PROGRAMACIÓN DEL ROBOT LEGO MINDSTORMS EV3 USANDO LA LIBRERÍA PYBRICKS DE PYTHON

FRANCISCO ALEJANDRO MEDINA A.

PROGRAMACION DEL ROBOT LEGO MINDSTORMS EV3 USANDO LA LIBRERÍA PYBRICKS DE PYTHON

FRANCISCO ALEJANDRO MEDINA AGUIRRE

UNIVERSIDAD TECNOLOGICA DE PERREIRA

FACULTAD DE INGENIERIAS

COLOMBIA, 2023

PREFACIO

La tecnología evoluciona a un ritmo acelerado, y la programación es un arte que nos permite dotar a las máquinas de comportamientos autónomos e inteligentes. Aprender a programar robots abre un sinfín de puertas al desarrollo profesional y la innovación.

Este documento busca introducir de forma amigable el apasionante mundo de la programación de robots usando Python, un lenguaje de programación reconocido por su sintaxis clara y sencilla. El lector aprenderá conceptos de programación mientras desarrolla conductas autónomas en los robots Lego Mindstorms.

Mediante explicaciones paso a paso, ejemplos prácticos y continua retroalimentación, el lector aprenderá a dominar las bases de Python y la programación de robots. Los temas cubiertos incluyen Fundamentos de Python, y la programación de robots Ev3. Al finalizar, el lector podrá dar vida a sus propias creaciones robóticas mediante Python.

Este documento está dirigido a estudiantes de ingeniería, entusiastas de la tecnología y cualquier persona que quiera aprender a programar robots. La metodología de aprender construyendo y practicando busca motivar el interés en estas fascinantes áreas del saber.

Tabla de contenido

TABLA DE FIGURAS.

INTRODUCCION

La programación es el proceso de crear instrucciones que pueden ser ejecutadas por una computadora u otro dispositivo para realizar tareas específicas. En la actualidad, la programación se ha vuelto omnipresente en nuestra vida cotidiana, desde aplicaciones móviles y sitios web, hasta electrodomésticos e incluso juguetes. Aprender a programar abre un mundo de posibilidades creativas y nos permite dar rienda suelta a nuestra imaginación.

Existen muchos lenguajes de programación para elegir. En este documento, nos enfocaremos en Python, que se ha convertido en uno de los más populares y de mayor crecimiento en los últimos años. Python es un lenguaje interpretado, multiplataforma y multiparadigma. Se distingue por tener una sintaxis clara y legible, además de ser fácil de aprender para principiantes. Estas características lo convierten en una excelente opción para introducirnos en el apasionante campo de la programación.

Junto con Python, utilizaremos el popular robot Lego Mindstorms EV3 para aprender programación de una forma práctica y entretenida. Lego EV3 nos permite construir distintos robots y dotarlos de comportamientos autónomos programando sus movimientos, sensores y acciones. De esta manera, podremos ver cómo cobra vida nuestro código en fascinantes proyectos robóticos.

A lo largo de este documento, obtendrás los fundamentos esenciales de Python y la programación en general, incluyendo conceptos como variables, operadores, funciones, clases, entre otros. Se utilizarán sencillos ejemplos para ilustrar cada elemento y así facilitar el aprendizaje. Luego, aplicaremos estos conocimientos para programar un Lego EV3 paso a paso, desde hacer simples movimientos hasta crear robots capaces de navegar laberintos y seguir líneas en el suelo.

Al final del documento habrás adquirido las habilidades para programar robots y otras aplicaciones en Python. La metodología utilizada te permitirá aprender, practicar y aplicar

conceptos de una manera activa, constructiva y progresiva. Asimismo, el enfoque práctico sobre proyectos con Lego EV3 integrará la programación con una actividad creativa y lúdica.

Los robots Lego Mindstorms son accesibles y fáciles de usar incluso para principiantes o niños, por lo que constituyen un camino ideal para iniciarse en la robótica. La versatilidad de Python y EV3 nos posibilita crear desde sencillos programas hasta complejos sistemas de Inteligencia Artificial.

CAPÍTULO 1. PROGRAMACION DEL ROBOT LEGO MIDSTORN EV3

1.1. QUE ES LA PROGRAMACIÓN.

La programación, o codificación, es el proceso de crear un conjunto de instrucciones estructuradas y organizadas, conocidas como algoritmos, que permiten indicarle a una computadora qué tareas debe realizar. Los algoritmos computacionales se escriben en lenguajes de programación que tanto las personas como las máquinas pueden entender.

Un algoritmo es una serie ordenada y finita de pasos u operaciones que permite hacer un cálculo, resolver un problema o ejecutar alguna tarea en particular. Sirve para describir el proceso necesario para obtener un resultado deseado. Algunos ejemplos de algoritmos son las recetas de cocina, los mapas para llegar a un destino y los manuales de ensamblaje.

En informática, un algoritmo debe ser preciso, definido, claro, lógico, secuencial y tener un principio y fin determinados. Los algoritmos computacionales son implementados en lenguajes de programación y ejecutados por un dispositivo tecnológico. Permiten automatizar tareas, procesar datos, controlar el funcionamiento de máquinas y crear software.

Los lenguajes de programación son vocabularios y sintaxis o gramáticas que permiten escribir algoritmos computacionales en términos y formatos entendibles tanto para humanos como para las máquinas. Cada lenguaje tiene palabras clave, estructuras y reglas propias para crear programas que solucionen problemas específicos.

Existe una amplia diversidad de lenguajes de programación como Python, JavaScript, C++, Java, Swift, PHP, Ruby, etc. Cada uno tiene propósitos, fortalezas y debilidades particulares. Algunos lenguajes son multiparadigma, es decir, soportan distintos estilos y enfoques de

programación como programación orientada a objetos, funcional, imperativa, declarativa, entre otros.

1.2. ¿QUÉ ES PYTHON?

Es uno de los lenguajes de programación de mayor crecimiento y popularidad en los últimos años. Fue creado a finales de la década de 1980 por el programador holandés Guido Van Rossum quien buscaba crear un lenguaje que fuera fácil de usar y aprender, incluso para no programadores.

Figura 1. Logo de lenguaje de programación Python

Python es multiparadigma, soporta programación orientada a objetos, imperativa y funcional. Se trata de un lenguaje interpretado, es decir que no necesita compilación y se ejecuta línea por línea. Esto permite ver resultados rápidamente sin procesos intermedios. También es multiplataforma, puede utilizarse en Windows, Mac, Linux y otros sistemas operativos.

La sintaxis de Python se caracteriza por ser muy legible y limpia, similar al inglés natural. Utiliza sangrías en lugar de corchetes para delimitar bloques de código, lo que le brinda mucho orden visual. También favorece la legibilidad el uso de nombres de variables explicativos. Asimismo, al ser un lenguaje de tipado dinámico, no es necesario declarar variables ni tipos de datos.

Python cuenta con una gran biblioteca estándar que incorpora muchas funciones útiles para distintas tareas de programación, evitando tener que reinventar la rueda. Además, tiene una amplia comunidad que desarrolla y mantiene bibliotecas o frameworks adicionales de código abierto para propósitos específicos, como ciencia de datos, web, robótica, inteligencia artificial, entre muchos otros.

La sintaxis sencilla de Python y su curva de aprendizaje suave lo convierten en un excelente primer lenguaje para aprender a programar. Permite enfocarse más en la lógica y estructuras de programación antes que en los detalles del lenguaje. Además, Python es utilizado por principiantes como expertos dado que es un lenguaje muy versátil y potente.

Python es un lenguaje interpretado de propósito general, cuenta con tipado dinámico y una sintaxis clara, ordenada, de fácil lectura muy cercana al inglés. Posee además una gran biblioteca estándar y muchos frameworks de código abierto. Por todas estas características se ha convertido en uno de los lenguajes de programación más populares para iniciar en el aprendizaje como para desarrollar todo tipo de aplicaciones.

Desde scripts simples hasta complejos programas, Python es una excelente elección como primer lenguaje dado su sintaxis sencilla y enfoque práctico, que permiten concentrarse más en la lógica de programación. Cuenta con el poder y escalabilidad para crecer profesionalmente como programador. Es un lenguaje confiable utilizado por grandes compañías y aplicaciones web conocidas como Google, Youtube, Instagram, Spotify, entre muchas otras.

Python sigue mejorando y evolucionando constantemente. La versión estable más reciente es Python 3, que introdujo cambios y mejoras con respecto a Python 2 que ya no se mantiene. La comunidad Python es muy activa en agregar características y actualizaciones por medio de PEPs (Python Enhancement Proposals). El lenguaje se adapta así a las tendencias y necesidades cambiantes en el desarrollo de software.

Python se ha consolidado como uno de los lenguajes imperdibles y esenciales para todo desarrollador. Su creciente adopción se debe a características clave como legibilidad, versatilidad, facilidad de uso, amplia biblioteca, comunidad y ecoistema activo. Python tiene las cualidades ideales para introducirse en la programación y es un gran compañero de viaje para crecer como profesional. Aprender Python abre la puerta a un sinfín de posibilidades en el apasionante mundo de la tecnología.

1.3. ¿QUÉ ES LEGO MINDSTORMS EV3?

El robot EV3 es un conjunto de robótica de la línea Mindstorms de Lego, fue lanzado en 2013. EV3 proviene de "Evolution 3" (Evolución 3) por ser la tercera generación de estos

kits de robótica programable de Lego (Cozmo, 2013). Este sistema está orientado a aficionados, estudiantes e incluso profesionales que desean adentrarse en el mundo de la robótica y programación de forma práctica e intuitiva.

Figura 2. Empaque de robot lego mindstorms EV3

El set básico de EV3 contiene aproximadamente 600 piezas construibles de Lego, incluyendo bloques, ejes, engranajes, ruedas, así como una unidad programable llamada ladrillo inteligente o brick EV3, la cual hace las veces de computadora (Lego, 2013). Este brick cuenta con un procesador ARM9 a 300MHz, memoria RAM de 64 MB, almacenamiento de 16 MB ampliable con tarjeta SD, pantalla LCD monocromática, altavoz y conectividad USB, Bluetooth y Wi-Fi (Gindrat, Lego Mindstorms EV3: building and programming a complex robot. Journal of Computing Sciences in Colleges, 2014).

Además del ladrillo, el kit trae tres servomotores interactivos, dos de tamaño grande y uno mediano, así como varios sensores para obtener información del entorno. Entre ellos el sensor de tacto, color, ultrasonido y giroscopio. Estos componentes de entrada y salida son fundamentales para que el robot pueda ejecutar acciones y reaccionar ante estímulos mediante la programación desarrollada (Lego, 2013).

Uno de los grandes atractivos de Lego EV3 es la posibilidad de ensamblar distintos modelos funcionales como vehículos, humanoides, grúas, animales o lo que la imaginación disponga,

utilizando las diversas piezas de Lego. Luego se programa su comportamiento a través del ladrillo EV3 con movimiento, luces, sonidos, acciones y reacciones de acuerdo con las entradas de sus sensores (Cozmo, 2013).

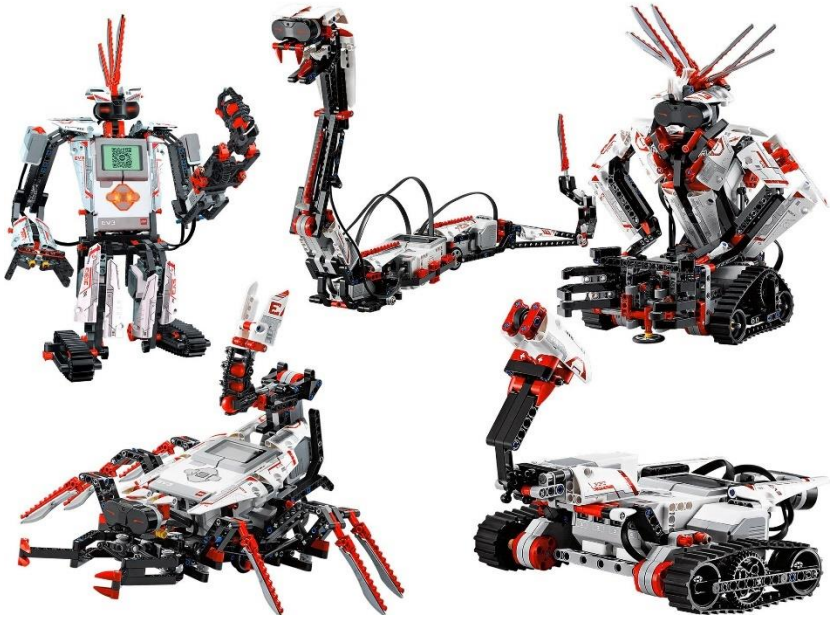

Figura 3. Algunos modelos para armar con lego EV3

Para crear los programas que controlarán los robots EV3 existen varias alternativas de software. La opción oficial es Lego Mindstorms EV3 Home Edition, un entorno de desarrollo visual que utiliza bloques tipo puzzle para definir el comportamiento deseado (Lego, 2013). También hay opciones avanzadas como EV3Basic o RobotC que permiten programación textual en lenguajes como BASIC, C, C++, Python, entre otros.

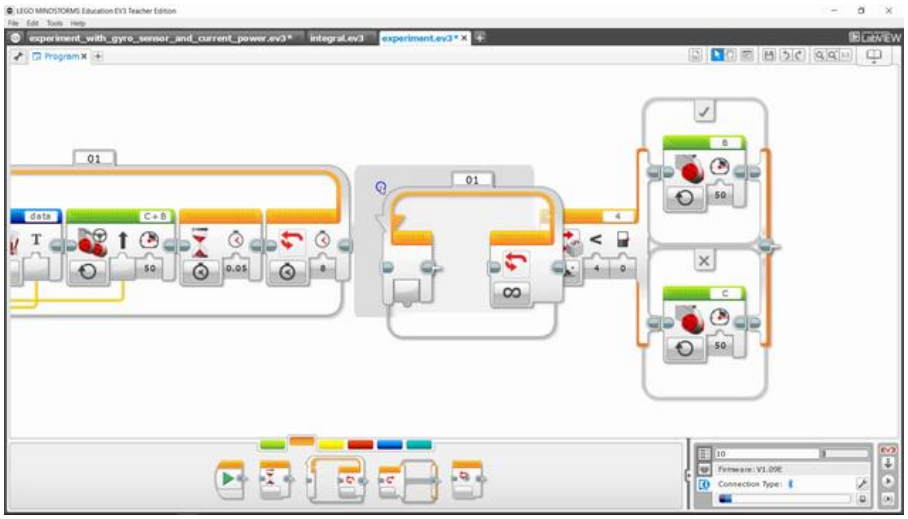

Figura 4. Software Lego Mindstorms EV3 Home Edition

Una de las principales ventajas de Mindstorms EV3 es su gran versatilidad para construir desde modelos básicos hasta robots muy complejos controlados por sofisticados programas. Su sistema modular con piezas ensamblables lo diferencia de otros kits que traen un solo modelo predeterminado (Berry, 2012; Bingi, 2020). Además, al ser producido por una empresa reconocida como Lego, cuenta con alta calidad en los materiales y buen soporte.

EV3 permite aprender sobre construcción, mecánica, electrónica, motores, sensores, estructuras Lego, pero sobre todo introduce conceptos esenciales de programación y robótica (Cozmo, 2013). Mediante la creación de distintos proyectos, los usuarios entienden cómo darle comportamientos autónomos a una máquina, algo que de otra forma sería muy abstracto.

En el área educativa, EV3 es utilizado ampliamente para la enseñanza de STEM (science, technology, engineering and math) dado que de forma práctica e intuitiva enseña sobre ciencia, tecnología, ingeniería y matemáticas (Gerasimova, 2020). Incluso universidades como el MIT y Harvard han incorporado robots de la línea Mindstorms en sus cursos de robótica e inteligencia artificial para experimentación. Lego Mindstorms EV3 es una excelente plataforma de iniciación en el apasionante mundo de la robótica y la programación. Permite construir, programar y dar vida a fascinantes creaciones mecánicas de forma sencilla e interactiva. EV3 conjuga lo mejor del divertido sistema Lego con tecnología de punta en robótica modular.

1.4. APLICACIONES DE PYTHON EN ROBÓTICA.

Python se ha convertido en uno de los lenguajes de programación más utilizados y versátiles para desarrollar aplicaciones en robótica dada su facilidad de uso y las poderosas librerías especializadas disponibles (Bingi, 2020).

Una de las principales ventajas de Python es que permite controlar de forma simple componentes robóticos como motores, sensores y cámaras desde el lenguaje de programación (Stern, 2017). Mediante el uso de bibliotecas específicas para robótica, se puede interactuar con hardwares de diferentes fabricantes sin necesidad de lidiar con los complejos detalles de bajo nivel.

Por ejemplo, Robot Operating System (ROS) provee una amplia infraestructura en Python para crear todo tipo de comportamientos y funcionalidades en robots (Quigley, 2009). ROS permite control de movimiento, percepción sensorial, navegación, visión por computadora y manipulación de objetos, entre muchas otras capacidades.

Figura 5. Logo de Robot Operating System

Asimismo, OpenCV es una reconocida biblioteca de Python enfocada en visión artificial y procesamiento de imágenes, indispensable en robótica para que un robot pueda “ver” e interpretar su entorno (Bradski, 2008). Python facilita la integración de OpenCV en proyectos de robótica para agregar funciones avanzadas de visión.

Figura 6. Logo de OpenCV

En cuanto a robótica educativa, Python ha ganado popularidad gracias a los recientes avances en kits de hardware/software como Raspberry Pi, MicroPython y CircuitPython (Richardson, 2020). Estas herramientas permiten programar de forma sencilla robots y dispositivos maker mediante Python sin grandes conocimientos previos.

