

UD 01

DESARROLLO DE SOFTWARE

ENTORNOS DE DESARROLLO
CFGs DAW

PARTE 1 DE 2: CONCEPTOS BÁSICOS Y LENGUAJES DE PROGRAMACIÓN

Autor: Sergio Badal

Fecha: 26-9-2020

Editado: Carlos Espinosa

Licencia Creative Commons versión 4.0



Reconocimiento - NoComercial - CompartirIgual (by-nc-sa): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.



ÍNDICE DE CONTENIDOS

1. CONCEPTOS BÁSICOS.....	1
1.1 Informática, binario, digital y software.....	1
1.2 Programar vs desarrollar software.....	2
1.3 Programa, librería, aplicación y suite.....	3
2. LENGUAJES DE PROGRAMACIÓN.....	4
2.1 Tipos de código (fuente, objeto, ejecutable y máquina).....	4
2.2 Ejecutables portables y scripts.....	6
2.3 Máquinas virtuales.....	7
2.4 Clasificación de los lenguajes.....	8
2.5 ¿Cómo escoger un lenguaje de programación?.....	12
3. CONTENIDO EXTRA PARA MENTES INQUIETAS.....	14
3.1 Actividad de investigación sobre lenguajes.....	14
3.2 Documental sobre robots, economía, clase media y el fin del mundo.....	15
4. BIBLIOGRAFÍA.....	16

1. CONCEPTOS BÁSICOS

1.1 Informática, binario, digital y software

La palabra **informática** viene de “información” y “automática”. La información es un conjunto de datos, y automático es que funciona por si mismo. Así pues la informática es el proceso de la información y su tratamiento, mediante un sistema automático, como puede ser un “ordenador”.



La informática permite automatizar el procesamiento de información. Toda la información que usa un “ordenador” debe estar representada en una secuencia de ceros y unos (dígitos) llamada **código binario**. Esto significa que cada unidad básica de información puede tener dos estados (0, 1). A dicha unidad se le conoce como **bit** o *binary digit* (b) y a la unión de ocho bits se le llama **byte** (B). Un byte puede representar 2^8 caracteres, es decir, 256 posibilidades distintas.

Cuando un dispositivo cualquiera maneja o almacena información en forma de bits (dígitos) decimos que ese dispositivo es **digital**. Cuando la información que maneja no es finita, como un reloj clásico o un termómetro de mercurio, decimos que es analógico.

El **software** es la parte intangible de un sistema informático, el equivalente al equipamiento lógico. Todo software está diseñado para realizar una tarea determinada en nuestro sistema y está presente en el **sistema operativo** (SO), en las aplicaciones que utilizamos y en, prácticamente, cualquier parte de un dispositivo electrónico moderno.



El software suele ser el traductor entre el hombre y la máquina. Es el encargado de comunicarse con el **hardware**, es decir, se encarga de traducir todas las órdenes que el usuario comunica a órdenes comprensibles por el hardware (p. ej. el driver de una impresora).

1.2 Programar vs desarrollar software

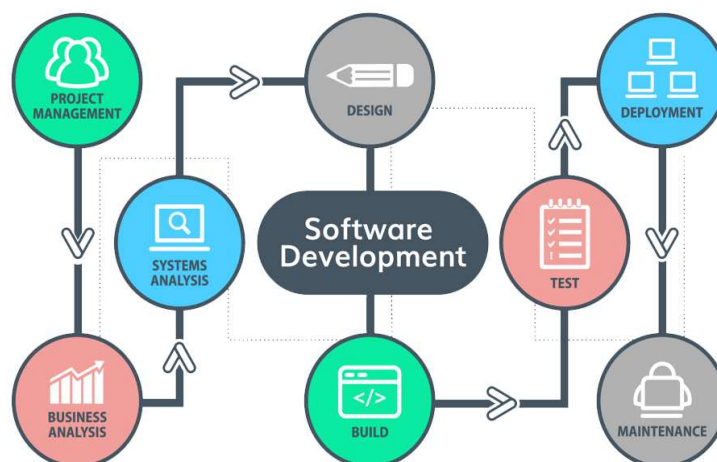
Verás muchas definiciones de lo que es **programar** pero, en esencia, consiste en darle órdenes y datos a un ordenador para recibir una serie de resultados o provocar cierto comportamiento. Para ser técnicamente correctos, usaremos **dispositivo** en lugar de ordenador e **instrucción** en lugar de orden.



