Introducción a la Programación

con Python

Noel Moreno Lemus, Ph.D.

Table of contents

Introducción

Introducción a la Programación *con Python* es un libro diseñado para aquellas personas que desean iniciarse en el mundo de la programación desde cero. En este libro te guiaremos a través de los conceptos básicos de la programación y te mostraremos cómo aplicarlos para crear programas simples y útiles.

> **Nota**
>
> Ya sea que estés interesado en la programación por diversión, para desarrollar habilidades profesionales o simplemente para comprender mejor la tecnología que te rodea, este libro es un excelente punto de partida.

¿Por qué aprender a programar?

Si quieres entender al menos un poco el mundo que te rodea es importante tener conocimientos básicos de matemáticas, física, química y biología. A esta lista ahora se suman las ciencias de la computación pues vivimos en un mundo cada vez más domidado por la tecnología, y tenemos dos opciones:

1. **la usamos para nuestro veneficio**
2. **la tecnología nos usa a nosotros**

Es por eso que, aunque solo sea con la finalidad de entender como funciona todo a nuestro alrededor, es importante tener nociones de ciencias de la computación en general y de programación en particular.

Dicho esto, hablemos un poco de la estructura de este libro y del lenguaje de programación que hemos elejido para escribir los códigos de los ejemplos del mismo.

Estructura del libro

Introducción a la Programación con Python está dividido en 9 capítulos:

1. Introducción a la programación y al lenguaje Python
    - 1.1. Concepto de programación
    - 1.2. Historia de la programación
    - 1.3. Lenguaje Python: características y ventajas
    - 1.4. Instalación del ambiente de desarrollo (Anaconda)
    - 1.5. Ambientes Virtuales en Python
2. Sistemas numéricos y representación de datos
    - 2.1. Sistema binario
    - 2.2. Sistema hexadecimal
    - 2.3. Conversión entre sistemas numéricos
    - 2.4. Representación de números enteros y reales en computadoras
3. Lenguaje máquina y lenguaje ensamblador
    - 3.1. Concepto de lenguaje máquina
    - 3.2. Estructura de una instrucción
    - 3.3. Lenguaje ensamblador: concepto y uso

4. Lenguajes de alto nivel y Python

    - 4.1. Características de los lenguajes de alto nivel
    - 4.2. Introducción a Python: sintaxis básica, variables y tipos de datos

5. Herramientas de control de flujo

    - 5.1. Condicionales: if, elif, else
    - 5.2. Bucles: for y while
    - 5.3. Funciones y parámetros

6. Estructuras de datos

    - 6.1. Listas y tuplas
    - 6.2. Diccionarios y conjuntos
    - 6.3. Operaciones comunes en estructuras de datos

7. Algoritmos de ordenamiento y búsqueda

    - 7.1. Algoritmo de ordenamiento por selección
    - 7.2. Algoritmo de ordenamiento por inserción
    - 7.3. Algoritmo de ordenamiento por burbuja
    - 7.4. Algoritmo de búsqueda secuencial
    - 7.5. Algoritmo de búsqueda binaria

8. Introducción a la complejidad algorítmica

    - 8.1. Notación Big O
    - 8.2. Análisis de tiempo y espacio
    - 8.3. Comparación de la eficiencia de algoritmos

9. Introducción a las funciones recursivas

    - 9.1. Concepto de recursión
    - 9.2. Casos base y recursivos
    - 9.3. Ejemplos de funciones recursivas en Python

¿Por qué Python?

Python es un lenguaje fácil de aprender y de usar, tiene una sintaxis clara y legible, amplia comunidad y recursos, versatilidad, enfoque en la eficiencia y facilidad de instalación y configuración. Además y gracias su uso extensivo en Inteligencia Artificial, Machine Learning y Ciencia de Datos; se ha convertido en uno de los lenguajes más usados en el mundo y de los más demandados por las empresas. *Podríamos decir que si sabes Python nunca te faltaran las oportunidades de trabajo.*

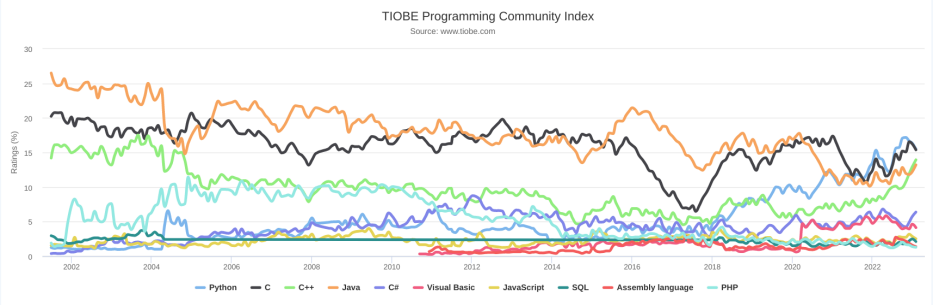

Figure 1: Python ha ido ganando popularidad en los últimos años (no ha parado de crecer desde el 2018), llegando a desplazar en los últimos meses a lenguajes como Java y C.

> **Work in Progress**
>
> This is a Work in Progress project. The idea is to summarize all ML topics and how I see and understand them.

Machine learning is a rapidly growing field that enables computers to learn from data, without being explicitly programmed. The goal of machine learning is to build models that can make predictions or take actions based on input data, and improve their performance over time through experience.

> **Note**
>
> Note that there are five types of callouts, including: `note`, `warning`, `important`, `tip`, and `caution`.

> **Tip With Caption**
>
> This is an example of a callout with a caption.

> **Expand To Learn About Collapse**
>
> This is an example of a 'folded' caution callout that can be expanded by the user. You can use `collapse="true"` to collapse it by default or `collapse="false"` to make a collapsible callout that is expanded by default.

Versiones usadas en este libro

```
import sys
print("Python version: {}".format(sys.version))
import pandas as pd
print("pandas version: {}".format(pd.__version__))
import matplotlib
print("matplotlib version: {}".format(matplotlib.__version__))
import numpy as np
print("NumPy version: {}".format(np.__version__))
import scipy as sp
print("SciPy version: {}".format(sp.__version__))
import IPython
print("IPython version: {}".format(IPython.__version__))
```

```
import sklearn
print("scikit-learn version: {}".format(sklearn.__version__))
```

```
Python version: 3.9.16 (main, Mar  8 2023, 14:00:05)
[GCC 11.2.0]
pandas version: 2.0.0
matplotlib version: 3.7.1
NumPy version: 1.24.2
SciPy version: 1.10.1
IPython version: 8.12.0

scikit-learn version: 1.2.2
```

1 Introducción a la programación y al lenguaje Python

Al final de este capítulo, los estudiantes entenderán los conceptos básicos de la programación y la importancia de los lenguajes de programación. También, aprenderán sobre las características y ventajas del lenguaje Python, cómo instalar el ambiente de desarrollo utilizando Anaconda y cómo trabajar con ambientes virtuales para administrar proyectos en Python.

1.1 ¿Qué es la programación?

Al final de esta sección, comprenderán el concepto de programación y conocerán la diferencia entre lenguajes de programación, algoritmos y pseudocódigo, así como los procesos de interpretación y compilación para la ejecución de programas.

1.1.1 Definición de programación

Según Wikipedia la programación es el proceso de crear un conjunto de instrucciones que le dicen a una computadora como realizar algún tipo de tarea. Pero no solo la acción de escribir un código para que la computadora o el software lo ejecute. Incluye, además, todas las tareas necesarias para que el código funcione correctamente y cumpla el objetivo para el cual se escribió.

1.1.2 Lenguajes de programación

1.1.3 Algoritmos y pseudocódigo

Un algoritmo es una secuencia finita de pasos, reglas o instrucciones bien definidas, que se utilizan para resolver un problema o llevar a cabo una tarea específica. Los algoritmos se pueden representar mediante diagramas de flujo, pseudocódigo u otros métodos. El pseudocódigo es una representación de alto nivel de un algoritmo, que utiliza una estructura similar a un lenguaje de programación pero sin seguir las reglas de sintaxis específicas. El pseudocódigo permite a los programadores diseñar y analizar algoritmos sin preocuparse por los detalles de implementación en un lenguaje de programación específico.

