UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

# Un estudio comparativo de los lenguajes de programación de la última década.

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Daras de la Fuente, Claudia

Tutor/a: Ramírez Quintana, María José

Cotutor/a: Lucas Alba, Salvador

CURSO ACADÉMICO: 2021/2022

# Agradecimientos

En primer lugar, me gustaría agradecer a mis tutores los sabios consejos que me ha ido dando durante todos estos meses de la realización del proyecto, también mencionar su gran interés y dedicación porque yo aprendiera lo máximo posible. Sin su experiencia y profesionalidad no hubiera sido posible realizar este trabajo.

En segundo lugar, agradezco a mis padres el apoyo que me han dado durante todo este tiempo, y que, gracias a los valores, como la responsabilidad y la constancia, que me han enseñado ha sido posible superar esta etapa de mi vida con el mayor éxito posible. Además, agradecerle a mi pareja y compañero de vida, César, su apoyo incondicional y su ánimo de cada día para poder ir superándome.

Por último, hacer una mención especial a los compañeros que me han acompañado en estos cinco años de doble grado, los cuales me han enseñado muchas cosas, en especial a Irene y Andrea por su apoyo y por los maravillosos momentos que hemos pasado juntas.

# Resumen castellano

El sector de la programación está en una continua actualización e innovación, ya sea por nuevas tendencias que surgen o por nuevas tecnologías emergentes. Pero para que la programación funcione se necesitan lenguajes lo más eficientes y productivos posibles. En este trabajo se va a hacer un recorrido por la historia de los lenguajes de programación, para comprender como se originaron y el porqué de la aparición de mucho otros. Seguido de esto, se hará una breve descripción de conceptos y clasificaciones sobre los lenguajes actualmente existentes y de los ámbitos donde más se están utilizando los lenguajes.

Se eligen diecisiete lenguajes de programación surgidos en la última década para poder ver si hay diferencias significativas con sus predecesores, o simplemente son mejoras sobre estos. Se muestran sus principales característica y puntos diferenciadores sobre los lenguajes en los que se basan, para que así el lector también sea capaz de juzgar si hay realmente un cambio revolucionario o no.

Finalmente se harán unas conclusiones sobre los datos analizados y se expondrán las limitaciones que han ido surgiendo a lo largo de este Trabajo de Fin de Grado.

# Resumen inglés

The programming sector is constantly updating and innovating, whether due to new trends or emerging technologies. But for programming to work, languages need to be as efficient and productive as possible. In this paper we will take a look at the history of programming languages, to understand how they originated and why many others have appeared. This will be followed by a brief description of concepts and classifications of the currently existing languages and the areas where the languages are most widely used.

Seventeen programming languages that have emerged in the last decade are chosen in order to see whether there are significant differences with their predecessors, or simply improvements on them. Their main features and points of differentiation from the languages on which they are based are shown, so that the reader will also be able to judge whether there really is a revolutionary change or not.

Finally, conclusions will be drawn about the data analysed and the limitations that have arisen during the course of this Final Degree Project will be explained.

# Resumen valenciano

El sector de la programació està en una contínua actualització i innovació, ja siga per noves tendències que sorgixen o per noves tecnologies emergents. Però perquè la programació funcione es necessiten llenguatges el més eficients i productius possibles. En este treball es va a fer un recorregut per la història dels llenguatges de programació, per a comprendre com es van originar i el perquè de l'aparició de molt altres. Seguit d'açò, es farà una breu descripció de conceptes i classificacions sobre els llenguatges actualment existents i dels àmbits on més s'estan utilitzant els llenguatges.

Es trien dèsset llenguatges de programació sorgits en l'última dècada per a poder veure si hi ha diferències significatives amb els seus predecessors, o simplement són millores sobre estos. Es mostren els seus principals característica i punts diferenciadors sobre els llenguatges en què es basen, perquè així el lector també siga capaç de jutjar si hi ha realment un canvi revolucionari o no.

Finalment es faran unes conclusions sobre les dades analitzats i s'exposaran les limitacions que han anat sorgint al llarg d'este Treball de Fi de Grau.

# Índice de contenido

# Índice de Ilustraciones

# Índice de tablas

# CAPÍTULO 1: Introducción

## 1.1 Introducción

Todas las personas para poder recibir o transmitir información necesitamos de un lenguaje que lo permita, y en términos generales, para poder realizar dicha comunicación utilizamos la unión de símbolos y letras para poder crear dicho lenguaje. Por lo que podemos definir un lenguaje como un sistema de signos y reglas para combinarlos (dando lugar a estructuras sintácticamente correctas) mediante el cual las personas nos comunicamos entre nosotros. Estos signos pueden ser de diferentes tipos, corporales, sonoros o incluso gráficos.

La informática necesita también la existencia del uso de los lenguajes, debido a que de esta manera podremos especificar las diferentes acciones que deseamos realizar sobre la computadora. Estos lenguajes de programación están destinados a diversos ámbitos, atendiendo a las características que presentan para poder facilitar algunas tareas y dificultar otras. Por lo que no hay un único tipo de lenguajes, sino que se clasifican según sus características, el paradigma, algunos conceptos que soportan y otros muchos criterios. Los que se detallan en este Trabajo de Fin de Grado son solo algunos de los más conocidos.

Normalmente, la finalidad de los lenguajes de programación es la resolución de problemas. Para ello, se escriben programas que constan de una colección de instrucciones construidas de acuerdo con las reglas sintácticas, y que pueden a su vez organizarse en módulos. Estas instrucciones son tratadas por un compilador, el cual las traduce para que el computador pueda interpretarlas correctamente. Finalmente, el computador ejecuta estas instrucciones traducidas para dar lugar a una salida que haya especificado el programador. Todos estos pasos son necesarios debido a que un ordenador solo está capacitado para realizar las cuatro operaciones más básicas (sumar, restar, multiplicar y dividir), recuperar y almacenar documentación y, por último, hacer comparación de dos valores alfabéticos o numéricos. Con estas operaciones el ordenador es capaz de llegar a resolver problemas mucho más complejos.

Por todo lo anteriormente descrito, podemos decir que los lenguajes de programación nos permiten expresar en operaciones más básicas la solución de un problema sencillo o complejo.

## 1.2 Motivación

Hoy en día existen infinidad de lenguajes de programación, lo cual presenta ventajas y desventajas al mismo tiempo. La principal desventaja de esta masiva creación de lenguajes es la dificultad de escoger el más adecuado para el proyecto que se quiera llevar a cabo, debido a que, hay lenguajes que presentan características muy similares. Por otro lado, una ventaja es

que si pruebas con un lenguaje y ves que no acaba de encajar, hay otros muchos que podrían encajar mejor, por esto mismo podríamos decir que es un arma de doble filo.

Constantemente se están desarrollando nuevos lenguajes de programación, ya que, al parecer cada compañía tecnológica desea tener su propio lenguaje, y que sea mejor que el de la competencia. Por ejemplo, Go el lenguaje de programación desarrollado en 2009 por Google, Hack el lenguaje de Facebook que lanzó en 2014 y Swift de Apple que salió al mercado también en 2014.

Por esta creación masiva de lenguajes es importante hacer un análisis sobre estos que están apareciendo para poder saber si están añadiendo nuevas características e innovando o, si por el contrario no están añadiendo ninguna innovación y simplemente son una mezcla de características de otros lenguajes ya existentes.

Sí que hay que destacar que la principal motivación de esta creación de lenguajes es debida principalmente a las necesidades de crear aplicaciones que cada vez tienen que ser más compatibles con un sin fin de sistemas operativos y diferentes funcionalidades. Los usuarios cada vez son más exigentes con los productos informáticos y eso obliga a los desarrolladores a adaptarse a estas necesidades. Por esto mismo, los programadores necesitan que cada vez sea más sencillo aprender nuevos lenguajes, para poder agilizar el aprendizaje y ser productivos lo antes posible. Vivimos en un mundo donde todo es de 'ya para ya', y en el mundo de la programación no iba a ser menos. Si estos nuevos lenguajes que se están creando simplemente agrupan características de otros ya existentes permiten que, con saber el predecesor, de un simple vistazo y mediante simples deducciones se pueda llegar a entender el nuevo lenguaje. Esto es algo así como cuando un español intenta leer en italiano, que al ser dos idiomas parecidos el lector puede ir intuyendo el significado de algunas de las palabras.

También ha sido objeto de motivación de este Trabajo de Fin de Grado la entrevista que realizó Stuart Feldman a Alan Kay. En esta entrevista Alan Key hizo algunas suposiciones sobre cómo sería el futuro de los lenguajes de programación. Mencionó que en torno al 1960 la gente ya lanzaba multitud de lenguajes de programación siendo que las máquinas aún eran pequeñas, lentas y había malas herramientas, por lo que se planteaba que en el futuro cuando mejoraran estas máquinas ¿qué pasaría entonces? (Feldman, 1994)

En la entrevista (Feldman, 1994) dijo que cada diez años aproximadamente se creaba un lenguaje principal novedoso y mientras tanto surgían lo que él llamó lenguajes híbridos, los cuales eran una simple mejora de otros ya existentes. Pero empezó a darse cuenta de que por culpa de las elecciones que estaban tomando los usuarios de los lenguajes de programación, la tendencia que había observado de creación de lenguajes principales cada 10 años iba a desaparecer. Él afirmó que la adopción de la mayoría de los lenguajes de programación ha sido y sería de manera accidental, porque se ha puesto más el ojo en lo fácil que es implementar un lenguaje, que en las características y funcionalidades que presentan, cosa que ha hecho que se utilicen más algunos lenguajes cuando había otros mucho mejores. Esto también motivó a Kay a pensar que no se crearían más lenguajes con diferencias significativas, sino que simplemente se llevarían a cabo mejoras sobre los ya existentes (Feldman, 1994).

En este Trabajo de Fin de Grado nos centraremos en intentar responder si lo que ya predijo Alan Kay de que no habría ningún nuevo lenguaje novedoso se ha ido cumpliendo o al contrario.

# 1.2 Objetivos

El objetivo principal de este Trabajo de Fin de Grado es llegar a analizar las características que incorporan los lenguajes de programación que han ido apareciendo durante la última década, poniendo el foco en si han incorporado cambios revolucionarios, o si simplemente consisten en una mezcla de características ya existentes (siguiendo las hipótesis y opiniones esgrimidas por Alan Key). Para llevar a cabo este análisis nos centraremos en 19 lenguajes de programación.

### 1.2.1 Objetivos secundarios

Para poder alcanzar el objetivo principal que hemos definido tenemos que establecer los siguientes objetivos secundarios:

- Realizar un análisis de la historia de la programación. Visualizar desde dónde hemos partido en este mundo de los lenguajes de programación.
- Analizar algunas de las características más conocidas de los lenguajes de programación que permiten categorizar los lenguajes.
- Definir las características principales de los lenguajes de programación más significativos de la última década.

Con el conjunto de estos tres objetivos y el principal podremos llevar a cabo este Trabajo de Fin de Grado.

# 1.3 Relación con las asignaturas

La principal finalidad de este Trabajo de Fin de Grado radica en poder llegar a demostrar cuáles son las habilidades y/o conocimientos que se han ido adquiriendo a lo largo de los cursos del grado en Ingeniería Informática impartidos por la Universidad politécnica de Valencia. En los siguientes puntos se comentan algunas de las asignaturas que más han influido en la realización de este trabajo.

- Fundamentos de Computadores, la cual ha ayudado a entender el funcionamiento de un procesador haciendo hincapié en la comprensión del lenguaje máquina que necesitan los computadores.
- Introducción a la informática y a la Programación, que nos introduce los elementos básicos de la informática y la programación con la ayuda de ejemplos en un lenguaje orientado a objetos como es Java.

- Programación, continuación en la formación en el desarrollo de programas. Ayuda a analizar, diseñar y validar algoritmos para resolución de problemas.
- Lenguajes, tecnologías y paradigmas de la programación, donde se han adquirido conocimientos de los diversos paradigmas de programación. También se ha enseñado qué implicaciones se derivan del diseño y uso de diferentes tipos de lenguaje para poder facilitar el proceso de aprendizaje de los lenguajes que van surgiendo.

## 1.4 Estructura

Este Trabajo de Fin de Grado se va a dividir en cuatro capítulos. En los siguientes puntos se hará una pequeña descripción de cada uno de estos, menos del primer capítulo que es este mismo.

En el capítulo 2 nos centraremos en los lenguajes de programación y algunas de sus clasificaciones. Es la parte más conceptual de este trabajo, que ayudará a entender mejor las definiciones que se van a exponer de los lenguajes de la última década. El primer apartado de este capítulo es el contexto de la historia de los lenguajes de programación. Continuaremos con una descripción de los niveles que presenta un lenguaje, los tipos de programación más conocidos actualmente, y finalmente se comentarán los diferentes tipos de modelos de memoria que existen.

En el capítulo 3 ya nos adentraremos más en el caso de estudio de este trabajo. Analizaremos los 19 lenguajes que hemos considerado más importantes de la última década, haciendo un resumen de sus principales características.

En el capítulo 4 tenemos un anexo donde aparece la explicación de algunos conceptos que se comentan a lo largo del trabajo que son más específicos y necesitan de su introducción.

# CAPÍTULO 2: Los lenguajes de programación

En este capítulo se van a analizar los conceptos más relevantes relacionados con los lenguajes de programación, para así cuando nos adentremos en el análisis de los lenguajes que aparecen en el Capítulo 4, sea más fácil entender los conceptos.

Comenzaremos por comentar la historia de los lenguajes de programación, cuales fueron los primeros y los motivos de sus apariciones, para comprender cuál es la motivación por seguir desarrollando nuevos lenguajes de programación. Seguidamente, se mostrarán las clasificaciones más significativas en las que se pueden categorizar los lenguajes de programación, además de hacer un inciso sobre lo que es un sistema de tipos y que significa el chequeo de tipificado, ya que en el ámbito de los lenguajes de programación son conceptos muy relevantes que es necesario conocer. Por último, se hará una explicación de cada uno de los tipos de programación más utilizados en la actualidad, lo que ayudará a aportar una visión más general de la situación actual en la que se encuentra el sector de la programación.

## 2.1 Historia de los lenguajes de programación

Aunque no nos demos cuenta, hoy en día no podríamos vivir sin los llamados lenguajes de programación. Vivimos rodeados de objetos construidos mediante códigos como, por ejemplo, las redes sociales, las aplicaciones de nuestros dispositivos móviles, los videojuegos, y los propios dispositivos con los que ejecutamos cualquier aplicación o programa.

En el comienzo de la programación únicamente existía un lenguaje de programación, que era el propio código binario, que como explicaremos más adelante en este trabajo también se denominaba código o lenguaje máquina. Este tipo de programación resultaba muy costosa y lenta, debido a que todas y cada una de las instrucciones que se querían escribir tenían que estar en código binario, con lo que el desarrollador se veía obligado a tener que conocer dónde se almacenaban los datos que se iban a utilizar en la memoria. Por todo esto, a mediados de los 50 se comenzó a crear una escritura más simbólica, que recibió el nombre de código ensamblador (*Assembly*). Esta simplificación hacía uso de abreviaturas mnemotécnicas para poder facilitar operaciones entre las cuales estaban la suma (*ADD*) y el copiado (*STORE).* Aun existiendo este código, la traducción al lenguaje máquina seguía teniendo que hacerse de manera manual, pero no se tardó mucho en crear un programa que se encargará de esta traducción, al que llamaron ensamblador (*Assembler*) (Trigo Aranda, 2004).

Con la llegada de los ordenadores a cada vez más empresas y entornos educativos, surgió la necesidad de sustituir a esos lenguajes más tediosos por otros que facilitaran mucho más el trabajo de los desarrolladores. Aparecieron lo que se denominó lenguajes de alto nivel, los cuales presentaban una estructura que se asemejaba mucho más a cómo pensamos los humanos, alejándose de cómo trabaja un ordenador. Entre estos lenguajes se encuentran BASIC, C y PASCAL (Trigo Aranda, 2004).

Antes de adentrarnos en conocer cuáles fueron los primeros lenguajes de alto nivel cabe hacer una diferenciación entre lo que es un intérprete y un compilador. Como ya se hacía con el código ensamblador se necesita de un traductor a código máquina para que el computador pueda ejecutar las órdenes que nosotros queremos. Para poder entender bien la diferencia entre un intérprete y un compilador vamos a recurrir a explicarlos con los lenguajes humanos.

Cuando una persona se va de viaje a otro país, pongamos Reino Unido, y no sabe nada de inglés recurre a una persona que le vaya traduciendo del castellano al inglés y a la inversa. Este traductor irá traduciendo frase a frase, lo que es lo mismo que hace un intérprete del código, el cual va traduciendo instrucción a instrucción, lo que suele favorecer la depuración y la interactividad. Ejemplos de lenguajes interpretados son LOGO y BASIC.

El otro tipo de traducción sería por ejemplo cuando un libro que se publica por un escritor español empieza a tener fama en otro país con otro idioma. No tendría ningún sentido que existiera un traductor que fuera traduciendo frase a frase el libro al lector. Por esto último existen las traducciones de los libros. Este sería el ejemplo de lo que hace un compilador, el cual traduce todo el programa a la vez, lo que lo deja listo para poder ser ejecutado. Esta es la forma en la que se consigue una mayor fluidez y rapidez en el tiempo de ejecución. Algunos lenguajes compilados son PASCAL, FORTRAN y COBOL.

Una vez aclarada esta diferencia pasamos a hacer un recorrido por los primeros lenguajes de alto nivel. A principios de los años 50 John Backus, que trabajaba con uno de los primeros ordenadores que sacó IBM al mercado, desarrolló el programa *SPEEDCODING,* y tomando a este programa como base empezó con la creación de un nuevo lenguaje para poder añadir al modelo de ordenador IBM 704 que iba a salir pronto al mercado. De esta forma, en 1956 se desarrolló FORTRAN[1]. Este lenguaje estaba pensado para poder resolver problemas técnicos y científicos, y para los que estaban familiarizados con la notación científica afirmaban que era bastante sencillo de aprender (Trigo Aranda, 2004).

A finales de los años 50, aparece una preocupación por la poca facilidad de portar un programa de un ordenador a otro, y también por la dificultad que había de leer códigos y modificarlos en caso de nuevas condiciones. Desde el Departamento de Defensa de los Estados Unidos, se organizó una conferencia sobre los lenguajes (*Conference on Data Systems Languages*). En esta conferencia participaron grandes empresas del momento como Honey Well e IBM y, además, la denominada gran dama de la informática Grace Hooper formaba parte del comité. De esa conferencia salieron las especificaciones de desarrollo del nuevo lenguaje COBOL[2]. A diferencia de FORTRAN este lenguaje se orientó más para tareas administrativas, aumentar la portabilidad de los programas y la legibilidad de los mismos, es decir, solventar las mayores preocupaciones que surgieron en ese entonces (Trigo Aranda, 2004).

