

CURSOS  
INTERSEMESTRALES



PROTECO

# Arduino

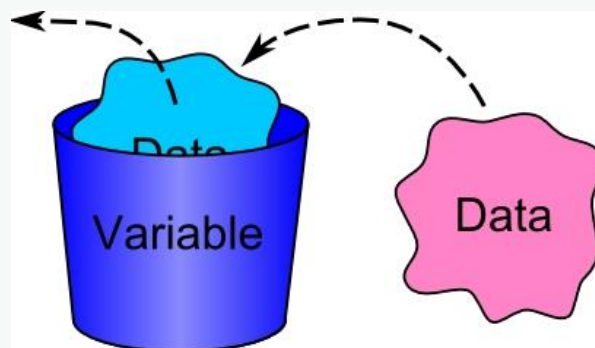
Introducción a la  
programación



09/01/18

# Variables

Área reservada en la memoria principal del microprocesador



Variables globales y variables locales

# Nombrar variables

- Se permiten dígitos, letras y guión bajo
- El primer carácter debe ser una letra o guión bajo
- No usar palabras reservadas

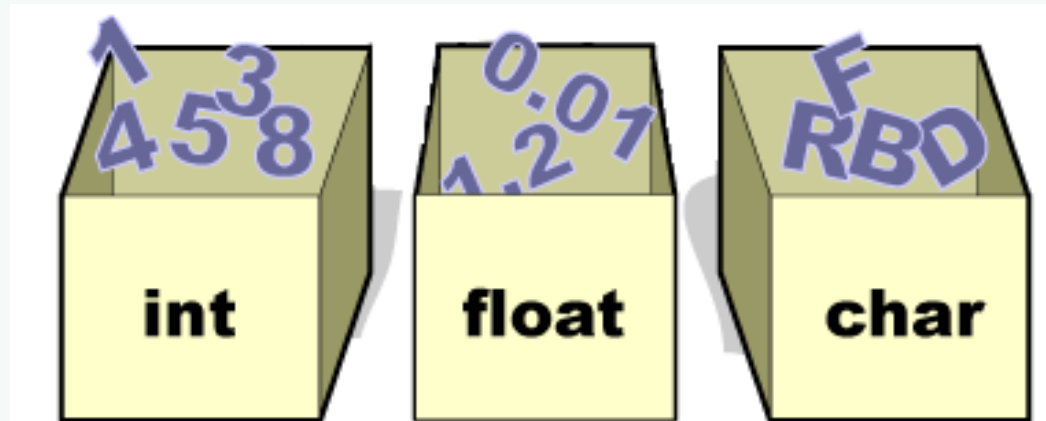


# Nombrar variables

Identificador	Explicación
<i>3id</i>	<i>El primer carácter debe ser una letra</i>
<i>Alumno#</i>	<i>El carácter # no es permitido</i>
<i>Codigo alumno</i>	<i>El espacio en blanco no es permitido</i>
<i>Codigo-alumno</i>	<i>El carácter - no es permitido</i>
<i>"alumno"</i>	<i>El carácter " no es permitido</i>



# Típos de datos



# boolean

- Almacena un 0 o un 1, HIGH o LOW, true o false
- Ocupa 1 byte de memoria

```
boolean boton1 = false;  
boolean boton2 = true;  
boolean estado1 = HIGH;  
boolean estado2 = 0;  
boolean estado3 = 1;
```



# byte

- Almacena un valor numérico de 1 byte
- Sin decimales
- Entre 0 y 255

```
byte motor1 = 0;  
byte motor2 = 255;  
byte motor3 = 200;  
byte motor4 = 178;
```



# int

- Almacena números enteros de 2 byte
- Rango entre 32,767 a -32,768

```
int tiempo = 2000;  
int ledPin = 13;  
int numero = -400;  
int valor1 = -3000;  
int i = 0;
```





# unsigned int

- Entero sin signo de 2 byte
- Rango de 0 a 65,535

```
unsigned int ledPin = 13;
```

```
unsigned int x;  
x = 0;  
x = x - 1;    // x = 65535  
x = x + 1;    // x = 0
```



# long

- Valores enteros de 4 bytes
- Rango de -2,147,483,648 a 2,147,483,647

```
long velocidad = 1004003;
```



# unsigned long

- Valores enteros sin signo de 4 bytes
- Rango de 0 a 4,294,967,295

```
unsigned long tiempo;
```



PROTECO

# float

- Valores real de 4 bytes
- Rango de  $3.4028235E+38$  a  $-3.4028235E+38$

```
float gravedad = 9.81;
```



# char

- Valor de carácter de 1 bytes
- Los caracteres literales se escriben con comillas simples
- Se almacenan como números

```
char letra = 'A';  
char letra = 65;
```

```
letra = letra + 1; //letra = B
```



# ASCII

DEC Value	Character
0	null
1	
2	
3	
4	
5	
6	
7	
8	
9	tab
10	line feed
11	
12	
13	carriage return
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	
31	

DEC Value	Character
32	space
33	!
34	"
35	#
36	\$
37	%
38	&
39	'
40	(
41	)
42	*
43	+
44	,
45	-
46	.
47	/
48	0
49	1
50	2
51	3
52	4
53	5
54	6
55	7
56	8
57	9
58	:
59	;
60	<
61	=
62	>
63	?

DEC Value	Character
64	@
65	A
66	B
67	C
68	D
69	E
70	F
71	G
72	H
73	I
74	J
75	K
76	L
77	M
78	N
79	O
80	P
81	Q
82	R
83	S
84	T
85	U
86	V
87	W
88	X
89	Y
90	Z
91	[
92	\
93	]
94	^
95	_

DEC Value	Character
96	`
97	a
98	b
99	c
100	d
101	e
102	f
103	g
104	h
105	i
106	j
107	k
108	l
109	m
110	n
111	o
112	p
113	q
114	r
115	s
116	t
117	u
118	v
119	w
120	x
121	y
122	z
123	{
124	
125	}
126	~
127	



PROTECO



# array

- Los arreglos almacenan un grupo de datos del mismo tipo
- Son colecciones de variables a las que se acceden mediante un índice.