Ejemplo de pseudocódigo para calcular la suma de los números del 1 al N:

```
Inicio
   Leer N
   Suma = 0
   Para i = 1 hasta N:
       Suma = Suma + i
   Fin Para
   Escribir Suma
Fin
```

1.1.4 Ejecución de programas

La ejecución de programas implica la interpretación y ejecución de las instrucciones escritas en un lenguaje de programación por parte de una computadora. Dependiendo del lenguaje de programación utilizado, existen dos enfoques principales para ejecutar programas: interpretación y compilación.

- Interpretación: Los lenguajes interpretados, como Python, utilizan un programa llamado intérprete que lee y ejecuta el código fuente línea por línea, convirtiendo las instrucciones en lenguaje de máquina que la computadora puede entender y ejecutar. Los programas interpretados suelen ser más lentos en la ejecución pero más rápidos en el desarrollo y depuración, ya que no es necesario compilar el código.
- Compilación: Los lenguajes compilados, como C++, requieren un programa llamado compilador que convierte el código fuente en un archivo ejecutable en lenguaje de máquina antes de que pueda ser ejecutado por la computadora. Los programas compilados suelen ser más rápidos en la ejecución pero requieren un proceso de compilación antes de ser ejecutados, lo que puede aumentar el tiempo de desarrollo y depuración.

1.2 Historia de la programación

1.2.1 Primeras máquinas programables

El inicio de la programación se remonta a las primeras máquinas capaces de realizar cálculos matemáticos. A continuación, se mencionan algunas de estas máquinas:

- La Máquina Analítica de Charles Babbage (1837): Babbage diseñó esta máquina mecánica para realizar cálculos matemáticos automáticamente utilizando tarjetas perforadas, que contenían las instrucciones para los cálculos. Aunque la Máquina Analítica nunca fue construida completamente, sus conceptos influyeron en el diseño de futuras computadoras.
- El Telar de Jacquard (1801): El telar de Jacquard, inventado por Joseph Marie Jacquard, fue una innovación en la industria textil que permitía controlar automáticamente el patrón de tejido utilizando

tarjetas perforadas. Aunque no era una máquina de cómputo, el concepto de tarjetas perforadas influyó en el almacenamiento y lectura de datos en las primeras computadoras.

- La Máquina de Turing (1936): Alan Turing propuso la Máquina de Turing como un modelo teórico para resolver problemas matemáticos y lógicos mediante la manipulación de símbolos en una cinta infinita. La Máquina de Turing es considerada como el modelo básico para las computadoras digitales modernas y la base de la teoría de la computación.

1.2.2 Evolución de los lenguajes de programación

La evolución de los lenguajes de programación ha sido una parte fundamental en el avance de la informática. Los lenguajes de programación han pasado por varias generaciones, desde el código máquina y ensamblador hasta los lenguajes de alto nivel que facilitan la escritura de programas.

- 1ª Generación: Lenguaje máquina (1940-1950): Las primeras computadoras utilizaban código máquina, una serie de números binarios que representaban instrucciones específicas para la máquina. Los programadores debían escribir programas en código máquina, lo cual era un proceso tedioso y propenso a errores.
- 2ª Generación: Lenguaje ensamblador (1950-1960): El lenguaje ensamblador fue un avance que permitía a los programadores escribir instrucciones utilizando mnemónicos, que eran más fáciles de entender y recordar que el código máquina. Los programas escritos en lenguaje ensamblador eran luego traducidos a código máquina por un ensamblador.
- 3ª Generación: Lenguajes de alto nivel (1950-presente): Los lenguajes de alto nivel, como FORTRAN, COBOL, LISP y ALGOL, permitieron a los programadores escribir instrucciones utilizando palabras

y símbolos más cercanos al lenguaje humano. Estos lenguajes facilitaron el desarrollo de programas y permitieron la creación de aplicaciones más complejas. Los lenguajes de alto nivel evolucionaron y dieron origen a otros lenguajes populares como C, Pascal, C++, Java, Python y JavaScript.

1.2.3 Paradigmas de programación

Los paradigmas de programación representan diferentes enfoques o estilos para organizar y estructurar programas informáticos. Algunos de los paradigmas de programación más comunes son:

- Programación imperativa: En este paradigma, los programas consisten en secuencias de instrucciones que modifican el estado del programa. Los programas imperativos suelen utilizar estructuras de control como bucles, condicionales y funciones. Ejemplos de lenguajes que soportan la programación imperativa incluyen C, C++, Java y Python.

- Programación declarativa: A diferencia de la programación imperativa, que se centra en cómo realizar una tarea, la programación declarativa se centra en describir qué se quiere obtener sin especificar cómo lograrlo. Un subconjunto de la programación declarativa es la programación funcional, que evita el uso de estados y datos mutables. Ejemplos de lenguajes de programación declarativa incluyen SQL, Prolog y Haskell.

- Programación orientada a objetos (OOP): La programación orientada a objetos es un enfoque que organiza el código en "objetos" que representan entidades del mundo real, con propiedades (atributos) y comportamientos (métodos). Este paradigma promueve la encapsulación, la herencia y el polimorfismo, lo que facilita el desarrollo de

sistemas complejos y la reutilización del código. Ejemplos de lenguajes que soportan la programación orientada a objetos incluyen Java, C++, C# y Python.

- Programación estructurada: La programación estructurada es un enfoque que organiza el código utilizando estructuras de control como secuencias, selección (condicionales) y repetición (bucles), evitando el uso de saltos incondicionales como "goto". La programación estructurada mejora la legibilidad y la mantenibilidad del código. Ejemplos de lenguajes que soportan la programación estructurada incluyen C, Pascal y Ada.

- Programación lógica: En la programación lógica, los programas consisten en una serie de declaraciones o hechos que describen relaciones y reglas lógicas. Los programas lógicos se ejecutan mediante un motor de inferencia que deduce nuevas conclusiones o resultados a partir de las declaraciones y reglas proporcionadas. La programación lógica es especialmente útil en la inteligencia artificial y la resolución de problemas que involucran relaciones complejas y restricciones. Un ejemplo de lenguaje de programación lógica es Prolog.

- Programación concurrente: La programación concurrente es un enfoque que permite la ejecución simultánea de múltiples tareas o procesos, lo cual es útil en aplicaciones que requieren la gestión de múltiples usuarios o eventos simultáneos. La programación concurrente se ocupa de problemas como la sincronización, la comunicación y la compartición de recursos entre tareas concurrentes. Ejemplos de lenguajes que soportan la programación concurrente incluyen Java, C#, Go y Erlang.

Cabe destacar que muchos lenguajes de programación modernos soportan múltiples paradigmas, lo que permite a los programadores elegir el enfoque más adecuado para cada problema o tarea.

1.3 Lenguaje Python: características y ventajas

1.3.1 Historia de Python

Python fue creado por Guido van Rossum a finales de la década de 1980 y su primera versión pública (Python 0.9.0) fue lanzada en 1991. Van Rossum quería crear un lenguaje de programación fácil de leer, escribir y mantener, con una sintaxis clara y simple. Python fue influenciado por lenguajes como ABC, Modula-3 y C. Con el tiempo, Python ha evolucionado y ganado popularidad, convirtiéndose en uno de los lenguajes de programación más utilizados en la actualidad.

1.3.2 Características del lenguaje Python

Python es un lenguaje de programación de alto nivel con una serie de características distintivas:

- Sintaxis clara y legible: La sintaxis de Python se basa en la indentación, lo que facilita la lectura y comprensión del código, incluso para principiantes.
- Tipado dinámico: Python es un lenguaje de tipado dinámico, lo que significa que las variables pueden cambiar de tipo en tiempo de ejecución. Esto permite mayor flexibilidad en la escritura de código.
- Orientación a objetos: Python es un lenguaje orientado a objetos, lo que permite la creación de clases y objetos para modelar y organizar el código de manera más estructurada.
- Biblioteca estándar extensa: Python incluye una amplia biblioteca estándar con módulos y funciones que facilitan la realización de tareas comunes como el manejo de archivos, comunicación en red y operaciones matemáticas.

