TECNOLÓGICO NACIONAL DE MÉXICO®

RoboKit

MANUAL TÉCNICO DEL LENGUAJE DE PROGRAMACIÓN

MAESTRA:

SONIA ALVARADO MARES

PRESENTADO POR:

ARMANDO JAIR IBAÑEZ PARRA

SERGIO ALEJANDRO GUZMÁN ÁLVAREZ

KEVIN AZIEL MANJARREZ MANZANO

Indice.

1. Introducción.
Propósito del Manual de RoboKit.

Propósito:

El manual de RoboKit tiene como objetivo proporcionar a los usuarios una guía completa y accesible para la comprensión, programación y utilización efectiva de la plataforma RoboKit en el contexto de la robótica educacional y recreativa. Este manual tiene la intención de servir como recurso integral que abarque desde los conceptos básicos hasta las aplicaciones más avanzadas, facilitando así el aprendizaje y la experimentación con la plataforma RoboKit.

Destinatarios:

El manual está especialmente diseñado para el siguiente público objetivo:

Estudiantes y Educadores de Robótica:

Dirigido a estudiantes de diversos niveles educativos y a educadores que deseen utilizar RoboKit como una herramienta efectiva para enseñar y aprender robótica de manera práctica y lúdica.

Entusiastas de la Robótica y Principiantes:

Orientado a aquellos que tienen interés en ingresar al mundo de la robótica sin la necesidad de conocimientos previos en electrónica o programación avanzada.

Desarrolladores y Programadores Aficionados:

Adecuado para desarrolladores y programadores que buscan explorar y ampliar sus habilidades en programación robótica utilizando una plataforma accesible y de bajo costo como RoboKit.

2. Historia y Contexto de RoboKit.

Breve Historia:

RoboKit surge como respuesta a la necesidad de una plataforma de programación accesible y educativa en el campo de la robótica. Su desarrollo se inició con la visión de democratizar el aprendizaje de la robótica, haciendo que este fascinante campo sea más accesible para estudiantes, educadores y entusiastas.

Durante el desarrollo inicial, se identificó una brecha en las opciones existentes en el mercado, donde las tecnologías disponibles eran demasiado costosas o requerían conocimientos técnicos avanzados para su utilización. RoboKit nace como una solución integral, asequible y diseñada específicamente para facilitar la programación de robots de manera intuitiva y divertida.

Contexto del Lenguaje de Programación:

Enfocado en la Educación y Recreación: RoboKit se desarrolló con un enfoque claro en la educación y la recreación. A diferencia de otros lenguajes de programación más complejos, RoboKit se diseñó para proporcionar una curva de aprendizaje suave y brindar una experiencia práctica y emocionante en el desarrollo de habilidades en programación y electrónica.

Simplicidad como Principio Fundamental: Desde sus inicios, el equipo de desarrollo de RoboKit ha mantenido la simplicidad como uno de los principios fundamentales del lenguaje. Esto se refleja en la sintaxis clara, las estructuras intuitivas y la facilidad de uso del IDE, todo diseñado para hacer que la programación robótica sea accesible para todos.

Enfoque en la Versatilidad: A lo largo de su evolución, RoboKit ha buscado la versatilidad, permitiendo a los usuarios crear una amplia variedad de proyectos robóticos, desde simples hasta aquellos que involucran un grado de complejidad. La plataforma se ha centrado en la capacidad de generalización, abordando aplicaciones diversas dentro del campo de la robótica.

Compromiso con la Ortogonalidad: La búsqueda de la ortogonalidad en RoboKit ha sido constante, permitiendo a los usuarios realizar operaciones de manera simple y eficiente. Este enfoque contribuye a la legibilidad del código y a la reducción de la complejidad innecesaria.

En conjunto, la historia de RoboKit se enmarca en la evolución de las tecnologías de programación destinadas a democratizar el acceso a la robótica y fomentar el aprendizaje práctico en este emocionante campo. Desde sus inicios, RoboKit ha mantenido su compromiso con la accesibilidad, la simplicidad y la versatilidad, convirtiéndose en una herramienta valiosa para aquellos que desean explorar el mundo de la robótica de manera educativa y recreativa.

3. Requisitos del Sistema para RoboKit.

Para ejecutar programas escritos en el lenguaje RoboKit, es necesario tener en cuenta los siguientes requisitos del sistema. Estos requisitos están diseñados para garantizar una experiencia de programación fluida y eficiente, así como la correcta ejecución de los programas en entornos específicos.

Requisitos de Software:

Sistema Operativo:

RoboKit es compatible con los sistemas operativos Windows, macOS y Linux.

