MEDAC
Instituto Oficial de Formación Profesional

ENTORNOS DE DESARROLLO

El software

1

ÍNDICE

OBJETIVOS

Identificar los lenguajes de programación.

Conocer los tipos de lenguajes de programación.

Conocer los requisitos hardware.

Conocer el concepto de programa informático.

/ 1. Introducción y contextualización práctica

A la hora de desarrollar una solución software es importante tener en cuenta la tecnología que vayamos a utilizar, así como el destino sobre el que se va a ejecutar la solución implementada.

E En la actualidad, a la hora de crear una aplicación, generalmente utilizamos lenguajes de alto nivel que se acercan al lenguaje natural, por lo que son más fáciles de utilizar que hace algunos años. Sin embargo, desde el punto de vista de la comunicación con el hardware, se sigue utilizando un modelo que es muy parecido al que se utilizaba hace algunos años, salvando algunas diferencias para adaptarse a los tiempos actuales.

Como podemos observar, ha existido una evolución paralela tanto de software como de hardware, sin embargo los principios básicos para la ejecución de los programas informáticos se sigue manteniendo.

Normalmente, los programas que se cargan en la CPU requieren una representación diferente a los lenguajes de alto nivel, pues éstos no son entendidos por el procesador.

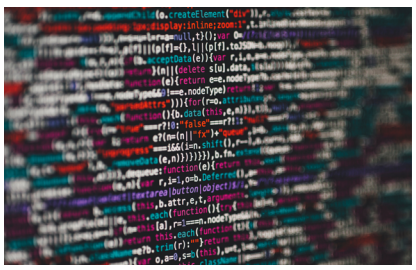

Fig. 1. El desarrollo software

En este tema nos centraremos en ver cómo se relaciona el software con el hardware para generar una representación que sea capaz de cargar a partir de un código fuente escrito en un lenguaje de alto nivel.

Escucha el siguiente audio en el que planteamos el caso práctico que iremos resolviendo a lo largo de esta unidad.

Audio intro. "Selección del paradigma de programación"

https://bit.ly/3kAoogA

/ 2. Programa informático

Podemos definir un programa informático como una receta detallada de cómo resolver un problema determinado. Para poder llegar a la solución en todos los casos, un programa informático no puede ser ambiguo. Además, debe especificar la secuencia de instrucciones lógicas necesarias para poder llegar a la solución buscada.

Un software, por el contrario, pudiera considerarse como un conjunto de programas informáticos que se combinan entre sí para un determinado fin. Este concepto es muy amplio, ya que software puede ser considerado desde una aplicación de facturas, la BIOS de un PC, un sistema operativo, o una máquina virtual que emula sistemas operativos completos dentro de otros.

Por otra parte, también hay que diferenciar entre programa informático y algoritmo, pues este último permite indicar todos los pasos necesarios para poder resolver un problema que no tiene por qué ser informático. Podemos decir que un programa informático está compuesto por diferentes algoritmos para poder resolver diferentes problemas, como, por ejemplo, la ordenación de una lista o encontrar el elemento más grande en un conjunto.

Para poder ejecutar un programa informático es necesario obtener un código que pueda ser comprendido por el procesador de la máquina en la que se está ejecutando. Por esta razón, cuando hablamos de programa informático, es necesario hacer referencia tanto al hardware como al software.

Nótese que para que un programa informático pueda ser ejecutado en la CPU es necesario que éste disponga de los datos e instrucciones necesarias para poder funcionar correctamente.

a. **Hardware y software en los equipos**

Cuando un programa informático va a ser ejecutado, el sistema operativo debe realizar la carga del programa, es decir, de sus datos e instrucciones en memoria principal (memoria RAM o volátil).

Una vez que el programa se encuentra en memoria principal, es posible realizar su carga en la CPU. Es importante destacar que todo sistema computacional actual sigue la arquitectura Von Neuman, con posibles modificaciones. De una manera sencilla, esta arquitectura especifica que todo computador requiere de una memoria principal, un sistema de entrada/salida y una CPU.