[1] FORTRAN: FORmula TRANslator

[2] COBOL: COmmon Business Oriented Language

Durante 1964, Thomas E. Kurtz y John G. Kemeny profesores del Dartmouth College desarrollaron un lenguaje que ayudará a los estudiantes a poder introducirse en los sistemas que denominaban como de tiempo compartido. A este nuevo lenguaje lo denominaron BASIC debido a lo sencillo que era comparado con los ya existentes. Este lenguaje se aplicó tanto para tareas administrativas como para problemas más científicos. El primer ordenador personal diseñado por Microsoft el *Altair* de MITS[3] se desarrolló en BASIC debido a lo sencillo que resultaba aprender este lenguaje y además su intérprete no ocupaba demasiado espacio de memoria. Más adelante, fue Microsoft quien se encargó de adaptar BASIC para utilizarlo en productos de Apple, en el PC de IBM y en los microordenadores. BASIC es un lenguaje que ha sabido ir adaptándose a las necesidades que se han ido encontrando en el mercado, por eso ha pasado en sus primeras versiones de ser interpretado y un poco ilegible a ser un lenguaje bastante estructurado y compilado (Trigo Aranda, 2004).

En 1967 Seymour Papert pensó que los alumnos de menos edad tenían que empezar a saber entender y escribir programas informáticos, por lo que se empezó a diseñar un lenguaje llamado LOGO para esta finalidad. No fue hasta 1980 que se difundió por el mundo gracias al libro "*Mindstorms: Children Computers and Powerful Ideas*". Este libro tuvo un buen recibimiento en el ámbito educativo, más concretamente en primaria y secundaria. Este lenguaje a lo largo de los años ha ido perdiendo gran fama debido a que se basa en la utilización de métodos recursivos y listas, que no son muy sencillas de aprender para niños como otros lenguajes que han ido apareciendo que sí son realmente sencillos para poder explicar cómo se desarrolla un programa (Trigo Aranda, 2004).

En 1969 se crea el sistema operativo UNIX por los investigadores Dennis Ritchie y Kenneth Thompson. Un año más tarde Thompson creó un lenguaje que denominó B, y dos años más tarde su compañero Ritchie basándose en B creó C. Este lenguaje lo desarrollaron para que no dependiera de la arquitectura hardware, por lo que consiguieron crear el lenguaje más portable del mercado de ese momento. Fue durante los 80 cuando Bjarne Stroustrup creó una ampliación del lenguaje C, que se acabó convirtiendo en 1984 en el compilador que hoy conocemos como C++, el cual se enfoca a la programación orientado a objetos (Trigo Aranda, 2004).

Un profesor del Instituto Politécnico Federal de Zurich, Niklaus Wirth empezó con el desarrollo de PASCAL. Este lenguaje permitió que muchas personas se introdujeran más fácilmente al mundo de la programación gracias a la estructuración que presentaba. PASCAL es considerado el lenguaje que facilita el aprendizaje de la programación utilizando programación estructurada, recursividad, punteros, descomposición modular, etc. Este lenguaje se definió en el libro "*PASCAL – User Manual and Report"*. Fue en 1983 cuando se estandarizó PASCAL y al poco tiempo se lanzó al mercado el compilador PASCAL (Trigo Aranda, 2004).

A mediados de 1972 apareció la idea revolucionaria de que se podría utilizar la lógica como lenguaje de programación, por lo que apareció PROLOG[4]. PROLOG es el denominado por

[3] Micro Instrumentation and Telemetry Systems (MITS)

[4] PROLOG: PROgramation LOGique

excelencia lenguaje declarativo. Hasta el momento todos los lenguajes que hemos ido describiendo se basan en instrucciones que indican todos los pasos que tiene que hacer el programa, lo que más adelante llamaremos lenguajes imperativos. Este lenguaje declarativo se caracterizaba por estar basado en descripciones y no en órdenes. Con el lenguaje PROLOG el desarrollador se centraba en darle conocimientos al computador, a la vez que le daba unas reglas, por lo que al hacerle preguntas al computador sobre los conocimientos aportados el computador hacía deducciones lógicas utilizando la información y reglas aportadas. Este lenguaje como es de esperar no se desarrolló con la idea de utilizarlo en ámbitos científicos, se pensó más en la inteligencia artificial y en resolver problemas lógicos (Trigo Aranda, 2004).

En 1975 el Departamento de Defensa consideró que sus inquietudes no se habían estado cubriendo con COBOL, por lo que se formó un equipo de trabajo para poder estudiar los lenguajes que habían ido apareciendo, y así ver si alguno de ellos podía cubrir mejor sus necesidades. Las condiciones que tenía cumplir el lenguaje que buscaban era que se pudieran desarrollar programas estructurados y modulares, que fueran sencillos de depurar y leer, que se pudieran gestionar periféricos y tenía que admitir el trabajar en paralelo. Como no encontraron ningún lenguaje que se adaptara a todo eso decidieron crear uno nuevo basándose en PASCAL, PL/I y ALGOL 68, ya que eran los que más se acercaban a lo buscado. Se convocó un concurso en el que un equipo que encabezaba Jean Ichbiah ganó con su propuesta. En principio se denominó DoD-1, pero finalmente se acabó llamando ADA en honor a Ada Lovelace (Trigo Aranda, 2004).

Por último, en 1990 James Gosling basándose en C y C++ creó lo que hoy en día es uno de los lenguajes más utilizados en Internet, JAVA. Este lenguaje no se creó con el ojo puesto en Internet, sino que realmente quería crear una interfaz más intuitiva para la electrónica de consumo como son las calculadoras, las televisiones y otros dispositivos. Debido a que esta electrónica de consumo no evolucionó como se esperaba se empezó a utilizar JAVA para Internet y se modificó para este uso (Trigo Aranda, 2004).

A partir de esta fecha empezaron a crearse un sinfín de lenguajes de programación. En este trabajo pondremos el foco en los lenguajes que han ido apareciendo durante la última década.

## 2.3 Lenguaje de programación

Un lenguaje de programación es un lenguaje formal, el cual está diseñado para efectuar procesos. Estos procesos son los que normalmente son llevados a cabo por máquinas, como por ejemplo computadoras (Trigo Aranda, 2004).

Este lenguaje se puede utilizar para crear diferentes programas que puedan controlar el comportamiento físico y también lógico de una máquina, como modo de comunicación humana o, lo más habitual, para expresar algoritmos con mayor precisión.

Los lenguajes de programación están formados normalmente por un conjunto de reglas sintácticas, semánticas y por un conjunto de símbolos, que ayudan a definir una estructura y el significado de los elementos y expresiones. La programación consiste en usar el lenguaje para escribir programas que se deben probar, depurar y compilar. Otra forma que tenemos de definir el concepto de 'programación' es el proceso de crear un programa de computadora, mediante la combinación de procedimientos lógicos. Estos procedimientos suelen seguir los siguientes pasos para alcanzar la resolución de un problema (Trigo Aranda, 2004).

1. Para resolver este problema se hace el desarrollo lógico pertinente.
2. Se escribe la lógica del programa empleando el lenguaje de programación que mejor se adapte al problema que se está tratando.
3. Se ensambla o compila el programa hasta llegar a obtener una versión traducida a lenguaje máquina.
4. Se prueba y depura el programa desarrollado.
5. Y, por último, se desarrolla la documentación necesaria sobre el programa realizado.

### 2.3.1 Clasificación por su cercanía o lejanía al nivel de la máquina

Normalmente la primera clasificación para un lenguaje de programación es mirar su nivel de abstracción sobre el procesador, es decir, analizar cuál es el principio por el que se aísla toda la información que no llega a ser relevante a un determinado nivel de conocimiento.

- ***Bajo nivel***

Este lenguaje proporciona poca o incluso ninguna abstracción del procesador de un computador, por lo que es muy fácil su traslado al lenguaje máquina. Lo más habitual es que se utilice para programar controladores (drivers) (Quero Catalinas, 2002).

Comúnmente es el denominado lenguaje ensamblador.

*Ilustración 1. Ejemplo ensamblador.*

```
DOSSEG
.model small
.stack 100h
.data
msgHello DB "Hola mundo!",13,10,"$"
.code
  mov ax,@data
  mov ds,ax

  mov dx,offset msgHello
  mov ah,9
  int 21h

  mov ax,4C00h
  int 21h
END
```

*Fuente: (Quobit, 2016)*

- ***Medio nivel***

Estos están entre los de bajo y alto nivel. Aunque muchas veces suelen ser confundidos con los de alto nivel, siguen manteniendo ciertos manejos del lenguaje de bajo nivel. Suelen ser los más utilizados para la creación de los sistemas operativos, debido a su manejo independiente de la máquina (manejo abstracto) y utilizando el poder y la eficiencia que presentan los de bajo nivel (Quero Catalinas, 2002).

El lenguaje C es uno de los lenguajes que se clasificarían en este tipo. Por ejemplo, este puede tratar las letras como si fueran números. Con este tipo de lenguaje no se puede concatenar cadenas de carácter con operadores como la suma. El programador es el responsable de llamar a las funciones correspondientes. Otras de sus características más destacadas es el uso de 'apuntadores'. Esta herramienta es muy útil para la implementación de algoritmos como tablas hash[5], listas ligadas, algoritmos de búsqueda y ordenamiento (Quero Catalinas, 2002).

En la siguiente imagen podemos ver un ejemplo de programa escrito en lenguaje C, donde se imprime la cadena 'Hola Mundo'.

*Ilustración 2. Ejemplo lenguaje C.*

ejemplo C: Hola Mundo!

```
#include <stdio.h>

int main()
{
  printf("Hola Mundo!\n");
  return 0;
}
```

*Fuente: (Google imagenes, 2021)*

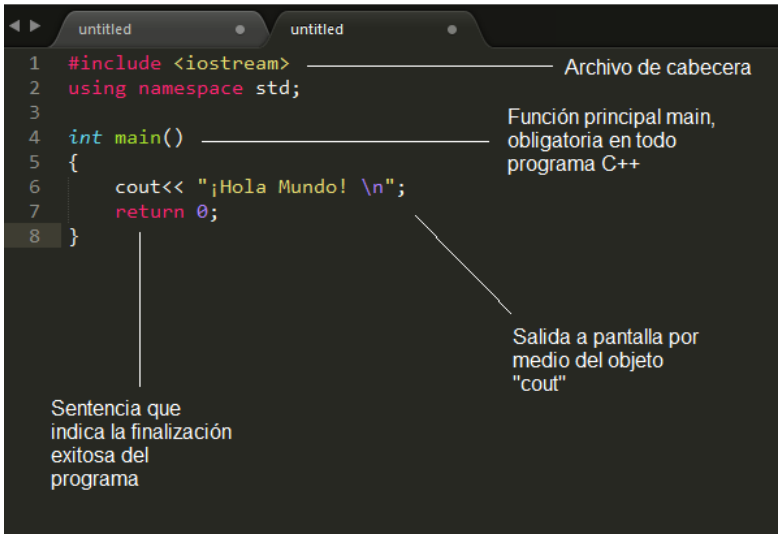
- ***Alto nivel***

Este tipo de lenguaje se caracteriza por expresar los algoritmos de una forma más parecida a la capacidad cognitiva humana, en vez de más cerca de la capacidad de la máquina. En este tipo de lenguajes se necesitan ciertos conocimientos de programación para poder llegar a realizar las instrucciones lógicas. El origen de estos lenguajes es debido a la necesidad de que el usuario pudiera solucionar problemas de procesamiento de los datos de una manera mucho más rápida

[5] Contenedor asociativo (diccionario) que permite almacenar y posteriormente recuperar elementos (valores) de manera eficiente, mediante un objeto dado (clave)

y fácil. Estos aparecieron a finales de 1950. Uno de los lenguajes que pertenecen a esta clasificación es C++ (Quero Catalinas, 2002).

*Ilustración 3. Ejemplo lenguaje C++.*




*Fuente: (APP, s.f.)*

C++ no es el único lenguaje de alto nivel, existen una cantidad inmensa de este tipo de lenguajes con sus diferentes versiones, es por esta razón que resulta difícil a veces su tipificación. Una forma de clasificación muy común es atendiendo al punto de vista de trabajar de los programas y también la filosofía de su creación (La revista informática, s.f.). Otros de los lenguajes de alto nivel más conocidos son Python, Java, JavaScript y PHP.

### 2.3.2 Clasificación por paradigma

- ***Imperativos***: se caracterizan por presentar las instrucciones paso por paso, es decir, describen de manera explícita qué pasos hay que llevar a cabo y cuál es la secuencia exacta para llegar finalmente a la solución buscada. Algunos de los lenguajes más conocidos que presentan alguna de las características de los lenguajes imperativos son (Rivero Espinosa, 2003):
  - Fortran
  - Java
  - Pascal
  - C#

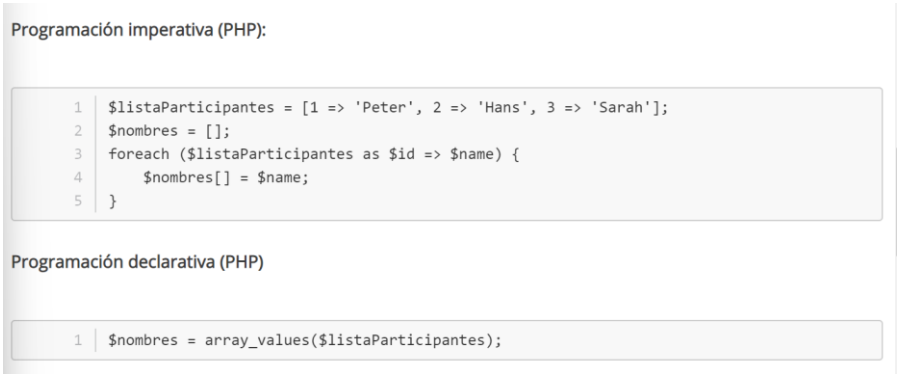
- ***Declarativos***: al contrario que con los imperativos aquí nos encontramos con un lenguaje que únicamente se centra en qué es lo que queremos obtener. Este

lenguaje describe directamente el resultado final a obtener. Algunos de los lenguajes más conocidos son (Rivero Espinosa, 2003):

- Haskell
- Lisp
- Prolog
- SQL
- QML

Para observar mejor la diferencia entre estos dos grupos de lenguajes exponemos un ejemplo culinario. El lenguaje imperativo sería la receta de un plato y por otro lado el declarativo sería únicamente fotos del plato final ya terminado. Otro ejemplo informático es obtener una lista con nombre utilizando programas imperativos y declarativos tal y como se muestra a continuación:

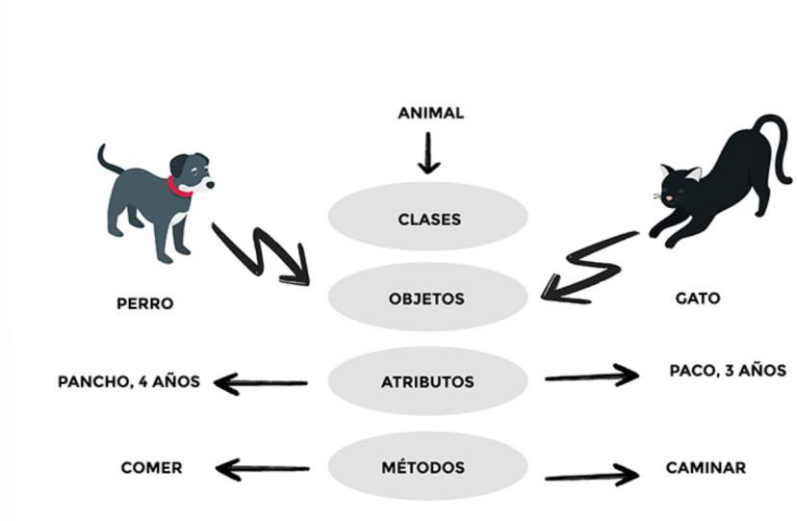
*Ilustración 4. Comparación imperativo con declarativo.*




*Fuente: (IONOS, 2020)*

- ***Orientado a objetos***: este estilo o modelo de programación se basa principalmente en el concepto de 'clases' y 'objetos'. Se utiliza para poder estructurar un programa en simples piezas reutilizables (clases) para poder hacer fácil la creación de instancias de objetos. Esto evita duplicar el código en exceso y así poder crear programas mucho más eficientes. Un ejemplo muy claro para visualizar estos conceptos es con la clase animal, donde los objetos serían diferentes animales (perro y gato) porque comparten características comunes. Cada uno de estos objetos tendrán atributos en común y en su caso atributos específicos, por otro lado, también tendrán métodos compartidos o métodos específicos de cada objeto (Rivero Espinosa, 2003).

*Ilustración 5. Esquema de una clase.*




*Fuente: (web, 2020)*

Algunos de los más famosos son:

- Java
- Ruby
- Scala
- PHP
- JavaScript

- ***Orientados al problema***: la función principal de estos lenguajes es facilitar la especificación de problemas, dejando más de lado los algoritmos de resolución. Un ejemplo de éstos son, por ejemplo, FORTRAN que fue diseñado para ingeniería, GPSS para las simulaciones y COBOL para los negocios (Rivero Espinosa, 2003).

- ***Programación funcional:*** la programación funcional es un paradigma de programación declarativa. Está basada en la utilización de funciones matemáticas, y normalmente los lenguajes basados en este tipo de programación suelen basarse en expresiones que devuelven un valor.

  En esta rama es el programador quien va a especificar qué es lo que se quiere hacer, no se tiene que preocupar con lidiar con los objetos y su estado. Por esto, se define como la programación donde en primer lugar están las funciones, y se centra en las expresiones que se pueden asignar a cualquier variable. Este tipo de lenguajes hace que sea mucho más legible el código, se parece más a leer y escribir en vez de programar.

El origen de este lenguaje se debe al cálculo lambda. El cálculo lambda es un sistema que se desarrolló en el siglo XX, donde se buscaba investigar cuál era la naturaleza de las funciones y de la computabilidad. Para explicar qué es esto del cálculo lambda definiremos dos funciones:

1. *L(x) = x*; donde L es una función que tiene un único argumento y devuelve ese mismo argumento

2. *S(x,y) = x + y*; donde S es una función con dos argumentos que devuelve la suma de ellos

La primera observación que podemos hacer es que la función S puede ser anónima, por lo que la escribiríamos: *x,y → x+y*; del mismo modo haríamos con la función L: *x → x.* Otra observación es que cualquier función que requiera de dos argumentos, como por ejemplo es S, se puede reescribir como una función con un único argumento: *x → (y → x + y)*, *esto* último es lo que comúnmente denominamos currificación. Como última observación podemos decir que una función puede aceptar a otra función como argumento. Siguiendo con las funciones definidas, podemos hacer que la función identidad acepte como argumento a la función S, por lo que tomamos la función *x → (y → x + y)* y le ponemos como argumento *z → z*; por lo que tendremos *x → (z → x + z)* (Cálculo Lambda, 2020).
Algunos de los ejemplos de este lenguaje son Scala, Erlang, Haskell y LISP (Rojas, 2020).

- Pattern Matching dentro de la programación funcional

El Pattern Matching o búsqueda de patrones es un proceso por el cual se busca una secuencia que se repita para resolver una incógnita. La clave para entender este proceso es saber que hay dos patrones o expresiones que se intentan igualar, y además hay unas variables libres que toman los valores necesarios para que esos dos patrones puedan ser iguales. Un ejemplo de Pattern Matching muy sencillo para entender bien sus componentes sería cuando ponemos *'x = 2';* donde x es la variable libre y cogerá el valor 2 para poder hacer igual los dos lados de la ecuación.

