

MANUAL AUXILIAR DE PROGRAMACIÓN

Medición e instrumentación
13/3/2019
Carlos Rivera

¿QUÉ ES UN ALGORITMO?

"Formalmente definimos un algoritmo como un conjunto de pasos, procedimientos sistemáticos o acciones específicas que nos permiten alcanzar un resultado o resolver un problema.¹"

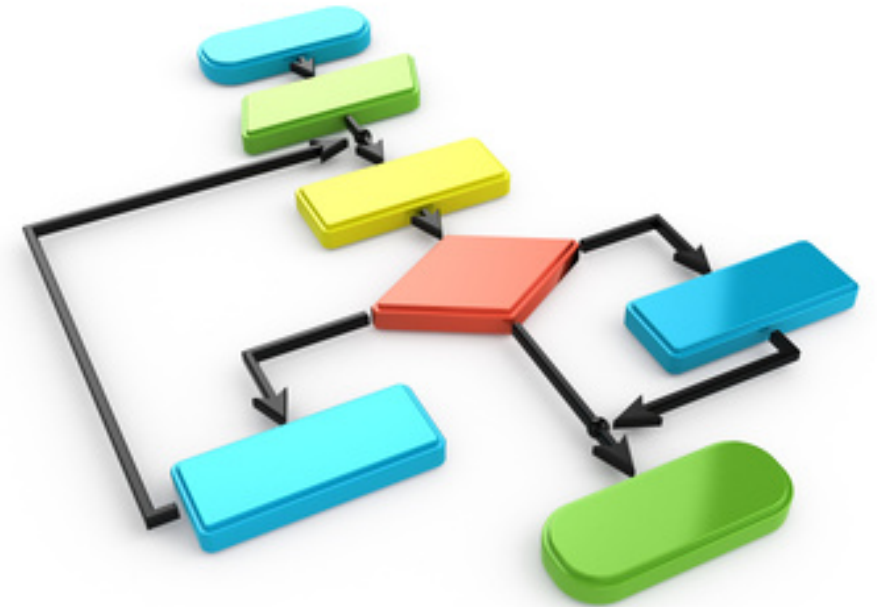


DIAGRAMA DE FLUJO




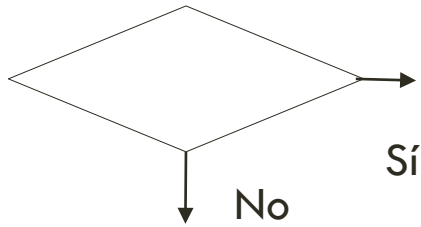
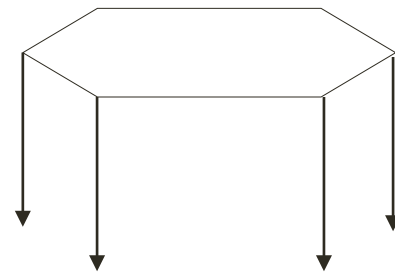
Símbolo	Explicación del símbolo
	Se emplea para marcar el <i>inicio</i> y <i>fin</i> del diagrama de flujo.
	Se emplea para introducir los datos de entrada. Expresa <i>lectura</i> .
	Representa un proceso. En su interior se colocan asignaciones, operaciones aritméticas, cambios de valor de celdas en memoria, etc.

DIAGRAMA DE FLUJO

Símbolo



Se destina para hacer una decisión.

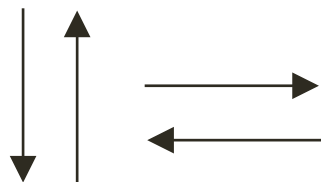
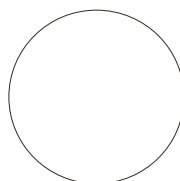
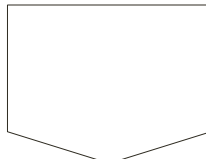
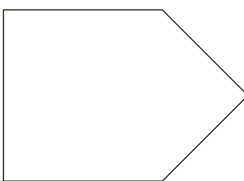


Se usa para representar una decisión múltiple, switch. En su interior se almacena un selector, y depende del valor de dicho selector, se sigue por una de las ramas o caminos selectivos.

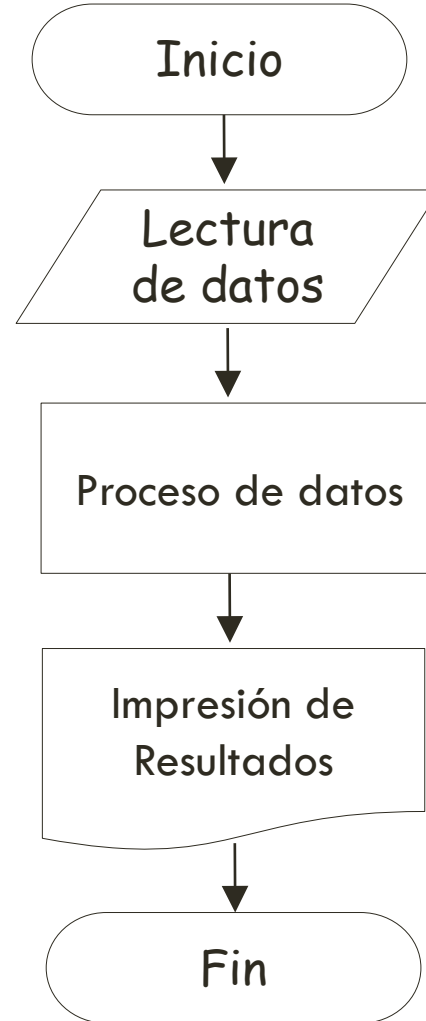


Se utiliza para representar la impresión de un resultado, por lo general, expresa escritura

DIAGRAMA DE FLUJO

Símbolo	Explicación del símbolo
	Expresan la dirección del flujo del diagrama
	Expresa conexión dentro de una misma página.
	Represan conexión entre páginas diferentes.
	Se utiliza para expresar un módulo de un problema, subproblema, que hay que resolver antes de continuar con el flujo normal del diagrama.

