Lenguajes de Programación, 2022-1
Nota de clase 1: Conceptos generales

Karla Ramírez Pulido Manuel Soto Romero Javier Enríquez Mendoza

23 de septiembre de 2021
Facultad de Ciencias UNAM

Formalmente, un lenguaje de programación se define por: [14]

Definición 1. (Lenguaje de Programación) *Un lenguaje de programación es una terna* $\mathcal{L} = \langle \mathcal{P}_\mathcal{L}, \mathcal{D}_\mathcal{L}, [\![\cdot]\!]_\mathcal{L} \rangle$ *donde*

- $\mathcal{P}_\mathcal{L} \neq \varnothing$ *es el conjunto de programas de* $\mathcal{L}$
- $\mathcal{D}_\mathcal{L} \neq \varnothing$ *es el conjunto de datos (de entrada y salida) de* $\mathcal{L}$
- $[\![\cdot]\!]_\mathcal{L}$ *es la función semántica de* $\mathcal{L}$ *tal que*

$$[\![\cdot]\!]_\mathcal{L} : \mathcal{P}_\mathcal{L} \to \mathcal{D}_\mathcal{L} \to \mathcal{D}_\mathcal{L}$$

Se entiende como lenguaje de programación, única y exclusivamente a aquellos formalismos que cumplen con la Definición 1. De esta forma, un lenguaje de programación es aquel formado por un conjunto de programas escritos en el mismo, que a partir de una entrada particular, generan una salida mediante la función semántica.

En este curso se conocerán y aplicarán los principios y componentes en el diseño de los lenguajes de programación y se presentarán algunas de las las herramientas básicas para analizar formalmente diversas de sus características. En esta primera nota se presentan aquellos conceptos generales que permitirán entender la terminología que ahonda a los Lenguajes de Programación, partiendo desde su historia.

1.1. Historia de los Lenguajes de Programación

Los lenguajes de programación en la actualidad proveen diversos mecanismos que facilitan al programador la realización de múltiples tareas con un nivel de abstracción lo más alto posible, estos lenguajes de programación son llamados de alto nivel, sin embargo, esto no siempre fue así. Anteriormente la programación de los dispositivos de cómputo podía llegar a ser una tarea tediosa, compleja y realizada por un grupo selecto de personas.

1.1.1. Los Fundamentos

Antes de que existieran los lenguajes de programación tal y como los conocemos hoy en día, se definieron algunos fundamentos teóricos que dieron origen a los conceptos básicos de lo que conocemos hoy en día como computadora, y junto con ella se originaron las teorías con la decidibilidad de lo que es computable o no. Estos fundamentos han sido de utilidad al crear los lenguajes de programación como se les conoce actualmente. A continuación se listan algunos de estos fundamentos [1]:

Lenguajes formales

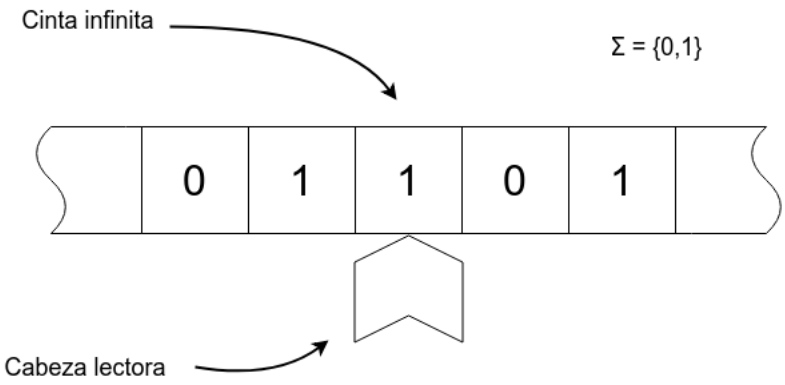

Figura 1: Una Máquina de Turing

Al igual que las matemáticas, los lenguajes de programación se pueden modelar usando lenguajes formales. Esto permite expresar ideas (programas) de forma concreta, breve y precisa. Al ser lenguajes formales, es posible construir modelos de los mismos, lo cual permite razonar sobre lenguajes de programación, manipularlos, demostrar ciertas propiedades sobre éstos, y obtener nuevos resultados a partir de otros ya conocidos, lo que a su vez permite encontrar respuestas a problemas sobre dichos modelos y generar extensiones del conocimiento [2].