- Multiplataforma: Python es compatible con diferentes sistemas operativos como Windows, macOS y Linux.
- Interactivo: Python permite la ejecución interactiva de código a través de un intérprete, lo que facilita la depuración y el aprendizaje del lenguaje.

1.3.3 Ventajas y aplicaciones de Python

Python ofrece varias ventajas que lo convierten en una opción popular para diferentes aplicaciones:

- Facilidad de aprendizaje: La sintaxis simple y legible de Python lo convierte en un lenguaje ideal para principiantes en programación.
- Versatilidad: Python es un lenguaje multiparadigma que se adapta a diferentes estilos de programación y puede utilizarse en diversos tipos de proyectos, desde aplicaciones web hasta análisis de datos y desarrollo de videojuegos.
- Comunidad activa: Python cuenta con una amplia comunidad de desarrolladores que contribuyen con bibliotecas y paquetes adicionales para extender las capacidades del lenguaje, así como con tutoriales, cursos y documentación para facilitar el aprendizaje de Python.
- Integración con otros lenguajes: Python puede interactuar con otros lenguajes de programación como C, C++ y Java, lo que permite la reutilización de código y la optimización de ciertas partes del programa.

1.3.3.1 Algunas aplicaciones comunes de Python incluyen:

- Desarrollo web: Con frameworks como Django y Flask, Python es una opción popular para crear aplicaciones web escalables y eficientes.

- Análisis de datos y ciencia de datos: Python es ampliamente utilizado en análisis de datos, inteligencia artificial y aprendizaje automático, gracias a bibliotecas como NumPy, pandas, TensorFlow y scikit-learn.

- Automatización y scripting: Python es ideal para automatizar tareas repetitivas y crear scripts para simplificar procesos, gracias a su facilidad de uso y a su extensa biblioteca estándar.

- Desarrollo de videojuegos: Python se utiliza en el desarrollo de videojuegos con la ayuda de bibliotecas y motores como Pygame y Panda3D.

- Desarrollo de aplicaciones de escritorio: Con bibliotecas como PyQt y Tkinter, Python permite crear aplicaciones de escritorio con interfaces gráficas de usuario.

- Internet de las cosas (IoT): Python también se utiliza en el desarrollo de aplicaciones para dispositivos IoT, gracias a su compatibilidad con plataformas como Raspberry Pi y su facilidad de integración con hardware.

1.3.4 Comunidad y recursos de Python

La comunidad de Python es uno de los aspectos más valiosos del lenguaje. Con una amplia base de usuarios y desarrolladores, es fácil encontrar ayuda, colaboración y recursos para aprender y mejorar en Python. Algunos de los recursos más relevantes incluyen:

- Documentación oficial de Python: La documentación oficial de Python (https://docs.python.org) es un recurso completo que incluye tutoriales, guías y referencias para todos los aspectos del lenguaje y su biblioteca estándar.

- Python Package Index (PyPI): PyPI (https://pypi.org) es un repositorio de paquetes y bibliotecas de terceros desarrollados por la comunidad de Python. Aquí puedes encontrar soluciones y extensiones para una amplia variedad de problemas y aplicaciones.
- Foros y comunidades en línea: La comunidad de Python es activa en foros como Stack Overflow (https://stackoverflow.com) y Reddit (https://www.reddit.com/r/Python), donde puedes hacer preguntas, compartir conocimientos y colaborar con otros desarrolladores.
- Conferencias y eventos: La comunidad de Python organiza eventos y conferencias en todo el mundo, como PyCon (https://www.pycon.org) y EuroPython (https://www.europython.eu), donde los desarrolladores pueden aprender, colaborar y conectarse con otros profesionales y entusiastas del lenguaje.
- Cursos y tutoriales: Existen numerosos cursos y tutoriales en línea, tanto gratuitos como de pago, que cubren diversos aspectos de Python, desde conceptos básicos hasta temas avanzados como ciencia de datos y desarrollo web. Algunos recursos populares incluyen Codecademy (https://www.codecademy.com), Coursera (https://www.coursera.org) y edX (https://www.edx.org).

1.4 Instalación del ambiente de desarrollo (Anaconda)

1.4.1 ¿Qué es Anaconda?

Anaconda es una distribución gratuita y de código abierto de los lenguajes de programación Python y R, que se utiliza ampliamente en la ciencia de datos, el aprendizaje automático y la computación científica. Anaconda simplifica la gestión de paquetes y entornos, proporcionando una plataforma unificada para instalar y administrar múltiples versiones de

paquetes y sus dependencias. Además, Anaconda incluye herramientas como Jupyter Notebook y JupyterLab, que facilitan el desarrollo y la colaboración en proyectos de análisis de datos y computación científica.

1.4.2 Instalación de Anaconda en diferentes sistemas operativos (Windows, macOS, Linux)

Para instalar Anaconda en tu sistema operativo, sigue los pasos a continuación:

Visita la página de descargas de Anaconda y selecciona la versión adecuada para tu sistema operativo (Windows, macOS o Linux).

Descarga el instalador de Anaconda correspondiente a tu sistema operativo.

Ejecuta el instalador y sigue las instrucciones en pantalla. Durante la instalación, se te preguntará si deseas instalar Anaconda para todos los usuarios del sistema o solo para tu usuario actual. Se recomienda instalar Anaconda para todos los usuarios si tienes permisos de administrador en tu computadora.

Windows: Ejecuta el archivo .exe descargado y sigue las instrucciones en pantalla.

macOS: Ejecuta el archivo .pkg descargado y sigue las instrucciones en pantalla.

Linux: Abre una terminal, navega hasta el directorio donde se encuentra el archivo .sh descargado y ejecuta el siguiente comando: bash `nombre_del_archivo.sh`, donde `nombre_del_archivo.sh` es el nombre del archivo descargado. Sigue las instrucciones en pantalla.

1.4.3 Verificación de la instalación y actualización de Anaconda

Una vez que hayas instalado Anaconda, es importante verificar que se haya instalado correctamente y actualizarlo a la última versión. Para hacer esto, sigue los pasos a continuación:

- Abre una terminal (o Anaconda Prompt en Windows) y ejecuta el siguiente comando para verificar que Anaconda está instalado correctamente: `conda --version`. Deberías ver la versión actual de conda en la respuesta.
- Para actualizar Anaconda a la última versión, ejecuta el siguiente comando: `conda update --all`. Este comando actualizará todos los paquetes y dependencias a sus últimas versiones compatibles.

1.4.4 Introducción a Jupyter Notebook y JupyterLab

Jupyter Notebook y JupyterLab son entornos de desarrollo interactivo que permiten crear y compartir documentos que contienen código, texto, imágenes y gráficos en vivo. Estas herramientas facilitan la exploración de datos, el análisis y la comunicación de resultados en un formato fácil de entender y compartir.

- Jupyter Notebook: Para iniciar Jupyter Notebook, abre una terminal (o Anaconda Prompt en Windows) y ejecuta el comando jupyter notebook. Esto abrirá una nueva ventana o pestaña en tu navegador web con la interfaz de Jupyter Notebook. Desde aquí, puedes crear nuevos cuadernos, abrir cuadernos existentes y administrar tus archivos
- JupyterLab: JupyterLab es una evolución de Jupyter Notebook que ofrece una experiencia de usuario más avanzada e integrada. Para iniciar JupyterLab, abre una terminal (o Anaconda Prompt en Windows) y ejecuta el comando jupyter lab. Esto abrirá una nueva

ventana o pestaña en tu navegador web con la interfaz de JupyterLab. JupyterLab presenta una interfaz similar a la de un entorno de desarrollo integrado (IDE), con una barra lateral que facilita la navegación y organización de tus archivos, así como la posibilidad de abrir múltiples cuadernos y otros documentos en pestañas.