EJEMPLO



TIPO DE DATOS

Tipos de datos básicos

	Descripción	Rango
int	Números enteros finitos	-32768 a +32767
float	Números reales finitos	3.4×10^{-38} a 3.4×10^{38}
long	Números enteros finitos de largo alcance	-2,147,483,648 a +2,147,483,647
double	Números reales de doble precisión	1.7×10^{-308} a 1.7×10^{308}
char	Es un caracter	Símbolos del abecedario, números o símbolos especiales, que van cerrados entre comillas.
string	Arreglo unidimensional de caracteres o cadena	

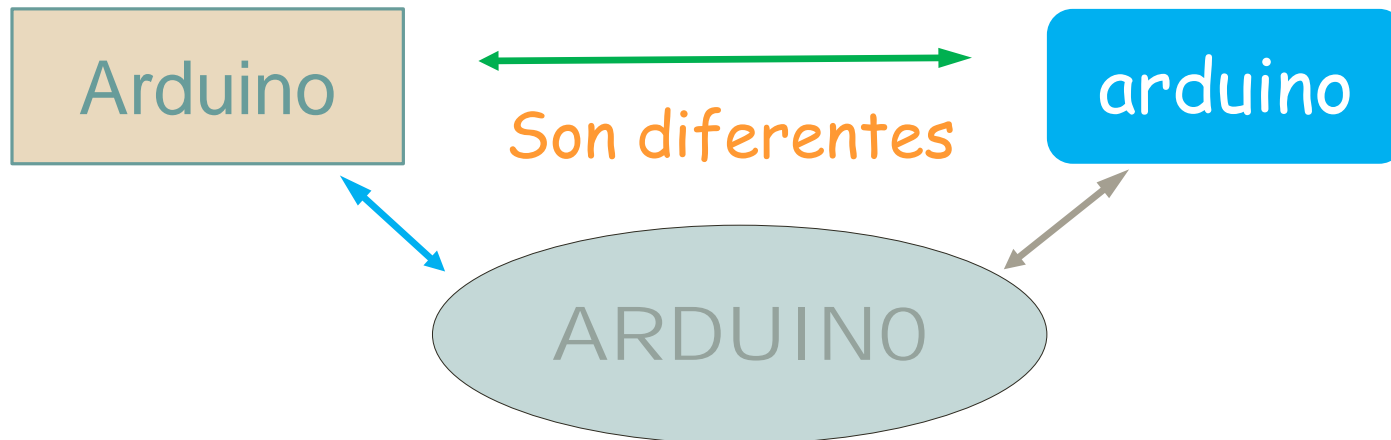
TIPO DE DATOS EN ARDUINO

Tipos de datos básicos		Descripción	Rango
void	↔	No tiene valor alguno, se emplea en la declaración de funciones.	
byte	↔	Almacena 8 bits de números no signados.	↔ 0-255
word	↔	Almacena 16 bits de números no signados.	↔ 0-65535
array	↔	Un arreglo es una colección de variables, a las que se accede a través de un número de índice.	

IDENTIFICADORES

Los datos que se registran en una computadora suelen guardarse en una casilla; a la cuales hay que asignarles un nombre, **un identificador**.

El **identificador** es único en su tipo, dado que los lenguajes de programación habitualmente hacen distinción entre mayúsculas y minúsculas.



PALABRAS RESERVADAS

Hay que tomar en cuenta que existen algunas palabras que no se pueden utilizar como identificadores, a éstas se les denominan palabras reservadas.

Structure

- setup()
- loop()

Control Structures

- if
- if...else
- for
- switch case
- while
- do... while
- break
- continue
- return
- goto

Further Syntax

- ; (semicolon)
- {} (curly braces)
- // (single line comment)
- /* */ (multi-line comment)
- #define
- #include

Arithmetic Operators

- = (assignment operator)
- + (addition)

Variables

Constants

- HIGH | LOW
- INPUT | OUTPUT | INPUT_PULLUP
- LED_BUILTIN
- true | false
- integer constants
- floating point constants

Data Types

- void
- boolean
- char
- unsigned char
- byte
- int
- unsigned int
- word
- long
- unsigned long
- short
- float
- double
- string - char array
- String - object
- array

Functions

Digital I/O

- pinMode()
- digitalWrite()
- digitalRead()

Analog I/O

- analogReference()
- analogRead()
- analogWrite() - PWM

Due & Zero only

- analogReadResolution()
- analogWriteResolution()

Advanced I/O

- tone()
- noTone()
- shiftOut()
- shiftIn()
- pulseIn()

Time

- millis()
- micros()
- delay()
- delayMicroseconds()

CONSTANTES

Las constantes son datos que no cambian durante la ejecución de un programa. Para poder designar a las constantes también se utilizan identificadores.

Podemos definir a una constante con el tipo de dato que nosotros deseemos utilizar, entero, real, caracter.

Las constantes se deben definir antes de comenzar el programa principal.

EJEMPLO

```
1  /*
2  **  Formas de definir una constante
3  */
4  const int    C1=15;          /* C1 es una constante de tipo entero*/
5  const float  Pi=3.1415;     /* Pi es una constante de tipo real*/
6  const char   ca1='s';       /* ca1 es una constante de tipo character*/
7  const int    buttonPin=2;    /* Aquí se asigna el pin 2 corresponde al botón*/
8  const int    ledPin=13;      /* El pin corresponde al LED*/
9  // Otra forma para definir constantes
10 #define      int    C1=15;    /* C1 es una constante de tipo entero*/
11 #define      float  Pi=3.1415; /* Pi es una constante de tipo real*/
12 #define      char   ca1='s'   /* ca1 es una constante de tipo character*/
13 #define      int    buttonPin=2; /* Aquí se asigna el pin 2 corresponde al botón*/
14 #define      int    ledPin=13; /* El pin corresponde al LED*/
15
```

VARIABLES

Una variable es aquella que puede adquirir distintos valores numéricos.

Una variable en programación, es una forma bastante simple de contener información y cambiar su valor durante la ejecución de un programa. Por lo general se declaran en el programa principal.

Al igual que las constantes, las variables para nombrarlas se necesitan de identificadores, pueden existir tipos de variables de todos los tipos de datos.