## Vectores

### Varios Valores

Vector	1	2	4	5	6	1	2	3	4
Dimension									

	Valor								
									
Vector	1	2	4	5	6	1	2	3	4
Dimension									
	0	1	2	3	4	5	6	7	8
									
	Indice								



# array

```
int pines[4];  
int leds[] = {4, 5, 6, 3};  
int sensoresValor[6] = {2, 4, -6, 3, 1, 2};  
int saludo[5] = "hola";  
char  
int y = leds[0]; // y = 4  
int x = leds[1]; // x = 5
```





# String

- Almacena una cadena de caracteres
- Se puede declarar con el tipo de dato String
- Se puede declarar con el tipo de dato char
- Es un arreglo de caracteres

```
String Str1 = "arduino";  
char Str2[15];  
char Str3[8] = {'a','r','d','u','i','n','o'};  
char Str4[8] = {'a','r','d','u','i','n','o','\0'};  
char Str5[] = "arduino";  
char Str6[8] = "arduino";  
char Str7[15] = "arduino";
```



# Tipos de datos

## Data Types

String()  
array  
boolean  
byte  
char  
double  
float  
int  
long  
short  
string  
unsigned char  
unsigned int  
unsigned long  
void  
word



PROTECO

# Sintaxis

- ; utilizado para la separación de instrucciones
- {} definen principio y final de un bloque
- // comentario de línea única
- /\* \*/ comentario multilínea



# Operadores

- Operadores aritméticos
- Operadores de comparación o relacionales
- Operadores lógicos básicos



PROTECO

# Operadores aritméticos

Operador aritmético	Notación	Ejemplo
Suma o adición	+	<code>y=y+2;</code>
Resta o sustracción	-	<code>x=x-3;</code>
Multiplicación	*	<code>z=y*4;</code>
División	/	<code>w=w/5;</code>
División en módulo	% sólo aplica a datos de tipo entero	<code>y=15%4</code>
Incremento unitario	++ incrementa en uno a su operando	<code>x++;</code>
Decremento unitario	-- decrementa en uno a su operando	<code>x--;</code>



# Partes de la división

The diagram illustrates the components of a long division problem. It shows the division of 965 by 5. The quotient is 193, and the remainder is 15. The dividend is 965, and the divisor is 5. The steps of the division are shown: 5 goes into 9 one time (1), 5 goes into 6 one time (1), and 5 goes into 5 one time (1). The remainder is 15.

$$\begin{array}{r} 193 \\ 5 \overline{) 965} \\ \underline{-5} \phantom{0} \\ 46 \phantom{0} \\ \underline{-45} \phantom{0} \\ 15 \end{array}$$

Labels and arrows:

- cociente** (quotient) points to the number 193.
- dividendo** (dividend) points to the number 965.
- divisor** (divisor) points to the number 5.
- residuo** (remainder) points to the number 15.



# Operador ++ y --

- $X++$  incrementa  $x$  y retorna antiguo valor
- $++X$  incrementa  $x$  y retorna nuevo valor
- $X--$  decrementa  $x$  y retorna antiguo valor
- $--x$  decrementa  $x$  y retorna nuevo valor



# Operador ++ y --

```
x = 5;  
y = x++;  
// x = 6  
// y = 5  
z = ++x;  
// x = 7  
// z = 7
```





# Operadores relacionales

Notación	Descripción
$x == y$	$x$ 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 $y$
$x \geq y$	$x$ es mayor o igual que $y$
$x \leq y$	$x$ es menor o igual que $y$



# Operadores lógicos

Operador	Notación
<b>and</b>	<b>&amp;&amp;</b>
<b>or</b>	<b>  </b>
<b>not</b>	<b>!</b>



# Prioridad de operadores

( ) [ ]	Llamada a una función e índice de un array
+ - ++ -- ! (<tipo>)	Signo más, signo menos, incremento, decremento, negación y conversión de tipo
* / %	Multiplicación, división, módulo
+ -	Suma y resta
< <= > >=	Menor que, menor o igual que, mayor que, mayor o igual que
== !=	Igual que y distinto que
&&	Conjunción
	Disyunción
= += -= *= /= %=	Operadores de asignación



# Estructura general

```
void setup() {  
  // put your setup code here, to run once:  
}  
  
void loop() {  
  // put your main code here, to run repeatedly:  
}
```



PROTECO

# Estructura general

- Sección de declaración de variables
- Se declaran variables, objetos, funciones y estructuras
- 
- Función setup()
- Realiza funciones de inicialización de periféricos, comunicaciones, variables, etc.
- 
- Función loop()
- Se repite su ejecución hasta que se desconecte el arduino.
- 

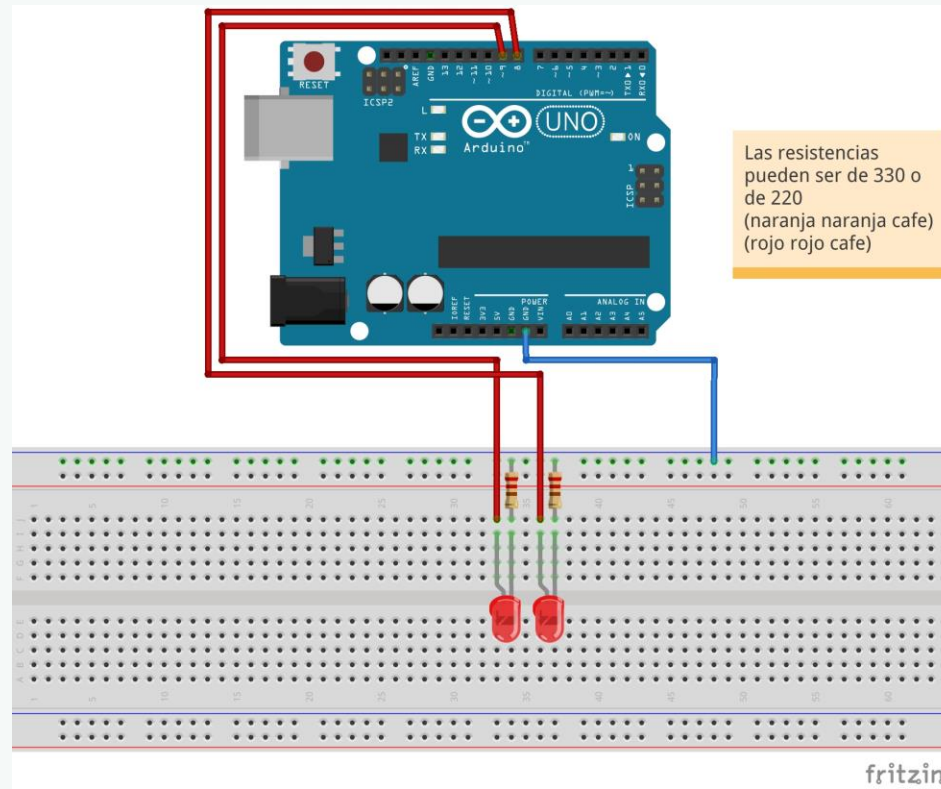


# Estructuras de control

- Modifican el flujo de ejecución de las instrucciones de un programa
- if
- if..else
- switch
- while
- do..while
- for



# Cableado



# if

- Después de evaluar la condición, si el resultado es verdadero, realiza la o las instrucciones definidas.