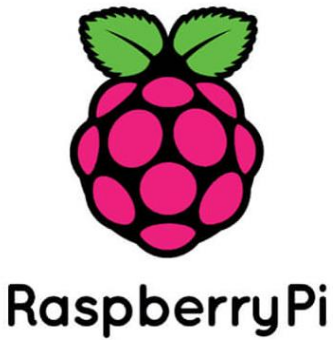

Figura 7. Logo de Rasberry Pi

Un ejemplo ampliamente utilizado en el ámbito educativo es el robot Lego Mindstorms EV3, el cual puede ser programado en Python de varias formas (Gouws, 2013). Por un lado, instalando el firmware EV3Dev que corre sobre Linux y permite ejecutar código MicroPython. O también transmitiendo programas en Python desde un PC hacia el EV3 usando bibliotecas como EV3Python o PyBricks.

Figura 8. Logo Lego mindstorms EV3

En cuanto a robots más avanzados, Python es el lenguaje de programación oficial y recomendado para los populares robots humanoides de SoftBank Robotics, Pepper y NAO (Gouaillier, 2009). Python provee una API simple y bien documentada para controlar de forma amigable todos los comportamientos y características de estos robots.

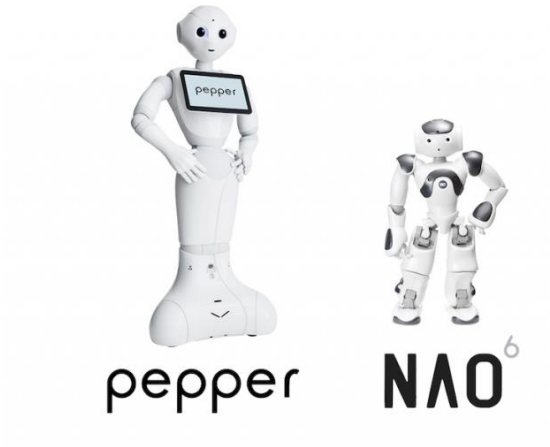

Figura 9. Robots Pepper y NAO de SoftBank Robotics

A nivel industrial, Python es ampliamente adoptado en aplicaciones de automatización y robótica en manufactura gracias a frameworks como Robot Framework que facilitan crear scripts de pruebas automatizadas para robots en este contexto (Niemelä, 2017). El uso de Python permite mayor productividad y calidad en entornos complejos de fabricación.

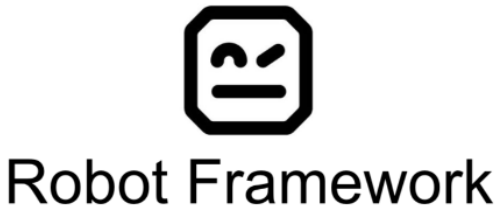

Figura 10. Logo Robot Framework

Python se ha consolidado como un lenguaje ideal para programar todo tipo de robots y proyectos de robótica en distintos niveles. Desde sencillos scripts para pequeños robots educativos hasta complejos sistemas de inteligencia artificial en humanoides. La versatilidad y potencia de Python, sumado a sus numerosas librerías especializadas, brindan las herramientas necesarias para dar vida a fascinantes creaciones robóticas.

CAPÍTULO 2. CONFIGURACIÓN DEL AMBIENTE DE DESARROLLO

Para poder empezar a programar el robot Lego Mindstorms EV3 utilizando el lenguaje Python, es necesario configurar adecuadamente nuestro ambiente de desarrollo, lo cual involucra instalar el software requerido, establecer la conexión entre la PC y el EV3, y agregar las bibliotecas de Python específicas para interactuar con el robot.

2.1. SOFTWARE REQUERIDO.

Lo primero que debemos tener es un brick o ladrillo programable EV3, el cual hace las veces de cerebro del robot. El kit básico de Lego Mindstorms EV3 viene con el brick EV3, motores, sensores y piezas mecánicas (Lego, LEGO® MINDSTORMS®, 2020).

Figura 11. Kit básico del lego EV3

Luego necesitamos un cable USB – mini USB para conectar físicamente el EV3 con la computadora y transferir los programas. Es recomendable un cable USB 2.0 estándar de al menos 1 metro de longitud y de buena calidad para una transmisión estable (Ribeiro, 2020).

El siguiente software imprescindible es un editor de texto o IDE (Integrated Development Environment) para escribir los programas en Python en nuestra computadora, para esto utilizaremos Visual Studio Code, este IDE es gratuito, liviano y con muchas extensiones útiles para Python (Stern, 2017), y para programar el lego Ev3

Figura 12. Logo Visual Studio Code

También debemos descargar e instalar un intérprete de Python en nuestro equipo. Las versiones recomendadas actualmente son Python 3.7 o superior, siendo 3.10 o 3.11 ideales para aprovechar las últimas características del lenguaje (Python Software Foundation, 2022). Podemos utilizar la distribución oficial de Python disponible en www.python.org.

2.2. CONFIGURACIÓN E INSTALACION DEL LEGO MINDSTORMS EV3 CON PYBRICKS

Una excelente opción para programar el EV3, es utilizar el lenguaje Python y la biblioteca PyBricks, la cual permite controlar el robot de manera sencilla e intuitiva. En esta parte del documento, exploraremos cómo instalar y configurar PyBricks para programar el EV3 utilizando Visual Studio Code como entorno de desarrollo integrado.

2.2.1. ¿Qué es PyBricks?

PyBricks es una biblioteca de Python open source creada por Lego en colaboración con expertos de la comunidad Python, enfocada en la facilidad de uso para programar robots de Lego (Lego, EV3 MicroPython, 2019). Provee una API (interfaz de programación de aplicaciones) de alto nivel para interactuar con los ladrillos programables de Lego como los robots EV3, NXT, y boots.

La biblioteca PyBricks presenta las siguientes características clave (PyBricks, 2022):

- Comunicación inalámbrica con los ladrillos Lego mediante Bluetooth o WiFi sin necesidad de cables.
- Abstracción de los detalles de bajo nivel, enfocándose en objetos de alto nivel como motores, sensores y controles remotos.
- Sintaxis limpia y minimalista parecida al inglés para maximizar legibilidad.
- Excelente documentación con amplios ejemplos y tutoriales.
- Posee una licencia de código abierto MIT, para promover su uso y contribución a la biblioteca.
- Compatibilidad con múltiples ladrillos programables de Lego. El mismo código Python puede controlar EV3, NXT, Boost, etc.

PyBricks está diseñada desde cero específicamente para proveer la mejor experiencia de programación de robots Lego en Python, tanto para principiantes como para usuarios avanzados.

2.2.2. Preparación del entorno de desarrollo.

Para poder utilizar PyBricks necesitaremos preparar nuestro entorno de desarrollo, el cual incluye el siguiente software:

- Visual Studio Code: es el editor de código que utilizaremos para escribir los programas Python que controlarán el EV3 (Microsoft., 2020).
- Python 3.7 o superior: el intérprete de Python necesario para ejecutar los programas. Se recomienda la versión 3.10 o 3.11 (Python Software Foundation, 2022).
- Extensión EV3 MicroPython para VS Code, esta extensión agrega soporte para MicroPython y PyBricks en VS Code.
- Biblioteca PyBricks: se instala mediante el comando pip install pybricks.

Una vez tenemos instaladas estas piezas podemos comenzar a escribir código PyBricks en VS Code para controlar el EV3.

2.2.3. Configurando comunicación EV3 y PC.

Existen dos formas principales de conectar e intercambiar datos entre la computadora y el ladrillo programable EV3 (Monk, 2020): mediante USB, inalámbricamente por Bluetooth o la red Wifi si se le integra al EV3 una USB Wifi compatible con este robot.

2.2.3.1 Conexión USB

Esta es la forma más común y simple de conectarse. Requiere un cable USB a Mini USB. Simplemente conectamos un extremo al puerto USB-host del EV3, y el otro extremo a un puerto USB disponible en la PC.

Luego encendemos el EV3 y en la PC se detectará como un dispositivo conectado. Esto permite transferir programas Python entre ambos. La conexión USB es ideal para las primeras pruebas y muy confiable en entornos con muchas señales inalámbricas.

2.2.3.2Conexión Bluetooth

Para conectarse de forma inalámbrica se debe vincular el EV3 con la PC mediante Bluetooth. Primero se activa el modo Bluetooth en el EV3 desde el menú "Wireless and Networks". Luego en la PC se busca el EV3 desde el administrador de dispositivos Bluetooth y se enlaza.

Al estar vinculados, PyBricks detectará el EV3 automáticamente. La conexión Bluetooth es más cómoda al evitar cables, pero puede presentar interrupciones en entornos con muchas interferencias. Requiere que la PC tenga Bluetooth o un adaptador USB.

2.2.3.3. Conexión Wifi

El robot Lego Mindstorms EV3 no tiene como tal un módulo Wifi incorporado, por lo que requiere un adaptador USB Wifi para poder conectarse a redes inalámbricas. Los requisitos del adaptador Wifi para programar el EV3 son:

- Compatibilidad con el EV3: El adaptador Wifi debe ser compatible con el hardware y sistema operativo del ladrillo EV3. Los más recomendados son los basados en el chip Realtek RTL8188CUS.
- Soporte para modo AP: El adaptador debe poder configurarse en modo de Punto de Acceso (AP) para crear una red Wifi directa con el EV3. Esto facilita la conexión inicial.

- Seguridad WPA2: Debe soportar el cifrado WPA2 (no sólo WEP) para conectarse de forma segura a redes modernas.
- Protocolo 802.11b/g/n: Es recomendable que soporte las bandas 2.4GHz en los estándares 802.11b/g/n para máxima compatibilidad.
- Interfaz USB 2.0: Requiere conexión USB 2.0 para una transmisión estable con el puerto USB del EV3.
- Bajo consumo energético: Debe consumir poca energía para no agotar rápido la batería del EV3.
- Antena flexible: Una antena flexible y extensible permite mejorar la señal en diferentes posiciones.

Los adaptadores que cumplen estos requisitos permitirán configurar el EV3 para programación vía WiFi de forma estable y confiable. Las marcas recomendadas son Edimax, TP-Link, Asus o D-Link.

2.2.4. El firmware del EV3

Para poder ejecutar los programas Python en el EV3, este debe tener instalado un firmware compatible con PyBricks (Lego, EV3 MicroPython, 2019). Hay dos opciones principales de firmware que funcionan:

2.2.4.1. Firmware oficial LEGO EV3 (versión 1.10E o superior)

Este es el firmware por defecto que trae el EV3. Contiene un intérprete de MicroPython especialmente adaptado para correr código PyBricks de forma remota mediante USB o conexión inalámbrica ya se por Bluetooth Wifi si se tiene un USB Wifi compatible con el robot. Este firmware es la opción recomendada para principiantes por su facilidad de uso.

2.2.4.2. EV3 MicroPython (versión 1.2 o superior)

Este firmware carga una imagen completa de MicroPython en el EV3 permitiendo ejecutar código Python directamente en el ladrillo sin necesidad de conexión con la PC. Es más avanzado y permite programación directa en el EV3, no sólo remota. Requiere más conocimientos para aprovechar todo su potencial.

2.2.5. Pasos para configurar y ejecutar un programa usando pybricks.

Paso 1: Preparar el ladrillo EV3

1. Descarga la imagen EV3 MicroPython desde el sitio web de LEGO Education.
2. Extrae el archivo de imagen a tu ordenador.
3. Inserta una tarjeta microSD en el lector de tarjetas de tu ordenador.
4. Usa una herramienta de flasheo, como Win32DiskImager, para flashear la imagen EV3 MicroPython en la tarjeta microSD.

Paso 2: Conectar el ladrillo EV3 al ordenador

1. Retira la tarjeta microSD del lector de tarjetas de tu ordenador.
2. Inserta la tarjeta microSD en la ranura para tarjetas SD del ladrillo EV3.
3. Enciende el ladrillo EV3.

Paso 3: Instalar la extensión LEGO MINDSTORMS EV3 Micropython para Visual Studio Code

1. Abre Visual Studio Code.
2. Ve a la pestaña Extensiones.
3. Busca la extensión LEGO MINDSTORMS EV3 Micropython.
4. Haz clic en el botón Instalar.
5. Haz clic en el botón Activar.

Paso 4: Conectar el ladrillo EV3 a Visual Studio Code

1. Conecta el ladrillo EV3 a tu ordenador usando un cable USB.
2. Haz clic en el icono EV3 Brick en la barra lateral de Visual Studio Code.

Paso 5: Escribir tu primer programa Python

1. Abre un nuevo archivo Python en Visual Studio Code.

   Escribe el siguiente código:

```
#!/usr/bin/env pybricks-micropython
from pybricks.hubs import EV3Brick
from pybricks.ev3devices import Motor
from pybricks.parameters import Port
from pybricks.tools import wait

'''Este programa es un ejemplo simple de cómo controlar
   un ladrillo EV3 usando Python y la biblioteca EV3
   MicroPython. Hace que el ladrillo EV3 emita un pitido,
   imprima un mensaje en la pantalla y luego espera 5 segundos.
'''
```

```
#---------------------------------------------------------
def main():
    # Crea tus objetos aquí.
    ev3 = EV3Brick()

    # Escribe tu programa aquí.
    ev3.speaker.beep()
    ev3.screen.print("Mi primer programa")
    wait(5000)

#---------------------------------------------------------
if __name__ == "__main__":
    main()
```

3. Guarda el archivo como my_first_program.py.

Paso 6: Ejecutar tu programa Python

1. Haz clic en el botón Ejecutar en la barra de herramientas de Visual Studio Code.

Tu robot LEGO EV3 se moverá hacia adelante durante 5 segundos.

2.2.6. Recursos de aprendizaje PyBricks.

Para dominar el uso de PyBricks en proyectos con el EV3, los siguientes recursos son muy recomendados:

- Sitio web oficial - Contiene excelente documentación con explicación detallada de todos los módulos, clases y métodos disponibles en PyBricks: https://pybricks.com
- Tutorial oficial de Lego - Provee una amplia guía paso a paso para construir y programar un robot tanque con PyBricks: https://education.lego.com/en-us/support/mindstorms-ev3/python-for-ev3
- Libro "Python for EV3" de Daniele Benedettelli - Un completo manual para aprender a programar el EV3 con Python y PyBricks.
- EV3Dev - Sitio web con gran información sobre programación del EV3 con MicroPython: https://www.ev3dev.org/docs/programming-languages/python/
- Foros de la comunidad - Sitios como EV3Dev, Stack Overflow y Reddit tienen foros activos para hacer preguntas y resolver dudas.

CAPUTILO 3. FUNDAMENTOS DE PYTHON

Python es un lenguaje de programación multiparadigma muy popular por su facilidad de uso y sintaxis clara y legible. Fue creado por Guido van Rossum a principios de la década de 1990 con el objetivo de ser un lenguaje accesible para todo tipo de programadores, incluso principiantes.

Python es interpretado, dinámicamente tipado y multiplataforma. Cuenta con una amplia biblioteca estándar incorporada y una gran comunidad que desarrolla módulos y paquetes adicionales de código abierto para expandir sus capacidades en distintas áreas de aplicación como ciencia de datos, desarrollo web, sistemas operativos, entre muchas otras (Python Software Foundation, 2022).

Python se ha consolidado como uno de los lenguajes de programación más populares y versátiles de los últimos tiempos. Desde aplicaciones web, videojuegos, inteligencia artificial hasta administración de sistemas, Python tiene un amplio uso en la industria tecnológica actual (Downey, The importance of Python programming skills. In A. Downey (Ed.), Think Python, 2016).

Su creciente adopción se debe en gran parte a que cuenta con una curva de aprendizaje suave para principiantes pero con el poder de escalar a aplicaciones complejas. A continuación exploraremos los elementos fundamentales del lenguaje que todo programador Python debe conocer.

3.1. SINTAXIS BÁSICA.

La sintaxis de un lenguaje de programación comprende las reglas que gobiernan la estructura y elementos que componen los programas. En Python se destaca por priorizar la legibilidad y facilidad de escritura mediante el uso de sangrías e identaciones para delimitar

bloques de código, en lugar de símbolos como paréntesis o llaves. Esto le brinda un formato visual muy ordenado y limpio (Sweigart, 2022).

Otra característica importante es que distingue entre mayúsculas y minúsculas (case sensitive), por lo que debemos mantener consistencia al nombrar variables y funciones. Los comentarios se indican con el símbolo de numeral (#), lo cual permite documentar y explicar diferentes partes del código sin afectar la ejecución. También se pueden hacer comentarios de varias líneas para explicar un bloque de código más grande, se inician con tres comillas simples (‘’’) o dobles (“””) al principio y al final del bloque.

Los programas de Python consisten en scripts o módulos almacenados en archivos con extensión “.py” que contienen las instrucciones interpretadas por el intérprete de Python.

La instrucción print es una de las más básicas y utilizadas en el lenguaje de programación Python, se usa para mostrar salidas de datos en la consola de comandos. Las salidas de datos pueden ser mensajes que se escriben entre comillas simples o dobles (Lutz, 2013).

Un ejemplo de un script simple es:

```
“””
   Este es un comentario de varias líneas
   Explica el funcionamiento de este bloque de código
“””
# Programa Hola Mundo
print('Hola Mundo') # Imprime string
print("Bienvenido a Python") # Otro string
nombre = "Juan" # Variable nombre
print(nombre) # Imprime variable
```

3.2 VARIABLES Y TIPOS DE DATOS.

Las variables son contenedores que almacenan valores o información para ser referenciados y manipulados en un programa. Python es un lenguaje dinámicamente tipado, lo que significa que no es necesario declarar explícitamente el tipo de dato al definir una variable, esto se determina automáticamente al asignarle un valor (Sweigart, 2022).