Ambos entornos, Jupyter Notebook y JupyterLab, permiten ejecutar código en celdas individuales e interactivas, lo que facilita la experimentación y depuración. Además, puedes agregar texto en formato Markdown para proporcionar explicaciones, fórmulas matemáticas y visualizaciones en línea con tu código.

Con Jupyter Notebook y JupyterLab, puedes crear documentos completos y autocontenidos que incluyan tanto el código como la explicación de tus análisis y resultados, lo que facilita la colaboración y la comunicación de tus proyectos. Estos documentos se pueden exportar a diversos formatos, como HTML, PDF y presentaciones de diapositivas, para compartirlos fácilmente con otras personas.

En resumen, la instalación de Anaconda proporciona un entorno de desarrollo completo y unificado para trabajar con Python, especialmente en proyectos de ciencia de datos y análisis. Con herramientas como Jupyter Notebook y JupyterLab, puedes explorar y analizar datos de manera interactiva y eficiente, así como compartir tus resultados y conocimientos de manera clara y accesible.

1.5 Ambientes Virtuales en Python

1.5.1 ¿Qué es un ambiente virtual?

Un ambiente virtual en Python es un entorno aislado en el que puedes instalar y administrar paquetes de manera independiente, sin afectar otros proyectos o el entorno global de Python en tu sistema. Cada ambiente

virtual tiene su propia versión de Python, así como sus propios paquetes y dependencias, lo que permite mantener diferentes configuraciones para diferentes proyectos.

1.5.2 Ventajas y uso de los ambientes virtuales

Los ambientes virtuales ofrecen varias ventajas, especialmente en proyectos de desarrollo y colaboración:

Separación de dependencias: Cada proyecto puede tener sus propias versiones de paquetes y dependencias, sin interferir con otros proyectos o con el entorno global de Python.

Facilita la colaboración: Al utilizar ambientes virtuales, los desarrolladores pueden compartir fácilmente las configuraciones de sus proyectos con otros colaboradores, lo que garantiza que todos trabajen con las mismas versiones de paquetes y dependencias.

Facilita la implementación y distribución: Los ambientes virtuales permiten empaquetar y distribuir proyectos con sus dependencias específicas, lo que facilita la implementación en diferentes entornos y sistemas.

1.5.3 Creación y activación de ambientes virtuales en Python

Para crear y activar un ambiente virtual en Python, sigue los pasos a continuación:

Abre una terminal (o Anaconda Prompt en Windows).

Navega hasta el directorio donde deseas crear el ambiente virtual.

Ejecuta el siguiente comando para crear un ambiente virtual con el nombre "nombre_del_ambiente": `python -m venv nombre_del_ambiente`. Puedes reemplazar "nombre_del_ambiente" con el nombre que prefieras para tu ambiente virtual.

Para activar el ambiente virtual, ejecuta uno de los siguientes comandos, según tu sistema operativo:

Windows: `nombre_del_ambiente\Scripts\activate`

macOS y Linux: `source nombre_del_ambiente/bin/activate`

Una vez activado, el nombre del ambiente virtual aparecerá entre paréntesis al comienzo del indicador de la terminal, indicando que estás trabajando en ese ambiente.

Para desactivar el ambiente virtual y volver al entorno global de Python, simplemente ejecuta el comando deactivate.

1.5.4 Instalación y administración de paquetes en ambientes virtuales

Una vez que hayas activado un ambiente virtual, puedes instalar y administrar paquetes de manera independiente en ese ambiente. Para hacer esto, utiliza el administrador de paquetes pip, que se incluye automáticamente en cada ambiente virtual.

Para instalar un paquete en el ambiente virtual, ejecuta el siguiente comando: `pip install nombre_del_paquete`. Puedes reemplazar "nombre_del_paquete" con el nombre del paquete que deseas instalar.

Para listar los paquetes instalados en el ambiente virtual, ejecuta el comando `pip list`.

Para actualizar un paquete a la última versión compatible, ejecuta el siguiente comando: `pip install --upgrade nombre_del_paquete`.

Para desinstalar un paquete del ambiente virtual, ejecuta el siguiente comando: `pip uninstall nombre_del_paquete`.

Recuerda que estas acciones solo afectarán el ambiente virtual activo y no modificarán el entorno global de Python

2 Sistemas numéricos, lenguajes de máquina y ensamblador

2.1 Introducción a los sistemas numéricos

Los sistemas numéricos son formas de representar y trabajar con números utilizando diferentes bases. La base de un sistema numérico determina la cantidad de dígitos que se utilizan para representar números en ese sistema. A continuación, se presentan los sistemas numéricos más comunes en informática y matemáticas.

2.1.1 Sistema decimal (base 10)

El sistema decimal es el sistema numérico que utilizamos en la vida cotidiana. Es un sistema de base 10, lo que significa que utiliza 10 dígitos diferentes (del 0 al 9) para representar números. Cada posición en un número decimal representa una potencia de 10, comenzando por 10^0 en el dígito más a la derecha y aumentando hacia la izquierda. Por ejemplo, el número decimal 1234 se puede descomponer como:

$$(1 \times 10^3) + (2 \times 10^2) + (3 \times 10^1) + (4 \times 10^0) = 1000 + 200 + 30 + 4 = 1234$$

2.1.2 Sistema binario (base 2)

El sistema binario es un sistema numérico de base 2 que utiliza únicamente dos dígitos: 0 y 1. Es fundamental en la informática y la tecnología de

la información, ya que las computadoras y otros dispositivos electrónicos representan y procesan información en forma de bits, que pueden tener uno de dos valores: 0 o 1. Cada posición en un número binario representa una potencia de 2, comenzando por 2^0 en el dígito más a la derecha y aumentando hacia la izquierda. Por ejemplo, el número binario 1101 se puede descomponer como:

$(1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = 8 + 4 + 0 + 1 = 13$ (en decimal)

2.1.3 Sistema hexadecimal (base 16)

El sistema hexadecimal es un sistema numérico de base 16 que utiliza 16 dígitos diferentes para representar números. Los primeros 10 dígitos son los mismos que en el sistema decimal (0 al 9), seguidos de las letras A, B, C, D, E y F, que representan los valores del 10 al 15. El sistema hexadecimal es útil en informática porque permite representar números binarios de manera más compacta y fácil de leer. Cada dígito hexadecimal representa 4 bits o una nibble. Por ejemplo, el número hexadecimal $1A3$ se puede descomponer como:

#(1 × 16^2) + (10 × 16^1) + (3 × 16^0) = 256 + 160 + 3 = 419# (en decimal)

2.1.4 Sistema octal (base 8)

El sistema octal es un sistema numérico de base 8 que utiliza 8 dígitos diferentes (del 0 al 7) para representar números. Cada posición en un número octal representa una potencia de 8, comenzando por 8^0 en el dígito más a la derecha y aumentando hacia la izquierda. Aunque el sistema octal no es tan común como los sistemas binario y hexadecimal en la informática moderna, todavía se utiliza en ciertas aplicaciones y contextos, como en la programación de sistemas embebidos y en sistemas operativos Unix para representar permisos de archivo.

Cada dígito octal representa 3 bits, lo que permite una representación más compacta de números binarios que el sistema decimal, aunque no tan compacta como el sistema hexadecimal. Por ejemplo, el número octal 527 se puede descomponer como:

$(5 \times 8^2) + (2 \times 8^1) + (7 \times 8^0) = 320 + 16 + 7 = 343$ (en decimal)

Los sistemas numéricos son esenciales para representar y trabajar con números en diferentes contextos. En la informática, los sistemas binario y hexadecimal son especialmente importantes debido a su relación directa con la representación y manipulación de información en dispositivos electrónicos y digitales. El sistema octal, aunque menos común, también tiene aplicaciones específicas en la programación y la administración de sistemas. Conocer y comprender estos sistemas numéricos es esencial para cualquier persona que trabaje en tecnología de la información y la informática.

