0. Introducción a la programación

Iniciación a la programación

Programación con Python

Institut Educació Secundària
El Caminàs

Curso de especialización en Ciberseguridad

Contenidos

1 Iniciación a la programación

Como en cualquier proceso de aprendizaje, hay que empezar por el principio. Es importante tener claros un conjunto de conceptos básicos que ayuden a comprender los conceptos más avanzados que vendrán posteriormente. En este caso, se trata de establecer qué es un programa, como funciona y qué es el modelo general para crear uno. Sólo una vez lo tengáis claro os podéis plantear sentaros ante el ordenador y empezar a programar.

1.1 ¿Qué es un programa?

Un primer paso para poder empezar a estudiar como es debido a hacer un programa informático es tener claro qué es un programa. En contraste con otras términos usados en informática, es posible referirse en un "programa" en el lenguaje coloquial sin tener que estar hablando necesariamente de ordenadores. Os podríais estar refiriendo en el programa de un ciclo de conferencias o de cine. Pero, a pesar de no tratarse de un contexto informático, este uso ya os aporta una idea general de su significado: un conjunto de acontecimientos ordenados de forma que suceden de forma secuencial en el tiempo, uno tras otro.

Otro uso habitual, ahora ya sí que vinculado al contexto de las máquinas y los autómatas, podría ser para referirse al programa de una lavadora o de un robot de cocina. En este caso, pero, lo que sucede es un conjunto no tanto de acontecimientos, sino de órdenes que el electrodoméstico sigue ordenadamente. Un vez seleccionado el programa que queremos, el electrodoméstico hace todas las tareas correspondientes de manera autónoma.

Por ejemplo, el programa de un robot de cocina para hacer una crema de maíz sería:

1. Espera a que introduzcáis maíz y mantequilla.
2. Gira durante un minuto, avanzando progresivamente de la velocidad 1 a la 5.
3. Espera a que introduzcáis leche y sal.
4. Gira durante 30 segundos a velocidad 7.
5. Gira durante 10 minutos a velocidad 3 mientras cuece a una temperatura de 90 grados.
6. Se para. La crema está lista!

Este conjunto de órdenes no es arbitrario, sino que sirve para llevar a cabo una tarea de cierta complejidad que no se puede hacer de un solo golpe. Se tiene que hacer paso a paso. Todas las órdenes están vinculadas entre sí para llegar a lograr este objetivo y, sobre todo, es muy importante el orden en que se llevan a cabo.

Entrando ya, ahora sí, en el mundo de los ordenadores, la manera como se estructura el tipo de tareas que estos pueden hacer tiene mucho en común con los programas de electrodomésticos. En este

caso, en lugar de transformar ingredientes (o lavar ropa sucia, si se tratara de una lavadora), lo que el ordenador transforma es información o datos.

> Un *programa informático* no es más que una serie de órdenes que se llevan a cabo secuencialmente, aplicadas sobre un conjunto de datos.

¿Qué datos procesa un programa informático? Bien, esto dependerá del tipo de programa:

- Un editor procesa los datos de un documento de texto.
- Una hoja de cálculo procesa datos numéricos.
- Un videojuego procesa los datos que dicen la forma y ubicación de enemigos y jugadores, etc.
- Un navegador web procesa las órdenes del usuario y los datos que recibe desde un servidor a Internet.

Por lo tanto, la tarea de un programador informático es escoger qué órdenes constituirán un programa de ordenador, en qué orden se tienen que llevar a cabo y sobre qué datos hay que aplicarlas, para que el programa lleve a cabo la tarea que tiene que resolver. La dificultad de todo será más grande o más pequeña dependiendo de la complejidad misma de aquello que hace falta que el programa haga. No es lo mismo establecer qué tiene que hacer el ordenador para resolver una multiplicación de tres números que para procesar textos o visualizar páginas a Internet.

> **Ejecutar un programa**
>
> Por "ejecutar un programa" se entiende hacer que el ordenador siga todas sus órdenes, desde la primera hasta la última.

Por otro lado, una vez hecho el programa, cada vez que lo ejecutáis, el ordenador cumplirá todas las órdenes del programa.

De hecho, un ordenador es incapaz de hacer absolutamente nada por si mismo, siempre hay que decirle qué tiene que hacer. Y esto se le dice mediante la ejecución de programas. A pesar de que desde el punto de vista del usuario puede parecer que cuando se pone en marcha un ordenador este funciona sin ejecutar ningún programa concreto, hay que tener en cuenta que su sistema operativo es un programa que está siempre en ejecución.

1.2 Tipo de órdenes que acepta un ordenador

Para llevar a cabo la tarea encomendada, un ordenador puede aceptar diferentes tipos de órdenes. Estas se encuentran limitadas a las capacidades de los componentes que lo conforman, del mismo modo que el programa de una lavadora no puede incluir la orden de gratinar, puesto que no tiene gratinador. Por lo tanto, es importante tener presente este hecho para saber qué se puede pedir al ordenador cuando creáis un programa.