Importante

*Es importante usar la palabra “dispositivo”, ya que hoy en día podemos programar desde un ordenador hasta una nevera, pasando por unas zapatillas deportivas o, por qué no, hasta una alfombra si estos manejan información digital. Cada vez más elementos de nuestra vida cotidiana aceptan órdenes en forma de comandos, líneas de código o programas, dentro de lo que se conoce como **Internet de las Cosas**, Internet of Things o con sus siglas en inglés **IoT**, un concepto que evoluciona del concepto clásico de domótica.*

Desarrollar software es o, al menos, debería ser mucho más que darle órdenes a un dispositivo, mucho más que programar. De hecho, el tiempo que dedicamos, o que deberíamos dedicar, a analizar, diseñar, probar, documentar y mantener el software es mucho mayor que el tiempo que dedicamos exclusivamente a escribir líneas de código. Preferimos decir **desarrollador de software** (o simplemente desarrollador) en lugar de programador, si consideramos que nuestra labor se basa en “algo más” que codificar.



1.3 Programa, librería, aplicación y suite

Del mismo modo que verás muchas definiciones y usos del término programar también verás muchas interpretaciones de los términos programa y aplicación.

Podríamos decir que un **programa** es una serie de ~~órdenes~~ instrucciones secuenciadas u ordenadas con una finalidad concreta y que devuelven un valor o realizan una función determinada. Por ejemplo, una función que dados números te devuelva la suma o un procedimiento que minimiza las ventanas de tu escritorio.

Por otro lado, una **librería** es un archivo, o contenedor lógico, que contiene una serie de programas. Por ejemplo, cualquiera de los archivos DLL que encuentras en la carpeta System32 de Windows.



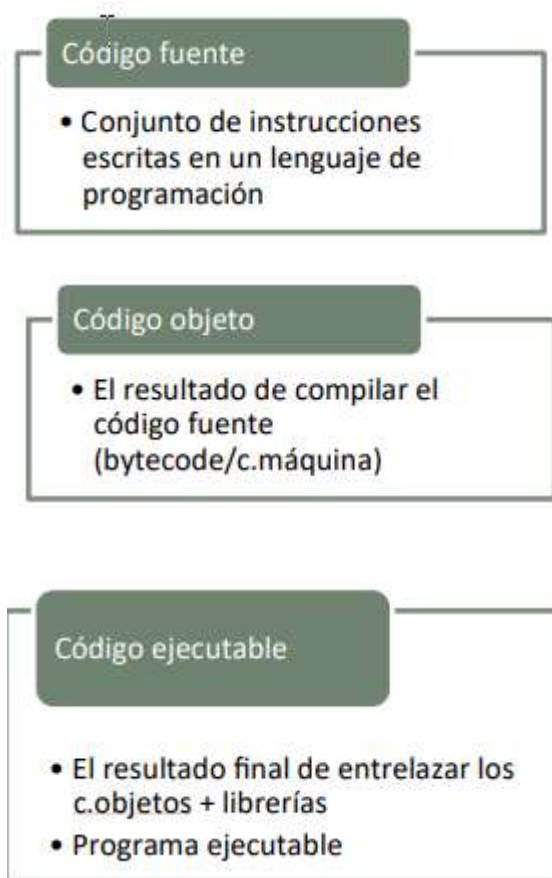
Importante

Nuestro código fuente puede usar funciones de cientos de líneas de código que otras personas ya han escrito y que están almacenadas en archivos externos llamados librerías. Imagina que estás programando una calculadora científica que necesita realizar cálculos complejos. Tú escribirás el código fuente de la calculadora y le dirás (ya verás cómo hacerlo) que utilice determinadas funciones almacenadas en determinados archivos.

A su vez, una **aplicación** está formada por varios programas con sus librerías correspondientes, aunque podrían constar solamente de un programa.

Por último, y con esto ya acabamos, cuando lo que tenemos son varias aplicaciones que pueden ejecutarse independientemente, una de otra, suele denominarse **suite** o “paquete integrado”, como por ejemplo nos sucede con MS Office o con Libre y Open Office.





