# Índice

1. Introducción....1
2. Lenguajes de Programación....3
3. Tipos y Características....3
3.1 Abstracción....3
3.1.1 Lenguajes de bajo nivel....4
3.1.1.1 Lenguaje máquina....4
3.1.1.2 Lenguaje ensamblador....4
3.1.2 Lenguajes de alto nivel....5
3.2 Implementación....5
3.2.1 Compilados....5
3.2.2 Interpretados....6
3.2.3 Mixtos....7
3.3 Paradigma de programación....7
2.3.1 imperativos....7
3.3.1.1 Estructurados....8
3.3.1.3 Orientados a objetos....9
3.3.3 Declarativos....10
3.3.3.1 Funcionales....10
3.3.3.2 Lógicos....11
3.4 Sistema de tipos. Tipificación....12
4. Conclusión....13
5. Bibliografía....13

# 1. Introducción

Los computadores están compuestos por una serie de unidades funcionales que desarrollan tareas especificas pero a su vez complementaria. En su diseño más simple y siguiendo el modelo de Von Newman estas unidades son: memoria principal, unidad de control, unidad aritmético lógica y los sistemas de entrada/salida. Estas unidades se encargan de interpretar y ejecutar una serie de instrucciones que son cargadas en memoria secuencialmente y que constituyen lo que denominamos un programa.

La máquina de Von Newman como todas sus sucesoras están compuestas por componentes de naturaleza electrónica por lo que desde un punto de vista lógico solo son capaces de entender valores de 0 o 1, lo que implica que tanto las instrucciones como los datos tendrán que ser codificados como una secuencia de los mismos conocido como **código máquina**. De este forma, la secuencia "0100010101011010" podría significar una instrucción para una máquina concreta.

Como podemos observar, este código resulta ininteligible para el ser humano por lo que la comunicación con el ordenador resulta inviable. Con el objetivo de solucionar este problema surgen los **lenguajes de programación** más cercanos al programador y que disponen una serie de traductores que se encargan de transformar ese programa en código entendible por la máquina.

Los primeros lenguajes que sustituyeron a los lenguajes máquina fueron los de bajo nivel como es el **lenguaje ensamblador**, muy eficiente pero muy cercano a la máquina en la que se va a ejecutar el programa. El crecimiento de la complejidad de los productos software hizo que estos lenguajes fueran sustituidos por **lenguajes de alto nivel** más cercanos al lenguaje natural que todos conocemos.

La evolución de los computadores ha posibilitado la automatización y optimización de una gran parte de las tareas que llevamos a cabo en nuestra vida diaria, tanto a nivel doméstico como empresarial, lo que se traduce en software más complejo y por tanto lenguajes de programación que les den soporte. Es por ello, que los **lenguajes de alto nivel** han sufrido una marcada evolución dando lugar a distintos paradigmas de programación que trataremos a lo largo del tema como elemento distintivo para establecer los tipos.

# 2. Lenguajes de Programación

Los lenguajes de programación son una serie de notaciones precisas para escribir programas que vienen definidos por un conjunto de símbolos a los que denominamos **alfabetos,** unas reglas para su combinación que dan lugar a la **sintaxis** del lenguaje y una **semántica** asociada a cada una de las construcciones sintácticas que indicarán una o varias de las tareas que el computador debe realizar.

Existen multitud de lenguajes de programación que utilizan modelos de computación muy diversos aunque todos ellos presentan algunos elementos semánticos comunes:

- Los programas se escriben a partir de sentencias compuestas por 2 partes; la **instrucción** que indicará de forma directa o indirecta el tipo de tarea que ejecutará el computador y los operandos que indican donde hallar o almacenar los datos o siguientes instrucciones que se van a ejecutar.
- Cada lenguaje definen un conjunto de datos sobre los que pueden trabajar así como su representación interna y tipificación.
- Se dispone de una serie de operadores que indican la tarea a realizar sobre los datos. Se pueden clasificar en; lógicos (AND, OR, NOT,...), aritméticos (+,-,*,/) y relacionales (<=,>=, ==,…).
- Permiten la escritura de comentarios, muy útiles para el desarrollador pero que serán ignorados por la máquina en el momento de su ejecución.
- Delimitadores; nos ayudan a definir la estructura básica del programa actuando como separadores de las distintas construcciones sintácticas, (".",";"," ",

# 3. Tipos y Características

Aunque no existe estipulada ninguna clasificación estricta de los mismos, trataremos de esquematizar los diferentes tipos de lenguajes de programación atendiendo a su nivel de abstracción, implementación, paradigma de programación y tipificación, lo que nos permitirá tener una visión crítica de las diferentes características que aportan cada uno de ellos con el fin de seleccionar el más adecuado para los proyectos que queramos desarrollar.