Java Runtime Environment (JRE):

La instalación de Java Runtime Environment (JRE) es imprescindible para ejecutar el entorno de desarrollo integrado (IDE) de RoboKit. Se recomienda la versión 8 o superior de JRE.

Entorno de Desarrollo Integrado (IDE) RoboKit:

Debe instalarse el IDE RoboKit proporcionado por el equipo de desarrollo para la creación y edición de programas.

Requisitos de Hardware:

Computadora Personal:

Se requiere una computadora personal con al menos las especificaciones mínimas para ejecutar el sistema operativo correspondiente y el entorno de desarrollo RoboKit de manera eficiente.

Conexión USB:

Para cargar y ejecutar programas en hardware RoboKit, es necesario disponer de al menos un puerto USB en la computadora para conectar el dispositivo principal (Block).

Hardware RoboKit (Block):

Para ejecutar programas en hardware, se necesita el dispositivo principal de RoboKit (Block), que controla los componentes electrónicos y ejecuta los programas previamente programados.

Microcontrolador PIC16F877:

Se debe disponer del microcontrolador PIC16F877 para la programación y ejecución de los programas en el hardware RoboKit.

Componentes Electrónicos:

Interface para SERVOS:

Se requiere una interfaz para controlar servomotores. Asegúrese de contar con el hardware necesario para conectar y controlar servomotores en sus proyectos RoboKit.

LEDs:

Incorporar LEDs en sus proyectos agrega capacidad de visualización y retroalimentación. Tener LEDs disponibles para su conexión a través del hardware RoboKit ampliará las posibilidades creativas.

Cables de Conexión:

Es necesario contar con cables o jumpers para conectar los diferentes componentes electrónicos al hardware RoboKit. Asegúrese de tener cables de longitud y tipo adecuados para su proyecto específico.

Notas Adicionales:

Es importante asegurarse de que el sistema cumpla con los requisitos de software y hardware antes de comenzar a programar en RoboKit.

Estos requisitos del sistema están diseñados para garantizar una experiencia de programación fluida y sin problemas con RoboKit, permitiendo a los usuarios desarrollar y ejecutar programas robóticos de manera efectiva.

4. Sintaxis del Lenguaje:

Palabras reservadas:

En RoboKit, las palabras reservadas son secuencias de símbolos del alfabeto que el propio lenguaje utiliza e identifica. A diferencia de un identificador, estas palabras ya están predefinidas y no pueden ser empleadas por el usuario para asignar un nombre a una variable o función. A continuación, se presenta la lista de todas las palabras reservadas.

Palabra	Descripción
port	Tipo de dato para definir puertos.
int	Tipo de dato entero.
real	Tipo de dato real.
string	Tipo de dato cadena.
boolean	Tipo de dato booleano.
state	Estado del componente de referencia.
temperature	Temperatura de un sensor de temperatura.
tone	Define un tono para la bocina.
true	Valor verdadero.
false	Valor falso.
array	Declara una estructura de datos de tipo arreglo.
list	Declara una estructura de datos de tipo lista.
get	Obtener un valor.

Palabra	Descripción
var	Definición de variable.
const	Definición de constante.
function	Definición de una función.

read	Función para leer un valor de entrada.
write	Función para escribir un valor de salida.
begin	Función begin. Indica el inicio de la función begin.
loop	Función loop. Indica el inicio de la función loop.
break	Indica la terminación de un bloque de instrucciones.
delay	Define un tiempo de espera.
print	Imprime texto en consola o LCD.
console	Objeto referente a consola del compilador.
lcd	Objeto referente a la pantalla LCD del dispositivo.
on	"Encendido, constante de valor high (1)."
off	"Apagado, constante de valor low (0)."
size	Tamaño de elementos de un conjunto contable.
get	Obtiene el valor almacenado en una lista.

alabra	**Descripción**
clear	Limpia la pantalla LCD o consola.
cursor	Define el cursor de la pantalla LCD.
return	Retorna el valor de la función.
moveMotor	Mueve un motor.
stopMotor	Detiene un motor.

if	Define una estructura de control de flujo condicional (if).
else	Define la parte falsa de la estructura condicional (if).
while	Define una estructura de bucle repetitivo mientras una condición sea verdadera.
do	Define la parte de la estructura de bucle do-while.
repeat	Define una estructura repetitiva para ejecutar un bloque de código un número específico de veces.
and	Operador lógico "Y".
or	Operador lógico "OR".
not	Operador lógico "NOT".
start	Inicia el estado de algunos componentes.
distance	Obtiene la distancia de un sensor de distancia.
temperature	Obtiene la temperatura de un sensor de temperatura.
insert	Inserta un dato en una lista.
append	Inserta un dato al final de una lista.
status	Retorna un valor booleano (true o false) para indicar si el componente está activo.