Python es sensible a las mayúsculas y minúsculas, lo cual quiere decir que distingue entre letras mayúsculas y minúsculas, por lo que debemos tener cuidado al nombrar variables, funciones, etc. Los nombres de variables no pueden iniciar con números, pero sí con guion bajo (_). Se recomienda utilizar la nomenclatura “snake_case” con guiones bajos para separar palabras en nombres de variables y funciones.

Los principales tipos de datos que se usarán en el código Python son:

- Enteros (int): para valores numéricos sin decimales. Ej: 25
- Flotantes (float): para valores numéricos con decimales. Ej: 3.1416
- Booleanos (bool): para valores True o False.
- Strings (str): para cadenas de texto. Ej: "Hola"
- Listas (list): colecciones de valores entre corchetes. Ej: [1, 2, 3]
- Tuplas (tuple): listas inmutables entre paréntesis. Ej: (1, True, "Hola")
- Diccionarios (dict): pares de clave-valor entre llaves. ej: {"nombre":"Juan","edad":20}

Un ejemplo de declaración de variables es:

```
# Variables en Python
entero = 15 # int
flotante = 10.5 # float
texto = "Bienvenido" # string
boleano = True # boolean
lista = [1, 2, 3] # list
tupla = ("a", "b", "c") # tuple
diccionario = {"nombre":"Ana", "edad":19} # dict
```

3.3. OPERADORES Y EXPRESIONES DE ASIGNACIÓN.

Los operadores son símbolos que representan acciones aplicables a valores y variables para realizar cálculos u operaciones. Los principales tipos de operadores en Python son (Downey, Think Python: How to think like a computer scientist (2nd ed.), 2015):

- Aritméticos: +, -, *, /, %, **, // para realizar operaciones matemáticas.
- Asignación: =, +=, -=, *=, /= para asignar valores.
- Comparación: ==, !=, >, <, >=, <= para comparar expresiones.
- Lógicos: and, or, not para combinar expresiones booleanas.
- Bitwise: &, |, ~, ^, >>, << para operar a nivel de bits.

Algunos ejemplos de uso:

```
# Operadores aritméticos
5 + 3 # Suma
10 - 4 # Resta
4 * 5 # Multiplicación
16 / 4 # División da un float
18 % 7 # Módulo o resto
5 ** 3 # Potencia
15 // 4 # División entera

# Operadores comparación
5 == 5 # Igualdad
10 != 5 # Desigualdad
15 > 10 # Mayor que

# Operadores lógicos
True and False # Y lógico
True or False # O lógico
not True # Negación

# Operadores bitwise
8 & 5 # AND a nivel de bits
9     | 3 # OR a nivel de bits
11^ 5 # XOR a nivel de bits
```

Estos operadores permiten realizar todo tipo de operaciones matemáticas, lógicas y a nivel de bits en Python. Se pueden combinar para crear expresiones un poco más complejas dentro del código, por ejemplo:

```
precio = 25
descuento = 12
precio_final = precio - (precio * descuento / 100)
print(precio_final) # Imprime 22.0
```

Python respeta las reglas convencionales de precedencia de operadores: paréntesis, exponenciación, multiplicación/división, suma/resta. Esto determina el orden en que se evalúan las expresiones para obtener el resultado correcto. Estos operadores permiten realizar todo tipo de manipulaciones y cálculos en un programa de Python según se necesite.

En Python existen diferentes operadores de asignación que nos permiten realizar una operación y asignación de forma simplificada en una sola expresión. El operador básico es el signo igual (=) que asigna el valor de la derecha a la variable de la izquierda (Sweigart, 2022):

```
x = 5 # asigna 5 a x
```

Además, están los operadores combinados como +=, -=, *=, /=, que combinan la operación con la asignación (Downey, 2015):

```
x += 3 # x = x + 3
x -= 2 # x = x - 2
x *= 4 # x = x * 4
x /= 2 # x = x / 2
```

Estos permiten acortar y simplificar expresiones donde queremos aplicar una operación a una variable existente.

También podemos encadenar múltiples asignaciones en una línea:

```
x = p = z = 0 # x, p y z asignados a 0
```

Donde los valores se asignan de derecha a izquierda.

En las asignaciones con expresiones podemos utilizar expresiones arbitrarias complejas del lado derecho:

```
x = (3 * 4) + (10 / 5)
```

Python evalúa la expresión, obtiene el resultado y luego lo asigna a la variable.

Las asignaciones múltiples permiten asignar varias variables en una sola expresión:

```
a, b, c = 10, 20, 30
```

Esto asigna 10 a ´a', 20 a 'b' y 30 a 'c' respectivamente.

Un uso muy práctico de la asignación múltiple es para intercambiar los valores de dos variables:

```
a = 10
b = 20
a, b = b, a # intercambia a y b
# a ahora es 20, b ahora es 10
```

las expresiones de asignación en Python nos permiten realizar operaciones y asignaciones simplificadas para escribir código más conciso y legible (Lutz, 2013), tienen varios usos para inicializar, modificar y manipular variables.

3.4 ESTRUCTURAS DE CONTROL.

Las estructuras de control de flujo permiten controlar el orden en que se ejecutan las instrucciones en un programa Python, de modo que podamos construir código más complejo y versátil que se adapte a distintas situaciones y requerimientos lógicos.

Python incluye varias estructuras de control que veremos a continuación:

3.4.1. Condicional if.

La sentencia if permite ejecutar código sólo si se cumple una condición booleana específica. Su sintaxis es:

```
if condicion:
  # código a ejecutar
```

Donde condicion puede ser cualquier expresión que evalúe a True o False. Por ejemplo:

```
x = 10
if x > 5:
  print("x es mayor a 5")
```

Esto imprimirá el mensaje porque la condición x > 5 se cumple.

Podemos encadenar múltiples condiciones con elif (abreviación de else if):

```
color = "rojo"
if color == "verde":
      print("El color es verde")
elif color == "rojo":
      print("El color es rojo")
elif color == "azul":
      print("El color es azul")
else:
      print("Color no reconocido")
```

Si ninguna condición if/elif se cumple, se ejecuta el bloque else opcional.

3.4.2. Ciclo for.

El ciclo for itera sobre los elementos de una secuencia que se repite, ejecutando el código en cada iteración. Por ejemplo, para iterar sobre una lista:

```
frutas = ["manzana", "banana", "kiwi"]
for fruta in frutas:
      print("La fruta es:", fruta)
```

O sobre un rango numérico con range():

```
for i in range(5):
   print(f"i vale {i}")
```

Esto imprime los números del 0 al 4.

También podemos iterar sobre caracteres de un string con for:

```
for letra in "Python":
   print(letra)
```

3.4.3. Ciclo while.

El ciclo while ejecuta un bloque de código mientras se cumpla una condición booleana. Por ejemplo:

```
i = 0
while i < 5:
   print(f"i vale {i}")
   i += 1 # incrementa i
```

Esto imprime el valor de 'i' mientras 'i' sea menor que 5, actualizando la variable en cada iteración.

3.4.4. Instrucciones break, continue, pass.

Estas instrucciones nos brindan mayor control sobre los ciclos:

- break: Termina completamente el ciclo actual
- continue: Vuelve al comienzo del ciclo
- pass: Equivale a no hacer nada, útil como marcado de posición o placeholder ósea un espacio en blanco o un símbolo que indica dónde se insertará una información particular más adelante

Por ejemplo:

```
# Rompe el ciclo si i es 3
for i in range(10):
  if i == 3:
    break
  print(i)

# Pasa a la siguiente iteración si i es impar
for i in range(10):
  if i % 2 != 0:
    continue
  print(i)

# Hace "nada"
for i in range(10):
  pass # placeholders
```

Dominar el uso de estructuras de control en Python nos permite crear programas versátiles capaces de hacer frente a todo tipo de situaciones y requerimientos lógicos. Estas construcciones de flujo son parte integral de cualquier lenguaje de programación.

3.5. FUNCIONES.

Las funciones son una característica esencial en cualquier lenguaje de programación. Una función agrupa un bloque de código para realizar una tarea o cálculo específico, lo cual promueve la reutilización, organización y modularidad del código (Lutz, 2013).

En Python podemos reconocer fácilmente las funciones porque su nombre va seguido de paréntesis. Veamos en detalle el concepto, sintaxis, tipos y usos de funciones en Python.

¿Qué es una función? Una función es un bloque de código identificado por un nombre que puede ser ejecutado invocando dicho nombre. Permite encapsular una secuencia de sentencias para realizar una tarea específica.

Las funciones pueden recibir entrada en forma de argumentos y parámetros, procesarlos de algún modo, y devolver una salida o resultado. Esto permite reutilizar las mismas instrucciones repetidamente sin tener que escribir el código desde cero (Sweigart, 2022).

3.5.1. Sintaxis de funciones.

La sintaxis básica para definir una función en Python es:

```
def nombre_funcion(parametros):
  # cuerpo de la función
  return resultado # opcional
```

Donde:

- def indica que se está definiendo una función.
- nombre_funcion es el identificador de la función.
- parámetros son los nombres para los datos de entrada. Son opcionales.
- return devuelve un resultado opcional al terminar.

Para ejecutar o llamar a la función se escribe su nombre seguido de paréntesis ():

```
nombre_funcion() # ejecuta la función
```

3.5.2. Parametrizando funciones.

Podemos definir funciones que reciban parámetros o argumentos como entrada. Estos se incluyen entre paréntesis en la definición:

```
def suma(a, b):
  total = a + b
  return total

resultado = suma(5, 8) # pasa 5 y 8 a los parámetros
print(resultado) # Imprime 13
```

Los parámetros actúan como variables locales dentro de la función.

3.5.3. Valores por defecto.

Podemos asignar valores por defecto a los parámetros para cuando no se pasen argumentos:

```
def saludar(nombre="Amigo"):
  print(f"Hola {nombre}")

saludar("Ana") # Salida: Hola Ana
saludar() # Salida: Hola Amigo
```

Esto hace la función más flexible y reusable.

3.5.4. Devolviendo valores.

Con return podemos devolver o retornar un valor desde la función:

```
def suma(a, b):
  return a + b

resultado = suma(10, 5) # se asigna el return a resultado
```

Si no hay return, la función devuelve None por defecto.

3.5.5. Tipos de funciones Python.

En Python encontramos distintos tipos de funciones:

- Funciones integradas: ya vienen incorporadas con Python como print(), len(), int(), etc.
- Módulos de la librería estándar: funciones agrupadas para realizar tareas específicas, como matemáticas, entrada/salida, web, OS, etc.
- Funciones definidas por el usuario: las que nosotros mismos creamos en nuestros programas para reutilizar código.
- Funciones anónimas o lambda: funciones sin nombre creadas en una sola línea con la palabra reservada lambda.
- Funciones recursivas: que se llaman a sí mismas para dividir problemas grandes en partes más pequeñas.

Veamos algunos ejemplos prácticos del uso de funciones en Python:

- Función para calcular el cubo de un número:

```
def cubicar(x):
  return x ** 3

resultado = cubicar(3)
print(resultado) # 27
```

- Función para saludar pasando nombre:

```
def saludar(nombre):
      print(f"¡Hola, {nombre}!")

saludar("Ana") # ¡Hola, Ana!
saludar("Pedro") # ¡Hola, Pedro!
```

- Funciones recursivas: Las funciones recursivas son un tipo de función en Python que se caracteriza por llamarse a sí misma dentro de su definición. Esto permite dividir problemas complejos en problemas más pequeños cada vez, hasta llegar a un caso base sencillo. (Downey, Think Python: How to think like a computer scientist (2nd ed.), 2015)

  La recursividad es una técnica muy útil para problemas que se pueden descomponer naturalmente en partes más simples del mismo tipo. Un ejemplo clásico son los algoritmos para calcular números factoriales:

```
def factorial(x):
  if x == 1:
    return 1
  else:
    return x * factorial(x-1)

print(factorial(5)) # 120
```

  Analicemos este ejemplo:

  - La función factorial() se llama a sí misma dentro del else.
  - El caso base es cuando x == 1, ahí se devuelve 1 y se termina la recursión.
  - La llamada recursiva es factorial(x - 1) que reduce el problema en 1 cada vez.

  De esta forma se calcula:

  - factorial(5) = 5 * factorial(4)
  - factorial(4) = 4 * factorial(3)
  - factorial(3) = 3 * factorial(2)
  - factorial(2) = 2 * factorial(1)
  - factorial(1) = 1 (caso base)

  Y se resuelve el problema combinando las llamadas recursivas.

  La recursión requiere:

  - Un caso base que termine la recursión.
  - Una llamada recursiva que acerque al caso base.

  Ventajas:

  - Soluciones elegantes y simples para problemas complejos.

- Código más corto y fácil de entender que iteraciones.

Desventajas:

- Alto consumo de memoria por las múltiples llamadas.
- Riesgo de recursión infinita si no hay caso base.

la recursión es una técnica avanzada que con práctica permite escribir algoritmos muy eficientes en Python. Debe usarse con precaución para evitar problemas de rendimiento o infinitos.

- Función lambda para sumar dos números:

```
sumar = lambda x, y: x + y

resultado = sumar(10, 15)
print(resultado) # 25
```

las funciones son bloques de código reutilizables que nos permiten crear programas Python más modulares, organizados y fáciles de mantener. Existen muchos tipos de funciones que podemos aprovechar en nuestros proyectos.

3.6. CLASES Y OBJETOS.

Python es un lenguaje que soporta ampliamente la programación orientada a objetos mediante clases y objetos. Esta es una forma muy útil y poderosa de organizar y diseñar nuestros programas.

3.6.1. ¿Qué es una clase?

Una clase es como un plano o blueprint para crear objetos. Define un conjunto de atributos (datos) y métodos (funciones) que serán compartidos por todos los objetos creados a partir de esa clase (Lott, 2022).

Las clases nos permiten modelar entidades del mundo real, como personas, vehículos, cuentas bancarias, etc. Cada objeto creado a partir de la clase se denomina instancia y contiene sus propios datos.

La sintaxis para definir una clase en Python es:

```
class NombreClase:

  # atributos de clase

  def __init__(self):
    # constructor initializer

  # métodos de clase

  def nombre_metodo(self):
    # código del método
```

Donde __init__() es el constructor que se ejecuta al crear un objeto. self hace referencia al objeto actual.

3.6.2. Creando objetos.

Podemos crear instancias u objetos a partir de una clase usando la notación:

```
obj1 = NombreClase() # crea objeto 1
obj2 = NombreClase() # crea objeto 2
```

Cada objeto tendrá sus propios atributos y compartirán los mismos métodos de clase.

3.6.3. Atributos.

Los atributos almacenan los datos asociados a un objeto particular. Se definen dentro de la clase y se accede con la notación punto:

```
obj1.atributo = valor # asignar valor
valor = obj1.atributo # acceder valor
```

3.6.4. Métodos.

Los métodos son funciones que representan comportamientos de los objetos. Se definen en la clase y se acceden también con notación punto:

```
obj1.metodo() # llamar método
```

Dentro de los métodos se utiliza self para referir los atributos del objeto actual.

Ejemplo de clase Persona: Veamos un ejemplo de una clase Persona con atributos nombre y edad, y métodos presentarse y cumplir_años():

```
class Persona:

  def __init__(self, nombre, edad):
    self.nombre = nombre # atributo
    self.edad = edad # atributo

  def presentarse(self):
    print(f"Hola soy {self.nombre}, tengo {self.edad} años")

  def cumplir_anos(self):
    self.edad += 1

# crear objetos
persona1 = Persona("Juan", 30)
persona2 = Persona("Ana", 18)

# llamar métodos
persona1.presentarse() # Hola soy Juan, tengo 30 años
persona2.cumplir_anos() # actualiza edad a 19
```

Esto demuestra los conceptos centrales de programación orientada a objetos en Python.

Una característica poderosa de manejar objetos es la herencia entre clases. Esto permite crear clases que hereden atributos y métodos de una clase padre, pudiendo definir comportamiento propio, la herencia proporciona reutilización de código y modelado de relaciones entre entidades del mundo real. La clase hija se define indicando entre paréntesis la clase padre de la que hereda.

las clases y objetos en Python nos permiten crear programas mejor estructurados, reutilizables y fáciles de mantener. Es un paradigma fundamental para desarrollar sistemas complejos de forma organizada y eficiente.

3.7. BIBLIOTECAS EN PYTHON.

Una biblioteca en Python es un conjunto de funciones y clases que se pueden utilizar en un programa. Las bibliotecas se pueden encontrar en línea o en el repositorio de paquetes de Python (PyPI).

Para utilizar una biblioteca en Python, primero debemos importarla. Esto se puede hacer usando el comando import. Por ejemplo, para importar la biblioteca math, podemos usar el siguiente código:

```
import math
```

Una vez que hemos importado una biblioteca, podemos acceder a sus funciones y clases usando el operador punto (.). Por ejemplo, para calcular el valor de pi, podemos usar el siguiente código:

```
import math
pi = math.pi
print(pi)
```

3.7.1. Tipos de bibliotecas en Python

Las bibliotecas en Python se pueden clasificar en dos tipos principales:

- Bibliotecas estándar: Estas bibliotecas forman parte del lenguaje Python, y están disponibles de forma predeterminada.
- Bibliotecas de terceros: Estas bibliotecas son desarrolladas por terceros, y deben instalarse por separado.

3.7.1.1. Bibliotecas estándar de Python.

La biblioteca estándar de Python incluye una amplia gama de funciones y clases que pueden utilizarse para realizar tareas comunes. Algunas de las bibliotecas estándar más importantes son:

- Biblioteca math: Esta biblioteca proporciona funciones para el cálculo numérico, como el cálculo de raíces cuadradas, logaritmos y funciones trigonométricas.

- Biblioteca random: Esta biblioteca proporciona funciones para generar números aleatorios.
- Biblioteca datetime: Esta biblioteca proporciona funciones para trabajar con fechas y horas.
- Biblioteca os: Esta biblioteca proporciona funciones para trabajar con el sistema operativo.
- Biblioteca sys: Esta biblioteca proporciona funciones para acceder a información sobre el sistema.