## 3.1 Abstracción

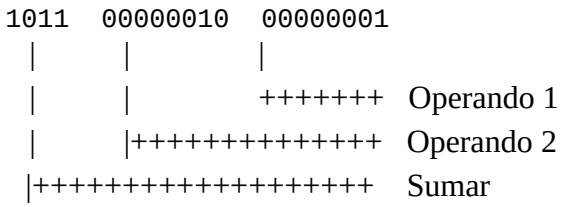
Podemos llevar a cabo una primera clasificación atendiendo a la cercanía del lenguaje de la máquina en la que se ejecuta, de este modo tenemos los lenguajes de bajo nivel más cercanos al lenguaje máquina que entienden los computadores y los de alto nivel, mas cercanos al lenguaje natural.

## 3.1.1 Lenguajes de bajo nivel

Este tipo de lenguajes tienen una fuerte dependencia de la máquina en la que se ejecutan llegando el programador a tener que reescribir el código en caso de que queramos ejecutarlo en distintos procesadores.

### *3.1.1.1 Lenguaje máquina*

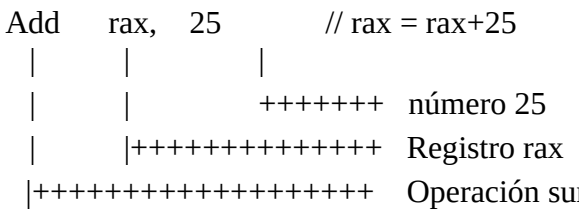
Se trata del lenguaje de más bajo nivel directamente entendible por el procesador. Todos los computadores definen una serie de instrucciones o tareas que pueden llevar a cabo y les asignan un código que vendrá dado por una combinación de ceros y unos. De esta forma, establecen que los primeros n bits corresponderán al código de operación que se quiere llevar a cabo y los n bits siguientes a los datos.




La implementación de un programa en este lenguaje, ininteligible para el ser humano, la detección de errores o adición de funcionalidades resulta inviable.

### *3.1.1.2 Lenguaje ensamblador*

Para facilitar la labor de los programadores, a principios de la década de 1950 se desarrollan una serie de etiquetas nemotécnicas que identificarán cada uno de los código de las operaciones que el procesador es capaz de realizar y direcciones simbólicas para identificar las posiciones de memoria donde almacenar y recuperar los datos e instrucciones, de esta forma se sustituyen los código numéricos por símbolos alfabéticos memorizables por el ser humano. Las computadoras siguen entendiendo solamente el lenguaje máquina por lo que se sirven de los programas ensambladores para llevar a cabo la traducción de las instrucciones a sus equivalentes en código máquina. Todas los procesadores actuales tienen códigos nemotécnicos para identificar sus operaciones aunque, cada fabricante define los suyos propios.




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

### 3.1.2 Lenguajes de alto nivel

Se trata de una serie de lenguajes de programación cercanos al lenguaje natural, normalmente el inglés y por tanto más comprensibles y fáciles de usar. A diferencia de los anteriores una sola instrucción puede ser traducida como múltiples instrucciones en lenguaje máquina. Estos lenguajes dan respuesta a una evolución del hardware y la demanda de software más complejo y robusto.

Los programas construidos son independientes de la máquina en la que se van a ejecutar y por tanto no requieren de ningún conocimiento del tipo de instrucciones que proveen. En lugar de trabajar con registros, direcciones de memoria o códigos de operación se trabaja con variables, , expresiones booleanas, funciones, hilos, bucles… Un programa intérprete o compilador se encargará de la traducción de cada una de estas estructuras sus correspondientes instrucciones en lenguaje máquina.

```
Scanner scanner = new Scanner(System.in);
int myAge = scanner.nextInt() ; //pedimos edad del usuario
 myAge++;                       //sumamos un año
System .out.println(myAge);     //mostramos por pantalla
```

