VARIABLE SCOPE & QUALIFIERS

Const

Descripción

La **const** palabra clave significa constante. Es un calificador de variable que modifica el comportamiento de la variable, haciendo que una variable sea de "solo lectura ". Esto significa que la variable se puede usar como cualquier otra variable de su tipo, pero su valor no se puede cambiar.

CODIGO EJEMPLO

Las constantes definidas con la **const** palabra clave obedecen las reglas de alcance de variables que rigen otras variables.

```
const float pi = 3.14;
float x;
// ....
x = pi * 2; // Está bien usar constantes en matemáticas.

pi = 7;     // Ilegal: no puede escribir (modificar) una constante.
```

Scope

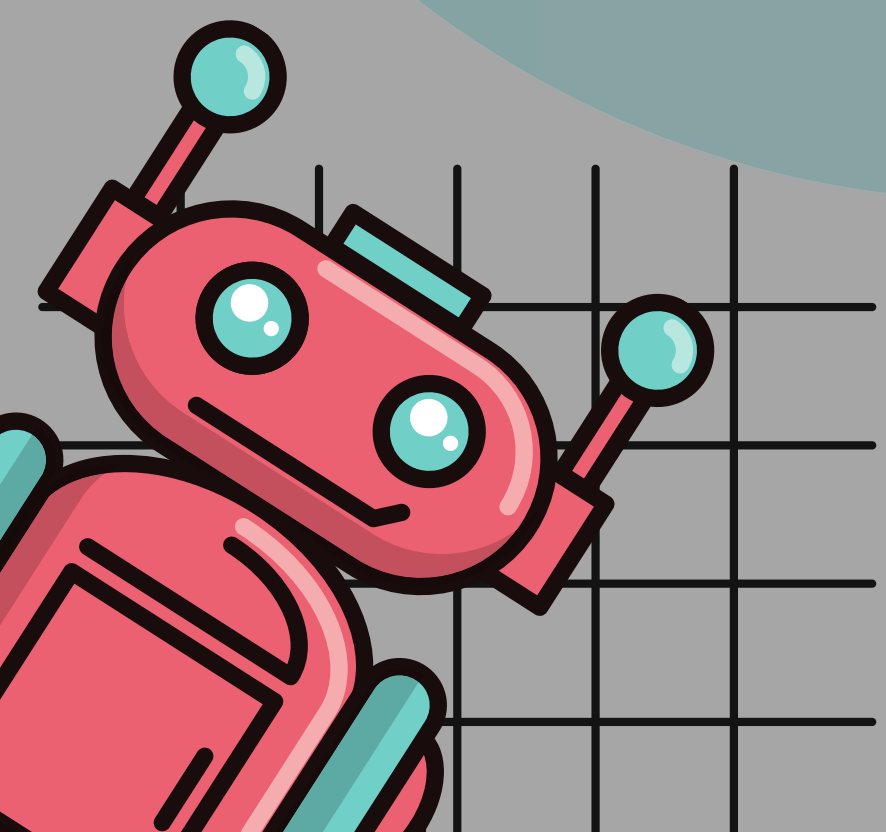
Descripción

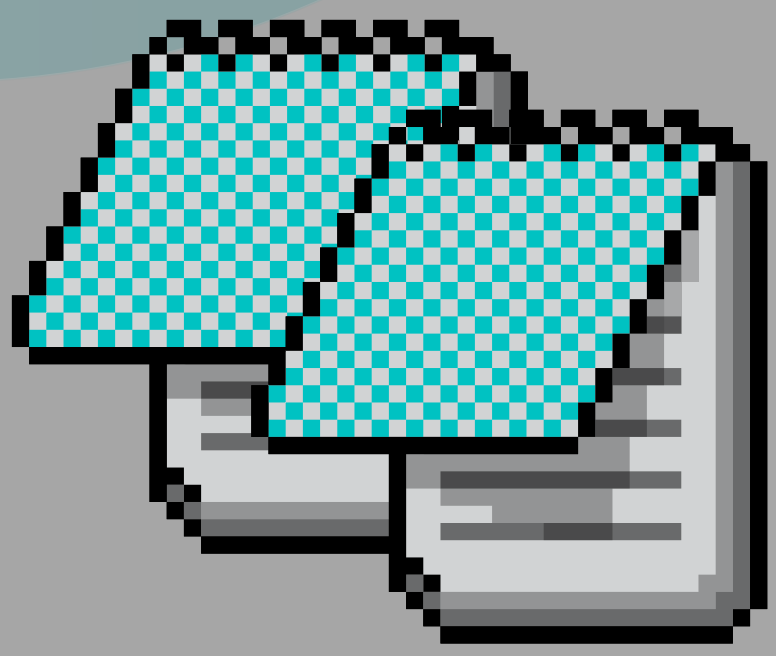
Una variable global es aquella que puede ser vista por cada función en un programa. Las variables locales solo son visibles para la función en la que se declaran. En el entorno Arduino, cualquier variable declarada fuera de una función (p. ej setup()., loop(), etc.), es una variable global .