3.7.1.2. Bibliotecas de terceros en Python

Existen muchas bibliotecas de terceros disponibles para Python. Estas bibliotecas suelen ser elaboradas por comunidades de desarrolladores, y proporcionan funciones y clases especializadas para tareas específicas. Algunas de las bibliotecas de terceros más populares son:

- Biblioteca numpy: Esta biblioteca proporciona funciones para el cálculo numérico de alto rendimiento.
- Biblioteca scipy: Esta biblioteca proporciona funciones para el análisis estadístico y científico.
- Bibliotecas pandas: Esta biblioteca proporciona funciones para trabajar con datos tabulares.
- Biblioteca matplotlib: Esta biblioteca proporciona funciones para la visualización de datos.
- Biblioteca flask: Esta biblioteca proporciona un marco web para el desarrollo de aplicaciones web.

3.7.2. Cómo instalar bibliotecas de terceros en Python

Para instalar una biblioteca de terceros en Python, podemos usar el administrador de paquetes de Python (pip). Pip es una herramienta que nos permite instalar y administrar paquetes de Python, para instalar una biblioteca de terceros usando pip, podemos usar el siguiente comando:

- pip install [nombre_de_la_biblioteca]

Por ejemplo, para instalar la biblioteca numpy, podemos usar el siguiente comando:

- pip install numpy

3.7.3. Ventajas de las bibliotecas en Python.

Las bibliotecas en Python ofrecen una serie de ventajas, entre las que se incluyen:

- Reutilización de código: Las bibliotecas nos permiten reutilizar código que ya ha sido desarrollado y probado por otros desarrolladores. Esto nos ahorra tiempo y esfuerzo.
- Especialización: Las bibliotecas proporcionan funciones y clases especializadas para tareas específicas. Esto nos permite centrarnos en la lógica de nuestro programa, sin tener que preocuparnos por implementar las funciones y clases básicas.
- Documentación: Las bibliotecas suelen estar bien documentadas, lo que nos facilita su uso.

Las bibliotecas en Python son una herramienta esencial para cualquier programador de Python, nos permiten reutilizar código, especializarnos en tareas específicas y ahorrar tiempo y esfuerzo.

CAPITULO 4. PROGRAMACIÓN DEL LEGO EV3 USANDO LA BIBLIOTECA PYBRICKS

La biblioteca PyBricks de Python nos permite programar el robot Lego Mindstorms EV3 de una forma muy sencilla e intuitiva. En este capítulo veremos cómo aprovechar PyBricks para controlar el EV3 mediante código Python escrito en el computador.

4.1. EL LADRILLO DEL LEGO EV3.

El ladrillo o brick programable EV3 es el cerebro del robot Lego Mindstorms EV3. Se encarga de recibir instrucciones de los programas y coordinar la interacción entre los motores, sensores, luces y otros componentes del robot (Lego, Explore the Lego Mindstorms Inventions., 2021).

Figura 13. Ladrillo Lego EV3

El EV3 es básicamente una pequeña computadora embebida optimizada para la robótica educativa. Cuenta con un procesador ARM9 a 300 MHz, 64 MB de memoria RAM y 16 MB de almacenamiento flash ampliable con tarjeta SD (Gindrat, Lego Mindstorms EV3: building and programming a complex robot. Journal of Computing Sciences in Colleges, 2014).

El sistema operativo que corre internamente en el EV3 es Linux, lo que le brinda gran potencia y flexibilidad. El firmware oficial de Lego trae el software EV3-G basado en iconos para programar mediante arrastrar y soltar comandos.

Una solución aún más sencilla es usar la biblioteca PyBricks de Python. Esta se instala directamente en el computador e interactúa de forma inalámbrica con el EV3 sin necesidad de modificar el sistema operativo del robot (PyBricks, 2022).

Sin embargo, también es posible reemplazar el firmware por alternativas como EV3Dev o EV3 micropython que permiten programar el robot en lenguajes de texto como Python. Esto nos da mucho más control y funcionalidades avanzadas.

4.1.1 Componentes del EV3.

El ladrillo EV3 contiene los siguientes componentes principales:

- Puerto USB: para conectar a la PC mediante cable USB.
- Puertos de entrada: para conectar sensores y obtener datos del entorno.
- Puertos de salida: para conectar motores y actuadores.
- Pantalla LCD: pequeña pantalla para mostrar texto, imagen y datos.
- Botones: botones de navegación arriba, abajo, izquierda, derecha, central y atrás.
- Bocina: para reproducir sonidos y efectos de audio.
- Luz de estado: led indicador de encendido, apagado, batería, etc.

Todos estos componentes pueden ser controlados y programados utilizando PyBricks para ejecutar las más diversas operaciones y tareas en el robot.

Veamos un programa muy simple para encender un led en el EV3 usando PyBricks en Python:

```
#!/usr/bin/env pybricks-micropython
from pybricks.hubs import EV3Brick
```

```
from pybricks.tools import wait
from pybricks.parameters import Color

'''
    Este programa controla la luz del ladrillo EV3
    para mostrar una luz roja durante un segundo.
'''
def main():
    # Inicializar el EV3
    ev3 = EV3Brick()

    # Encender una luz roja
    ev3.light.on(Color.RED)

    # Esperar un 5 segundos
    wait(5000)

    # Apagar la luz
    ev3.light.off()

#------------------------------------------------------------
if __name__ == "__main__":
   main()
```

Sólo necesitamos importar la clase EV3Brick , luego obtener una referencia al ladrillo e indicar al led que se encienda.

La gran ventaja de PyBricks es precisamente esa simplicidad y facilidad de uso, ideal para introducir a nuevos usuarios en la robótica con Python de forma rápida e intuitiva.

Los robots Lego Mindstorms EV3 pueden ser equipados con una gran variedad de motores, sensores y otros componentes que les permiten moverse, percibir su entorno y ejecutar tareas muy sofisticadas. Veamos cómo programar estos dispositivos usando Python y la biblioteca PyBricks.

4.1.2. Motores del lego EV3.

Uno de los componentes más importantes del robot Lego Mindstorms EV3 son sus motores interactivos, los cuales permiten dotar de movimiento y acción a los modelos que construyamos. Existen diferentes tipos de motores EV3, cada uno con propiedades y características únicas.

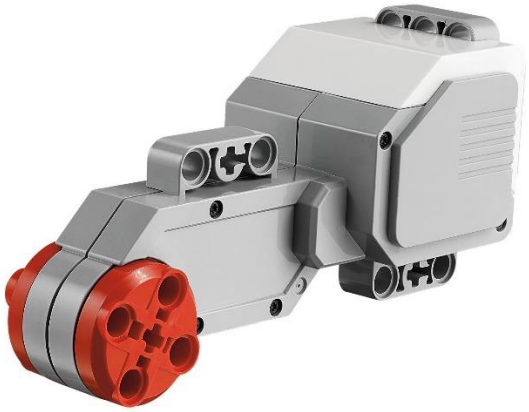

Figura 14. Servomotor Grande Lego EV3

4.1.2.1. Tipos de motores EV3.

El kit básico del EV3 trae 3 servomotores cuyo funcionamiento se basa en un motor eléctrico de corriente directa combinado con un encoder o sensor de rotación incremental que permite monitorear con gran precisión parámetros como velocidad, dirección y posición (Jordan, J., & Fern, A. , 2019).

Los 3 motores son idénticos en su funcionamiento excepto por su tamaño y forma:

Motor grande: tiendo una velocidad sin carga de 160 rpm aproximadamente y torque de 20 Ncm (más lento, pero más potente).

- Motor mediano: velocidad de 240 rpm sin carga y torque de 8 Ncm. Es más rápido pero menos potente (más rápido, pero menos potente).
- Motor mini: alcanza 390 rpm sin carga y tiene un torque de 1.5 Ncm. Para proyectos pequeños por su reducido tamaño.

Figura 15. Servomotor mediano lego EV3

También existen otras opciones como el motor eléctrico, el servo motor y el turbo motor que se venden por separado y amplían las capacidades del EV3 (Gindrat, Lego Mindstorms EV3: building and programming a complex robot. Journal of Computing Sciences in Colleges, 2014).

4.1.2.2. Control de motores usando la biblioteca pybricks.

Esta librería es una excelente opción, enfocada en facilidad de uso. Permite controlar el EV3 vía Bluetooth desde Python en el PC utilizando una sintaxis muy clara y minimalista.

Veamos un ejemplo de control de motores con PyBricks:

```
#!/usr/bin/env pybricks-micropython
from pybricks.hubs import EV3Brick
from pybricks.ev3devices import Motor
from pybricks.parameters import Port
from pybricks.tools import wait

def main():
    # Inicializar el EV3
    ev3 = EV3Brick()

    # Inicializar un motor en el puerto A
    test_motor = Motor(Port.A)

    # Mueve el motor a una velocidad rotacional de
    # 500 grados por segundo, durante 5 segundos
    test_motor.run(500)
    wait (5000)
    # Detiene el motor
    test_motor.stop()
#-------------------------------------------------------------

if __name__ == "__main__":
   main()
```

Podemos hacer operaciones más avanzadas como:

- Encender el motor hacia atrás o adelante.
- Rampas de aceleración/desaceleración.
- Rotar a un ángulo absoluto o relativo específico.
- Leer precisamente la posición y velocidad actual.

Los motores le dan "vida" a nuestros robots permitiéndoles desplazarse y realizar trabajo físico de forma autónoma mediante la programación en Python.

4.1.2.3. Consideraciones de diseño.

Al integrar los motores en un robot EV3 debemos tomar en cuenta ciertas consideraciones de diseño (Jordan, J., & Fern, A. , 2019):

- Tracción: relación entre torque y fricción con el piso, importante en robots móviles.
- Holgura: juego mecánico en ruedas y ejes que puede afectar precisión.
- Centro de masa: debe estar bien ubicado para estabilidad.
- Relación de transmisión: velocidad de rotación de las ruedas respecto al motor.
- Fricción en las ruedas: superficies con mayor tracción proveen mejor control.

Combinando una construcción mecánica adecuada con una programación rigurosa en Python, podemos crear robots Lego EV3 verdaderamente asombrosos y competitivos.

Los motores son el corazón del sistema robótico del EV3 y nos permiten dar rienda suelta a nuestra creatividad e imaginación para construir todo tipo de máquinas y vehículos integrados con Python.

Además de los tres servomotores estándar, existen varios otros motores compatibles con el EV3 que permiten expandir las posibilidades de nuestros proyectos (Gindrat, Lego Mindstorms EV3: building and programming a complex robot. Journal of Computing Sciences in Colleges, 2014):

- Motor eléctrico: similar a los servomotores, pero más económico y sin encoder integrado. Útil para accionar ruedas en robots simples.
- Servo motor: permite control de posición angular exacta entre 0 y 300 grados. Ideal para brazos robóticos u otras articulaciones.
- Turbo motor: tiene engranaje reforzado y mayor torque, hasta 25 N-cm. Para proyectos de alta potencia.
- Motor mediano con tacómetro: versión del motor mediano con sensor de rotación absoluto incorporado. Mayor precisión.

La biblioteca PyBricks soporta sin problemas estos motores alternativos, permitiendo controlarlos fácilmente con Python al igual que los servomotores estándar, sólo debemos especificar el tipo de motor al crear el objeto:

```
from pybricks.ev3devices import ElectricMotor, ServoMotor,
TachoMotor
```

```
motor_electrico = ElectricMotor(Port.A)
servo_motor = ServoMotor(Port.B)
tacho_motor = TachoMotor(Port.C)
```

De esta forma podemos combinar distintos motores según los requerimientos específicos de cada proyecto de robot EV3.

4.1.2.4. La clase Motor de la biblioteca PyBricks.

Esta clase incluye varios métodos útiles para controlar los motores del Lego EV3 (PyBricks, 2022):

- run(speed): Hace girar el motor a una velocidad constante, donde speed es la velocidad rotacional del motor (en grados/s).
- stop(): Detiene el motor suavemente.
- brake(): Frena bruscamente el motor.
- run_time(speed, time, then=Stop.COAST, wait=True): Hace gira el motor a una velocidad contante durante un tiempo dado, donde: speed es la velocidad rotacional del motor (en grados/s), time (en ms) es la duración del movimiento del motor, then=Stop indica que debe hacer el hacer motor después del tiempo de movimiento, y wait (bool) indica si el motor espera a que el movimiento se complete antes de que continue con el programa.
- run_angle(speed, rotation_angle, then=Stop.HOLD, wait=True): Hace girar el motor a una velocidad constante un ángulo específico, donde speed (en grados/s) es la velocidad rotacional del motor, rotation_angle (en deg) es el angulo que el motor debe rotar, then=Stop indica que hace el motor después de la rotación, y wait (bool) indica si el motor espera a que el movimiento se complete antes de que continue con el programa.
- run_target(speed, target_angle, then=Stop.HOLD, wait=True): Hace girar el motor a una velocidad constante hasta alcanzar un ángulo absoluto como objetivo, donde speed (en grados/s) es la velocidad rotacional del motor(puede ser positiva o negativa), target_angle (en deg) es el ángulo que el motor debe rotar, then=Stop indica que hace el motor después de la rotación, y wait (bool) indica si el motor espera a que el movimiento se complete antes de que continue con el programa.
- track_target(target_speed): Gira el motor para alcanzar y mantener una velocidad objetivo.
- hold(): Mantiene el motor en su posición actual estable.
- reached_angle(angle): Retorna True si el motor alcanzó cierto ángulo.
- speed(): Devuelve la velocidad actual en rpm.
- angle(): Devuelve el ángulo actual del motor en grados.

La clase Motor de PyBricks proporciona un amplio control sobre los motores del EV3 en Python de una forma simple e intuitiva.

Ejemplo moviendo un motor a una velocidad fija:

```
from pybricks.ev3devices import Motor
from pybricks.parameters import Port

motor = Motor(Port.B)

# mueve el motos a una velocidad
# rotacional de 50 grados por segundo
motor.run(50)
```

Ejemplo rotando el motor un ángulo específico:

```
# mueve el motor a una velocidad
# rotacional constante de 30 grados sobre
# segundo en un ángulo de 360 grados
motor.run_angle(30, 360)
```

Ejemplo deteniendo el motor suavemente:

```
# Detener motor lentamente
motor.stop()
```

Ejemplo verificando si el motor alcanzó cierto ángulo:

```
if motor.reached_angle(90):
  ev3.screen.print("Ángulo alcanzado!")
```

Ejemplo leyendo valores del motor:

```
ev3.screen.print(motor.speed()) # Velocidad actual
ev3.screen.print(motor.angle()) # Ángulo actual
```

La API de motores de PyBricks facilita tareas comunes de control de forma concisa y clara.

Los motores EV3 junto con Python nos dan todas las herramientas para hacer realidad las más ambiciosas ideas de proyectos robóticos como la implementación de robots sumo,

vehículos todo terreno, robots que navegan entre laberintos, brazos robóticos articulados, drones cuadrúpedos e impresoras 3D. ¡Tu imaginación es el único límite!

4.2. SENSORES EV3.

Algunos sensores comunes del EV3 son (Lego, Explore the Lego Mindstorms Inventions., 2021):

- Sensor de color: detecta intensidad y color de objetos mediante luz visible o infrarroja.
- Sensor de ultrasonido: calcula distancia a objetos mediante ondas de sonido, con un alcance máximo 2 metros.
- Sensor giroscópico: mide la orientación y ángulos de inclinación del EV3.
- Sensor táctil: detecta toques y golpes como un interruptor simple.
- Sensor infrarrojo: puede detectar luz infrarroja reflejada por objetos macizos.

Para leer un sensor con PyBricks, creamos una instancia de su clase correspondiente y llamamos métodos como .color() o .distance():

```
from pybricks.ev3devices import UltrasonicSensor

sonar = UltrasonicSensor(Port.S4)

distancia = sonar.distance() # lectura en mm
if distancia < 200:
  # detener motores si obstáculo cerca
```

4.2.1. Sensor de color

El sensor de color es uno de los sensores más versátiles del robot Lego Mindstorms EV3. Nos permite detectar colores e intensidad de luz en objetos y superficies. Veamos en detalle sus capacidades y cómo podemos programarlo en Python con la biblioteca PyBricks.

Figura 16. Sensor de color Lego EV3

El sensor de color del EV3 utiliza un emisor de luz y un sensor CMOS para detectar la intensidad y frecuencia de la luz reflejada (Jordan, J., & Fern, A. , 2019). Funciona tanto con luz visible como infrarroja.

4.2.1.1. Especificaciones

Sus especificaciones técnicas son:

- Tipo: Sensor de luz ambiental y de color
- Entradas: 6 componentes de luz (rojo, azul, verde, rojo intenso, azul intenso, verde intenso)
- Resolución: 3% de intensidad de luz ambiental
- Rango de medición: de 0 a 100% de intensidad de reflexión
- Modos: Color, intensidad reflejada, sensor de luz ambiental
- Interfaz: I2C y SMBus
- Alcance: 1-2 cm aproximadamente con fuente de luz integrada

El sensor puede identificar 7 colores primarios - negro, azul, verde, amarillo, rojo, blanco y café. También detecta intensidad de luz ambiental y niveles de gris. Requiere calibración inicial en algunos casos para máxima precisión.