2.2 Uso de sistemas numéricos en las computadoras modernas

2.2.1 Representación de datos en binario

Las computadoras modernas utilizan el sistema binario para representar y almacenar datos. Veamos algunos ejemplos concretos de cómo se representan diferentes tipos de datos en binario:

- Números enteros: Los enteros se representan utilizando bits en un formato llamado "complemento a dos". Por ejemplo, el número decimal 13 se representa en binario como 1101, mientras que el número decimal -13 se representa como 11110011 (para un total de 8 bits o un byte).
- Números reales: Los números reales (números con decimales) se representan generalmente utilizando el estándar IEEE 754 de punto flotante. Por ejemplo, el número decimal 0.5 se representa como

00111111000000000000000000000000 en formato binario de punto flotante de precisión simple (32 bits).

- Caracteres: Los caracteres se representan utilizando códigos binarios, como ASCII o Unicode. Por ejemplo, el carácter "A" se representa en binario como 01000001 en ASCII (8 bits), mientras que el carácter "Ω" se representa en binario como 11000010 10101000 en Unicode (UTF-8, 16 bits).

2.2.2 Operaciones lógicas y aritméticas en binario

Las computadoras realizan operaciones lógicas y aritméticas en binario utilizando circuitos electrónicos llamados "unidades aritmético-lógicas" (ALU). Algunos ejemplos de operaciones en binario incluyen:

- Suma: La suma binaria es similar a la suma decimal, pero solo se lleva a cabo si la suma de los bits es 2. Por ejemplo, 1101 (13 en decimal) + 1010 (10 en decimal) = 10111 (23 en decimal).

- Resta: La resta en binario se realiza utilizando el complemento a dos del número que se va a restar y luego sumando los números. Por ejemplo, para calcular 1101 (13 en decimal) - 1010 (10 en decimal), primero encontramos el complemento a dos de 1010, que es 0110, y luego sumamos: 1101 + 0110 = 10011 (3 en decimal).

- AND lógico: El AND lógico compara dos números bit a bit y devuelve un 1 si ambos bits son 1, y 0 en caso contrario. Por ejemplo, 1101 AND 1010 = 1000.

- OR lógico: El OR lógico compara dos números bit a bit y devuelve un 1 si al menos uno de los bits es 1, y 0 en caso contrario. Por ejemplo, 1101 OR 1010 = 1111.

2.2.3 Direcciones de memoria y registros

Las direcciones de memoria y los registros son números binarios que se utilizan para acceder y almacenar datos en las computadoras. Por ejemplo:

- Direcciones de memoria: Cada ubicación de memoria en una computadora tiene una dirección única, que es un número binario. Por ejemplo, la dirección de memoria 0001001101010111 puede contener el valor binario 01101010.
- Registros: Los registros son pequeñas áreas de almacenamiento de alta velocidad en la CPU (unidad central de procesamiento) de una computadora. Los registros almacenan datos y direcciones de memoria utilizados en operaciones aritméticas y lógicas. Por ejemplo, un registro puede contener el valor binario 1101, que se utilizará en una operación de suma con otro valor almacenado en otro registro.

2.2.4 Transmisión de datos y protocolos de comunicación

La transmisión de datos entre componentes de una computadora o entre computadoras y dispositivos externos se realiza utilizando protocolos de comunicación, que definen cómo se representan y transmiten los datos en forma binaria. Algunos ejemplos de protocolos de comunicación y cómo utilizan datos binarios incluyen:

- USB (Universal Serial Bus): El protocolo USB se utiliza para conectar dispositivos externos a una computadora, como teclados, ratones y unidades de almacenamiento. Los datos se transmiten en paquetes binarios a través de cables USB. Por ejemplo, un paquete USB puede contener una secuencia binaria como 0101010111010010, que representa una serie de datos o comandos transmitidos entre la computadora y el dispositivo.

- Ethernet: Ethernet es un protocolo de comunicación utilizado para redes de área local (LAN) que interconectan computadoras y dispositivos. Los datos se transmiten en tramas (frames) binarias que contienen información como direcciones de origen y destino, así como datos de carga útil. Por ejemplo, una trama Ethernet puede contener una secuencia binaria como 10101010101010101011010101010101, que representa una serie de datos o comandos transmitidos entre dispositivos en la red.
- Wi-Fi: Wi-Fi es un protocolo de comunicación inalámbrica utilizado para conectar dispositivos a redes de área local (LAN) y a Internet. Los datos se transmiten en paquetes binarios utilizando ondas de radio. Por ejemplo, un paquete Wi-Fi puede contener una secuencia binaria como 11010101010101010110101011, que representa una serie de datos o comandos transmitidos entre dispositivos en la red.

2.3 Ejemplos detallados de sistemas numéricos

2.3.1 Conversión entre sistemas numéricos

En esta sección, se presentan ejemplos detallados de la conversión entre diferentes sistemas numéricos, como decimal, binario, octal y hexadecimal.

Ejemplo 1: Conversión de decimal a binario Número decimal: 25 Número binario: 11001

Pasos:

- Dividir el número decimal entre 2 y anotar el residuo.
- Dividir el cociente obtenido entre 2 y anotar el residuo.
- Repetir el proceso hasta que el cociente sea 0.
- Los restos obtenidos en orden inverso forman el número binario equivalente.

Ejemplo 2: Conversión de binario a decimal Número binario: 10110 Número decimal: 22

Pasos:

- Multiplicar cada dígito binario por 2 elevado a la potencia de su posición (de derecha a izquierda, empezando desde 0).
- Sumar los resultados.

Ejemplo 3: Conversión entre decimal, octal y hexadecimal Número decimal: 342 Número octal: 526 Número hexadecimal: 156

Pasos:

- Para convertir de decimal a octal o hexadecimal, dividir el número decimal entre 8 u 16, respectivamente, y anotar los restos hasta que el cociente sea 0.
- Para convertir de octal o hexadecimal a decimal, multiplicar cada dígito por la base (8 u 16) elevada a la potencia de su posición.

2.3.2 Representación de números enteros y reales en computadoras

En esta sección, se discuten ejemplos de representación de números enteros y reales en computadoras usando el sistema de punto flotante IEEE 754.

Ejemplo 1: Representación de un número entero en complemento a 2 Número decimal: -15 Número binario en complemento a 2: 11110001

Ejemplo 2: Representación de un número real en punto flotante IEEE 754 Número decimal: 0.15625 Representación en punto flotante de precisión simple: 00111110101000000000000000000000

2.3.3 Problemas al representar números decimales o reales en números binarios

Existen problemas al representar números decimales o reales en números binarios en las computadoras. El problema principal surge debido a las limitaciones en la precisión y el rango de los números de punto flotante que se utilizan para representar los números reales en computadoras.

Las computadoras representan números decimales o reales utilizando el formato de punto flotante, que está definido por el estándar IEEE 754. Este formato divide un número en tres componentes: signo, exponente y mantisa. La mantisa contiene los dígitos significativos del número, mientras que el exponente determina la posición del punto decimal.

Sin embargo, hay ciertos números decimales que no se pueden representar exactamente en binario debido a la naturaleza finita de la representación de punto flotante. Por ejemplo, el número decimal 0.1 no se puede representar exactamente en binario. En su lugar, se representará como una aproximación que puede tener una diferencia muy pequeña con respecto al valor real. Esta diferencia, llamada error de redondeo, puede acumularse y causar problemas en cálculos que involucren muchas operaciones de punto flotante.

Además, la representación de punto flotante tiene un rango y precisión limitados. Esto significa que números muy grandes o muy pequeños pueden causar problemas de desbordamiento (overflow) o subdesbordamiento (underflow). También puede haber pérdida de precisión cuando se realizan operaciones con números que tienen magnitudes muy diferentes.

Para abordar estos problemas, los programadores y científicos de la computación deben ser conscientes de las limitaciones y posibles errores al trabajar con números decimales o reales en computadoras. También es posible utilizar bibliotecas matemáticas especializadas, como la biblioteca decimal en Python, que proporcionan una mayor precisión y control sobre la representación de números decimales en computadoras. Sin embargo,

estas soluciones suelen ser más lentas que las operaciones de punto flotante nativas y pueden no ser adecuadas para todas las aplicaciones.