Ada Lovelace y la Máquina Analítica

En 1834 Charles Babbage inició el diseño de un dispositivo de cómputo al que llamó *Máquina Analítica*, esta máquina sería capaz de calcular el valor numérico de cualquier fórmula o función, dada una entrada, en donde el usuario podía especificar el método de solución, la forma en que trabajaría la máquina sería completamente automática [6].

Ada Lovelace, conocida como la primera programadora fue la principal colaboradora del trabajo de Babbage. Ada escribió un programa para calcular los números de Bernoulli, este programa fue descrito en un diagrama que exponía paso a paso cómo debía realizar el cálculo la máquina, dicho programa se encontraba representado en un lenguaje entendible por la máquina para ser ejecutado.

El Cálculo λ

El Cálculo λ[1], es una de las teorías más importantes de los lenguajes de programación, fue introducido por Alonzo Church en el año de 1930, con el fin de formalizar el concepto de computabilidad[2]. Se dice que es el lenguaje de programación universal más pequeño del mundo y consiste de un lenguaje formal para describir términos y definir funciones, usando una única regla de transformación llamada β-reducción que permite sustituir variables[3]. La Nota de Clase 3 profundiza más sobre este tema.

Máquinas de Turing

Las máquinas de Turing son un modelo abstracto de cómputo propuesto por Alan Turing en 1936, que al igual que el Cálculo λ, proveen una definición formal sobre lo que es o no computable. Es una máquina de estados finita compuesta por una cinta infinita que incluye símbolos de un alfabeto finito (denotado por Σ), cuenta con una cabeza lectora que puede leer o escribir en dicha cinta. A partir del símbolo que se está leyendo y el estado actual, una Máquina de Turing escribe un nuevo símbolo en la posición actual y se mueve hacia la izquierda o derecha de acuerdo a lo especificado por una *función de transición*. La función de transición es el "programa" que especifica cómo se comporta la máquina, algo similar a los lenguajes de programación actuales [4]. La Figura 1 muestra una Máquina de Turing.

1.1.2. Lenguajes Ensambladores

En su inicios, programar una computadora electrónica, como la ENIAC (*Electronic Numerical Integrator And Computer)* consistía en conectar y desconectar cables para configurar el hardware, lo cual era una tarea tediosa y que pocos podían realizar.

[1] En inglés λ-calculus.
[2] Estudia los problemas de decisión que se pueden resolver mediante un algoritmo.

Con la llamada *Arquitectura de von Neumann* se resuelve este problema al tener un conjunto de instrucciones almacenadas en la memoria de la computadora y ejecutando las mismas en la Unidad Central de Proceso (CPU, en inglés)[7]. Sin embargo, estas instrucciones debían escribirse en un lenguaje que entendiera la computadora, como lo es el sistema binario. De esta forma, una instrucción en lenguaje máquina luce de la siguiente manera [8]:

0110 1001 1010 1011

Cada parte de esta instrucción indica a la computadora qué acción realizar. Por ejemplo, los primeros cuatro bits en la instrucción anterior (0110 de izquierda a derecha), podrían indicar que se trata de una suma, los siguientes 8 bits podrían representar los valores a sumar y los últimos cuatro bits (1011) indican el registro de memoria en que se almacenará el resultado de la suma[3].