La estructura interna del ordenador se divide en una serie de componentes, todos comunicados entre si, tal como muestra la siguiente figura de manera muy simplista, pero suficiente para empezar. Cada orden de un programa está vinculada de una manera u otra a alguno de estos componentes.

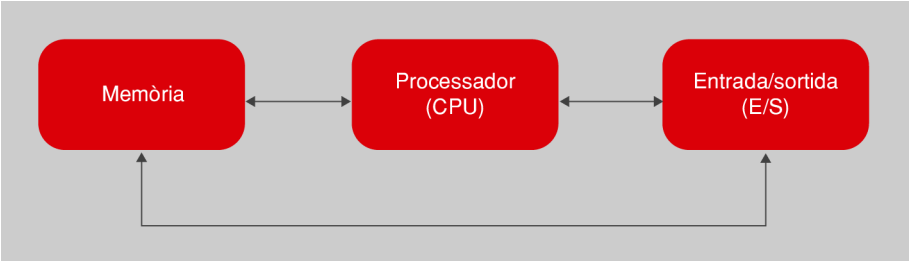


Figure 1: Componentes básicos de un ordenador

> **Procesador**
>
> El procesador también es conocido popularmente por sus siglas en inglés: CPU (Central Processing Unit, unidad central de procesamiento).

El **procesador** es el centro neurálgico del ordenador y el elemento que es capaz de llevar a cabo las órdenes de manipulación y transformación de los datos. Un conjunto de datos se puede transformar de muchas maneras, según las capacidades que ofrezca cada procesador. Aún así, hay muchas transformaciones que todos pueden hacer. Un ejemplo es la realización de operaciones aritméticas (suma, resto, multiplicación, división), tal como hacen las calculadoras.

La **memoria** permite almacenar datos mientras estas no están siendo directamente manipuladas por el procesador. Cualquier dato que tiene que ser tratado por un programa estará en la memoria. Mediante el programa se puede ordenar al procesador que guarde ciertos datos o que los recupere en cualquier momento. Normalmente, cuando se habla de memoria a este nivel nos referimos a memoria dinámica o RAM (Random Access Memory, memoria de acceso aleatorio). Esta memoria no es persistente y una vez acaba la ejecución del programa todos los datos con las cuales trataba se desvanecen. Por lo tanto, la información no se guardará entre sucesivas ejecuciones diferentes de un mismo programa.

En ciertos contextos es posible que nos encontramos también con memoria ROM (Read-Only memory, memoria sólo de lectura). En esta, los datos están predefinidos de fábrica y no se puede almacenar nada, sólo podemos leer el que contiene. Hay que decir que no es el caso más habitual.

El **sistema de entrada/salida** (abreviado como E/S) permite el intercambio de datos con el exterior del ordenador, más allá del procesador y la memoria. Esto permite traducir la información procesada en acciones de control sobre cualquier periférico conectado al ordenador. Un ejemplo típico es establecer una vía de diálogo con el usuario, ya sea por medio del teclado o del ratón para pedirle

información, como por la pantalla, para mostrar los resultados del programa. Este sistema es clave para convertir un ordenador en una herramienta de propósito general, puesto que lo capacita para controlar todo tipo de aparatos diseñados para conectarse.

Otra posibilidad importante del ordenador, atendidas las limitaciones del sistema de memoria, es poder interactuar con el hardware de almacenamiento persistente de datos, como un disco duro.

> **Un ordenador es como una pizzería**
>
> Si se quiere hacer un símil con nuestro mundo de cada día, un ordenador es como la cocina de una pizzería que acepta pedidos telefónicos. Hacer un pedido equivale a pedir el inicio de la ejecución de un programa. Para llevar a cabo este pedido, habrá que manipular una serie de ingredientes, que representarían los datos. El cocinero con sus enseres (horno, pastador, etc.) serían el procesador, puesto que manipulan y transforman los ingredientes. La nevera, los armarios o los contenedores, de donde el cocinero puede sacar ingredientes o donde los puede desar mientras no los está manipulando, representarían la memoria. El sistema de entrada/salida serían los elementos de comunicación con el exterior de la pizzería, como el motorista que trae la pizza o el teléfono que el cocinero puede utilizar para pedir que le traigan nuevos ingredientes cuando se le acaban, pedir información adicional al usuario ("Se ha acabado el pimiento, va bien si ponemos cebolla?"), o avisarlo de algún acontecimiento ("Me sabe mal, tardará algo más del previsto"). De hecho, continuando con el símil, el cocinero prepara una pizza siguiendo un conjunto de pasos. En este caso la receta son las órdenes que tiene que seguir el programa. Y si el cocinero no tiene la receta no puede llevar a cabo el pedido.
>
> **Lenguaje natural**
>
> El lenguaje natural es aquel que empleamos los humanos para comunicarnos habitualmente.

