Lenguajes de Programación 2023-1
Nota de clase 1: El estudio de los lenguajes
Introducción

Javier Enríquez Mendoza

17 de agosto de 2022

El estudio de los lenguajes tiene una historia muy extensa e interesante, que remonta a miles de años. Si bien la lingüística como ciencia tiene su origen en el siglo XIX, cuando gracias al descubrimiento del sánscrito se pudieron comparar diferentes lenguas y así reconstruir un supuesto lenguaje *original*, los lenguajes han acompañado a la humanidad prácticamente desde sus orígenes.

No se sabe a ciencia cierta cuándo o cómo fue el origen de los lenguajes, existen diferentes teorías sobre esto. Pero si es bien sabido la importancia que los lenguajes tienen para la humanidad y la relación que tienen con la cultura y el pensamiento humano. Como dijo Leibniz *"Los lenguajes son el mejor espejo de la mente humana"*.

Es a través del lenguaje que como personas podemos comunicarnos con otros, expresar las ideas que surgen en nuestro cerebro y de esa forma poder compartir el conocimiento. En cada momento y aspecto de nuestras vidas los lenguajes juegan un papel principal, es gracias al lenguaje que tenemos música, poesía, literatura, relaciones sociales, avances científicos, educación, entre muchas otras cosas. Incluso en este momento el estar leyendo este texto es posible gracias a los lenguajes.

La lingüística es la disciplina encargada del estudio del origen, la evolución y la estructura del lenguaje. El propósito de ésta es definir una serie de reglas o leyes que rijan las estructuras fundamentales de los lenguajes. Esto lo logra mediante un estudio sobre estas estructuras para poder definir tres tipos de reglas: **sintácticas**, **semánticas** y **pragmáticas**.

Durante miles de años los lenguajes han funcionado a la perfección como una herramienta de comunicación entre personas. Es por esto que cuando surge la necesidad de establecer un canal de comunicación entre humanos y otros seres u objetos siempre se busca la creación de un lenguaje que nos lo permita.

> *"Language is the foundation of civilization. It is a glue that holds people together, and it is the first weapon drawn in a conflict."*[1]

[1] Frase de la película Arrival (2016) de Denis Villeneuve, en donde una lingüista salva al mundo tras la llegada de una nave extraterrestre.

En este curso se estudiaran las reglas de sintaxis, semántica y pragmática que existen en una categoría de lenguajes muy especifica, los lenguajes de programación. El propósito principal es definir formalmente estos lenguajes para su estudio y de esa forma entender como es que nos comunicamos con una computadora.

1 Lenguajes de programación

De la misma forma en la que empleamos lenguajes naturales para establecer comunicación entre personas, la forma en la que nos comunicamos con un ordenador es mediante lenguajes a los que llamamos **lenguajes de programación**.

Estos lenguajes sirven como interfaz entre las programadoras y programadores y la computadora, esto quiere decir que estos lenguajes tienen que estar definidos en un contexto intermedio en el cual sean comprensibles tanto para las personas que escribimos los programas como para los ordenadores que se encargan de ejecutarlos. Las ideas que comunicamos mediante un lenguaje de programación son ideas **algorítmicas** que corresponden a cómputos o especificaciones de cómputos que la maquina debe llevar a cabo.

Como se estudio en el curso de Autómatas y Lenguajes Formales la noción de lo computable se puede formalizar usando modelos teóricos de computo como Maquinas de Turing, funciones recursivas μ o cálculo lambda. Entonces decimos que un lenguaje es computacionalmente **completo** (también llamado Turing completo) si puede expresar todas las funciones formalmente computables en los modelos anteriores.

Para definir formalmente un lenguaje de programación para su estudio, es necesario considerar tres aspectos básicos: **sintaxis**, **semántica** y **pragmática**.

1.1 Sintaxis

Definición 1.1 (Sintaxis)**.** Es la forma en la que los elementos lingüísticos se combinan para formar expresiones correctas de un lenguaje.