Este tipo de instrucciones conforman un lenguaje llamado *lenguaje de máquina*, sin embargo, este lenguaje no resulta ser un lenguaje tan claro y fácil de leer por los humanos. Debido a esto, se crearon los *lenguajes ensambladores* que realizan un mapeo entre las etiquetas llamadas mnemotécnicos y el lenguaje de máquina. Por ejemplo, una operación de suma, representada por el código 0110, se convierte en la instrucción `ADD` (suma). La siguiente línea de código en lenguaje ensamblador representa la instrucción en lenguaje máquina 0110 1001 1010 1011 (Sumar la variable `M` con la `N` y almacenar el resultado en el registro `P`) [8]:

```
ADD M, N, P
```

Combinando este tipo de instrucciones se conformaron los primeros lenguajes de programación con una sintaxis y semántica bien definida. Sin embargo, estos lenguajes presentan una gran desventaja, (al igual que el lenguaje de máquina) que es la dependencia a cada tipo de arquitectura de cada máquina.

Hoy en día, este tipo de lenguajes tienen sus aplicaciones muy reducidas y se centran principalmente en control de procesos y de dispositivos electrónicos [8].

1.1.3. Lenguajes de Alto Nivel

Los lenguajes de programación de alto nivel, son los más utilizados en la actualidad, ya que (1) fueron diseñados con los objetivos de ser más entendibles para las personas y (2) porque los programas escritos en este tipo de lenguajes no dependen de ninguna arquitectura en particular. Como consecuencia de esto pueden ser ejecutados en cualquier computadora a diferencia de los lenguajes máquina o ensambladores.

La Tabla 1.1 muestra algunos de los lenguajes de programación más utilizados hoy en día junto con información acerca de sus creadores y sus principales características.

Nombre	***Diseñador(es)***	***Año de creación***	***Paradigma(s)***	***Usos***
FORTRAN	John Backus	1954	Estructurado Orientado a Objetos	Aplicaciones científicas y de ingeniería; programas que evalúan el desempeño y posicionamiento de supercomputadoras.
COBOL	Comisión CODASYL	1950	Estructurado Orientado a Objetos	Aplicaciones comerciales que requieren de una manipulación precisa y eficiencia de grandes volúmenes de datos.
LISP	John McCarthy	1959-1960	Funcional	Representación de expresiones simbólicas y manejo de listas.

[3]Esto varía entre las diferentes arquitecturas que existen de computadoras.

Nombre	***Diseñador(es)***	***Año de creación***	***Paradigma(s)***	***Usos***
ALGOL	John Backus	1960	Estructurado	Utilizado principalmente en aplicaciones científicas.
SIMULA	Ole Johan Dahl / Kristen Nygaard	1965	Orientado a Objetos	Simulación de eventos simultáneos.
Pascal	Niklaus Wirth	1969	Estructurado	Se diseñó para la enseñanza de la programación estructura y fue popular en cursos universitarios durante varias décadas.
C	Dennis Ritchie	1972	Estructurado	Aplicaciones de escritorio, aplicaciones científicas, en particular simulaciones.
Prolog	Alain Comerauer, Philippe Roussel	1972	Lógico	Inteligencia artificial, sistemas expertos, reconocimiento del lenguaje natural.
ML	Universidad de Edimburgo	1973	Funcional Estructurado	Principalmente académicos.
Scheme	Guy L. Steele	1975	Funcional Estructurado	Manejo de listas, definición de macros, aplicaciones de escritorio, verificación de programas.
Smalltalk	Alan Kay	1980	Orientado a Objetos	Lenguaje de propósito general, programas de escritorio y web principalmente.
C++	Bjarne Stroustrup	1983	Estructurado Orientado a Objetos	Programas de escritorio, programación web, graficación por computadora y animación.
Haskell	Universidad de Glasgow	1990	Funcional	Principalmente en la academia, sin embargo, se tienen algunas aplicaciones en la industria aeroespacial, de finanzas y algunas empresas de diseño de hardware.
Python	Guido van Rossum	1991	Orientado a Objetos Funcional	Acceso a bases de datos, servicios web, aplicaciones de escritorio y web, gráficos 3D, juegos.
PL/SQL	Oracle	1993	Estructurado	Manipulación de tablas y consultas dentro de una o más bases de datos.
Racket (PLT Scheme)	Grupo PLT	1994	Funcional Estructurado Orientado a objetos	Diseñado para la creación de nuevos lenguajes de programación y enseñanza de la programación.
Java	Sun Microsystems	1995	Orientado a Objetos	Aplicaciones empresariales, aplicaciones web, aplicaciones de escritorio, programación de dispositivos móviles.
Ruby	Yukihiro Matsumoto	1995	Orientado a Objetos	Principalmente en la programación web a través del *framework* Ruby on Rails.
OCaml	Xavier Leroy	1996	Funcional Orientado a oObjetos	Aplicaciones de escritorio, web, dispositivos móviles, implementación del asistente de pruebas Coq.