Partiendo de esta descripción de las tareas que puede llevar a cabo un ordenador según los elementos que lo componen, un ejemplo de programa para multiplicar dos números es el mostrado a la **Tabla.1**. Lo tenéis expresado en lenguaje natural. Notad como los datos tienen que estar siempre almacenadas a la memoria para poder operar.

Tabla 1 Un programa que multiplica dos números usando lenguaje natural

Orden para dar	Elemento que lo efectúa
1. Lee un número del teclado	E/S (teclado)
2. Guarda del número en memoria	Memoria
3. Lee otro número del teclado	E/S (teclado)
4. Guarda del número en memoria	Memoria

Orden para dar	Elemento que lo efectúa
5. Recupera los números de la memoria y hace la multiplicación	Procesador
6. Guarda el resultado en memoria	Memoria
7. Muestra el resultado a la pantalla	E/S (pantalla)

1.3 Crear un programa ejecutable...

Para crear un programa hay que establecer qué órdenes se tienen que dar al ordenador y en qué secuencia. Ahora bien, hoy en día los ordenadores todavía no entienden el lenguaje natural (cómo se utiliza a la **tabla.1**), puesto que está lleno de ambigüedades y aspectos semánticos que pueden depender del contexto.

> **Artificial** Por artificial entendemos aquello que no ha evolucionado a partir del uso entre humanos, sino que ha sido creado expresamente, en este caso para ser usado con los ordenadores.
>
> **Lenguaje de programación** Para especificar las órdenes que tiene que seguir un ordenador el que se usa es un lenguaje de programación. Se trata de un lenguaje artificial diseñado expresamente para crear algoritmos que puedan ser llevados a cabo por el ordenador.

Igual como hay muchas lenguas diferentes, también hay muchos lenguajes de programación, cada uno con sus características propias, que los hacen más o menos indicados para resolver unos tipos de tareas u otras. Todos, pero, tienen una sintaxis muy definida, que hay que seguir para que el ordenador interprete correctamente cada orden que se le da. Es exactamente lo mismo que pasa con las lenguas del mundo: para expresar los mismos conceptos, el castellano y el japonés usan palabras y normas de construcción gramatical totalmente diferentes entre sí.

> En un lenguaje de programación determinado, la agrupación de órdenes concretas que se pide al ordenador que haga se denomina **conjunto de instrucciones**.

Normalmente, el conjunto de instrucciones de un programa se almacena dentro de un conjunto de ficheros. Estos archivos los edita el programador (vosotros) para crear o modificar el programa. Para los programas más sencillos basta con un único fichero, pero para los más complejos pueden hacer falta más de uno.

Los lenguajes de programación se pueden clasificar en diferentes categorías según sus características. De hecho, algunas de las propiedades del lenguaje de programación tienen consecuencias importantes sobre el proceso que hay que seguir para poder crear un programa y para ejecutarlo. Hay

dos maneras de clasificar los lenguajes de programación (estas dos categorías no son mutuamente excluyentes):

- Según si se trata de un **lenguaje compilado o interpretado**. Esta propiedad afecta los pasos que hay que seguir para llegar a obtener un fichero ejecutable. O sea, un fichero con el mismo formato que el de las aplicaciones que podéis tener instaladas a vuestro ordenador.
- Según si se trata de un **lenguaje de nivel alto o bajo**. Esta propiedad indica el grado de abstracción del programa y si sus instrucciones están más o menos estrechamente vinculadas al funcionamiento del hardware de un ordenador.

1.3.1 ...en lenguaje máquina

El lenguaje máquina o código máquina es el lenguaje que elegiríamos si quisiéramos hacer un programa que trabajara directamente sobre el procesador. Es interesante de conocer porque ayuda a entender el proceso de generación de un programa.

En este lenguaje, cada una de las instrucciones se representa con una secuencia binaria, en ceros (0) y unos (1), y todo el conjunto de instrucciones del programa queda almacenado de manera consecutiva dentro de un fichero de datos en binario. Si lo intentáis abrir con un editor de texto, lo que veréis en pantalla son símbolos totalmente incomprensibles.

> **Transistor**
>
> El transistor es el componente básico de un sistema digital. Se puede considerar como un interruptor, en qué 1 indica que pasa corriendo y 0 que no pasa.

Cuando se pide la ejecución de un programa en lenguaje máquina, este se carga a la memoria del ordenador. A continuación, el procesador va leyendo una por una cada una de las instrucciones, las descodifica y las convierte en señales eléctricas de control sobre los elementos del ordenador para que hagan la tarea pedida. A muy bajo nivel, casi se puede llegar a establecer una correspondencia entre los 0 y 1 de cada instrucción y el estado resultante de los transistores dentro de los chips internos del procesador.