La sintaxis define la forma del lenguaje de programación, es decir, la descripción del conjunto de cadenas que serán considerados programas válidos. Existen dos niveles: la sintaxis concreta y la sintaxis abstracta, estos niveles se estudiaran a detalle mas adelante en el curso.

Para dar una especificación de la sintaxis se utilizan gramáticas libres de contexto con notación de Backus-Naur.

Ejemplo 1.1 (Sintaxis de EA)**.** Gramática formal para definir la sintaxis del lenguaje de expresiones aritméticas (EA).

$$e \quad ::= \quad n \quad | \quad e + e \quad | \quad e * e$$

en donde $n \in \mathbb{N}$.

1.2 Semántica

Definición 1.2 (Semántica)**.** Es la rama de la lingüística y la lógica que relaciona expresiones sistemáticamente correctas con su significado. Es el estudio de la relación que existe entre expresiones y símbolos de un lenguaje con lo que representan.

La semántica define el significado de instrucciones y expresiones del lenguaje. Usualmente es descrita de manera informal en la documentación del lenguaje mediante manuales técnicos que describen en lenguaje natural el funcionamiento de las expresiones del lenguaje. También pueden ser descritas formalmente mediante técnicas matemáticas que pueden ser operacional, denotativa o axiomática, ésta última es la forma en la que se definirá la semántica de un lenguaje en este curso, usualmente usando funciones o relaciones de evaluación como se muestra en el ejemplo siguiente.

Ejemplo 1.2 (Semántica de EA)**.** Función semántica para el lenguaje de expresiones aritméticas.

$$[\![\cdot]\!] : \mathsf{EA} \to \mathbb{N}$$

en donde $\mathbb{N}$ es el conjunto de los números naturales. Esto quiere decir que la función semántica recibe una expresión del lenguaje EA y regresa un número natural y se define de la siguiente forma:

$$\begin{aligned} [\![n]\!] &= n \\ [\![e_1 + e_2]\!] &= [\![e_1]\!] +_{\mathbb{N}} [\![e_2]\!] \\ [\![e_1 * e_2]\!] &= [\![e_1]\!] \times_{\mathbb{N}} [\![e_2]\!] \end{aligned}$$

en donde los operadores $+_{\mathbb{N}}, \times_{\mathbb{N}}$ representan las funciones de suma y producto definidas para los naturales respectivamente.

A partir de la definición anterior se puede implementar un interprete para expresiones del lenguaje EAB mediante una función recursiva `eval` como sigue:

```
eval n           = n
eval (e1 + e2)   = eval e1 + eval e2
eval (e1 * e2)   = eval e1 * eval e2
```

1.3 Pragmática

Definición 1.3 (Pragmática)**.** Es el estudio de los principios que regulan el uso de un lenguaje, es decir, las condiciones que determinan el empleo de una expresión así como la interpretación que se le da a ésta.

Dentro de la pragmática de un lenguaje se estudian los agentes que le dan contexto a la interpretación de ciertas expresiones de un lenguaje. En el caso concreto de los lenguajes de programación la pragmática se refiere a dos componentes principales, que son los que dan contexto en el uso del lenguaje:

Bibliotecas es el conjunto de funciones previamente definidas en un lenguaje, las cuales están disponibles para utilizarse por los programadores. Por lo general estas bibliotecas son de propósitos o funcionalidades especificas, pero no es necesario que sea de esa forma.

> **Observación.** Es común encontrar en la literatura en español que se hace referencia a las bibliotecas como librerías, esto es un error de traducción de la palabra en ingles *library*. Es a lo que en el estudio de las lenguas se conoce como un *falso amigo*.
>
> Es incorrecto referirse a las bibliotecas como librerías pues el concepto viene del hecho que una biblioteca nos permite como desarrolladoras y desarrolladores el uso de las funciones sin algún costo, que es como funciona una biblioteca al prestarnos libros y no venderlos como lo hace una librería.

