LENGUAJES Y PARADIGMAS DE PROGRAMACIÓN

Al programar se debe ser claro y preciso con las instrucciones que damos. Estas instrucciones están conformadas por una serie de pasos llamados **algoritmos**. Estos algoritmos son interpretados por lenguajes de programación que se dividen en dos grupos:

- **Lenguajes específicos**
  - Resuelven problemas puntuales
    - Por ejemplo, un lenguaje de programación para realizar gráficos matemáticos
- **Lenguajes generales**
  - Permiten desarrollar una infinidad de aplicaciones distintas
    - Por ejemplo, un sitio web

Además, se clasifican en dos clases:

- **Lenguajes de alto nivel**
  - Son los que se encuentran más cercanos al lenguaje natural que al lenguaje máquina de 0 y 1
  - Nos permite escribir código de manera más natural y rápida para enfocarnos en funcionalidades interesantes
  - Abstracción de cosas internas (0 y 1) y permite desarrollo rápido
  - Ejemplo:
    - Javascript
- **Lenguajes de bajo nivel**
  - Son utilizados para dar instrucciones muy específicas y utilizar al máximo los recursos disponibles
    - Para ello debemos estar atentos, no solo a la funcionalidad que queremos desarrollar, sino también en qué hardware

TIPADO

Los lenguajes tipados **fuerte** y **débil** se distinguen según si permiten o no violaciones de los tipos de datos una vez declarados

- **Lenguajes de tipado débil**
  - En estos lenguajes no indicamos, la mayoría de veces, el tipo de variable.
    - Podemos asignar, por ejemplo, un valor entero a una variable que anteriormente tenía una cadena de texto
  - También podemos operar con variables de distintos tipos
  - Su mayor ventaja es que es mucho más rápida de desarrollar
  - Su desventaja es que podemos cometer muchos más errores si no tenemos cuidado

    - Ejemplos:
        - Javascript
        - Perl
        - Lisp
        - SWI Prolog
        - PHP
- **Lenguajes de tipado fuerte**
    - En estos lenguajes se nos obliga a indicar el tipo de dato al declarar una variable
        - Dicho tipo no puede ser cambiado una vez definida la variable
    - Su ventaja es que al ser código más expresivo, cometeremos menos errores
    - Su desventaja es que son mucho más estrictos a la hora de programar y hay que escribir mucho más código
    - Ejemplos:
        - C++
        - Java
        - Python
        - C#
        - Typescript
        - Go
- **Lenguajes de tipado estático**
    - La comprobación de tipificación **se realiza durante la compilación** y no durante la ejecución
    - Permite que los errores de tipificación sean detectados antes y que la ejecución del programa sea más eficiente y segura
    - Ejemplos:
        - C
        - C++
        - Java
        - Haskell
- **Lenguajes de tipado dinámico**
    - La comprobación de tipificación **se realiza durante su ejecución** y no durante la compilación
    - Este es más flexible, a pesar de ejecutarse más lentamente y ser más propenso a contener errores de programación
    - Ejemplos:
        - Javascript
        - Perl
        - Python
        - Lisp

FRAMEWORKS [MARCOS DE TRABAJO]

- Es una estructura previa/esqueleto que se puede aprovechar para desarrollar un proyecto
- Es una especia de plantilla, un esquema conceptual, que simplifica la elaboración de una tarea, ya que solo es necesario complementarlo de acuerdo a lo que se quiere realizar
- Ejemplos:
    - React
    - Angular
    - Spring MVC
    - Rails
    - Vue.js
    - Django
    - ExpressJS
    - Laravel

PARADIGMAS DE PROGRAMACIÓN

Un **paradigma** es una forma de pensar bajo un modelo preestablecido

- Un lenguaje es una herramienta y hay distintas herramientas para distintas soluciones. Siempre debemos analizar el contexto, tiempos, con qué equipo y herramientas contamos, si hay presupuesto y qué queremos lograr
- La mejor manera de conocer un paradigma de programación es investigar y programar en un lenguaje característico de ese paradigma. No hace falta ser un experto. Solo el hecho de conocerlo nos brinda más herramientas a la hora de desarrollar
- HISTORIA
    - Antes la programación era muy estructurada
        - El código se ejecutaba línea tras línea
        - Las simulaciones eran extremadamente complejas
    - Ole-Johan Dahi y Kristen Nygaard
        - Noruega, Oslo - Década de los 60 - Centro de cómputo
        - Trabajaban en simulaciones de naves
        - Decidieron codear con otro set de reglas que facilitaran las simulaciones
            - Allí nació el lenguaje de programación Simula 67 y el paradigma de programación orientado a objetos

- **Tipos**
    1. **Programación imperativa**
        - Indica a la computadora qué debe hacer y en qué secuencia, a través de instrucciones sucesivas
            - Paradigma estructurado

