Lenguajes del Programación
Unidad 1: Introducción

Componentes de los Lenguajes de Programación

Manuel Soto Romero*

Semestre 2026-1
Facultad de Ciencias UNAM

En el estudio de la teoría de lenguajes de programación, es esencial comprender los componentes fundamentales que los constituyen a dichos lenguajes: sintaxis, semántica y pragmática:

- ⋆ La sintaxis define las reglas y estructuras que permiten escribir código de manera correcta y coherente estableciendo el esqueleto del lenguaje.
- ⋆ La semántica por su parte, se enfoca en el significado y la lógica detrás de esas estructuras, asegurando que las instrucciones se interpreten y ejecuten de manera adecuada, produciendo los resultados esperados.
- ⋆ La pragmática aborda el uso práctico del lenguaje, incluyendo los *idioms* o patrones de uso común que optimizan la eficiencia y claridad del código, y las bibliotecas que proporcionan soluciones previamente construidas para problemas frecuentes.

En esta nota estudiaremos y ejemplificaremos de forma breve cada uno de estos conceptos, lo cual no sólo nos facilitará el aprendizaje y el uso adecuado de un lenguaje de programación, sino que también potenciará nuestras habilidades al programar.

1. Sintaxis

Definición 1. (*Sintaxis de un lenguaje de programación*)

Definimos, de manera informal, la ***sintaxis*** *de un lenguaje de programación como el conjunto de reglas y estructuras que especifican cómo deben escribirse los símbolos y palabras reservadas de un programa para que sea aceptado por el traductor del lenguaje.*

Estas reglas determinan la forma correcta de expresiones, instrucciones y otros constructores de los lenguajes, asegurando que el código fuente siga un formato estructuralmente coherente. Debido a esto necesitamos de herramientas formales y notaciones que nos permitan especificar la sintaxis eliminando ambigüedades. Algunas de éstas son:

- ⋆ Gramáticas formales
- ⋆ Árboles de sintaxis

*Un agradecimiento a Juan Mario Sosa Romo por sus aportes durante la revisión de esta nota como parte de su servicio social.

- ⋆ Analizadores léxicos y sintácticos

Con estas herramientas es posible definir y verificar de manera precisa y formal la sintaxis de un lenguaje de programación asegurando que los programas escritos en dichos lenguajes sigan un formato coherente y comprensible por las máquinas. Aunque hablaremos de estos conceptos durante la primera parte del curso, la profundización en su estudio es más adecuada en un curso de Compiladores donde se revisa cómo aplicar expresiones regulares y gramáticas formales para construir analizadores léxicos y sintácticos.

Ejemplo 1

Tenemos el siguiente programa escrito en el lenguaje de programación Python:

```
def saludo(nombre):
  print("¡Hola, " + nombre + "!")
```

En este código, la sintaxis de Python especifica las siguientes reglas:

- ⋆ Al definir funciones se usa la palabra reservada `def` seguida del nombre de la función y dos paréntesis que pueden contener parámetros. La línea termina con dos puntos.
- ⋆ El cuerpo de la función debe tener un nivel de sangría adicional para indicar que las líneas de código pertenecen a la función.
  En Python, estos niveles de sangría son esenciales y forman parte de la sintaxis.
- ⋆ Dentro del cuerpo de la función, se llama a otra que toma un argumento como entrada.

El ejemplo anterior ilustra la especificación informal de algunas características sintácticas de Python. Profundizaremos más adelante, en otra nota, en cómo realizar una especificación formal de la sintaxis de un lenguaje de programación.

2. Semántica

Definición 2. (*Semántica de un lenguaje de programación*)

Definimos, de manera informal, la ***semántica*** *de un lenguaje de programación como el significado que se le da a las construcciones sintácticas definidas por el lenguaje.*

Mientras que la sintaxis se ocupa de las reglas y la estructura correcta del código, la semántica se enfoca en lo que dicho código hace cuando se ejecuta. En otras palabras, la semántica determina el comportamiento del programa: cómo se interpretan y qué efectos tienen las instrucciones y expresiones del lenguaje sobre el estado del programa y sus resultados.

Para especificar formalmente la semántica de un lenguaje de programación, se utilizan varias herramientas y enfoques. Los más comunes son:

- ⋆ Semántica operacional
- ⋆ Semántica denotativa

- ⋆ Semántica axiomática

En este curso profundizaremos en cómo realizar estas formalizaciones. Aunque la semántica de un lenguaje de programación puede darse de manera informal mediante explicaciones en lenguaje natural o el desarrollo de manuales de uso para los distintos lenguajes, la formalización tiene grandes beneficios, como:

- ⋆ Una especificación formal elimina ambigüedades y proporciona una descripción precisa y clara del comportamiento del lenguaje, lo cual es de gran importancia para las personas que programan y diseñan lenguajes, así cómo para la implementación de traductores de lenguaje.