Una de las ventajas que tiene el Pattern Matching es que reduce en gran medida el uso de condiciones *'IF'* al hacer una comprobación de una cadena vacía. Te permite crear dos funciones con el mismo nombre y una poner un parámetro fijo de cadena vacía y la otra con un parámetro que sea una variable libre. Al llamar a la función si no le pasas ningún parámetro por defecto ejecutará la función que hemos definido con la cadena vacía, y si la llamamos con un valor ejecutará la otra función con la variable libre. Veamos un ejemplo en lenguaje ELIXIR para clarificar este concepto (Rivero Espinosa, 2003):

*Ilustración 6. Definición de las dos funciones.*

```
iex(9)> defmodule Test do
...(9)>     def foo("") do
...(9)>         {:error}
...(9)>     end
...(9)>     def foo(v) do
...(9)>         {:ok, v}
...(9)>     end
...(9)>
 end
```

*Fuente: (Tomás, 2016)*

*Ilustración 7. Ejecución de las funciones.*

```
iex(10)>
 Test.foo("")
{:error}
iex(11)> Test.foo("CampusMVP")
{:ok, "CampusMVP"}
```

*Fuente: (Tomás, 2016)*

En este ejemplo lo que se pretende es que si se hace una llamada a la función con una cadena vacía devuelve el mensaje de error, y si no devuelve un {: ok, __} donde '__' es el valor que se le pasa como parámetro.

Todos los lenguajes que tienen Pattern Matching utilizan alguna sentencia especial para hacer posible definir varios patrones y así poder devolver valores diferentes según el patrón que se elija. Por seguir con el ejemplo anterior, podemos ver en la siguiente imagen como ELIXIR utiliza la palabra reservada '*case*' para poder definir los patrones. Las sentencias entre corchetes serían en este caso los patrones y lo que debe devolver es lo que sigue a la flecha. En el ejemplo vemos como al pasar el 42 como parámetro la variable '*result*' coge el valor 42, por encajar con el patrón que devuelve v.

*Ilustración 8. Definición de patrones.*

```
iex(12)> result = case Test.foo(42) do
...(12)>     {:ok, v} -> v
...(12)>     {:error} -> -1
...(12)> end
42
iex(13)> result
42
```

*Fuente: (Tomás, 2016)*

- **Multiparadigma:** un lenguaje clasificado como multiparadigma es aquel que es capaz de soportar más de uno de los paradigmas que hemos ido mencionando en este punto. Según lo define Bjarne Stroustrup, famoso científico de la computación, es cuando se puede realizar un programa utilizando más de un estilo de programación. El objetivo fundamental que persiguen estos lenguajes multiparadigma es ayudar a los desarrolladores a utilizar el mejor paradigma para cada tarea, admitiendo de esta forma que un único paradigma no es capaz de resolver todos los problemas eficientemente o de la forma más fácil, por esto se hace la combinación de dos o más paradigmas. Por ejemplo, lenguajes como visual Basic, PHP o C++ combinan el paradigma imperativo y la orientación a objetos (Diccionario Sensagent, 2021).

### 2.3.3 Generaciones de lenguajes de programación

Otra forma de clasificar los lenguajes de programación sería según su potencia (Rivero Espinosa, 2003):

- ***Primera generación***

Esta va unida totalmente a los lenguajes de bajo nivel, debido a que únicamente son los lenguajes máquina, los que como hemos comentado anteriormente no requieren de ningún tipo de traducción. En estos casos el compilador puede leerlo directamente.

- ***Segunda generación***

Aquí también solo aparece lenguaje de bajo nivel como es el lenguaje ensamblador. Este requiere de una traducción, aunque es bastante simple porque una instrucción equivale a una línea de código máquina.

- ***Tercera generación***

Aquí ya entran los lenguajes de alto nivel. Normalmente este tipo de lenguajes presentan un gran número de instrucciones con códigos que llegan a ser difíciles de entender, a su vez que difíciles de mantener y depurar. Suelen estar orientados a archivos y desarrollados por lote. Al tratarse de lenguajes de alto nivel requerirán de varias instrucciones para ser traducidos al lenguaje máquina.

- ***Cuarta generación***

Estos están por encima de los de alto nivel. Requieren que se les especifique la tarea que hay que realizar y el propio sistema será el que determine cómo ejecutarla. Ofrecen opciones predeterminadas para que el programador en este caso se enfrente a una interfaz y no interactúe directamente con el código. Por esto los errores serán mucho más sencillos de localizar.

### 2.3.4 Sistema de tipos

Un tipo es un conjunto de valores que presentan el mismo significado, así como las relaciones y operaciones que se aplican sobre esos valores. Su función principal es ayudar a los programadores a ordenar las ideas. Asocian tipos con un conjunto de mensajes, por lo que se puede entender más fácilmente que tipo va a tener una variable en concreto. Por otro lado, a los propios lenguajes de programación les sirve tener un tipificado específico para así poder detectar errores más rápidamente en las asignaciones de variables. Por último, también facilita que el entorno de desarrollo sea mucho más intuitivo y pueda asistir al escribir un programa. (Passerini, 2012)

### 2.3.5 Chequeo de tipificado

La verificación de los tipos puede ser durante la compilación (comprobación estática) o durante la propia ejecución del programa (comprobación dinámica). Cabe destacar que actualmente hay lenguajes que permiten combinar el tipado estático y dinámico, es decir, algunos lenguajes estáticos permiten escribir código que no sea verificado de forma estática. Por otro lado, si un lenguaje de programación impone de una manera más restrictiva el tipificado diremos que es un lenguaje fuertemente tipado, y por el contrario lo llamaremos débilmente tipificado (Rivero Espinosa, 2003).

## 2.4 Tipos de programación

En el mundo de la programación hay diferentes ramas donde elegir, aunque todas tienen su raíz en el pensamiento algorítmico y lógico. Sin embargo, hay que tener muy claro que en la práctica cada rama o especialización es un mundo totalmente diferente. Cada una de las áreas requiere de unas metodologías, herramientas y lenguajes específicos de programación.

### 2.4.1 Programación web

Cuando hablamos de la programación web nos vamos a referir a todo lo que puede funcionar sobre un navegador como puede ser Firefox, Chrome, Safari o muchos otros. Para continuar con este apartado es conveniente diferenciar los sitios webs y las aplicaciones web.

Por lo general, los sitios webs son páginas que simplemente se limitan a mostrar información. El ejemplo más clásico son los sitios que son diseñados y publicados a través de un sistema que se encarga de la gestión de contenidos (por sus siglas en inglés CMS). Los sitios web más conocidos son Joomla y WordPress (Della Pittima, 2021).

Por otro lado, las aplicaciones web se basan en el desarrollo de un software más complejo que hace que sea necesario utilizar uno o más de un lenguaje de programación diferente. Normalmente es necesario diferenciar dos partes, lo que ve el usuario por pantalla (el navegador) y lo que el usuario no ve (ejecución del lado del servidor). También requiere conocer la estructura de la programación web. Las aplicaciones web que suelen sonar más son entre otras Google Drive, WhatsApp web y Spotify web (Della Pittima, 2021).

Los fundamentos de la programación web se basan principalmente en el modelo llamado Modelo-Vista-Controlador (MVC), el cual recibe su nombre por las tres partes que lo componen. Este modelo es una pauta en el diseño de software que es utilizado comúnmente para poder implementar interfaces de usuario, lógica de control y datos. Hace un énfasis en la separación entre la visualización y la lógica de negocio (MDN contributors, 2020).

- Modelo: maneja la lógica y los datos de negocios. Es la parte que define qué datos debería tener la aplicación.

- Vista: se ocupa de la presentación y el diseño de los datos que le proporciona el modelo.

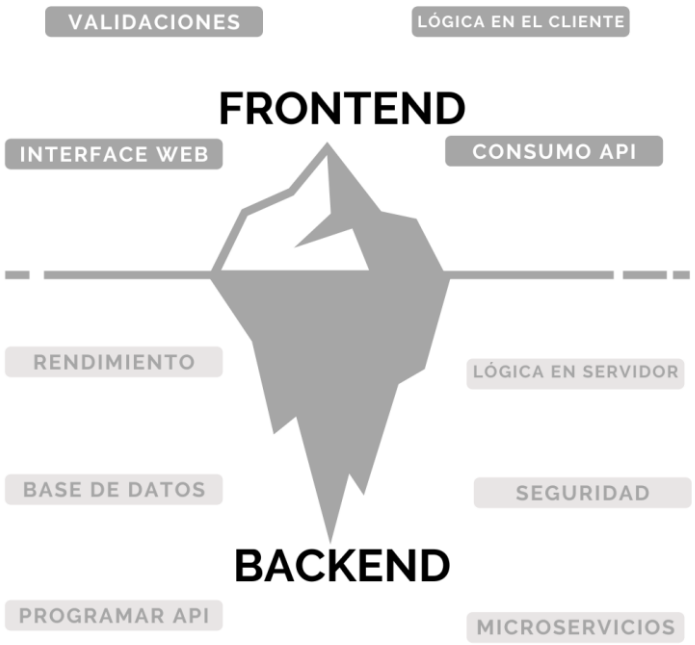
- Controlador: es la parte encargada de ir actualizando el modelo y/o la vista a consecuencia de las entradas de los usuarios.

Por lo explicado antes de la diferenciación entre lo que ve el usuario en el navegador y lo que se ejecuta por el servidor podemos definir dos perfiles (Della Pittima, 2021):

- Front-End: Es la parte encargada de desarrollar los programas que se ejecutan en el navegador, por lo que es la parte que el usuario final verá. Los lenguajes más comunes para esta parte son HTML, JavaScript y CSS.

- Back-End: es la parte encargada de desarrollar los programas por parte del servidor, es decir, donde se decide la mayor parte de la lógica de negocio y donde se crea la API que es la interfaz. Esto es debido a que es la mejor forma de que el Front pueda interactuar y comunicarse con el Back. Los lenguajes más comunes en este lado son Python, PHP, Java y C#.

*Ilustración 9. Resumen back-end y front-end.*




*Fuente: elaboración propia*

### 2.4.2 Programación móvil

La programación móvil o Mobile hace referencia al desarrollo de las aplicaciones para teléfonos móviles o smartphones. En esta rama existen dos grandes y diferenciadas ramas: Android de Google e IOS de Apple.

Hay un tipo de aplicaciones llamadas nativas que son las que se desarrollan sobre el propio sistema operativo, y se deben desarrollar especialmente para cada uno de los sistemas. Cada una de las plataformas cuenta con sus propios lenguajes de programación y herramientas, lo que quiere decir que una aplicación desarrollada para Android no funciona sobre una de IOS y a la inversa. Lo más usual es que para desarrollar aplicaciones para Android sea necesario conocer Kotlin o Java, y por el lado de IOS es necesario conocer Swift.

Para resolver la problemática de que algo hecho para Android funcione para IOS aparecen las aplicaciones multiplataforma. Estas aplicaciones se desarrollan para que una única aplicación pueda funcionar correctamente para cualquiera de los dos sistemas. Para conseguir esto, se hace uso de un lenguaje de programación común para los dos sistemas y por debajo de este hay otro lenguaje que hace de traductor para cada sistema. Algunos ejemplos de este tipo de lenguaje son React Native e Ionic (Della Pittima, 2021).

Otra de las alternativas que podemos encontrar para solucionar la incompatibilidad entre IOS y Android son las Progressive Web Apps (PWA), que son aplicaciones web pero que hacen la función de una nativa[6]. Cabe destacar que para hacer uso de las PWA será necesario tener los conocimientos requeridos para la programación web (Della Pittima, 2021).

### 2.4.3 Machine Learning (Aprendizaje Automático)

La programación del aprendizaje automático o Machine learning es una parte de la IA (inteligencia artificial), pero no se suele asociar con programación como tal, ya que en realidad se trata de un conjunto de algoritmos que se usan en las aplicaciones. Esta disciplina se ocupa del aprendizaje automático de grandes volúmenes de datos.

Hoy en día almacenamos y también generamos un volumen inmenso de datos, que día a día va creciendo, por lo que ha surgido el término *'Big Data'*. Este término hace referencia al volumen inmenso de datos que existen en el mundo actual. Muy unido a este término encontramos la ciencia de datos, la cual se encarga del análisis de estos datos. Por esto último podemos afirmar que el aprendizaje automático va unido a la ciencia de datos, que engloba al Big Data y a la minería de datos.

Este tipo de programación está ligada al desarrollo de los algoritmos que hacen posible los modelados y la búsqueda de los patrones que se encuentran en los datos, con un objetivo claro que es llegar a predecir comportamientos. Los lenguajes de programación más conocidos para esta rama son R y Python (*EDteam*, 2020).

### 2.4.4 Programación de videojuegos

El desarrollo de videojuegos se suele componer de diferentes perfiles, y entre ellos se encuentra la programación. El proceso del desarrollo suele componerse del diseño y de la creación de diferentes escenarios y personajes. Los motores de juegos o Engine más conocidos para poder

[6] Aplicación nativa: aplicación que ha sido desarrollada en el lenguaje que es específico de un sistema operativo.

desarrollar videojuegos es Unity 3D que está basado en C# y Unreal Engine que está basado en C++ *(EDteam*, 2020).

### 2.4.5 Programación embebida

Son de los programas más sencillos. Se encuentran integrados en un chip o placa electrónica, por eso reciben el nombre de 'embebidos'. Su uso más común es en electrodomésticos. El conocimiento requerido para esta programación es la electrónica digital, cómo funcionan los sensores para poder tomar diversa información del medio y los protocolos de comunicación. Los lenguajes que predominan en esta rama son C y Java (Della Pittima, 2021).

### 2.4.6 Programación de escritorio

La programación de escritorio recibe ese nombre debido a que es necesario la instalación del sistema en el sistema operativo (Linux, Windows y MacOS) del propio ordenador. Cuando aún no se utilizaban los teléfonos móviles y las aplicaciones web esta era la rama más conocida y utilizada (Della Pittima, 2021).

### 2.4.7 Programación de sistemas operativos (SO)

La programación de SO hace referencia no solo al desarrollo de éstos, sino también a su mantenimiento. Es el software más cercano al hardware, por lo que son programas que interaccionan de forma directa con el hardware. Estos son los lenguajes que antes hemos definido como de bajo nivel (lenguaje ensamblador) (Della Pittima, 2021).

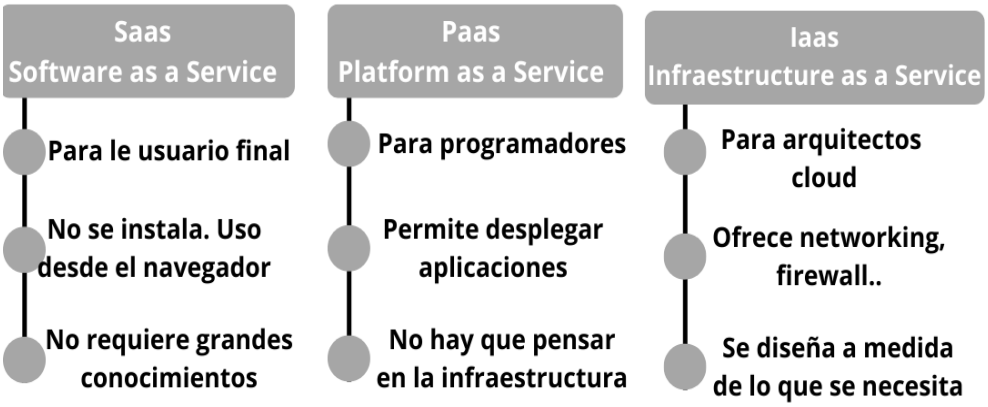
### 2.4.8 Programación de realidad virtual y aumentada

Esta programación es conocida desde hace relativamente poco tiempo, se suele utilizar para dispositivos como cascos o lentes que ocupan toda la visión del individuo que las lleve puestas. Estos dispositivos sirven para transportarte a otros lugares existentes o imaginarios. Este tipo de tecnología tan innovadora se desarrolla con lenguajes como C#, JavaScript, Python o Java (*EDteam*, 2020).

### 2.4.9 Programación en la nube o cloud computing

Hoy en día, el mundo de la nube es algo que nos acompaña a todos y cada uno de nosotros. Ofrece servicios de almacenamiento, redes, bases de datos, análisis, software e inteligencia artificial gracias a internet. Es un servicio que normalmente se paga por lo que se usa, por lo que resulta muy favorecedor para los usuarios. Hay tres modelos de servicios en la nube:

*Ilustración 10. Resumen de los modelos de servicios en la nube.*




*Fuente: elaboración propia*

# CAPÍTULO 3: CASO DE ESTUDIO

En este apartado nos centraremos en analizar algunos de los lenguajes que han ido apareciendo en la última década. Los iremos analizando por orden cronológico incluyendo en esta relación los que más importancia y uso han tenido, ya que, algunos de los que han aparecido no han sido muy utilizados.

Las características que se van a mencionar sobre cada uno de los lenguajes son las más significativas, dado que analizar al completo cada uno de los lenguajes sería prácticamente imposible debido principalmente a que son lenguajes vivos que los van mejorando y modificando.

## 3.1 Lenguajes de programación de la última década.

### 3.1.1 Rust (Rust, 2010)

| | |
|---|---|
| Apareció en: | 2010 |
| Diseñador por: | Graydon Hoare |
| Paradigma: | Multiparadigma |
| Influido por: | Alef, C#, C++, Calmp4, Common, Lisp, Cyclone, Erlang, Haskell, Hermes, Limbo, Napier, Napier88, Newsqueak, NIL, Ruby, Shather, Standard ML, Ocaml, Racket y Swift |
| Ha influido a: | C#, Elm, Idris, Swift |
| Sistema operativo: | Linux, macOS, Windows |

Rust dio sus primeros pasos en el 2006 de la mano del ingeniero de la empresa Mozilla Graydon Hoare. Lo que este ingeniero pretendía era integrar algunas ideas que eran conocidas y buenas en un lenguaje nuevo, ya que, anteriormente solo se había integrado en lenguajes muy pobres.

Este proyecto fue creciendo hasta que se anunció en 2010**.** Algunos de los que podríamos decir que son sus predecesores son: Alef, C++, Erlang, Hermes, Limbo, Napier, Napier88, Newsqueak, NIL, Sather, Standard ML.

Se define como un lenguaje de programación multiprograma y de alto nivel. El propósito de su creación fue mejorar el rendimiento y la seguridad, haciendo especial énfasis en la concurrencia segura.

Si observamos sus sintaxis es muy similar a C++, pero a diferencia de este lenguaje puede garantizar, haciendo uso de un verificador de referencias, la seguridad de la memoria. Para alcanzar esta seguridad de memoria, Rust no utiliza recolección de basura y, a su vez, mantiene como opcional el recuento de referencias (Pastor, 2021). Por esas razones muchos usuarios lo denominan 'el sucesor de los legendarios C y C++'.

