrucción suma arquitectura x86-64 (Amd64 o Intel 64)

3. Clasificación



3.1 Nivel de Abstracción

- Lenguajes de alto nivel:
 - Cercanos al lenguaje natural
 - Independientes de la máquina en la que se va a ejecutar.
 - Se trabaja con variables, expresiones, funciones hilos y bucles en lugar de registros, direcciones de memoria y códigos de operación.
 - 1 Instrucción → Múltiples instrucciones en código máquina



3. Clasificación



3.1 Nivel de Abstracción

- Lenguajes de alto nivel:

```
8 void showTable2()
9 {
10     for (int i = 0; i < 10; i++)
11     {
12         printf("2 x %d = %d \n", i, 2 * i);
13     }
14 }
15
16 int main(int argc, char const *argv[])
17 {
18     char message[12] = "Hello world";
19     greet(message);
20     showTable2();
21     return 0;
22 }
```



```
//add ad campaign
public function add(){

    $adInfo = $this->getData();

    unset($adInfo['u_id']);

    $adInfo['id_user'] = $this->userId;

    $campaignId = $adInfo['id_campaign'];

    if(isset($adInfo['user_not_null'])){
        $adInfo['user_not_null'] = InputSanitizeHelper::getJSONArrayFromString($adInfo['user_not_null']);
    }

    $cCampaignBackService = new CCampaignBackService($this->db);

    return $cCampaignBackService->addAd($adInfo, $campaignId);

}
```

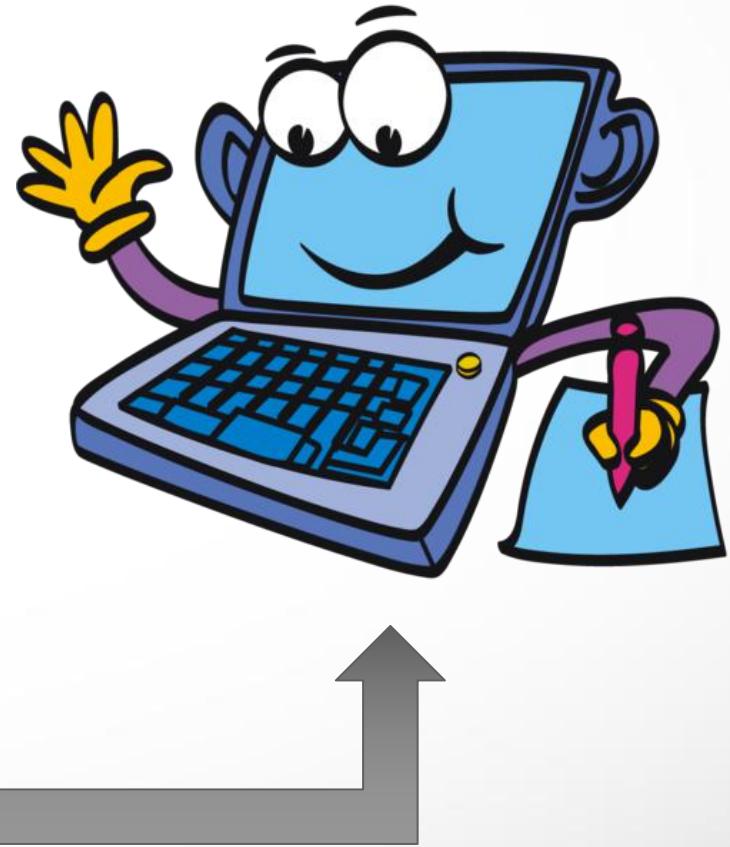
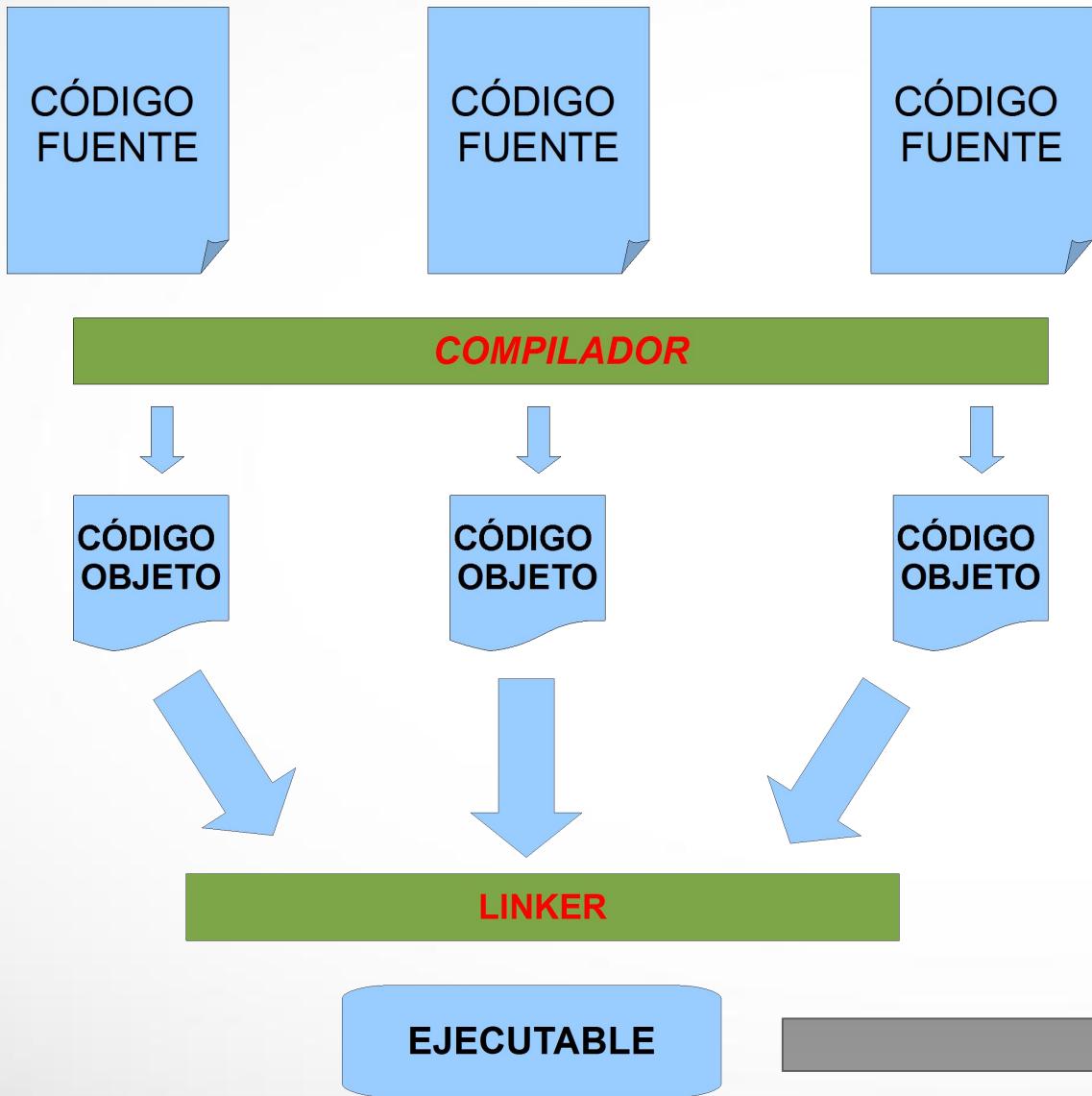
3. Clasificación