1.  
`if (condición 1 )`  
`instrucción 1;`
2.  
`if (condición 1 )`  
`{`  
`instrucción 1;`  
`instrucción 2;`  
`instrucción 3;`  
`}`





# if

```
int x = 1; //Asignamos un entero a la variable x
int y = 2; //Asignamos un entero a la variable y

int pinLed1 = 8; //Estas variables van a facilitar el uso en las funciones
int pinLed2 = 9; //Estamos asignando los numeros que van a indicar el número de pin para los leds

void setup() {

  pinMode(pinLed1,OUTPUT); //Declaramos cada pin como salida
  pinMode(pinLed2,OUTPUT); // pinMode( número de pin, OUTPUT o INPUT )
}

void loop() {

  if(y > x){ //Damos la condición y entre llaves las instrucciones a realizar

    digitalWrite(pinLed1,HIGH); //Con esta función estamos mandando voltaje por los pines
    digitalWrite(pinLed2,HIGH); //digitalWrite( número de pin, HIGH o LOW)

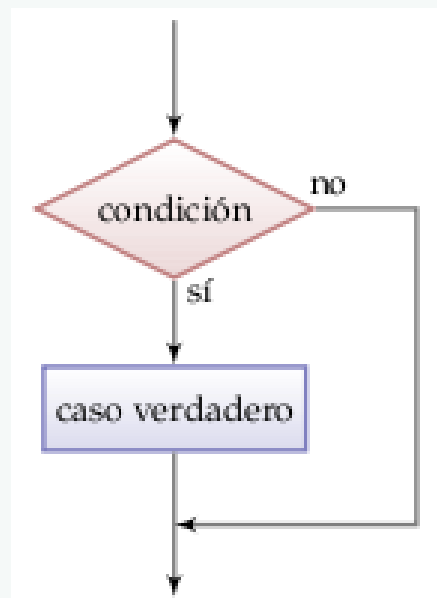
  }

}
```



PROTECO

if



PROTECO

# if..else

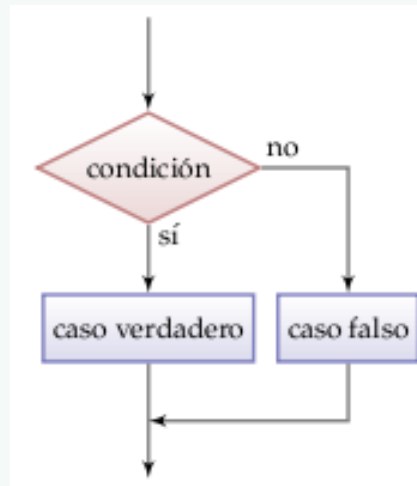
- Permite evaluar una condición y elegir entre dos opciones. Si es verdadera, ejecuta las instrucciones a continuación del if, si es falsa, ejecuta las instrucciones a continuación del else. No ambas.

```
1.
  if (condición )
    instrucción;
  else
    instrucción;

2.
  if (condición )
  {
    instrucción 1;
    instrucción 2;
  }
  else
  {
    instrucción 3;
    instrucción 4;
  }
```



# if..else



PROTECO

# if..else

```
int x = 1;
int y = 2;

int pinLed1 = 8; //Declaramos los pines
int pinLed2 = 9;

void setup() {

  pinMode(pinLed1,OUTPUT); //Declaramos cada pin como salida
  pinMode(pinLed2,OUTPUT); // pinMode( número de pin, OUTPUT o INPUT )
}

void loop() {

  if(y > x){

    digitalWrite(pinLed1,HIGH); //Con esta función estamos mandando voltaje por los pines
    digitalWrite(pinLed2,LOW); //digitalWrite( número de pin, HIGH o LOW)

  }else{//Se ejecuta si la condición es falsa

    digitalWrite(pinLed1,LOW);
    digitalWrite(pinLed2,HIGH);

  }

}
```



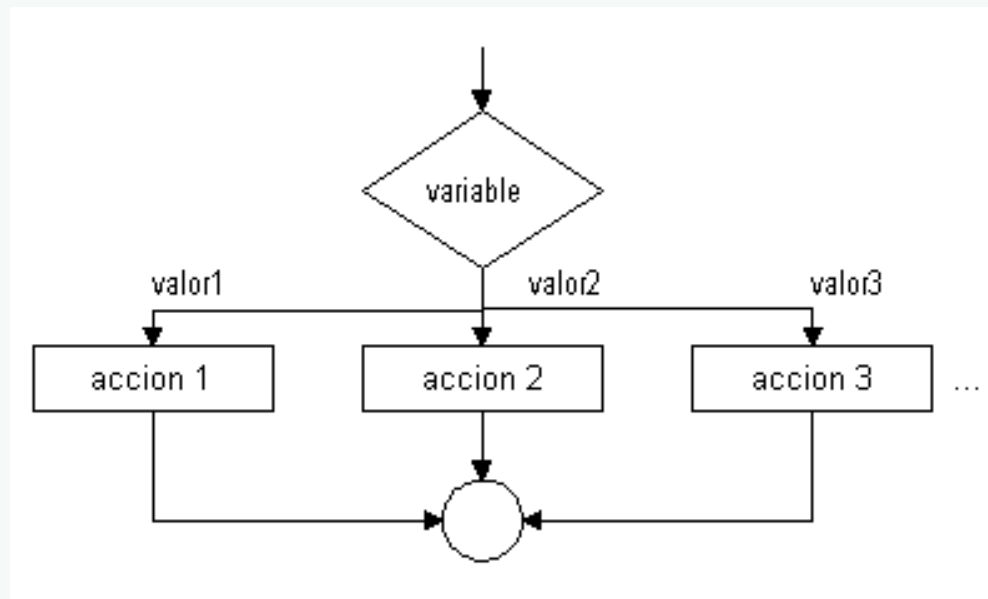
# switch

- Permite elegir entre dos o más opciones, evalúa la expresión que se encuentra dentro del paréntesis y el resultado se compara con valores alternativos.
- La expresión debe ser de tipo entero o carácter

```
switch (expresion)
{
    case 1: instruccion 1;
            break;
    case 2: instruccion 2;
            break;
    .
    .
    case n: instruccion n;
            break;
    default:
            instruccion n +1;
}
```



# switch



# switch

```
int x = 1;

int pinLed1 = 8;
int pinLed2 = 9;

void setup() {
  pinMode(pinLed1, OUTPUT);
  pinMode(pinLed2, OUTPUT);
}

void loop() {
  switch(x){ //x es la variable que va a evaluar

    case 1: //se ejecuta si x = 1
      digitalWrite(pinLed1, HIGH);
      digitalWrite(pinLed2, LOW);
      break; //si no pusiesemos break, la evaluación continuaría

    case 2: //se ejecuta si x = 2
      digitalWrite(pinLed1, LOW);
      digitalWrite(pinLed2, HIGH);
      break;
      //Es importante dejar un espacio despues de "case"
    case 3: //se ejecuta si x = 3
      digitalWrite(pinLed1, HIGH);
      digitalWrite(pinLed2, HIGH);
      break;
      //El programador decide si poner o no un default
    default: //se ejecuta si x = cualquier otro valor
      digitalWrite(pinLed1, LOW);
      digitalWrite(pinLed2, LOW);
  }
}
```





