SHERPA

FULL D'AJUDA

ARDUINO

FUNCIONS

I/O Digital
I/O Analògic
I/O Avançat
Temps
Números aleatoris
Comunicació per port sèrie

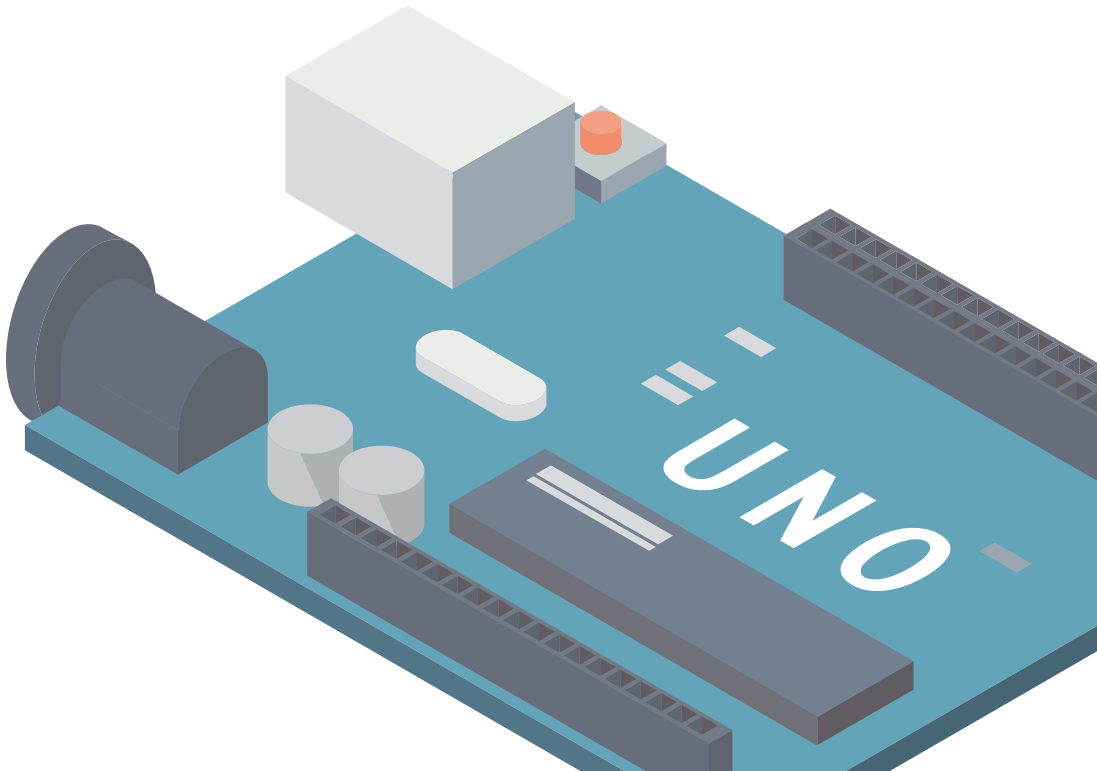
VARIABLES

Tipus de dades
Qualificadors i àmbit d'una variable

ESTRUCTURA

Estructures de control
Operadors
- Aritmètics
- De comparació
- Booleans

Sintaxi addicional



FUNCIONS

Les funcions permeten tenir un espai de codi endreçat. Aquestes fan que tot el programa sigui més compacte, ja que les seccions de codi s'utilitzen una vegada rere una altra. Hi ha dues funcions que sempre han de ser en un programa d'Arduino:

void setup()

Funció que determina les ordres que s'executaran en iniciar la placa.

```
void setup() {
//ordres de la rutina
}
```

void loop()

Funció principal del programa. És l'espai on es determinen les ordres que s'executaran mentre la placa Arduino estigui encesa. La placa llegirà des del primer ordre fins a l'últim, moment en el qual tornarà de nou a l'inici per repetir la mateixa seqüència.

```
void loop() {
//ordres de la rutina
}
```

I/O Digital

digitalRead()

Llegeix el valor d'un pin digital

```
digitalRead(pin)
```

pin: número de pin

pinMode()

Configura un pin perquè es comporti com a INPUT (entrada) o com a OUTPUT (sortida).

```
pinMode(pin, mode)
```

pin: número de pin
mode: INPUT, OUTPUT

digitalWrite()

Escriu el valor HIGH o LOW en un pin digital.