*Ejemplo 3. instrucciones en java*

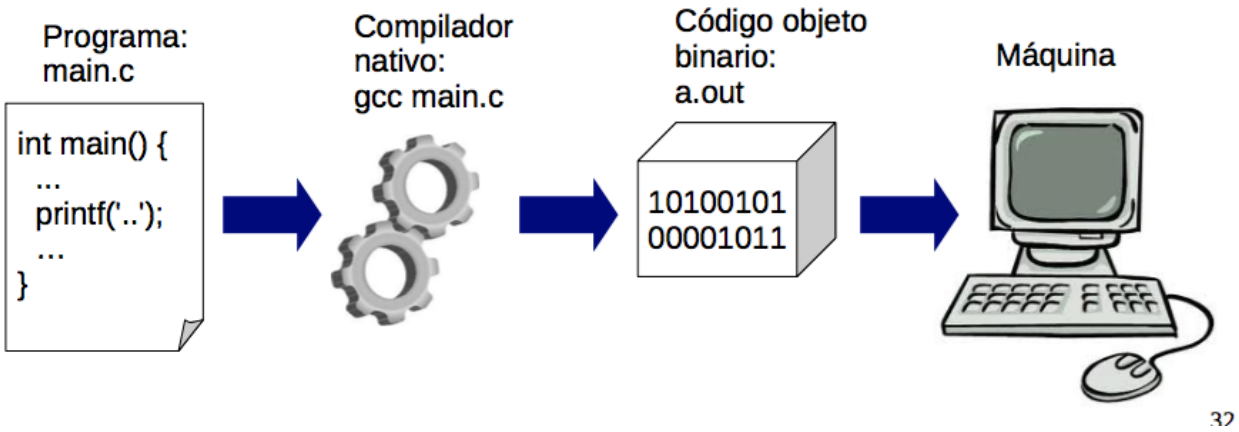
## 3.2 Implementación

Se trata de una clasificación en función del tipo de programa que utilizan para traducir las instrucciones de alto nivel a lenguaje máquina. Existen tres opciones; **compilados, interpretados y mixtos**

### 3.2.1 Compilados

Los lenguajes compilados se caracterizan porqué el código fuente ha de ser traducido a lenguaje máquina de forma previa a su ejecución para ello el programa compilador comprobará que no existan errores léxicos, sintácticos y/o semánticos en el código fuente del programa y generará un ejecutable para la plataforma objeto de compilación.

Lenguajes como Swift, Haskell, C, C++, Objective-C, Go son ejemplos de este lenguaje



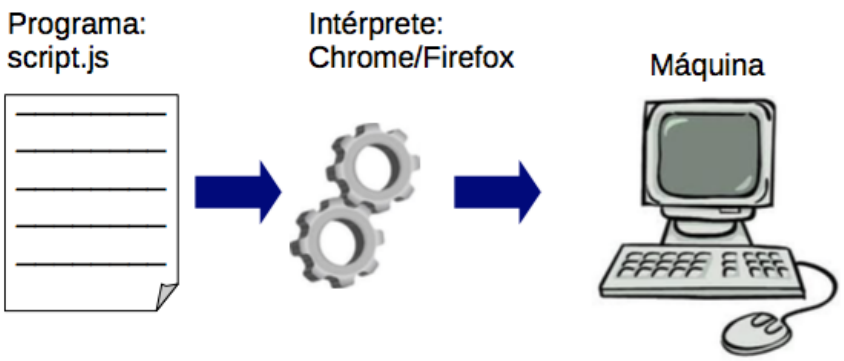

En general, **un lenguaje compilado está optimizado para el momento de la ejecución.**

### 3.2.2 Interpretados

En este caso el código fuente es cargado en la computadora en la que se quiere ejecutar y un programa intérprete se encargará de ir traduciendo las instrucciones una a una al lenguaje máquina a medida que se van necesitando.

La velocidad en tiempo de ejecución respecto a los programas compiladas se ve afectada ya que tenemos que incluir el paso de traducción. No obstante, la velocidad en tiempo de desarrollo se ve beneficiada ya que no tenemos que estar compilando el código fuente cada vez que queremos ejecutarlo.

Lenguajes como php han desarrollado cacheé para los programas en producción , como es "opcache" de forma que cuando al ejecutar una instrucción se guarda un bytecode en memoria compartida para que en cada ejecución del programa no se tenga que llevar a cabo la fase de interpretación equiparando el rendimiento a los programas compilados.