4.2.1.2. Programación con PyBricks.

La biblioteca PyBricks de Python provee una clase ColorSensor para interactuar fácilmente con el sensor de color del EV3. Un ejemplo sencillo es:

```
#!/usr/bin/env pybricks-micropython
```

```
from pybricks.hubs import EV3Brick
from pybricks.ev3devices import ColorSensor
from pybricks.parameters import Port
from pybricks.tools import wait

'''
    Este programa captura el nombre de color de un objeto
    y la intensidad de luz ambiental en una escala.
    del 0 al 10.
'''
def main():
    # Inicializar el EV3
    ev3 = EV3Brick()

    # Inicializar el sensor de color en el puerto 4
    sensor_color = ColorSensor(Port.S4)

    # Imprimir el color y la intensidad de luz ambiental
    # en la pantalla del EV3
    ev3.screen.print(sensor_color.color())
    ev3.screen.print(sensor_color.ambient())

    # Esperar 5 segundos
    wait(5000)
#-------------------------------------------------------------
if __name__ == "__main__":
   main()
```

El método .color() devuelve el color detectado como valor enumerado (Color.BLACK, Color.BLUE, etc).

Para calibrar el sensor se utiliza el método .calibrate_white() colocando una superficie blanca frente al sensor. Esto mejora la precisión de detección.

También podemos configurar el sensor en otros modos de operación:

```
# Modo de reflexión de luz
sensor_color.mode = ColorSensor.REFLECT
ev3.screen.print(sensor_color.reflection())

# Modo de detección de luz ambiental
sensor_color.mode = ColorSensor.AMBIENT
ev3.screen.print(sensor_color.ambient())
```

Algunos ejemplos prácticos de uso del sensor de color:

- Seguir una línea negra sobre fondo blanco
- Identificar y clasificar objetos por color
- Detectar tipos de superficie por intensidad reflejada

- Leer códigos de colores o patrones
- Calibrar con un color de referencia antes de lecturas precisas

El sensor de color resulta indispensable en una gran variedad de proyectos robóticos donde se requiere que el EV3 obtenga información sobre colores y luz del entorno. Combinado con Python y PyBricks permitirá implementar los comportamientos autónomos fascinantes.

4.3.2 Sensor de ultrasonido.

El sensor de ultrasonido es uno de los dispositivos de entrada más útiles en el Lego EV3. Permite al robot detectar objetos y calcular distancias en un rango aproximado de 0 a 250 cm. Analicemos en detalle cómo funciona y cómo programarlo con Python.

Figura 17. Sensor ultrasónico lego EV3

4.3.2.1. Especificaciones

El sensor de ultrasonido del EV3 funciona emitiendo ondas de ultrasonido a 40kHz y midiendo el tiempo que tardan en rebotar en un objeto y regresar (Jordan, J., & Fern, A. , 2019).

Sus especificaciones técnicas son:

- Tipo: sensor de proximidad por ultrasonido
- Rango de detección: de 1 a 250 cm aproximadamente.
- Resolución: +/- 1 cm
- Ángulo efectivo: 15 grados aproximadamente.

- Interfaz: I2C y SMBus
- Alimentación: 5-6 VDC

El sensor incorpora un emisor de ultrasonido, un micrófono, circuitos de control y un procesador ARM7 que se encarga de los cálculos para determinar distancia. Requiere calibración ocasional para máximo rendimiento.

4.3.2.1. Programación con PyBricks.

La biblioteca PyBricks provee una clase UltrasonicSensor para interactuar fácilmente con este sensor en Python. Un ejemplo básico es:

```
#!/usr/bin/env pybricks-micropython
from pybricks.hubs import EV3Brick
from pybricks.ev3devices import UltrasonicSensor
from pybricks.parameters import Port
from pybricks.tools import wait

''' Este programa lee la distancia en mm de un
    sensor de ultrasonido
'''

def main():
    # Inicializar el EV3
    ev3 = EV3Brick()

    # Inicializar el sensor de ultrasonido en el puerto 2
    sensor_ultrasonido = UltrasonicSensor(Port.S2)

    # Imprimir la distancia en mm
    ev3.screen.print(sensor_ultrasonido.distance())

    # Esperar 5 segundos
    wait(5000)

#-------------------------------------------------------------
if __name__ == "__main__":
   main()
```

El método .distance() dispara un pulso de ultrasonido y calcula la distancia al objeto más cercano detectado en un rango de 0 a 2550 mm.

Otros métodos útiles son:

- presence(): Devuelve True/False si detecta objeto dentro de 200 mm.
- listen(): Retorna el tiempo entre un eco inicial y un eco final en microsegundos.

Algunos ejemplos de uso del sensor de ultrasonido:

- Evitar obstáculos en un robot móvil midiendo distancia a objetos.
- Calcular distancia desde el EV3 a una pared u otro robot.
- Estimar volumen de objetos sólidos midiendo diferentes lados.
- Crear un seguidor de objetos manteniendo cierto rango de distancia.
- Generar un escaneo 360 grados girando el sensor para mapear entorno.

Por ejemplo, para detectar objetos a menos de 10 cm de distancia:

```
#!/usr/bin/env pybricks-micropython
from pybricks.hubs import EV3Brick
from pybricks.ev3devices import UltrasonicSensor
from pybricks.parameters import Port
from pybricks.tools import wait

''' Este programa dectecta objetos usando el sensor ultrasonico
    y emite un pitido cuando detecta un objeto a menos de 100 mm
    de distancia
 '''
def main():
    # Inicializar el EV3
    ev3 = EV3Brick()

    # Inicializar el sensor de ultrasonido en el puerto 2
    sensor_ultrasonido = UltrasonicSensor(Port.S2)

    # Repetir 10 veces
    for i in range(10):
        d=sensor_ultrasonido.distance()
        ev3.screen.print("Distancia --> ",d)
        # Si la distancia es menor a 100 mm
        if (d < 100):
            # Emitir un pitido
            ev3.speaker.beep()
        # Esperar 0.5 segundos
        wait(500)

#-----------------------------------------------------------
if __name__ == "__main__":
   main()
```

El sensor de ultrasonido abre un mundo de posibilidades de interacción autónoma del EV3 con su entorno físico mediante medición de distancias con el poder de Python.

4.3.3. Sensor Giróscópico del Lego EV3.

El sensor giróscópico es un dispositivo clave para dotar al robot Lego EV3 de mayor orientación y equilibrio. Permite medir ángulos e inclinación con gran precisión. Analicemos su funcionamiento y aplicaciones.

Figura 18. Sensor giroscopio lego Ev3

4.3.3.1. Especificaciones.

El sensor giróscópico del EV3 posee un giroscopio MEMS de 3 ejes que detecta orientación espacial y rotación del ladrillo de control con respecto a los ejes X, Y y Z (Jordan, J., & Fern, A. , 2019). Sus especificaciones técnicas son:

- Tipo: sensor MEMS de 3 ejes.
- Rango de medición: ±7 g de aceleración lineal, ±1200 grados/seg rotación.
- Resolución: 0.01 grados en rotación, 0.0025 grados de inclinación.
- Frecuencia de muestreo: 100 Hz.
- Interfaz: I2C/SPI.

El sensor incorpora tecnología de auto-test y calibración. Mide ángulos absolutos permitiendo conocer la orientación precisa respecto a un eje de referencia. No requiere calibración manual.

4.3.3.2. Programación en PyBricks.

La biblioteca PyBricks provee una clase GyroSensor para leer fácilmente los valores del sensor giróscopico en Python:

```
#!/usr/bin/env pybricks-micropython
from pybricks.ev3devices import GyroSensor
from pybricks.parameters import Port

# Inicializar el EV3
ev3 = EV3Brick()

sensor_giroscopio = GyroSensor(Port.S2)

# Ángulo actual
angulo = sensor_giroscopio.angle()

# Velocidad angular actual
vel_angular = sensor_giroscopio.speed()
```

El método .angle() devuelve la rotación del EV3 en grados en el momento de encender el sensor.

El método .speed() indica la velocidad angular actual en grados/segundo, la valores positivos para giro son en sentido horario.

El método .reset_angle(0) reinicia el ángulo de referencia a cero (0) grados.

Algunos ejemplos de aplicación del sensor giróscopico:

- Mantener el equilibrio en un robot bípedo o cuadrúpedo midiendo inclinación.
- Girar un robot móvil un ángulo determinado midiendo la rotación.
- Detección de caídas o colisiones monitorizando aceleraciones.
- Control de estabilidad y corrección de rumbo en drones, aviones o barcos teledirigidos.
- Sistemas de estabilización de cámara sobre bases móviles o brazos robóticos.

Por ejemplo, para girar un robot exactamente 90 grados:

```
#!/usr/bin/env pybricks-micropython
from pybricks.robotics import DriveBase
```

```
robot = DriveBase(...)

# Girar mientras no se alcance objetivo
while sensor_giroscopio.angle() < 90:

  robot.drive(100, 0) # girar

# Detenerse
robot.stop()
```

De esta manera podemos realizar movimientos y navegación precisa utilizando el sensor giróscópico para medición de ángulos y rotación.

Sus capacidades permiten implementar comportamientos muy avanzados de estabilidad y control de movimiento en una amplia variedad de proyectos robóticos con Lego EV3.

4.3.4 Sensor Táctil del Lego EV3.

El sensor táctil es ideal para detectar contacto físico y golpes en el robot Lego EV3. Funciona como un interruptor simple que activa o desactiva un circuito eléctrico al ser presionado. Veamos cómo aprovecharlo en nuestros proyectos.

Figura 19. Sensor de tacto lego EV3

4.3.4.1 Especificaciones.

El sensor táctil del EV3 consta de un resorte conductor y un botón interno que hacen contacto al ser presionado cerrando un circuito eléctrico (Jordan, J., & Fern, A. , 2019), sus especificaciones son:

- Tipo: interruptor táctil de presión por resorte.
- Entradas: botón presionable.
- Salidas: circuito abierto/cerrado.
- Fuerza de operación: 1-3 Newtons aprox.
- Recorrido de botón: 2 mm - 4 mm.
- Material: plástico ABS resistente a impactos.
- Conexión: cable 2-pin.

El sensor táctil no requiere ninguna configuración o calibración. Simplemente se conecta a un puerto de entrada y está listo para usarse.

4.3.4.2. Programación en PyBricks.

La biblioteca PyBricks provee una clase TouchSensor para leer el estado del sensor táctil. Un ejemplo básico es:

```
#!/usr/bin/env pybricks-micropython
from pybricks.ev3devices import TouchSensor
from pybricks.parameters import Port
from pybricks.hubs import EV3Brick
from pybricks.tools import wait

def main():
    # Inicializar el EV3
    ev3 = EV3Brick()

    # Inicializar el sensor de de tacto por el puerto 4
    sensor_tactil = TouchSensor(Port.S4)

    while True:
        if sensor_tactil.pressed():
            # Hacer algo cuando se presiona
            ev3.screen.print("Presionado!")
        else:
            # Acción por defecto
            ev3.screen.print("Soltado")

#------------------------------------------------------------
if __name__ == "__main__":
   main()
```

El método .pressed() devuelve True mientras el botón esté presionado, y False cuando se suelta.

Algunos ejemplos de uso del sensor táctil:

- Detectar cuando el EV3 choca contra un obstáculo en un robot móvil.
- Sensor bumper que active un retroceso o cambio de trayectoria.
- Disparar una acción cuando se presiona, como lanzar una pelota.
- Botones sencillos de control manual del EV3.
- Sistema de seguridad con alarma activada al presionar.

La simplicidad y bajo costo del sensor táctil lo hacen ideal para una amplia gama de aplicaciones en proyectos de robótica Lego.

4.3.5. Sensor infrarrojo

El sensor infrarrojo es un dispositivo electrónico muy útil que permite al robot Lego EV3 detectar obstáculos y medir distancias de forma precisa mediante infrarrojos. Veamos cómo funciona y cómo aprovecharlo en nuestros proyectos.

Figura 20. Sensor Infrarrojo

4.3.5.1. Especificaciones Técnicas

El sensor infrarrojo del EV3 funciona emitiendo luz infrarroja y midiendo la intensidad de la luz reflejada para determinar la proximidad de objetos cercanos (Jordan, J., & Fern, A. , 2019), sus especificaciones son:

- Tipo: sensor de proximidad infrarrojo activo.
- Rango de detección: de 0 a 100 cm aproximadamente.
- Resolución: +/- 1 cm.
- Ángulo de haz: < 30 grados.
- Salida: valores analógicos entre 0 y 100%.
- Interfaz: I2C/SMBus.
- Alimentación: 5-6 VDC desde el EV3.
- Dimensiones: 43.2 x 30.5 mm.

4.3.5.2. Programación en PyBricks

La biblioteca PyBricks de Python provee la clase InfraredSensor para leer fácilmente el sensor infrarrojo:

```
#!/usr/bin/env pybricks-micropython
from pybricks.ev3devices import InfraredSensor
from pybricks.parameters import Port
from pybricks.hubs import EV3Brick
from pybricks.tools import wait

def main():
    # Inicializar el EV3
    ev3 = EV3Brick()

    # Inicializar el sensor de infrarojo por el puerto 4
    infrarrojo = InfraredSensor(Port.S4)

    distancia = infrarrojo.distance() # Lee distancia en cm
    ev3.screen.print("Distancia:", distancia)

    # Esperar 5 segundos
    wait(5000)

#-------------------------------------------------------------

if __name__ == "__main__":
        main()
```

El método .distance() devuelve la distancia al objeto más cercano detectado en un rango de 0 a 100 cm.

Otros métodos útiles son:

- .presence(): Devuelve bool si detecta objeto a < 25 cm.

- .beacon(channel): Lectura de baliza infrarroja en canal especificado.
- .proximity(): Porcentaje de intensidad de luz reflejada.

Algunos ejemplos de uso del sensor infrarrojo:

- Medir distancia a pared para que un robot no choque.
- Detector de proximidad para parar antes de colisionar.
- Sensor de posición en una baliza infrarroja.
- Calibración de color de líneas comparando intensidad.
- Comunicación de baliza a baliza entre robots.
- Detección de fuentes de calor mediante infrarrojos.

Por ejemplo, para medir distancia a objetos:

```
while True:
  if infrarrojo.distance() < 15:
    # Obstáculo cerca, detenerse
  else:
    # Avanzar
```

De esta manera un robot móvil puede esquivar obstáculos midiendo en todo momento la distancia con el sensor infrarrojo.

El sensor infrarrojo del EV3 es una importante herramienta para dotar a los robots de percepción del entorno, especialmente en tareas de navegación, detección y medición. Mediante infrarrojos, el sensor amplía enormemente las capacidades del robot para interactuar con su entorno físico, utilizando la biblioteca PyBricks, podemos aprovechar fácilmente toda la potencia de este sensor en nuestras aplicaciones.

4.3.6. Pantalla LCD del lego EV3.

La pantalla LCD incorporada en el ladrillo EV3 permite mostrar información visual muy útil para interacción con el usuario o depuración de programas. Analicemos sus capacidades y cómo controlarla en Python.

4.3.6.1. Especificaciones.

La pantalla del EV3 es un display LCD monocromo de 178x128 pixels con retroiluminación LED, orientado en formato horizontal (Jordan, J., & Fern, A. , 2019), sus especificaciones técnicas son:

- Tipo: LCD TFT monocromo.

- Resolución: 178 x 128 pixels.
- Área visible: 36 x 24 mm.
- Retroiluminación: 4 LEDs blancos.
- Interfaz: SPI.
- Contraste: control por firmware.
- Diagonal: 47 mm.

La pantalla es controlada por el sistema operativo del EV3. El firmware de Lego trae fuentes y controladores optimizados para este display.

4.3.6.2 Programación con PyBricks.

La biblioteca PyBricks de Python provee la clase screen para controlar la pantalla del EV3 de forma sencilla:

```
#!/usr/bin/env pybricks-micropython
from pybricks.hubs import EV3Brick
from pybricks.tools import wait

# Inicializar el EV3
ev3 = EV3Brick()

# Mostrar texto en pantalla
ev3.screen.print("Hola Mundo!")

# Esperar 5 segundos
wait(5000)
```

El método .print() permite imprimir una cadena especificada. Otros métodos útiles son:

- .clear(): Limpia la pantalla.
- draw_text(x, y, "Texto"): Escribe un texto en las coordenadas (x,y)
- .draw_pixel(x, y): Dibuja un pixel en las coordenadas (x,y).
- .draw_box(x, y, ancho, alto): Dibuja rectángulo.
- .draw_circle(x,y,radio,relleno): Dibuja un círculo.
- .draw_line(x1,y1,x2,y2): Dibuja una línea.
- .load_image(imagen): Dibuja la imagen en la pantalla, algunas imágenes son:
    - ImageFile.ANGRY
    - ImageFile. NEUTRAL
    - ImageFile. QUESTION_MARK
    - ImageFile. PINCH_RIGHT
    - ImageFile. WARNING

- ImageFile. AWAKE
- ImageFile. STOP_1
- ImageFile.EV3_ICON
- ImageFile. HURT
- ImageFile. ACCEPT
- ImageFile. DECLINE

La pantalla LCD resulta muy útil para:

- Mostrar estado, mensajes, valores de sensores, etc.
- Visualizar imágenes, gráficos, formas simples.
- Interfaz gráfica básica de usuario.
- Animaciones y efectos visuales.
- Depuración y troubleshooting de programas.

Por ejemplo, para un juego que muestra vidas restantes:

```
#!/usr/bin/env pybricks-micropython
from pybricks.hubs import EV3Brick
from pybricks.tools import wait
from pybricks.ev3devices import TouchSensor
from pybricks.parameters import Port
from pybricks.media.ev3dev import Image, ImageFile

# Inicializar el EV3
ev3 = EV3Brick()

# Inicializar el sensor de tacto por el puerto 4
sensor_tactil = TouchSensor(Port.S4)
vidas = 5
ev3_img = Image(ImageFile. DECLINE)
while vidas > 0:
    texto_vidas= "Vidas: " + str(vidas)
    ev3.screen.draw_text(30,30,texto_vidas)

    if sensor_tactil.pressed():
        vidas -= 1
        ev3.screen.clear()
        ev3.screen.load_image(Image(ev3_img))
        wait(500)
        ev3.screen.clear()
```

Las capacidades gráficas son limitadas dada la resolución monocromática, pero agregan gran valor en proyectos robóticos al permitir comunicación visual directa. La pantalla LCD combinada con Python expande enormemente las posibilidades del Lego EV3.