- ⋆ Facilita la verificación y validación de programas, permitiendo demostrar formalmente que un programa cumple con su especificación. Esto es esencial en contextos críticos donde los errores pueden tener consecuencias graves.

- ⋆ Asegura que el comportamiento del lenguaje sea consistente en diferentes plataformas y entornos de implementación, lo que es fundamental para lograr la portabilidad del software.

- ⋆ Proporciona una base teórica para optimizaciones y mejoras tanto del lenguaje como de sus implementaciones. Las técnicas formales permiten identificar oportunidades para optimizaciones seguras y efectivas.

- ⋆ Permite el desarrollo de herramientas automáticas, como analizadores estáticos, verificadores de tipos y generadores de código (con o sin IAG[1]), que dependen de una comprensión formal del comportamiento del lenguaje.

La forma en que estudiaremos estas formalizaciones será diseñando pequeños lenguajes de programación y formalizando su semántica de distintas maneras, nos ocuparemos de esto más adelante.

Ejemplo 2

Tenemos ahora el siguiente programa escrito en el lenguaje de programación Python:

```
x = 5
y = 10
z = x + y
print(z)
```

En este código, la semántica de cada línea de código es la siguiente:

- ⋆ `x = 5` Asigna el valor entero `5` a la variable `x`. Esto significa que `x` ahora tiene como valor `5`.
- ⋆ `y = 10` Asigna el valor entero `10` a la variable `y`. Esto significa que `y` ahora tiene como valor `10`.
- ⋆ `z = x + y` Suma los valores `x` y `y`, que son `5` y `10` respectivamente, y asigna el resultado a la variable `z`. Esto significa que `z` ahora contiene el valor `15`.
- ⋆ `print(z)` Envía el valor de `z` a la salida estándar. Como `z` contiene `15`, se imprimirá `15`.

[1] Inteligencia Artificial Generativa

3. Pragmática

Definición 3. (*Pragmática de un lenguaje de programación*)

Definimos, de manera informal, la ***pragmática*** *de un lenguaje de programación como la manera en que éstos se usan para resolver problemas del mundo real.*

Incluye el conocimiento sobre las convenciones, los patrones de uso común (*idioms*), y las bibliotecas y herramientas disponibles que facilitan la programación de manera eficiente y efectiva. Abarca las mejores prácticas y las estrategias empleadas por las personas que programan para escribir código que no sólo resuelva problemas, sino que también sea legible, modificable y eficiente. Otros ejemplos de pragmática son:

- ⋆ Patrones de diseño
- ⋆ Convenciones para el nombrado de identificadores
- ⋆ Control de versiones
- ⋆ Pruebas unitarias
- ⋆ Documentación
- ⋆ Gestión de dependencias
- ⋆ Reestructuración[2] de código
- ⋆ Uso de bibliotecas y marcos de trabajo[3]
- ⋆ Buenas prácticas de programación

Aunque son un componente importante de los lenguajes de programación, su estudio es más adecuado para los cursos del área de Ingeniería de Software por lo tanto no discutiremos mucho sobre la pragmática en este curso.

Ejemplo 3

Supongamos que estamos desarrollando una aplicación web en Python. En lugar de construir todo desde cero, decidimos usar Django, un marco de trabajo web de alto nivel. Esta decisión es un ejemplo de pragmática donde Django ofrece un conjunto de herramientas y convenciones que facilitan y aceleran el desarrollo.

Al utilizar Django, se puede desarrollar una aplicación web funcional y segura de forma rápida y con menos errores, aprovechando las herramientas y convenciones que ofrece el marco de trabajo. Esto es un enfoque pragmático que maximiza la eficiencia y la calidad del desarrollo.

[2] *Refactoring*
[3] *Frameworks*

4. Conclusión

Comprender los componentes fundamentales de los lenguajes de programación (sintaxis, semántica y pragmática) permite no sólo escribir código correcto, sino también entender su significado y aplicarlo con eficacia en contextos reales. La sintaxis establece las reglas de construcción, la semántica proporciona el significado de esas construcciones, y la pragmática guía su uso eficiente en la práctica profesional. Esta perspectiva integral no sólo facilita el aprendizaje de nuevos lenguajes, sino que también sienta las bases para diseñarlos, analizarlos y enseñar su uso de forma rigurosa y consciente.

Referencias

[1] S. Krishnamurthi, *Programming Languages: Application and Interpretation*. 2007.
[2] B. C. Pierce, *Types and Programming Languages*. MIT Press, 2002.
[3] M. Gabbrielli y S. Martini, *Programming Languages: Principles and Paradigms*. Springer, 2023.
[4] G. Winskel, *The Formal Semantics of Programming Languages: An Introduction*. MIT Press, 1993.
[5] H. R. Nielson y F. Nielson, *Semantics with Applications: An Appetizer*. Springer, 2007.
[6] K. D. Lee, *Foundations of Programming Languages*. Springer, 2017.