Estructuras de control:

RoboKit cuenta con varias estructuras de control que permiten manejar el flujo del programa. Las principales estructuras de control en RoboKit son:

Estructura Condicional (if-else):

Descripción: Permite tomar decisiones basadas en una condición lógica. Si la condición es verdadera, se ejecuta el bloque de código dentro del if. Si la condición es falsa, se ejecuta el bloque de código dentro del else.
Ejemplo:

```
var int numero = 10!
if (numero % 2 == 0) {
  console.print("Es par")!
} else {
  console.print("Es impar")!
}
```

Estructura Condicional Switch:

Descripción: Permite tomar decisiones basadas en el valor de una expresión variable, comparándolo con diferentes casos.
Ejemplo:

```
var int velocidad = 5!
var int desplazamientoX!

switch (velocidad) {
  case 1:
    desplazamientoX = desplazamientoX + velocidad!
    break!
  case 2:
    desplazamientoX = desplazamientoX + velocidad + 0.5!
    break!
}
```

Estructura Repetitiva (repeat):
Descripción: Se utiliza para ejecutar un conjunto de instrucciones un número específico de veces.
Ejemplo:

```
var int temporal!

repeat (5) {
  temporal = temporal + 1!
}
```

Estructura Repetitiva (do-while):
Descripción: Similar a la estructura while, pero garantiza que el bloque de código se ejecute al menos una vez, incluso si la condición inicialmente es falsa.
Ejemplo:

```
var int a, b, c, d!

a = 5
b = 2
c, d = 0!

do {
  c = a + b!
  d = a * b!
} while (c < 10 AND d < 10)!
```

Estos ejemplos son representativos de cómo se utilizan las estructuras de control en RoboKit para tomar decisiones y controlar el flujo del programa.

Tipos de datos:

Tipos de Datos en RoboKit:

RoboKit ofrece una variedad de tipos de datos para manipular información. A continuación, se describen los tipos de datos basados en la tabla proporcionada:

int:
Descripción: Tipo de dato entero que puede almacenar valores en el rango de -65536 a 65535.
Límite: -65536 a 65535
Espacio en Memoria: 6-7 bytes

string:
Descripción: Tipo de dato cadena que puede contener hasta 30 caracteres.
Límite: Cadena de 30 caracteres
Espacio en Memoria: 100 bytes

decimal:
Descripción: Tipo de dato decimal utilizado para números decimales en el rango de 1.4E-45 a 3.4028235E38.
Límite: 1.4E-45 a 3.4028235E38
Espacio en Memoria: 8-9 bytes

boolean:
Descripción: Tipo de dato booleano que puede almacenar valores 0 o 1 (representando false o true).
Límite: 0 o 1 (false o true)
Espacio en Memoria: 4-5 bytes

puerto:
Descripción: Tipo de dato utilizado para definir puertos de conexión a sensores o actuadores. Puede ser proximity, temperature, LED, LED_RGB, LCD, button, motor.
Límite: Puertos específicos de hardware
Espacio en Memoria: 3-11 bytes

sensor:
Descripción: Tipo de dato utilizado para representar diferentes tipos de sensores como distance, state, time, degree.
Límite: Sensores específicos de hardware
Espacio en Memoria: 4-8 bytes

Funciones y métodos:

Funciones y Métodos en RoboKit:

Las funciones y métodos en RoboKit permiten organizar y reutilizar código de manera efectiva. A continuación, se detalla la sintaxis para definir y llamar funciones/métodos, basándonos en la información proporcionada sobre RoboKit:

Definición de Funciones:

```
function nombreFuncion(parametro1, parametro2, ...) {
   # Bloque de código de la función
}
```

Ejemplo:

```
function suma(a, b) {
   return a + b!
}
```

Llamada a Funciones:

```
var resultado = nombreFuncion(valor1, valor2, ...)!
```

Ejemplo:

```
var resultado = suma(3, 5)!
```

Definición de Métodos:

```
objeto.metodo(parametro1, parametro2, ...) {
   # Bloque de código del método
}
```

Ejemplo:

```
motor1.mover(45)!
```

Llamada a Métodos:

objeto.metodo(parametro1, parametro2, ...)!

Ejemplo:
motor1.mover(45)!

Retorno de Valores:
Las funciones y métodos pueden tener un valor de retorno utilizando la palabra clave return.

Ejemplo:
```
function cuadrado(numero) {
   return numero * numero!
}
```

Uso de Variables Locales y Globales:
Las funciones y métodos pueden utilizar variables locales y globales para almacenar datos temporal o permanentemente.

Ejemplo:
```
var global = 10!

function modificarGlobal() {
   global = 20!
}
```

Estas son las principales características sintácticas para definir y llamar funciones y métodos en RoboKit, brindando a los usuarios la capacidad de modularizar su código y mejorar la legibilidad y mantenibilidad del programa.