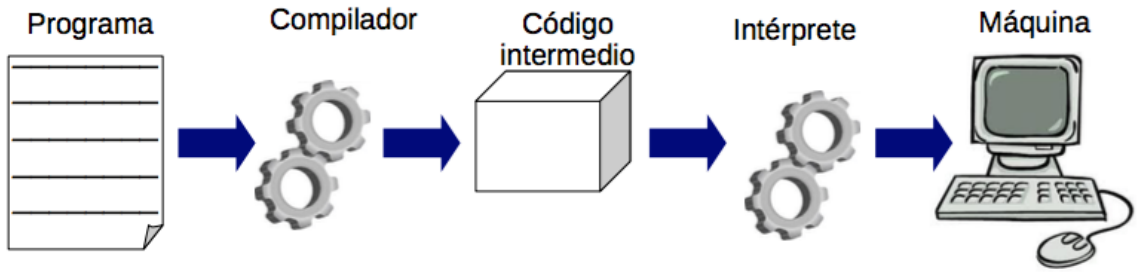



Ejemplos de lenguajes interpretados son; Php, Python, Ruby, Javascript

### 3.2.3 Mixtos

Con el objetivo de escribir programas que una vez compilados se pudieran ejecutar en cualquier sistema operativo y procesador, surgen una serie de lenguajes que mantienen características de los 2 anteriores, para ello, en una primera fase el código fuente es compilado a un lenguaje intermedio o "bytecode" que después es interpretado.

El lenguaje más representativo es Java. En una primera fase, a través del JDK podemos implementar el programa y compilarlo a bytecode, posteriormente necesitamos instalar el JRE que contiene la máquina virtual con el intérprete en la plataforma donde queramos ejecutar el programa.




Otro lenguajes que siguen este modelo es Elixir.

## 3.3 Paradigma de programación

A medida que el software ocupa una posición más importante en todos los ámbitos de la sociedad; medicina, negocios, educación, trámites administraciones, el software se vuelve más complejo y las necesidades de calidad aumentan. De este modo, surgen una serie de enfoques para la construcción de programas conocidos como **paradigmas de programación** y que definen un conjunto de reglas, patrones y estilos de codificación.

Algunos de los lenguajes actuales dan soporte a múltiples paradigmas, no obstante, la adopción de uno u otro dependerá del producto que queramos construir.

### 2.3.1 imperativos

Los lenguaje imperativos basan la construcción del programa en una serie de sentencias que establecen explícitamente como la computadora debe manipular los datos presentes en memoria y/o como se debe recoger y/o enviar información a los dispositivos de E/S. De esta forma indicamos paso a paso como el computador debe realizar la tareas. Para ello proporcionan una serie comandos u órdenes.

Dentro de esta categoría se engloba los lenguajes estructurados y los orientados a objetos (esta última puede verse como una extensión de los lenguajes imperativos).