EJEMPLO

```
1  /*
2  **  Formas de definir variables
3  */
4  void main (void)
5  {
6
7      int    w,x,y,z;    /*Declaración de variables de tipo entero*/
8      float  a,b,c,d;    /*Declaración de variables de tipo real */
9      char   Ca1,Ca2;    /*Declaración de variables de tipo caracter*/
10
11 }
```

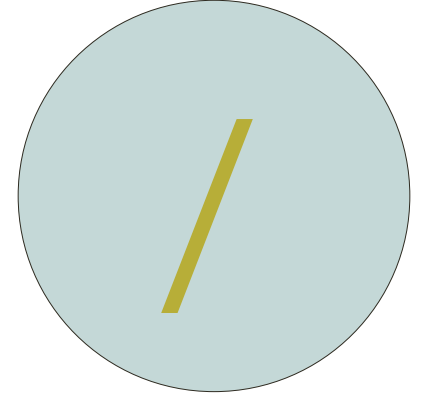
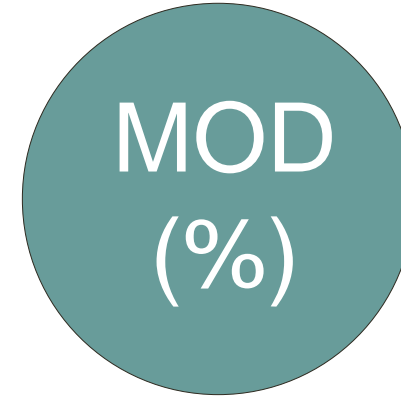
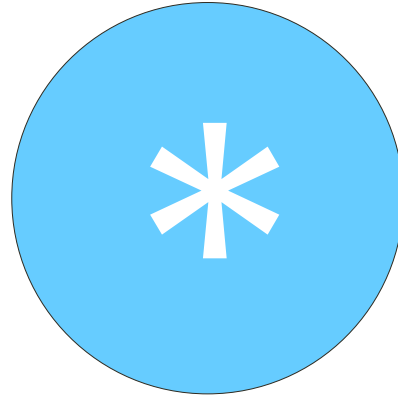
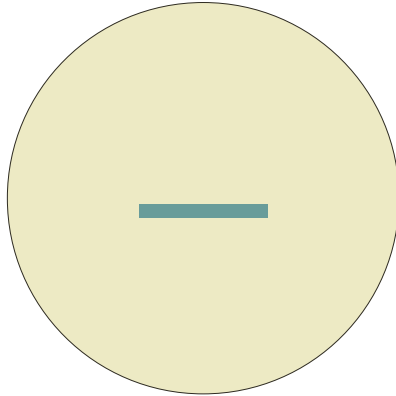
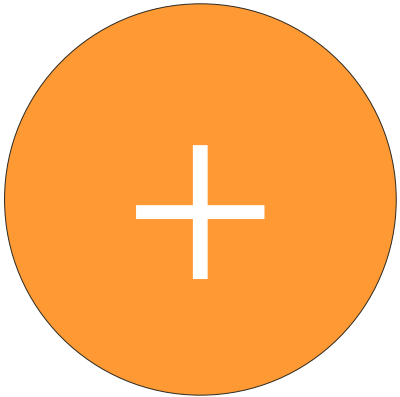
OPERADORES

Del latín opus que significa trabajar, los operadores laboran o ayudan para realizar una operación.

Los operadores más frecuentes en la programación son:

- Lógicos
- Aritméticos
- Relacionales

OPERADORES ARITMÉTICOS

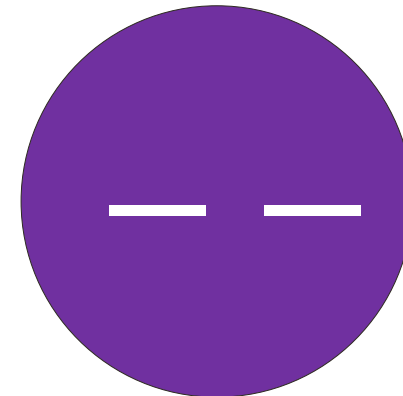
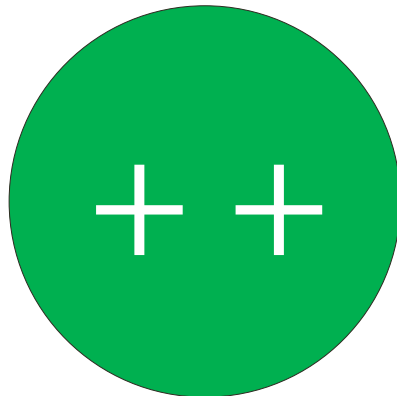
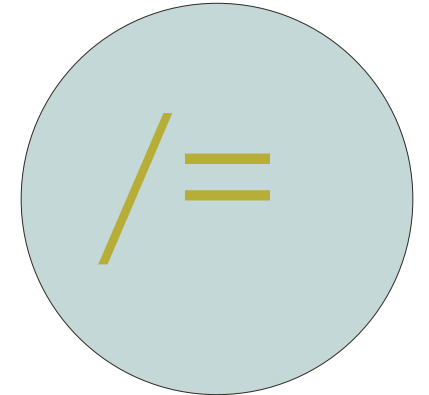
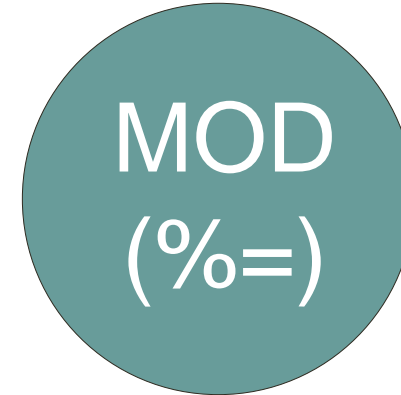
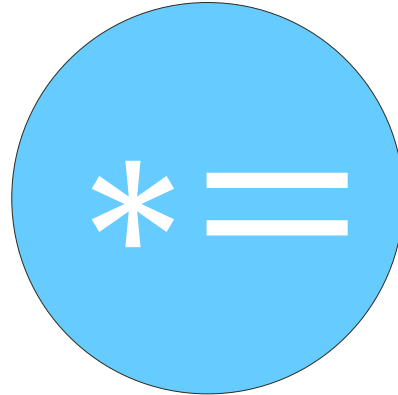
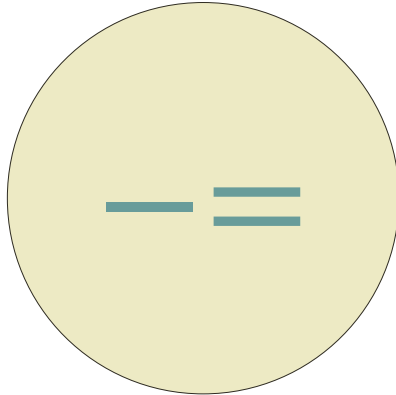
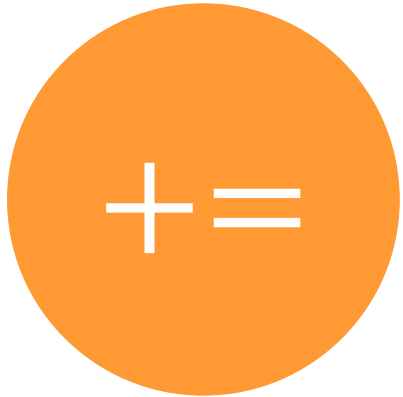