Rust hace que las variables por defecto sean inmutables, lo que quiere decir que una vez se le asigna un valor a una variable, al compilarlo, si se encuentra otra asignación a esa misma variable va a devolver un error. Solo puede asignarse un valor a una variable inmutable. Esta funcionalidad hace que el código sea mucho más fácil de razonar, aunque a su vez resulta muy útil tener variables mutables, por lo que Rust ofrece la palabra reservada 'mut' que poniéndola delante del nombre de la variable la convierte en mutable. Por ejemplo, en el siguiente código podemos ver como primeramente se declara una variable '*x =5* ', la imprime por pantalla, y luego se le asigna un '6' y se vuelve a imprimir. Al compilar este código el compilador devolverá un error por ser las variables por defecto inmutables.

*Ilustración 11. Variables inmutables Rust.*

```
fn main() {
    let x = 5;
    println!("The value of x is: {x}");
    x = 6;
    println!("The value of x is: {x}");
}
```

*Fuente: (Klabnik & Nichols, 2021)*

Sin embargo, en este lenguaje hay una característica que ellos denominan *sombreado*. Se trata de poder declarar una misma variable ya declarada anteriormente. A la primera variable declarada le dicen que está sombreada por esta segunda declaración, por lo que la segunda variable va a eclipsar de cierta forma a la primera. Veamos un ejemplo:

*Ilustración 12. Variables sombreadas Rust.*

```
fn main() {
    let x = 5;

    let x = x + 1;

    {
        let x = x * 2;
        println!("The value of x in the inner scope is: {x}");
    }

    println!("The value of x is: {x}");
}
```

*Fuente: (Klabnik & Nichols, 2021)*

Lo que devolvería el programa de esta última ilustración no sería un error, sino que serían las siguientes líneas, donde una variable va eclipsando a la anterior como hemos comentado:

*Ilustración 13. Salida variable sombreada Rust.*

```
$ cargo run
   Compiling variables v0.1.0 (file:///projects/variables)
    Finished dev [unoptimized + debuginfo] target(s) in 0.31s
     Running `target/debug/variables`
The value of x in the inner scope is: 12
The value of x is: 6
```

*Fuente:(Klabnik & Nichols, 2021)*

Otras de las características que definen a Rust es que es un lenguaje de tipado estático, debe conocer cuáles son los tipos de las variables en el momento que se quiera compilar. El compilador también tiene la funcionalidad de inferir el tipo de datos según el uso que le podamos ir dando a la variable en todo el código, pero en el caso que pueden servir muchos tipos diferentes se deberá especificar.

Sin embargo, lo que más destaca este lenguaje es por lo que denominan *Ownership,* que no es más que un conjunto de reglas que utiliza Rust para poder administrar la memoria de la forma más eficiente posible. Algo que es común a todos los programas es que tiene que prestar atención a la forma en la que usan la memoria mientras son ejecutados y, la mayoría de estos buscan trozos de la memoria que no se están utilizando y lo borran. Este proceso es más conocido comúnmente como 'recolección de basura'. Rust no utiliza este sistema de gestión de memoria, lo que hace es que la memoria pasa a ser administrada por un sistema de propiedad (*Ownership*) donde existen unas reglas que el compilador se encarga de verificar, y que, si son violadas, el programa no podrá ser compilado. Estas reglas que hemos mencionado son:

- Cada valor en este lenguaje tiene un "*propietario*"
- Solo está permitido que haya un propietario por valor al mismo tiempo
- Cuando este propietario se queda fuera del alcance, entonces el valor se podrá eliminar liberando la memoria.

En la siguiente imagen podemos ver más claramente estos conceptos. Hay dos puntos clave en la vida de las variables, cuando se declara la variable *s* entra en el alcance, hasta que cuando se cierra el corchete termina su alcance, borrando así de memoria su valor.

*Ilustración 14. Ownership Rust.*

```
{                      // s is not valid here, it's not yet declared
    let s = "hello";   // s is valid from this point forward

    // do stuff with s
}                      // this scope is now over, and s is no longer valid
```

*Fuente:(Klabnik & Nichols, 2021)*

Esta gestión de memoria ha sido muy llamativa por la gran eficiencia que presenta. Por lo que es una de sus características más importantes. Por ello muchas empresas importantes como pueden ser: AWS, ARM, GitHub, Microsoft, Google y Facebook están empezando a utilizar este lenguaje. Otra de las cosas que más recalcan de este lenguaje es que ofrece un rendimiento como C++ y además se interesa más por la seguridad del código (Pastor, 2021).

### 3.1.2 Ceylon (Ceylon, 2011)

| | |
|---|---|
| Apareció en: | 2011 |
| Diseñador por: | Red Hat |
| Paradigma: | Orientado a objetos |
| Influido por: | Java, Objective-C, C++, Smaltalk, Eiffel |
| Sistema operativo: | Multiplataforma |

Ceylon fue desarrollado por *Red Hat*, más concretamente por el desarrollador *Gaving* King que también creo *Hibernate*[7]*. S*e caracteriza por ser un lenguaje de programación orientado a objetos. Es muy similar al lenguaje de programación *Java,* ya que, mucha de su sintaxis proviene de este lenguaje. Presenta un tipado estático con inferencia de tipos que se ejecuta sobre la propia máquina virtual de Java (JVM[8]), y que también se puede compilar a JavaScript (SG, 2013).

El objetivo de crear este nuevo lenguaje fue para cubrir algunas deficiencias que presenta Java, aunque no llega a ser un sustituto, ya que, como se ha comentado también puede tener código en JavaScript, además de ejecutarse en la propia JVM. El poder ser ejecutado en esta máquina hace que Ceylon sea un lenguaje portátil para una gran variedad de arquitecturas (SG, 2013).

[7] Hibernate: herramienta que ayuda en el mapeo de atributos entre una base de datos y el modelo de objetos de una aplicación.

[8] JVM: Java Virtual Machine

Lo que pretende Ceylon es mejorar la legibilidad del lenguaje Java, la incorporación de dispositivos de lenguaje funcional, como son las funciones de orden superior[9], típico del lenguaje C, y también mejorar la modularidad[10]. Estas características es lo que diferencia principalmente a Ceylon ya que, al estar pensado para software empresarial de gran escala, se necesitarán diversos bloques para poder componer este software. El tipo de proyectos al que va destinado Ceylon son aquellos que van a contar con decenas de miles de líneas de código. Esto último implica que los equipos de desarrolladores vayan a estar formado por más de 10 personas, y teniendo en cuenta en el mundo que vivimos seguramente no se encuentren ubicadas ni en el mismo lugar geográfico. También serán proyectos que en un futuro tendrán que ser mejorados o corregidos, y hay que tener en cuenta que los que se encarguen de estas tareas de mantenimiento pueden ser otros programadores diferentes. Por todo esto, Ceylon se basa firmemente en la importancia sobre la legibilidad y la predictibilidad (SG, 2013).

Ceylon tiene que ser sumamente fácil de entender y legible por terceros ajenos al desarrollo del proyecto, y además debe facilitar la trazabilidad de los errores tanto de compilación como de ejecución, para poder llegar lo antes posible a la causa. Con esta base, Ceylon intenta hacer mucho más sencilla la creación de proyectos empresariales de gran volumen (SG, 2013).

Debido a que Ceylon es un lenguaje orientado a objetos, el código que se escribe se basa en el concepto de clases, el cual es un tipo que incluye un conjunto de operaciones o métodos y a su vez define cómo se inicializará el objeto de dicha clase por medio de su inicializador de clase, que es muy parecido al constructor de Java. No hay opción a crear más de un inicializador de clase, al igual que no hay deconstructor (EcuRed, 2021).
A continuación, se mostrará un ejemplo de una clase Ceylon simple

[9] Funciones de orden superior: las funciones pueden tomar otras funciones como entrada o salida.

[10] Modularidad: característica de algunos sistemas que ayuda a ser visto como la unión de varios bloques o partes que gracias a que interactúan entre ellos alcanzan un objetivo.

*Ilustración 15. Ejemplo clase de Ceylon.*

```
01    doc "Simple Counting Class"
02    class Counter( Natural? start ) {
03
04      doc "Class Initializer"
05      variable Natural count := 0;
06      if (exists start) {
07        count := start;
09      }
10
11      doc "The incrementer"
12      shared void increment() {
13        count++;
14      }
15
16      doc "The getter"
17      shared Natural currentValue {
18        return count;
19      }
20
21      doc "The setter"
22      shared assign currentValue {
23        count := currentValue;
24      }
25
26    }
```

*Fuente: (EcuRed, s.f)*

En la clase, el usuario puede proporcionar o no un valor, y éste será denotado con 'Type?'. El inicializador de la clase es el que define la variable privada, y luego cuál será la lógica de la inicialización. Lo primero que hace el inicializador es comprobar si la variable inicial existe, si es así, se utiliza para su conteo y luego, su método 'shared' definirá el incrementador, por lo que cuando el método es llamado simplemente se incrementa el contador. Por último, se definen los métodos 'getter' y 'setter', que funcionan del mismo modo que los métodos 'get' y 'set' de Java. Cabe destacar el uso de la palabra clave 'assign', la cual es utilizada para crear un atributo de variable, en este ejemplo utilizado para definir el valor del contador. En Ceylon también encontramos como para asignar un valor a un variable se utiliza ': =', mientras que '=' se utiliza para valores inmutables.

*Ilustración 16. Uso de la clase de ejemplo.*

```
01    Counter cnt = Counter(1);
02    cnt.increment();
03    writeLine( c.currentValue );
```

*Fuente: (EcuRed, s.f)*

Este bloque de código mostrado en la ilustración 16 nos enseña cómo se utilizaría el código mostrado en la ilustración 15. En Ceylon no es necesario el uso de la palabra 'new' por lo que la instancia de la clase se hace con la línea 01, y en el ejemplo es denominada 'Counter' y se le pasa el parámetro 1. A continuación, se incrementa el contador con el método 'increment ()' y luego se muestra por pantalla.

### 3.1.3Dart (Dart, 2011)

| | |
|---|---|
| Apareció en: | 2011 |
| Diseñador por: | Google |
| Paradigma: | Orientado a objetos |
| Influido por: | C#, JavaScript, Java y CoffeScript |
| Sistema operativo: | Multiplataforma |

Dart ha sido creado por Google y fue presentado en la Conferencia de Aarhus (Dinamarca), más concretamente en 2011. Es un lenguaje de programación estructurado para la web y basado en clases, es decir, orientado a objetos. Actualmente es un proyecto en etapas tempranas de desarrollo y de código abierto. Busca ser eficiente, simple y escalable, lo que conforma las buenas prácticas de la programación en la actualidad.

En un principio, y según anunciaron, Dart estaría disponible para los navegadores Chrome, Firefox 4+ y Safari 5+, aunque la idea era ir añadiendo plataformas a esta lista, para siempre estar a la vanguardia.

La idea principal de Google al crear este lenguaje era que pudiera ser utilizado tanto por una única persona como por un grupo de desarrolladores. Para cumplir esto se basa en la utilización de tipos opcionales, es decir, se puede comenzar a programar sin añadir los tipos y añadirlos después cuando realmente sean necesarios, lo que hace que el código de Dart sea más flexible y familiar para el usuario. Por ello decidieron que el tipo de tipado (estático o dinámico) quedara a elección de cada programador o grupo de programadores.

La familiaridad es buscada para reducir al máximo posible la curva de aprendizaje, por lo que se podría decir que Google buscaba que fuera un lenguaje fácil de aprender e intuitivo. Para ayudar en esto último, desde el momento de su publicación, en la propia página web de Dart se encuentran diversos videotutoriales, donde también se aceptan colaboraciones de otros usuarios, lo que ayuda mucho a que sea fácil aprenderlo y que al plantearse una duda haya una comunidad que pueda ayudarse entre sí.

El código de Dart puede ser ejecutado de dos maneras. La primera es haciendo uso de un compilador que traduzca el código para poder ejecutarlo en el motor de JavaScript y la otra manera, es hacer uso de una máquina virtual nativa (disponible en Chrome). Lo que va muy ligado a la idea de ser un lenguaje altamente disponible para diferentes entornos, lo que resulta más atractivo para aquellas empresas que trabajan en multitud de proyectos diferentes.

Como ellos mismos lo definen, se trata de un lenguaje flexible y estructurado a la vez. Estructurado por que los programas no son creados desde cero, sino que hay unas estructuras con control básico, por lo que se obtiene un código ordenado y claro, en el que solo tienes que ir añadiendo tus condiciones. Aunque también te permite empezar de cero, pero eso ya se queda a gusto del desarrollador.

Por todo esto podríamos definir tres grandes objetivos perseguidos por Dart:

- Funcionar sobre cualquier navegador
- Sintaxis familiar y fácil de aprender.
- Lenguaje estructurado y flexible.

Dart está principalmente diseñado para resolver algunos problemas que presenta JavaScript y así llegar a ofrecer unos mejores resultados, aunque cabe destacar que en la presentación de este lenguaje no se hizo una mención directa a JavaScript. (EcuRed, s.f).

A continuación, se muestra un ejemplo de la función Fibonacci y otra función principal que controla la ejecución:

*Ilustración 17. Ejemplo código Dart.*

```
int fib(int n) {
 if (n < = 1) return n;
 return fib(n - 1) + fib(n - 2);
}
 main() {
 print('fib(20) = ${fib(20)}');
}
Una clase y una función principal para calcular la distancia entre dos puntos en un plano X-Y:
class Point {
 Point(this.x, this.y);
 distanceTo(Point other) {
   var dx = x - other.x;
   var dy = y - other.y;
   return Math.sqrt(dx * dx + dy * dy);
 }
 var x, y;
}
main() {
 Point p = new Point(2, 3);
 Point q = new Point(3, 4);
 print('distance from p to q = ${p.distanceTo(q)}');
}
```

*Fuente: (EcuRed, s.f)*

Como bien se aprecia en la figura anterior el código de Dart resulta muy parecido a lenguajes como C/C++, PHP o Java, por lo que ayuda en su legibilidad y reduce la curva de aprendizaje significativamente.

### 3.1.5 Red (Red, 2011)

| | |
|---|---|
| Apareció en: | 2011 |
| Diseñador por: | Nenad Rakočević |
| Paradigma: | Multiparadigma |
| Influido por: | Rebol y C |
| Ha influido a: | C#, Elm, Idris, Swift |
| Sistema operativo: | Linux, macOS, Windows y síbala |

Red fue presentado por Nenad Rakočević en el año 2011, y lo definieron como un lenguaje de programación imperativo y funcional. Es un lenguaje de programación que tiene como finalidad superar las limitaciones que presenta el lenguaje de programación Rebol, pero manteniendo en gran medida sus sintaxis y semántica. Al igual que su predecesor, Red es totalmente compatible con la metaprogramación (Glosario 4.1) y a su vez con los DSL (lenguajes específicos de dominio) (Hmong, 2021-b).

Red pretende ser un lenguaje de programación de pila completa, es decir, que se pueda utilizar tanto a nivel alto (DSL y HUI[11]), como a bajo nivel (controladores de dispositivos y sistemas operativos). Para poder ofrecer esta característica de pila completa Red se distribuye en dos partes (Hmong, 2021-b):

- Red: es un lenguaje donde se trata el código como datos, es decir, se trata de un lenguaje homoicónico (Glosario 4.2). Es capaz de soportar la metaprogramación. La biblioteca que tiene en tiempo de ejecución está escrita por la otra parte que se denomina Red/System, el cual utiliza un método híbrido. Por un lado, compilará lo que pueda deducir de manera estática, y si no es así hará uso de un intérprete integrado (Hmong, 2021-b).

- Red/System: es una parte muy similar al lenguaje C, aunque está empaquetado en la estructura de Rebol. Un ejemplo de esto sería que en vez de escribir "if (x > y) {print("Hello\n");}"

[11] HUI (graphical user interface): la interfaz gráfica de usuario que permite interactuar a los usuarios con dispositivos electrónicos.

se escribiría *"if z > y [print "Hello"]". Esta parte trata de proporcionar funciones de programación al sistema* (Hmong, 2021-b).

Uno de los objetivos de Red es permanecer totalmente independiente de cualquier otra herramienta. Para cumplir dicho objetivo Red autogenera su propio código, por lo que se hace posible la realización de las compilaciones cruzadas[12] de programas desde cualquier plataforma que sea compatible con cualquier otra. Todo esto se hace a través de un intérprete de línea de comandos (Hmong, 2021-b).

Los siguientes puntos muestran las metas que se propusieron cuando presentaron este lenguaje en el 'Software Freedom Day 2011', que es donde vio la luz este lenguaje (Programming_language, s.f):

- Compacidad: maximizar la productividad siendo muy expresivo.
- Simplicidad: no es necesario un entorno integrado de desarrollo de software para el diseño de aplicaciones.
- Velocidad: maximizar la rapidez del lenguaje.
- Verde: que deje la menor huella posible en el medioambiente, ya que, los recursos son limitados.
- Portabilidad.
- Flexibilidad.
- Ubicuidad: que sea capaz de ser difundido por todas partes.

### 3.1.6 OPA (OPA, 2011)

| | |
|---|---|
| Apareció en: | 2011 |
| Diseñador por: | MLstate |
| Paradigma: | Multiparadigma: funcional e imperativo |
| Influido por: | OCaml, Erlang y JavaSCript |
| Sistema operativo: | Linux, macOS y Windows |

OPA es un lenguaje de programación web lanzado en 2011 y es de código abierto. Fue publicado bajo la Licencia Pública General Affero de GNU. Se puede hacer uso de este lenguaje tanto del lado del cliente como del lado del servidor, ya que, como hemos explicado en apartados

[12] Compilación cruzada: hace referencia a cuando un compilador es capaz de crear el código ejecutable para una plataforma distinta en la que se está ejecutando dicho compilador.

anteriores la programación web tiene dos partes y lo que se persigue con este lenguaje es simplificar dichas partes en un solo lenguaje (Kripkit, 2021).

Los programas escritos en OPA son compilados en node.js por la parte del servidor, y por la parte del cliente en JavaScript. Hace uso de un compilador que consigue que todas las comunicaciones entre estas dos partes de la programación web sean totalmente automáticas y mucho más sencillas para el desarrollador. La principal ventaja de este lenguaje es que hace que los usuarios no se vean en la necesidad de tener que instalar un complemento en el navegador. Otros de los puntos que se definen como ventaja es que este lenguaje trabaja con una base de datos NoSql, a la vez que también pretende eliminar las dependencias entre diversos lenguajes y crear un nuevo lenguaje con el que se puede llegar a todo (Análisis de sistemas, 2015).

OPA está formada por un servidor web, un motor de ejecución distribuido y una base de datos. Utiliza una tipificación fuerte y estática, lo que resulta muy útil para la protección contra ataques, y es un lenguaje de tipo funcional (Kripkit, 2021).