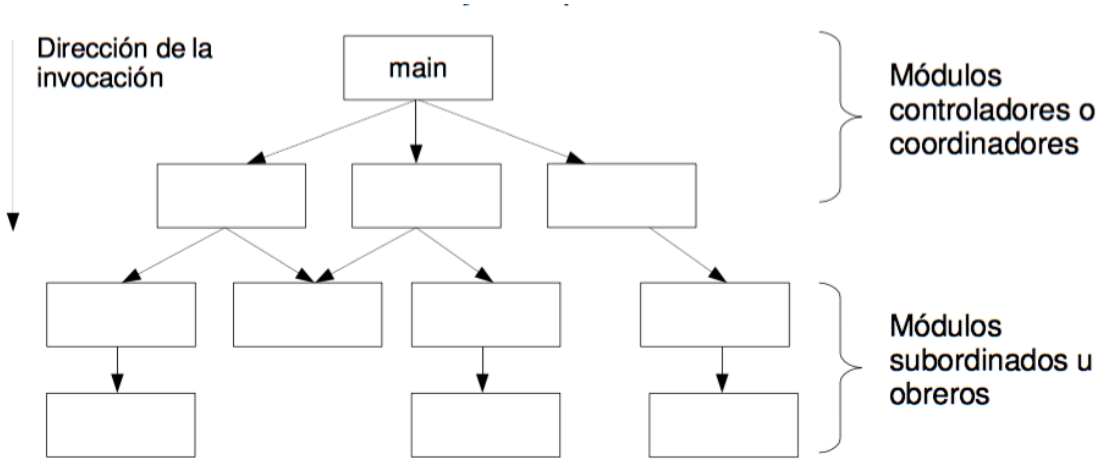
### *3.3.1.1 Estructurados*

La programación estructurada surge para aumentar la claridad del código desarrollado permitiendo que pueda ser leído secuencialmente. Para ello, elimina el uso de saltos GOTO y se basan en el empleo de estructuras de control de flujo jerárquicas con un solo punto de entrada y uno de salida. Estas construcciones lógicas son la *secuencia, selección condicional e iteración*.

El problema de estos programas es que todo el código se concentra en un único bloque por lo que si el programa es demasiado grande resulta difícil su manejo. Actualmente cuando nos referimos a este tipo de lenguajes incluimos los paradigmas de **programación procedimental y modular.**

La programación **procedimental** basa su funcionamiento en la declaración y llamada rutinas, se trata de una agrupación de instrucciones que se identifican por una cabecera o firma formada por un nombre identificativo de la tarea que realizan y los argumentos de entrada que servirán para enviar información necesaria para la ejecución de la rutina.

La **programación modular**, es una evolución de la programación estructurada que permite la división del programa en partes más manejables denominadas módulos o librerías independientes, facilitando la reutilización de código, el trabajo simultáneo de desarrolladores y el mantenimiento ya que trabajamos con pequeños módulos que permiten localizar el problema más rápidamente.



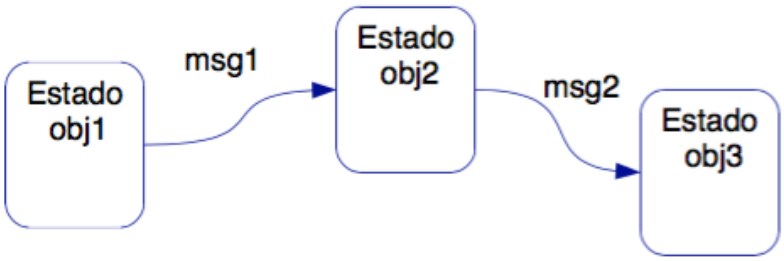

Los lenguajes más representativos de este paradigma son C, Pascal, Fortran y Modula-2. No obstante actualmente la mayoría de lenguajes de programación imperativos como Python o Php soportan este tipo de programación.

```
#include <stdio.h>

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

*Calculo del número de fibonnaci en C*

### *3.3.1.3 Orientados a objetos*




Se trata de un extensión del modelo imperativo, En la programación orientada a objetos un programa puede verse como un conjunto de objetos que interaccionan entre sí para llevar a cabo tareas. Cada uno de estos objetos consta de una estructura de datos que definen su estado, un comportamiento que constituye las operaciones que se pueden llevar a cabo sobre él y una identidad.

De este modo se proporciona un modelo de programación mas cercano a la realidad, ya que el mundo se compone de cosas, es decir objetos; una silla, un coche…

Como lenguajes representativos de este paradigma podemos citar: Java, C++, Objetive- C, Ada, SmallTalk… Sin embargo, lenguajes históricamente más procedimentales como php en sus versiones más recientes proporcionan estructuras para este paradigma.

```
<?php
 //Definición de la clase
public class Dog {

   //estado de la clase a partir de variables
   private $name;
   private $age;
```

```
  private $breed;
  private $weight;

  function __construct($name, $age, $breed){
     this.name = $name;
     this.age = $age;
     this.breed = $bredd;
  }