EJEMPLO

```
1 /*
2 ** Ejemplo de operadores aritméticos
3 **
4 */
```

5	//Operador aritmético	Operación	Ejemplo	Resultado
6	+	Suma	s= 4+3;	s=7
7	-	Resta	r=4-3;	r=1
8	*	Multiplicación	m=4*3;	m=12
9	/	División	d=4/3;	d=1
10	%	Módulo o residuo	mod=4%3;	mod=1

OPERADORES ARITMÉTICOS SIMPLIFICADOS, DE INCREMENTO Y DECREMENTO



EJEMPLO

1	/*				
2	**	Ejemplo de operadores aritméticos simplificados			
3	*	a,b,x & y son números enteros			
4	*/				
5	//	Operador aritmético	Forma simplificada	Ejemplos	Equivalencia
6	//		de uso		Resultado
7	+	+=		x=3;	x=3
8	+			y=4;	y=4
9	+			x+=5;	x=8
10	+			x+=y;	x=12
11					
12	-	--		a=10;	a=10
13	-			b=2;	b=2
14	-			a-=3;	a=7
15	-			a-=b;	a=5
16					
17					
18	*	*=		x=5;	x=5
19	*			y=6;	y=6
20	*			x*=4;	x=20
21	*			x*=y;	x=120

CONTINUACIÓN DEL EJEMPLO

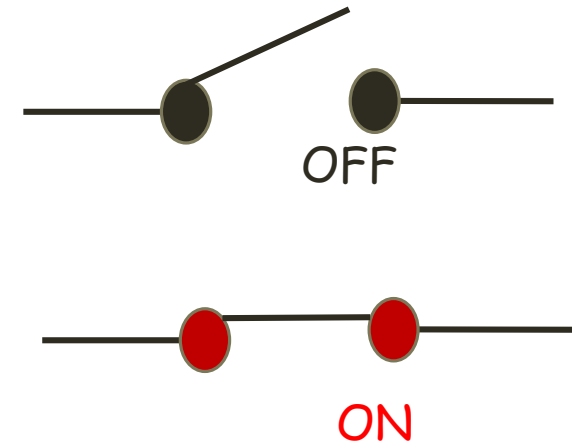
1	/*				
2	**	<i>Ejemplo de operadores aritméticos simplificados</i>			
3	*	<i>a,b,x & y son números enteros</i>			
4	*/				
5	//	<i>Operador aritmético</i>	<i>Forma simplificada</i>	<i>Ejemplos</i>	<i>Equivalencia</i>
6	//		<i>de uso</i>		<i>Resultado</i>
7	/		/=	x=25;	x=25
8	/			y=3;	y=3
9	/			x/=5;	x=5
10	/			x/=y;	x=1
11					
12	%		%=	a=8;	a=8
13	%			b=2;	b=2
14	%			a%=3;	a=2
15	%			a%=b;	a=0

EXPRESIONES LÓGICAS O BOOLEANAS

Propuestas por George Boole, están constituidas por números, constantes o variables y operadores lógicos o relacionales.

Los valores que pueden tomar son:

Falso	Verdadero
0	1
OFF	ON
LOW	HIGH

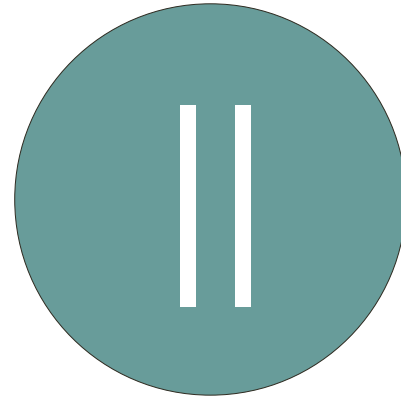
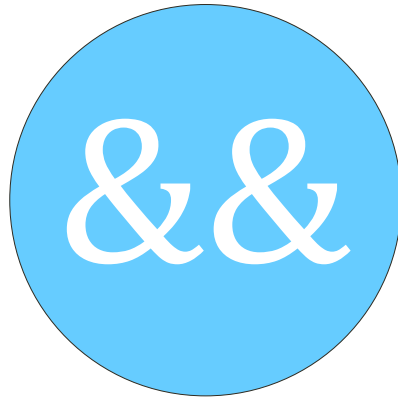


OPERADORES RELACIONALES

Los operadores relaciones se emplean para comparar dos operandos, éstos pueden ser números, caracteres, cadenas de caracteres, constantes o variables.

1	/*			
2	**	Operadores relaciones y sus usos		
3	**	prot, variable entera.		
4	*/			
5	//	Operador	Descripción	Ejemplos
6	//	relacional	del operador	Resultados
7	==	Igual a	prot='a'=='b';	prot=0
8	!=	Diferente de	prot='a'!='b';	prot=1
9	<	Menor estricto que	prot=15<45;	prot=1
10	>	Mayor estricto que	prot=25>45;	prot=0
11	<=	Menor igual que	prot=315<=550;	prot=1
12	>=	Mayor igual que	prot=221>=65;	prot=1

OPERADORES LÓGICOS



OPERADORES LÓGICOS

Ellos nos permite formular condiciones complejas a partir de condiciones simples.

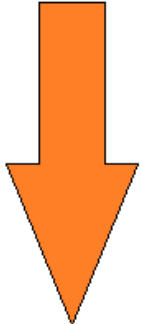
```
1  /*
2  **   Operadores lógicos
3  **   x,y son variables enteras
4  */
```

	Operador	Descripción del operador	Ejemplos	Resultados
7	!	Negación	<code>x=(! (15>45));</code> <code>/*0!=1*/</code>	<code>x=1</code>
8	!		<code>y=(!0);</code>	<code>y=1</code>
9	&&	Conjunción	<code>x=((20>10) && (13<=23));</code>	<code>x=1</code>
10	&&		<code>y=0 && 1;</code>	<code>y=0</code>
11		Disjunción	<code>x=((15>8) (8<=4));</code>	<code>x=1</code>
12			<code>y=0 1;</code>	<code>y=1</code>

PRIORIDAD DE LOS OPERADORES

1	/*	
2	**	<i>Jerarquía de los diferentes operadores</i>
3	*/	
4		Operadores
5		()
6		!, ++, --
7		*, /, %
8		+, -
9		=, !=, <, >, <=, >=
10		&&,
11		+=, -=, *=, /=, %=
12		,

Jerarquía Mayor



Menor

PROGRAMA

"Un programa es un conjunto de instrucciones que sigue la computadora para alcanzar un resultado específico.¹"

El programa se escribe en un lenguaje de programación, el cual está constituido por un conjunto de reglas sintácticas y semánticas.