Al conjunto de TODAS las ~~órdenes~~ instrucciones que forman un programa le llamaremos **código fuente**. En el caso de java, el código fuente se guarda en un fichero con la extensión “.java”.

Como hemos dicho, tenemos que transformar el código fuente de nuestro programa al código que entienda nuestro hardware. Esto se realiza con los compiladores, que generan una especie de código intermedio al que llamamos **código objeto, bytecode o código intermedio**. En el caso de Java, el compilador genera un archivo de extensión “.class”.

Si a estos objetos les añadimos las funciones que hemos usado de librerías y las peculiaridades del sistema operativo en el que se va a ejecutar obtenemos el **código ejecutable**, que es el conjunto de ~~órdenes~~ instrucciones directamente interpretable por el dispositivo que estamos usando. Suele usarse un **enlazador**.

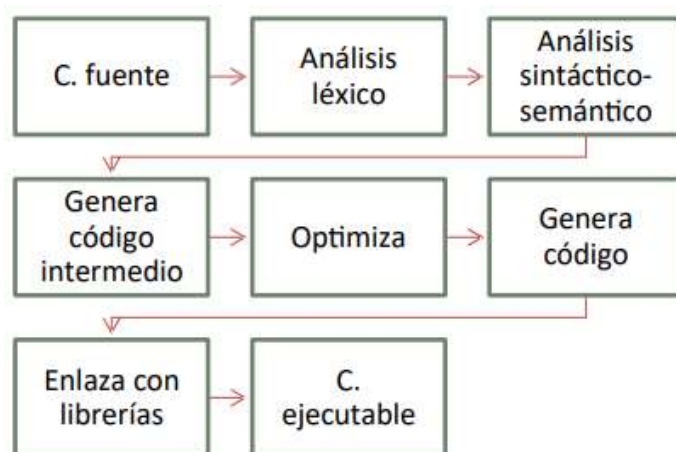
Piensa que, un mismo código fuente, para poder ser ejecutado en diferentes sistemas operativos necesita ser compilado de varias maneras. Por ejemplo, la calculadora de la que hablamos puede que tenga un único código fuente pero seguro que tendrá diferentes ejecutables, uno por cada sistema operativo en el que quieras que funcione.



Importante

Nuestro código fuente puede usar funciones de cientos de líneas de código que otras personas ya han escrito y que están almacenadas en archivos externos llamados librerías. Imagina que estás programando una calculadora científica que necesita realizar cálculos complejos. Tú escribirás el código fuente de la calculadora y le dirás (ya verás cómo hacerlo) que utilice determinadas funciones almacenadas en determinados archivos.

El esquema de traducción que sigue el compilador es el que se muestra en la siguiente figura:



Interesante

Verás que se usa de manera indistinta los términos **plataforma, arquitectura, máquina, procesador y sistema operativo** cuando hablamos de que una aplicación determinada está compilada para poder “correr” (ser ejecutada) en diferentes entornos. Ten en cuenta que, en la mayoría de los casos, estos términos son sinónimos.

2.2 Ejecutables portables y scripts

Sobre lo que te acabamos de contar, existen varias peculiaridades.

Hay distintos tipos de archivos ejecutables, dependiendo cómo se traten las ~~órdenes~~ instrucciones:

- **Ejecutables portables.** Se pueden ejecutar en varias plataformas. Por ejemplo, un ejecutable Java es portable ya que utiliza un bytecode no asociado a un procesador en concreto.
- **Ejecutables no portables.** Destinado a una plataforma concreta. Por ejemplo un ejecutable en C.

Sin embargo en un sentido más general, un programa ejecutable no tiene por qué necesariamente contener código máquina, sino que puede tener instrucciones a interpretar por otro programa. Este tipo de ejecutables son conocidos con el nombre de **scripts**. Ejemplo: Script en bash de Linux o un script en PHP.

Determinar si un archivo es ejecutable es sobre todo una cuestión de convención. Unos sistemas operativos se basan en la extensión de archivo (como la terminación .exe) y otros lo hacen leyendo los metadatos (como los bits de permiso de ejecución en Unix).

