



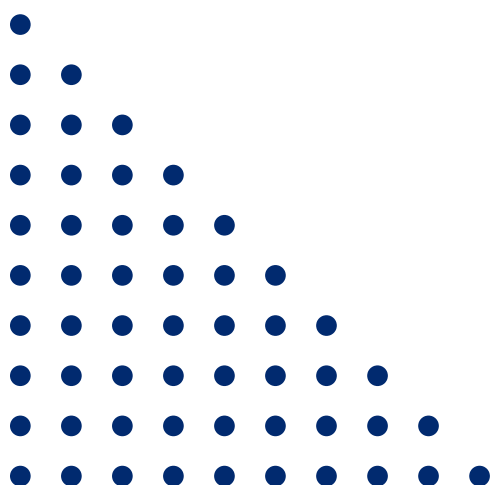
INSTITUTO TECNOLÓGICO DE
CHETUMAL

KIDCODE LENGUAJE PARA NIÑOS

PROYECTO

Lenguaje Autómatas

INGENIERÍA EN SISTEMAS COMPUTACIONALES
VERSIÓN PARA LA GRAMÁTICA



PRESENTA

IRENE ARREGUIN
GAEL MARTINEZ

16/12/2024



Introducción


El lenguaje KidCode está diseñado para introducir a los niños en el mundo de la programación. Su objetivo es ayudarles a entender los conceptos fundamentales de la programación mediante el uso de palabras y estructuras simples, facilitando su aprendizaje de manera accesible y divertida.

Objetivo:

El objetivo de este documento es proporcionar una guía completa y accesible para el aprendizaje y uso del lenguaje de programación KidCode. A través de explicaciones claras, ejemplos prácticos y ejercicios interactivos, se busca facilitar la comprensión de los conceptos básicos de programación a los niños, promoviendo un aprendizaje dinámico y divertido. Este manual está diseñado para ayudar a los usuarios a desarrollar habilidades de programación esenciales, utilizando una sintaxis sencilla y comprensible, con el fin de fomentar la creatividad, la resolución de problemas y el pensamiento lógico en los jóvenes programadores.

Beneficios:

Facilita el aprendizaje de programación en niños: Al utilizar una sintaxis sencilla y estructuras comprensibles, KidCode permite que los niños puedan aprender los conceptos básicos de programación sin sentirse abrumados.





Descripción General

Funciones principales:

Declaración de variables:

- Las variables son contenedores que almacenan datos. La declaración de variables permite asignarles un nombre y un valor inicial. Este concepto es esencial porque permite que el programa trabaje con diferentes tipos de datos, como números, cadenas de texto o valores booleanos.
- Ejemplo en KidCode: guardar e $x = 10$.


Ciclos y condiciones:

- Los ciclos permiten ejecutar un bloque de código repetidamente hasta que se cumpla una condición. Los más comunes son los bucles for y while.

Comparaciones booleanas:

- Las comparaciones booleanas son operaciones que devuelven un valor True o False. Se utilizan para comparar valores entre sí y tomar decisiones dentro del código.
- Operadores comunes: $=$, $!=$, $<$, $>$, $<=$, $>=$.

Operaciones matemáticas:

- Las operaciones matemáticas permiten realizar cálculos dentro de un programa, como suma, resta, multiplicación y división.
 - Ejemplo: $y = x + 5$.
- 



Descripción General

Precedencia de operaciones:

- La precedencia de operaciones determina el orden en que las operaciones son ejecutadas en una expresión. Las operaciones matemáticas tienen una jerarquía (por ejemplo, la multiplicación se realiza antes que la suma).

Agrupaciones:

- Las agrupaciones se utilizan para controlar el orden de ejecución de las operaciones en una expresión, mediante el uso de paréntesis.

Manejo de arreglos:

- Los arreglos (o listas) permiten almacenar múltiples valores en una sola variable. El manejo de arreglos incluye operaciones como acceder a elementos de la lista, agregar o eliminar elementos, y recorrerlos mediante ciclos.



Descripción General

Modo de uso de las Funciones:

Declaración de variables:

En KidCode, para declarar una variable, se utiliza la palabra clave "guardar" seguida del tipo de variable que se desea crear (ya sea un número entero o una cadena de texto). Después, se coloca el nombre de la variable, y para asignarle un valor, se utiliza el signo "=" seguido del valor que queremos almacenar en la variable.

Estructura General:

`guardar [tipo_de_variable] [nombre_de_variable] = [valor]`

Tipo de Valor:

- Enteros: (e)
- Flotantes: (fl)
- Cadenas (c)
- Booleanos: t (true) y f (false)
- Ejemplo:
 - `guardar e suma = 0.`
 - `guardar fl promedio = 3.14.`
 - `guardar c mensaje = "Hola".`
 - `guardar booleano = t.`

Ciclos y Condiciones:

En KidCode, dependiendo del tipo de ciclo o condición que quieras usar, las palabras reservadas varían. Cada una de ellas tiene una estructura propia que permite ejecutar bloques de código de manera repetitiva o condicional algo importante es que deben de terminar con la palabra clave fin. A continuación, te explico cada una de ellas:

Descripción General

Modo de uso de las Funciones:

Ciclos y Condiciones:

SI O IF:

La estructura "si" permite ejecutar un bloque de código solo si se cumple una condición.