Tabla: 1.1: *Lenguajes de alto nivel*

1.2. Clasificación de los Lenguajes de Programación

En la actualidad existen miles de lenguajes de programación que pueden clasificarse de acuerdo a sus características o funcionalidades. Los principales criterios de clasificación son:

- De acuerdo al nivel de abstracción.
- De acuerdo al paradigma de programación.
- De acuerdo al propósito con el cuál fueron creados.

1.2.1. Clasificación de acuerdo al nivel

Para separar los lenguajes de programación de acuerdo al nivel de abstracción (con respecto al hardware), se clasifican en varias categorías. Existen dos grandes categorías en las que se pueden clasificar los lenguajes de programación, (1) *de alto nivel* o (2) *de bajo nivel*, sin embargo, algunos autores consideran una tercera categoría (3) *de medio nivel*. Todos los lenguajes pertenecen a alguna de éstas tres.

Lenguajes de alto nivel: Son independientes a la arquitectura de la computadora, son más comprensibles para el programador y más sencillos de usar.

Lenguajes de bajo nivel: Son lenguajes que presentan muy poca o nula abstracción respecto a las instrucciones de ensamblador. Los comandos del lenguaje se traducen casi directamente a instrucciones del procesador. Los programas escritos en estos lenguajes tienden a ser poco portables.

Lenguajes de medio nivel: Se encuentran entre las otras dos categorías, son aquellos que no son muy cercanos a las instrucciones del procesador pero si puede controlarse el hardware con ellos.

1.2.2. Clasificación de acuerdo al paradigma

Los lenguajes de programación se dividen principalmente en dos paradigmas, el *declarativo* y el *imperativo*.

En el paradigma imperativo los programas son una secuencia de instrucciones, con ciclos `while`, `repeat`, `for`, etcétera como principales estructuras de control. Las asignaciones juegan un papel de suma importancia. En este paradigma el programa describe *cómo* se resuelve un problema.

Dentro de este paradigma existen otras subcategorías:

- **Estructurado:** Los programas son secuencias lineales de código que se van ejecutando en orden una después de la otra.
- **Orientado a Objetos:** Todos los elementos de estos lenguajes son modelados como objetos, con funcionamiento y atributos.

Por otro lado en el paradigma declarativo los programas son vistos como una sucesión de definiciones, siendo la recursión la principal estructura de control. No se tienen ciclos ni operaciones de asignación. Es decir el programa específica *qué* se debe calcular, siendo el cómo completamente irrelevante.

Las principales subcategorías de este paradigma son:

- **Funcional:** Basados en el Cálculo Lambda y la Teoría de Categorías. Es un modelo de programación en el que se enfatiza en el uso de funciones y sus aplicaciones más que en comandos y sus ejecuciones.
- **Lógico:** Basado en la lógica de primer orden las expresiones del lenguaje son Cláusulas de Horn.

Estos son los principales paradigmas de los lenguajes de programación, sin embargo es difícil encontrar lenguajes modernos que correspondan puramente a un paradigma, pues implementan características de diferentes paradigmas. A estos lenguajes se les conoce como *lenguajes multiparadigma*.