El conjunto de instrucciones que es capaz de descodificar correctamente un procesador y convertir en señales de control es específico para cada modelo y está definido por su fabricante. El diseñador de cada procesador se inventó la sintaxis y las instrucciones del código máquina de acuerdo con sus necesidades cuando diseñó el hardware. Por lo tanto, las instrucciones en formato binario que puede descodificar un tipo de procesador pueden ser totalmente incompatibles con las que puede descodificar otro. Esto es lógico, puesto que sus circuitos son diferentes y, por lo tanto, las señales eléctricas de control que tiene que generar son diferentes para cada caso. Dos secuencias de 0 y 1 iguales

pueden tener efectos totalmente diferentes en dos procesadores de modelos diferentes, o resultar incomprensibles para alguno.

> Un programa escrito en lenguaje de máquina es específico para un tipo de procesador concreto. No se puede ejecutar sobre ninguno otro procesador, salvo que sean compatibles. Un procesador concreto sólo entiende directamente el lenguaje de máquina especificado por su fabricante.

A pesar de que, como se puede ver, en realidad hay muchos lenguajes máquina diferentes, se usa esta terminología para englobarlos a todos. Si se quiere concretar más se puede decir "lenguaje máquina del procesador X".

Ahora bien, estrictamente hablando, si optarais para hacer un programa en lenguaje de máquina, nunca lo haríais generando directamente ficheros con todo secuencias binarias. Sólo os tenéis que imaginar el aspecto de un programa de este tipo en formato imprimido, consistente en una enorme retahíla de 0 y 1. Sería totalmente incomprensible y prácticamente imposible de analizar. En realidad el que se usa es un sistema auxiliar de mnemotécnicos en el cual se asigna a cada instrucción en binario un identificador en formato de texto legible, más fácilmente comprensible para los humanos. De este modo, es posible generar un programa a partir de ficheros en formado texto.

> Esta compilación de mnemotécnicos es el que se conoce como el **lenguaje ensamblador.**

A título ilustrativo, la **tabla.2** muestra las diferencias de aspecto entre un lenguaje máquina y ensamblador equivalentes para un procesador de modelo 6502. Sin entrar en más detalles, es importante mencionar que tanto en lenguaje máquina como en ensamblador cada una de las instrucciones se corresponde a una tarea muy simple sobre uno de sus componentes. Hacer que el ordenador haga tareas complejas implica tener que generar muchas instrucciones en estos lenguajes.

Tabla 2. Tabla de equivalencia entre lenguaje ensamblador i lenguaje máquina.

Instrucción ensamblador	Lenguaje máquina
LDA #6	1010100100000110
CMP &3500	110011010000000000110101
LDA &70	1010010101110000
INX	11101111

1.3.2 ...mediante un lenguaje compilado

> **Editores de texto simples**
>
> Un editor de texto simple es aquel que permite escribir sólo texto sin formato. Son ejemplos el Bloc de Notas (Windows), Gedit o Emacs (Unix).

Para crear un programa lo que haremos es crear un archivo y escribir el conjunto de instrucciones que queremos que el ordenador ejecute. Para empezar será suficiente con un editor de texto simple.

> Una vez se ha acabado de escribir el programa, el conjunto de ficheros de texto resultante donde se encuentran las instrucciones que contiene se llama el **código fuente**.

Este sistema de programar más cómodo para los humanos hace surgir un problema, y es que los ficheros de código fuente no contienen lenguaje máquina y, por lo tanto, resultan incomprensibles para el procesador. No se le puede pedir que lo ejecute directamente; esto sólo es posible usando lenguaje máquina. Para poder generar código máquina hay que hacer un proceso de traducción desde los mnemotécnicos que contiene cada fichero a las secuencias binarias que entiende el procesador.

> El proceso denominado **compilación** es la traducción del código fuente de los ficheros del programa en ficheros en formato binario que contienen las instrucciones en un formato que el procesador puede entender. El contenido de estos ficheros se denomina **código objeto**. El programa que hace este proceso se denomina **compilador**.

Para el caso del ensamblador el proceso de compilación es muy sencillo, puesto que es una mera traducción inmediata de cada mnemotécnico a la secuencia binaria que le corresponde. En principio, con esto ya habría bastante para poder hacer cualquier programa, pero ceñirse sólo al uso de lenguaje ensamblador comporta ciertas ventajas e inconvenientes que hacen que en realidad no sea usado normalmente, sólo en casos muy concretos.

Por el lado positivo, con ensamblador el programador tiene control absoluto del hardware del ordenador a nivel muy bajo. Prácticamente se puede decir que controla cada señal eléctrica y los valores de los transistores dentro de los chips. Esto permite llegar a hacer programas muy eficientes en que el ordenador hace exactamente aquello que le decís sin ningún tipo de ambigüedad. En contraposición, los programas en ensamblador sólo funcionan para un tipo de procesador concreto, no son portables. Si se tienen que hacer para otra arquitectura, normalmente hay que empezar de cero. Además —y es el motivo de más peso para pensárselo dos veces si se quiere usar este lenguaje— crear un programa complejo requiere un grado enorme de experiencia sobre cómo funciona el hardware del procesador, y el trabajo sería considerable. Esto hace que sean programas complicados de entender y que haya que dedicar mucho tiempo a hacerlos.