# Temporización

- delay() - detiene el programa la cantidad de tiempo especificado (milisegundos)
- 
- millis() - da el número de milisegundos desde que inició el programa
- 
- delayMicroseconds()
- 
- micros()
- 



# Temporización

- `int pinLed1 = 8;`
- `int pinLed2 = 9;`

```
void setup() {  
  
  pinMode(pinLed1, OUTPUT);  
  pinMode(pinLed2, OUTPUT);  
  
}  
  
void loop() {  
  
  digitalWrite(pinLed1, HIGH);  
  digitalWrite(pinLed2, HIGH);  
  delay(1000); //enciende los leds y espera un segundo  
  digitalWrite(pinLed1, LOW);  
  digitalWrite(pinLed2, LOW);  
  delay(1000); //apaga los leds y espera un segundo  
  
}
```



# Temporización

- ```
int pinLed1 = 8;
```
- ```
int pinLed2 = 9;
```
- ```
void setup() {  
  
  pinMode(pinLed1, OUTPUT);  
  pinMode(pinLed2, OUTPUT);  
  
}
```
- ```
void loop() {  
  
  digitalWrite(pinLed1, HIGH);  
  digitalWrite(pinLed2, HIGH);  
  delay(millis()); //enciende los leds y espera  
                    //la cantidad de tiempo transcurrido  
  digitalWrite(pinLed1, LOW);  
  digitalWrite(pinLed2, LOW);  
  delay(millis()); //apaga los leds y espera  
                    //la cantidad de tiempo transcurrido  
  
}
```



# while

- En esta estructura permite repetir una serie de instrucciones mientras que se cumpla una condición

```
expression 1;  
while (expression 2)  
{  
  instruccion 1;  
  expresion 3;  
}
```

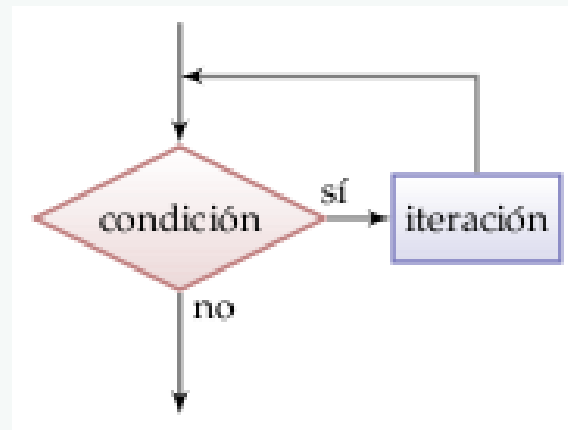
donde:

- *expression 1* siempre será el valor de inicio de la variable de control (asignación).
- *expression 2* es la condición booleana.
- *expression 3* es la forma en que cambia la variable de control (asignación).
- *instruccion 1* instrucciones a ejecutar

- En arduino, no es obligatorio una expresión 3



# while



# while

```
int x = 5;
int y = 0;

int pinLed1 = 8;
int pinLed2 = 9;

void setup() {

  pinMode(pinLed1, OUTPUT);
  pinMode(pinLed2, OUTPUT);

}

void loop() {
  while(x > y){//el ciclo se va a repetir hasta que sea falso

    digitalWrite(pinLed1, HIGH);
    digitalWrite(pinLed2, LOW);
    delay(500);
    digitalWrite(pinLed1, LOW);
    digitalWrite(pinLed2, HIGH);
    delay(500);
    x--; // x = x - 1

  }

  digitalWrite(pinLed1, LOW);
  digitalWrite(pinLed2, LOW);
}
```



# do..while

- En esta estructura, la condición del ciclo se prueba al final del mismo.
- En esta estructura es indispensable escribir las llaves

- 
- 
- 
- 
- 
- 
- 
- 
- 

```
expresion 1;  
do  
{  
instruccion 1;  
expresion 3;  
} while(expresion 2);
```

donde:

*expresion 1* es la asignación de inicio.

*expresion 3* es la asignación de variación.

*instruccion 1* son las instrucciones a ejecutar.

*expresion 2* debe ser una expresión booleana.

- En arduino no es obligatorio una expresión 3



# do..while

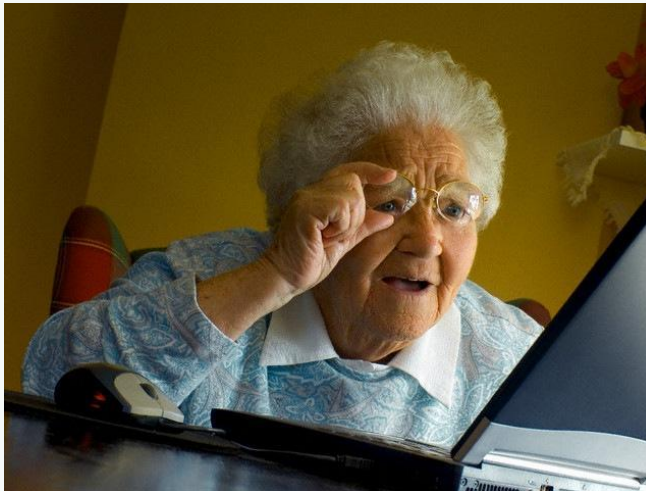


PROTECO



# do..while

- ¿Por qué no se apaga?



```
int x = 5;
int y = 0;

int pinLed1 = 8;
int pinLed2 = 9;

void setup() {

  pinMode(pinLed1, OUTPUT);
  pinMode(pinLed2, OUTPUT);

}

void loop() {

  do
  {
    x--; // x = x - 1
    digitalWrite(pinLed1, HIGH);
    digitalWrite(pinLed2, LOW);
    delay(500);
    digitalWrite(pinLed1, LOW);
    digitalWrite(pinLed2, HIGH);
    delay(500);

  } while(x > y);

  digitalWrite(pinLed1, LOW);
  digitalWrite(pinLed2, LOW);

}
```



# do..while

- Los leds no paran porque la instrucción inicial siempre se realiza a pesar de que la condición sea falsa, al estar en la función loop(), esta se sigue repitiendo

- 



PROTECO

# do..while

```
int x = 5;
int y = 0;

int pinLed1 = 8;
int pinLed2 = 9;

void setup() {
  pinMode(pinLed1, OUTPUT);
  pinMode(pinLed2, OUTPUT);
}

void loop() {
  do
  {
    x--; // x = x - 1
    digitalWrite(pinLed1, HIGH);
    digitalWrite(pinLed2, LOW);
    delay(500);
    digitalWrite(pinLed1, LOW);
    digitalWrite(pinLed2, HIGH);
    delay(500);

  } while(x > y);
  //cuando se deje de cumplir la condición
  //empezará a realizar lo siguiente
  digitalWrite(pinLed1, LOW);
  digitalWrite(pinLed2, LOW);
  delay(2000);
}
```