        - Sigue una línea de pensamiento donde se suele ejecutar una instrucción a la vez y uno se rige en un acotado set de instrucciones
        - Es muy utilizado para el desarrollo de sistemas
    - Paradigma de programación orientado a objetos
        - El código puede agruparse de tal forma que llegue a representar una entidad y que interprete mensajes
        - Su fortaleza yace en utilizar abstracciones y crear entidades.
2. **Programación declarativa**
    - Describe el resultado final que se busca -qué debe realizar el programa.
        - Paradigma funcional
            - Se basa en un concepto muy simple: el de las funciones matemáticas.
            - La fortaleza de este paradigma radica en que siempre que a la función X se le pasa el valor A, esta siempre va a devolver el valor B.
                - Esta propiedad de **devolver el mismo valor** se le conoce como **inmutabilidad** y es característico de este paradigma.
        - Paradigma lógico
            - En lugar de desarrollar pasos e instrucciones, utiliza reglas lógicas para consultar al sistema y él mismo infiere qué hacer con base a las reglas lógicas establecidas
                - Paradigma de programación lógica ---> Instrucciones ---> Reglas lógicas
3. **Paradigma de programación con lenguaje específico de dominio**
    - Los lenguajes que encontramos acá tratan de resolver problemáticas superespecíficas
4. **Multiparadigma**
    - Lenguajes de programación que pueden utilizar más de un paradigma

DEL CÓDIGO AL EJECUTABLE

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

- .js
- .html
- .css

- .java
- .rb

Gracias a la compilación es que la máquina puede interpretar y ejecutar nuestro programa

El primer compilador de la historia, el **A-0**, fue desarrollado en 1952 por la científica en computación Grace Hopper

- Código fuente
  - El código fuente es una colección de instrucciones de computadora escritas usando un lenguaje de programación legible por humanos
- Código de máquina
  - El código de máquina es una secuencia de sentencias en lenguaje de máquina o binario
  - Es el resultado obtenido después de que el compilador convierta el código fuente en un lenguaje que pueda ser comprendido por el procesador

FORMAS DE INTERPRETACIÓN DE CÓDIGO

- **Compilador**
  - Es una aplicación que traduce (compila) el código fuente en un código que el procesador puede comprender y ejecutar
  - Este código de máquina se almacena en forma de archivo ejecutable
  -
  - El resultado de la compilación debería poder ejecutarse correctamente, siempre y cuando la máquina donde se compile sea similar a donde se ejecute:
    - Tener una arquitectura de CPU similar
    - Tener un sistema operativo similar

- Hay otras dos formas para que los programas sean entendidos y ejecutados por una máquina, independientemente de la arquitectura
  - **Intérprete**
    - Traduce el código fuente línea a línea en cada sistema donde se ejecuta el código fuente, así lo traduce en determinado momento a código máquina que la misma entiende
      - Esto permite que un código programa pueda ser independiente de la arquitectura, ya que el código

fuente no es compilado previamente a código máquina para crear el ejecutable
        - El proceso de traducción funciona mucho más rápido que en un compilador, pero la ejecución es más lenta y se necesita una gran cantidad de memoria
    - **Máquinas virtuales** (Virtual Machine = VM)
        - Cuando escribimos un programa, el código fuente va a ser compilado a código máquina, pero no de nuestra máquina física, sino al código que entienda la VM
            - Va a poder ejercutarse en máquinas VM
            - Va a poder ejecutarse en máquinas con otros tipos de arquitectura y sistema operativo ya que hay una versión de la VM que se encarga de esto
                - No requerimos volver a compilarlo, sino que la VM va a realizar la traducción por nosotros al sistema en el que se ejecute
                - Estas VM son propias del lenguaje de programación y son actualizadas por empresas que se dedican a esto
- No podemos elegir cómo se va a ejecutar nuestro código ya que este viene de la mano de cómo el lenguaje de programación fue diseñado

PROS y CONTRAS DE LAS FORMAS DE INTERPRETACIÓN DE CÓDIGO
La diferencia radica en el **performance** y en el **rendimiento**

- Cuando los código fuente se **compilan** y se obtiene el código máquina específico de una arquitectura o sistema, este se ejecuta **velozmente**
- Con la ventaja de portabilidad de las **máquinas virtuales**, se tiene la **desventaja** de **bajo performance** ya que el código, en lugar de ejecutarse en la máquina específica, se va a ejecutar en esta máquina virtual que hace de intermediaria con la física
- Con los lenguajes **interpretados** tenemos la **desventaja** de que la traducción se va realizando línea por línea cada vez que se ejecuta y es ese proceso de traducción el que ralentiza el proceso de ejecución del código
- Si bien estos lenguajes que corren en una **máquina virtual o interpretados** suelen tener una **desventaja** en el **performance**, los mismos nos permiten escribir código que sabemos que van a funcionar independientemente de dónde corran