El 'Hola mundo' tradicional que produce un servidor web en una página estática se puede escribir en OPA de una forma muy sencilla como muestra la siguiente ilustración (Binsztok et al., 2013):

*Ilustración 18. Hola mundo en OPA.*

```
Servidor . start ( Server . http ,  { title :  "Hola"  , página :  function ()  {    }  } )
```

*Fuente: (Binsztok et al., 2013)*

Su compilación, como hemos comentado anteriormente, se hará en un archivo ejecutable independiente JS:

*Ilustración 19. Compilación en OPA.*

```
$ opa hello_web.opa
```

*Fuente: (Binsztok et al., 2013)*

Al ejecutar dicho archivo comprimido de la siguiente forma se iniciará la aplicación web:

*Ilustración 20. Inicio de aplicación OPA.*

```
$ ./hello_web.js
```

*Fuente: (Binsztok et al., 2013)*

Por lo que hemos definido de este lenguaje se puede concluir que OPA es un lenguaje orientado y pensado totalmente para la web, lo cual obliga a establecer elementos de interoperabilidad[13], permitiendo así su ejecución en prácticamente todos los navegadores webs existentes.

### 3.1.7 Elixir (Elixir, 2011)

| | |
|---|---|
| Apareció en: | 2011 |
| Diseñador por: | José Valim |
| Paradigma: | Multiparadigma: funcional con Pattern Matching, concurrente, distribuido y orientado a procesos |
| Influido por: | Erlang, Ruby y Clojure |
| Sistema operativo: | Linux, macOS y Windows |

Elixir lo podemos definir como un lenguaje de programación dinámico[14]. Se distingue por ser bastante robusto y estable, debido principalmente a que los programas que se crean con este lenguaje se ejecutan en la máquina virtual BEAM que es de Erlang. Los programas se compilan y dan como resultado bytecode[15], que seguidamente es ejecutado por la MV. Al utilizar la MV de Erlang hace que sea posible reutilizar diferentes funcionalidades de ese lenguaje.

Este lenguaje sigue el paradigma de programación funcional, dejando de lado la programación por objetos y centrándose de lleno en módulos y funciones. Hay que destacar que utiliza el

[13] Interoperabilidad: capacidad de los sistemas de información de compartir datos y a su vez hacer posible el intercambio de información.

[14] Lenguaje de programación dinámico: las operaciones que se realizan en la compilación pueden llegar a realizarse durante la ejecución. Por ejemplo, es posible agregar nuevos métodos a un objeto durante la ejecución de un programa.

[15] ByteCode: código intermedio que es más abstracto que el código máquina. Es tratado como un archivo binario.

Pattern Matching y como hemos visto en la descripción de esta funcionalidad (Punto 1.4.10.1) es una característica que hace al lenguaje muy potente.

La línea de código para generar el 'Hola mundo' sería de la siguiente forma, donde *'puts'* es una función, que como podemos ver en la siguiente ilustración, no necesita del uso de paréntesis (García Pérez, 2018).

*Ilustración 21. Ejemplo 'Hola mundo'.*

```
IO.puts "Hola Mundo"
```

*Fuente: (García Pérez, 2018)*

Al tratarse de un lenguaje dinámico no presenta la necesidad de especificar el tipo de datos a utilizar, el tipo se asigna en tiempo de ejecución, por lo que este lenguaje presenta un tipado dinámico. En la siguiente ilustración podemos ver este hecho, donde ninguna de las 3 variables tiene definido el tipo, siendo cada una de un tipo diferente.

*Ilustración 22. Declaración de variables en ELIXIR.*

```
#Variables

curso = "Elixir"
version = 1.6
completado = true
#Con <> podemos concatenar strings!
titulo = "Curso de " <> curso <> "."
```

*Fuente: (García Pérez, 2018)*

Las funciones de ELIXIR se pueden definir como el centro de las aplicaciones (como en cualquier otro lenguaje funcional), por este motivo se ha hecho bastante sencillo el poder definirlas con este lenguaje. Por ejemplo, para definir una función que sume dos variables que se pasan como parámetros podríamos escribir:

*Ilustración 23. Definición de una función en ELIXIR.*

```
def suma(valor1, valor2) do
 valor1 + valor2
end
```

*Fuente: (García Pérez, 2018)*

Todas las funciones que se definan en este lenguaje deben de retornar un valor, pueden ser anónimas y no requieren el uso de los corchetes, ya que se utilizan las palabras reservadas '*do*' y 'end' para designar el principio y fin de la función. También se permite asignar funciones a variables como observamos en la siguiente ilustración:

*Ilustración 24. Asignación de una función anónima.*

```
suma = fn (valor1, valor2) -> valor1 + valor2 end
```

*Fuente: (García Pérez, 2018)*

Por lo que podemos ver de los ejemplos en las ilustraciones anteriores, Elixir es un lenguaje con una sintaxis bastante intuitiva, de forma que sin haber visto antes este lenguaje se podría seguir fácilmente su código, por lo que una vez más se crea un lenguaje de programación donde prima la simplicidad del lenguaje.

### 3.1.8 Elm (Elm, 2012)

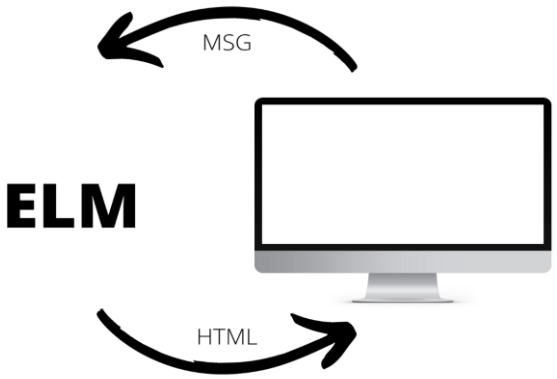
| | |
|---|---|
| Apareció en: | 2012 |
| Diseñador por: | Evan Czaplicki |
| Paradigma: | Programación funcional |
| Influido por: | Haskell, Standard ML, OCaml y F# |
| Sistema operativo: | Linux, macOS y Windows |

El lenguaje ELM es un lenguaje funcional, utilizado para la parte del cliente en una web, y esto es fácil de hacer por su compilación en JavaScript, y de hecho el mayor uso de este lenguaje es para la programación web.

ELM maneja los nombres como si de funciones se tratase. El valor que se utiliza para la unión de los tipos es una función, y como es de esperar, cada una de las funciones suele estar compuesta por otras funciones parciales que se aplican a sus argumentos. También se define como un lenguaje fuertemente tipado, debido a que sus tipos son genéricos si no se especifica otra cosa, aunque hay que destacar que ELM no hace posible operar con tipos incorrectos, por lo que una recomendación para este lenguaje es siempre definir muy claramente el tipo (*Introduction · An Introduction to Elm*, 2021).

Para poder transmitir como es ELM es necesario saber que este lenguaje se basa en eventos, donde cada evento se junta en conjuntos de mensajes discretos, que a su vez son definidos en unos tipos de mensajes. Las tres fuentes de eventos que predominan en este lenguaje son las típicas de un lenguaje web: la ejecución de algunos comandos, las acciones que puedan realizar los usuarios en la vista de la web y los eventos a los que están suscritos. Por lo explicado y por concretar un poco más, podemos ver en la siguiente imagen como es la arquitectura en la que se basa ELM:

*Ilustración 25. Arquitectura básica ELM.*




*Fuente: elaboración propia*

En la imagen se puede ver lo explicado en el párrafo anterior, ELM envía HTML para que se muestre por pantalla, el ordenador envía un evento o mensaje para informar de lo que está sucediendo. Esto hace que se garantice la reactividad, debido a que, existe una comunicación bidireccional en cada uno de los programas que se desarrollen con este lenguaje.

Como muchos expertos han definido, el diamante de este lenguaje son los tipos de uniones que ya hemos comentado antes. Al final de toda página suelen aparecer botones para retroceder, poder ir hacia delante o ir a cualquier otra página por un número, y esta funcionalidad en ELM se hace de una forma muy sencilla. Lo primero es definir el tipo de unión que podría hacerse como en la siguiente ilustración:

*Ilustración 26. Definición del tipo de unión en ELM.*

```
type NextPage
  = Prev
  | Next
  | ExactPage Int
```

*Fuente: (Elm — Functional programming, 2016)*

Y esta definición se empezará a utilizar como una variable al hacer el envío de un mensaje como, por ejemplo:

*Ilustración 27. Ejemplo de utilización del tipo de unión en ELM.*

```
type Msg
  = ...
  | ChangePage NextPage
```

*Fuente: (Elm — Functional programming, 2016)*

Para finalizar, se definirá en la función que se encarga de la actualización en cada caso que queramos como se puede ver en la siguiente imagen:

*Ilustración 28. Función actualización ELM.*

```
update msg model =
  case msg of
    ChangePage nextPage ->
      case nextPage of
        Prev ->
          ...
        Next ->
          ...
        ExactPage newPage ->
          ...
```

*Fuente: (Elm — Functional programming, 2016)*

Como hemos visto en 3 sencillos pasos hemos creado un enlace, lo que es mucho más sencillo de programar que en otros lenguajes más antiguos. Por lo que podemos decir que ELM pretende simplificar las funciones más comunes de una aplicación web, para dejar que los programadores se centren más en innovaciones o funcionalidades más complejas o únicas.

### 3.1.9 TypeScript (TypeScript, 2012)

| | |
|---|---|
| Apareció en: | 2012 |
| Diseñador por: | Microsoft |
| Paradigma: | Multiparadigma |
| Influido por: | JavaScript, Java y C++ |
| Sistema operativo: | Multiplataforma |

TypeScript es un lenguaje de programación que vio la luz por primera vez durante octubre de 2012. Microsoft estuvo definiéndolo y desarrollándolo durante dos años, en aquel momento el proyecto se llamó Strada. La primera versión que salió al mercado fue la 0.8 (NubeColectiva, 2020).

Como ellos mismo lo definen, "TypeScript es JavaScript con sintaxis para tipos" (TypeScript, 2021). Es un lenguaje fuertemente tipado, está totalmente basado en JavaScript, es lo que se denomina un superset *(Glosario 4.3)* de este último. Esta última cualidad permite que los programadores que estén realizando proyectos en JavaScript los continúen haciendo con TypeScript sin ningún tipo de problema ni impedimento, por lo que ahora se ha vuelto común ver proyectos que utilizan ambos lenguajes a la par. Prueba de esto último es que equipos como los de Google, Basecamp, Microsoft, Lyft y otras muchas han integrado este lenguaje en sus proyectos, más concretamente en el caso de Google lo utiliza para desarrollar con Angular (Chacón, 2021).

Al igual que JavaScript, este lenguaje sirve para hacer programación móvil, programación web, crear aplicaciones desde el lado del servidor, aplicaciones de escritorio y crear diferentes interfaces entre otras funcionalidades (Chacón, 2021).

El origen de este lenguaje es la necesidad de crear productos escalables donde JavaScript no podía llegar, debido a que cuando nació este lenguaje, JavaScript no soportaba los módulos, ni clases y no incluía herramientas que facilitaran el propio flujo del desarrollo. Lo que Microsoft quería con este nuevo lenguaje era conseguir una mayor escalabilidad, un lenguaje más limpio y sólido (Hernández, 2018).

La principal diferencia que podemos destacar entre JavaScript y TypeScript es que este último agrega una sintaxis adicional para poder permitir una integración mucho más estrecha con el editor. El código escrito en este lenguaje se convierte fácilmente en JavaScript, por lo que se permite ejecutar TypeScript en cualquier lugar donde se pueda ejecutar JavaScript (Node.js, Deno o un navegador) (TypeScript, 2021).

El tipado estático es una de sus características a destacar y otra de sus diferencias con JavaScript. Todas y cada una de sus variables tienen definido un tipo de dato y a estas variables solo se pueden asignar, como es de esperar, valores con el tipo que les corresponde. Esta funcionalidad ayuda al autocompletado de código, con sus posibles recomendaciones de qué argumentos se reciben o que tipo devuelve una función y ayuda a poder hacer un mejor análisis de detección de errores. En la siguiente ilustración podemos ver cómo se crea una variable 'edad' con el tipo *number*, y desde ese momento no se le podrá asignar otro tipo de datos (Chacón, 2021).

*Ilustración 29. Tipado estático de TypeScript.*

```
let edad : number; //Asignamos el tipo number para la variable edad
edad = 20; // La variable ahora sólo puede asignar valores del tipo number
```

*Fuente:(Hernández, 2018).*

Como hemos visto, este lenguaje es de tipado estático, pero existe la palabra reservada *any* que hace que la variable pueda seguir un tipado dinámico. En el siguiente ejemplo vemos como se declara la variable 'otraVariable2' con el tipo *any*, y después se le pone un String.

*Ilustración 30. Tipado dinámico TypeScript.*

```
//Tipado dinámico
let otraVariable2: any;

otraVariable2 = "Profile Software Services";
```

*Fuente: (Chacón, 2021).*

Como en muchos de los lenguajes orientados a objetos, TypeScript permite definir si una variable es pública o privada, por lo que siendo privada no se podrá acceder a ella desde fuera de la propia clase donde se le define. Dentro de estas clases se permite diseñar con el patrón *Decorator*, lo que hace que se puedan añadir nuevas funcionalidades o comportamientos a un objeto de manera más dinámica en tiempo de ejecución colocándolos dentro de otros objetos que los envuelven (Chacón, 2021).

Estas son las características más significativas de TypeScript que, por lo que podemos ver, no son nada nuevo, simplemente incorporaciones que hacen mucho más fácil la tarea del programador. Perfectamente podría haberse sacado una nueva versión de JavaScript e incorporar estas modificaciones, que es lo que muchos expertos aseguran, que TypeScript es simplemente una versión actualizada de su predecesor.

### 3.1.10 Julia (Julia, 2012)

| | |
|---|---|
| Apareció en: | 2012 |
| Diseñador por: | Stefan Karpinski, Alan Edelman, Viral Shah y Jeff Bezanson |
| Paradigma: | Multiparadigma |
| Influido por: | C, Mathematica, MATLAB, Perl, Python, R, Ruby y Lisp. |
| Sistema operativo: | Linux, macOS, Windows y FreeBSD |

El lenguaje de programación Julia fue lanzado al mercado en el 2012, y fue desarrollado por Stefan Karpinski, Alan Edelman, Viral Shah y Jeff Bezanson, investigadores informáticos del instituto *Massachusetts Institute of Technology* (Lauwens & Downey, 2020). Este lenguaje fue creado con el objetivo de penetrar en el mundo de la programación con un alto rendimiento. Más concretamente, hicieron este lenguaje para el cálculo científico, Maching Learning, álgebra lineal, computación distribuida, minería de datos y computación paralela, ámbitos muy demandados por cualquier empresa actualmente. Los repositorios de código abierto se mantienen en la compañía *Julia Computing*, además que ellos mismo se encargan de desarrollar productos más comerciales que ayudan a utilizar e implementar este lenguaje más fácilmente (Tokio School, 2020).

Se trata de un lenguaje dinámico, pero que a su vez presenta las ventajas que proporciona un lenguaje compilado. Esta doble funcionalidad es gracias a que se utiliza un compilador *just in time* (JLT) que está a su vez basado en Low Level Virtual Machine (LLVM), lo que hace posible crear código de máquina que sea totalmente nativo (Tokio School, 2020). Cuenta con una consola para ejecutar comandos read-eval-print-loop (REPL)[16] y una interfaz bastante sencilla que facilita la ejecución de scripts. Con esto último hace posible hacer operaciones poco complejas de una manera más dinámica, utilizando un código rápido y sencillo de interpretar y, por supuesto de escribir (Lauwens & Downey, 2020).

Algunas de las características más destacables de este lenguaje son:

- Mediante la combinación de tipos de argumentos se permite definir cuáles van a ser los comportamientos de las funciones.
- Tipado dinámico.
- Contiene un gestor de paquetes totalmente integrado, para así poder actualizar o eliminar de forma más sencilla los ficheros del software.
- Permite la metaprogramación con algunas herramientas tipo Lisp.

[16] REPL: herramienta que permite una revisión de las expresiones del lenguaje a la hora de procesar un dato.

- Permite llamar a funciones con lenguaje Python gracias al paquete integrado *PyCall,* por lo que para los desarrolladores acostumbrado a programar en Python esto les facilita mucho para poder convertir sus aplicaciones con este nuevo lenguaje

Uno de sus puntos fuertes es la resolución de problemas numéricos complejos, los cuales suelen presentar un gran volumen de datos a procesar (Lauwens & Downey, 2020). Pero lo que realmente ha vuelto conocido en los últimos años a este lenguaje es su utilización para el Maching Learning. Aun siendo un lenguaje relativamente nuevo en el ámbito del análisis de datos, este lenguaje es bastante funcional, gracias a los programadores activos que van compartiendo bibliotecas, haciendo más fácil el uso de este lenguaje con el tiempo. También, para poder facilitar la programación en este sector, Julia incorpora bibliotecas de Fortran y C enfocadas al álgebra lineal. También incorpora una generación de números aleatorios y el procesamiento de cadenas y señales (Tokio School, 2020).

### 3.1.11 Pure Script (PureScript, 2013)

| | |
|---|---|
| Apareció en: | 2013 |
| Diseñador por: | Phil Freeman |
| Paradigma: | Funcional |
| Influido por: | Haskell, JavaScript |
| Sistema operativo: | |

PureScript es un lenguaje que fue desarrollado por Phil Freeman y que apareció en 2013. Presenta características muy similares a Haskell y se compila en JavaScript, esto hace que pueda ser perfectamente ejecutado en un buscador y pueda hacer interacciones con prácticamente cualquier página web.

Es un lenguaje que proporciona una sintaxis mucho más sencilla y ligera que sus predecesores. Está compuesto de un sistema de tipos que es capaz de soportar las abstracciones más potentes, además de ser capaz de generar un código inteligible y rápido. En resumen, PureScript pretende coger todo lo mejor de la programación funcional agregando de JavaScript la programación flexible y rápida.

PureScript contiene un tipado estático, por lo que, los tipos solo aparecen en tiempo de compilación y no en tiempo de ejecución. Estos tipos están inspirados en el lenguaje Haskell. También está soportada la inferencia de tipos, lo que ayuda a hacer muchas menos anotaciones que en otros muchos lenguajes, lo que se convierte en una ventaja en vez de en un ancla para avanzar. En el siguiente ejemplo vemos cómo se define una variable que contiene un número, pero en ningún lado se define el tipo de dicha variable:

*Ilustración 31. Definición variable PureScript.*

```
1 iAmANumber =
2   let square x = x * x
3   in square 42.0
```

*Fuente: (Acereda & Freeman, 2021).*

Ellos mismos, desde su página oficial definen 5 grandes beneficios de utilizar este lenguaje:

- Se puede compilar y se puede reusar el código de JavaScript.
- Consta con un muy buen sistema de herramientas y editor que ayuda a hacer reconstrucciones al momento.
- Consta con una comunidad activa que ayuda a resolver problemas o dudas más rápidamente.
- Es fácil crear aplicaciones mediante técnicas funcionales y tipos expresivos.

Estos puntos reflejan a la perfección que el objetivo perseguido por sus creadores es seguir facilitando el trabajo de un programador, cogiendo lenguajes ya existentes y buscando sus puntos débiles para así mejorarlos y enfocarlos para cubrir más ámbitos donde aplicar el lenguaje.

