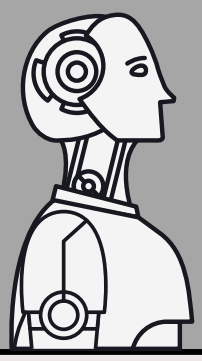
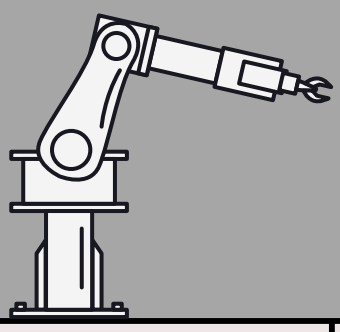
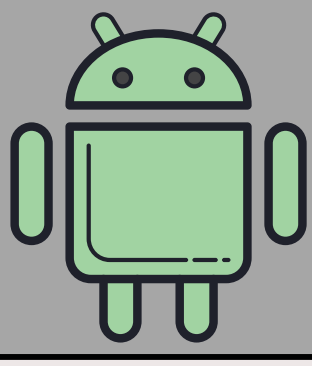
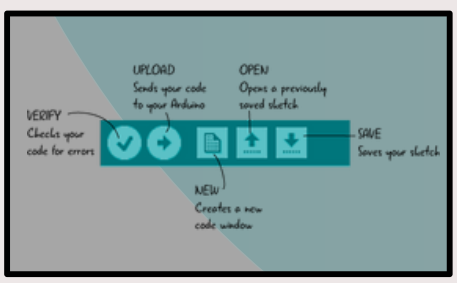




preview.jpg



ESTRUCTURA

Bosquejo ×

loop()

setup()

Estructura de control ×

do…while

else

for

if

return

while

Sintaxis adicional ×

#define (definir).

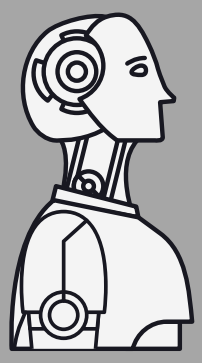
#include (incluir).

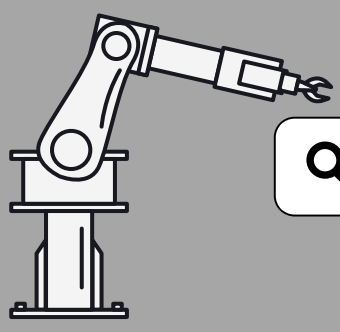
/* */ (comentario de bloque).

// (comentario de una sola línea).

; (punto y coma).

{} (llaves).





BOSQUEJO(SKETCH)

ARDUINO

Setup()

Descripción

La **setup()** función se llama cuando se inicia un boceto. Úselo para inicializar variables, modos pin, comenzar a usar bibliotecas, etc. La **setup()** función solo se ejecutará una vez, después de cada encendido o reinicio de la placa Arduino.

CÓDIGO DE EJEMPLO

```
int buttonPin = 3;

void setup() {
  Serial.begin(9600);
  pinMode(buttonPin, INPUT);
}

void loop() {
  // ...
}
```

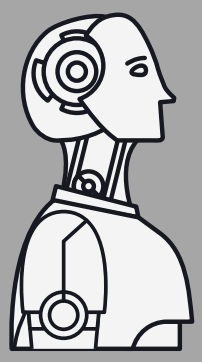
Loop()

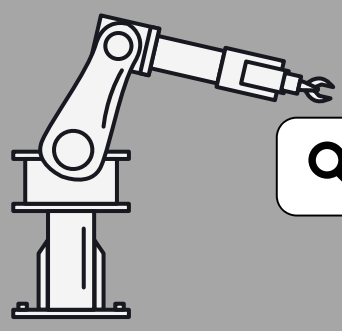
Descripción

Después de crear una **setup()** función, que inicializa y establece los valores iniciales, la **loop()** función hace exactamente lo que sugiere su nombre y se repite consecutivamente, lo que permite que su programa cambie y responda. Úselo para controlar activamente la placa Arduino.

CÓDIGO DE EJEMPLO

```
int buttonPin = 3;

// La configuración inicializa el serial y el pin del botón

void setup() {
  Serial.begin(9600);
  pinMode(buttonPin, INPUT);
}

// El bucle comprueba el pin del botón cada vez,
// y enviará serial si se presiona

void loop() {
  if (digitalRead(buttonPin) == HIGH) {
    Serial.write('H');
  }
  else {
    Serial.write('L');
  }

  delay(1000);
}
```





ESTRUCTURA DE CONTROL

do...while

Descripción

El **do...while** ciclo funciona de la misma manera que el whileciclo, con la excepción de que la condición se prueba al final del ciclo, por lo que el ciclo do siempre se ejecutará al menos una vez.

Sintaxis

```
do {
  // statement block
} while (condition);
```

Parámetros

condition: una expresión booleana que se evalúa como trueo false.