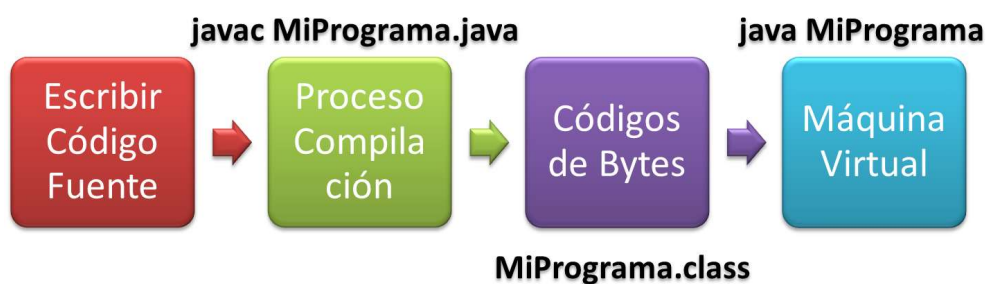
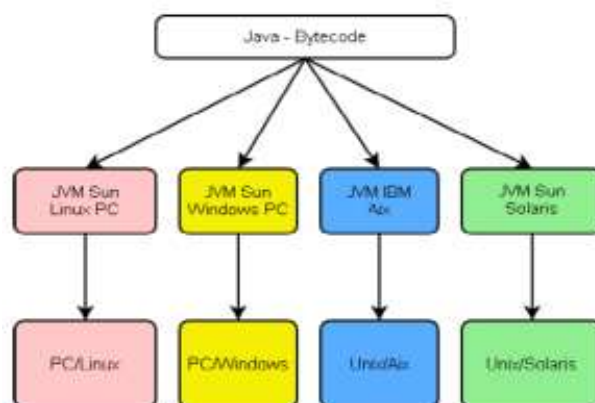
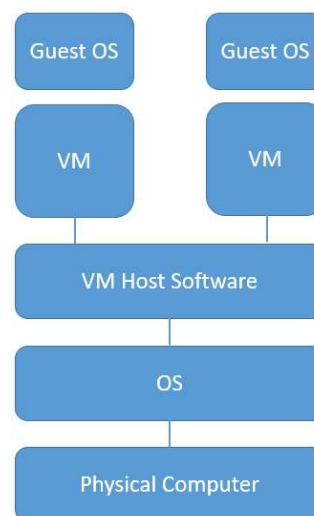
2.3 Máquinas virtuales

Algunos lenguajes como Java usan una aplicación que emula un SO dentro de otro SO y que se conoce como **máquina virtual**.

Existen dos tipos de máquinas virtuales:

- **Las máquinas virtuales de sistema:** Se simula un ordenador completo dentro del nuestro a través de un programa de virtualización. Estos programas simulan una arquitectura hardware lo que permite instalar sistemas operativos diferentes a los creados para nuestra arquitectura. Otras aplicaciones son realizar pruebas o simular una red de ordenadores. Ejemplos: VMWare, VirtualBox...

- **Las máquinas virtuales de proceso:** se utilizan para independizar la ejecución de un proceso del hardware subyacente. El bytecode (**código objeto**) es interpretado por la máquina virtual que se encarga de traducir las operaciones para que funcionen con el hardware específico de la máquina que estemos usando. Sirve de entorno de ejecución para estos programas. La más conocida es la máquina virtual de java o JVM (Java Virtual Machine). La JVM es un ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial (el Java bytecode), el cual es generado por el compilador del lenguaje Java.



2.4 Clasificación de los lenguajes

Existen casi tantas clasificaciones de lenguajes como lenguajes en sí, dependiendo de los autores. Aquí te presentamos una de las muchas posibles:



a) Tipo de ejecución¹

- **Lenguajes interpretados**

- El uso de los lenguajes interpretados ha venido en crecimiento y cuyos máximos representantes son los lenguajes usados para el desarrollo web entre estos **Ruby, Python, PHP** (se interpreta del lado del servidor), **JavaScript** y otros como **Perl, Smalltalk, MATLAB, Mathematica**.
- Básicamente un lenguaje interpretado es aquel en el cual sus instrucciones o más bien el código fuente, escrito por el programador en un lenguaje de alto nivel, es traducido por el interprete a un lenguaje entendible para la máquina paso a paso, instrucción por instrucción.
- Los lenguajes interpretados permiten el **tipado dinámico de datos**, es decir, no es necesario inicializar una variable con determinado tipo de dato sino que esta puede cambiar su tipo en condición al dato que almacene entre otras características más.
- A los lenguajes interpretados los vemos más en software de entornos web o terminales de comandos, ya que requieren de menores recursos y de acceso a archivos determinados.