- si: Inicia la estructura condicional.
- [condición]: Es la expresión que se evalúa, en este caso, si x es igual a 15.
- :: Indica el inicio del bloque de código que se ejecutará si la condición es verdadera.
- [instrucciones]: El bloque de código que se ejecuta si la condición es verdadera. En este caso, se utiliza mostrar para mostrar un mensaje en pantalla. Y debe terminar con un punto.

Estructura:

si [condición]:

[bloque de código].

fin

Ejemplo:

si edad > 18:

mostrar("Eres mayor de edad").

fin

Descripción General

Modo de uso de las Funciones:

Ciclos y Condiciones:

SI NO (ELSE IF):

La palabra si no se usa para comprobar una condición alternativa si la condición del si no se cumple.

- si no: Introduce una nueva condición que solo se evalúa si la anterior (del si) es falsa.
- [otra_condición]: En este caso, evalúa si x es mayor que 15. Y debe terminar con un punto.
- [otras_instrucciones]: Si la condición del si no se cumple, se ejecutan estas instrucciones. Y debe terminar con un punto.
-

Estructura:

si [condición]:

[instrucciones].

si no [otra_condición]:

[otras_instrucciones].

fin

Ejemplo:

si x == 15:

mostrar("x es igual a 15").

si no x > 15:

mostrar("x es mayor que 15").

fin

Descripción General

Modo de uso de las Funciones:

Ciclos y Condiciones:

SINO (ELSE):

La palabra sino captura todos los casos en los que ninguna de las condiciones anteriores se cumpla, y es la última opción a evaluar en una estructura condicional.

- sino: Si ninguna de las condiciones previas se cumple, se ejecutan las instrucciones bajo sino.

Estructura:

si [condición]:

[instrucciones].

si no [otra_condición]:

[otras_instrucciones].

sino:

[instrucciones_finales]

Ejemplo:

si x == 15:

mostrar("x es igual a 15").

si no x > 15:

mostrar("x es mayor que 15").

sino:

mostrar("x es menor que 15").

fin

Descripción General

Modo de uso de las Funciones:

Ciclos y Condiciones:

MIENTRAS (WHILE):

La estructura mientras repite un bloque de código mientras una condición sea verdadera. Cuando la condición se vuelve falsa, el ciclo termina.

- mientras: Inicia el ciclo de repetición.
- [condición]: La condición que se evalúa en cada iteración. Mientras esta condición sea verdadera, el ciclo continuará.
- :: Indica el comienzo del bloque de código que se repetirá.
- [instrucciones]: El bloque de código que se ejecuta en cada iteración del ciclo. En este caso, muestra un mensaje y aumenta el valor de x. Y debe terminar con un punto.

Estructura:

mientras [condición]:

[instrucciones].

fin

Ejemplo:

mientras x < 20:

mostrar("x es menor que 20").

x = x + 1.

fin

Descripción General

Modo de uso de las Funciones:

Ciclos y Condiciones:

PARA (FOR):

La estructura para se utiliza para realizar una repetición un número determinado de veces. Se debe especificar un inicio, una condición de continuación y un paso de incremento o decremento.

- **para:** Indica el inicio de un ciclo con número determinado de iteraciones.
- **[variable]:** La variable que controlará el número de iteraciones (en este caso, *i*).
- **[valor_inicial]:** El valor inicial que tomará la variable (en este caso, 0).
- **[condición]:** La condición que debe cumplirse para que el ciclo siga ejecutándose (en este caso, $i < 5$).
- **[paso]:** El valor por el cual se incrementa o decrementa la variable después de cada iteración (en este caso, el paso es implícito y sería 1).
- **::** Marca el inicio del bloque de código que se ejecutará en cada iteración.
- **[instrucciones]:** El bloque de código que se ejecuta en cada iteración. En este caso, se imprime el número de la iteración.

Estructura:

```
para [variable] = [valor_inicial], [condición]; [paso]:  
    [instrucciones].
```

fin

Ejemplo:

```
para i = 0, i < 5:
```

```
    mostrar("Iteración:", i).
```

```
fin
```



Descripción General

Modo de uso de las Funciones:

Comparaciones Booleanas:

IGUAL A (==):

Este operador verifica si dos valores son exactamente iguales. Si lo son, la comparación devuelve verdadero; de lo contrario, devuelve falso.

- `x == 10`: Evalúa si el valor de `x` es igual a 10.
- Si la condición es verdadera, se ejecuta el bloque de código asociado.

Estructura:

`[expresión1] == [expresión2]`

Ejemplo:

si `x == 10`:

 mostrar("x es igual a 10").

fin


DIFERENTE DE (X=):

Este operador verifica si dos valores no son iguales. Si son diferentes, devuelve verdadero; de lo contrario, devuelve falso.

- `x x= 5`: Evalúa si el valor de `x` no es igual a 5.
- Si la condición es verdadera, se ejecuta el bloque de código asociado.