Idioms son las convenciones no escritas que se utilizan al programar en un lenguaje en especifico, son reglas de estilo que las programadoras y programadores siguen al desarrollar en un lenguaje.

Por ejemplo el uso de *camel case* o *snake case* para los identificadores dependiendo del lenguaje en el que estamos programando.

2 Clasificaciones de los lenguajes de programación

En la actualidad hay una gran cantidad de lenguajes de programación, los cuales comparten algunas características de diseño entre sí, lo que nos permite clasificarlos en diferentes categorías según estas similitudes. Las principales clasificaciones para los lenguajes se dan:

Por la forma en la que se describen los cómputos Dependiendo del lenguaje que estemos utilizando la manera en que se escriben las instrucciones dentro de un programa puede cambiar. Hay dos grandes ramas:

- **Imperativos:** los programas se definen como instrucciones que afectan su estado. Se concentran en el *¿cómo?* resolver un problema.
- **Declarativos:** los programas se definen a partir de condiciones que definen el problema a atacar. Definen el *¿qué?* se quiere resolver.

Por su propósito Existen lenguajes de programación que fueron creados para cumplir con una tarea particular, estos lenguajes se consideran de **propósito específico**. De igual forma existen lenguajes que son utilizados para la resolución de distintas tareas, estos son lenguajes de **propósito general**. La mayoría de los lenguajes actualmente son de propósito general.

Por la forma en la que son ejecutados El proceso de ejecución de un programa depende del lenguaje en el que fue desarrollado, se dividen en dos grandes categorías:

- **Compilados**: la ejecución de estos lenguajes es mediante un compilador, que se encarga de traducir todo el código a una representación en otro lenguaje (usualmente lenguaje máquina). Algunos lenguajes compilados son: Haskell, Java, C, etc.
- **Interpretados**: en estos lenguajes se va ejecutando cada instrucción del programa de forma directa, sin necesidad de una traducción completa del código, esta ejecución se da mediante un interprete. Como ejemplos de lenguajes interpretados tenemos a Python, Scheme, JavaScript, etc.

Por su paradigma Un paradigma es un estilo fundamental de programación definido por la forma

de dar soluciones a problemas. Proporciona y determina la visión que el programador tiene sobre la ejecución de un programa. Los principales paradigmas de programación son: **procedimental**, **funcional**, **lógico** y **orientado a objetos**.
En la actualidad este criterio de clasificación es difuso debido a que la mayoría de los lenguajes no se diseñan siguiendo puramente un paradigma, sino que adoptan características de dos o mas paradigmas, esto se conoce como **multiparadigma**.

La clasificación por paradigma es la que estudiaremos con mayor detalle a lo largo del curso. Veamos ahora algunas de las principales características de los paradigmas clásicos de programación.

2.1 Paradigma Procedimental

En este paradigma los programas se escriben como una secuencia de instrucciones orientado hacia el estado. La evaluación de un cómputo se da mediante alteraciones implícitas a la memoria. Los principales elementos de este paradigma son la abstracción procedimental, los ciclos y la asignación de variables. Las variables sirven como una abstracción de las celdas de memoria. Algunos lenguajes pertenecientes a este paradigma son: C, Rust, Cobol y Fortran.

Ejemplo 2.1. Algoritmo de ordenamiento `Quick Sort` implementado con estilo procedimental en el lenguaje de programación Rust.

```rust
fn quick_sort(param_int_array: &mut [i32], start: usize, end: usize) {
    if start >= end {
        return;
    }
    let pivot = partition(param_int_array, start, end);

    quick_sort(param_int_array, start, (pivot - 1) as usize);
    quick_sort(param_int_array, (pivot + 1) as usize, end);
}
```

2.2 Paradigma Funcional

Los programas se definen como una colección de funciones que especifican el problema que se está atacando. Estas funciones se combinan mediante composición para construir nuevas funciones con comportamientos mas complejos. La recursicón es la herramienta de control principal. En estos lenguajes no se tiene un manejo explicito de la memoria, los cómputos están expresados por aplicación y composición de funciones, en donde los valores intermedios son pasados directamente a otras funciones. Algunos ejemplos de lenguajes de programación funcionales son: Haskell, Miranda, Gofer que son lenguajes puramente funcionales, también existen lenguajes funcionales impuros como lo son Lisp, Scheme, Racket, ML entre otros.