Las reglas sintácticas especifican la formación de instrucciones válidas, mientras que las semánticas especifican el significado de estas instrucciones.

ESTRUCTURAS SELECTIVAS

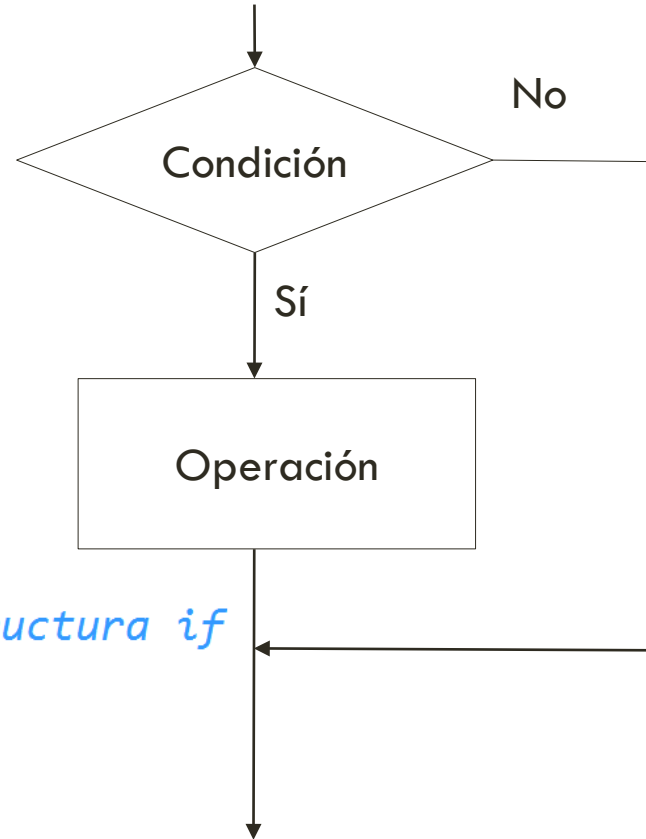
Las estructuras lógicas selectivas se encuentran en la solución algorítmica de casi todo tipo de problemas.

Estas estructuras se utilizan cuando se debe tomar una decisión en el desarrollo de la solución del problema.

Estructuras algorítmicas selectivas que estudiaremos son:

- *if*
- *if-else*
- *switch*

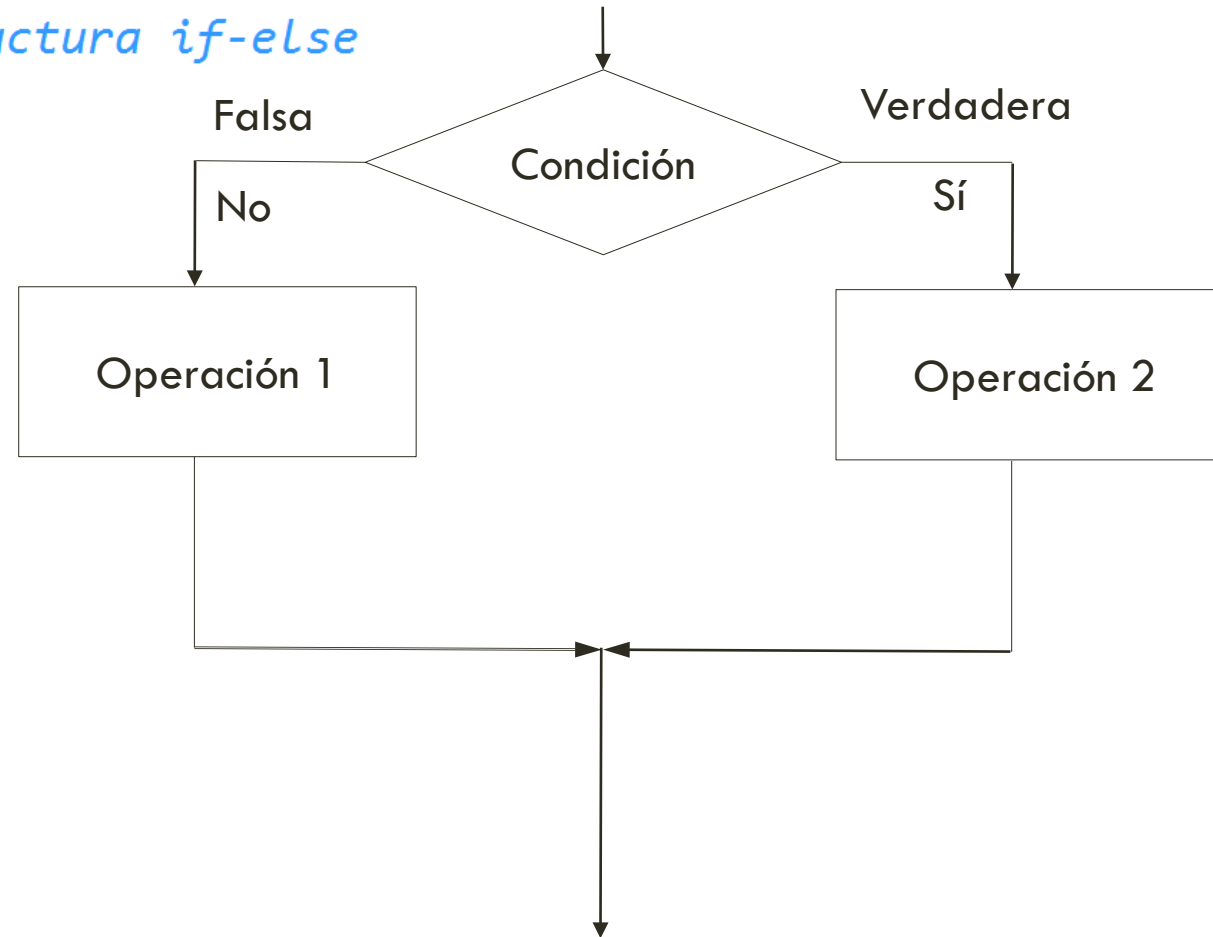
ESTRUCTURA SIMPLE IF



```
1  /*  
2  Declaración de estructura if  
3  */  
4  if(<condición>)  
5      <operación>;  
6      . . .
```

