Lenguajes de Programación

Introducción

En este curso estudiaremos la estrucutura, forma y orden de Lenguajes de Programación

0.1 ¿Por qué estudiarlos?

Estudiar los Lenguajes de Programación es una labor fundamental para los Informáticos, ya que permite resolver problemas de forma eficiente, expandir la capacidad de expresar ideas, incrementar la facilidad para aprender nuevos lenguajes, comprender mejor el funcionamiento de los distinos sistemas que componen el área y expandir la visión de la computación y la informática.

0.2 ¿Para qué sirven?

Los Lenguajes de Programación tienen diferentes aplicaciones, dentro de las cuáles se hallan: aplicaciones de Negocio, aplicaciones Científicas, Ciencia de Datos, Inteligencia Artificial, aplicaciones Web, entre otras ramas.

0.3 Paradigmas

Los Paradigmas son la filosofía detrás de casa Lenguaje, indican primoridalmente su lógica y operación. Muchos de éstos tienen alguna base Matemática, ya sea en el área de Cálculo o Algebra. Es importante notar que muchos lenguajes son Multi-Paradigma, oséa que pueden pertenecer a más de alguno de los mecionados próximamanete.

Imperativo: Lenguaje basado en un orden lineal de instrucciones y funciones. Es uno de los paradigmas más comúnes dentro del área, ya que resulta ser bastante intuitivo, además de ser uno de los Paradigmas más antiguos.

Funcional: Lenguaje basado en el cálculo Lambda, opera a través de funciones y recursión.

Declarativo: Lenguaje basado alrededor de un sistema de consultas, las cuáles operan dentro de una serie de reglas lógicas establecidas por el programador. Se buscan respuestas sin necesariamente definir el método a emplear.

Orientado a Objetos: Lenguaje construido en base al concepto de objetos; Unidades Lógicas que se componen de Funciones y Parámetros únicos. Suele ser una extensión del Paradigma Imperativo, aunque también se puede hallar en otras formas.

0.4 Abstracciones

Las abstracciones corresponden a una forma explícita de visualizar las operaciones y conceptos que manejan los programas descritos por los Lenguajes de Programación.

Primero tendremos a los datos, éstos contendrpan la información que luego será sujeta a operaciones bajo el lenguaje empleado.

Los datos primitivos corresponden a la unidad más básica de información, la cuál es comunmente operada de forma más directa por el computador, por ejemplo, los enteros, los caractéres, los números reales, entre otros.

Los datos compuestos corresponden a unidades compuestas y estructuradas de datos primitivos.

Los datos simples correspoden a aquellos datos que vienen descritos bajo el Lenguaje de programción empleado, pueden ser primitivos o compuestos. Por ejemplo, algunos datos primitivos simples serían los enteros, carácteres. Mientras que los datos simples compuestos corresponderían a los strings y arreglos.

Segundo, existen las abstracciones de control, las cuales corresponden al formato de las operaciones o instrucciones que el código irá a ejecutar.

Las sentencias corresponden a las instrucciones directas que se le piden al programa para que ejecute, por ejemplo, una suma, una resta, cambio de valor, ejecución de una función, un print, etc.

Las estrcuturas de control corresponden a las condiciones que se deben cumplir para poder realizar para poder ejecutar alguna sentencia, o inclusive entrar a otra condición, por ejemplo, el "if", el "while", el "for", etc.
La abstracción procedural corresponderá a la invocación de algún procedimiento con nombre y con parametros de entrada, resultando en un cambio directo o indirecto al programa, por ejemplo, suma(x,z), mayor(x,y), print(x), append(x,y), etc.

La concurrencia permite realizar cálculos en paralelo, por ejemplo, pedir hilos del CPU, asignación de núcelos de GPU, asignación de tareas, etc.

Tercero y último, los tipos de datos abstractos (TDA), son datos y estructuras definidas dentro del mismo código. Abstrae un conjunto de datos y operaciones que son propias a este para poder instanciarse más adelante en el código, por ejemplo, los structs, las classes y los objetos.

0.5 Implementaciones

Compilación: Traduce el código a lenguaje de máquina para poder ejecutarse directamente por el procesador del computador, permitiendo mayor cercanía con los componentes de este y entregando un rendimiento superior. Suele hallarse en lenguajes de bajo nivel.

Interpretación: Utiliza una máquina virtual que interpreta el código durante su ejecución, permitiendo más libertades a la hora de escribir el código y que éste puede trasladarse a distinas arquitecturas con mayor facilidad. Suele hallarse en lenguajes de muy alto nivel

Híbrido: Se compilan a un lenguaje intermedio, que luego es interpretado por una máquina virtual, entregando un balance entre los otros dos métodos. Suele hallarse en lenguajes de nivel intermedio.

0.6 Compilación VS. Intepretación

Estas dos implementaciones de lenguajes tienen dos formas drásticamente diferentes a la hora de ejectutar el código, por ende es crucial analizar sus diferencias.
En los lenguajes compilados, el programa fuente es analizado léxicamente, es decir, se verifica que no hallan fallas en la escritura de símbolos. Luego este pasa por un analizador sintáctico, el cuál verifica que el formato, las estructuras de control y la definición de parámetros esté bien escrita. Finalmente, y si pasa por ambos analisis exitosamente, se generará el código máquina por el compilador, el cúal contendrá el código en un formato que el computador entiende y puede ejecutar de forma directa.
Mientras que en los lenguajes interpretados, el código fuente se traslada a un interpretador que recibe datos de entrada en el vuelo para poder ejecutarlos.