4.3.7. Bocina e Indicadores Sonoros del Lego EV3.

El ladrillo programable EV3 incorpora una pequeña bocina y la capacidad de reproducir distintos sonidos y efectos, los cuales resultan muy útiles para dotar al robot de capacidad de comunicación sonora.

4.3.7.1 Especificaciones.

La bocina del EV3 es un transductor electroacústico que convierte señales eléctricas en ondas sonoras (Jordan, J., & Fern, A. , 2019), sus especificaciones son:

- Tipo: bocina dinámica de imán móvil.
- Potencia: 1 W aprox.
- Impedancia: 8 ohms.
- Respuesta de frecuencia: 300-20kHz.
- Voltaje: 5VDC.
- Nivel sonoro: Hasta 85 dB a 10 cm.

Adicionalmente, el firmware del EV3 trae canciones pregrabadas y un sistema de efectos sonoros programables.

Los indicadores sonoros son especialmente útiles cuando la pantalla no es visible o como alertas. Por ejemplo, el EV3 emite distintos tonos para encendido, apagado, conexión bluetooth, entre otros estados.

4.3.7.2 Programación con PyBricks.

La biblioteca PyBricks de Python provee la clase Speaker para control de sonidos:

```
#!/usr/bin/env pybricks-micropython
from pybricks.hubs import EV3Brick
from pybricks.tools import wait
from pybricks.media.ev3dev import SoundFile
from pybricks.media.ev3dev import Image, ImageFile

# Inicializar el EV3
ev3 = EV3Brick()

# Dice "Sonido uno"
ev3.speaker.say("Sonido uno")
```

```
# Reproducir sonido de beep
ev3.speaker.beep()
wait(1000)
# Dice "Sonido dos"
ev3.speaker.say("Sonido dos")
# Reproducir sonido de beep con frecuencia y duración
ev3.speaker.beep(frequency=500, duration=500)

# Dice "Sonido tres"
ev3.speaker.say("Sonido tres")
# Reproduce el sonido de un archivo
ev3.speaker.play_file(SoundFile.SONAR)

ev3.speaker.say("Sonido cuatro")
# Reproduce el sonido de un archivo
ev3.speaker.play_file(SoundFile.DOG_BARK_1)
```

Algunos métodos útiles son:

- .play_file(Archivo_de_sonido) Reproduce el archivo de sonido, algunos sonidos son:
    - SoundFile.AIRBRAKE
    - SoundFile.AIR_RELEASE
    - SoundFile.BACKUP_ALERT
    - SoundFile.BOO
    - SoundFile.BRAKE
    - SoundFile.CHIME
    - SoundFile.CLICK_2
    - SoundFile.DOOR_SLAM
    - SoundFile.DOWN
    - SoundFile.ENGINE_FAST
    - SoundFile.ERROR
    - SoundFile.EV3
- .play_note() Reproduce una nota musical.
- .say() Convierte el texto a voz.
- .set_speech_options() Cambia voz y velocidad.

Usos comunes:

- Reproducir efectos sonoros y alertas.
- Hablar mensajes y texto como interfaz.
- Añadir música y sonido en proyectos y juegos.
- Servir como altavoz para reproducir cualquier sonido.
- Debugging al reproducir códigos morse.

4.3.8. Leds del Lego Mindstorms EV3.

Los leds o diodos emisores de luz son componentes electrónicos que permite emitir luz de diferentes colores de forma simple en el robot Lego EV3. Analicemos las capacidades de estos indicadores.

4.3.8.1. Especificaciones.

El ladrillo EV3 incluye 3 leds que funcionan como luces indicadoras del estado del robot (Jordan, J., & Fern, A. , 2019), Sus especificaciones son:

- Tipo: diodos emisores de luz (LED).
- Colores: rojo, verde, ámbar y azul.
- Corriente: 20mA típica, 30mA máxima.
- Tensión de operación: 4.5V DC aprox.
- Intensidad luminosa: 280-400 mcd típica.
- Ángulo de emisión: 60 grados.
- Tamaño: 5mm de diámetro.

No requieren configuración, están incorporados dentro del ladrillo principal del EV3

4.3.8.2. Programación en PyBricks.

La biblioteca PyBricks de Python provee la clase light para controlar los leds del EV3:

```
#!/usr/bin/env pybricks-micropython
from pybricks.hubs import EV3Brick
from pybricks.parameters import Color
from pybricks.tools import wait

# Inicializar el EV3
ev3 = EV3Brick()

# Encender una luz naranja
ev3.light.on(Color.ORANGE)
# Esperar un 5 segundos
wait(500)
# Apagar la luz
ev3.light.off()
```

El método light.on(color) permite encender la luz led del Brick, donde el parámetro color puede ser RED, ORANGE y RED

EL método light.off() apaga el led.

Usos comunes:

- Luces indicadoras de estado (batería, conexión, etc).
- Indicadores luminosos en proyectos y juegos.
- Árbol de navidad parpadeando leds.
- Señales visuales de operación o alertas.
- Simular semáforo o señalización de tráfico.

Por ejemplo, un patrón de leds en un while:

```
#!/usr/bin/env pybricks-micropython
from pybricks.hubs import EV3Brick
from pybricks.parameters import Color
from pybricks.tools import wait

# Inicializar el EV3
ev3 = EV3Brick()
while True:
    # Encender una luz naranja
    ev3.light.on(Color.ORANGE)
    # Esperar un 5 segundos
    wait(500)
    # Apagar la luz
    ev3.light.off()

    # Encender una luz Roja
    ev3.light.on(Color.RED)
    # Esperar un 5 segundos
    wait(500)
    # Apagar la luz
    ev3.light.off()

    # Encender una luz Verde
    ev3.light.on(Color.GREEN)
    # Esperar un 5 segundos
    wait(500)
    # Apagar la luz
    ev3.light.off()
```

Los leds pueden añadir efectos visuales interesantes de forma sencilla en todo tipo de proyectos con el EV3. Su versatilidad y bajo costo los hacen ideales para indicadores simples.

CAPITULO 5. PROYECTOS PRÁCTICOS

A continuación, se presenta una colección de proyectos prácticos para construir diferentes robots funcionales utilizando el ladrillo programable Lego Mindstorms EV3 y el lenguaje de programación Python con la biblioteca PyBricks.

Los proyectos están pensados para ser abordados de forma incremental, aplicando los conceptos explorados en los capítulos previos para resolver desafíos de robótica y programación cada vez más complejos. Comenzando con robots simples que se mueven en patrones básicos, se avanza gradualmente hacia la incorporación de diferentes sensores, permitiendo que los robots perciban y reaccionen a su entorno, y culminando con un robot móvil autónomo capaz de navegar su entorno sin supervisión.

Cada proyecto se explica en detalle, analizando la lógica implementada y el código necesario, de modo que al finalizar este capítulo el lector estará capacitado para crear y programar sofisticados robots utilizando el Lego EV3.

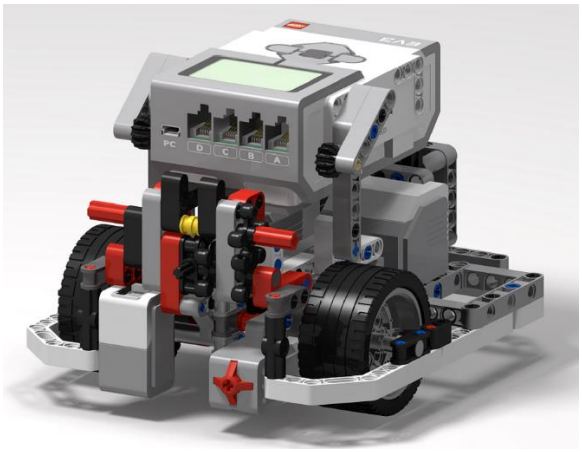

Figura 21. Robot Sumo armado con Lego EV3

5.1. PROGRAMA PARA QUE EL ROBOT HAGA 10 CUADROS.

Este programa en PyBricks hace que un robot Lego EV3 se mueva describiendo un recorrido cuadrado, avanzando ambos motores durante 1 segundo, deteniendo el motor izquierdo y girando el derecho durante 1 segundo para lograr un giro de +/- 90 grados, y repitiendo esta secuencia 10 veces mediante un bucle for, logrando así un movimiento en el que el robot recorre un total de 10 cuadrados gracias a los giros alternados.

```
#!/usr/bin/env pybricks-micropython
from pybricks.hubs import EV3Brick
from pybricks.ev3devices import Motor
from pybricks.parameters import Port
from pybricks.tools import wait
def main():
    # Inicializa el brick
    ev3 = EV3Brick()

    # Inicializa los motores
    left_motor = Motor(Port.A)
    right_motor = Motor(Port.C)

    for i in range(10):
        for j in range(4):
            # Mueve el motor los motores A y C
            # a una velocidad rotacinal 500 grados por segundo
            left_motor.run(500)
            right_motor.run(500)
            # Espera 1000 milisegundos
            wait(1000)
            # Detiene los motores A
            left_motor.hold()
            # Mueve el motor C a una velocidad rotacional
            # de -500 grados por segundo
            right_motor.run(-500)
            # Espera 1000 milisegundos
            wait(1000)
    # Detiene los motores A y C
    left_motor.stop()
    right_motor.stop()
#----------------------------------

if __name__ == "__main__":
   main()
```

5.2. PROGRAMA PARA RECORRIDO EN ESPIRAL.

Este programa en PyBricks hace que un robot Lego EV3 se mueva describiendo una trayectoria en espiral, mediante un bucle for que se repetirá 50 veces, en cada iteración se mueven ambos motores durante un tiempo determinado por la variable time, luego se detiene el motor izquierdo y se invierte el derecho durante 1000 milisegundos logrando un giro, al final del bucle se incrementa time en 200 ms, de esta forma en cada iteración el tiempo de avance es mayor, dando como resultado un movimiento en espiral cuyo radio se va agrandando progresivamente.

```
#!/usr/bin/env pybricks-micropython
from pybricks.hubs import EV3Brick
from pybricks.ev3devices import Motor
from pybricks.parameters import Port
from pybricks.tools import wait

def main():
    # Inicializa el brick
    ev3 = EV3Brick()

    # Inicializa los motores
    left_motor = Motor(Port.A)
    right_motor = Motor(Port.C)

    # Inicializa el tiempo de espera
    time = 200
    for i in range(50):
        # Mueve el motor los motores A y C
        left_motor.run(500)
        right_motor.run(500)
        wait(time)

        left_motor.hold()
        right_motor.run(-500)
        wait(1000)

        # incrementa el tiempo de espera
        time += 200
    # Detiene los motores A y C
    left_motor.stop()
    right_motor.stop()

#----------------------------------

if __name__ == "__main__":
   main()
```

5.3. PROGRAMA DE RECORRIDO ALEATORIO.

Este programa en PyBricks hace que un robot Lego EV3 se mueva de forma aleatoria durante 2 segundos mediante un bucle infinito. En cada iteración, primero hace avanzar a ambos motores juntos durante dos segundos. Luego genera un número aleatorio entre 0 y 9, si es mayor o igual a 5 detiene el motor A y retrocede sólo con el motor C, de lo contrario detiene el motor C y retrocede el motor A. Esto provoca giros y cambios de dirección aleatorios. Finalmente vuelve a detener ambos motores después de un segundo antes de repetir. Este proceso en bucle genera un movimiento irregular impredecible del robot, avanzando, girando y retrocediendo al azar gracias al uso de la función randint() para introducir aleatoriedad en el programa.

```
#!/usr/bin/env pybricks-micropython
from pybricks.hubs import EV3Brick
from pybricks.ev3devices import Motor
from pybricks.parameters import Port
from pybricks.tools import wait, StopWatch
import random

# Generar un número entero aleatorio entre 0 y 9
def aleatorio (n,m):
    numero_aleatorio = random.randint(n, m)
    return numero_aleatorio

#----------------------------------
def main():
    # Inicializa el brick
    ev3 = EV3Brick()

    # Inicializa los motores
    left_motor = Motor(Port.A)
    right_motor = Motor(Port.C)

    # Inicializa el tiempo de espera
    while True:
        # Para los motores A y C
        left_motor.stop()
        right_motor.stop()
        # Mueve el motor los motores A y C
        left_motor.run(500)
        right_motor.run(500)
        # Espera 2000 milisegundos
        wait(2000)
        # Genera un número aleatorio entre 0 y 9
        n = aleatorio(0, 9)
        if (n>=5):
            # Para el motor A
            left_motor.hold()
            # Mueve el motor C a una velocidad rotacional
            right_motor.run(-500)
        else:
            # Para el motor C
            right_motor.hold()
```

```
            # Mueve el motor A con una velocidad rotacional
            left_motor.run(-500)
        # Espera 1000 milisegundos
        wait(1000)
    # Detiene los motores A y C
    left_motor.stop()
    right_motor.stop()

#----------------------------------

if __name__ == "__main__":
    main()
```

5.4. PROGRAMA CON SENSOR DE TACTO.

Este programa en PyBricks utiliza un sensor de tacto en el robot Lego EV3 para detectar colisiones, mediante un bucle infinito se verifica continuamente si el sensor está presionado, en caso positivo invierte el giro de ambos motores durante 2500 milisegundos para retroceder, detiene el motor izquierdo y gira el derecho durante 3000 milisegundos, de esta forma cuando el robot colisiona con un obstáculo reacciona retrocediendo y girando para sortearlo, en caso que el sensor no detecte presión simplemente ambos motores siguen avanzando normalmente.

```
#!/usr/bin/env pybricks-micropython
from pybricks.ev3devices import TouchSensor
from pybricks.ev3devices import Motor
from pybricks.parameters import Port
from pybricks.hubs import EV3Brick
from pybricks.tools import wait


def main()
    # Inicializar el EV3
    ev3 = EV3Brick()

    # Inicializar el sensor de tacto por el puerto 4
    sensor_tactil = TouchSensor(Port.S4)
    # Inicializar los motores en el puerto A y C
    motor_A = Motor(Port.A)
    motor_C = Motor(Port.C)
    while True:
    if sensor_tactil.pressed():
        # Retroceder y girar al chocar
        motor_A.run(-250)
        motor_C.run(-250)
        wait(2500)
        motor_C.stop()
        motor_A.run(-100)
        wait(3000)
    else:
        # Avanzar normalmente
        motor_A.run(250)
```

```
        motor_C.run(250)
#-------------------------------------------------------------

f __name__ == "__main__":
    main()
```

Así el EV3 puede reaccionar al contacto físico. Las aplicaciones son muy variadas dada la versatilidad del sensor táctil para detectar interacción con objetos.

5.5. PROGRAMA CON SENSOR ULTRASONICO.

Este programa en PyBricks utiliza el sensor de ultrasónico en el robot Lego EV3 para detectar objetos que están al frente del robot, mediante un bucle infinito se verifica continuamente la distancia del sensor con el objeto, en caso de que la distancia sea menor 150 milímetros, invierte el giro de ambos motores durante 2500 milisegundos para retroceder, detiene el motor izquierdo y gira el derecho durante 3000 milisegundos, de esta forma cuando el robot colisiona con un obstáculo reacciona retrocediendo y girando para sortearlo, en caso que el sensor no detecte presión simplemente ambos motores siguen avanzando normalmente.

```
#!/usr/bin/env pybricks-micropython
from pybricks.ev3devices import UltrasonicSensor
from pybricks.ev3devices import Motor
from pybricks.parameters import Port
from pybricks.hubs import EV3Brick
from pybricks.tools import wait


def main():
    # Inicializar el EV3
    ev3 = EV3Brick()

    # Inicializar el sensor de Ultrasonico por el puerto 2
    sensor_ultrasonido = UltrasonicSensor(Port.S2)
    # Inicializar los motores en el puerto A y C
    motor_A = Motor(Port.A)
    motor_C = Motor(Port.C)
    while True:
        d= sensor_ultrasonido.distance()
        if (d<150):
            # Retroceder y girar al chocar
            motor_A.run(-250)
            motor_C.run(-250)
            wait(2500)
            motor_C.stop()
            motor_A.run(-100)
            wait(3000)
        else:
            # Avanzar normalmente
```

```
            motor_A.run(250)
            motor_C.run(250)
#-------------------------------------------------------------

if __name__ == "__main__":
   main()
```

5.6. PROGRAMA CON EL SENSOR INFRARROJO.

Este programa en PyBricks implementa un robot móvil Lego EV3 capaz de detectar y esquivar obstáculos utilizando un sensor infrarrojo, mediante un bucle infinito lee continuamente la distancia al objeto más cercano, si esta distancia es menor a 15 centímetros determina que hay un obstáculo próximo por lo que frena el robot y retrocede para sortearlo, en caso contrario si el camino está libre simplemente avanza en línea recta, de esta forma el robot es capaz de navegar un entorno de manera autónoma detectando obstáculos y evitándolos gracias al sensor infrarrojo y el control de motores mediante PyBricks.

```
#!/usr/bin/env pybricks-micropython
from pybricks.ev3devices import InfraredSensor
from pybricks.ev3devices import Motor
from pybricks.parameters import Port
from pybricks.hubs import EV3Brick
from pybricks.tools import wait


def main():
    # Inicializar el EV3
    ev3 = EV3Brick()

    # Inicializar el sensor infrrojo por el puerto 4
    infrarrojo = InfraredSensor(Port.S4)
    # Inicializar los motores en el puerto A y C
    motor_A = Motor(Port.A)
    motor_C = Motor(Port.C)
    while True:
        d= infrarrojo.distance()
        if (d<15):
            # Retroceder y girar al chocar
            motor_A.run(-250)
            motor_C.run(-250)
            wait(2500)
            motor_C.stop()
            motor_A.run(-100)
            wait(3000)
        else:
            # Avanzar normalmente
            motor_A.run(250)
            motor_C.run(250)
```

```
#-------------------------------------------------------------

if __name__ == "__main__":
    main()
```