ESTRUCTURA SELECTIVA DOBLE IF-ELSE

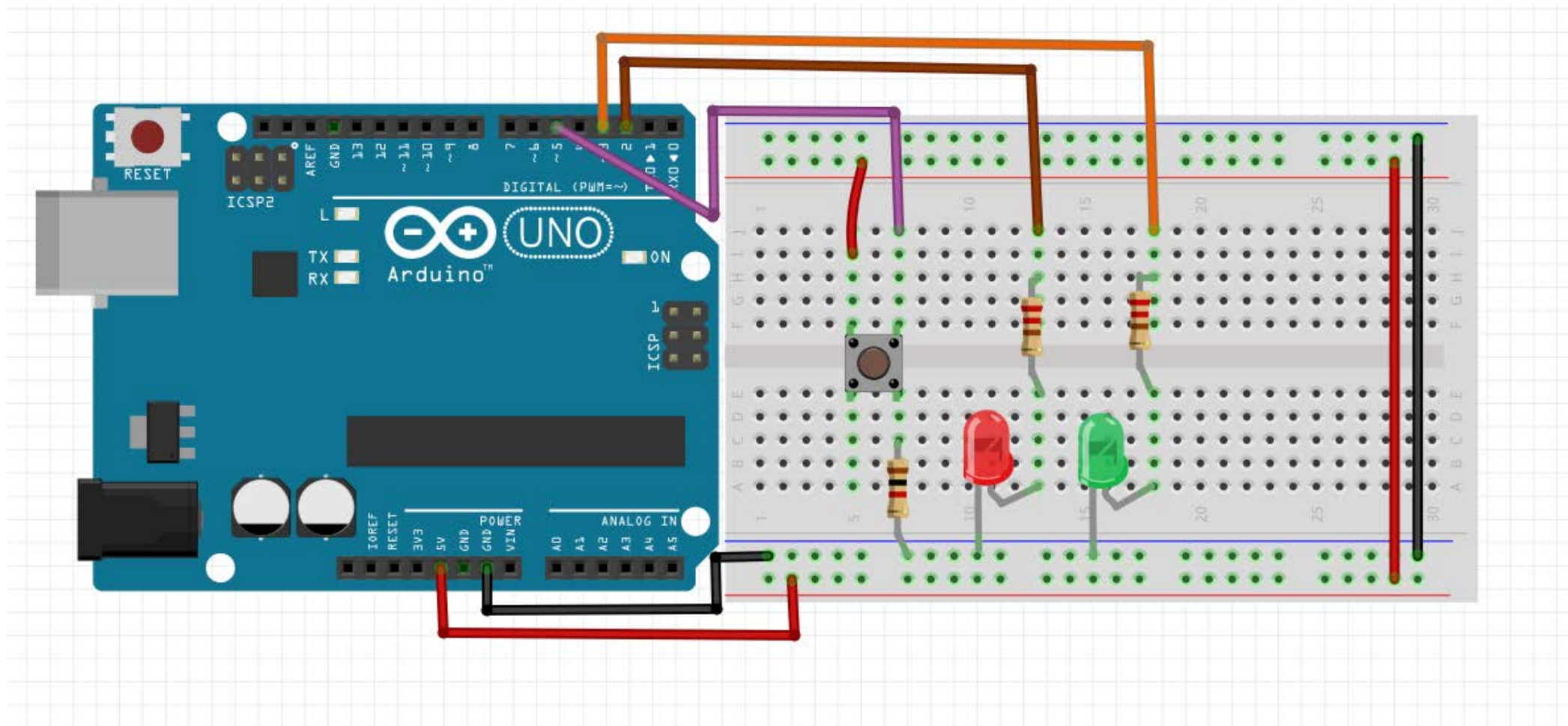
```
1  /*  
2  Declaración de estructura if-else  
3  */  
4  if(<condición>)  
5  {  
6    <operación1>;  
7  }  
8  else  
9  {  
10   <operación2>;  
11 }  
12
```



EJEMPLO CON ARDUINO

```
1  /*Código de dominio público
2  * Utilización de una sentencia de control
3  * Sentencia de selección <if>
4  */
5  const int buttonPin = 5;      // Se asigna el pin número 5 (PWM) digital al botón
6  const int ledPin = 2 ;        // Se asigna el pin número 2 (PWM) digital a el primer LED (verde)
7  const int ledPin2= 3;        //Se asigna el pin número 3 (PWM) digital a el segundo LED (rojo)
8
9  int buttonState = 0;          // Se declara una variable de tipo entero; para el estado del botón.
10
11 void setup()
12 {
13     pinMode(ledPin, OUTPUT);    //Se inicializa al pin del LED1 como una salida
14     pinMode(ledPin2, OUTPUT);   //Se inicializa al pin del LED2 como una salida
15     pinMode(buttonPin, INPUT);  //Se inicializa al pin del botón como una entrada
16 }
17
18 void loop()
19 {
20     buttonState = digitalRead(buttonPin); //Lee el valor del estado del botón (si está o no apretado)
21     if (buttonState == HIGH)              // Verifica si el botón está apretado
22     {                                     //Si está apretado realiza lo siguiente:
23         digitalWrite(ledPin,HIGH);         //Enciende el segundo LED(verde)
24         digitalWrite(ledPin2,LOW);         // Se apaga el segundo LED(rojo)
25     }
26     else                                 //En caso contrario
27     {
28         digitalWrite(ledPin,LOW);          //Se apaga el primer LED(verde)
29         digitalWrite(ledPin2,HIGH);        //Enciende el segundo LED
30     }
31 }
```

CIRCUITO DE SELECCIÓN (IF-ELSE)



¿QUÉ ES UNA FUNCIÓN?

