CLASE 10 LENGUAJES Y PARADIGMAS DE PROGRAMACION

LENGUAJES DE PROGRAMACION:

Para poder traducir lo que queremos que nuestra app haga, tenemos que ser muy especifico al programar ya que tradicionalmente los sistemas carecen de inteligencia para interpretar cosas ambiguas. Por lo tanto, son muy importantes las instrucciones que le damos conformadas por una serie de pasos, llamados algoritmos. Estas instrucciones se dan a través de un lenguaje formal que no permite que haya varias interpretaciones de lo que queremos transmitir así podremos ser siempre claros y precisos. Los llamamos lenguajes de programación. Existe muchos lenguajes de programación y cada uno tiene su particularidad.   
Generalmente nos encontramos con dos grandes grupos: Específicos y generales. Los específicos resuelven problemas puntuales. Luego tenemos los lenguajes de programación generales, que permiten desarrollar una infinidad de apps distintas casi independiente del contexto como un sitio web de mascotas o un comercio electrónico.   
Hay lenguajes de alto nivel y bajo nivel. Los lenguajes de alto nivel son aquellos que se encuentran mas cercanos al lenguaje natural que al lenguaje binario, que nos permite escribir códigos de manera mas rápida(Javascript: Donde se nos permite ser super eficientes a la hora de programar ya que estamos abstraídos de las cosas internas de la maquina de esos cero y unos pudiendo concentrarnos en desarrollar funcionalidades y sistemas super interesantes.   
Los lenguajes de bajo nivel son utilizados para tareas muy especificas y utilizar al máximo los recursos disponibles. Pero tendríamos que estar atentos a que funcionalidad que queremos desarrollar sino tambien a que hardware.

Lenguajes de programacion:

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

1-Tipado débil: en estos lenguajes no indicamos el tipo de variable. Aquí podemos asignar, por ejemplo, un valor entero a una variable anteriormente tenia una cadena.   
Su principal ventaja es que es mucho más rápido de desarrollar.  
Una clara desventaja es que podemos cometer muchos errores si no tenemos cuidado.   
Javascript, perI, LISP, SWI Prolog, php.

2-Tipado fuerte: 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 mas expresivo, cometeremos menos errores.  
La desventaja es que son mucho mas estrictos a la hora de programar y que hay que escribir mucho más código.  
C++ Java, python, C#, TypeScript, Go

3-Tipado estático: 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.  
C, C++, Java, Haskell

4-Tipado dinámico: la comprobación de tipificación se realiza durante su ejecución en vez de durante la compilación.   
\*Comparado con el tipado estático, este es mas flexible, a pesar de ejecutarse mas lentamente y ser mas propenso a contener errores de programación.  
Javascript, perI, Python, LISP.

![Imagen que contiene Gráfico

Descripción generada automáticamente]()Posicionamiento de cada lenguaje de programación:

Frameworks: Marco de trabajo  
Es una estructura previa/esqueleto que se puede aprovechar para desarrollar un proyecto.  
Es una especie de plantilla, un esquema conceptual, que simplifica la elaboración de una tarea, ya que solo es necesario complementarlo de acuerdo con lo que se quiere realizar.

PARADIGMAS DE PROGRAMACION:

Década de los 60 y en el centro de computo de noruego en Oslo donde Ole-Johan Dahl y Kristen Nygaard trabajaron en simulaciones de naves para descubrir como se comportan cada una de sus partes al modificar sus cualidades. En su momento la forma de programar era muy estructurada y los lenguajes de programación seguían este modelo o paradigma en donde uno debía escribir funciones y el código se ejecutaba línea tras línea. Las simulaciones eran extremadamente complejas y abordarlas con el modelo de programación estructurada sumaba complejidad en vez de facilitar el desarrollo. Entonces Dahl y Nygaard pensaron, ¿Y si vemos el problema desde otra perspectiva? ¿Por qué tenemos que seguir el modelo estructurado? Podríamos armar otro modelo? Por ahí uno que nos permite representar los objetos de la nave? Entonces decidieron cambiar el modelo, escribir código pero con otro set de reglas en donde pudieron literalmente representar los distintos objetos que componen una nave. Cada objeto tenia un estado interno y funciones, así como una nave puede tener un estado de encendido y apagado, que una función permite encenderla o apagarla. Alli nació el lenguaje de programación Simula67 y el paradigma de programación orientada a objetos.