PROTECO

# for

- Esta estructura se utiliza generalmente cuando la repetición está definida. Se maneja como un contador

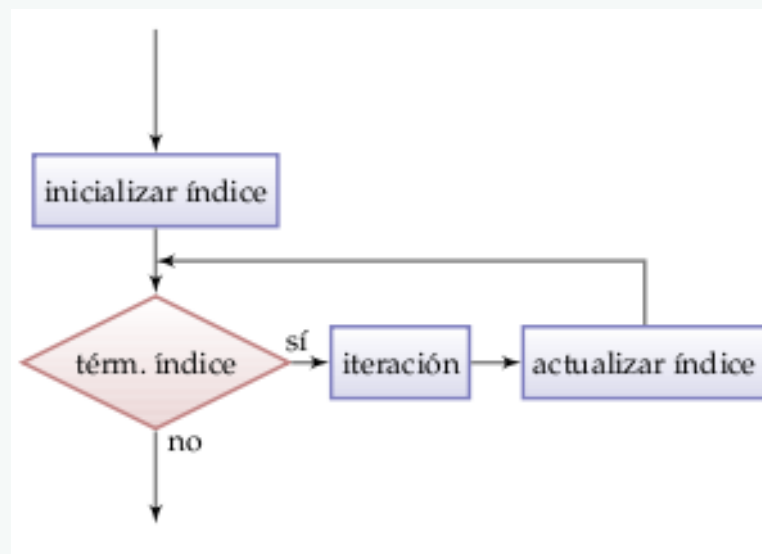
```
for ( expresion 1; expresion 2; expresion 3)  
    instruccion 1
```

donde:

<i>expresion 1</i>	es el nombre de la variable de control y su valor de inicio.
<i>expresion 2</i>	es la que evalúa la condición de control.
<i>instruccion 1</i>	son las instrucciones que han de repetirse.
<i>expresion 3</i>	instrucción de incremento o decremento de la variable de control.



# for



PROTECO

# for

```
int x = 5;

int pinLed1 = 8;
int pinLed2 = 9;

void setup() {
  pinMode(pinLed1, OUTPUT);
  pinMode(pinLed2, OUTPUT);
}

void loop() {
  for( x ; x > 0 ; x-- ){//for(variable ; condición ; cambio de la condición)

    digitalWrite(pinLed1, HIGH);
    digitalWrite(pinLed2, HIGH);
    delay(500);
    digitalWrite(pinLed1, LOW);
    digitalWrite(pinLed2, LOW);
    delay(500);
  }
}
```



# for



```
int pinLed1 = 8;
int pinLed2 = 9;

void setup() {
  pinMode(pinLed1,OUTPUT);
  pinMode(pinLed2,OUTPUT);
}

void loop() {
  for( int i = 5 ; i > 0 ; i-- ){//for(variable ; condición ; cambio de la condición)

    digitalWrite(pinLed1,HIGH);
    digitalWrite(pinLed2,HIGH);
    delay(500);
    digitalWrite(pinLed1,LOW);
    digitalWrite(pinLed2,LOW);
    delay(500);
  }
}
```



PROTECO

# Funciones

- Las funciones permiten crear pequeños programas de código que realizan una tarea y vuelven a la zona de código desde donde fueron llamadas.
- 
- 
- Son útiles cuando hay que realizar la misma acción varias veces en un mismo programa





# Funciones

- **int** en este caso indica el tipo de dato devuelto
- **int x** e **int y** son los parámetros que hay que pasar y tienen que ser de tipo entero
- **resultado** es el valor que devuelve la función

```
int FuncionMultiplicar (int x, int y) {  
    int resultado;  
    resultado = x * y;  
    return resultado;  
}
```



# Funciones

- Para llamar la función se le envía los parámetros del tipo de datos que la función espera.

- 
- 

```
void loop(){  
    int i = 2;  
    int j = 3;  
    int k;  
    k = FuncionMultiplicar (i, j); // k ahora contiene 6  
}
```

- 
- La función debe ser declarada fuera de cualquier otra función. Por encima o por debajo de loop()



# Void

- Cuando una función no va a regresar información, se declara con void como tipo de dato.

- 
- 

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



# Funciones

- ```
int pinLed1 = 8;
```
- ```
int pinLed2 = 9;
```
- ```
void setup() {  
  pinMode(pinLed1, OUTPUT);  
  pinMode(pinLed2, OUTPUT);  
}  
  
void prenderSegundos (int x){//declaramos la función  
  digitalWrite(pinLed1,HIGH);  
  digitalWrite(pinLed2,HIGH);  
  delay(x*1000);  
  digitalWrite(pinLed1,LOW);  
  digitalWrite(pinLed2,LOW);  
  delay(x*1000);  
}  
  
void loop() {  
  prenderSegundos(1);//llamamos a la función  
    // utilizamos a 1 como parámetro  
}
```