1.3.2.1 Lenguajes compilados de alto nivel

> **Programas de bajo nivel**
>
> Se considera que el código máquina y el ensamblador son los lenguajes de nivel más bajo existentes, puesto que sus instrucciones dependen directamente del tipo de procesador.

Actualmente, para generar la inmensa mayoría de programas se utilizan los llamados lenguajes de alto nivel. Estos ofrecen un conjunto de instrucciones que son fáciles de entender para el ser humano y, por lo tanto, poseen un grado de abstracción más alto que el lenguaje ensamblador (puesto que no están vinculados a un modelo de procesador concreto). Cada una de las instrucciones se corresponde a una orden genérica en qué lo más importante es su aspecto funcional (qué se quiere hacer), sin que importe como se materializa esto en el hardware del ordenador ni mucho menos en señales eléctricas. En cualquier caso, hay que advertir que esta clasificación no siempre es absoluta. Se puede decir que un lenguaje es de "nivel más alto o bajo que otro", según el grado relativo de abstracción de sus instrucciones y su proximidad al funcionamiento interno del hardware de un ordenador.

El proceso para generar un programa a partir de un lenguaje de nivel alto es muy parecido al que hay que seguir para hacerlo usando el lenguaje ensamblador, puesto que las instrucciones también se escriben en formado texto dentro de ficheros de código fuente. La ventaja adicional es que el formato y la sintaxis ya no están ligados al procesador, y por lo tanto, pueden tener el formato que quiera el inventor del lenguaje sin que tenga que tener en cuenta el hardware de los ordenadores donde se ejecutará. Normalmente, las instrucciones y la sintaxis han sido elegidas para facilitar la tarea de creación y comprensión del código fuente del programa.

De hecho, en los lenguajes de nivel alto más frecuentes, entre los cuales está el que aprenderéis a usar en este módulo, las instrucciones dentro de un programa se escriben como una secuencia de sentencias.

> Una **sentencia** es el elemento mínimo de un lenguaje de programación, a menudo identificado por una cadena de texto especial, que sirve para describir exactamente una acción que el programa tiene que hacer.

Por lo tanto, a partir de ahora se usará el término sentencia en lugar de instrucción cuando el texto se refiera a un lenguaje de este tipo.

Una vez se han acabado de generar los ficheros con su código fuente, estos también se tienen que compilar para traducirlos a código objeto. Ahora bien, en este caso, el proceso de traducción a código objeto será bastante más complicado que desde ensamblador. El compilador de un lenguaje de nivel alto es un programa mucho más complejo. En cuanto al proceso de compilación, una consecuencia adicional del hecho de que el lenguaje no dependa directamente del tipo de procesador es que desde un mismo código fuente se puede generar código objeto para diferentes procesadores. Sólo hay que

disponer de un compilador diferente para cada tipo de procesador que se quiera soportar. Por lo tanto, un mismo código fuente original puede servir para generar programas que funcionen con diferentes tipos de procesador sin tenerlo que modificar cada vez.

Puesto que para cada fichero de código fuente se genera un fichero de código objeto, después del proceso de compilación hay un paso adicional llamado enlazamiento (*link*), en el cual estos dos códigos se combinan para generar un único fichero ejecutable. Coloquialmente, cuando os pedimos que compiláis un programa ya se suele dar por hecho que también se enlazará, si se tercia. Aún así, formalmente se consideran dos pasos diferenciados.

La siguiente figura muestra un esquema que sirve de resumen del proceso de generación del fichero ejecutable usando un lenguaje compilado.

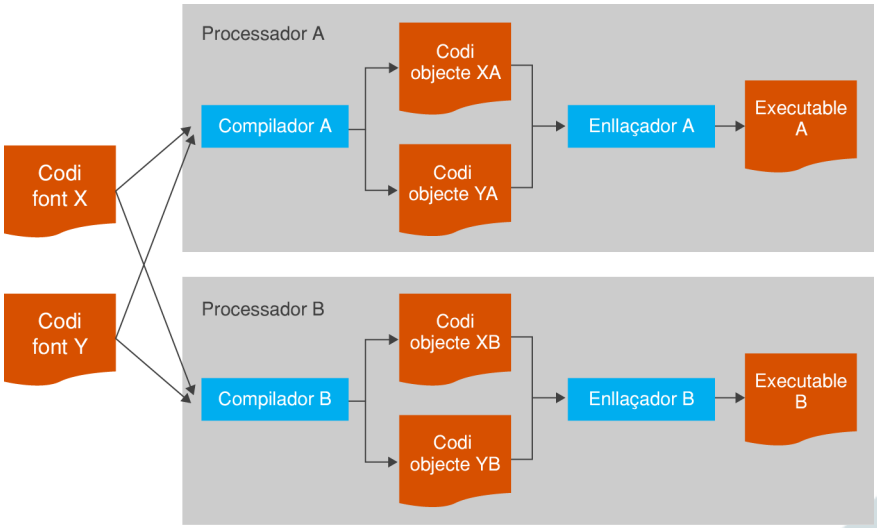


Figure 2: Proceso de compilación (y enlazamiento) del código fuente

Algunos ejemplos de lenguajes de nivel alto compilados muy populares son C o Pascal. Cómo se ha visto, el ensamblador también es un lenguaje compilado, pero de nivel bajo.