CODIGO EJEMPLO

A veces también es útil declarar e inicializar una variable dentro de un forbucle. Esto crea una variable a la que solo se puede acceder desde dentro de los forcorchetes - loop.

```
int gPWMval;  // Cualquier función verá esta variable

void setup() {
  // ...
}

void loop() {
  int i;    // "i" Solo es "visible" dentro de "bucle"
  float f;  // "f" Solo es "visible" dentro de "bucle"
  // ...

  for (int j = 0; j < 100; j++) {
    // Solo se puede acceder a la variable j dentro de los corchetes del bucle for
  }
}
```





VARIABLE SCOPE & QUALIFIERS

Static

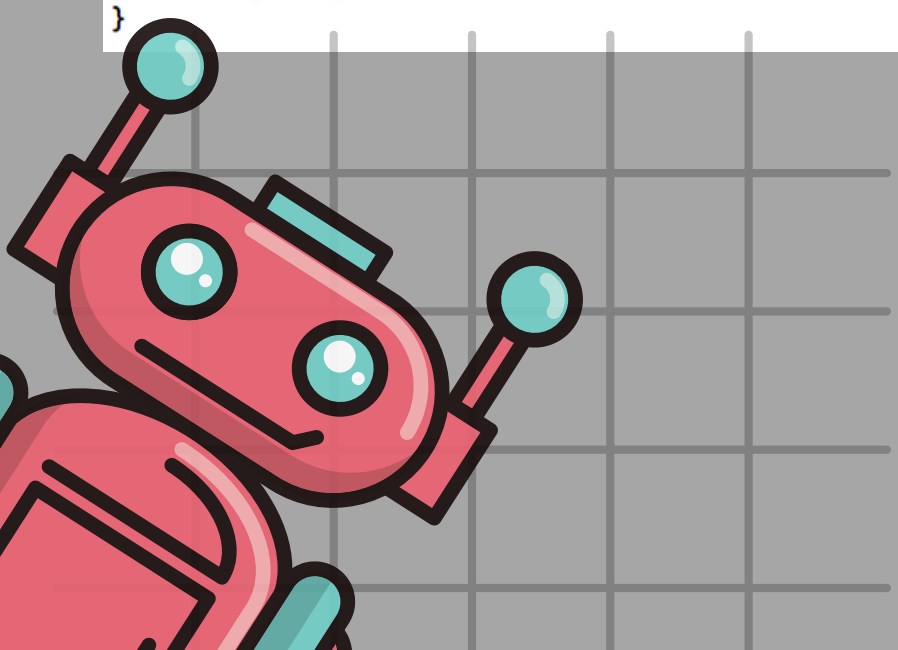
Descripción

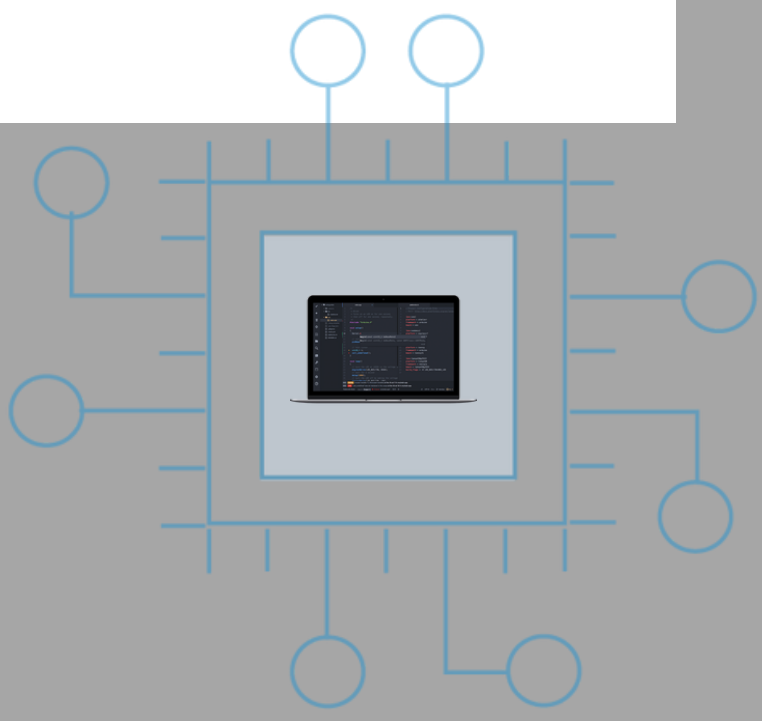
La **static** palabra clave se utiliza para crear variables que son visibles para una sola función. Sin embargo, a diferencia de las variables locales que se crean y destruyen cada vez que se llama a una función, las variables estáticas persisten más allá de la llamada a la función, preservando sus datos entre las llamadas a la función.

CODIGO EJEMPLO

Las variables declaradas como estáticas solo se crearán e inicializarán la primera vez que se llame a una función.

```
#define randomWalkLowRange -20
#define randomWalkHighRange 20
int stepsize;

int thisTime;

void setup() {
  Serial.begin(9600);
}

void loop() {
  //  test randomWalk function
  stepsize = 5;
  thisTime = randomWalk(stepsize);
  Serial.println(thisTime);
  delay(10);
}

int randomWalk(int moveSize) {
  static int place; // variable para almacenar valor en caminata aleatoria - declarada estática para que almacene.
  // Valores entre llamadas de función, pero ninguna otra función puede cambiar su valor.

  place = place + (random(-moveSize, moveSize + 1));

  if (place < randomWalkLowRange) {                              // Comprobar los límites inferior y superior

