**Clase 10**

**Lenguaje de programación**

**Tipado de lenguajes y framework:**

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

**Tipado débil:**

En estos lenguajes no indicamos, la mayoría de las veces, el tipo de variable. Aquí podemos asignar, por ejemplo, un valor entero a una variable que anteriormente tenía una cadena. Pero, no solo eso, también podemos operar con variables de distintos tipos.

Su principal ventaja es que es mucho más rápido de desarrollar, pero una clara desventaja es que podemos cometer muchos más errores si no tenemos cuidado

*Ej: JavaScript, perl, Lisp, Swi Prolog, PHP*.

**Tipado fuerte:**

En estos lenguajes se nos obliga a indicar el tipo de dato al declarar la variable. Además, dicho tipo no puede ser cambiado una vez definida la variable.

La ventaja es que, al ser código más expresivo, cometemos menos errores.

La desventaja, que son mucho más estrictos a la hora de programar y que hay que escribir mucho más código.

*Ej: C++, Java, Python, C#, Type Script, Go.*

**Tipado estático:**

Aquí, la comprobación de tipificación se realiza durante la compilación y no durante la ejecución. Comparado con el tipado dinámico, el estático permite que los errores de tipificación sean detectados antes y que la ejecución del programa sea más eficiente y segura.

*Ej: C, C++, Java, Haskell*

**Tipado dinámico:**

La comprobación de tipificación se realiza durante su ejecución en vez de durante la compilación. Comparado con el tipado estático, este es más flexible, a pesar de ejecutarse más lentamente y ser más propenso a contener errores de programación.

*Ej: JavaScript, Perl, Python, Lisp*

**Frameworks (Marco de trabajo):**

Es una estructura previa / esqueleto que se puede aprovechar para desarrollar un proyecto.

El framework es una especie 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.

*Ej: React, Spring MVC, Vue.js, Express js, Angular, Rails, Django, Laravel.*

Las instrucciones están conformadas por una serie de pasos llamadas *algoritmos*, estos, los damos a través de un lenguaje formal que no permite que haya varias interpretaciones de lo que se quiere transmitir así podremos ser siempre claros y precisos.

Los lenguajes formales que utilizamos para ver estas instrucciones los llamamos lenguajes de programación.

Cada uno tiene su particularidad. En general nos encontramos con dos grandes grupos:

* **Lenguajes específicos**: resuelven problemas puntuales, como un lenguaje de programación para realizar gráficos matemáticos.
* **Lenguajes generales**: permiten desarrollar aplicaciones distintas, casi independiente del contexto como un sitio web o un comercio electrónico.

Los **lenguajes de alto nivel** son aquellos que se encuentran más cercanos al lenguaje natural que al lenguaje máquina de 0 y 1. Nos permite escribir código de manera mas natural y rápida para enfocarnos en funcionalidades interesantes. Se nos permite ser super eficientes a la hora de programar, ya que estamos abstraídos de las cosas internas de la máquina.

Los **lenguajes de bajo nivel** son usados para dar instrucciones muy específicas, y utilizar al máximo los recursos disponibles, pero para ello vamos a tener que estar atentos, no solo a la funcionalidad que queremos desarrollar, sino también en qué hardware.

**Paradigmas**

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

* **Paradigma estructurado**: sigue una línea de pensamientos donde se suele ejecutar una instrucción a la vez y uno se rige en un acotado set de instrucciones. Este paradigma es muy usado para el desarrollo de sistemas.

Ejemplo: una función “esPar” recibe un número y devuelve el mensaje “verdadero”, si el número es par, y “falso”, si es impar.

* **Paradigma de programación orientado a objetos**: el código puede agruparse de tal forma que lleve a representar una entidad y que interprete mensajes. La fortaleza del paradigma de la programación orientada a objetos yace en utilizar abstracciones y crear entidades.

Ejemplo: un código representa un carrito de compra. Otro código representa un producto con su precio. Luego, puedo agregarle la responsabilidad al carrito que vaya agregando productos para luego preguntarle el costo total.

* **Paradigma funcional**: se basa en un concepto muy simple y es 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. 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.