1 Cesc1989, L. T. L. E. (2012, 2 septiembre). *Lenguajes de programación: Compilados vs Interpretados*. Otro Espacio Blog. <https://otroespacioblog.wordpress.com/2012/09/02/lenguajes-de-programacion-compilados-vs-interpretados/>

- **Lenguajes compilados**

- Un lenguaje compilado es aquel cuyo código fuente, escrito en un lenguaje de alto nivel, es traducido por un compilador a un archivo ejecutable entendible para la máquina en determinada plataforma. Con ese archivo se puede ejecutar el programa cuantas veces sea necesario sin tener que repetir el proceso por lo que el tiempo de espera entre ejecución y ejecución es ínfimo.
- Dentro de los lenguajes de programación que son compilados tenemos la familia **C** que incluye a **C++, Objective C, C#** y también otros como **Fortran, Pascal, Haskell y Visual Basic**.
- A los lenguajes compilados los vemos más en software de escritorio ya que requieren de mayores recursos y de acceso a archivos determinados. También por el peso mayor que estos suelen tener en sus archivos ejecutables.

- **Lenguajes virtuales**

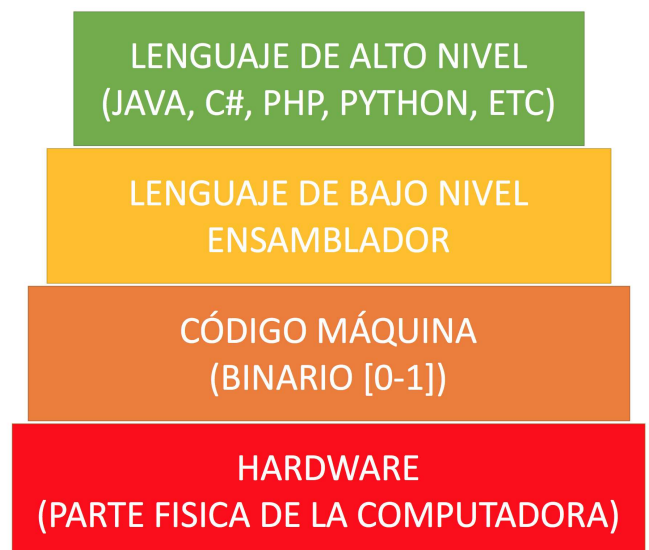
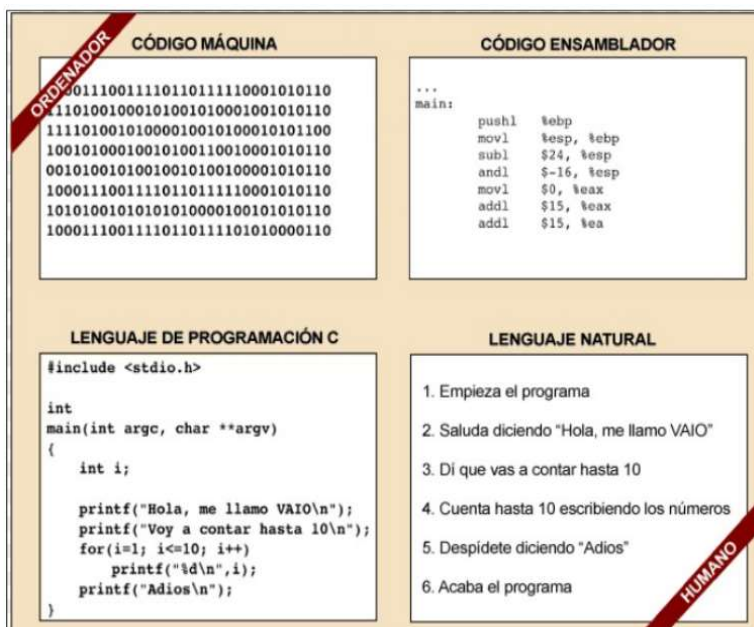
- Algunos autores clasifican a lenguajes como Java lenguajes virtuales o híbridos. Es un caso particular ya que hace uso de una máquina virtual que se encarga de la traducción del código fuente por lo que hay veces es denominado compilado e interpretado o virtual. Otra ventaja de la máquina virtual que usar Java es que le permite ejecutar código Java en cualquier máquina que tenga instalada la JVM.