    place = randomWalkLowRange + (randomWalkLowRange - place);    // Reflejar el número en dirección positiva
  }
  else if (place > randomWalkHighRange) {
    place = randomWalkHighRange - (place - randomWalkHighRange); // Reflejar el número en dirección negativa

  }

  return place;
}
```





VARIABLE SCOPE & QUALIFIERS

volatile

Descripción

volatile es una palabra clave conocida como calificador de variable , generalmente se usa antes del tipo de datos de una variable, para modificar la forma en que el compilador y el programa posterior tratan la variable.

Declarar una variable volatilees una directiva para el compilador. El compilador es un software que traduce su código C/C++ al código de la máquina, que son las instrucciones reales para el chip Atmega en el Arduino.

int or long volatiles

Descripción

Si la volatilevariable es más grande que un byte (por ejemplo, un int de 16 bits o una longitud de 32 bits), el microcontrolador no puede leerlo en un solo paso, porque es un microcontrolador de 8 bits. Esto significa que mientras su sección de código principal (por ejemplo, su ciclo) lee los primeros 8 bits de la variable, la interrupción ya podría cambiar los segundos 8 bits.

CODIGO EJEMPLO

```
// Parpadea el LED durante 1 s si la entrada ha cambiado
// en el segundo anterior.

volatile byte changed = 0;

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  attachInterrupt(digitalPinToInterrupt(2), toggle, CHANGE);
}

void loop() {
  if (changed == 1) {
    // ¡toggle() ha sido llamado desde interrupciones!

    // Restablecer cambiado a 0
    changed = 0;

    // LED parpadeante durante 200 ms
    digitalWrite(LED_BUILTIN, HIGH);
    delay(200);
    digitalWrite(LED_BUILTIN, LOW);
  }
}

void toggle() {
  changed = 1;
}
```