La CPU estaría formada por una unidad aritmético - lógica (ALU), registros y unidad de control.

Video 1. "Tipos de lenguajes de programación "

https://bit.ly/2Hwv5YQ

/ 3. ¿Qué es un lenguaje de programación?

Un lenguaje de programación está formado por un conjunto de sentencias y expresiones que permiten que el programador se comunique con la máquina de una manera sencilla. Los lenguajes de programación actuales se parecen al lenguaje natural, lo que facilita en gran medida su aprendizaje y utilización. Sin embargo, no se pueden ejecutar directamente sobre el procesador, sin realizar un proceso de traducción, conocido como compilación.

a. **Lenguajes de programación y nivel de abstracción**

Decimos que los lenguajes de programación actuales son lenguajes de alto nivel, entendiendo por ello el nivel de abstracción que poseen. El nivel de abstracción puede ser definido como la capacidad que tiene un individuo para poder obviar los detalles y aún así poder realizar las tareas encomendadas. De esta forma, cuando decimos que el

lenguaje de programación tiene un alto nivel de abstracción, hacemos referencia a lo que oculta al programador, pues éste no necesita conocer, por ejemplo, el conjunto de instrucciones máquinas disponibles en su CPU.

Además de los lenguajes de alto nivel (de abstracción), existen lenguajes que se acercan más a las características del hardware sobre el que se ejecutan.

Ejemplos de lenguajes de alto nivel (de abstracción) pueden ser Java, C++, Kotlin, etc. Ejemplos de lenguajes de nivel de abstracción media son C, Fortran, etc. El lenguaje ensamblador posee el nivel de abstracción más bajo, pues depende de las características de cada máquina.

A medida que los lenguajes de programación han ido evolucionando, los lenguajes que se consideraban de alto nivel, en la actualidad se consideran de nivel de abstracción medio, es por ejemplo el caso del lenguaje C.

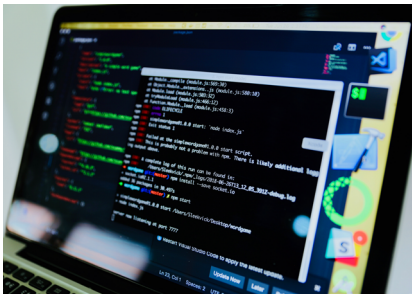

Fig. 2. Entorno de programación

Audio 1. "Especialización de lenguajes de programación"
https://bit.ly/31K3U6d

/ 4. Caso práctico 1: "Elección de lenguaje para canales de comunicación con clientes"

Planteamiento. Tras una reunión con el CTO de la empresa (Chief Technology Officer, máximo responsable técnico de los sistemas de información), a nuestro equipo se le asigna la tarea de crear una plataforma para el intercambio de experiencias para los últimos videojuegos que la empresa ha desarrollado.

Para ello, se decide crear tanto un portal web como diferentes aplicaciones móviles para que los usuarios puedan acceder por diferentes vías.

Nudo. ¿Cómo afrontarías este reto? ¿Qué tipo de lenguaje de programación utilizarías?

Desenlace. Para este tipo de entornos cabría preguntarse si sería una buena idea utilizar lenguajes de bajo nivel de abstracción. Tenemos que considerar que cuanto menor es el nivel de abstracción, más dependiente del procesador será la solución que creemos. Además, tenemos que considerar que vamos a crear un portal web y, al menos, dos aplicaciones móviles (Android y iOS).

Debido a ello, el mejor enfoque es utilizar lenguajes que tengan un alto nivel de abstracción y nos "oculten" todos los problemas que podemos encontrar con la dependencia de la plataforma sobre la que se va a ejecutar nuestro código.

Un posible enfoque sería utilizar tecnología web para crear el portal, con lenguajes como HMTL, CSS y JavaScript para el front-end y PHP o Java para el servidor.

Incluso, en este sentido podríamos utilizar herramientas de gestor de contenido como Drupal o Wordpress. Una ventaja de estas herramientas es que ofrecen la posibilidad de realizar consultas directamente a través de un API Rest desde el móvil.