**Interesante**

En definitiva no se puede decir que uno es mejor que el otro sin tener en cuenta los aspectos mencionados anteriormente por eso es bueno aprender uno de cada tipo, ¿no? Así como programadores desarrolladores cubriremos dos terrenos totalmente distintos.

b) Nivel de abstracción

- Respecto del nivel de abstracción, podría definirse como el nivel de cercanía al lenguaje natural, siendo un lenguaje de alto nivel el que se parece más al lenguaje natural y un lenguaje de bajo nivel el que es más similar al lenguaje máquina.
- Los **lenguajes de programación de bajo nivel** son los que dan un conjunto de instrucciones aritmético-lógicas que no permiten el uso de funciones que no estén ya contempladas en la arquitectura del hardware. Estos suelen ser una mera traducción del código máquina para facilitar su legibilidad y escritura.
- Los **lenguajes de programación de alto nivel** son los más abstractos. Están más cerca de la forma natural de habla humana que de las instrucciones de código máquina. Son los lenguajes más usados en la actualidad e incluyen ejemplos como C++, Java, PHP, Python... (Cuesta, 2014).
- Algunos autores clasifican a lenguajes como C un **lenguaje de programación de medio nivel** puede acceder a registros del sistema y direcciones de memoria, todas propias de lenguajes de bajo nivel.²



2 Álvarez, G. (2019, 29 abril). Niveles de lenguajes de programación. Kyocode. <https://www.kyocode.com/2018/09/niveles-de-lenguajes-de-programacion/>

c) Paradigma de programación



Importante

Un paradigma de programación es un conjunto de teorías, estándares y métodos que juntos representan un medio de organización de conocimientos. Dicho con otras palabras, es un conjunto de características comunes que comparten varios lenguajes de programación. De ahí que existan lenguajes multiparadigma, al no encajar exactamente en ningún paradigma concreto y cumplir solo unas características de uno y de otro o, también se da el caso, de que cumpla todas las características de más de un paradigma.

- Los lenguajes **imperativos** se basan en conocer el estado de la máquina y modificarlo mediante instrucciones. Se llaman imperativos porque usamos órdenes para decir al ~~ordenador~~ dispositivo qué debe hacer y cómo hacerlo. La mayoría de arquitecturas de ordenadores siguen una filosofía imperativa por lo que la traducción de un lenguaje imperativo a código máquina es más sencilla que la de los lenguajes declarativos.

- Son lenguajes imperativos prácticamente todos los que conocemos: Perl, C, Java, PHP, Python ...
- Dentro de esta categoría están:
 - Los **estructurados** que solo permiten tres estructuras: La secuencia, la selección y la iteración. Esto hace innecesario el uso de saltos en el código y se suelen prohibir para mejorar la legibilidad aunque, a fecha de hoy, seguimos usando saltos en mucho programas que hacemos con instrucciones como el “break” en los “switch”.³
 - Los **orientados a objetos** que van un paso más allá juntando los datos y las operaciones que se realicen sobre esos datos en un único módulo o clase. Los objetos se definen como un conjunto de estado (los datos o atributos), el comportamiento (los métodos o funcionalidad) y la identidad (generalmente un atributo) que permite diferenciarlo del resto.

- Los lenguajes **declarativos** fijan un objetivo pero no el camino para llegar a él. Se le indica al ordenador qué queremos obtener y se le detalla una descripción del problema. La solución se encuentra con estos elementos siguiendo una lógica interna. Son lenguajes más cercanos a las matemáticas.

- Son lenguajes declarativos Prolog, LISP y uno que seguro que te resulta familiar: SQL.