Veamos un ejemplo en Python que demuestra el error de redondeo al tratar de representar el número decimal 0.1 en binario utilizando números de punto flotante:

```
a = 0.1
b = 0.2
c = 0.3

suma = a + b

print("Resultado de la suma de a y b:", suma)
print("Resultado de la variable c:", c)

# Comparamos si la suma de a y b es igual a c
if suma == c:
    print("La suma de a y b es igual a c.")
else:
    print("La suma de a y b NO es igual a c.")
```

```
Resultado de la suma de a y b: 0.30000000000000004
Resultado de la variable c: 0.3
La suma de a y b NO es igual a c.
```

Como puedes ver, la suma de a y b no es exactamente igual a c, aunque matemáticamente debería serlo. Esto se debe al error de redondeo en la representación binaria de los números decimales 0.1 y 0.2. El resultado de la suma es ligeramente mayor que 0.3, lo que provoca que la comparación falle.

Para solucionar este problema, puedes utilizar la función `isclose()` del módulo math, que compara dos números de punto flotante con una tolerancia especificada:

```
import math

if math.isclose(suma, c):
    print("La suma de a y b es aproximadamente igual a c.")
else:
    print("La suma de a y b NO es aproximadamente igual a c.")
```

```
La suma de a y b es aproximadamente igual a c.
```

2.3.4 Codificación de caracteres e imágenes

En esta sección, se explican ejemplos de codificación de caracteres utilizando ASCII y UTF-8, así como la codificación de imágenes en formatos como BMP y JPEG.

Ejemplo 1: Codificación ASCII Carácter: 'A' Código ASCII: 65 (en decimal), 41 (en hexadecimal)

Ejemplo 2: Codificación UTF-8 Carácter: 'ñ' Código UTF-8: 0xC3 0xB1 (en hexadecimal)

Ejemplo 3: Codificación de imágenes BMP Una imagen BMP de 2x2 píxeles en blanco y negro se puede representar en binario como: 00000000 00000000 11111111 11111111

2.3.5 Ejemplos de operaciones aritméticas y lógicas en binario

En esta sección, se presentan ejemplos de operaciones aritméticas y lógicas realizadas en números binarios.

Ejemplo 1: Suma binaria 1101 (13 en decimal) +1011 (11 en decimal) 11000 (24 en decimal)

Ejemplo 2: Resta binaria 1101 (13 en decimal) -1011 (11 en decimal) 0010 (2 en decimal)

Ejemplo 3: Multiplicación binaria 1101 (13 en decimal)

1011 (11 en decimal) 1101 1101 1101 10011111 (143 en decimal)

Ejemplo 4: División binaria 1101 (13 en decimal) / 1011 (11 en decimal) Cociente: 0001 (1 en decimal) Residuo: 0010 (2 en decimal)

Ejemplo 5: Operaciones lógicas en binario A: 1101 B: 1011

AND: 1101 1011 1001

OR: 1101 1011 1111

XOR: 1101 1011 0110

NOT (complemento a 1 de A): 0010

Estos ejemplos ilustran cómo se realizan operaciones básicas en sistemas numéricos y cómo se representan y manipulan datos en computadoras modernas. Estos conceptos son fundamentales para comprender cómo funcionan las computadoras y cómo se ejecutan programas y algoritmos en ellas.

2.4 Lenguajes de máquina y ensamblador

2.4.1 Lenguaje de máquina: concepto y características

El lenguaje de máquina es el lenguaje de bajo nivel más básico que una computadora es capaz de entender y ejecutar directamente. Está compuesto por una secuencia de instrucciones en forma de códigos binarios (ceros y unos) que representan operaciones específicas que el procesador de la computadora puede realizar. A continuación, se detallan las características del lenguaje de máquina y se proporcionan ejemplos concretos.

2.4.1.1 Características del lenguaje de máquina:

- Específico del hardware: Cada tipo de procesador tiene su propio conjunto de instrucciones y, por lo tanto, su propio lenguaje de máquina. Esto significa que un programa escrito en lenguaje de máquina para un tipo de procesador no funcionará en otro tipo de procesador sin modificaciones.
- Difícil de leer y escribir para los humanos: El lenguaje de máquina es una secuencia de códigos binarios, lo que lo hace difícil de leer, escribir y depurar para los programadores humanos en comparación con los lenguajes de alto nivel.
- Rápido y eficiente: Dado que el lenguaje de máquina es el lenguaje nativo del procesador, su ejecución es muy rápida y eficiente. No hay necesidad de compiladores o intérpretes para convertir el código en instrucciones ejecutables por la máquina.

2.4.1.2 Ejemplos concretos de lenguaje de máquina:

Supongamos que queremos realizar una operación simple, como sumar dos números almacenados en los registros R1 y R2 y guardar el resultado en el registro R3. A continuación, se muestra un ejemplo de cómo podría verse esta operación en lenguaje de máquina:

```
Instrucción en ensamblador: ADD R3, R1, R2
Representación en lenguaje de máquina (hipotética): 110010010011
```

Cabe mencionar que la representación en lenguaje de máquina varía según la arquitectura del procesador. En el ejemplo anterior, se muestra una representación hipotética en formato binario.

Otro ejemplo sería cargar un valor en un registro y luego moverlo a otro registro:

```
Instrucción en ensamblador: LOAD R1, 42
Representación en lenguaje de máquina (hipotética): 101000010010101010

Instrucción en ensamblador: MOVE R2, R1
Representación en lenguaje de máquina (hipotética): 110100010001
```

En este caso, primero cargamos el valor 42 (en binario, 101010) en el registro R1 y luego movemos el contenido de R1 al registro R2. Nuevamente, las representaciones de lenguaje de máquina son hipotéticas y variarán según la arquitectura del procesador.

Estos ejemplos ilustran cómo el lenguaje de máquina es extremadamente básico y difícil de leer y escribir para los humanos. A pesar de esto, es la base sobre la cual se construyen todos los programas y aplicaciones que utilizamos a diario.

2.4.2 Lenguaje ensamblador: concepto y características

El lenguaje ensamblador es un lenguaje de programación de bajo nivel que utiliza mnemotécnicos (abreviaturas) para representar instrucciones y datos en lugar de códigos binarios. Un programa en lenguaje ensamblador es más fácil de leer y escribir para los humanos que el lenguaje de máquina, pero aún así está muy cerca del hardware y requiere un conocimiento profundo de la arquitectura del procesador. A continuación, se detallan las características del lenguaje ensamblador y se proporcionan ejemplos concretos.

Características del lenguaje ensamblador:

- Mnemotécnicos: El lenguaje ensamblador utiliza mnemotécnicos para representar instrucciones y registros, lo que facilita la lectura y escritura por parte de los humanos en comparación con los códigos binarios del lenguaje de máquina.

- Específico del hardware: Al igual que el lenguaje de máquina, el lenguaje ensamblador es específico para cada tipo de procesador y su conjunto de instrucciones. Un programa en lenguaje ensamblador para un procesador no funcionará en otro tipo de procesador sin modificaciones.
- Necesita un ensamblador: Los programas escritos en lenguaje ensamblador deben ser convertidos en lenguaje de máquina utilizando un programa llamado ensamblador. El ensamblador convierte las instrucciones mnemotécnicas en sus equivalentes binarios.

A continuación, se presentan ejemplos de programas simples en lenguaje ensamblador. Tenga en cuenta que estos ejemplos son genéricos y pueden variar según la arquitectura del procesador y el conjunto de instrucciones.