Si: `pinMode(pin, OUTPUT)`
→ HIGH = 5V; LOW = 0V (GND)

```
digitalWrite(pin, val)
```

pin: número de pin
val: *HIGH* o *LOW*

SHERPA

I/O Analògic

analogRead()

Llegeix el valor d'un pin analògic. S'assignen valors enters d'entre 0 i 1023.

```
analogRead(pin)
```

pin: número de pin

analogWrite()

Escriu un valor analògic en un pin de tipus PWM. S'assignen valors enters d'entre 0 i 255.

```
analogWrite(pin, val)
```

pin: número de pin (sortida PWM)
val:valor entre 0 (sempre apagat) i 255 (sempre encès)

I/O Avançat

tone()

Genera una ona de la freqüència especificada en un pin. Si no es determina durada, l'ona continuarà fins que es cridi a `noTone()`. Sols pot generar un to cada vegada.

```
tone(pin, freq)
tone(pin, freq, duration)
```

pin: numero del pin
freq: la freqüència del to en hertzs
duration: durada del to en milisegons

noTone()

Atura l'ona disparada per `tone()`.

```
noTone(pin)
```

pin: numero del pin

pulseIn()

Llegeix un pols (`HIGH` o `LOW`) en un pin. Mesura la longitud d'un pols (en microsegons).

```
pulseIn(pin, val)
```

pin: numero del pin
val: tipus de pols a llegir HIGH o LOW

Temps

delay()

Atura el programa durant un temps especificat (en mil·lisegons).

```
delay(ms)
```

ms: número de milisegons per fer una pausa

millis()

Retorna el número de mil·lisegons des de que la placa ha començat a executar un programa.

```
time = millis()
```

Números aleatoris

random()

Genera nombres aleatoris.

```
random(max)
random(min, max)
```

min: el número més baix del valor aleatori (aquest inclòs)
max: el número més alt del valor aleatori (aquest inclòs)

Comunicació per port sèrie

available()

Obté el nombre de bytes (caràcters) disponibles per a la seva lectura des del port sèrie.

```
Serial.available()
```

begin()

Estableix la velocitat de dades en bits per segon per a la transmissió de dades en sèrie.

```
Serial.begin(speed)
Serial.begin(speed, config)
```

speed: velocitat en bits per segon
config: estableix dades i bits d'aturada

read()

Llegeix les dades entrants per port sèrie.

```
Serial.read()
```

print()

Imprimeix dades al port sèrie com a text ASCII llegible.

El paràmetre `format` (opcional) especifica la base a utilitzar: `BIN` (binari o base 2), `OCT` (octal o base 8), `DEC` (decimal o base 10) i `HEX`(hexadecimal, o base 16). Per als nombres reals (`float`), aquest paràmetre especifica el nombre de decimals a utilitzar.

```
Serial.print(val)
Serial.print(val, format)
```

val: el valor a imprimir
format: base a utilitzar

write()

Escriu dades binàries al port sèrie. Aquestes dades s'envien com un byte o una sèrie de bytes.

```
Serial.write(val)
Serial.write("str")
```

val: un valor a enviar com un sol byte
str: string per a enviar com una sèrie de bytes

VARIABLES

Una variable és una forma d'**anomenar** i **emmagatzemar** un valor perquè el programa el pugui fer servir posteriorment.

Totes les variables s'han de **declarar** abans de ser utilitzades, això vol dir que es defineix la seva tipologia. És recomanable, també, assignar un **valor inicial** per tal de no trobar errors en la primera lectura.

És important la posició dins el codi on es defineix una variable per poder utilitzar-la dins el programa.

Tipus de dades

`bool`

Variable que sols admet dos valors, `true` o `false`.

```
bool var = val;
```

var: nom de la variable
val: valor a assignar a aquesta variable, *true* o *false*

`int`

Els `int` (*integers* o nombres enters) són el tipus de dades primari per a l'emmagatzematge de nombres.

```
int var = val;
```

var: nom de la variable
val: valor a assignar a aquesta variable

`float`

Els `float` (*floating-point* o nombres reals) són el tipus de dades per a números amb punt decimal.

Els `float` tenen només 6-7 dígits decimals de precisió.

```
float var = val;
```

var: nom de la variable
val: valor a assignar a aquesta variable

char

Dada que emmagatzema el valor d'un caràcter. Els caràcters s'emmagatzemen com a números. Per conèixer el valor d'un caràcter consultem la taula ASCII.

```
char var = val;
char var = 'val';
```

var: nom de la variable
val: valor numèric ASCII a assignar a aquesta variable, que pot ser el valor numèric ASCII
'val': caràcter a guardar

string

Dada que està formada per més d'un caràcter.

```
string var = "val"
```

var: nom de la variable
"val": cadena de caràcters

array

Matriu o taula que emmagatzema valors. Per accedir a aquests valors utilitzem el nom de la matriu i un número d'índex (posició que ocupa el valor dins la matriu).

```
tvar array [index] = {val}
tvar array [index] = {'val'}
tvar array [index] = {"val"}
```

tvar: tipus de variable
array: nom de la matriu
index: posició o número d'índex dins la matriu
val: valor numèric
'val': caràcter
"val": cadena o matriu de caràcters

Qualificadors i àmbit d'una variable

const

La paraula `const` es tradueix com a constant. És un classificador que modifica el comportament d'una variable, fent-la "només de lectura". Això significa que el valor de la variable no es pot canviar.

```
const tvar var = val
```

const: qualifica una variable com a constant
tvar: tipus de variable
var: nom de la variable
val: valor a assignar a aquesta variable

ESTRUCTURA

Estructures de control

Les estructures de control d'un llenguatge de programació permeten prendre mesures basades en condicions, és a dir, controlen el flux d'execució del programa.