Ejemplo 2.2. Algoritmo de ordenamiento `Quick Sort` en el lenguaje de programación puramente funcional Haskell.

```haskell
quicksort :: (Ord a) => [a] -> [a]
quicksort []     = []
quicksort (x:xs) = quicksort less ++ x:(quicksort more)
 where less = filter (<x)  xs
       more = filter (>=x) xs
```

2.3 Paradigma Lógico

Se definen programas mediante declaraciones lógicas que especifican las características que tiene la solución que se busca del problema a tratar y mediante un proceso de *backtracking* el lenguaje encuentra la solución que cumple con estas reglas haciendo una búsqueda de pruebas. Al igual que en la programación funcional, en la lógica no es posible una manipulación explicita de la memoria. El lenguaje de programación lógico mas conocido y utilizado es Prolog.

Ejemplo 2.3. Algoritmo de ordenamiento `Quick Sort` en el lenguaje de programación de paradigma lógico Prolog.

```prolog
quicksort([X|Xs],Ys) :-
    partition(Xs,X,Left,Right),
    quicksort(Left,Ls),
    quicksort(Right,Rs),
    append(Ls,[X|Rs],Ys).
quicksort([],[]).
```

2.4 Paradigma Orientado a Objetos

Los programas se expresan como una colección de objetos que interactúan entre si. La evaluación se da mediante el intercambio de mensajes entre los objetos. Estos objetos están agrupados en clases que define jerarquías. Las características principales de este paradigma son las clases, el encapsulamiento de datos y la herencia. Algunos lenguajes de programación orientados a objetos son Java, C++, Ruby, Scala, etc.

Ejemplo 2.4. Algoritmo de ordenamiento `Quick Sort` en el lenguaje de programación Orientado a Objetos Java.

```java
private static <T extends Comparable <T>> void
quickSort(T[] a, int ini, int fin) {
    if (fin - ini < 1)
        return;
    int i = ini + 1, j = fin;
    while (i < j)
        if (a[i].compareTo(a[ini]) > 0 &&
            a[j].compareTo(a[ini]) < = 0)
            swap(a, i++, j--);
    else if (a[i].compareTo(a[ini]) <= 0)
        i++;
    else
        j--;
    if (a[i].compareTo(a[ini]) > 0)
        i--;
    swap(a, ini, i);
    quickSort(a, ini, i-1);
    quickSort(a, i+1, fin);
}
```

En lo que resta del curso estudiaremos una serie de lenguajes que nos ayudaran a abstraer e implementar las principales características de cada uno de los paradigmas descritos anteriormente. De igual forma con estos lenguajes estudiaremos los conceptos principales que se deben tener en cuenta en el proceso de diseño de un lenguaje de programación.

Referencias

[1] Miranda Perea F., González Huesca L., Nota de Clase del curso de Lenguajes de Programación, Facultad de Ciencias UNAM, Semestre 2021-1.

[2] Ramírez Pulido K., Soto Romero M., Enríquez Mendoza J., Nota de Clase del curso de Lenguajes de Programación, Facultad de Ciencias UNAM, Semestre 2021-2

[3] Harper R., Practical Foundations for Programming Languages. Working draft, 2010.

[4] Mitchell J., Foundations for Programming Languages. MIT Press 1996.

[5] Krishnamurthi S., Programming Languages Application and Interpretation; Version 26.04.2007.

[6] Chomsky, N., Knowledge of Language: Its Nature, Origin, and Use; Praeger, Convergence (New York, N.Y.), 1989.