5.7. PROGRAMA SEGUIDOR DE LÍNEA.

Este programa en PyBricks implementa un seguidor de línea usando un sensor de color en un robot Lego EV3, mediante un bucle infinito se lee continuamente el valor de reflexión del sensor, si es muy bajo significa que detecta una línea negra por lo que gira el robot hacia la izquierda, si el valor es muy alto indica que se salió de la línea y debe girar a la derecha para retomarla, en caso que el valor esté en un rango medio avanza en línea recta, hasta encontrar los valores de reflexión requeridos para tomar de nuevo la línea negra, de esta forma es capaz de seguir el recorrido de una línea negra en el piso guiando el movimiento del robot.

```
#!/usr/bin/env pybricks-micropython
from pybricks.hubs import EV3Brick
from pybricks.ev3devices import Motor, ColorSensor
from pybricks.parameters import Port

def main()
  ev3 = EV3Brick()
  left_motor = Motor(Port.A)
  right_motor = Motor(Port.C)
  color_sensor = ColorSensor(Port.S1)

  while True:
    reflection = color_sensor.reflection()

    if reflection < 30: #Izquierda
      left_motor.run(-100)
      right_motor.run(100)

    elif reflection > 60: #Derecha
      left_motor.run(100)
      right_motor.run(-100)

    else:
      left_motor.run(500)
      right_motor.run(500)

#-------------------------------------------------------------
if __name__ == "__main__":
    main()
```

El seguidor de línea también se puede implementar leyendo el color del sensor, si es negro significa que detecta una línea negra por lo que gira el robot hacia la izquierda, si el color es blanco, indica que se salió de la línea y debe girar a la derecha para retomarla, en caso de que el valor no sea negro o blanco, avanza en línea recta hasta encontrar de nuevo el color negro o blanco.

```
    #!/usr/bin/env pybricks-micropython
    from pybricks.hubs import EV3Brick
    from pybricks.ev3devices import Motor, ColorSensor
    from pybricks.parameters import Port, Color
    def main():
        ev3 = EV3Brick()
        left_motor = Motor(Port.A)
        right_motor = Motor(Port.C)
        color_sensor = ColorSensor(Port.S1)

        while True:
        color = color_sensor.color()

        if color == Color.BLACK: # Izquierda
            left_motor.run(-100)
            right_motor.run(100)

        elif color == Color.WHITE: # Derecha
            left_motor.run(100)
            right_motor.run(-100)

        else:
            left_motor.run(500)
            right_motor.run(500)

    #-------------------------------------------------------------
    if __name__ == "__main__":
main()
```

CAPÍTULO 6: TESTING Y DEPURACIÓN DE PROGRAMAS CON PYBRICKS

El testing o prueba de programas es una parte fundamental del desarrollo de software, este permite verificar que nuestro código funciona correctamente antes de desplegarlo a un entorno real. En robótica esto cobra especial importancia ya que errores en el software pueden hacer que un robot físico presente comportamientos erráticos, inseguros o dañinos.

En este capítulo veremos técnicas y buenas prácticas para probar programas de robots Lego Mindstorms EV3 escritos en Python utilizando la biblioteca PyBricks. También cubriremos los errores más comunes y cómo solucionarlos mediante un proceso llamado depuración o debugging.

6.1 PROBANDO PROGRAMAS

A continuación, se presentan diversas estrategias para probar nuestro código del robot EV3 con PyBricks (McRoberts, 2021):

- Probar segmentos pequeños de código de forma aislada: Probamos cada nueva función o característica por separado antes de integrarlas al programa completo. Esto ayuda a identificar errores más fácilmente.
- Probar con diferentes conjuntos de entradas: Hay que verificar el código con valores de input válidos y no válidos. Por ejemplo, números positivos y negativos, dentro y fuera de rangos límite.
- Probar múltiples flujos de ejecución: Probar cada camino posible en condicionales e iteraciones. Por ejemplo, las cláusulas if-else deben validarse tanto para True como False.
- Simular componentes faltantes: Si no tenemos un sensor físico, podemos simularlo pasando valores predefinidos para probar esa parte del código.

- Pruebas unitarias: Son pequeños fragmentos de código para probar unidades individuales como funciones, de forma aislada y controlada. Ayudan a probar partes específicas mejor.
- Pruebas de integración: Una vez que las unidades individuales funcionan, debemos probar como trabajan en conjunto utilizando pruebas de integración.
- Pruebas de caja negra vs. caja transparente: Las pruebas de caja negra verifican inputs y outputs sin ver los detalles internos. Las pruebas de caja transparente verifican además el flujo interno paso a paso.
- Pruebas de humo: Provocar intencionalmente fallos, como apagar el EV3 durante la ejecución, para verificar la robustez y manejo de errores.
- Pruebas de regresión: Volver a probar características existentes tras agregar nuevas, para detectar efectos colaterales o bugs introducidos.

Buenas prácticas para pruebas efectivas:

- Cubrir todos los casos y flujos posibles, no sólo los normales.
- Tener un buen conjunto de datos de prueba representativos.
- Automatizar pruebas repetitivas cuando sea posible.
- Documentar los casos y resultados de prueba.
- Probar frecuentemente a lo largo del ciclo de desarrollo, no sólo al final.
- Que otra persona pruebe el código para encontrar errores nuevos.

6.2 ERRORES COMUNES Y SOLUCIONES.

Algunos errores y excepciones comunes en programas de EV3 con PyBricks son (Sweigart, 2022):

- Errores de sintaxis: Como errores de tipado, mala indentación, puntuación errónea, causan que el programa no se ejecute. Revisar cuidadosamente el código y los mensajes de error para detectarlos.
- Nombres indefinidos: Usar variables, funciones o módulos que no han sido definidos aún. Puede ayudar nombrar de forma explicativa.
- Tipos de datos incorrectos: Por ejemplo, tratar de sumar un string y un número. Hay que revisar que los tipos coincidan en operaciones.
- Accesos inválidos: Tratar de acceder un elemento fuera del rango de un arreglo o una lista por ejemplo. Revisar índices y rangos.

- Iteraciones infinitas: Tener cuidado con loops infinitos que pueden congelar el programa. Probar límites y condiciones de salida.
- Errores lógicos: El programa funciona, pero no hace lo que debería. Revisar la lógica de decisiones y secuencias. Depurar con prints.
- Errores de integración: Piezas individuales funcionan, pero fallan al combinarlas. Probar exhaustivamente la integración de componentes.
- Excepciones: Errores que interrumpen el flujo como al dividir por cero. Manejar excepciones con try-except donde sea probable que ocurran.
- Problemas de concurrencia: Cuando dos procesos o threads acceden a un recurso compartido simultáneamente. Utilizar exclusiones mutuas.
- Errores aleatorios o intermitentes: Los más difíciles de diagnosticar. Intentar reproducirlos y aislar casos para entender las causas.

Para identificar y resolver estos y otros errores, podemos aplicar técnicas de depuración:

- Leer detenidamente los mensajes de error y excepciones para entender dónde se origina el problema.
- Usar prints para mostrar valores de variables en diferentes puntos y ver dónde se desvía el flujo.
- Revisar en detalle los registros, traces y dumps que provee el sistema operativo del EV3.
- Utilizar un depurador para ejecutar paso a paso viendo el estado del programa. Visual studio code integra uno.
- Aislar partes del código y datos para replicar el error en un caso mínimo.
- Validar supuestos y teorías de donde puede estar el error sistemáticamente.
- Repasar código recientemente agregado o modificado donde puede originarse el bug nuevamente introducido.
- Consultar con otros programadores para un análisis de código crítico.
- Como último recurso, reescribir partes del código desde cero o reflexionar sobre el diseño.

Un proceso metódico y constante de pruebas ayuda a detectar y corregir errores tempranamente. La depuración efectiva requiere de una mente crítica y creatividad para resolver los inevitables bugs de la programación. Estas habilidades fundamentales se desarrollan con la experiencia y son esenciales para todo programador profesional.

CONCLUSIONES

Python es un excelente lenguaje para iniciarse en robótica dada su curva de aprendizaje suave y la gran cantidad de bibliotecas especializadas disponibles. Permite desarrollar comportamientos sofisticados en robots de manera simple e interactiva, resultando ideal para propósitos educativos. La posibilidad de controlar fácilmente hardware como el EV3 abre un mundo de posibilidades creativas.

El ladrillo programable Lego EV3 proporciona una plataforma de hardware versátil y de alta calidad para montar y dar vida a distintos modelos de robots. Su sistema modular con motores, sensores y componentes interconectables mediante programación permite construir desde robots simples hasta muy complejos según el nivel de habilidad.

La biblioteca PyBricks de Python brinda una API limpia e intuitiva para controlar el EV3 de forma remota desde la PC sin necesidad de modificar el sistema operativo del robot. Su sintaxis sencilla y enfocada a principiantes la hace ideal para introducir la programación de robots con Python de manera rápida y práctica.

Los proyectos presentados proveen ejemplos incrementalmente más complejos que permiten aplicar los conceptos explorados para dotar a un robot EV3 de comportamientos autónomos como seguir líneas, evitar obstáculos y navegar entornos basándose en lecturas de sensores mediante el código Python desarrollado.

Las buenas prácticas de testing y depuraciones cubiertas son esenciales para verificar el correcto funcionamiento de los programas y corregir errores. Probar exhaustivamente diferentes casos y manejar excepciones de forma adecuada es indispensable en el desarrollo de software robótico confiable.

GLOSARIO

- **API**: Interfaz de programación de aplicaciones que expone funciones de un sistema.
- **ARM:** Arquitectura de CPU utilizada en el EV3 y otros sistemas embebidos.
- **Assert:** Función para verificar resultados esperados en pruebas.
- **Biblioteca:** Colección de funciones y clases reutilizables.
- **Brick:** Otro nombre para el ladrillo EV3.
- **Debugging**: Proceso de encontrar y arreglar defectos en programas.
- **Dumps:** Tipos de datos del sistema operativo que contienen una copia de la memoria del kernel del sistema operativo del robot.
- **Embedded**: Sistemas computarizados dedicados a una tarea específica integrados en un dispositivo.
- **EV3:** Ladrillo programable que hace de computadora del robot Lego Mindstorms EV3.
- **Excepción:** Error detectado durante ejecución que altera flujo normal.
- **Firmware:** Software que controla los componentes electrónicos de un dispositivo.
- **Frameworks**: Conjunto de herramientas y librerías para desarrollar aplicaciones.
- **GPIO**: Pines de propósito general para conectar sensores y actuadores.
- **Interpretado**: Lenguajes que se ejecutan línea por línea sin compilación previa.
- **LCD**: Pantalla de cristal líquido para mostrar información visual.
- **Motor:** Actuador que provee movimiento y trabajo mecánico al robot.
- **Multiparadigma**: Lenguaje que soporta varios estilos de programación.
- **Packages:** Módulos que agrupan bibliotecas relacionadas.
- **Pip:** Herramienta para instalar y administrar paquetes Python.
- **PWM**: Modulación por ancho de pulso para controlar potencia en motores.

- **PyBricks:** Biblioteca de Python para controlar el EV3 de forma remota.
- **PyPI:** Repositorio de paquetes Python para instalar bibliotecas.
- **RAM**: Memoria de acceso aleatorio volátil para almacenar datos en uso.
- **Robot:** Máquina programable capaz de realizar tareas físicas de forma autónoma.
- **Sensor:** Dispositivo que permite al robot detectar el entorno.
- **SPI**: Bus de comunicación para transferir datos entre componentes.
- **Struct**: Estructuras de datos personalizadas compuestas por varios tipos.
- **Testing**: Realizar pruebas sistemáticas para verificar comportamiento de código.
- **Thread**: Segmento de ejecución concurrente e independiente.
- **Tipado dinámico:** Tipo de variable se define en tiempo de ejecución, no explícitamente.
- **Traces:** Tipo de datos del sistema operativo que registra eventos que ocurren en el kernel del sistema operativo del robot.
- **Try/except:** Manejar excepciones ejecutando código alternativo.

ANEXO A. BUENAS PRÁCTICAS DE PROGRAMACIÓN

La programación es una actividad creativa y altamente técnica que presenta muchos desafíos. Como todo proceso de construcción, requiere de una serie de principios rectores que garanticen resultados óptimos, eficientes y confiables. En este apéndice exploraremos algunas de las mejores prácticas establecidas por expertos para lograr un código de calidad.

Si bien no existen reglas absolutas, adoptar las siguientes recomendaciones en la medida de lo posible puede marcar una gran diferencia en la facilidad de desarrollo, mantenimiento y confiabilidad nuestros programas (McConnell, 2004) (Hunt, 1999). Después de todo, ¡la única regla inquebrantable es que funcionen!

Formateo consistente de código

Un código con formato limpio, ordenado y estandarizado facilita su lectura y comprensión significativamente (Stellman, 2020). Esto aplica tanto para nosotros mismos como para otros programadores que deban leerlo. Un estilo de codificación coherente nos ayuda a reconocer patrones rápidamente.

Algunas directrices para lograr esto:

• Ser consistentes en el uso de sangrías e indentaciones para delimitar bloques y métodos. La mayoría de los lenguajes como Python usan 4 espacios por tabulación.

• Limitar las líneas de código a una longitud máxima recomendada de 79 caracteres para evitar desplazamiento horizontal (Wilson Giese, 2019).

• Separar bloques de código relacionados verticalmente con líneas en blanco.

• Incluir comentarios relevantes y omitir los innecesarios.

Un buen formateo visual reduce la carga cognitiva para comprender qué está ocurriendo en el código. Esto es crítico en secciones complejas. Siempre podemos complementar con comentarios para explicar la lógica detrás.

Programación modular

La modularidad consiste en dividir un programa en partes más pequeñas, independientes y manejables, conocidas como módulos o componentes. Cada módulo se especializa en una tarea específica. Esta técnica presenta grandes beneficios (McLaughlin, 2006):

• Reutilización: Los módulos pueden emplearse fácilmente en otros proyectos.

• Organización: Permite focalizarse en porciones pequeñas aisladas.

• Mantenimiento: Es más sencillo realizar cambios en una pieza independiente.

En Python es común crear módulos separando funcionalidades en diferentes archivos o carpetas. También se aplican conceptos de Programación Orientada a Objetos mediante clases. Idealmente cada clase o módulo debe ser autocontenido y realizar una única función de negocio, la modularidad promueve código más flexible y gestionable.

Simplicidad y legibilidad

La simplicidad en la programación es una meta que todos perseguimos. Un código simple tiende a minimizar la complejidad innecesaria lo máximo posible (Martin, 2008). Esto hace que sea más fácil de leer y entender para humanos, una característica invaluable.

Podemos pensar en la simplicidad en varias dimensiones:

- Minimizar duplicación mediante funciones y módulos reutilizables.
- Evitar código denso y difícil de comprender. Siempre que sea posible, es preferible dividir en pedazos más pequeños.
- Limitar anidamientos profundos de estructuras de control como bucles y condicionales.
- Descomponer problemas en trozos manejables directos en lugar de soluciones complejas.

La legibilidad se relaciona con qué sencillo es para otra persona entender nuestro código (Gorman, 2015). Un código ilegible genera una gran fricción cada vez que necesitamos modificarlo o corregir errores.

Algunas sugerencias:

• Usar variables y nombres significativos que den pistas sobre su utilidad y contenido.

• Preferir nombres largos y completos vs. abreviaturas crípticas.

• Añadir comentarios relevantes para complementar el código.

La simplificación y legibilidad pueden conseguirse gradualmente refactorizando código existente o prestando especial atención en los nuevos desarrollos.

Documentación efectiva

La documentación efectiva es esencial para entender un programa a profundidad, tanto para nosotros en el futuro como para otros potenciales colaboradores (Krile, 2006). Incluir artefactos explicativos relevantes es una cortesía que ahorra muchas horas de investigación.

Tipo de documentos útiles:

• Comentarios en el código mismo para partes clave o complejas.

• Documentación técnica con diagramas de clases y módulos.

• Documentación de la API para usuarios finales (ReadMe, docstrings).

• Ejemplos de código y tutoriales paso a paso (Microsoft, Comment your code, 2022a).

La documentación más valiosa es la que se mantiene actualizada. Una estrategia es crearla junto al código durante el desarrollo. También debemos evitar sobre-documentar y preferir calidad sobre cantidad (Brown, 2011). Comentarios irrelevantes contribuyen al ruido.

Control de versiones

El control de versiones permite registrar todos los cambios realizados en nuestro código a lo largo del tiempo para poder recuperar versiones previas fácilmente. Entre las herramientas más populares se encuentran Git, Mercurial y Subversion (Loeliger, 2012).

Algunas ventajas del control de versiones:

- Conserva un historial completo de todas las revisiones y contribuciones de un proyecto a lo largo de su desarrollo.
- Permite trabajar simultáneamente entre múltiples miembros de un equipo con integración de cambios centralizada.
- Habilita mantener diversas ramas o variantes experimentales del código en forma paralela.
- Simplifica en gran medida la fusión y consolidación de mejoras entre ramas.

Una práctica recomendada es utilizar sistemas de control de versiones incluso en proyectos personales para llevar registro de los avances (Stojanov, 2022). Las confirmaciones o commits deben ser frecuentes e incluir mensajes descriptivos relevantes de los cambios.

Manejo de errores y excepciones

Es casi inevitable que ocurran errores tanto por fallas del programa como por entradas inválidas del usuario. Prever estrategias apropiadas de manejo de errores es indispensable para desarrollar aplicaciones robustas frente a situaciones anómalas (Subramaniam, 2014).