Ejemplo 1: Programa para sumar dos números

```
; Programa en ensamblador para sumar dos números

; Datos
num1 DW 5      ; Define el primer número (5)
num2 DW 7      ; Define el segundo número (7)
resultado DW 0 ; Define una variable para almacenar el resultado

; Código
mov ax, num1   ; Carga el primer número en el registro AX
mov bx, num2   ; Carga el segundo número en el registro BX
add ax, bx     ; Suma los números en AX y BX y guarda el resultado en AX
mov resultado, ax ; Guarda el resultado en la variable "resultado"

; Fin del programa
```

Ejemplo 2: Programa para calcular el factorial de un número

```
; Programa en ensamblador para calcular el factorial de un número

; Datos
num DW 5       ; Define el número (5)
factorial DW 1 ; Define una variable para almacenar el resultado

; Código
mov cx, num    ; Carga el número en el registro CX

loop_start:    ; Etiqueta del inicio del bucle
mul cx         ; Multiplica el valor en AX por CX y guarda el resultado en AX
dec cx         ; Decrementa el valor en CX en 1
jnz loop_start ; Salta al inicio del bucle si CX no es 0

mov factorial, ax ; Guarda el resultado en la variable "factorial"

; Fin del programa
```

2.4.3 Relación entre lenguajes de máquina y ensamblador

El lenguaje ensamblador y el lenguaje máquina están estrechamente relacionados, ya que ambos son lenguajes de programación de bajo nivel utilizados para interactuar directamente con el hardware del procesador. El lenguaje ensamblador es una representación más legible del lenguaje máquina, utilizando mnemotécnicos en lugar de códigos binarios para representar instrucciones y datos. A continuación, se presentan ejemplos que ilustran la relación entre el lenguaje ensamblador y el lenguaje máquina.

Ejemplo 1: Sumar dos números

Lenguaje ensamblador:

```
mov ax, 5
mov bx, 7
```

```
add ax, bx
```

Lenguaje de máquina (representación hexadecimal):

```
B8 05 00
BB 07 00
01 D8
```

En este ejemplo, el programa en lenguaje ensamblador utiliza mnemotécnicos para cargar los valores 5 y 7 en los registros AX y BX, y luego suma estos valores. El programa equivalente en lenguaje de máquina utiliza códigos binarios (representados en hexadecimal) para realizar la misma tarea. El ensamblador es responsable de convertir el programa en lenguaje ensamblador a lenguaje de máquina.

2.4.4 Ejemplo end-to-end

Aunque aun no hemos hablado de los lenguajes de programación de alto nivel, vamos a ver un ejemplo de inicio a fin de como sería todo el proceso de creación de un algoritmo para resolver un problema en un lenguaje de alto nivel como C, cómo este código debe ser compilado a ensamblador y después a lenguaje máquina para finalmente ser ejecutado por el ordenador.

A continuación, se presenta el proceso de creación, compilación y ejecución de un programa simple en C, incluyendo la conversión a código ensamblador y código máquina para una arquitectura i386 de 64 bits.

- Crear el archivo fuente en C:

Escribamos un programa simple en C que sume dos números y muestre el resultado en la consola. Guarde este código en un archivo llamado `suma.c`.

```c
#include <stdio.h>

int main() {
    int a = 5;
    int b = 7;
    int suma = a + b;
    printf("La suma de %d y %d es %d\n", a, b, suma);
    return 0;
}
```

- Generar el código ensamblador: Antes de compilar el programa en C a código máquina, podemos convertirlo en código ensamblador utilizando el compilador GCC. Esto nos permite ver las instrucciones en lenguaje ensamblador que corresponden a nuestro programa en C.

Ejecute el siguiente comando en la terminal para generar el archivo de código ensamblador suma.s:

```
gcc -S -masm=intel -m64 suma.c -o suma.s
```

El archivo `suma.s` generado puede verse así:

```
    .file   "suma.c"
    .intel_syntax noprefix
    .section    .rodata
.LC0:
    .string "La suma de %d y %d es %d\n"
    .text
    .globl  main
    .type   main, @function
main:
    push    rbp
```

```
    mov     rbp, rsp
    sub     rsp, 16
    mov     DWORD PTR [rbp-4], 5
    mov     DWORD PTR [rbp-8], 7
    mov     edx, DWORD PTR [rbp-4]
    mov     eax, DWORD PTR [rbp-8]
    add     eax, edx
    mov     DWORD PTR [rbp-12], eax
    mov     eax, DWORD PTR [rbp-12]
    mov     edx, DWORD PTR [rbp-8]
    mov     ecx, DWORD PTR [rbp-4]
    mov     esi, eax
    mov     edi, OFFSET FLAT:.LC0
    mov     eax, 0
    call    printf
    mov     eax, 0
    leave
    ret
```

- Compilar el código en C a código máquina:

Ahora que hemos generado el código ensamblador, podemos compilar nuestro programa en C a código máquina. Utilice el siguiente comando para compilar el programa en un archivo ejecutable llamado suma:

```
gcc -m64 suma.c -o suma
```

La salida puede verse así:

```
0000000000001139 <main>:
    1139:   55                      push   rbp
    113a:   48 89 e5                mov    rbp,rsp
    113d:   48 83 ec 10             sub    rsp,0x10
```

```
    1141:   c7 45 fc 05 00 00 00    mov    DWORD PTR [rbp-0x4],0x5
    1148:   c7 45 f8 07 00 00 00    mov    DWORD PTR [rbp-0x8],0x7
    114f:   8b 55 fc                mov    edx,DWORD PTR [rbp-0x4]
    1152:   8b 45 f8                mov    eax,DWORD PTR [rbp-0x8]
    1155:   01 d0                   add    eax,edx
    1157:   89 45 f4                mov    DWORD PTR [rbp-0xc],eax
    115a:   8b 45 f4                mov    eax,DWORD PTR [rbp-0xc]
    115d:   8b 55 f8                mov    edx,DWORD PTR [rbp-0x8]
    1160:   8b 4d fc                mov    ecx,DWORD PTR [rbp-0x4]
    1163:   89 c6                   mov    esi,eax
    1165:   48 8d 3d 94 0e 00 00    lea    rdi,[rip+0xe94]        # 2000 <_IO_stdin_used+0x2
    116c:   b8 00 00 00 00          mov    eax,0x0
    1171:   e8 aa fe ff ff          call   1020 <printf@plt>
    1176:   b8 00 00 00 00          mov    eax,0x0
    117b:   c9                      leave
    117c:   c3                      ret
```

Esta salida muestra el código máquina en notación hexadecimal (por ejemplo, `55, 48 89 e5, 48 83 ec 10`, etc.) a la izquierda y las correspondientes instrucciones en lenguaje ensamblador a la derecha. Estos códigos hexadecimales representan las instrucciones binarias que el procesador x86-64 ejecuta.

Tenga en cuenta que la salida específica puede variar dependiendo de la versión del compilador y la plataforma en la que se ejecute.

- Ejecutar el programa:

Finalmente, ejecute el programa compilado utilizando el siguiente comando:

```
./suma
```

3 Lenguajes de alto nivel y Python

3.1 Características de los lenguajes de alto nivel

Los lenguajes de programación se pueden clasificar en diferentes categorías según su nivel de abstracción y complejidad. Los lenguajes de alto nivel son aquellos que están diseñados para ser más fáciles de usar y entender para los programadores, en comparación con los lenguajes de bajo nivel que están más cerca del lenguaje de la máquina y son más difíciles de entender y utilizar. En el contexto de la programación, las características de los lenguajes de alto nivel se pueden resumir de la siguiente manera:

- Abstracción: Los lenguajes de alto nivel proporcionan un alto nivel de abstracción que permite a los programadores enfocarse en la lógica y la funcionalidad del programa, en lugar de preocuparse por detalles de bajo nivel como la gestión de la memoria y la asignación de registros de la CPU.

- Legibilidad: Los lenguajes de alto nivel están diseñados para ser fácilmente legibles por los humanos, con una sintaxis clara y bien definida que ayuda a los programadores a entender y escribir código de manera más eficiente.

- Portabilidad: Los lenguajes de alto nivel se pueden escribir una vez y ejecutar en diferentes plataformas y sistemas operativos, lo que significa que el mismo programa puede ejecutarse en diferentes dispositivos sin necesidad de reescribirlo.