Ejemplo: la solución funcional al problema de si un número es par o no es muy similar al estructurado, debemos crear una función “esPar” que reciba un número y nos diga si es par o impar.

* Paradigma **lógico:** en lugar de desarrollar pasos e instrucciones, utiliza reglas lógicas para consultar al sistema y el mismo infiere que hacer en base a las reglas lógicas establecidas.

Ejemplo: reglas lógicas: toda persona cuyo saldo sea negativo es deudor. A todo deudor se le aplica una tasa de interés del 10%.

Con este set lógico podríamos preguntar: ¿Cuál es la tasa de interés de Juan? El sistema responde analizando si Juan es una persona, si es deudor o no y si aplica o no la tasa de interés.

* **Paradigma de programación con lenguaje específico de dominio**: los lenguajes que encontramos acá tratan de resolver problemáticas superespecíficas.

Ejemplo: cuando queremos consultar una base de datos de un supermercado para saber qué productos tenemos en la categoría de electrodomésticos.

* **Multiparadigma:** a lo largo de la evolución de la programación, con nuevos desafíos y paradigmas ha habido lenguajes que han modificado su estructura para permitir dar soluciones en distintos paradigmas.

Ejemplo: en JavaScript se puede escribir código tanto con el paradigma estructurado como con programación orientada a objetos e incluso utilizar el paradigma funcional.

Esto no quiere decir que mientras más paradigmas tenga un lenguaje es mejor. Un lenguaje es una herramienta y hay distintas herramientas para distintas soluciones. Siempre debemos analizar el contexto, tiempos, con qué equipo contamos, ¿Hay presupuesto? ¿cuáles son las herramientas que disponemos para trabajar? ¿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.

**Paradigmas de programación:**

* **Programación imperativa:** indica a la computadora qué debe hacer y en qué secuencia, a través de instrucciones sucesivas.
  + **Programación estructurada**: utiliza tres estructuras: secuencia, estructura condicional -if y switch- y estructuras de iteración -bucles, for y while-.
  + **Programación orientada a objetos**: utiliza objetos y la interacción entre ellos para resolver problemas.
* **Programación multiparadigma**: lenguajes de programación que pueden utilizar más de un paradigma.
* **Programación declarativa**: describe el resultado final que se busca -qué debe realizar un programa-.
  + **Programación lógica**: se basa en los conceptos de lógica matemática.
  + **Programación funcional**: basado en el concepto matemático y el principio de función.
* **Programación con lenguaje específico de dominio**: engloba a los lenguajes desarrollados para resolver un problema específico.

**Del código al ejecutable**

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

**Código fuente**: es una colección de instrucciones de computadora escritas usando un lenguaje de programación legible por humanos.

**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.

**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.

**Intérprete**: traduce el código fuente línea a línea y lo ejecuta directamente. 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.

¿Cómo hace la máquina para interpretar todos los distintos lenguajes que podamos llegar a utilizar en distintos archivos?

Las instrucciones, tienen que pasar del código que escribimos a un código que la máquina entienda. Esta traducción se llama compilación. La cual toma todo el código fuente y lo transforma en el programa ejecutable escrito en código máquina. Listo para ser ejecutado en nuestros dispositivos.

El resultado de la compilación (el ejecutable), debería poder ejecutarse en cualquier computadora, siempre y cuando, la maquina en donde se compile sea similar a donde se ejecute (Similar arquitectura del CPU, sistema operativo similar).

Hay otras dos formas para que los programas sean entendidos y ejecutados por una máquina independientemente por su arquitectura: *Máquinas virtuales e Interpretes*

Cuando el código fuente se compila y uno obtiene el código máquina especifico de una arquitectura de sistema, este se ejecuta velozmente.

Con la ventaja de portabilidad de las máquinas virtuales, tenemos la desventaja de que el código, en lugar de ejecutarse en la máquina específica se va a ejecutar en esta maquina 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 ese proceso de traducción, ralentiza la operación de ejecución del código.

Si bien estos lenguajes que corren en una máquina virtual o interpretados suelen tener una desventaja en la performance, los mismos nos permiten escribir códigos que sabemos que van a funcionar independientemente de donde corran, que es algo clave en la toma de decisión sobre en qué lenguaje desarrollar