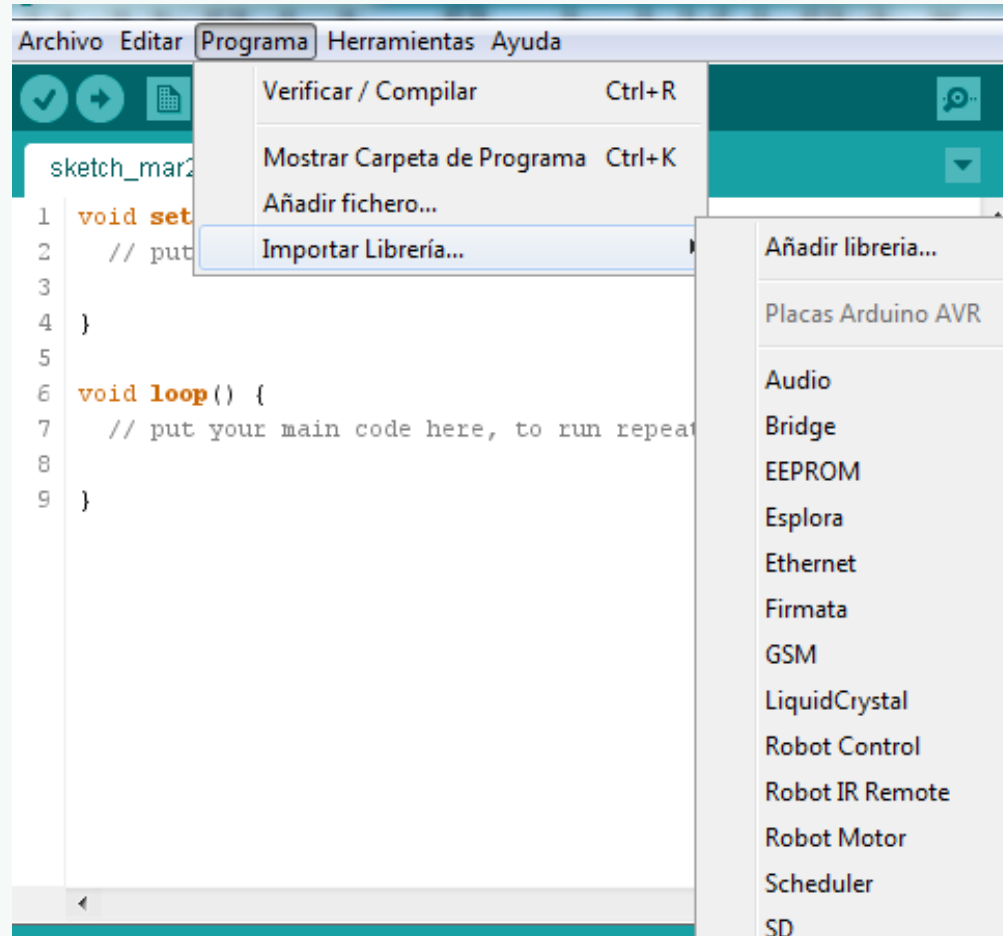
# Funciones

- 
- 
- 

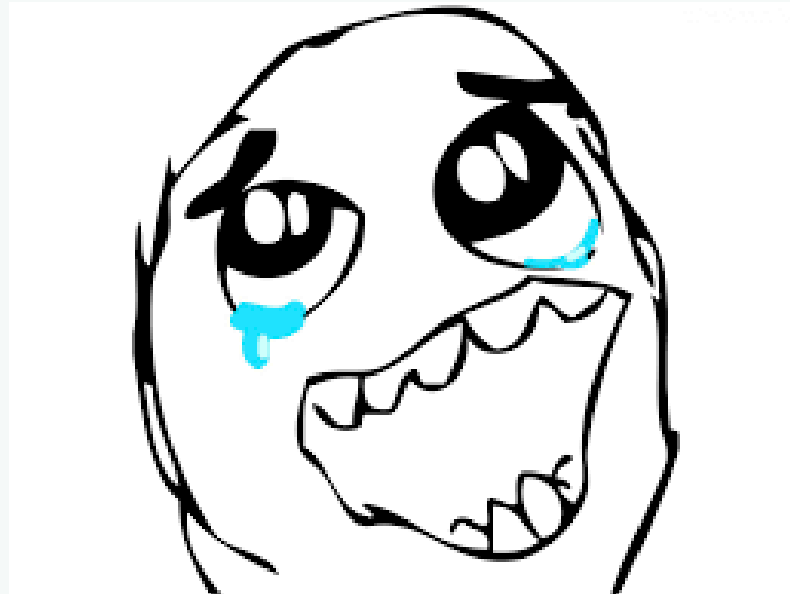
```
int r;  
int pinLed1 = 8;  
int pinLed2 = 9;  
  
void setup() {  
  
  pinMode(pinLed1,OUTPUT);  
  pinMode(pinLed2,OUTPUT);  
  
}  
  
int sumaNumeros (int x, int y){  
  
  return x+y; // la función va a regresar la suma  
              // de los parámetros  
  
}  
  
void loop() {  
  
  r = sumaNumeros(1, 2); //r = 3  
  digitalWrite(pinLed1,HIGH);  
  digitalWrite(pinLed2,HIGH);  
  delay(r*1000); // va a esperar 3 segundos  
  digitalWrite(pinLed1,LOW);  
  digitalWrite(pinLed2,LOW);  
  delay(r*1000); // va a esperar 3 segundos  
  
}
```