### 3.1.12 Crystal (Crystal, 2014)

| | |
|---|---|
| Apareció en: | 2014 |
| Diseñador por: | Ary Borenszweig, Juan Wanerman y Brian Cardiff |
| Paradigma: | Multiparadigma: orientado a objetos y concurrente |
| Influido por: | Ruby, C, Rust, Go, C# y Python |
| Sistema operativo: | Linux, MacOs, Windows |

Crystal es un lenguaje de programación que empezó a desarrollarse en junio del 2011. El objetivo principal que se propusieron sus desarrolladores era juntar la productividad de Ruby con la eficiencia, seguridad y velocidad que proporcionan lenguajes compilados como C.

Este lenguaje utiliza un sistema estático con inferencia, una integración fácil con el lenguaje C, permite generar y evaluar código en el mismo tiempo de ejecución y, también, permite hacer

un código nativo totalmente eficiente. Vamos a ver estos puntos con más detalle (Deusto Formación, 2019):

- Sintaxis parecida a Ruby. Cualquier informático que haya estado trabajando con Ruby al utilizar este lenguaje va a sentir que está programando en un lenguaje totalmente conocido. Aunque cabe destacar que Crystal también tiene otras características que no posee su predecesor, como puede ser el paralelismo y la concurrencia. Estos dos conceptos los han incluido por creer que es de las pocas cosas que le faltaban a Ruby para ser un lenguaje mucho más eficiente.
- Tipado estático por inferencia. Como ya hemos comentado anteriormente, el tipado estático es aquel que comprueba los tipos durante la compilación y no durante la ejecución. Esto hace que la ejecución sea mucho más eficiente y segura. Lo que aporta la inferencia es que los tipos se puedan asignar de manera automática sin que el desarrollador tenga que especificarlo.
- Integración con C. Crystal hace muy sencillo poder utilizar las bibliotecas de C, y su propia documentación explica cómo hacer esto. Además, no es que Crystal pueda llamar al código C, sino que también el código C puede ejecutar el código escrito en Crystal.
- Generar y evaluar en tiempos de compilación. Ruby no es un lenguaje compilado, y esta es una de las características que incorpora Crystal, por lo que su ejecución se vuelve mucho más rápida. Esto también implica que sea en tiempo de compilación donde se buscan los errores, para así poder generar un código óptimo. También se incorpora la posibilidad de ejecutar el código de forma directa, lo que hace que sea útil depurar el código, pero no resulta un código tan bueno.
- Código nativo eficiente. Crystal es un lenguaje que corre con la LLVM (Low Level Virtual Machine) (tecnología que es utilizada como framework de compilación), lo que ayuda a poder generar código máquina de forma similar a como lo hace el lenguaje C. La utilización de LLVM ayuda a que el proceso de generar el código y su compilación sea mucho más ágil y eficiente.

### 3.1.13 Hack (HACK, 2014)

| | |
|---|---|
| Apareció en: | 2014 |
| Diseñador por: | Marcos de la Cruz, David Carvajal, Alok Menghrajani, Drew Paroski |
| Paradigma: | Multiparadigma: concurrente, orientado a objetos, funcional e imperativo |
| Influido por: | PHP, Java y C# |
| Sistema operativo: | Multiplataforma |

Hack es un lenguaje de programación desarrollado por Facebook. Su principal finalidad ha sido ayudar a los programadores de esta compañía a poder actualizar el código PHP que estaban utilizando. Hack lo definen ellos mismos como una combinación de C++, C y funciones de PHP,

que hace que sea un lenguaje muy completo y sencillo de aprender. También afirman que Hack se puede considerar como una versión actualizada de PHP, pero con la diferencia de que se necesita de la máquina virtual HHVM para poder llegar a ejecutar los programas desarrollados en este lenguaje (Crespo, 2014).

El tiempo que se tardaba con PHP en encontrar los errores fue determinante para que el equipo de Facebook se pusiera manos a la obra para desarrollar este nuevo lenguaje. Por esto, a parte de la rapidez, este lenguaje incorpora una mejor depuración que resulta muy sencilla (Crespo, 2014).

Un ejemplo de la similitud entre PHP y Hack se puede apreciar en la siguiente imagen, donde los dos códigos hacen la misma función. La primera diferencia es que un archivo PHP empieza con <?php y un archivo Hack empieza por <?hh.

*Ilustración 32. Comparación PHP y HACK.*

```php
<?php

function dbfetch() {
  $r = mysql_query('SELECT ...');
  if ($r === false || mysql_num_rows($r) == 0) {
    return null;
  }

  return new DBData(mysql_fetch_array($r));
}

function test() {
  $data = dbfetch();
  $data->doFunStuff();
}
```

```hack
<?hh

function dbfetch(): ?DBData {
  $r = mysql_query('SELECT ...');
  if ($r === false || mysql_num_rows($r) == 0) {
    return null;
  }

  return new DBData(mysql_fetch_array($r));
}

function test(): void {
  $data = dbfetch();
  $data->doFunStuff();
}
```

*Fuente: (Crespo, 2014)*

Por el contrario, aunque PHP y Hack sean muy similares, hay una diferencia bastante importante si el desarrollador estaba acostumbrado a utilizar PHP y HTML en el mismo código. Esta funcionalidad de combinación con código HTML no la ofrece Hack, por lo que será un gran punto decisorio para los programadores que contemplen empezar a usar este nuevo lenguaje (HACK, 2014).

Las funciones en Hack permiten especificar los tipos de variables de retorno y para los argumentos, es decir, las funciones en Hack se declaran con tipos. Este hecho lo podemos observar en la siguiente imagen, donde se declara una función que simplemente devuelve el contrario de lo que se le pasa por parámetro (HACK, 2014).

*Ilustración 33. Funciones en Hack.*

```
<?hh
// Las funciones en Hack se declaran con tipos.
function negar(bool $x): bool {
    return !$x;
}
```

*Fuente:* (HACK, 2014)

En la propia página de Hack podemos ver dos características principales de este lenguaje. El desarrollo rápido, ya comentado, y la comprobación de tipo instantánea. Esta última característica se refiere a la verificación de manera incremental de los archivos a medida que los vas editando. También consta con la característica de ser asíncrono, lo que significa que permite la multitarea cooperativa, debido a que permite la combinación de solicitudes de entrada y salida. Incluye un amplio conjunto de tipos genéricos y una salida muy familiar a la que proporciona XML, por lo que se hace más cómoda su interpretación (HACK, 2021).

### 3.1.14 Swift (Apple, 2014)

| | |
|---|---|
| Apareció en: | 2014 |
| Diseñador por: | Apple |
| Paradigma: | Multiparadigma |
| Influido por: | C#, Objective-C, Python, Rust, Ruby y Haskell |
| Sistema operativo: | MacOS, iOS, iPadOS, tvOS, watchOS, FreeBDS, GNU/Linux y Windows |

Swift es un lenguaje de programación lanzado durante el 2014 que ha sido desarrollado por Apple con la finalidad de poder realizar apps concretamente para sus dispositivos (IOS, MAC, Apple TV y Apple Watch). Es un lenguaje calificado como multiparadigma.

Se dice que es el sucesor del anterior lenguaje de Apple, Objetvie-C, que hasta el momento que apareció Swift era el lenguaje utilizado para desarrollar sus aplicaciones. Ellos mismos definieron este lenguaje en su presentación como "más *sencillo, intuitivo y potente que Objetcive-C*" (Apple, 2021). Swift resalta por ser un lenguaje conciso y claro, lo que ayuda a reducir en gran medida los errores por tener una sintaxis mucho más entendible para los desarrolladores. Volvemos a la tendencia de simplificar las cosas para el desarrollador.

Es un lenguaje de código abierto, por lo que cualquiera puede desarrollar con Swift, no solo lo pueden utilizar los propios desarrolladores de Apple. Lo que la mayoría de los usuarios destacan de este lenguaje es su rapidez, igualándolo con el lenguaje C. También permite encontrar errores mucho más rápido que otros lenguajes, por lo que se evitan la aparición de bugs[17] que pueda tener la aplicación. Esto último va enlazado a que Swift es un lenguaje que se ha pensado para usar también en la nube, por lo que algo que tenían claro que iba a ser imprescindible es que facilitara la estabilidad y la seguridad del servicio (Apple, 2021).

Es calificado como sencillo de aprender, por contar con una comunidad muy grande de usuarios que van colgando consejos y librerías, y contar con un libro que con cada actualización del lenguaje va publicando Apple para una mayor facilidad de aprendizaje de los desarrolladores (Apple, 2021).

Es un lenguaje fuertemente tipado, sin embargo, cabe destacar que gracias a su inferencia de tipos no siempre es necesario declararlos. Tiene una división de los tipos de datos:

- Tipo de referencia: utilizado cuando se comparten datos mutables. Cuando se asigna un tipo de estos se consigue que al modificar una instancia se pueda ver reflejado el cambio en todas las constantes y/o variables que la compartan.
- Tipo de valor: utilizado para cuando se trabaja con diversos hilos, ya que, al usar este tipo de archivo guarda el contenido en una copia.

Por lo que vemos el lanzamiento de este lenguaje ha incorporado al viejo lenguaje Objective-C algunas funcionalidades diferenciadoras para poder facilitar el trabajo, agilizarlo y hacerlo mucho más seguro.

### 3.1.15 Raku (Raku, 2015)

| | |
|---|---|
| Apareció en: | 2015 |
| Diseñador por: | Larry Wall |
| Paradigma: | Multiparadigma |
| Influido por: | Perl, Python, Haskell y Ruby |
| Sistema operativo: | Multiplataforma |

Raku es un lenguaje de programación que pertenece a la familia de los lenguajes de programación de Perl, su primer nombre fue Perl6, pero al incorporar características tan distantes de Perl5 se decidió llamarlo Raku. Este lenguaje fue anunciado en el 2015 en la

[17] Bug: Defecto o error en el software.

conferencia de Perl de ese mismo año. Como ellos mismo definieron pretendían eliminar las "verrugas" del lenguaje. Lo que querían eran mantener las cosas fáciles y convertir las cosas difíciles en más sencillas para el programador. Es un lenguaje de alto nivel, con un tipado gradual[18]. Es otro lenguaje catalogado como multiparadigma, ya que, soporta programación orientada a objetos y funcional (Guía Raku, 2021).

Entre las "verrugas" que querían eliminar encontramos la confusión que había en el uso de sigilos[19] para los contenedores y, por otro lado, la ambigüedad que se producía entre las funciones y el impacto sintáctico que tienen los identificadores de los archivos que no tienen formato. Uno de los problemas que mantiene de su anterior versión Perl 5 es que proporciona tal cantidad de funciones que resulta muy difícil aprender todas ellas (Guía Raku, 2021). Además de que muchas de estas funciones pueden ser bastante parecidas, por lo que dificultan la elección de la que mejor se adapta a la solución buscada.

Las principales características que podrían definir a este lenguaje serían (Preguntadores.net, 2020):

- El tipado gradual que hemos comentado anteriormente, similar a TypeScript o Julia.
- De la misma forma que Haskell, Raku permite definir diversas declaraciones de funciones para diversos valores de entrada, lo que se llama coincidencia de patrones.
- Permite definir diversos cuerpos de funciones para cuando los tipos de argumentos de entrada son diferentes. Esta característica es similar a la coincidencia de patrones, pero en este caso hace una diferenciación sobre el tipo y no sobre el número de variables.

También hay que destacar que Raku diferencia entre funciones y mutadores. Las funciones no cambian el estado del objeto que se pasa, mientras que los mutadores sí modifican el estado. En el siguiente ejemplo vemos cómo se declara un array de números, y posteriormente se ejecuta el mutador *push* y la función *short. Push* añade un número más, por lo que modifica el estado del array, y la función *short* únicamente ordena la secuencia de números (Guía Raku, 2021).

[18] Tipado gradual: sistema de tipos que permite dar tipo a algunas variables y a otras no. Cuando se le asigna tipo se corregirá en tiempo de compilación, y cuando no se le asigne tipo se le asignará en tiempo de ejecución (Swamy et al., 2014).

[19] Sigilo: símbolo que se añade al nombre de las variables, que suele mostrar el tipo de alcance o datos de dicha variable. Normalmente es el prefijo '$'

*Ilustración 34. Funciones y mutadores en Raku.*

Script

```
my @números = [7,2,4,9,11,3];

@números.push(99);
say @números;       #1

say @números.sort; #2
say @números;       #3

@números.=sort;
say @números;       #4
```

Salida

```
[7 2 4 9 11 3 99] #1
(2 3 4 7 9 11 99) #2
[7 2 4 9 11 3 99] #3
[2 3 4 7 9 11 99] #4
```

*Fuente: (Guía Raku, 2021)*

### 3.1.4 Kotlin (Kotlin, 2016)

| | |
|---|---|
| Apareció en: | 2016 |
| Diseñador por: | Graydon Hoare |
| Paradigma: | Orientado a objetos |
| Influido por: | Java, Scala, Groovy, C# y Gosu. |
| Ha influido a: | C#, Elm, Idris, Swift |
| Sistema operativo: | Cualquiera que soporte la JVM o tenga intérprete de JavaScript |

Kotlin es un lenguaje de programación orientado a objetos que ha sido desarrollado por JetBrains. Su nombre hace referencia a una isla que se encuentra cercana a San Petersburgo, que es donde se desarrolló dicho lenguaje. Fue durante el 2011 cuando la empresa anunció este nuevo lenguaje de programación después de un largo año de desarrollo, aunque no fue hasta el 2016 cuando se liberó el código fuente. En la presentación de este lenguaje, Dimitry Jemerov (líder de JetBrains), puso hincapié en que la creación de este lenguaje se debía a que los otros que ya estaban en el mercado no tenían las características que ellos buscaban y necesitaban. El

único lenguaje que podía acercarse a las prestaciones que ellos buscaban era 'Scala', pero que aun así destacaba por su lento tiempo de compilación. Por esto último, definieron como objetivo prioritario conseguir que Kotlin tuviera las prestaciones de Scala y que compilara tan rápido como Java. (EcuRed, s.f)

Kotlin es fuertemente tipado y de tipado estático. Se ejecuta en la máquina virtual de Java (JVM), y a su vez, puede ser compilado a código de JavaScript. Permite hacer proyectos mixtos mezclando código en Kotlin y en Java a la vez. Al generar código en Java es totalmente compatible con Android, y actualmente existe el proyecto Kotlin/Native que hace que sea posible generar ejecutables para escritorio y para IOS, algo que hace que pueda ser altamente utilizado para cualquier dispositivo móvil (Kotlin, s.f).

Centrándonos en la sintaxis de Kotlin encontramos que la declaración de listas de parámetros y variables se hace poniendo primero el identificador seguido de dos puntos y después el tipo de dato, al igual que en lenguajes como Pascal, Haxe, Go y Scala (lenguajes que son bastante conocidos).

*Ilustración 35. Ejemplo de declaración de variables en Kotlin.*

```
val xPos: Int = 1 // Asignación junto a declaración
val yPos: Int   // Declaración
yPos = 5    // Asignación
```

*Fuente: (EcuRed, s.f)*

En la misma línea que el lenguaje 'Scala' los puntos y comas como final de sentencia son totalmente opcionales, simplemente con un salto de línea el compilador ya entiende que la declaración ha terminado. En el ejemplo de la ilustración 11 podemos apreciar este hecho. Por otro lado, ningún tipo en Kotlin se puede hacer nulo, lo que tiene es estructuras específicas para trabajar con dichos nulos.

*Ilustración 36. Ejemplo de tipo de nulos en Kotlin.*

```
var a:String = "" a = null; // No se permite var b:String? = "" b = null; // Permitido
```

*Fuente: (Kotlin, s.f)*

En este lenguaje no existe el operador ternario, más comúnmente conocido como condicional, pero si existen los ifs, en la ilustración 13 se puede ver un ejemplo de este tipo de estructuras.

*Ilustración 37. Ejemplo de estructura 'IF' en Kotlin.*

```
fun clamp(v:Int, min:Int, max:Int) = if (v < min) min else if (v > max) max else v
```

*Fuente: (Kotlin, s.f)*

En este lenguaje no hay distinción entre clases y tipos primitivos, una de las consecuencias de este hecho es que hasta los tipos primitivos empiezan por mayúscula (Char, Long, Int, Double, Boolean, Float, Byte...).

Al utilizar el lenguaje Scala como base es de suponer que las declaraciones sean muy parecidas o prácticamente iguales. Se utiliza 'Var' para las declaraciones mutables, 'Fun' para las funciones y 'Val' para las declaraciones que son inmutables.

*Ilustración 38. Ejemplo de declaraciones en Kotlin.*

```
var mutable = 1 val immutable = 2 fun myfunc() { }
```

*Fuente: (Kotlin, s.f)*

Otra de las facilidades que presenta es que no hace falta utilizar la palabra reservada 'return', simplemente poniendo un '=' bastaría. Tampoco son necesarias las llaves { } en caso de funciones simples como la de la ilustración 14.

*Ilustración 39. Ejemplo de función simple en Kotlin.*

```
fun sum(a:Int, b:Int) = a + b
```

*Fuente: (Kotlin, s.f)*

Una de las diferencias con Scala es que en este lenguaje sí tenemos la presencia de las funciones 'get' y 'set'. En la siguiente ilustración se muestra un ejemplo de cómo sería la función 'get' y 'set' de la variable *myValue*.

*Ilustración 40. Ejemplo de las funciones 'get' y 'set' de Kotlin.*

```
private var myValue:Int = 1
val immutableDouble:Int get() = myValue * 2
var double:Int
    get() = myValue * 2
    set(value) { myValue = value / 2 }
```

*Fuente: (Kotlin, s.f)*

Además de funciones miembros (clases y métodos), Kotlin soporta el uso de funciones y programación por procedimientos. Igual que en muchos otros lenguajes, el punto de entrada a cualquier programa desarrollado en Kotlin es una función a la que llamamos 'main'. Esta función recibirá un array donde estarán los argumentos que se han pasado desde la línea de comandos, al igual que pasa en C y C++. Por último, hay que destacar que Kotlin soporta la interpolación de variables dentro de las propias cadenas de texto, al igual que lo hacen los Shell scripts de Perl y Linux/Unix.

Como hemos podido ver, aunque Kotlin se basa principalmente en Scala no deja de copiar algunas de las características más importantes de lenguajes tan famosos como Java, C o C++. Este lenguaje no es más que otro de los muchos que pretende hacer más fácil la programación, agrupando las características que han considerado mejores de cada uno de los lenguajes ya existentes para formar este nuevo.