En idiomas como Python disponemos de bloques try/catch para interceptar excepciones y prevenir fallas catastróficas:

```
try:
    operacion_riesgosa()
except SpecificException:
   # Tomar medidas de recuperación
finally:
   # Asegura de que el programa se detenga
   # independientemente de si ocurrió la excepción o no
```

Ejemplo:

```
#!/usr/bin/env pybricks-micropython
from pybricks.hubs import EV3Brick
from pybricks.ev3devices import Motor
from pybricks.parameters import Port
from pybricks.tools import wait
```

```
def main():
    # Inicializa el brick
    ev3 = EV3Brick()

    # Inicializa los motores
    left_motor = Motor(Port.A)
    right_motor = Motor(Port.C)

    for i in range(1):
        for j in range(4):
            # Mueve el motor los motores A y C
            # a una velocidad rotacinal 500 grados por
            #segundo
            left_motor.run(500)
            right_motor.run(500)
            # Espera 1000 milisegundos
            wait(1000)
            # Detiene los motores A
            left_motor.hold()
            # Mueve el motor C a una velocidad rotacional
            # de -500 grados por segundo
            right_motor.run(-500)
            # Espera 850 milisegundos
            wait(850)
    # Detiene los motores A y C
    left_motor.stop()
    right_motor.stop()
#---------------------------------

if __name__ == "__main__":
   try:
       main()
   except OSError as e:
        # Manejo de errores relacionados con la conexión del
        # hardware
        texto = "Error en la conexión del hardware:" + str(e)
        ev3.screen.print(texto)
        wait(1000)

   except Exception as e:
        # Manejo de cualquier otro tipo de error
        texto = "Ha ocurrido un error:" + str(e)
        ev3.screen.print(texto)
        wait(1000)
   finally:
        # Asegurarse de que el robot se detenga
        left_motor.stop()
        right_motor.stop()
        ev3.screen.print("Programa finalizado")
        wait(1000)
```

Otras buenas prácticas:

• Arrojar excepciones propias para forzar manejo cuando ocurren casos no permitidos.

• Registrar trazas de errores en logs para investigar after the fact.

• Mostrar mensajes claros al usuario final cuando errores recuperables.

• Utilizar mecanismos como aserciones para Validar precondiciones y rango de variables.

A mayor robustez en el manejo de errores, mayor confiabilidad tendrá el sistema (Venners, 2003) . Planificar posibles puntos de quiebre y casos borde ayuda a mitigarlos.

Otras recomendaciones

Adicionalmente existen otras prácticas que todo programador debe incorporar en su caja de herramientas:

• Realizar pruebas unitarias para aislar y probar componentes específicos.

• Efectuar validaciones estrictas de entradas de usuario.

• Inspeccionar código mediante revisiones formales e informales entre colegas.

• Probar exhaustivamente en entornos simulados antes de liberar a un sistema real.

• Configurar registros o logs relevantes para trazabilidad y depuración en producción.

• Monitorear desempeño mediante métricas y analítica para mejora continua.

• Investigar sobre patrones de diseño y arquitecturas óptimas antes de desarrollar.

En conclusión, Si bien no existen fórmulas precisas para crear códigos perfectos desde la primera vez, incorporar buenas prácticas ayuda a mitigar muchos dolores de cabeza tanto para nosotros como para nuestros usuarios y otros desarrolladores. Mediante la experiencia también descubriremos nuestros propios estilos, trucos y estándares. No existe receta ideal; lo importante es construir sistemas que funcionen y permitan mejorarlos continuamente. Recuerda que ¡Nunca se deja de aprender como programador!

ANEXO B. REGLAS DE NOMENCLATURA PARA NOMBRE DE IDENTIFICADORES

A la hora de programar, uno de los retos frecuentes es decidir qué nombres asignar a los diferentes identificadores que utilizamos, como variables, funciones, clases, módulos y otros elementos. Definir una estrategia clara y consistente de nomenclaturas tiene gran impacto en la legibilidad, mantenimiento y calidad de nuestro código.

En este anexo exploramos algunas convenciones recomendadas por la comunidad para escoger los nombres de identificadores de forma óptima en diversos lenguajes de programación.

Longitud de los identificadores

Un primer debate recurrente es si utilizar nombres largos o cortos para nuestras variables y componentes (AurumByte, 2017) (Utsav, 2021). Existen diferencias marcadas según lenguajes:

- En Python, Ruby y Go prefieren nombres completos no abreviados para una máxima claridad.
- C# y Java también favorecen nombres explícitos pero suelen ser más concisos.
- En C y C++ históricamente usaban identificadores muy cortos para ahorrar digitación.

En términos generales, se recomienda encontrar un balance entre concisión y significado pleno. Los límites de longitud presentan un tope natural para forzar a condensar más.

Convenciones según tipo de identificador

Las convenciones de nomenclatura también pueden variar según la categoría:

- Variables y funciones: nombres en minúscula separados por guiones bajos. Ej: dias_de_pago, obtener_totales().
- Constantes: mayúsculas separadas también por guiones. Ej: MAX_USERS, COLOR_ROJO.
- Clases: primer letra mayúscula de cada palabra (UpperCamelCase). Ej: FacturaDeVenta, ClientePreferencial.
- Módulos: igual que clases pero sin espacios o guiones. Ej: PrincipalAnalytics, RoboticVisionModule.
- Métodos y propiedades: primer verbo/sustantivo en minúscula luego UpperCamelCase.
- Ej: obtener_saldo(), nombreCompleto. (Gómez, 2020) (Microsoft, Naming Guidelines, 2020b)

Nombres significativos

Más allá de las convenciones sintácticas, es crucial que los nombres aporten valor semántico representando los datos o conducta del identificador lo más precisamente posible dentro del dominio de la aplicación.

Unas recomendaciones para lograrlo son:

- Usar nombres completos en lugar de abreviaciones crípticas.
- Respetar consistentemente palabras y conceptos específicos del dominio.
- Para variables booleanas que indican estado, prefijar con “is”, “has”, “can", etc. Ej: isAuthorized, hasPermisosAdmin.
- Incluir unidades, tipos y propósito en los nombres de variables numéricas.
- Preferir nombres activos de acciones para métodos (get, calculate, print). (Python, 2020). (Microsoft, Naming Guidelines, 2020b)

En resumen, definir reglas uniformes de nomenclatura como las presentadas facilita la lectura fluida del código y reduce la carga cognitiva asociada para cualquiera que deba comprenderlo. Forma parte de las mejores prácticas de un programador profesional.

ANEXO C. GENERACIÓN DE DOCUMENTACIÓN CON DOXYGEN

Doxygen es una popular herramienta open source para generar documentación a partir de código fuente de forma automática, manteniendo la documentación sincronizada (van Heesch, s.f.). Permite crear manuales técnicos, documentos de APIs y gráficos de relaciones de clases ahorrando mucho tiempo. Admite varios lenguajes como C++, Python, Java, y es altamente personalizable. Veamos cómo aprovechar Doxygen:

Agregar comentarios especiales

Para que Doxygen procese nuestro código, debemos insertar marcadores de documentación como:

```
/**
* @brief Breve descripción
*
* Explicación detallada
* que abarca
* múltiples líneas
*/
void funcion_ejemplo()
{
     //Sentencias
}
```

Esto generará descripciones y detalles de funciones y estructuras.

Otros marcadores útiles son:

- @brief Breve descripción de la subrutina
- @param - Detalla parámetros de funciones
- @return - Documenta valores retornados
- @file - Descripción de archivo fuente

Ejemplos de documentación de código usando Doxygen

1. Función para sumar dos números de C/cpp

```
/**
* @brief Suma dos números enteros
*
* @param num1 Primer sumando
* @param num2 Segundo sumando
* @return Resultado de la suma
*/
int sumar(int num1, int num2)
{
  return num1 + num2;
}
```

2. Método para validar credenciales en java

```
/**
* Comprueba si las credenciales de acceso son válidas
*
* @param usuario Nombre de usuario
* @param clave Clave de acceso
* @throws ExcepcionAcceso en caso de datos inválidos
* @return Verdadero si las credenciales coinciden
*/
bool validarCredenciales(String usuario, String clave)
{
   // Código de comparación
}
```

2. Función recursiva de Fibonacci en python

```
""" Calcula números de Fibonacci
    Utiliza recursión para calcular cualquier
    término de la secuencia de Fibonacci.
    @param n Posición deseada
    @returns Valor del término solicitado
"""
```

```
def fibonacci(n):
    # Casos base
    # Llamada recursiva
```

4. Método para serializar objeto a JSON

```
/// <summary>
/// Convierte el objetoactualauna representación JSON
/// </summary>
/// <returns>CadenaJSONdel objeto</returns>
```

5. Función para calcular raíz cuadrada

```
// RaizCuadrada obtiene lar aiz cuadrada de un número
//
// @param: Un número flotante mayor oigual a 0
// @return: Raíz cuadrada del número
```

Generar archivos de configuración

Doxygen lee un archivo de configuración para customizar el proceso de documentación y el formato de salida (Spinellis, 2022). Un ejemplo:

```
OUTPUT_DIRECTORY = Docs
FILE_PATTERNS = *.c *.h *.py
RECURSIVE = YES

GENERATE_HTML = YES
GENERATE_LATEX = NO
```

Esto generaría documentación HTML en formato web de los archivos fuente indicados. Doxygen posee decenas de opciones de personalización para tipos de gráficos, formatos de salida, lenguajes y muchas más opciones.

Ejecutar para generar documentación

Una vez insertados marcadores y definido el archivo de configuración, se ejecuta Doxygen especificando dicho archivo:

```
doxygen Doxyfile
```

Esto procesa todos los archivos y carpetas produciendo documentación en el formato deseado como HTML, Docbook, LaTeX, RTF o XML (Kuzmin, 2022).

La estructura de salida depende de cada proyecto, pero típicamente indexa por carpetas, clases y espacios de nombres facilitando la navegación al usuario.

Integración en pipelines CI/CD

Idealmente, debemos incorporar la generación de documentación de Doxygen dentro de la canalización de integración y entrega continua de nuestra aplicación (Ferguson, 2022).

De esta forma, cada commit o merge request que actualice código fuente regenerará automáticamente la documentación resultante. Esto asegura que los manuales permanezcan siempre actualizados.

Plataformas como GitLab, CircleCI y Jenkins permiten configurar jobs que ejecuten Doxygen como parte de sus pipelines manteniendo la documentación sincronizada.

En conclusión, Doxygen es una excelente herramienta para automatizar la creación de documentos técnicos y del API a partir del código fuente anotado con sus marcadores especiales. Promueve que la documentación se mantenga siempre actualizada como parte del ciclo de vida de desarrollo.

REFERENCIAS

AurumByte. (2017). *Naming Conventions in Programming Languages*. Retrieved from https://www.aurumbyte.com/naming-convention-types-in-programming-languages/

Berry, J. K. (2012). Lego Mindstorms programmable robotics kits for education. International Journal of Information and Education Technology.

Bingi, S. K. (2020). *Applications of Python for robotics: A comprehensive survey. Journal of Systems Architecture*. Retrieved from https://doi.org/10.1016/j.sysarc.2020.101853

Bradski, G. &. (2008). In *Learning OpenCV: Computer vision with the OpenCV library. O'Reilly Media, Inc.*

Brown, W. J. (2011). *Does Code Documentation Save Money? A Controlled Experiment with Professional Developers. Center for Advanced Studies in Software Engineering (CASE). .*

Cozmo. (2013, Agosto 15). *Lego Mindstorms EV3 - game changer or not?* Retrieved from Forbes: https://www.forbes.com/sites/moorinsights/2013/08/15/lego-mindstorms-ev3-game-changer-or-not/

Downey, A. (2015). In *Think Python: How to think like a computer scientist (2nd ed.).* O'Reilly Media.

Downey, A. (2016). In *The importance of Python programming skills. In A. Downey (Ed.), Think Python.* O'Reilly Media.

EV3Dev. (2022). *EV3Dev: Python for EV3*. Retrieved from https://www.ev3dev.org/docs/programming-languages/python/

Ferguson, C. (2022). *Best Documentation Practices with Doxygen and GitHub Pages*. Retrieved from https://embeddedartistry.com/blog/2022/01/05/best-documentation-practices-with-doxygen-and-github-pages/

Gerasimova, V. I. (2020). In *LEGO Mindstorms EV3 Robotic Sets in STEM Education of Schoolchildren. Lecture Notes in Networks and Systems,* (pp. 88, 420-427).

Gindrat, A. D. (2014). In *Lego Mindstorms EV3: building and programming a complex robot. Journal of Computing Sciences in Colleges* (pp. 150-151).

Gindrat, A. D. (2014). *Lego Mindstorms EV3: building and programming a complex robot. Journal of Computing Sciences in Colleges.*

Gómez, M. J. (2020). *Python naming conventions: rules and good practices*. Retrieved from https://peps.python.org/pep-0008/#function-and-variable-names

Gorman, M. M. (2015). *Python by Example.* Packt Publishing Ltd.

Gouaillier, D. H. (2009). In *Mechatronic design of NAO humanoid. In 2009 IEEE International Conference on Robotics and Automation* (pp. 769-774). IEEE.

Gouws, D. B. (2013). In *Computational Thinking in Educational Activities Using LEGO Mindstorms. In Southern African Computer Lecturers' Association Conference SACLA 2013* (pp. 261-269). ACM.

Hunt, A. T. (1999). In *The Pragmatic Programmer(1999).* Addison-Wesley.

Jordan, J., & Fern, A. . (2019). *Lego Mindstorms EV3 Python programming full tutorial*. Retrieved from https://pythonforkids.co.uk/ev3/

Krile, T. (2006). *The Craft of Software Documentation Writing in a Lean Development Group. .* Agile Conference.

Kuzmin, A. (2022). *Doxygen documentation for C/C++*. Retrieved from https://devconnected.com/doxygen-documentation-for-c-c++/

Lego. (2013). User Guide: Lego Mindstorms EV3 (User guide). In T. L. Group.

Lego. (2019). *EV3 MicroPython*. Retrieved from https://lego.github.io/EV3MicroPython/

Lego. (2020). *LEGO® MINDSTORMS®*. Retrieved from https://www.lego.com/en-us/themes/mindstorms/about

Lego. (2021). *Explore the Lego Mindstorms Inventions.* Retrieved from https://www.lego.com/en-us/themes/mindstorms/explore

LEGO® MINDSTORMS® Education EV3. (2022). *Program in Python with EV3*. Retrieved from https://education.lego.com/en-us/product-resources/mindstorms-ev3/teacher-resources/python-for-ev3/

Loeliger, J. M. (2012). *Version control with Git.* O'Reilly Media, Inc.

Lott, M. (2022). In *Object-oriented programming in Python. In Beginning Python* (pp. 337-337). John Wiley & Sons.

Lutz, M. (2013). In *Learning Python (5th ed.).* O'Reilly Media.

McConnell, S. (2004). In C. complete. Pearson Education.

McLaughlin, B. D. (2006). *Head First Object-Oriented Analysis and Design.* O'Reilly Media.

McRoberts, M. (2021). In *Beginning LEGO MINDSTORMS EV3.* Apress.

Microsoft. (2020b). *Naming Guidelines*. Retrieved from https://learn.microsoft.com/en-us/dotnet/standard/design-guidelines/naming-guidelines

Microsoft. (2022a). *Comment your code*. Retrieved from https://docs.microsoft.com/en-us/learn/modules/python-comments-documenting-code/

Microsoft. (2020). *Visual Studio Code*. Retrieved from https://code.visualstudio.com/

Monk, S. (2020). In *Programming Lego EV3 My Blocks.* Apress.

Niemelä, R. (2017). In *obotic Process Automation with Python and Robot Framework. In 2017 European Conference on Software Architecture Workshops (ECSAW)* (pp. 21-22). IEEE.

PyBricks. (2022). *PyBricks - Python for LEGO.* Retrieved from https://pybricks.com

Python. (2020). *Style Guide for Python Code*. Retrieved from https://peps.python.org/pep-0008/#prescriptive-naming-conventions

Python Software Foundation. (2022). *About Python*. Retrieved from https://www.python.org/about/

Quigley, M. C. (2009). In *ROS: an open-source Robot Operating System. In ICRA workshop on open source software (Vol. 3, No. 3.2, p. 5).*

Ribeiro, M. &. (2020). In *Beginning robotics programming in Python with Raspberry Pi and LEGO Mindstorms. Apress.*

Richardson, M. &. (2020). In *Getting started with raspberry Pi Pico and CircuitPython. Maker Media, Inc.*

Spinellis, D. (2022). *Documenting Source Code with Doxygen*. Retrieved from https://www.spinellis.gr/blog/20220213/

Stellman, A. G. (2020). In *Head First Coding.* O'Reilly Media.

Stern, C. &. (2017). In *Education, An open-source Python and Arduino learning tool for educational robotics. International Journal of Technology and Design Education* (pp. 521-539).

Stojanov, A. (2022). *Best Practices for Version Control*. Retrieved from https://www.perforce.com/blog/vcs/best-practices-version-control

Subramaniam, V. (2014). *Programming Groovy: Dynamic Productivity for the Java Developer .* The Pragmatic Bookshelf.

Sweigart, A. (2022). In *Python basics. In Automate the Boring Stuff with Python (2nd ed.)* (pp. 1-15).

Utsav, S. U. (2021). *Naming Convention Guide for Programmers*. Retrieved from https://towardsdatascience.com/a-naming-convention-guide-for-programmers-ec259d80df81

van Heesch, D. (n.d.). *Doxygen Manual: Main Page*. Retrieved from https://www.doxygen.nl/manual/index.html

Venners, B. (2003). *The Making of Java.* McGraw-Hill.

Wilson Giese, A. M. (2019). *Clean Code Concepts in Python.* Retrieved from https://www.section.io/engineering-education/clean-code-python/