1.3.2.2 Errores de compilación El compilador es fundamental para generar un programa en un lenguaje compilado, ya sea de nivel alto o bajo. Para poder hacer su trabajo de manera satisfactoria y generar código objeto a partir del código fuente hace falta que las instrucciones sigan perfectamente la sintaxis del lenguaje elegido. Por ejemplo, hay que usar sólo las instrucciones especificadas en el lenguaje y hacerlo en el formato adecuado. Si no es así, el compilador es incapaz de entender la orden que se quiere dar al ordenador y no sabe como traducirla a lenguaje máquina.

> Cuando el compilador detecta que una parte del código fuente no sigue las normas del lenguaje, el proceso de compilación se interrumpe y anuncia que hay un **error de compilación**

Cuando pasa esto, habrá que repasar el código fuente e intentar averiguar donde está el error. Normalmente, el compilador da algún mensaje sobre lo que considera que está mal.

Hay que ser conscientes que un programador puede llegar a dedicar una buena parte del tiempo de la generación del programa a la resolución de errores de compilación. Ahora bien, que un programa compile correctamente sólo quiere decir que se ha escrito de acuerdo con las normas del lenguaje de programación, pero no aporta ninguna garantía que sea correcto, es decir, que haga correctamente la tarea para la cual se ha ideado.

> **Los lenguajes de programación y el lenguaje natural**
>
> Intentando hacer un símil entre un lenguaje de programación y el lenguaje natural, si una persona que habla castellano es como un compilador, que es capaz de entender o traducir una frase siempre que se sigan las normas de esta lengua, sin un poco de imaginación se le puede hacer difícil entender la frase: "helado kérem un comra". Hay palabras en un orden extraño, y además hay otras que no pertenecen al castellano o que no simplemente no existen…
>
> Por otro lado, la frase "El helado conduce una hoja de papel" puede ser gramaticalmente correcta y no tener ningún error de sintaxis. Alguien que hable castellano la puede entender. Ahora bien, está claro que algo no encaja. En la comprensión del significado de un lenguaje hay aspectos que van más allá de la sintaxis, y los lenguajes de programación no son excepción.

1.3.3 … mediante un lenguaje interpretado

En contraposición de los lenguajes compilados, tenemos los lenguajes interpretados. En este caso, no se hace una distinción interna entre nivel alto y bajo, puesto que la inmensa mayoría de lenguajes interpretados son de nivel alto. Lo que interesa es entender la idea general del funcionamiento y las diferencias con los compilados. Cómo en el caso de los lenguajes compilados, los programas también se escriben en ficheros de texto que contienen código fuente. La divergencia surge inmediatamente después de acabar de escribirlos, en la manera como se genera un fichero ejecutable. El quid de la cuestión es que, precisamente, ni se genera ningún código objeto ni ningún fichero ejecutable. Se trabaja directamente con el fichero de código fuente. Una vez este está escrito, el proceso de creación del programa ejecutable ha finalizado.

> **Intérprete**
>
> Alerta, un intérprete no traduce el código fuente del programa a código objeto y entonces lo eje-

> cuta. Lo que hace es ejecutar diferentes instrucciones de su propio código según cada instrucción leída del código fuente.

Imagináis un programa que acepta una serie de datos que codifican unas instrucciones, las va leyendo una por una y las va procesando de forma que actúa de una forma o de otra, es decir, ejecuta una parte u otra de su propio código objeto según el tipo de instrucción leída. A fin de cuentas, sería un programa que imita el comportamiento de un procesador, pero a escala de software. Pues esto es exactamente un intérprete.

> Un lenguaje interpretado se ejecuta indirectamente, mediante la ayuda de un programa auxiliar llamado intérprete, que procesa el código fuente y gestiona la ejecución.

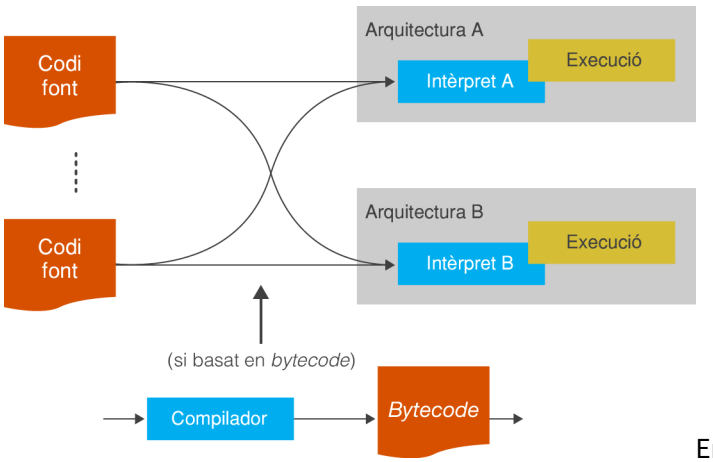
Igual que en los lenguajes compilados, puede suceder que el programador haya incluido sin darse cuenta algún error de sintaxis en las instrucciones. En este caso, será el intérprete quién mostrará el error y se negará a ejecutar el programa hasta que haya sido solucionado.