# Librerías



# Ejercicios



PROTECO

# Semáforo para invidentes

- 
- 
- 

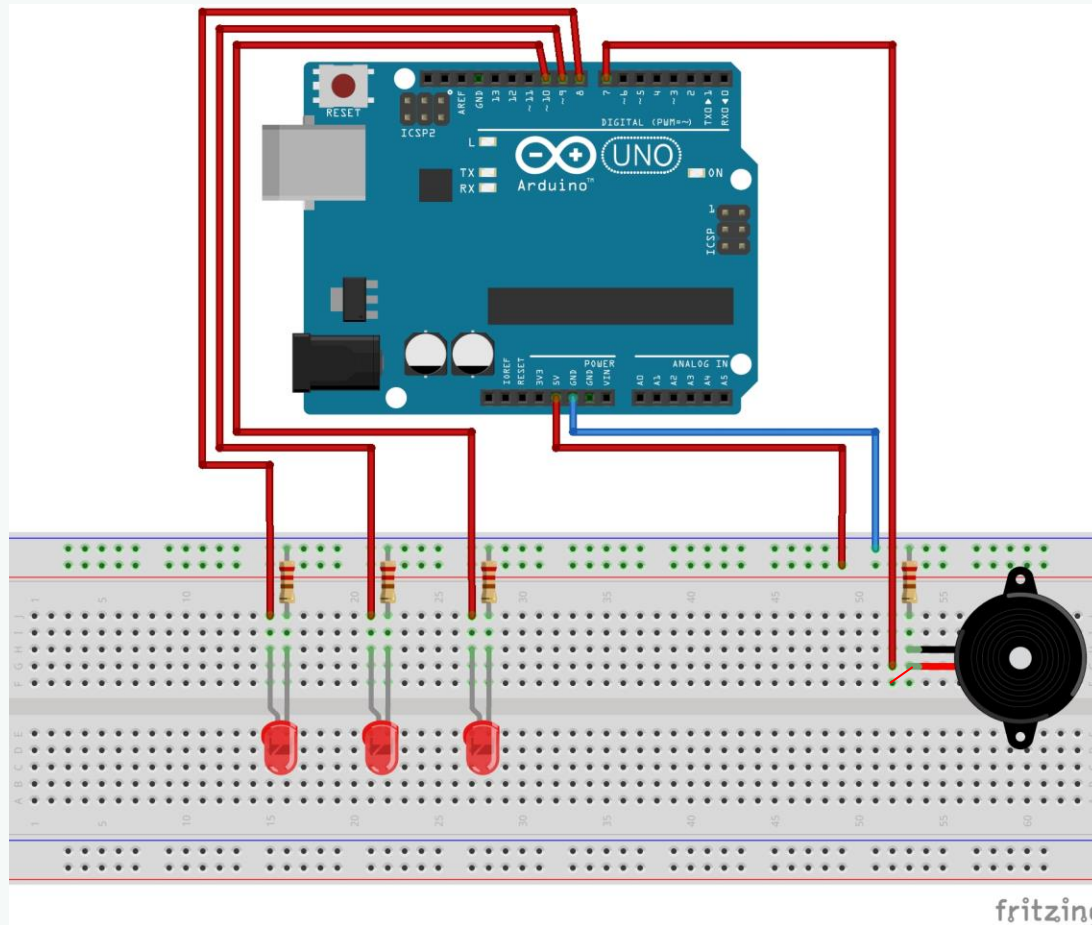


PROTECO



# Semáforo para invidentes

- 
- 
- 



PROTECO

# Sin 7

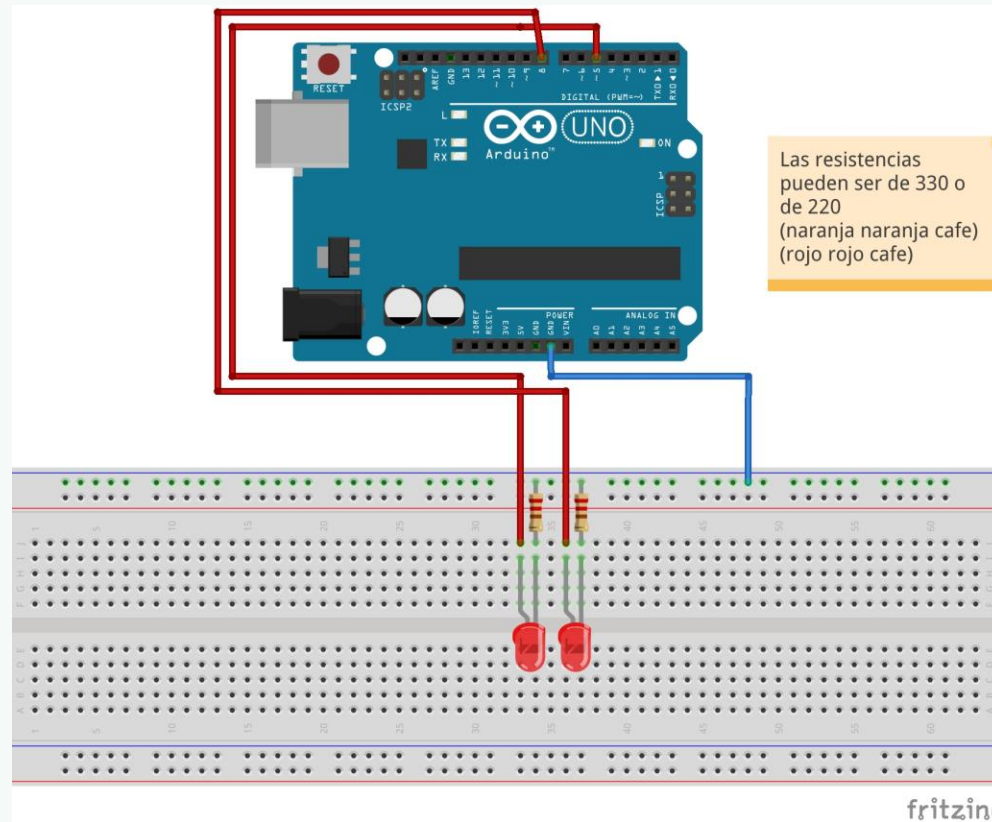
- No multiplos de 7 ni números que incluyan 7

| 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   | 10  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 11  | 12  | 13  | 14  | 15  | 16  | 17  | 18  | 19  | 20  |
| 21  | 22  | 23  | 24  | 25  | 26  | 27  | 28  | 29  | 30  |
| 31  | 32  | 33  | 34  | 35  | 36  | 37  | 38  | 39  | 40  |
| 41  | 42  | 43  | 44  | 45  | 46  | 47  | 48  | 49  | 50  |
| 51  | 52  | 53  | 54  | 55  | 56  | 57  | 58  | 59  | 60  |
| 61  | 62  | 63  | 64  | 65  | 66  | 67  | 68  | 69  | 70  |
| 71  | 72  | 73  | 74  | 75  | 76  | 77  | 78  | 79  | 80  |
| 81  | 82  | 83  | 84  | 85  | 86  | 87  | 88  | 89  | 90  |
| 91  | 92  | 93  | 94  | 95  | 96  | 97  | 98  | 99  | 100 |
| 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 |
| 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 120 |
| 121 | 122 | 123 | 124 | 125 | 126 | 127 | 128 | 129 | 130 |
| 131 | 132 | 133 | 134 | 135 | 136 | 137 | 138 | 139 | 140 |
| 141 | 142 | 143 | 144 | 145 | 146 | 147 | 148 | 149 | 150 |



PROTECO

# Sin 7



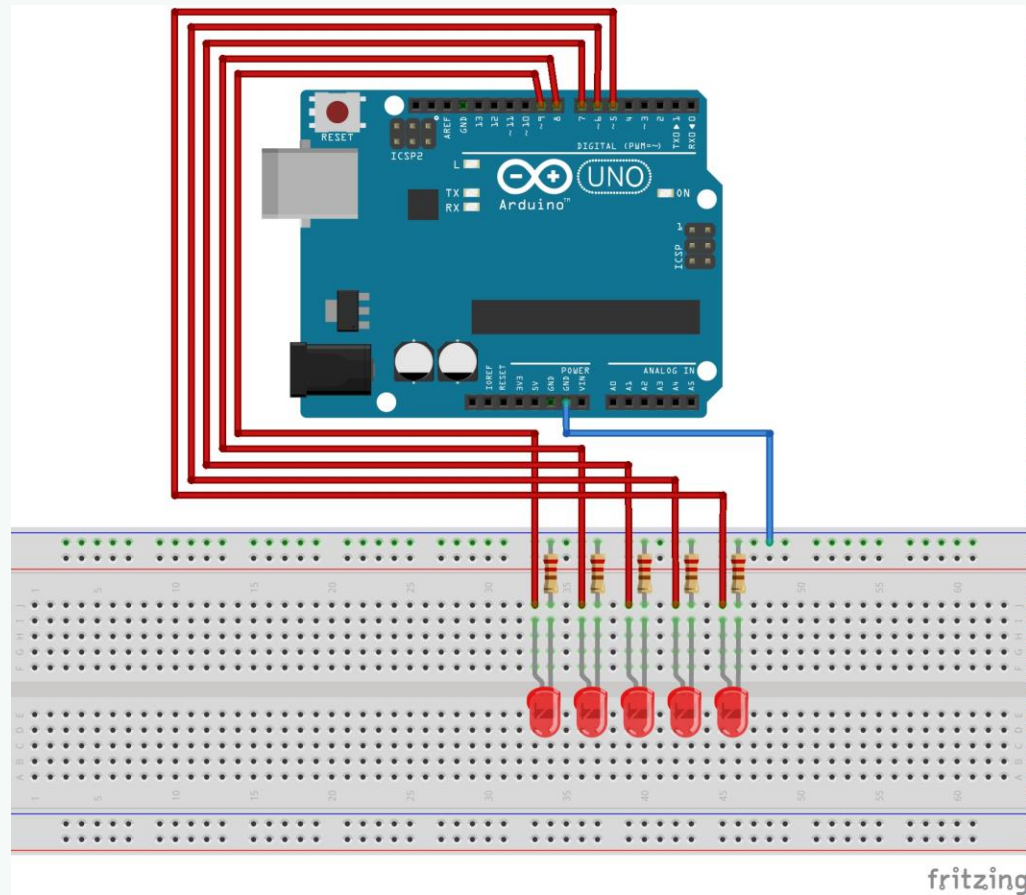
# Fibonacci

0 1 1 2 3 5 8 13 21  
34 55 89 144 233 377  
610 987 1597 2584....