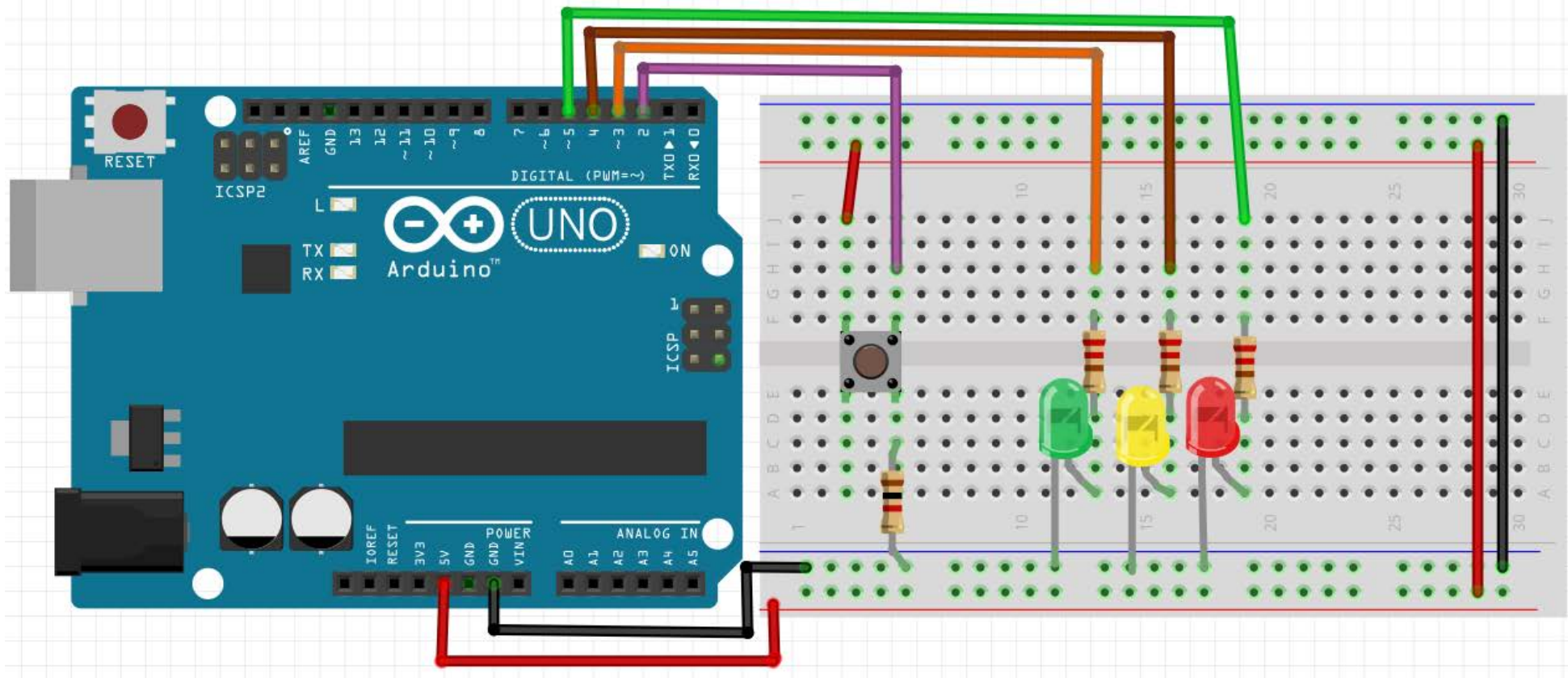
Una función o subrutina, desde el punto de vista de programación es una herramienta para resolver problemas complejos y de gran tamaño en procesos o subrutinas más pequeñas que se encuentran en una categoría menor al problema inicial, o simplemente como una reducción de problemas. <<"Divide y vencerás">>

```
1  /*
2  **Estructura de declaración de una función
3  */
4  <valor devuelto><nombre de la función>(Parámetros)
5  {
6      instrucciones; /*Cuerpo de la función*/
7  }
```


EJEMPLO

```
1  /*
2  ** Calculadora de dos número enteros con operaciones básicas con uso de funciones
3  ** @author: J.C. Rivera
4  ** @version: 24/02/2016
5  */
6  #include <stdio.h> /*Biblioteca estándar de variables de entrada y salida */
7
8  int Sum(int x, int y) /* Prototipo de función. */
9  {
10     int suma= x+y;
11     return suma;
12 }
13
14 int Rest(int x, int y)/* Prototipo de función. */
15 {
16     int resta= x-y;
17     return resta;
18 }
19
20 int Mult(int x, int y)/* Prototipo de función. */
21 {
22     int mult= x*y;
23     return mult;
24 }
25
26 int Div(int x, int y)/* Prototipo de función. */
27 {
28     int div;
29     if(y==0)
30         return 0;
31     else
32         div =x/y;
33     return div;
34 }
```

EJEMPLO CON ARDUINO UTILIZANDO INTERRUPCIONES



¿QUÉ ES UNA INTERRUPCIÓN?

Una interrupción nos permite responder a eventos "externos" mientras se realiza otra cosa.

El mecanismo de interrupción nos brinda una manera de evitar perder tiempo en el procesador, evitando rutinas cíclicas. <<O sea, evitar el ciclo void loop>>

Por ejemplo: Si usted cocina algo (arroz) y tiene que esperar 20 minutos para que se cuece, en lugar de esperar podemos poner a un contador o minuterero para que nos avise o "interrumpa".



INTERRUPCIONES

En el programa se utilizará una variable con identificador de tipo volátil, así como las de tipo constante, entera o flotante.

Sólo que está variable es cargada desde la RAM (Random Access Memory) en lugar del espacio del almacenamiento de memoria del registro.

Supondremos que nuestra ("memoria") o lugar de almacenamiento es un triángulo como el de la siguiente diapositiva; y siempre se accede a ella por los "pisos":