3 Extended Learning Institute (ELI) at Northern Virginia Community College (NOVA). (s. f.). Reading: Structured Programming | Introduction to Computer Applications and Concepts. Scribbr.
<https://courses.lumenlearning.com/zeliite115/chapter/reading-structured-programming/>

2.5 ¿Cómo escoger un lenguaje de programación?

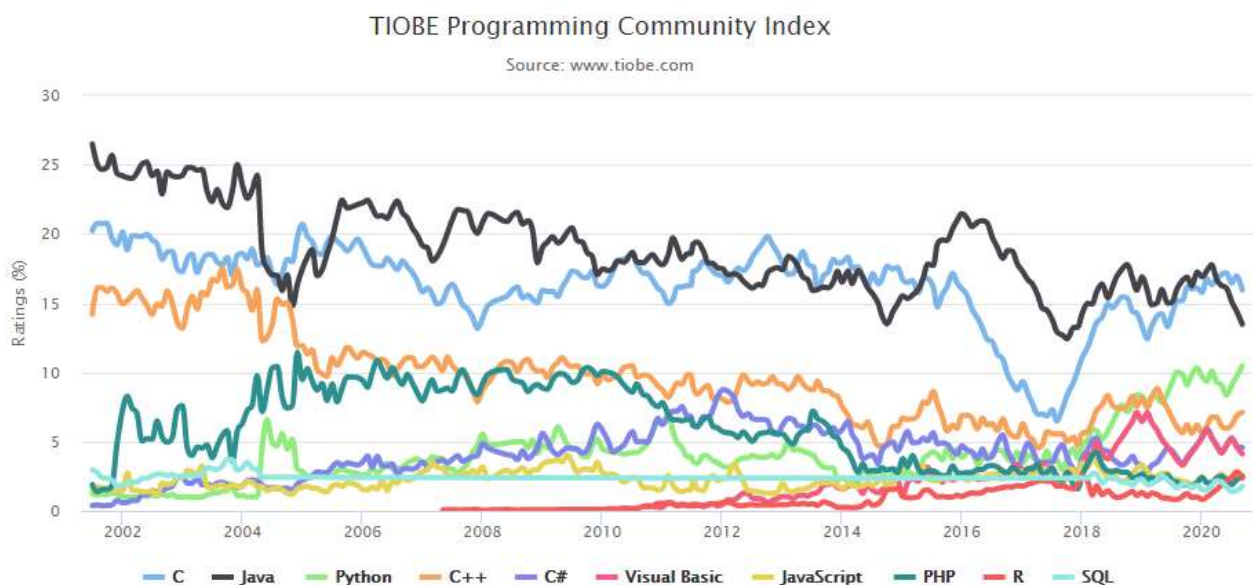
Resolver esta pregunta podría llevarnos días enteros divagando entre todos los posibles paradigmas y tipos de lenguajes y llegaríamos a la conclusión de: DEPENDE.

Cada lenguaje de programación tiene unas características determinadas que lo hacen más o menos apropiado para el contexto donde lo vayas a usar, pudiendo ser determinante estos factores:

1. El sector productivo al que va dirigida tu aplicación (medicina, académico, videojuegos...)
2. En qué dispositivo va a ejecutarse (móviles, escritorio, IoT...)
3. La plataforma sobre la que va a ejecutarse (Windows, Linux, Mac..)

Saber qué idioma es más hablado del mundo es tan “sencillo” como contar el número de personas que lo hablan pero saber cuál es el lenguaje más usado es algo más complicado. Para que te hagas una idea de “cómo está el patio” aquí tienes uno de los índices más reputados que miden eso tan difícil de medir como es “cuanto se usa un lenguaje de programación”. Se llama **índice TIOBE** y lo puedes consultar aquí:

- <https://www.tiobe.com/tiobe-index/>



3. CONTENIDO EXTRA PARA MENTES INQUIETAS

3.1 Actividad de investigación sobre lenguajes



¿Sabes cuál es el lenguaje de programación más utilizado en el mundo (según TIOBE)? Incluye una captura de la evolución según TIOBE. Existe otro ranking de lenguajes de programación llamado PYPL. ¿Coinciden en los mismos lenguajes? Estudia por qué difieren en sus resultados.

POSIBLE RESPUESTA

- TIOBE quizás sea el más conocido. Lo elabora la empresa Tiobe, especializada en evaluación de calidad de programas software. Según indican en su propia página establece la popularidad de los lenguajes en función del número de resultados que se producen en los 25 buscadores más utilizados.