### 3.1.16 Ballerina (Ballerina, 2017a)

| | |
|---|---|
| Apareció en: | 2017 |
| Diseñador por: | WSO2 |
| Paradigma: | Multiparadigma |
| Influido por: | Java, JavaScript, Go, Rust y C# |
| Sistema operativo: | Multiplataforma |

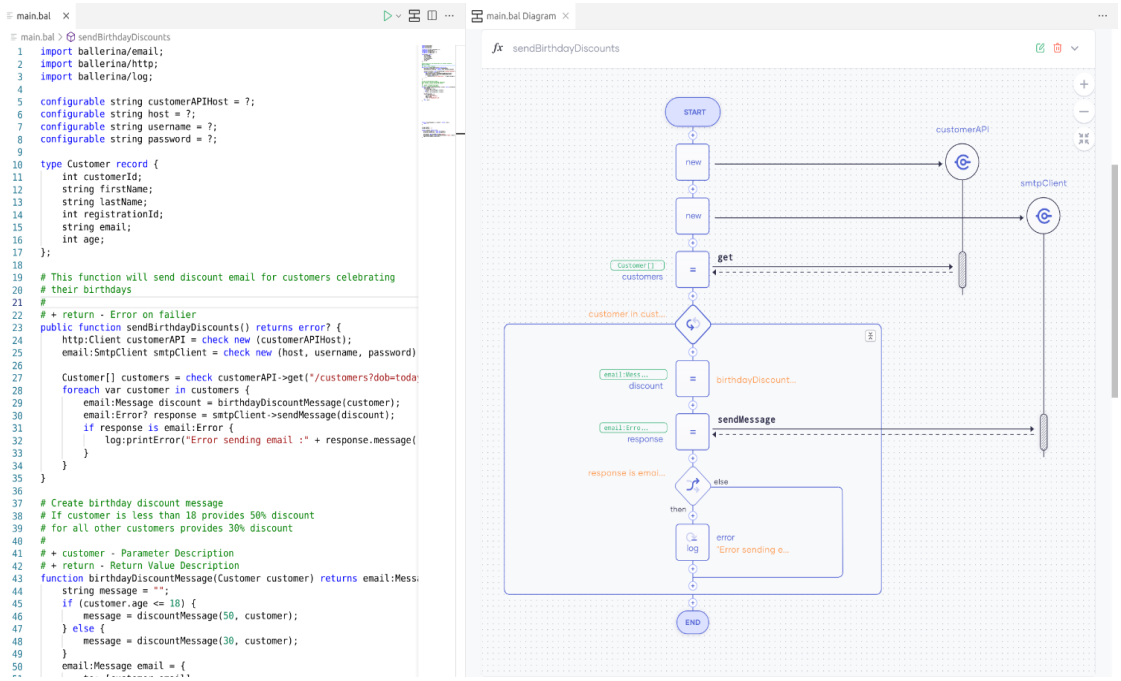
Ballerina es un lenguaje de programación que ha sido diseñado por WSO2, enfocado en aplicaciones de la nube, más concretamente para computación distribuida[20], microservicios y

[20] Sistema distribuido: sistema que tiene sus componentes situados en diferentes computadores en red. Se suelen comunicar mediante mensajes.

APIs[21]. Por esto último sabemos que ha sido creado debido al incremento en el uso de servicios digitales, los dispositivos conectados y las aplicaciones móviles. Fue en 2017 cuando se anunció públicamente que se empezaba a desarrollar este nuevo lenguaje, pero no fue hasta 2019 que se lanzó su primera versión (Ballerina, 2017).

Este lenguaje es considerado como de propósito general, por lo que incorpora una sintaxis que resulta familiar. El código se representa de una manera gráfica en forma de unos diagramas de secuencia, es decir, que cada uno de los programas escritos en Ballerina tiene un diagrama de eventos que te da una vista rápida de los procesos que están involucrados y cuál es la secuencia de mensajes entre estos procesos. En la siguiente imagen vemos un ejemplo de este diagrama de secuencia junto con su código escrito en Ballerina (Ballerina, 2017).

*Ilustración 41. Diagrama de Ballerina.*




*Fuente: (Ballerina, 2017)*

Al poder tener esta visión del código junto al diagrama hace que sea mucho más sencillo entender el programa que se ha desarrollado, hasta para alguien que no ha estado presente en el proceso de desarrollo. También cuenta con un manejo explícito de los errores, y esto mismo combinado con el sistema de tipos estático y la seguridad que ofrece en la concurrencia hace que las aplicaciones o programas escritos con Ballerina sean bastante fáciles y confiables de mantener (Ballerina, 2017).

[21] API (Applicaton Programming Interfaces): es un conjunto de protocolos y definiciones que es utilizado para poder integrar y desarrollar el software de las aplicaciones, lo que permite una comunicación eficaz entre dos aplicaciones.

Desde un principio este lenguaje fue creado para poder ser capaz de soportar lenguajes sencillos y también de los más complejos y potentes. Este lenguaje facilita mucho las cosas debido a que cuenta con un soporte nativo para Swaggeer, REST, CML y JSON. Otra de las cosas que lo diferencian es que hace sencillo el poder conectarse con redes sociales tan conocidas como son Twitter y Facebook, funcionalidad imprescindible en el mundo que nos encontramos, donde todo se mueve a través de las redes sociales.

### 3.1.17 Bosque (Microsoft, 2019)

| | |
|---|---|
| Apareció en: | 2019 |
| Diseñador por: | Microsoft |
| Paradigma: | Multiparadigma |
| Influido por: | JavaScript, TypeScript y ML |
| Sistema operativo: | Multiplataforma |

Bosque es un lenguaje que fue desarrollado por Microsoft, y como ellos mismos lo definieron es un lenguaje que quiere ir más allá de la programación estructurada[22]. Este lenguaje permite a los desarrolladores poder organizar su código en bloques y composiciones naturales, a la vez que se benefician de la simplicidad y corrección típica de la programación funcional.

Este lenguaje resulta muy versátil en su llamada a funciones. En la siguiente imagen podemos ver cómo una misma función que suma dos números, se puede llamar de diferentes formas. Tiene una gran versatilidad a la hora de escribir código.

*Ilustración 42.Función suma en Bosque.*

```
function add2(x: Nat, y: Nat): Nat {
    return x + y;
}

add2(2, 3)     //5
add2(x=2, y=3) //5
add2(y=2, 5)   //7
```

*Fuente: (Marron, 2021)*

[22] Programación estructurada: teoría que está orientada a mejorar la calidad, claridad y tiempo de desarrollo mediante la utilización de funciones o subrutinas.

Bosque es un lenguaje que se basa en los tipos de datos y sintaxis de TypeScript, otro de los lenguajes que ha desarrollado Microsoft a lo largo de la historia. Como ya hemos comentado antes, este lenguaje pretende eliminar algunas complejidades de la programación más clásica utilizando bucles y llamadas a subrutinas, lo que resulta eficiente, pero a veces muy engorroso de entender lo que el código está haciendo si no es el propio desarrollador el que lee el código. Bosque elimina el uso de bucles, y lo que hace es crear una función llamada *Functor*, que es la que replica la tarea del bucle, pero de una forma mucho más sencilla de crear y entender en el código. En la siguiente imagen se puede ver una comparativa entre un bucle en JavaScript y un bucle en Bosque.

*Ilustración 43. Bucle en Bosque*

```
//Imperative Loop (JavaScript)
var: number[] a = [...];
var: number[] b = [];
/**
Pre: b.length == 0
**/
for(var i = 0; i < a.length; ++i) {
    /**
    Inv: b.length == i /\
         forall i in [0, i) b[i] == a[i]*2
    **/
  b.push(a[i]*2);
}
/**
Post: b.length == a.length /\
      forall i in [0, a.length) b[i] == a[i]*2
**/
```

```
//Functor (Bosque)
var a = List[Int]@{...};
/**
Pre: true
**/
var b = a.map[Int](fn(x) => x*2);
/**
Post: List[Int]::eq(fn(x, y) => y == x*2, a, b)
**/
```

*Fuente: (Locura informática digital, 2021)*

Como vemos las líneas se reducen en gran medida comparado con un bucle convencional, y esto es solo un ejemplo de lo más sencillo posible. En código de millones de líneas esto puede suponer un ahorro de más de la mitad de éstas, lo que hace más fácil poder ir al grano y llegar a lo importante de un vistazo más rápido.

## 3.2 Resumen de los lenguajes y sus características.

A continuación, podemos ver una tabla resumen de los lenguajes analizados y los conceptos que previamente ya habíamos definido. Cabe destacar que en la columna de 'Tipo de programación' indicamos en algunos casos 'Multipropósito', lo que significa que están pensados para diferentes tipos de programación y no para uno únicamente. Los demás casos, aunque ponga un tipo concretamente también puede adaptarse a otros, aunque no es lo más habitual. Se ha omitido

una columna indicando el nivel de máquina al que pertenecen estos lenguajes, debido a que, todos pertenecen a 'Alto nivel'.

*Tabla 1. Resumen de los lenguajes analizados.*

| Lenguaje de programación | Paradigma | Tipado | Chequeo de tipificado | Tipo de programación |
|---|---|---|---|---|
| Rust | Multiparadigma | Inferencia de tipos | Estático (compilación) | Multipropósito |
| Ceylon | Orientado a objetos | Fuertemente tipado | Estático (compilación) | Multipropósito |
| DART | Orientado a objetos | Inferencia de tipos | Estático y dinámico (elige le programador) | Programación web |
| Kotlin | Orientado a objetos | Fuertemente tipado | Estático (compilación) | Multipropósito |
| Red | Multiparadigma | Inferencia de tipos | Dinámico (ejecución) | Programación de SO |
| OPA | Multiparadigma | Fuertemente tipado | Estática (compilación) | Programación web |
| Elixir | Multiparadigma | Inferencia de tipos | Dinámico (ejecución) | Multipropósito |
| Elm | Funcional | Fuertemente tipado | Estática (compilación) | Programación web |
| TypeScript | Multiparadigma | Fuertemente tipado | Estática (compilación) | Programación web |
| Julia | Multiparadigma | Inferencia de tipos | Dinámico (ejecución) | Programación de paquetes |
| PureScript | Funcional | Inferencia de tipos | Estático y dinámico (elige le programador) | Multipropósito |
| Crystal | Multiparadigma | Inferencia de tipos | Estática (compilación) | Multipropósito |
| Hack | Multiparadigma | Inferencia de tipos | Estático y dinámico (elige le programador) | Programación web |
| Swift | Multiparadigma | Inferencia de tipos | Estático | Multipropósito |
| Raku | Multiparadigma | Inferencia de tipos | Estático y dinámico (elige le programador) | Multipropósito |
| Ballerina | Multiparadigma | Fuertemente tipado | Estática (compilación) | Programación en la nube |
| Bosque | Multiparadigma | Inferencia de tipos | Estática (compilación) | Multipropósito |

Como resumen general de esta tabla podemos ver que la mayoría son multipropósito, facilitando así que con un único lenguaje pueda cubrir diversos ámbitos. No se enfocan en un solo tipo de programación, queda a disposición del desarrollador, sabiendo las características

que tiene el poder elegir cuál es el que mejor se adapta a su propósito. También vemos que la mayoría son multiparadigma, lo que confirma que estos nuevos lenguajes son solo combinación de lo ya existente, eligiendo así las mejores funcionalidades de cada paradigma según sus desarrolladores. Esto confirma la hipótesis de Alan Key de que los lenguajes que se fueran a crear en las siguientes décadas no aportarían realmente características novedosas, sino que presentarían una combinación de aspectos ya existentes. Por último, podemos ver cómo buscan facilitar su uso al usuario, por eso abunda el uso de inferencia de tipos, para que el programador se despreocupe de tener que ir especificando los tipos de datos, y así, según el uso que se le da a la variable el programa le asignará un tipo u otro.

## 3.3 Uso de los lenguajes estudiados en la actualidad.

Como ya hemos ido comentando a lo largo de este Trabajo de Fin de Grado, la tecnología ha ido evolucionando día a día, por lo que se han ido desarrollando dispositivos nuevos o actualizaciones de los ya existentes. Con la finalidad de que estos dispositivos funcionen es necesario el desarrollo de aplicaciones, las cuales se desarrollan mediante los lenguajes de programación. Estos nuevos lenguajes que han ido apareciendo y aparecerán tiene como principal finalidad lograr que la comunicación entre el hombre y la máquina sea lo más sencilla posible.

Cada año diversas compañías se encargan de hacer un ranking de los lenguajes más utilizados basándose en diferentes métricas. Una de estas métricas es las veces que se ha utilizado el lenguaje y el propio contexto donde ha sido aplicado. También se tiene muy en cuenta las ofertas de trabajo que se publican, es decir, si un año salen muchas ofertas para programadores que conozcan JAVA, este lenguaje en el ranking tendrá bastante peso.

TIOBE es una de estas organizaciones que cada mes publica una lista con los lenguajes que consideran más populares. Ellos mismos indican que “no se trata del mejor lenguaje de programación o del lenguaje en el que se han escrito la mayoría de las líneas de código” (TIOBE, 2021), pero sí aseguran que por todas las métricas a las que ellos prestan atención deben estar esos lenguajes en el ranking. Por eso, el ranking de TIOBE se define como un indicador de popularidad, y sus calificaciones se basan en los lenguajes que utilizan los ingenieros que están calificados, los proveedores más reconocidos y los cursos de empresas reconocidas (TIOBE a, 2021).

El último informe publicado en su página oficial es sobre julio de 2022. En esta nos muestra que los 4 lenguajes principales son Python, C, Java y C++, clásicos entre los lenguajes de programación. Solo estos 4 ya ocupan el 40% de la cuota de mercado. En este último informe también destacan que lenguajes más nuevos como son Rust, Dart, Kotlin o TypeScript, que han sido objeto de análisis en este Trabajo de fin de Grado, se están acercando con gran fuerza al top 20 (TIOBE a, 2021).

Tabla 2. Top 20 TIOBE

| Julio 2022 | Julio 2021 | Lenguaje de programación | Calificación |
|---|---|---|---|
| 1 | 3 | Python | 13,44% |
| 2 | 1 | C | 13,13% |
| 3 | 2 | Java | 11,59% |
| 4 | 4 | C++ | 10,00% |
| 5 | 5 | C# | 5,65% |
| 6 | 6 | Visual Basic | 4,97% |
| 7 | 7 | JavaScript | 1,78% |
| 8 | 9 | Assembly language | 1,65% |
| 9 | 10 | SQL | 1,64% |
| 10 | 16 | Swift | 1,27% |
| 11 | 8 | PHP | 1,20% |
| 12 | 13 | Go | 1,14% |
| 13 | 11 | Classic Visual Basic | 1,07% |
| 14 | 20 | Delphi/ Object Pascal | 1,06% |
| 15 | 17 | Ruby | 0,99% |
| 16 | 21 | Objective-C | 0,94% |
| 17 | 18 | Perl | 0,78% |
| 18 | 14 | Fortran | 0,76% |
| 19 | 12 | R | 0,76% |
| 20 | 19 | MATLAB | 0,73% |

*Fuente: (TIOBE a, 2021)*

De los lenguajes analizados solo encontramos a Swift en el top 20, más concretamente en la posición 10, lo que es una posición bastante alta para un lenguaje tan reciente. En la siguiente tabla podemos ver cómo algunos de los lenguajes objeto de estudio de este trabajo ya van apareciendo por el top 50 realizado por TIOBE.

*Tabla 3. Top 50 TIOBE.*

| Julio 2022 | Lenguaje de programación | Calificación |
|---|---|---|
| 27 | Julia | 0,49% |
| 29 | Rust | 0,42% |
| 32 | Dart | 0,31% |
| 35 | Kotlin | 0,28% |
| 37 | TypeScript | 0,24% |

Fuente: (TIOBE a, 2021)

# CAPÍTULO 4: CONCLUSIONES

## 4.1 Conclusiones

El número de lenguajes de programación que existen hoy en día en el mercado no deja de aumentar, lo que hace que sea cada vez más complicado para los desarrolladores elegir cuál es el lenguaje más adecuado para su aplicación o proyecto. Por lo que hemos ido viendo durante el transcurso de este trabajo, la mayoría de los lenguajes que están apareciendo se basan en uno ya existente, o en varios, para poder mejorarlo y hacerlo más sencillo. Esto último hace que sean totalmente competitivos y vayan dejando atrás a sus predecesores.

La combinación de diferentes paradigmas de programación es otra de las características que hemos observado en los lenguajes estudiados y que los hace realmente útiles, debido a que, un mismo lenguaje puede ser utilizado en diferentes ámbitos, por lo que ya no será necesario aprender diferentes lenguajes para tener control sobre diferentes ámbitos. Ahora simplemente con uno hay acceso a diferentes paradigmas de programación.

Para los desarrolladores más clásicos, estos lenguajes también han resultado ser una gran revolución, debido a que, al incorporar características de los más antiguos, no hace falta aprenderlos desde cero. Se puede desarrollar con los nuevos lenguajes reutilizando mucho de su código y sintaxis, ya que nos van a resultar muy familiar. Modernidad y sencillez es lo que hace destacar a estas nuevas generaciones de lenguajes.

Hay que mencionar que en el mundo en el que vivimos actualmente se necesita cada vez aplicaciones más potentes y actualizadas. Nuestro día a día está repleto de aplicaciones o servicios que requieren de un lenguaje para poder ser programados, y que cumplan con nuestros objetivos. Para poder mantener estos programas o servicios se necesita de equipos de desarrollo muy grandes, donde los miembros pueden no encontrarse en el mismo lugar geográfico y que los más común es que vayan entrando nuevos integrantes. Por esto último es necesario que los lenguajes que vayan apareciendo, si quieren sobrevivir, tienen que hacer la comprensión del código lo más sencilla posible para que alguien que empieza de cero, en un proyecto ya creado, no se sienta incapacitado y pueda integrarse en el grupo de trabajo lo antes posible.

Hay muchas empresas que se encargan de hacer un seguimiento de estos nuevos lenguajes para poder ofrecer a sus clientes la mejor solución posible. Una de estas empresas es Softteck que hace un especial énfasis sobre lenguajes como son Rust, Kotlin, Golang (Go) y Swift. Esta empresa en concreto se dedica a facilitar soluciones y servicios de transformación digital, y en su informe suele hacer una reflexión sobre los lenguajes que más aparecen en las búsquedas o más se utilizan en las empresas. En este informe que se llama '*Next-Gen Software: Languages & Tools'* destacan que, como el sector de las tecnologías está en auge y cada vez se demandan más programadores o desarrolladores, el contar con cada vez más lenguajes que hacen sencillo el aprendizaje y su programación facilita mucho las cosas. Antes se buscaban personas especialistas, debido a que no era sencillo que en poco tiempo una persona pudiera programar

con gran soltura, pero con la llegada de las nuevas generaciones esto ha cambiado, y ha supuesto una gran revolución en todo el sector (Sofftek, 2021).

Por todo esto descrito podemos afirmar que no se está desarrollando ningún tipo de programación diferente de la que ya existía, simplemente se está haciendo un gran esfuerzo en mejorar lo que ya existe, haciendo mezcla de las características que más convienen basándose en los tipos de dispositivos y sus requerimientos. Combinación, sencillez y adaptación a las nuevas tecnologías son las tres características que mejor podrían definir estas nuevas generaciones de lenguajes que están surgiendo, y que gracias a estas características están ganando terreno a sus predecesores.

## 4.2 Limitaciones del trabajo

La primera limitación de este trabajo es el tiempo, debido a que, se han realizado diversas tareas de recopilación de información en un periodo determinado, por lo que no se va a mantener actualizado durante el tiempo. Los lenguajes de programación están vivos, por lo que pasado un tiempo las características definidas en este trabajo puede que ya no concuerden al cien por cien con la realidad. La segunda limitación encontrada a la hora de hacer este trabajo es que resulta muy complicado analizar diecisiete lenguajes de programación por completo, por lo que en este trabajo solo se han recogido las ideas más importantes y significativas que se han considerado relevantes. Este es el motivo, por el que en la mayoría de los lenguajes se haya adjuntado un enlace a sus páginas principales, para que aquel que pueda estar interesado en continuar informándose tenga la posibilidad de hacerlo. La tercera limitación que encontramos es que se necesita utilizar un lenguaje muy específico, por lo que muchas veces se ha recurrido a utilizar pies de página para poder hacer una definición breve del concepto utilizado en la explicación, esto hace que pueda ser más tediosa la lectura de este Trabajo de Fin de Grado.