1.2.3. Clasificación de acuerdo al propósito

Algunos lenguajes son diseñados especialmente para ajustarse o resolver una necesidad particular, estos se conocen como *lenguajes de propósito específico*. Por ejemplo:

- Fortran, *FORmula TRANslator*
- Algol, *ALGOrithmic Language*
- COBOL, *COmmon Business-Oriented Language*
- LISP, LISt Processing

Como puede observarse en su nombre, estos lenguajes se diseñaron para realizar tareas específicas. Aunque más adelante (en los 60s y 70s) fueron liberados de estas restricciones y así surgieron los lenguajes de propósito general, que son los lenguajes diseñados para desarrollar software de más de un propósito. Por ejemplo:

- JAVA puede usarse para desarrollar páginas web así como para hacer juegos o aplicaciones para dispositivos móviles.
- RACKET puede usarse para diseñar lenguajes de programación así como para definir funciones matemáticas.
- PYTHON, PERL y RUBY pueden usarse tanto para desarrollo web como para desarrollar aplicaciones de escritorio.

La mayoría de los lenguajes de programación modernos son diseñados para ser de propósito general. Sin embargo diseñar un buen lenguaje de programación de propósito general es una tarea difícil y por lo tanto no es sorprendente que los diseñadores de lenguajes regresen poco a poco al diseño de lenguajes de propósito específico.

1.3. Componentes de los Lenguajes de Programación

Un lenguaje de programación se compone generalmente de[9]:

Sintaxis Se refiere a la forma de escribir las sentencias de un programa, dentro de un lenguaje de programación. "La sintaxis tiene como principal función analizar el orden correcto de las palabras a fin de que las frases, oraciones, textos e ideas sean expresados de manera correcta para que pueda llegar el mensaje que se desea transmitir" [12].

Ejemplo 1. Dependiendo de cada lenguaje se podrá definir que un número o variable de tipo entera, se deberá escribir su tipo como: `INTEGER` o `int`, es decir, la sintaxis determina la forma como se deben de escribir cada uno de los elementos del lenguaje.

Dicho de otro modo, para declarar una variable, la sintaxis nos indica que debemos colocar el tipo junto a un identificador, un símbolo = y un valor, en este caso numérico.

Semántica Se refiere al significado que se otorga a cada una de las sentencias escritas, de acuerdo a la sintaxis definida previamente en el lenguaje.

Ejemplo 2. Si algún programador escribe la siguiente línea que cumple con las reglas de sintaxis descritas en el ejemplo 1:

```
int a = 7;
```

El significado que se le da a lo escrito en este es que está asignando a la variable `a` se le asigna el tipo entero el cual tiene un valor numérico de 7, por lo que el programador da un significado asociando a un tipo específico a dicha variable. □

Bibliotecas Se refiere al conjunto de funciones previamente definidas en un lenguaje las cuales, están disponibles para utilizarse por los programadores. En muchos lenguajes de programación ya se tienen varias bibliotecas que presentan funciones, métodos o procedimientos (dependiendo del paradigma del lenguaje de programación) que permiten que los programadores hagan uso de éstas. Ejemplos de bibliotecas pueden ser `java.lang.Math` de JAVA o `Data.List` de HASKELL.

Convenciones de programación (*Idioms*) Se refiere a las formas en las que se puede expresar algún tipo de elemento dentro de un lenguaje de programación, dicho de otra forma son convenciones de programación no escritas que todos los programadores de un lenguaje conocen.

Ejemplo 3. En Java el nombre de las clases debe iniciar con una letra mayúscula por convención.

```
public class Foo {
...
```

1.4. Ejercicios

1.4.1. Historia de los Lenguajes de Programación

1. Construir una tabla con algunos de los lenguajes de programación más representativos. Por cada lenguaje en la tabla se deberá mencionar:
   (a) Nombre
   (b) Año de creación
   (c) Diseñador(es)
   (d) Usos principales
   (e) Principales características

2. Realizar un resumen sobre alguno de los siguientes personajes históricos[4]. El resumen deberá contener:
   (a) Breve contexto histórico contemporáneo al personaje, es decir, situación social, política y avances históricos sobre la computación.
   (b) Principales contribuciones y/o creaciones relacionadas a las Ciencias de la Computación.
   (c) Impacto; consecuencias directas o indirectas en las Ciencias de la Computación debido a dichas contribuciones.
   (d) Contribuciones directas o indirectas al área de los lenguajes de programación.
   (e) Ejemplos de influencia en los lenguajes de programación modernos basadas en las contribuciones del personaje.