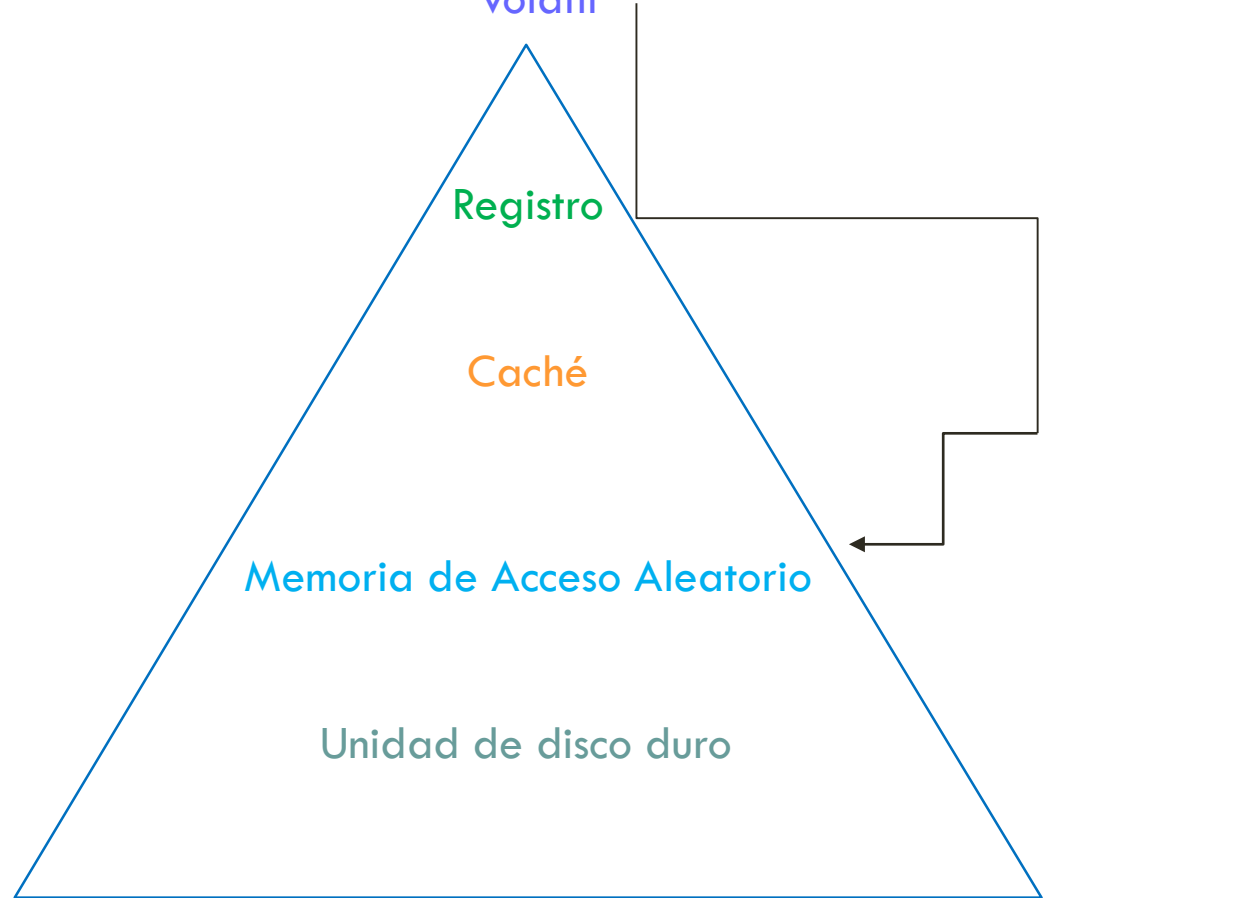
- Registros
- Caché
- RAM
- Unidad de Disco Duro



El Tiempo de acceso incrementa

Variable
de tipo
volátil

La cantidad de almacenamiento incrementa



RECOMENDACIONES EN INTERRUPCIONES

Cuando se escribe una Interrupt Service Routine (ISR):

- *Realizarlas lo más cortas posibles.*
- *No usar delay(); .*
- *No utilizar Serial.print(); .*
- *Declarar variables volátiles dentro del código (main) principal.*
- *No intentar conmutar las interrupciones (off or on).*
- *No hay variables de entrada o valores de retorno. Todos los cambios deben ser hechos en las variables globales.*

CÓDIGO

```
1  /*
2   * Ejemplo 2.-Cambio de estado a través de interrupciones
3   * 29/04/2016
4   */
5  // Las constantes no cambian su valor
6  // Se asignan Los números de pines a las constantes:
7  const byte buttonPin = 2;           // Se asigna el número del botón al pin 2
8  const byte ledPin = 4;              // Se asigna el número de pin al LED #1
9  const byte ledPin2 = 5;            // Se asigna el número de pin al LED #2
10 const byte ledPin3 = 6;            // Se asigna el número de pin al LED #3
11 // Las variables que cambiarán
12 volatile byte buttonState = 0;      // Variable de tipo volatile para leer el estado del "pushbutton" (botón)
13 void CambioEstado()                // Función que nos permite cambiar el estado
14 {
15     buttonState = digitalRead(buttonPin); // Lee el estado del botón
16     buttonState=!buttonState;           // Asigna el estado opuesto del botón
17 }
18
19 void setup()
20 {
21     pinMode(ledPin, OUTPUT);           // Establece ledpin como salida
22     pinMode(ledPin2, OUTPUT);          // Establece ledpin como salida
23     pinMode(ledPin3, OUTPUT);          // Establece ledpin como salida
24     // Se inicializa el pushbutton como entrada
25     pinMode(buttonPin, INPUT);
26     attachInterrupt(0, CambioEstado, CHANGE);
27 }
```

```

28 void loop()
29 {
30   while(!buttonState){           //Mientras el botón no esté presionado ejecutará el siguiente proceso
31     digitalWrite(ledPin3, HIGH); // Se enciende el ledPin3
32     delay(300);                  // Espera 0.3 segundos
33     digitalWrite(ledPin3, LOW);  // Apaga el ledPin3
34     delay(300);                  // Espera 0.3 segundos
35     digitalWrite(ledPin2, HIGH); // Se enciende el ledPin2
36     delay(300);                  // Espera 0.3 segundos
37     digitalWrite(ledPin2, LOW);  // Apaga el ledPin3
38     delay(300);                  // Espera 0.3 segundos
39     digitalWrite(ledPin, HIGH);  // Se enciende el ledPin
40     delay(300);                  // Espera 0.3 segundos
41     digitalWrite(ledPin, LOW);   // Apaga el ledPin
42     delay(300);                  // Espera 0.3 segundos
43   }
44   buttonState = digitalRead(buttonPin); // Lee el estado del botón
45   if(buttonState==0)                  // Si el botón es igual 0, se ejecuta el siguiente proceso.
46   {
47     digitalWrite(ledPin2, HIGH); // Se enciende el ledPin2
48     delay(3000);                  // Espera 3 segundos
49     digitalWrite(ledPin2, LOW);   // Apaga el ledPin2
50     delay(300);                  // Espera 0.3 segundos
51     digitalWrite(ledPin, HIGH);   // Se enciende el ledPin
52     delay(3000);                  // Espera 0.3 segundos
53     digitalWrite(ledPin, LOW);    // Apaga el LED
54     delay(300);                  // Espera 0.3 segundos
55   }
56 }

```


REFERENCIAS

- [1] Cairó, Osvaldo., Fundamentos de programación. Piensa en C, Pearson Educación, 2006
- [2] Nick Gammon, “Interruptions”, Internet: <http://gammon.com.au/interrupts> 8/01/2012 consultado [5/5/20116]
- [3] Referencias del lenguaje en arduino, Internet: <https://www.arduino.cc/en/Reference/HomePage> [Dominio Público]
- [4] Sánchez J., P. Cantos M., Microcontroller. High Performance Systems and Programming, CRC Press Taylor and Francis Group, 2014 p.p.[147]