CÓDIGO DE EJEMPLO

```
int x = 0;
do {
  delay(50);           // Esperar a que los sensores se estabilicen
  x = readSensors();   // Revisa los sensores
} while (x < 100);
```

else

Descripción

Permite un mayor control sobre el **if...else** flujo de código que la ifdeclaración básica, al permitir que se agrupen múltiples pruebas. Se elseejecutará una cláusula (si es que existe) si la condición en la ifdeclaración da como resultado false. El elsepuede continuar con otra ifprueba, de modo que se pueden ejecutar varias pruebas mutuamente excluyentes al mismo tiempo.

Sintaxis

```
if (condition1) {
  // do Thing A
}
else if (condition2) {
  // do Thing B
}
else {
  // do Thing C
}
```

CÓDIGO DE EJEMPLO

```
if (temperature >= 70) {
  // Danger! Shut down the system.
}
else if (temperature >= 60) { // 60 <= temperature < 70
  // ¡Advertencia! Se requiere atención del usuario.
}
else { // la temperatura < 60
  // ¡Seguro! Continuar con las tareas habituales.
}
```

for

Descripción

La **for** sentencia se utiliza para repetir un bloque de sentencias entre llaves. Normalmente se utiliza un contador de incrementos para incrementar y terminar el bucle.

Sintaxis

```
for (initialization; condition; increment) {
  // statement(s);
}
```

Parámetros

initialization: sucede primero y exactamente una vez.

condition: cada vez que pasa por el bucle, conditionse prueba; si es true, el bloque de instrucciones y se ejecuta el incremento , la condición se vuelve a probar.

increment: se ejecuta cada vez que pasa por el bucle cuando conditiones true.

CÓDIGO DE EJEMPLO

```
// Dim an LED using a PWM pin
int PWMpin = 10;  // LED en serie con resistencia de 470 ohmios en el pin 10

void setup() {
  // no se necesita configuración
}

void loop() {
  for (int i = 0; i <= 255; i++) {
    analogWrite(PWMpin, i);
    delay(10);
  }
}
```

if

Descripción

La ifdeclaración comprueba una condición y ejecuta la siguiente declaración o conjunto de declaraciones si la condición es 'verdadera'.

Sintaxis

```
if (condition) {
  //statement(s)
}
```

Parámetros

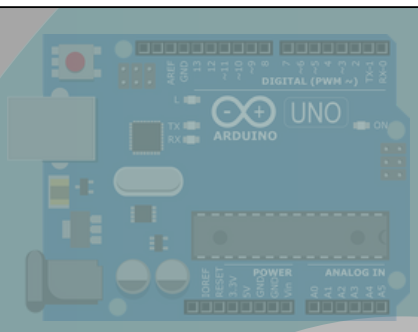
condition: una expresión booleana (es decir, puede ser trueo false).

CÓDIGO DE EJEMPLO

```
if (x > 120) digitalWrite(LEDpin, HIGH);

if (x > 120)
digitalWrite(LEDpin, HIGH);

if (x > 120) {digitalWrite(LEDpin, HIGH);}

if (x > 120) {
  digitalWrite(LEDpin1, HIGH);
  digitalWrite(LEDpin2, HIGH);
}
// Todas son correctas
```

OPERADORES DE COMPARACIÓN

```
x == y (x es igual a y)
x != y (x no es igual a y)
x < y (x es menor que y)
x > y (x es mayor que y)
x <= y (x es menor o igual que y)
x >= y (x es mayor o igual que y)
```



return

Descripción

Finalice una función y devuelva un valor de una función a la función que llama, si lo desea.

Sintaxis

```
return;
return value;
```

Parámetros

value: Tipos de datos permitidos: cualquier tipo variable o constante.

CÓDIGO DE EJEMPLO

Una función para comparar la entrada de un sensor con un umbral.

```
int checkSensor() {
  if (analogRead(0) > 400) {
    return 1;
  }
  else {
    return 0;
  }
}
```

while

Descripción

Un **while** bucle se repetirá continuamente, e infinitamente, hasta que la expresión dentro del paréntesis, () se vuelva falsa. Algo debe cambiar la variable probada, o el ciclo while nunca saldrá.

Sintaxis

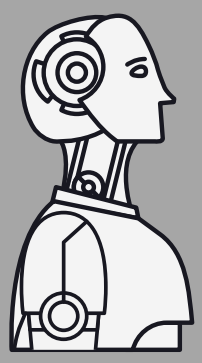
```
while (condition) {
  // statement(s)
}
```

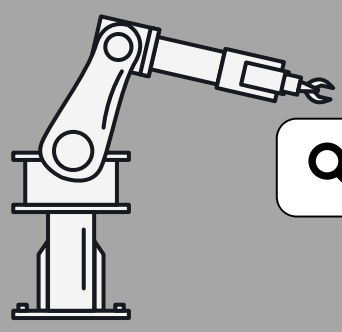
Parámetros

condition: Una expresión booleana que se evalúa como trueo false.

CÓDIGO DE EJEMPLO

```
var = 0;
while (var < 200) {
  // Hacer algo repetitivo 200 veces
  var++;
}
```





SINTAXIS ADICIONAL

#define (definir)

Descripción

#define es un componente útil de C++ que permite al programador dar un nombre a un valor constante antes de compilar el programa.