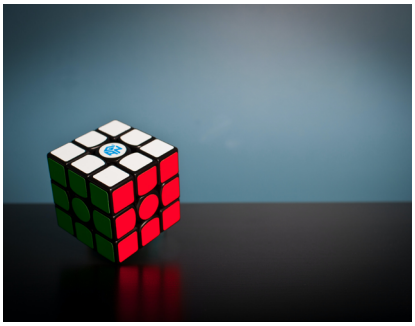

Fig. 3. Encontrar la solución en el software es como un puzle

/ 5. Paradigmas de programación

Una de las formas más utilizadas para la clasificación de los lenguajes de programación es el paradigma de programación utilizado. Generalmente, se suelen diferenciar dos grandes grupos: lenguajes imperativos y lenguajes declarativos.

Por un lado, los lenguajes **imperativos** permiten crear algoritmos a través del conjunto de instrucciones y expresiones del lenguaje. Por otro lado, los lenguajes **declarativos** solo especifican los valores que se esperan al final de un flujo, sin especificar qué realizar para llegar a este resultado. De esta forma, en los lenguajes declarativos es necesario indicar las relaciones lógicas y matemáticas para poder llegar al resultado.

Tradicionalmente, y debido a la forma de programar del ser humano, los lenguajes imperativos son más utilizados. Dentro de los lenguajes imperativos podemos diferenciar algunos subparadigmas como la programación orientada objetos, programación procedimiental, etc. Dentro del paradigma declarativo, podemos encontrar otros subtipos como el funcional, lógico, etc.

En la actualidad, muchos de los nuevos lenguajes que surgen son lenguajes imperativos que incorporan nuevas funcionalidades que, curiosamente, provienen de la programación funcional. Por ejemplo, tanto Kotlin como Swift, utilizados para el desarrollo de aplicaciones Android y iOS respectivamente, incorporan el concepto de funciones lambdas. Este tipo de funciones se denominan funciones de orden superior en la programación funcional. Como podemos ver, un paradigma de programación puede utilizar ideas de otro para mejorar y facilitar la interacción con el programador.

Fig. 4. Código de ejemplo

/ 6. Fases del proceso de compilación

Como los lenguajes compilados son los más utilizados en la actualidad, en este apartado nos centraremos en analizar las diferentes fases por las que pasa un código fuente hasta generar el código objeto y posteriormente al código ejecutable.

Como verás a lo largo de muchas asignaturas del ciclo:

- **El código fuente** es aquel que hemos escrito en el lenguaje de programación que hayamos utilizado.
- **El código objeto** es el resultado de finalizar el proceso de compilación.
- **El código ejecutable** es aquel que resulta de enlazar el código objeto con las librerías necesarias. De esta forma el código ejecutable es el único que puede 'entrar' en el procesador para ser ejecutado.

Para llegar al código ejecutable el compilador debe de realizar las siguientes etapas o fases:

1. **Análisis léxico**. Esta fase está centrada en comprobar que los elementos que se utilizan en la entrada de nuestro archivo pertenecen al lenguaje de programación que estamos utilizando.
2. **Análisis sintáctico.** Permite determinar si los elementos que provienen del analizador léxico vienen en el orden correcto.
3. **Análisis semántico.** Se centra en determinar si las sentencias escritas por el programador tienen sentido.
4. **Generación de código intermedio**. Una vez que finalizan todas las fases de análisis se supone que el código del programador no tienen ningún error previo, por lo tanto se puede generar una representación intermedia. Nótese que esta fase es opcional. La representación intermedia es independiente del procesador en el que se va a ejecutar el programa.

5. **Optimización de código intermedio**. Al igual que en la fase anterior se trata de una fase opcional cuyo objetivo es intentar mejorar el código generado para tener un mejor rendimiento.

6. **Generación de código final.** En esta fase somos capaces de generar código objeto que dependerá del conjunto de instrucciones de la CPU utilizada.