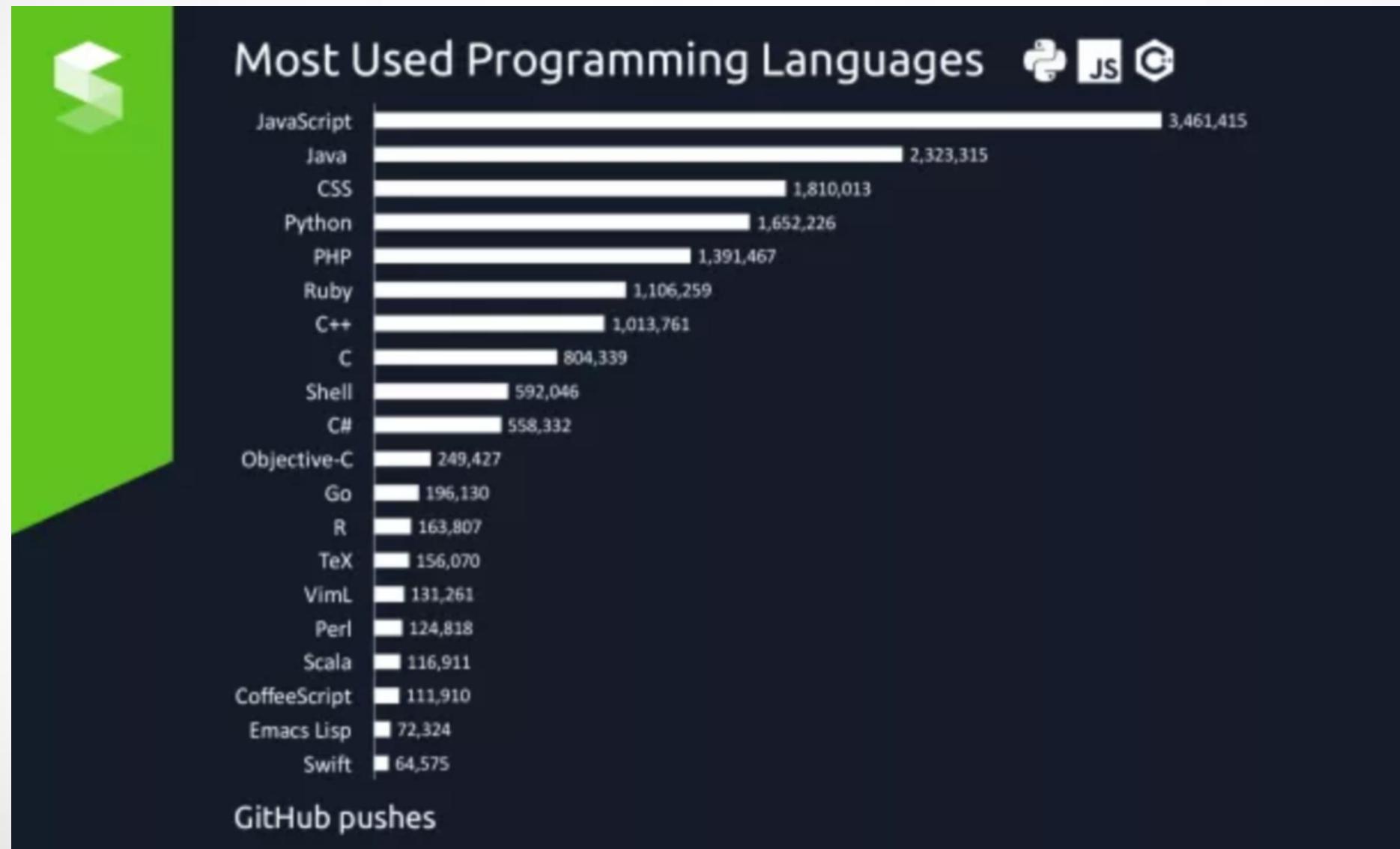
3.2 Implementación

- Lenguajes compilados
 - El programa ha de ser traducido a lenguaje maquina en una fase previa a la ejecución.
 - Comprobación de tipos y errores léxicos, sintácticos y semánticos en una fase previa a la ejecución.
 - Mayor rapidez de ejecución vs mayor tiempo de desarrollo.
 - Han de ser compilados para cada plataforma en la que queramos ejecutar el software

3. Clasificación

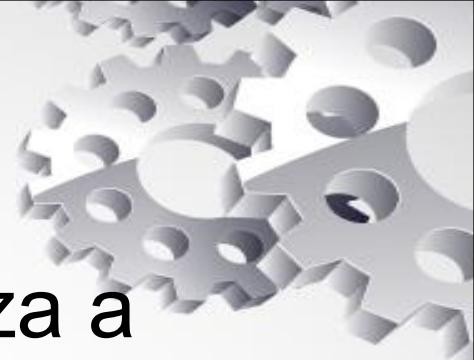


4. Lenguajes más utilizados



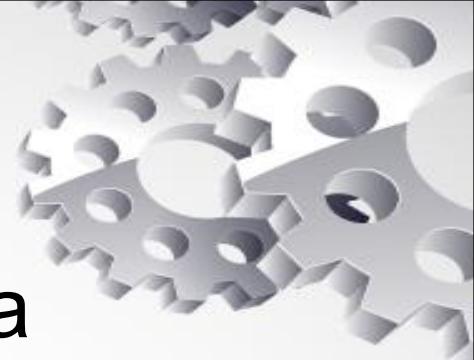
5. Tipos de código.

- El código cambia de estado desde que se empieza a escribir hasta que está listo para ser ejecutado.
- **Código fuente:** Conjunto de instrucciones ordenadas escritas en un lenguaje de programación (editor)
- **Código objeto:** Código resultante de compilar el código fuente. (compilador)
 - Lenguaje compilado -> código máquina
 - Lenguaje virtual o mixto -> bytecode
- **Código ejecutable:** Programa que se ejecuta directamente en el sistema



5. Tipos de código.

- Para convertir el programa fuente en un programa entendible por el ordenador es necesario utilizar “traductores”. Existen 3 tipos:
 - ***Ensambladores***: traducen el código fuente escrito en ensamblador a código máquina ejecutable por el procesador.
 - ***Intérpretes***: Se encargan de traducir cada instrucción o sentencia a lenguaje máquina y ejecutarla.
 - ***Compiladores***: Convierten código fuente en lenguaje máquina o en un código intermedio (bytecode).