Sintaxis

#define constantName value

Parámetros

constantName: El nombre de la macro a definir.
value: El valor a asignar a la macro.

CÓDIGO DE EJEMPLO

```
#define ledPin 3
// El compilador reemplazará cualquier mención
//de ledPin con el valor 3 en el momento de la compilación.
```

#include (Incluir)

Descripción
#include se utiliza para incluir bibliotecas externas en su boceto.

Sintaxis
#include <LibraryFile.h>
#include "LocalFile.h"

Parámetros
LibraryFile.h: cuando se utiliza la sintaxis de corchetes angulares, se buscará el archivo en las rutas de las bibliotecas.

LocalFile.hla carpeta del archivo que usa la directiva para el archivo especificado, luego las rutas de las bibliotecas si no se encontró en la ruta local. #includeUtilice esta sintaxis para archivos de encabezado en la carpeta del boceto.

CÓDIGO DE EJEMPLO

```
#include <Servo.h>

Servo myservo;  // Crear objeto servo para controlar un servo

void setup() {
  myservo.attach(9);  // Conecta el servo en el pin 9 al objeto servo
}

void loop() {
  for (int pos = 0; pos <= 180; pos += 1) { // Va de 0 grados a 180 grados
    // in steps of 1 degree
    myservo.write(pos);              // Decirle al servo que vaya a la posición en la variable 'pos'
    delay(15);                       // Espera 15 ms para que el servo alcance la posición
  }
  for (int pos = 180; pos >= 0; pos -= 1) { // Va de 180 grados a 0 grados
    myservo.write(pos);              // Decirle al servo que vaya a la posición en la variable 'pos'
    delay(15);                       // Espera 15 ms para que el servo alcance la posición
  }
}
```

/* */ (Comentario de bloque)

Descripción

Los comentarios son líneas en el programa que se utilizan para informarse a sí mismo oa otros sobre la forma en que funciona el programa.

El comienzo de un comentario de bloque o un comentario de varias líneas está marcado por el símbolo /*y el símbolo */marca su final. Este tipo de comentario se llama así porque puede extenderse por más de una línea; una vez que el compilador lee /*, ignora lo que sigue hasta que encuentra un archivo */.

CÓDIGO DE EJEMPLO

```
/* Este es un comentario valido */

/*
  Parpadear
  Enciende un LED durante un segundo, luego se apaga durante un segundo, repetidamente.

  Este código de ejemplo es de dominio público.
  (Otro comentario válido)
*/

/*
  if (gwb == 0) { // El comentario de una sola línea está bien dentro de un comentario de varias líneas
    x = 3;          /* Pero no otro comentario de varias líneas: esto no es válido */
  }
// no olvide el comentario de "cierre": ¡tienen que estar equilibrados!
*/
```

// (Comentario de una sola línea)

Descripción

Los comentarios son líneas en el programa que se utilizan para informarse a sí mismo oa otros sobre la forma en que funciona el programa.

Un comentario de una sola línea comienza con // (dos barras inclinadas adyacentes). Este comentario termina automáticamente al final de una línea. Todo lo que sigue // hasta el final de una línea será ignorado por el compilador.

CÓDIGO DE EJEMPLO

```
// El pin 13 tiene un LED conectado en la mayoría de las placas Arduino.
// Dale un nombre:
int led = 13;
digitalWrite(led, HIGH); //Encienda el LED (ALTO es el nivel de voltaje)
```

; (Punto y coma)

Descripción

Se utiliza para finalizar una declaración.

Olvidar terminar una línea con un punto y coma resultará en un error de compilación.

CÓDIGO DE EJEMPLO

```
int a = 13;
```

{} (Llaves)

Descripción

Las llaves (también denominadas simplemente "llaves" o "corchetes") son una parte importante del lenguaje de programación C++. Se utilizan en varias construcciones diferentes, que se describen a continuación, y esto a veces puede resultar confuso para los principiantes.

Las llaves desequilibradas a menudo pueden conducir a errores de compilación crípticos e impenetrables que a veces pueden ser difíciles de rastrear en un programa grande. Debido a sus variados usos, las llaves también son increíblemente importantes para la sintaxis de un programa y mover una llave una o dos líneas a menudo afectará dramáticamente el significado de un programa.

CÓDIGO DE EJEMPLO

Los principales usos de las llaves se enumeran en los ejemplos a continuación.

```
Funciones
void myfunction(datatype argument) {
  // Cualquier afirmación
}
```

```
Bucles

while (boolean expression) {
  // Cualquier afirmación
}

do {
  // cualquier afirmación
} while (boolean expression);

for (initialisation; termination condition; incrementing expr) {
  // Cualquier afirmación
}
```

```
Declaraciones condicionales

if (boolean expression) {
  // Cualquier afirmación
}

else if (boolean expression) {
  // Cualquier afirmación
}
else {
  // Cualquier afirmación
}
```