Video 2. "Proceso de compilación"
https://bit.ly/2NDBKEf

/ 7. Caso práctico 2: "Lenguaje para videojuegos"

Planteamiento. En nuestra empresa de videojuegos se nos plantea la posibilidad de comenzar un nuevo proyecto para la creación de un juego que tiene todos los ingredientes para ser el juego del año. Por ello, dado que somos el equipo con más experiencia, se nos asigna dicho proyecto.

Nudo. ¿Qué tipo de lenguaje utilizarías para esta solución? ¿Sería mejor un lenguaje imperativo o un lenguaje declarativo?

Desenlace. Según hemos visto en el tema, los lenguajes imperativos son los más utilizados, pues los lenguajes declarativos se vinculan mucho con la programación lógica y la inteligencia artificial.

Dicho esto, cabe destacar que un videojuego está formado por muchos módulos, por ejemplo, el motor gráfico, la estructura de datos interna, el motor de inteligencia artificial, etc.

En este contexto, podemos ver que puede que tengamos que utilizar ambos tipos de paradigmas.

Por ejemplo, lo más adecuado sería utilizar lenguajes orientados a objetos como C++ para poder crear el motor gráfico y las estructuras de datos.

En ocasiones, para elementos que requieran mejor rendimiento sería posible utilizar el lenguaje C (con menor nivel de abstracción) pero que nos daría un mejor rendimiento.

Para el motor de inteligencia artificial podríamos utilizar cualquier lenguaje declarativo que permita crear la inteligencia artificial. Un lenguaje típico de ello podría ser Haskel.

Fig 5. La construcción de un videojuego requiere de muchas capas

/ 8. Resumen y resolución del caso práctico de la unidad

A lo largo de este tema hemos presentado las principales características de los **lenguajes de programación** actuales. Existen diferentes tipos de lenguajes acorde a los diferentes tipos de dimensiones que utilicemos para medirlo.

Hemos visto que los lenguajes pueden ser de **alto, medio o bajo nivel de abstracción** atendiendo a lo cercano que esté al lenguaje natural humano.

Además, hemos presentado el concepto de **paradigma de programación**, diferenciando dos grandes grupos. Por un lado tendríamos el paradigma de programación imperativa y por otro lado el paradigma de programación declarativa. Como hemos visto en el tema, en la actualidad, el paradigma más utilizado es el imperativo, sin embargo, cada vez es más común ver como se adoptan funcionalidades del paradigma declarativo.

También hemos presentado el **modelo de arquitectura** más utilizado en los computadores creado por Von Neuman. Aunque este modelo arquitectónico fue creado hace muchos años sigue siendo utilizado en la actualidad aunque con algunas modificaciones.

Para finalizar hemos conocido las diferentes fases por la que pasa el proceso de compilación a la hora de generar un código que pueda ser ejecutado por el procesador.

Resolución del caso práctico de la unidad

Como hemos visto en este tema, existen diferentes paradigmas de programación que se pueden ajustar en mayor o menor medida a diferentes tipos de problemas. Esto quiere decir que no existe un paradigma mejor que otro, pues lo que tenemos que considerar es el problema por resolver y ajustar el lenguaje que utilizamos al problema en cuestión.

En el caso que se nos plantea no tiene mucho sentido utilizar la orientación a objetos en un campo relacionado con la inteligencia artificial. Como hemos estudiado en la unidad, para este tipo de problemas es mejor utilizar un paradigma declarativo, ya que estos lenguajes están mejor diseñados para ello.

/ 9. Bibliografía

Villalba, C., Moraleda, A. & lez, M. (2011). Lenguajes de programación. Madrid: Universidad Nacional de Educación a Distancia.

Aho, A., Sethi, R., Ullman, J., Suárez, P. & López, P. (1998). Compiladores: principios, técnicas y herramientas. México: Addison-Wesley.

Alexander, A. (2017). Functional programming, simplified: simple, step-by-step approach to learning functional programming. Boulder, Colo: Alvin Alexander.