Jacquard Loom	Konrad Zuse	Benjamin Crawford Pierce	John McCarthy
Ada Lovelace	Haskell Curry	Corrado Böhm	Edsger Dijkstra
Alonzo Curch	Dennis Ritchie	Dana Stewart Scott	Donald Knuth
Alan Turing	Kathleen Booth	James Gosling	Niklaus Wirth

1.4.2. Clasificación de los Lenguajes de Programación

1. Construir una tabla con al menos diez lenguajes de programación. Por cada lenguaje en la tabla se deberá mencionar:
   (a) Clasificación de acuerdo al nivel.
   (b) Clasificación de acuerdo al paradigma.
   (c) Clasificación de acuerdo al propósito.

[4] También puedes proponer otro personaje

1.4.3. Componentes de los Lenguajes de Programación

1. Usando el lenguaje de programación de tu preferencia, dar un ejemplo de cada uno de los siguientes conceptos y justificar. No es necesario que sean ejemplos demasiado elaborados.

   (a) Sintaxis.

   (b) Semántica.

   (c) Convenciones de programación (*Idioms*)

   (d) Bibliotecas

Referencias

[1] Samuel A. Rebelsky, *Programming Languages,* Notas de clase, Grinell College revisión 99S. Consultado el 20 de noviembre de 2018 [`http://www.math.grin.edu/~rebelsky/Courses/CS302/99S/Outlines/outline.02.html`]

[2] Favio E. Miranda, Elisa Viso, *Matemáticas Discretas*, Las Prensas de Ciencias, Segunda Edición, 2016.

[3] Raúl Rojas, *A Tutorial Introduction to the Lambda Calculus*, ArXiv, 2015.

[4] Bobby Kleinberg, Notas de clase, Cornell Universitu, revisión 2012sp. Consultado el 7 de enero de 2019. [`http://www.cs.cornell.edu/courses/cs4820/2012sp/handouts/turingm.pdf`]

[5] History of Computers, *The Analitical Engige of Charles Babbage,* Consultado el 7 de enero de 2019. [`https://history-computer.com/Babbage/AnalyticalEngine.html`]

[6] José Galaviz, *Organización y Arquitectura de Computadoras,* Notas de clase, Facultad de Ciencias UNAM, revisión 2015-2.

[7] Imelda Avalos, Introducción a la programación, Universidad Autónoma de Nayarit, Consultado el 7 de enero de 2019. [`http://correo.uan.edu.mx/~iavalos/introprog.htm#Lenguajes`]

[8] EcuRed, *Enciclopedia Cubana*. Consultado el 7 de enero de 2019. [`https://www.ecured.cu/`]

[9] Shriram Krishnamurthi, *Programming Languages: Application and Interpretation*, Primera Edición, 2007.

[10] Sitio oficial de significados. [`https://www.significados.com/sintaxis/`]

[11] Codeforwin, *Classification of programming languages*, Consultado el 10 de julio de 2019. [`https://codeforwin.org/2017/05/programming-languages-classification.html`]

[12] Keith, London, *4, Programming Introduction to Computers. 24 Russell Square London WC1: Faber and Faber Limited.* The 'high' level programming languages are often called autocodes and the processor program, a compiler.

[13] Stephen, Levy. *Hackers: Heroes of the Computer Revolution*, Penguin Books. 1994.

[14] Favio E. Miranda, A. Liliana Reyes, et. al. *La Máquina de Turing en el ámbito de los Lenguajes de Programación*, Miscelanea Matemática Vol. 56, pp. 145-178. Sociedad Matemática Mexicana. 2013.