6. Paradigmas de programación

- La computación cada vez abarca más ámbitos de nuestra vida diaria
- Software más complejo y con mayores requerimientos
- Proporcionan diferentes enfoques para abordar un problema
- Define un conjunto de reglas, patrones y estilos de programación.



6. Paradigmas de programación

- Gran cantidad de lenguajes
- Si solo conocemos **un lenguaje** de programación o paradigma corremos el riesgo de **quedarnos obsoletos** y crear **software de baja calidad**.
- Si aprendemos **programación por conceptos** seremos capaces de elegir el paradigma que mejor se adapte a nuestro proyecto



6. Paradigmas de programación



- Debemos tomar la píldora adecuada

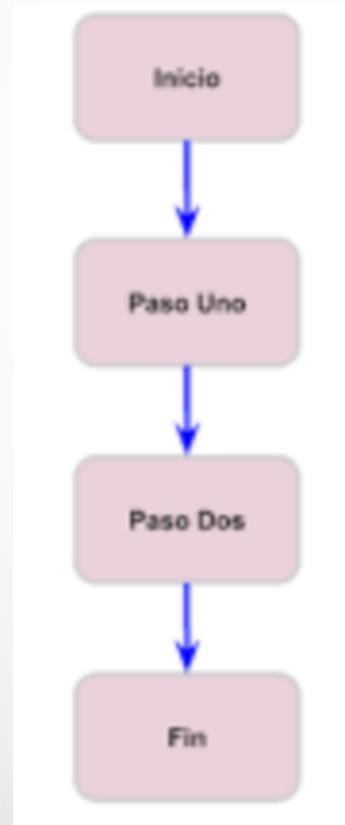


6. Paradigmas de programación

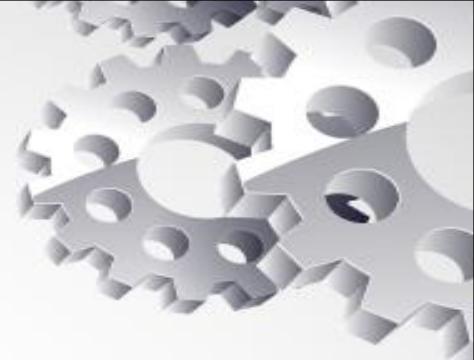


6.1 Paradigma imperativo

- El programa se basa en una serie de sentencias que establecen explícitamente como la computadora ha de resolver un problema



6. Paradigmas de programación



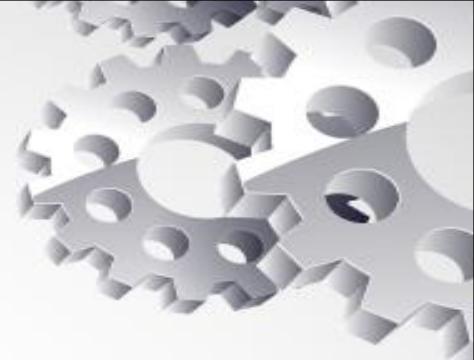
6.1.1 Programación estructurada

- Se emplea el uso de estructuras de control jerárquicas (1 solo punto de entrada y 1 de salida)
- Estructuras lógicas de secuencia, selección condicional e iteración
- Actualmente incluye al programación **modular** y **procedimental**

```
#include <stdio.h>

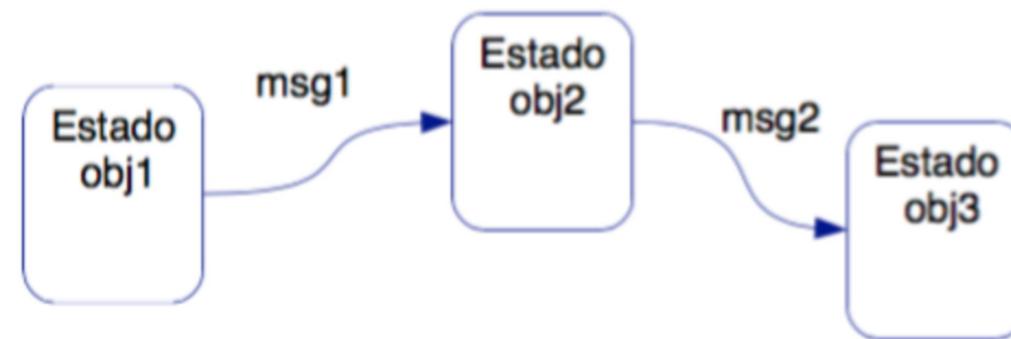
int getFibonacci(int n){
    if (n==0) {
        return 0;
    }else if (n==1){
        return 1;
    }
    return getFibonacci(n-2)+getFibonacci(n-1);
}
int main(int argc, char const *argv[]){
    getFibonacci(10);
}
```