> Coloquialmente, la generación de bytecode a partir del código fuente se denomina igualmente **compilar**.

Algunos lenguajes interpretados usan una aproximación híbrida. El código fuente se compila y como resultado se genera un fichero de datos binarios llamados **bytecode**. Este bytecode, no obstante, no es formalmente código objeto, puesto que no es capaz de entenderlo el hardware de ningún procesador. Sólo un intérprete lo puede procesar y ejecutar. Simplemente es una manera de almacenar más eficiente y en menos espacio, en formato binario y no en texto, las instrucciones incluidas en el código fuente. Este es el motivo por el cual, a pesar de necesitar un proceso de compilación, estos lenguajes no se consideran realmente compilados y se continúan clasificando como interpretados.

Por sus características, los lenguajes interpretados no requieren un proceso posterior de enlazamiento.

La siguiente figura muestra un esquema del proceso de ejecución de un programa en lenguaje interpretado. Notad que en el caso de un lenguaje con bytecode, lo que se proporciona al intérprete son ficheros con la versión del código fuente previamente compilado en bytecode, y no el código fuente directamente.



Entre los lenguajes interpretados más conocidos encontramos Javascript, PHP o Perl. Muchos son lenguajes de script, que permiten el control de aplicaciones dentro de un sistema operativo, llevar a cabo procesos por lotes (batch) o generar dinámicamente contenido web. Entre los lenguajes interpretados basados en bytecode, Python y Java son los más populares.

1.4 Entornos integrados de desarrollo

> Un **IDE** (Integrated Development Environment o en torno integrado de desarrollo) es una herramienta que integra todo lo que hace falta para generar programas de ordenador, de forma que el trabajo sea mucho más cómodo.

Una vez se ha descrito el proceso general para desarrollar y llegar a ejecutar un programa, se hace evidente que hay que tener instalados y correctamente configurados dos programas completamente diferentes e independientes en vuestro ordenador para desarrollarlos: editor, por un lado, y compilador (incluyendo el enlazador) o intérprete por la otra, según el tipo de lenguaje. Cada vez que queráis modificar y probar vuestro programa tendréis que ir alternando ejecuciones entre los dos. Realmente, sería mucho más cómodo si todo ello se pudiera hacer desde un único programa, que integrara los tres. Un editor avanzado desde el cual se pueda compilar, enlazar si se tercia, e iniciar la ejecución de código fuente para comprobar si funciona.

> **Ejemplos de IDE**
>
> Algunos ejemplos de IDE son Visual Studio, para cualquier y Visual Basic; Netbeans y Eclipse, para los lenguajes Java y Ruby; Dev-Pascal, para el lenguaje Pascal, o el Dev-C, para el lenguaje C.

La utilización de estas herramientas agiliza increíblemente el trabajo del programador. Además, los IDE más modernos van más allá de integrar editor, compilador y enlazador o intérprete, y aportan otras características que hacen todavía más eficiente la tarea de programar. Por ejemplo:

- Posibilidad de hacer resaltar con códigos de colores los diferentes tipos de instrucciones o aspectos relevantes de la sintaxis del lenguaje soportado, para facilitar la comprensión del código fuente.
- Acceso a documentación y ayuda contextual sobre las instrucciones y sintaxis de los lenguajes soportados.
- Detección, y en algunos casos incluso corrección, automática de errores de sintaxis en el código, de manera similar a un procesador de texto. Así, no hay que compilar para saber que el programa está mal.
- Apoyo simultáneo del desarrollo de lenguajes de programación diferentes.
- Un depurador, una herramienta muy útil que permite pausar la ejecución del programa en cualquier momento o hacerla instrucción por instrucción, de forma que permite analizar como funciona el programa y detectar errores.
- En los más avanzados, sistemas de ayuda para la creación de interfaces gráficas.

En definitiva, usar un IDE para desarrollar programas es una opción muy recomendable. Aún así, hay que tener presente que son programas más complejos que un simple editor de texto y, como pasaría con cualquiera otro programa, hay que dedicar un cierto tiempo a familiarizarse con estos y con las opciones de que disponen.