- Bibliotecas y herramientas: Los lenguajes de alto nivel suelen contar con amplias bibliotecas y herramientas, que permiten a los programadores utilizar funciones predefinidas y herramientas de programación para simplificar la tarea de codificar y acelerar el proceso de desarrollo.

- Productividad: Los lenguajes de alto nivel están diseñados para ser más productivos y eficientes para los programadores, ya que permiten crear aplicaciones más rápidamente en comparación con los lenguajes de bajo nivel.

3.2 Ejemplos de Lenguajes de Alto Nivel

Algunos ejemplos de lenguajes de alto nivel populares:

- Python: es un lenguaje de programación interpretado, multiplataforma, de alto nivel y con tipado dinámico. Es muy popular debido a su sintaxis clara y concisa, lo que lo hace fácil de leer y escribir. Además, cuenta con una gran cantidad de bibliotecas y módulos que permiten desarrollar aplicaciones de manera eficiente.

- Java: es un lenguaje de programación orientado a objetos, portable y de alto nivel. Es muy popular en el desarrollo de aplicaciones empresariales y web. Java se ejecuta en una máquina virtual Java (JVM), lo que lo hace altamente portable y compatible con diferentes sistemas operativos.

- C#: es un lenguaje de programación orientado a objetos y de alto nivel desarrollado por Microsoft. Se utiliza principalmente para desarrollar aplicaciones de escritorio y web en el entorno de desarrollo .NET. C# es fácil de leer y escribir, lo que lo convierte en una buena opción para desarrolladores que recién comienzan.

- JavaScript: es un lenguaje de programación interpretado, multiplataforma y de alto nivel utilizado principalmente para el desarrollo de aplicaciones web. JavaScript se ejecuta en el navegador web del usuario y se utiliza para agregar interactividad y dinamismo a las páginas web.
- Ruby: es un lenguaje de programación interpretado, orientado a objetos y de alto nivel. Es muy popular en el desarrollo de aplicaciones web y móviles debido a su sintaxis simple y concisa, lo que lo hace fácil de leer y escribir.
- PHP: es un lenguaje de programación interpretado, multiplataforma y de alto nivel utilizado principalmente para el desarrollo de aplicaciones web. Es especialmente adecuado para la programación web del lado del servidor y se utiliza en la mayoría de los sistemas de gestión de contenidos (CMS) como WordPress y Drupal.

3.3 Lenguajes Compilados vs Interpretados

Los lenguajes de programación se pueden clasificar en dos categorías principales: lenguajes compilados y lenguajes interpretados. La principal diferencia entre ellos es cómo se traduce el código fuente del programa en código de máquina que la computadora puede entender y ejecutar.

3.3.1 Lenguajes compilados

Los lenguajes compilados son aquellos en los que el código fuente se traduce en código de máquina antes de su ejecución. El proceso de traducción se lleva a cabo por un programa llamado compilador, que toma el código fuente y lo convierte en un archivo ejecutable que la computadora puede entender y ejecutar directamente. Algunos ejemplos de lenguajes compilados son C, C++, Fortran y Ada.

3.3.1.1 Ventajas de los lenguajes compilados:

- Mayor velocidad de ejecución: El código compilado se traduce directamente en código de máquina que la computadora puede ejecutar rápidamente, lo que resulta en un mayor rendimiento en general.
- Detección de errores: El compilador detecta errores de sintaxis y de tipo antes de la ejecución, lo que puede ahorrar tiempo y aumentar la calidad del código.

3.3.1.2 Desventajas de los lenguajes compilados:

- Tiempo de compilación: La compilación puede llevar mucho tiempo, especialmente para programas grandes, lo que puede retrasar el proceso de desarrollo.
- Falta de portabilidad: Los programas compilados suelen estar diseñados para una plataforma y arquitectura de hardware específicas, lo que significa que no se pueden ejecutar en diferentes plataformas sin ser recompilados.

3.3.2 Lenguajes interpretados

Los lenguajes interpretados son aquellos en los que el código fuente se ejecuta directamente por un programa llamado intérprete. El intérprete lee el código fuente línea por línea y lo traduce en código de máquina en tiempo real. Algunos ejemplos de lenguajes interpretados son Python, Ruby, PHP y JavaScript.

3.3.2.1 Ventajas de los lenguajes interpretados:

- Portabilidad: Los programas interpretados se pueden ejecutar en diferentes plataformas sin necesidad de ser recompilados, lo que los hace altamente portátiles.

- Fácil depuración: Los errores son más fáciles de detectar y depurar, ya que el intérprete detiene la ejecución en la línea que contiene el error.

3.3.2.2 Desventajas de los lenguajes interpretados:

- Menor velocidad de ejecución: El proceso de interpretación puede ser más lento que la ejecución de código compilado, lo que puede afectar el rendimiento del programa.
- Detección de errores: Los errores de sintaxis y de tipo pueden no ser detectados hasta que el código se esté ejecutando, lo que puede aumentar el tiempo de depuración.

3.4 Introducción a Python: sintaxis básica, variables y tipos de datos

Python es un lenguaje de programación de alto nivel y de propósito general, que se ha vuelto muy popular en la última década debido a su facilidad de uso y a la amplia gama de aplicaciones que se pueden desarrollar con él. A continuación, se presenta una introducción a la sintaxis básica de Python, así como a las variables y tipos de datos que se utilizan en este lenguaje.

3.4.1 Sintaxis básica de Python

Python utiliza una sintaxis simple y legible que lo hace fácil de aprender para los principiantes. Algunas características de la sintaxis básica de Python son:

- Indentación: Python utiliza la indentación en lugar de los corchetes para indicar el alcance del código, lo que significa que la legibilidad del código es muy importante en Python. Por ejemplo, un bloque de código puede ser:

```
if x > 0:
    print("x es mayor que cero")
else:
    print("x es menor o igual que cero")
```

- Comentarios: Los comentarios en Python se escriben con el símbolo de almohadilla (#) y se utilizan para explicar el código y hacerlo más legible para otros programadores.

```
# Esto es un comentario
```

- Variables: En Python, las variables se crean simplemente asignándoles un valor. Por ejemplo:

```
x = 10
nombre = "Juan"
```

3.4.2 Variables y tipos de datos

Python tiene varios tipos de datos predefinidos que se pueden utilizar para almacenar diferentes tipos de información. Algunos de los tipos de datos más comunes en Python son:

- Números: Python admite diferentes tipos de números, como enteros, flotantes y complejos.

```
entero = 10
flotante = 3.14
complejo = 2 + 3j
```

- Cadenas: Las cadenas se utilizan para almacenar texto y se escriben entre comillas simples o dobles.

```
cadena = "Hola Mundo!"
```

- Listas: Las listas se utilizan para almacenar una colección ordenada de elementos y se pueden modificar.

```
lista = [1, 2, 3, 4]
```

- Tuplas: Las tuplas son similares a las listas, pero son inmutables, es decir, no se pueden modificar después de su creación.

```
tupla = {1, 2, 3, 4}
```

- Diccionarios: Los diccionarios se utilizan para almacenar pares clave-valor y se pueden modificar.

```
diccionario = {"nombre": "Juan", "edad": 30}
```

3.4.3 Operadores

Python tiene una amplia variedad de operadores que se pueden utilizar para realizar diferentes tipos de operaciones. Algunos de los operadores más comunes en Python son:

- Aritméticos: `+, -, *, /, //, %, **`
- Comparación: `==, !=, <, >, <=, >=`
- Lógicos: `and, or, not`
- De asignación: `=, +=, -=, *=, /=`

En resumen, Python es un lenguaje de programación de alto nivel y de propósito general que se ha vuelto muy popular debido a su sintaxis simple y legible, así como a su amplia gama de tipos de datos y operadores. Python es utilizado en una amplia variedad de aplicaciones, como el desarrollo web, la ciencia de datos y la inteligencia artificial.

4 Conclusion

In summary, this book has no content whatsoever.