   //comportamiento a través de métodos
   public function eat(): void{
        $weight++;
   }
}
```

Ejemplo definición clase en php7

### 3.3.3 Declarativos

Son lenguajes de programación de alto nivel, orientados a buscar la solución del problema sin preocuparse por la forma de llegar a ello, para ello se basan en un conjunto de definiciones o ecuaciones que describen lo que debe ser calculado, es decir le indicamos el "qué" queremos dejando el "como" al intérprete de dicho lenguaje. Este tipo de lenguajes son más cercanos al razonamiento matemáticos pero son menos utilizados en la construcción de software comercial.

Dentro de los lenguajes declarativos destacan los lenguajes lógicos y los lenguajes funcionales.

#### *3.3.3.1 Funcionales*

Los programas funcionales se basan en la declaración de funciones, no entendidas como subprogramas clásicos del lenguaje imperativo sino como funciones puramente matemáticas que se le aplican sucesivamente a los datos de entrada para obtener el resultado esperado.

Una de las características propias de estos lenguajes es la no existencia de asignaciones de variables y la falta de construcciones estructuradas como la secuencia o la iteración, lo que obliga en la práctica a que todas las repeticiones de instrucciones se lleven a cabo por medio de funciones recursivas.

El código desarrollado utilizando programación funcional tiende a ser más conciso y expresivo por lo que en los últimos años lenguajes imperativos como java, Python o Javascript, incorporan estructuras y características para utilizar este enfoque en la construcción de programas, surgiendo los denominados lenguajes funcionales híbridos. Estos son menos dogmáticos al permitir conceptos tomados de los lenguajes imperativos, como la secuencia de instrucciones o la asignación de variables.

Respecto a los lenguajes funcionales puros podemos nombrar, por su importancia, Erlang, Haskell o R.

```
Fib :: int → int
fib 0 = 0;
fib 1 = 1
fib n = fib(n-2)+fib(n-1)
```

*Algoritmo de Fibonacci en haskell*

### *3.3.3.2 Lógicos*

La programación lógica se basa en la definición de un conjunto de hechos y reglas lógicas, **predicados,** que definen nuestro conocimiento de forma que el motor o intérprete se encargará de procesar ese conocimiento e inferir conclusiones que den respuesta a nuestras consultas

El lenguaje de programación lógico por excelencia es PROLOG por lo que veremos una serie de ejemplos para tratar de entender su funcionamiento.

Los hechos son verdades incondicionales mientras que las reglas son verdades condicionales asociadas a una lógica, las 2 juntas definirán la base de conocimiento

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

*Base de conocimiento en prolog*

A partir de aquí podemos hacerles consultas al intérprete que a partir de su base de conocimiento inferirá el resultado.

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

Este paradigma es ampliamente utilizado en inteligencia artificial y sistemas expertos como reconocimiento de lenguaje natural,  demostración automática de teoremas o robótica.

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 puede inferir la respuesta.

## 3.4 Sistema de tipos. Tipificación

Se trata de un  aspecto muy importante en la calidad de un lenguaje de programación, ya que representan los tipos de datos soportados y el conjunto de reglas para su manipulación.

La mayoría de lenguajes de programación soportan datos primitivos (enteros, double, char, long) y compuestos (array, string,…), la verdadera diferencia estriba en las reglas para su manipulación. Podemos identificar distintos sistemas de tipos:

- **Estático vs dinámico:** Indica si la comprobación de los tipos se hace en tiempo de compilación (estático) o en tiempo de ejecución (dinámico).
- **Débil vs fuerte:** Cuando el lenguaje ejecuta conversiones implícitas (débil) entre expresiiones de distintos tipos para evitar errores.
- **Implícito vs explícito:** Cuando el tipo de las variables/expresiones se declara de manera explicita o se infiere automáticamente.

De esta forma podemos decir que java es un lenguaje con sistema de tipos estático, fuerte y explicito, php o javascript dinámico, débil e implícito y haskell estático, débil e implícito

# 4. Conclusión

Existen una gran cantidad de lenguajes de programación, conocerlos todos resulta inviable y si nos centramos en uno solo para desarrollar todo nuestros proyectos tenemos un alto riesgo de obsolescencia que nos llevará a la creación de productos software de baja calidad.

Si aprendemos programación a través de conceptos y conocemos los diferentes tipos de lenguajes de programación y sus características, seremos capaces de, dado un problema que se nos presente elegir el mejor paradigma de programación y lenguaje para la construcción de la solución, solo deberemos aprender la sintaxis del lenguaje elegido.

# 5. Bibliografía y webgrafía

Kubar. Arvind (2014). Introduction to programming Languages". Ohio. CRC Press

Bird, R. (2000). Introducción a la programación funcional con haskell. Prentice Hall

Kernighan, B (1991). El Lenguaje de programación C. 2da Edición. Prentice Hall

Ramos A y Ramos M.J. (2014). Entornos de Desarrollo. Madrid. Garceta

Herbert (2014). Java 8. Anaya

2018 Wikipedia – Lenguajes programación - https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n