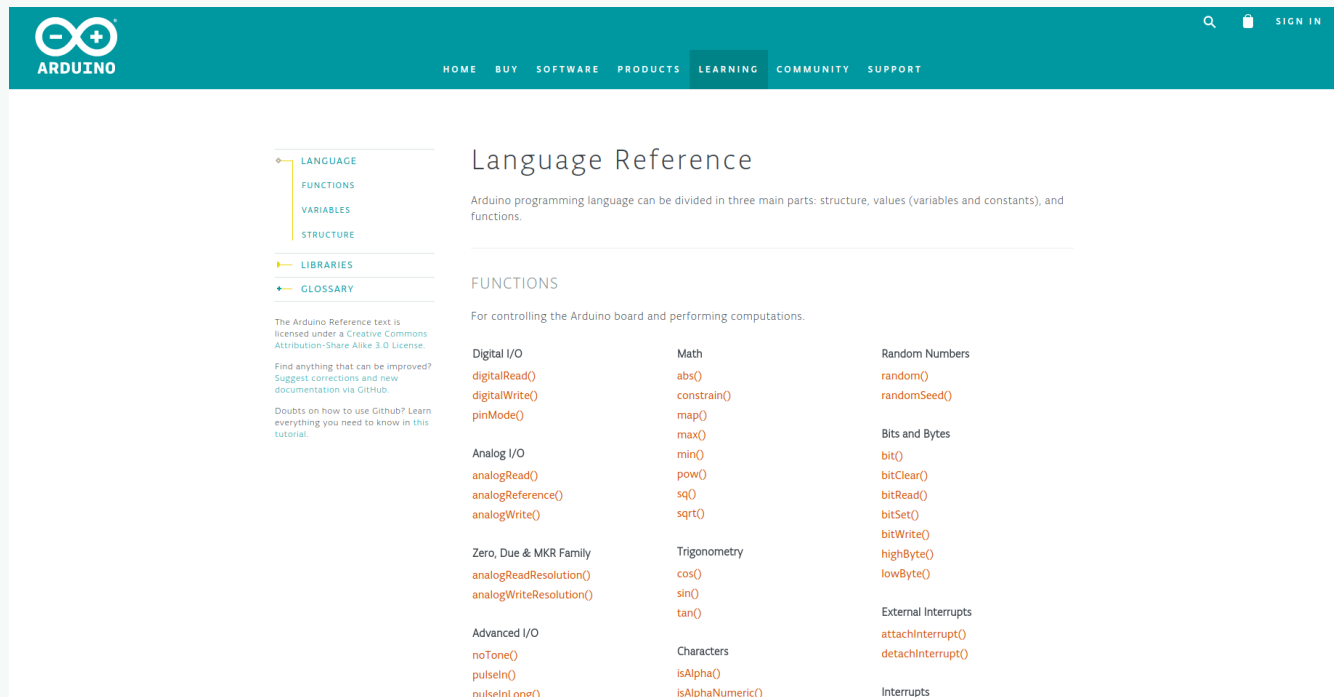
PROTECO

# Tarea



# Referencias

<https://www.arduino.cc/reference/en/>



The screenshot shows the Arduino Reference website. The header is teal with the Arduino logo and navigation links: HOME, BUY, SOFTWARE, PRODUCTS, LEARNING (highlighted), COMMUNITY, and SUPPORT. A search icon and 'SIGN IN' link are on the right. The main content area is white. On the left, a sidebar contains a vertical menu with 'LANGUAGE' (selected), 'FUNCTIONS', 'VARIABLES', and 'STRUCTURE'. Below this are 'LIBRARIES' and 'GLOSSARY'. A note mentions the Creative Commons Attribution-Share Alike 3.0 license and provides a link to GitHub for suggestions. The main title is 'Language Reference', followed by a brief description of the programming language parts. Below is a 'FUNCTIONS' section with the heading 'For controlling the Arduino board and performing computations.' and a grid of function categories: Digital I/O, Math, Random Numbers, Analog I/O, Trigonometry, Bits and Bytes, Zero, Due & MKR Family, Characters, External Interrupts, Advanced I/O, and Interrupts. Each category lists several functions in red text.

**LANGUAGE**  
FUNCTIONS  
VARIABLES  
STRUCTURE

**LIBRARIES**  
**GLOSSARY**

The Arduino Reference text is licensed under a Creative Commons Attribution-Share Alike 3.0 License.  
Find anything that can be improved? Suggest corrections and new documentation via GitHub.  
Doubts on how to use GitHub? Learn everything you need to know in this tutorial.

## Language Reference

Arduino programming language can be divided in three main parts: structure, values (variables and constants), and functions.

### FUNCTIONS

For controlling the Arduino board and performing computations.

|                                                                                        |                                                                                           |                                                                                                                |
|----------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|
| <b>Digital I/O</b><br>digitalRead()<br>digitalWrite()<br>pinMode()                     | <b>Math</b><br>abs()<br>constrain()<br>map()<br>max()<br>min()<br>pow()<br>sq()<br>sqrt() | <b>Random Numbers</b><br>random()<br>randomSeed()                                                              |
| <b>Analog I/O</b><br>analogRead()<br>analogReference()<br>analogWrite()                | <b>Trigonometry</b><br>cos()<br>sin()<br>tan()                                            | <b>Bits and Bytes</b><br>bit()<br>bitClear()<br>bitRead()<br>bitSet()<br>bitWrite()<br>highByte()<br>lowByte() |
| <b>Zero, Due &amp; MKR Family</b><br>analogReadResolution()<br>analogWriteResolution() | <b>Characters</b><br>isAlpha()<br>isAlphaNumeric()                                        | <b>External Interrupts</b><br>attachInterrupt()<br>detachInterrupt()                                           |
| <b>Advanced I/O</b><br>noTone()<br>pulseIn()<br>pulseInLong()                          | <b>Interrupts</b>                                                                         |                                                                                                                |



PROTECO