![Interfaz de usuario gráfica, Aplicación, Teams

Descripción generada automáticamente]()![Texto

Descripción generada automáticamente con confianza baja]()Paradigma: Forma de pensar bajo un modelo prestablecido.  
-**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. Este paradigma 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. La fortaleza del paradigma de la programación orientada a objetos yace en utilizar abstracciones y crear entidades.  
-**Paradigma funcional**: Se basa en un concepto muy simple y es el de las funciones matemáticas. La fortaleza 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.

![Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente]()  
-**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.  
Paradigma de programación lógica 🡪 instrucciones 🡪 Reglas lógicas.

![Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente con confianza media]()  
**-Paradigma de programación con lenguaje especifico de dominio**: Los lenguajes que encontramos acá tratan de resolver problemáticas super específicas.

![Diagrama

Descripción generada automáticamente]()  
**-Multiparadigma**: A lo largo de la evolución de la programación, con nuevos desafíos y paradigmas ha habido lenguajes que han modificado su estructura para poder permitir dar soluciones en distintos paradigmas.

![Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente]()

![Diagrama

Descripción generada automáticamente]()

DEL CODIGO AL EJECUTABLE:

Escribimos programas utilizando los lenguajes de programación, pero usualmente un programa puede estar conformado por muchos archivos escritos en distintos lenguajes.   
¿Como hace una máquina para interpretar lenguajes distintos en una e-commerce?  
Las instrucciones tienen que pasar del código que escribimos a un código que la maquina entienda. Esta traducción se llama **compilación**.   
**\*La compilación** toma todo el código fuente y lo transforma en el famoso programa ejecutable escrito en código máquina. Listo para ser ejecutado en nuestros dispositivos.  
El código maquina solo funciona en nuestra maquina? Esto dependerá del código máquina. El resultado de la compilación, o sea el ejecutable, debería poder ejecutarse correctamente siempre y cuando la maquina donde se compile sea similar a donde se ejecute.   
Y esto quiere decir tener una similar Arquitectura de CPU y también un sistema operativo similar.  
Si tenemos un ejecutable (código maquina A y sistema A) y queremos ejecutarlo en una maquina servidor e-commerce con Arquitectura B y sistema operativo B no se va a poder ejecutar. Por lo que tenemos que llevar el código fuente a una maquina similar a la B para que se compile correctamente y tener un programa ejecutable para esa máquina.  
Podría ser mi código independiente de la plataforma donde corra?  
Hay otras dos formas para que los programas sean entendidos y ejecutados por una maquina independientemente de la arquitectura: una se conoce como maquinas virtuales y a la otra como intérpretes.   
**\*Máquina virtual**: Imaginemos que tenemos nuestra máquina, pero virtual a la cual llamaremos VM (Virtual Machine) Ahora cuando escribimos un programa el código fuente va a ser compilado a código máquina, pero no de nuestra maquina física sino al código que entienda la VM. Por ende, nuestro código va a poder ![Diagrama

Descripción generada automáticamente]()ejecutarse en máquinas VM.

Hay empresas que se encargan de mantener estas máquinas virtuales lo más eficiente posible y actualizadas ante cambios de arquitectura.  
**\*Interprete**: Este proceso hace un análisis línea por línea en cada sistema donde se ejecuta el código fuente así reduce en el momento a un 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.   
¿Podemos elegir como se va a ejecutar nuestro código?  
¿Si va a ser compilado, interpretado o con una VM? No podemos elegirlo, porque esto viene de la mano de como el lenguaje de programación fue diseñado.   
Pros y contras de estas 3 formas de interpretación del código:  
La diferencia radica en la performance y en el rendimiento.   
-Cuando el código fuente se compila y uno obtiene el código maquina 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 vez de ejecutarse en la maquina especifica se va a ejecutar en esta maquina virtual que hace de intermediaria con la física.  
-Con los lenguajes interpretados tenemos la desventaja que la traducción se va realizando línea por línea cada vez que se ejecuta, y es ese proceso de traducción el que ralentiza la operación de ejecución del código.  
Si bien estos lenguajes que corren en una maquina 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. Algo que sin duda es un factor clave en la toma de decisión sobre que lenguaje desarrollar.

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 maquina**: Es una secuencia de sentencias en lenguaje de maquina o binario. Es el resultado obtenido después 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 maquina se almacena en forma de archivo ejecutable.

![Diagrama

Descripción generada automáticamente]()

**Interprete**: 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.

![Diagrama

Descripción generada automáticamente]()