1.5 Elementos principales de un programa

Los elementos principales de un programa informático clásico se dividen en varios grupos dependiendo del paradigma, del lenguaje y de la finalidad. Por ejemplo, en C es necesaria la existencia de un punto de entrada llamado main, o en los primeros lenguajes de servidor como PHP se ejecutaba el fichero completo. A pesar de la variedad de elementos de un programa, prácticamente todos poseen los siguientes bloques:

- **Bloque de declaraciones**. Incluye la declaración y normalmente la instanciación de todos los objetos y elementos a procesar como constantes o variables.
- Bloque de instrucciones

  Acciones sobre los elementos definidos en el bloque de declaración que permiten lograr el objetivo del programa. Dentro de este se puede diferenciar a su vez 3 grupos de instrucciones:

  - **Entrada**: Su función es obtener la información aportada desde el exterior necesaria para realizar el procesamiento. Esta información se almacena en los elementos definidos en el bloque de declaraciones.
  - **Proceso**: Instrucciones cuya finalidad es alcanzar el objetivo del programa a partir de la información proporcionada en la entrada.

  - **Salida**. Una vez realizado los cálculos este bloque se encarga de almacenar, por ejemplo en un fichero o en una base de datos o mostrarlo en algún dispositivo de salida como un monitor o impresora.

En la mayoría de los lenguajes actuales, además de los bloques anteriores se tienen también:

- **Espacio de nombre o paquete.** Indica el ámbito o alcance del código de forma que se puedan encapsular para su posterior uso.
- **Bloque de uso de elementos externos**. Indica las clases o funciones externas que se van a utilizar en el programa, denominados librerías o paquetes dependiendo del lenguaje, por ejemplo C o Java y que posteriormente serán enlazados, por ejemplo en C se utiliza la palabra reservada #include o Java la palabra import.
- **Bloque de definición del fichero/clase** en el que se incluyen comentarios como el autor, el tipo de licencia, el uso o función del código o clase.

1.6 Vuestro primer programa

Hay diferentes lenguajes de programación, algunos realmente muy diferentes entre sí. Antes de seguir adelante hay que elegir uno que será el usado para practicar todos los conceptos de programación básica que veréis de ahora en adelante. Una vez aprendáis a programar en un lenguaje, dominar otros lenguajes os será muy fácil, puesto que muchos de los conceptos básicos, e incluso algunos aspectos de la sintaxis, se mantienen entre diferentes lenguajes de nivel alto.

Un lenguaje muy popular parecido a Java es el C. La sintaxis del Java está claramente basada en la de este lenguaje.

La siguiente gráfica muestra la relevancia de los lenguajes de programación existentes.

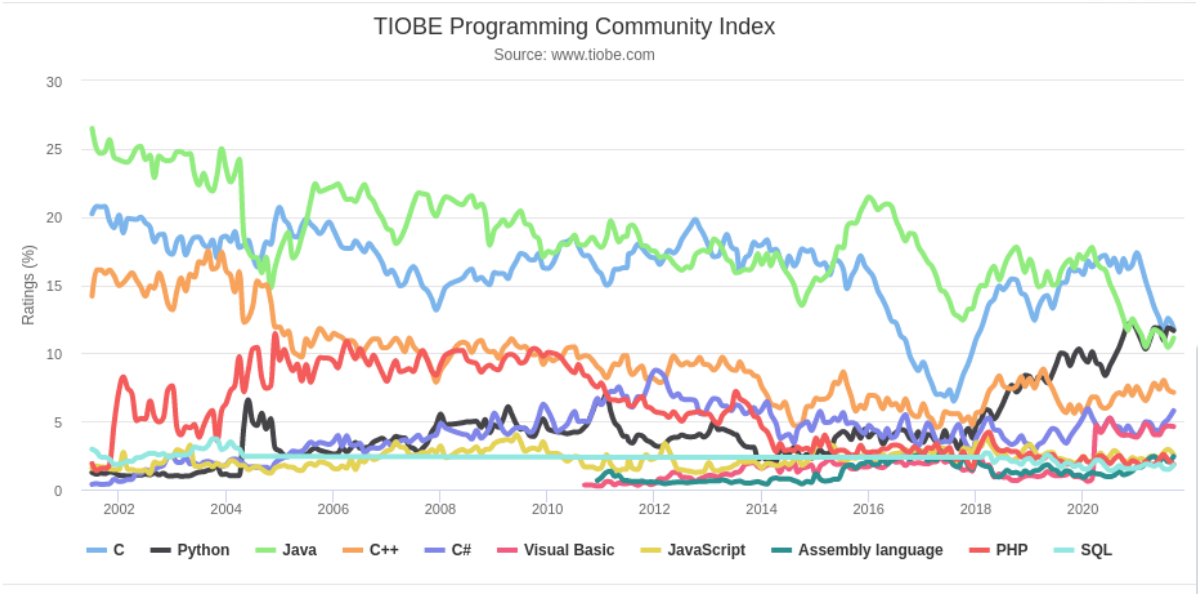


Figure 3: Índide TIOBE

Fuente: https://www.tiobe.com/tiobe-index/

1.7 Python

Durante las próximas sesiones aprenderemos los fundamentos de la programación mediante Python, lenguaje muy utilizado en ciberseguridad porque permite automatizar muchas tareas.

En el siguiente libro en HTML se explican estos fundamentos https://coherentpdf.com/python/index.html

Credits

IES El Caminàs