Como el código en esta implementación no pasa por ningún tipo de análisis léxico previo a al ejecución, puede darse la situación de que el código tenga fallas o esté mal escrito, pero que durante la ejecución se pueda correr el programa sin problemas. Este modelo también es mucho menos estricto a la hora de designar variables, pedir memoria y definir parámetros, ya que esa tarea se le delega al interpretador.

A gran escala

Por supuesto la programación se hace conmúnmente entre equipos grandes de gente, lo cuál hará crucial adaptar buenas prácticas la hora de escribir, revisar, editar y corroboar al código que se realice en estos entornos.
La modularización de programas permite organizar las distinas piezas del proyecto en un formato compacto y que no sea abrumador a la hora de leer, oséa que reduce la carga cognitiva, ya sea para personas que ven el código por primera vez, o para hacer la lectura a futuro mucho más amena.
La compilación separada permitirá que no se deba compilar el programa completo cada vez que se realize un cambio a alguna porción del código.
La reutilización de código entre diferentes programas hará que escribir nuevas funciones sea más fácil, ya que no es necesario re-inventar la rueda cada vez que se realizen nuevos proyectos o se extienda el código.
Los ambientes de desarrollo entregan diferentes herramientas y facilidades para apoyar el proceso en todo el ciclo de vida del software.

Aspectos a considerar

Finalmente, es importante siempre tener en cuenta el contexto bajo el cuál se desarrolle

Fundamentos

Dentro de los fundamentos de los lenguajes de programación, estarán estos cinco conceptos básicos, que permitirán desmenuzar el código y poder identificar sus partes.

Lenguaje: Conjunto de cadenas de símbolos que son propios al lenguaje particular.
Sintáxis: Conjunto de reglas y formatos que definen las secuncias correctas de los elemetnos de un lenguaje de programción.
Semántica: Significado de los signos lingüisticos y combinaciones.
Alfabeto: Conjunto de símbolosm letras o tokens con el cuál se puede formar una cadena dentro de cierto lenguaje.
Gramática: Estructura matemática con un conjunto de reglas de formación que definen las cadenas de caractéres admitidos en un determinado lenguaje.

0.7 Sintaxis

La sintaxis de un lenguaje será uno de los factores más importantes a la hora de reconocer su funcionamiento y lógica. Por ejemplo, el uso de indentaciónes, la delimitación de las sentencias o condiciones, la declaración de variables u objetos, etc. Existen método formales para describir la sintaxis de un programa, por ejemplo EBNF.

La sintaxis diverge en distintos tipos. Primero la sintáxis léxica, la cuál define reglas para los distints símbolos básicos o palabras "token". Luego está la sintaxis concreta, la cual refiere a la respresentación real de un programa usando símbolos del alfabeto, chequea si el programa está gramáticamente correcto, por ejempo, dureante declaración de variables (int, String, array, char, float, etc), establecimiento de condiciones (if, while, for). Finalmente, estará la sintaxis abstracta, la cuál lleva solo la información esencial del programa, como la puntuación, uso correcto de los corchetes y paréntesis, valores las varaibles, asignaciones de memoria explícita, etc.

0.8 Semántica

La semántica también diverge en dos categorías. Primero estará la semántica estática, la cuál define las restricciones sobre la estructura de los textos válidos que resutan defíciles de expresar en el formalísmo estándar de la sintáxis. Define reglas que se pueden verificar en tiempo de compilación. "Gramática con Atributos" se aplica para especificarla. La semántica dinámica, o de ejecución, deifne el comportamiento que se produce durante la ejecución

del programa. Se puede espceificar mediante lenguaje natural, aunque es deseable formalizarla para mayor precisión.

0.9 Tokens

Los "tokens" como mencionamos anteriormente, corresponden a unidades sintácticas que serán la clave para identificar el cuerpo del lenguaje de programación empleado. Se categorizan por los siguientes tipos:

1) Palabras claves y reservadas (ej: if, while, struct)

2) Literales y constantes (ej: 3.1415, "hola")

3) Identificadores (ej: suma, getBalance)

4) Símbolos de operadores (ej: 3.1415, "hola")

5) Símbolos especiales (ej: blancos, espacios, delimitadores y paréntesis)

6) Comentarios (ej: Un comentario, /* Otro comentario */

0.10 Expresiones Regulares

Permiten describir patrones de cadenas de caractres. Son útiles para especificar y reconocer los "tokens" previamente mencionados. Cada expresión regular tiene asociado un autómata finito. Las expresiones regulares tienen las siguientes propiedades. Concatenación, expresa una secuencia de los elementos. Cuantificación, permite especificar frecuencias, ya sea una ocurrencia, más de una y una o ninguna. Alternación, permite elegir entre alternativas. Agrupración, permite agrupar varios elementos con paréntesis redondeos (no permite recursión).