Estructura:

`[expresión1] x= [expresión2]`





Descripción General

Modo de uso de las Funciones:

Ejemplo:

si $x = 5$:

 mostrar("x no es igual a 5").

fin

Comparaciones Booleanas:

MENOR QUE (<):

Este operador verifica si un valor es estrictamente menor que otro.

Devuelve verdadero si lo es, y falso en caso contrario.

- $x < 10$: Evalúa si el valor de x es menor que 10.
- Si la condición es verdadera, se ejecuta el bloque de código asociado.

Estructura:

[expresión1] > [expresión2]

Ejemplo:

si $x > 20$:

 mostrar("x es mayor que 20").

fin

MAYOR QUE (>):

Este operador verifica si un valor es estrictamente mayor que otro.

Devuelve verdadero si lo es, y falso en caso contrario.





Descripción General

Modo de uso de las Funciones:

- $x > 20$: Evalúa si el valor de x es mayor que 20.
- Si la condición es verdadera, se ejecuta el bloque de código asociado.

Estructura:

[expresión1] > [expresión2]

Ejemplo:

si $x > 20$:

 mostrar("x es mayor que 20").

fin

MAYOR O IGUAL QUE (\geq)

Este operador verifica si un valor es mayor o igual a otro. Devuelve verdadero si lo es, y falso en caso contrario.

- $x \geq 100$: Evalúa si el valor de x es mayor o igual a 100.
- Si la condición es verdadera, se ejecuta el bloque de código asociado

Estructura:

[expresión1] \geq [expresión2]

Ejemplo:

si $x \geq 100$:

 mostrar("x es mayor o igual a 100").

fin





Descripción General

Modo de uso de las Funciones:

Operaciones matemáticas:

Las operaciones matemáticas en KidCode permiten realizar cálculos y manipular valores numéricos. Estas son esenciales para la lógica y resolución de problemas en programación. A continuación, se detallan los operadores matemáticos disponibles y su funcionamiento.

SUMA (+):

El operador + se utiliza para sumar dos valores numéricos.

- $5 + 3$: Suma los valores 5 y 3, dando como resultado 8.
- El valor se almacena en la variable resultado.

Estructura:

`[valor1] + [valor2]`

Ejemplo:

`guardar resultado = 5 + 3.`

`mostrar("El resultado es: "; resultado).`

RESTA (-):

El operador - se utiliza para restar un valor numérico de otro.

- $10 - 4$: Resta el valor 4 del valor 10, dando como resultado 6.
- El resultado se almacena en la variable diferencia.

Estructura:

`[valor1] - [valor2]`





Descripción General

Ejemplo:

```
guardar diferencia = 10 - 4.
```

```
mostrar("La diferencia es: "; diferencia).
```

División (/):

El operador / se utiliza para dividir un valor entre otro.

- 20 / 4: Divide el valor 20 entre 4, dando como resultado 5.
- El valor se guarda en la variable cociente.

Estructura:

```
[valor1] / [valor2]
```

Ejemplo:

```
guardar cociente = 20 / 4.
```

```
mostrar("El cociente es: "; cociente).
```

Multiplicación (*)

El operador * se utiliza para multiplicar dos valores numéricos.

- 6 * 3: Multiplica los valores 6 y 3, dando como resultado 18.
- El resultado se almacena en la variable producto.

Estructura:

```
[valor1] * [valor2]
```

Ejemplo:

```
guardar producto = 6 * 3.
```

```
mostrar("El producto es: "; producto).
```





Descripción General

Raíz cuadrada (//)

El operador `//` se utiliza para obtener la raíz cuadrada de un número.

- `16 // 2`: Calcula la raíz cuadrada de 16, dando como resultado 4.
- El resultado se guarda en la variable raíz.

Estructura:

`[valor] // 2`

Ejemplo:

```
guardar raíz = 16 // 2.
```

```
mostrar("La raíz cuadrada es: "; raíz).
```

Potencia (**):

El operador `**` se utiliza para elevar un número a una potencia específica.

- `3 ** 2`: Eleva el número 3 al cuadrado (3×3), dando como resultado 9.
- El valor se almacena en la variable potencia.

Estructura:

`[base] ** [exponente]`

Ejemplo:

```
guardar potencia = 3 ** 2.
```

```
mostrar("La potencia es: "; potencia).
```





Descripción General

Precedencia de Operadores: Las operaciones se realizan siguiendo la jerarquía matemática:

- Paréntesis primero.
- Potencias después.
- Multiplicación y división.
- Finalmente, suma y resta.

Ejemplo:

```
guardar resultado = (2 + 3) * 4.
```

```
mostrar("Resultado: "; resultado). # Resultado: 20
```

Manejo de Arreglos en KidCode

En KidCode, los arreglos son estructuras que permiten almacenar múltiples valores del mismo tipo. Su uso es muy importante para gestionar listas, datos repetidos o colecciones.

Declaración de Arreglos:

Para declarar un arreglo, se utiliza la palabra clave arreglo, seguida del tipo de dato y el tamaño.

Estructura:

arreglo [nombre] de [tipo] tamaño [número].

Ejemplo:

arreglo miArreglo de e tamaño 10.