6. Paradigmas de programación



6.1.2 Programación Orientada a Objetos (POO)

- Extensión del **modelo imperativo**.
- **Conjunto de objetos** con un **estado** y un **comportamiento** que **interaccionan** entre sí para llevar a cabo las tareas que requiere el programa.
- Sigue teniendo un enfoque imperativo.



6. Paradigmas de programación



6.2 Declarativos

- Orientados a buscar la solución del problema sin preocuparse por la forma de llegar a ello.
- Le indicamos el "que" dejando al intérprete el "como"
 - Lenguajes lógicos
 - Lenguajes funcionales

6. Paradigmas de programación



6.2.1 Declarativos

– Funcional

- Funciones matemáticas que se le aplican sucesivamente a los datos.

Fib :: int → int

fib 0 = 0;

fib 1 = 1

fib n = fib(n-2)+fib(n-1)

VS

```
int getFibonacci(int n){  
    if (n==0) {  
        return 0;  
    }else if (n==1){  
        return 1;  
    }  
    return getFibonacci(n-2)+getFibonacci(n-1);  
}  
int main(int argc, char const *argv[]){
```

- Existen enfoques funcionales híbridos menos pragmáticos con los que intentaremos trabajar en programación mediante **java**

6. Paradigmas de programación



6.2.1 Funcionales

- Java enfoque funcional vs híbrido

Calculo media de edad personas > 18 años

```
int totalEdad=0;  
int totalPersonas=0;
```

Imperativo

```
for (Persona p: lista) {  
  
    if (p.getEdad()>=18) {  
  
        totalEdad+=p.getEdad();  
        totalPersonas++;  
    }  
}
```

FUNCIONAL

(más cercano al pensamiento humano)

```
resultado=list.stream().filter(persona->persona.getEdad()>=18)  
.mapToInt(persona->persona.getEdad())  
.average();
```

6. Paradigmas de programación



6.2.2 Lógicos

- Definen un conjunto de hechos y reglas lógicas (predicados) que definen la base del conocimiento.
- Muy utilizados en robótica (PROLOG)

Hecho	Regla
esPato(lucas). % lucas es un pato	
esPato(donald). % donald es un pato	
esPato(gilito). % gilito es un pato	
sobrino(jorgito,donald). % jorgito es sobrino de Donald	esPato(S) :- sobrino(S,T), esPato(T). % Si es sobrino de un pato será un pato
sobrino(jaimito,donald). % jaimito es sobrino de Donald	esPato(P) :- tienePlumas(P), haceCuac(P).
tienePlumas(daisy).	% Es pato si tiene plumas y hace cuac
tienePlumas(piolin).	cuac

```
?- esPato(Jorgito).
Yes
?- esPato(X) % listado de todos los patos
lucas;
donald;
gilito;
daysy
?- esPato(piolin)
No
```

6. Paradigmas de programación



6.2.2 Lógicos

Ej. Robot

– *En el campo de la robótica los individuos constantemente se hacen la pregunta "¿Qué hago ahora?" La respuesta se consigue a través de 3 fuentes; la información recibida de los sensores, la memoria y las órdenes de los seres humanos que constituyen la base de conocimiento de forma que el intérprete Prolog pueda inferir la respuesta."*

7. Tipificación

- Representa las reglas para la manipulación de los tipos de datos.
 - **Estático vs dinámico:** Comprobación en tiempo de compilación o ejecución.
 - **Débil vs fuerte:** Si el lenguaje ejecuta conversiones implícitas (débil) para evitar errores.
 - **Implícito vs explícito:** El tipo de las variables es declarado de forma explícita o se infiere automáticamente.

Tipos de Datos	Memoria
boolean	1
byte / unsigned char	1
char	1
int	2
word / unsigned int	2
long	2
unsigned long	4
float / double	4
string	11
array	11