Sep 2020	Sep 2019	Change	Programming Language	Ratings	Change
1	2	▲	C	15.95%	+0.74%
2	1	▼	Java	13.48%	-3.18%
3	3		Python	10.47%	+0.59%
4	4		C++	7.11%	+1.48%
5	5		C#	4.58%	+1.18%

- El PYPL o Índice de Popularidad de los Lenguajes de Programación, se elabora analizando el número de búsquedas en Google de tutoriales sobre un determinado lenguaje. Obtiene los datos de Google Trends, por lo que argumentan que es mucho más transparente en su cálculo ya que cualquiera podría elaborarlo para su país o lengua, por ejemplo. También es cierto, que el análisis, al ser exclusivo en un solo buscador, aunque sea el líder, puede ser más limitado.

Worldwide, Sept 2020 compared to a year ago:				
Rank	Change	Language	Share	Trend
1		Python	31.56 %	+2.9 %
2		Java	16.4 %	-3.1 %
3		Javascript	8.38 %	+0.3 %
4		C#	6.5 %	-0.8 %
5		PHP	5.85 %	-0.5 %

FUENTE: <https://www.digitallearning.es/blog/rankings-de-lenguajes-de-programacion/>

3.2 Documental sobre robots, economía, clase media y el fin del mundo

El 27 de abril del año pasado se estrenó el documental #MiEmpleoMiFuturo

- **Un documental sobre robots, economía, clase media... y el fin del mundo**

La campaña cuenta con una carta abierta a los políticos para pedirles que empiecen a hablar de sobre cómo la automatización de tareas está cambiando la estructura del sector productivo.


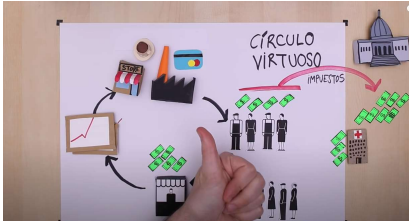

- La carta se puede firmar a traves de: <https://www.change.org/miempleomifuturo>

Se trata de la pieza central de la campaña lanzada por la Fundación Cotec para provocar una reflexión sobre la necesidad de **acompañar con iniciativas políticas la llegada de la cuarta revolución industrial:** Robots, algoritmos, internet de las cosas, coches autónomos...

Las máquinas están sustituyendo muchos empleos. La clase media está en peligro. El futuro de los jóvenes está amenazado. ¿Estamos a tiempo de hacer algo?...

El debate está abierto en las redes con la etiqueta **#MiEmpleoMiFuturo**

Aquí tienes el trailer de la presentación y las dos partes del corto que no te dejará indiferente:

Estreno del corto #MiEmpleoMiFuturo (1 min)	Primera parte del corto (15 min)	Segunda parte del corto (15 min)
		
https://youtu.be/89bNFRZM0D0	https://youtu.be/htAnVeMtrr8	https://youtu.be/-z5z8aGRSQ0

4. BIBLIOGRAFÍA

- i. Cuesta Vicente, Sergio, S. C. V. (2011, 17 octubre). *Tema 1 - Desarrollo de Software*. Apuntes de Sergio Cuesta Vicente. docs.google.com/viewer?a=vπd=sites&srcid=ZG9tZW5pY29zY2FybGF0dGkuZXN8c2N1ZXN0YXxneDoyZGQ2MjBjNDcwODIyYmFj
- ii. The Open University, (2019, 6 noviembre). *An introduction to software development*. The Open University - England. open.edu/openlearn/science-maths-technology/introduction-software-development/content-section-0?active-tab=description-tab
- iii. Javier Garzás, J. G. (consultado online 2020, 2 octubre). *Javier Garzás - Mentor Ágil, Ágil Coach. Gestión de proyectos y equipos*. Blog oficial de Javier Garzás. javiargarzas.com
- iv. Iglesias, C. C. (2020). *Entornos de Desarrollo (GRADO SUPERIOR)*. RA-MA S.A. Editorial y Publicaciones.
- v. Aldarias, F. (2012): *Apuntes de Entornos de Desarrollo*, CEEDCV