Estructura básica de un programa en Robokit.

La estructura del programa en RoboKit sigue un patrón específico para organizar y ejecutar el código de manera eficiente. A continuación, se describe la estructura general del programa en RoboKit:

Estructura del Programa en RoboKit:

```
# ---------- Declaraciones y Asignaciones ----------
# En esta sección se declaran variables, constantes y se realizan asignaciones.
# También se pueden definir funciones personalizadas.

# Declaración de variables
var int contador!
var int distancia!

# Asignaciones
contador = 0!
distancia = 10!
```

```
# Definición de funciones
function suma(a, b) {
    return a + b!
}

# ---------- Bloque de Inicio ----------
begin {
    # Este bloque se ejecuta automáticamente al iniciar el programa.

    # Inicialización de sensores o actuadores
    start sensor1!
}

# ---------- Bloque Loop ----------
loop {
    # Este bloque se ejecuta en un ciclo continuo después del bloque de inicio.

    # Ejemplo de uso de funciones
    var resultado = suma(3, 5)!
    console.print("El resultado de la suma es:", resultado)!

    # Lógica del programa, operaciones, control de sensores, actuadores, etc.

    # Ejemplo de incremento de contador
    contador = contador + 1!

    # Condiciones, ciclos, y acciones se pueden realizar aquí.
    # ...
}
```

Ejemplo de Programa en RoboKit:

```
# Declaraciones y Asignaciones
var int contador!
contador = 0!

# Definición de Funciones
function incrementar(valor) {
    return valor + 1!
}

# Bloque de Inicio
begin {
    console.print("Iniciando Programa!")!
}

# Bloque Loop
loop {
```

```
    # Incremento del contador utilizando una función personalizada
    contador = incrementar(contador)!
    console.print("Valor actual del contador:", contador)!

    # Condición para detener el programa cuando el contador llega a 10
    if (contador >= 10) {
        console.print("Fin del Programa!")!
        break!
    }
}
```

Este ejemplo muestra la estructura típica de un programa en RoboKit, donde se realizan declaraciones, se definen funciones, y se utiliza un ciclo continuo (loop) para ejecutar las operaciones principales del programa. Se realiza una lógica simple de incremento de contador, utilizando una función personalizada, y se imprime el valor del contador en la consola. El programa termina cuando el contador alcanza el valor de 10.

5. Operadores y Expresiones en RoboKit:

RoboKit proporciona varios tipos de operadores que permiten realizar diversas operaciones en las expresiones de un programa. A continuación, se detalla cada tipo de operador junto con su explicación y precedencia.

Operadores Aritméticos:

+ (Suma): Realiza la suma de dos operandos.
var resultado = 3 + 5! # resultado será 8.

- (Resta): Realiza la resta de dos operandos.
var resultado = 10 - 7! # resultado será 3.

*** (Multiplicación):** Realiza la multiplicación de dos operandos.
var resultado = 4 * 6! # resultado será 24.

/ (División): Realiza la división de dos operandos.
var resultado = 15 / 3! # resultado será 5.

Operadores Lógicos:
and (Y): Devuelve true si ambas expresiones son verdaderas.
var resultado = (5 > 3) and (10 == 10)! # resultado será true.

or (O): Devuelve true si al menos una de las expresiones es verdadera.
var resultado = (4 < 2) or (7 >= 7)! # resultado será true

not (NO): Niega el valor de la expresión.
var resultado = not (3 == 3)! # resultado será false.

Operadores Relacionales:

< (Menor que): Devuelve true si el operando izquierdo es menor que el operando derecho.
var resultado = 8 < 12! # resultado será true

> (Mayor que): Devuelve true si el operando izquierdo es mayor que el operando derecho.
var resultado = 15 > 10! # resultado será true.

== (Igual que): Devuelve true si ambos operandos son iguales.
var resultado = 5 == 5! # resultado será true.

Precedencia de Operadores:

La precedencia determina el orden en que se evalúan las operaciones. En RoboKit, la precedencia de operadores sigue las reglas comunes de las matemáticas, donde los paréntesis tienen la mayor precedencia.

Mayor Precedencia:

- Paréntesis ()
- Operadores unarios (not)
- Multiplicación (*)
- División (/)
- Menor Precedencia:
- Suma (+)
- Resta (-)
- Relacionales (<, >, ==)
- Lógicos (and, or)

Es recomendable utilizar paréntesis para clarificar la precedencia cuando hay expresiones complejas.
Por ejemplo:
var resultado = (3 + 5) * 2!
Este cálculo primero suma 3 y 5, luego multiplica el resultado por 2.