Por último, en este trabajo solo se han analizado diecisiete lenguajes de los que han ido surgiendo en la última década, aunque ha habido muchos más. La decisión de elegir un grupo reducido y no todos, ha sido por tiempo y relevancia. No se ha considerado relevante nombrar a lenguajes que no han tenido un uso extendido, y por eso solo se realiza el análisis de los que más han sido utilizados o nombrados.

# CAPÍTULO 5: Glosario

## 5.1 Metaprogramación

La metaprogramación o más conocida como "*metaprogramming"*, por su nombre en inglés, se basa en escribir programas que utilizan a su vez otros programas como si fueran datos. También incluye los programas que como salida tienen otro programa, o los programas que se manipulan a sí mismos (auto programación) (JF, 2018).

Algunos de los conceptos más importantes de la metaprogramación son:

- Reflexión: que se define como la capacidad que posee un programa para poder modificar u observar su estructura. Lo más común es que esto ocurra en tiempo de ejecución. Lo normal es que al compilar un programa se pierde información sobre cómo es la estructura del programa mientras se genera el lenguaje ensamblador, sin embargo, los programas que cuentan con reflexión guardan dicha estructura como metadatos en el código que se genera. Esta propiedad se puede dividir en dos partes la introspección que es la capacidad que se posee para poder auto examinarse y la intercesión que es la capacidad de poder modificarse su propio significado o comportamiento (JF, 2018).
- Reificación: es la capacidad que hace posible convertir algo que es abstracto en un dato totalmente explícito. Un ejemplo de esto es crear un tipo de datos para poder acceder a una posición de memoria (JF, 2018).

## 5.2 Homoicónico

La homoicidad es una característica que presentan algunos lenguajes de programación. Esta propiedad consta de que un programa escrito en él puede ser manipulado como un dato o datos usando el propio lenguaje, lo que implica que internamente su representación pueda inferirse solamente con leer el programa. Por esto muchas veces suele resumirse esta propiedad como aquella que permite tratar el "*código como datos"* (Sterling & Shapiro, 2021).

## 5.3 Superset

Se dice que un lenguaje X es superset de otro Y cuando X se sobrescribe en base al lenguaje Y. De esta forma el lenguaje X obtiene muchas de las ventajas de su predecesor y mejora sus deficiencias. De esta manera se ayuda a los programadores a continuar sus proyectos sin necesidad de reescribir todo el código. Continúan los proyectos con el lenguaje X (Hunabku, 2020).

# Bibliografía


Acereda, J., & Freeman, P. (2021). *Read PureScript mediante ejemplos | Leanpub*. Recuperado 4 de julio de 2022, de https://leanpub.com/purescriptmedianteejemplos/read. Último acceso: 04/07/2022

Análisis de sitemas. (2015). *4. OPA | ANÁLISIS DE SISTEMAS*. https://analisisdesistemas1.wordpress.com/lenguajes-de-programacion/opa/. Último acceso: 06/07/2022

Apple. (2014). *Swift - Apple (ES)*. https://www.apple.com/es/swift/. Último acceso: 07/05/2022

Ballerina. (2017). *Ballerina Home*. https://ballerina.io/. Último acceso: 15/07/2022

Binsztok, H., Koprowski, A., & Swarczewskaja, I. (2013). *Opa : up and running*. 151.

Cálculo Lambda. (2020). *Cálculo lambda - Wikipedia, la enciclopedia libre*. https://es.wikipedia.org/wiki/Cálculo_lambda. Último acceso: 15/07/2022

Ceylon. (2011). *Eclipse Ceylon: Welcome to Ceylon*. https://ceylon-lang.org/. Último acceso: 25/04/2022

Chacón, J. L. (2021, octubre 25). *TypeScript: qué es, diferencias con JavaScript y por qué aprenderlo*. https://profile.es/blog/que-es-typescript-vs-javascript/#¿Como_funciona. Último acceso: 21/07/2022

Crespo, A. (2014). *HACK, el nuevo lenguaje de porgramación desarrollado por Facebook*. https://www.redeszone.net/2014/03/23/hack-el-nuevo-lenguaje-de-programacion-desarrollado-por-facebook/ Último acceso: 20/08/2022

Crystal. (2014). *Crystal - A languaje for humans and computers*.

Dart. (2011). *Dart programming language | Dart*. https://dart.dev/. Último acceso: 20/08/2022

Della Pittima, M. (2021, mayo 27). *(38) CUÁLES SON LAS RAMAS DE LA PROGRAMACIÓN | LinkedIn*. https://www.linkedin.com/pulse/cuáles-son-las-áreas-de-la-programación-marcos-della-pittima/?originalSubdomain=es. Último acceso: 20/04/2022

Deusto Formación. (2019). *Crystal un lkenguaje de programación un tanto desconocido*. https://www.deustoformacion.com/blog/programacion-tic/crystal-lenguaje-programacion-tanto-desconocido#:~:text=Crystal es un lenguaje de,lenguaje compilado como es C. Último acceso: 22/08/2022

Diccionario Sensagent. (2021). *Lenguaje de programación multiparadigma : definición de Lenguaje de programación multiparadigma y sinónimos de Lenguaje de*

*programación multiparadigma (español)*. Recuperado 25 de agosto de 2022, de http://diccionario.sensagent.com/Lenguaje de programación multiparadigma/es-es/. Último acceso: 29/08/2022

EcuRed. (2021). *Ceylon (lenguaje de programación) - EcuRed*. Recuperado 22 de agosto de 2022, de https://www.ecured.cu/Ceylon_(lenguaje_de_programación) . Último acceso: 20/08/2022

*EDteam*. (2020). *¿Cuáles son las áreas de la programación?* | https://ed.team/blog/cuales-son-las-areas-de-la-programacion. Último acceso: 06/07/2022

Elixir. (2011). *The Elixir programming language*. https://elixir-lang.org/. Último acceso: 25/04/2022

Elm. (2012). *Elm - delightful language for reliable web applications*. https://elm-lang.org/. Último acceso: 05/04/2022

Feldman, S. (1994). A Conversation with Alan Kay. *Interactions*, *1*(2), 13-22. https://doi.org/10.1145/174809.174811. Último acceso: 10/04/2022

García Pérez, E. I. (2018, abril 29). *Elixir el lenguaje de programación*. https://codigofacilito.com/articulos/elixir. Último acceso: 10/04/2022

Guía Raku. (2021). *Guía de Raku*. Recuperado 3 de agosto de 2022, de https://raku.guide/es/#_qué_es_raku. Último acceso: 10/04/2022

HACK. (2014). *HACK*. https://hacklang.org/. Último acceso: 29/04/2022

Hernández, U. (2018, junio 3). *Qué es TypeScript*. https://codigofacilito.com/articulos/typescript. Último acceso: 22/04/2022

Hmong. (2021). *Cuneiforme (lenguaje de programación)*. https://hmong.es/wiki/Cuneiform_(programming_language) . Último acceso: 06/04/2022

Hmong. (2021). *Rojo (lenguaje de programación) IntroducciónyCaracterísticas*. Recuperado 23 de agosto de 2022, de https://hmong.es/wiki/Red_(programming_language) . Último acceso: 03/05/2022

Hunabku. (2020, abril 20). *Javascript – El lenguaje de programación más utilizado - Hunabku*. https://hunabku.mx/javascript-el-lenguaje-de-programacion-mas-utilizado. Último acceso: 01/05/2022

JF, S. G. (2018). *Metaprogramación ¿qué es y para qué sirve?* https://riunet.upv.es/handle/10251/103846. Último acceso: 07/07/2022

Julia. (2012). *The Julia Programming Language*. https://julialang.org/. Último acceso: 20/04/2022. Último acceso: 07/07/2022

Kotlin. (2016). *Kotlin Programming Language*. https://kotlinlang.org/. Último acceso: 16/07/2022

Kripkit. (2021). *Opa (lenguaje de programación) - Ejemplo, Diseño y funcionalidad | KripKit*. Recuperado 6 de junio de 2022, de https://kripkit.com/opa-lenguaje-de-programacin/. Último acceso: 16/06/2022

Lauwens, B., & Downey, A. (2020, octubre 25). *Introducción a la Programación en Julia*. https://introajulia.org/#_por_qué_julia. Último acceso: 20/03/2022

Locura informática digital. (2021). *Bosque, El Nuevo Lenguaje de Programación de Microsoft ⚡*. Recuperado 3 de agosto de 2022, de https://www.locurainformaticadigital.com/2019/04/22/bosque-nuevo-lenguaje-programacion-microsoft/. Último acceso: 20/08/2022

Marron, M. (2021). *GitHub - microsoft/BosqueLanguage: The Bosque programming language is an experiment in regularized design for a machine assisted rapid and reliable software development lifecycle.* Recuperado 3 de agosto de 2022, de https://github.com/Microsoft/BosqueLanguage. Último acceso: 25/07/2022

Martínez, R., & García-Beltrán, A. (2000). Breve Historia De La Informática. *Universidad complutense de madrid*, 1-14. Último acceso: 25/07/2022

MDN contributors. (2020, diciembre 8). *MVC - Glosario | MDN*. https://developer.mozilla.org/es/docs/Glossary/MVC. Último acceso: 25/07/2022

Microsoft. (2019). *Bosque Programming Language*. https://www.microsoft.com/en-us/research/project/bosque-programming-language/ Último acceso: 25/07/2022

Naciones Unidas. (2015). *Objetivos y metas de desarrollo sostenible - Desarrollo Sostenible*. https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/#. Último acceso: 22/07/2022

NubeColectiva. (2020, abril 4). *Que es TypeScript y otros Detalles | Blog Nube Colectiva*. https://blog.nubecolectiva.com/que-es-typescript-y-otros-detalles/. Último acceso: 22/07/2022

OPA. (2011). *The Opa Language*. http://opalang.org/. Último acceso: 22/07/2022

Pastor, J. (2021, abril 30). *Rust es el lenguaje de moda y hay quien cree que se convertirá en el sucesor del legendario C*. https://www.xataka.com/aplicaciones/rust-lenguaje-moda-hay-quien-cree-que-se-convertira-sucesor-legendario-c. Último acceso: 22/07/2022

Preguntadores.net. (2020, marzo 20). *¿Cuáles son algunas de las ventajas del lenguaje de programación Raku? | Preguntadores.net*. https://preguntadores.net/q/cuales-son-algunas-de-las-ventajas-del-lenguaje-de-programacion-raku. Último acceso: 22/07/2022

Programacion II. (2021). *Tipos de memoria*. https://sites.google.com/site/programacioniiuno/temario/unidad-1---manejo-de-memoria-dinmica/tipos-de-memoria. Último acceso: 21/08/2022

PureScript. (2013). *PureScript*. https://www.purescript.org/. Último acceso: 20/04/2022

Quero Catalinas, E. (2002). *Sistemas operativos y lenguajes de programación*. . Último acceso: 20/04/2022Raku. (2015). *Raku Programming Language*. https://www.raku.org/ Último acceso: 20/04/2022

Red. (2011). *Red*. https://www.red-lang.org/. Último acceso: 20/07/2022

Rivero Espinosa, J. (2003). *«Historia de la programación»*. *100025022*, 1-115. Último acceso: 20/07/2022

Rojas, A. (2020, noviembre 19). *Programación funcional. Qué es y características*. https://www.incentro.com/es-ES/blog/que-programacion-funcional. Último acceso: 20/03/2022

Rust. (2010). *Rust Programming Language*. https://www.rust-lang.org/. Último acceso: 20/03/2022

Rust. (2014). *The Rust Programming Language - The Rust Programming Language*. https://doc.rust-lang.org/book/title-page.html. Último acceso: 20/03/2022

SG. (2013). *¿Por qué me Llama la Atención Ceylon? | SG Buzz*. https://sg.com.mx/buzz/por-que-ceylon. Último acceso: 14/04/2022

Sofftek. (2021). *Next-Gen Software : Languages & Tools Índice*. Último acceso: 09/03/2022

Sterling, L., & Shapiro, E. Y. (2021). *Homoiconicidad HistoriayUsos y ventajas*. Recuperado 4 de julio de 2022, de https://hmong.es/wiki/Homoiconicity. Último acceso: 09/03/2022

Swamy, N., Fournet, C., Rastogi, A., Bhargavan, K., Chen, J., Strub, P. Y., & Bierman, G. (2014). Gradual typing embedded securely in JavaScript. *Conference Record of the Annual ACM Symposium on Principles of Programming Languages*, 425-437. https://doi.org/10.1145/2535838.2535889. Último acceso: 02/04/2022

TIOBE. (2021). *Home - TIOBE*. Recuperado 25 de julio de 2022, de https://www.tiobe.com/. Último acceso: 20/08/2022

TIOBE a. (2021). *TIOBE Index - TIOBE*. Recuperado 25 de julio de 2022, de https://www.tiobe.com/tiobe-index/. Último acceso: 27/08/2022

Tokio School. (2020). *Lenguaje de programación Julia: ideal para Machine Learning*. https://www.tokioschool.com/noticias/lenguaje-programacion-julia/. Último acceso: 27/08/2022

Tomás, E. (2016, marzo 16). *Pattern matching en lenguajes de programación funcionales | campusMVP.es*. https://www.campusmvp.es/recursos/post/Pattern-matching-en-lenguajes-de-programacion-funcionales.aspx. Último acceso: 20/04/2022

Topkorae. (2021). *P4 (lenguaje de programación) - TOPKORAE.com*. Recuperado 4 de julio de 2022, de https://topkorae.com/wiki/es/P4_(programming_language) . Último acceso: 01/03/2022

Trigo Aranda, V. (2004). Historia y evolución de los lenguajes de programación. *Manual formativo de ACTA, 34*, 85-95. https://www.acta.es/medios/articulos/informatica_y_computacion/034083.pdf Último acceso: 20/04/2022

TypeScript. (2012). *TypeScript: JavaScript With Syntax For Types.* https://www.typescriptlang.org/. Último acceso: 25/04/2022

# ANEXO

# Anexo 1. Objetivos de desarrollo sostenible.

Los lideres mundiales el 25 de septiembre de 2015 se reunieron para adoptar un conjunto de objetivos para poder alcanzar diecisiete objetivos, los cuales se agrupan principalmente en tres bloques: proteger el planeta, asegurar la prosperidad para todo el mundo y erradicar la pobreza. Estos objetivos se han marcado para ir cumpliéndolos hasta 2030, por eso lo denominan la Agenda 2030. En la siguiente imagen se pueden ver estos diecisiete objetivos.

*Ilustración 44. Objetivos para el Desarrollo Sostenible.*


*Fuente: (Naciones Unidas, 2015)*

En este apartado se pretende evidenciar cómo contribuye la tecnología a los ODS, y es un hecho que uno de los aliados esenciales para poder alcanzar estos objetivos es la utilización de la tecnología.

Desde hace mucho tiempo se viene dando una revolución sobre el compromiso social que tenemos como individuos, la pandemia ha dejado al descubierto que no se puede continuar como hasta el momento en áreas como la sostenibilidad social y ambiental. Toda la sociedad está volcada en demandar a las empresas una actuación no solo sobre los beneficios económicos, sino que pide una implicación total sobre el impacto medioambiental y social.

En la siguiente tabla se muestra el grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS):

*Tabla 4. Tabla resumen sobre el grado de relación del trabajo con los ODS.*

| **Objetivos de Desarrollo Sostenibles** | **Alto** | **Medio** | **Bajo** | **No Procede** |
|---|---|---|---|---|
| ODS 1. **Fin de la pobreza.** | | | | **X** |
| ODS 2. **Hambre cero.** | | | | **X** |
| ODS 3. **Salud y bienestar.** | | | | **X** |
| ODS 4. **Educación de calidad.** | **X** | | | |
| ODS 5. **Igualdad de género.** | | | | **X** |
| ODS 6. **Agua limpia y saneamiento.** | | | | **X** |
| ODS 7. **Energía asequible y no contaminante.** | | | | **X** |
| ODS 8. **Trabajo decente y crecimiento económico.** | **X** | | | |
| ODS 9. **Industria, innovación e infraestructuras.** | **X** | | | |
| ODS 10. **Reducción de las desigualdades**. | | **X** | | |
| ODS 11. **Ciudades y comunidades sostenibles**. | | | | **X** |
| ODS 12. **Producción y consumo responsables.** | | | | **X** |
| ODS 13. **Acción por el clima.** | | | | **X** |
| ODS 14. **Vida submarina.** | | | | **X** |
| ODS 15. **Vida de ecosistemas terrestres.** | | | | **X** |
| ODS 16. **Paz, justicia e instituciones sólidas.** | | | | **X** |
| ODS 17. **Alianzas para lograr objetivos.** | | | | **X** |

En los objetivos que se ha marcado como 'No procede' es por el hecho de que los lenguajes de programación en sí solos no pueden ayudar a cumplirlos, tendrían que ser parte de proyectos donde al utilizar ese leguaje se hicieran sistemas para poder, por ejemplo, hacer un uso más eficiente del agua que se consume una fábrica (ODS6. Agua limpia y saneamiento). Los otros objetivos que sí se han marcado como relevantes se analizan con más detalle a continuación:

- ODS 4. Educación de calidad. Como hemos concluido en este trabajo los nuevos lenguajes de programación facilitan el aprendizaje tanto a los desarrolladores novatos como a los más experimentado. Esto ayuda a que una persona con interés sobre este mundo de la programación pueda adaptarse más fácilmente que si la dificultad de aprender los lenguajes fuera más elevada.
- ODS 8. Trabajo decente y crecimiento económico. Al ser los lenguajes de programación sencillos de aprender, ayudan a las empresas a formar en menos tiempo a sus empleados, por lo que les ofrecerán un trabajo decente. Como las formaciones durarán menos tiempo las empresas se verán recompensadas de forma económica más rápidamente.

- ODS 9. Industria, innovación e infraestructuras. Es gracias a los lenguajes que se puede seguir innovando en la tecnología que tienen actualmente las empresas. Gracias a estas innovaciones las industrias serán mucho más eficientes y ecológicas, ya que se pondrá el foco principalmente en tecnologías que ayuden a las empresas a cumplir con estos objetivos.
- ODS 10. Reducción de las desigualdades. Este mundo de la programación hace fácil el poder aprender, hay millones de libros y páginas web donde se pueden encontrar desde cursos básicos, hasta cursos muy avanzados. De esta manera se pueden reducir las desigualdades entre personas. Todos tenemos en términos generales las mismas herramientas para aprender a programar, es decir, las mismas oportunidades de aprender, por lo que se elimina la desigualdad.

Como conclusión, los lenguajes de programación no ayudan directamente a cumplir los ODS, pero si se les da un buen uso se podrá innovar tecnológicamente para poder alcanzarlos en el menor tiempo posible.