Clase 10: Lenguajes y paradigmas de programación.

Lenguajes de programación.

Los sistemas carecen de la capacidad de interpretar cosas ambiguas. Por esto, es importante que las instrucciones que le damos estén conformadas por una serie de pasos llamados algoritmos y se den a través de un lenguaje formal, para  evitar malentendidos.

Los lenguajes formales que usamos para dar estas instrucciones se llaman lenguajes de programación y cambian con el tiempo.

Los lenguajes pueden dividirse en distintas categorías:

* Según el contexto.
  + Específicos: resuelven problemas puntuales.
  + Generales: Permiten realizar una serie de aplicaciones distintas independientemente del contexto.
* Según su similitud al lenguaje humano:
  + Alto nivel: Cómo JavaScript. Estos lenguajes están abstraídos de las cosas internas de la máquina y permiten un rápido desarrollo.
  + Bajo nivel: Más cercanos al lenguaje binario. Son utilizados para dar instrucciones muy específicas Y para utilizar al máximo los recursos disponibles.
* Según su tipado:
  + Dinámico: La comprobación de tipificación se realiza durante su ejecución en vez de durante la compilación. Se ejecuta más lentamente y es más propenso a cometer errores. Puede ser fuerte o débil según si permite o no violaciones de los tipos de datos una vez declarado.
    - Fuerte: Nos obliga a indicar el tipo de dato al declarar la variable. Ventaja: El código es más expresivo y cometemos menos errores. Desventaja: Son más estrictos a la hora de programar y hay que escribir más código. Ejemplo: Python.
    - Débil: No indicamos en la mayoría de las veces el tipo de variable. Ventaja: es mucho más rápido para desarrollar. Desventaja: Podemos cometer muchos más errores. Ejemplo: Javascript.
  + Estático: La comprobación de la tipificación se realiza durante la compilación. Esto permite que la ejecución del programa sea más eficiente y segura.

Paradigmas de programación.

Década de los 60. Centro de Cómputo Noruego en Oslo. En ese momento utilizaban la programación estructurada, linea por linea. Las simulaciones eran extremadamente complejas y esta programación dificultaba más el trabajo. Para solucionar esto es que nace el paradigma de POO.

El paradigma es la forma de pensar bajo un modelo preestablecido.

Los paradigmas de programación son:

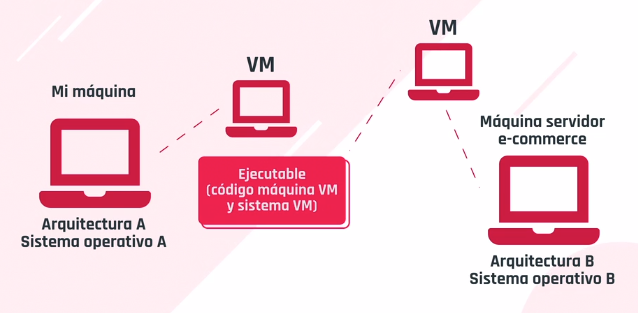
* Programación imperativa: Indica a la pc que debe hacer y en qué secuencia a través de instrucciones sucesivas.
  + Paradigma estructurado: Una instrucción a la vez y uno se rige por un acotado set de instrucciones. Es muy utilizado para el desarrollo de sistemas. Utiliza tres estructuras: Secuencia, estructura condicional (if y switch) y estructura de iteración (for y while).
  + Paradigma de programación orientado a objetos: Se utilizan abstracciones y se crean entidades, estas entidades (grupo de código) interpretan mensajes.
* Programación declarativa: Describe el resultado final que se busca.
  + Paradigma funcional: Basado en funciones matemáticas y principio de función. Propiedad de inmutabilidad propia del mismo, la función x siempre va a devolver el valor B cuando se le pase el valor A.
  + Paradigma lógico: En lugar de desarrollar pasos e instrucciones utiliza reglas lógicas de la matemática para consultar el sistema y el mismo infiere qué hacer en base a las reglas lógicas establecidas.
* Paradigma de programación con lenguaje específico de dominio: Engloba los lenguajes desarrollados para resolver un problema específico.
* Multiparadigma: Con la evolución de la programación han habido lenguajes que han modificado su estructura para poder permitir dar soluciones en distintos paradigmas. JS utiliza el paradigma estructurado, la programación orientada objetos e Incluso el paradigma funcional.

Del código al ejecutable.

Un programa puede estar conformado por muchos archivos escritos en distintos lenguajes.

Para que la máquina pueda interpretar todos estos lenguajes distintos primero los programas deben pasar del código que escribimos a un código que la máquina entienda, a esto se le llama **compilación**. La compilación toma todo el código fuente y lo transforma en el famoso programa ejecutable escrito en código máquina, y debería poder ejecutarse correctamente siempre y cuando la máquina donde se compile sea similar a la máquina donde se ejecute (SO y arquitectura del CPU similar).

Hay dos formas de que un programa sea comprendido correctamente por el sistema, independientemente de la arquitectura. La máquina virtual, donde la compilación se realiza de acuerdo a la arquitectura de la VM (virtual machine), y podrá ser ejecutado por otras VMs.



La otra forma es mediante un intérprete, donde el análisis se realiza linea por linea en cada sistema y se traduce en el momento a un código que la máquina entienda. En este proceso, el código fuente no es compilado anteriormente a código máquina para crear el ejecutable. El intérprete corre sobre una máquina virtual, por ejemplo la consola de google.

La diferencia entre estos procesos radica en la performance y el rendimiento. En la compilación, donde el código máquina que se obtiene es de acuerdo a la arquitectura del sistema, este se ejecuta velozmente. En la VM se obtiene portabilidad pero se agrega un intermediario. Y con la interpretación el proceso se realiza linea por linea, lo que lo vuelve más lento.

Código fuente: Colección de instrucciones de computadora escritas usando un lenguaje de programación legible por los humanos.

Código máquina: Resultado obtenido luego de la compilación, secuencia de sentencias en lenguaje máquina que puede ser comprendido por el procesador.