break

S'utilitza per a sortir d'un bucle `while` o d'una declaració de `switch case`.

```
break;
```

continue

Omet la resta de la repetició actual d'un bucle `while`. Continua comprovant l'expressió condicional del bucle i segueix amb qualsevol repetició posterior.

```
continue;
```

return

Finalitza una funció i retorna un valor on s'ha cridat aquesta funció al codi.

```
return;
return val;
```

val: qualsevol tipus de variable o constant

while

Un bucle `while` es repeteix contínuament i infinitament fins que l'expressió dins del parèntesi, () es torna `false`.

```
while (cond) {
// declaració o declaracions(s)
}
```

cond: expressió booleana amb dos valors possibles, *true* o *false*, que determina uns paràmetres a complir

if

Comprova una condició i executa la següent expressió o conjunt d'expressions si la condició és `true`.

```
if (cond) {
//declaració o declaracions
}
```

cond: expressió booleana amb dos valors possibles, *true* o *false*, que determina uns paràmetres a complir

if…else

Permet un major control sobre el flux de codi que l'expressió `if`, permetent agrupar múltiples proves. S'executarà un `else` (si existeix) si la condició `if` és `false`.

Cada prova procedirà a la següent fins que es trobi una prova que sigui `true`. Quan aquesta es troba, s'executa el seu bloc de codi associat.

Un `else if` es pot utilitzar amb o sense un `else` i viceversa. `else` és la condició que contempla tot allò que no s'ha complert prèviament amb `if` o `else if` i és opcional dins l'estructura. Es permet un nombre il·limitat d'aquestes branques `else if`.

```
if (cond1) {
// fes A
}
else {
// fes B
}
```

```
if (cond1) {
// fes A
}
else if (cond2) {
// fes B
}
else if (cond3) {
// fes C
}

[…]

else if (condX) {
// fes X
}
```

```
if (cond1) {
// fes A
}
else if (cond2) {
// fes B
}
else if (cond3) {
// fes C
}

[…]

else {
// fes X
}
```

const: califica una variable com a constant
tvar: tipus de variable
var: nom de la variable
val: valor a assignar a aquesta variable

switch…case

Compara el valor d'una variable amb els valors especificats en cada `case`. Quan es troba un valor d'un `case` que coincideix amb el de la variable, s'executa el codi dins d'aquest `case`.

L'expressió `break` s'utilitza normalment al final de cada `case`. Sense una declaració `break`, la declaració `switch` continuarà executant les expressions següents fins que s'arribi a un `break`, o al final de la declaració `switch`.

`default` és la condició que contempla tot allò que no s'ha complert prèviament amb `switch` o `case`, i és opcional dins l'estructura.

```
switch (var) {
case val1:
// declaracions
case val2:
// declaracions

[…]

case valX:
// declaracions
}
```

```
switch (var) {
case val1:
// declaracions
case val2:
// declaracions

[…]

case valX:
// declaracions
default:
// declaracions
break;
}
```

```
switch (var) {
case val1:
// declaracions
break;
case val2:
// declaracions
break;

[…]

case valX:
// declaracions
break;
}
```

```
switch (var) {
case val1:
// declaracions
break;
case val2:
// declaracions
break;

[…]

case valX:
// declaracions
break;
default:
// declaracions
break;
}
```

var: variable el valor de la qual s'ha de comparar amb diversos casos. Es permeten dades del tipus int i char.
val1, val2…valX: constants de valor de la variable. Es permeten dades del tipus int i char.

Operadors

Un operador és un símbol que indica al programa que es duguin a terme certes operacions matemàtiques o lògiques.

Operadors aritmètics

*****

Multiplicació.

```
mult = op1 * op2;
```

+

Suma.

```
sum = op1 + op2;
```

-

Resta.

```
rest = op1 - op2;
```

/

Divisió.

```
resultat = numerador / denominador;
```

=

S'anomena operador d'assignació; assigna un valor a allò que es troba a la dreta d'aquest símbol dins el programa.

```
var = val
```

var: variable declarada
val: valor o expressió avaluada

SHERPA

Operadors de comparació

Comparen la variable a l'esquerra de l'operador amb el valor o la variable a la dreta de l'operador.
Es recomana comparar variables del mateix tipus de dades incloent el tipus signe/sense signe.

!=

not equal to

Retorna `true` quan els dos operands no són iguals.

```
x != y;
```

<

less than

Retorna `true` quan l'operand de l'esquerra és més petit que l'operand de la dreta.

```
x > y;
```

<=

less than or equal to

Retorna `true` quan l'operand de l'esquerra és més petit o igual que l'operand de la dreta.

```
x <= y;
```

==

equal to

Retorna `true` quan els dos operands són iguals.

```
x == y;
```

>

greater than

Retorna `true` quan l'operand de l'esquerra és més gran que l'operand de la dreta.

```
x > y;
```

>=

greater than or equal to

Retorna `true` quan l'operand de l'esquerra és més gran o igual que l'operand de la dreta.

```
x >= y;
```

Operadors booleans

!

logical NOT

Dona com a resultat `true` si l'operand és `false` i viceversa.

Es pot utilitzar dins d'una condició `if`.

```
if (!x) {
// si x no es true
// declaració o declaracions
}
```

També es pot utilitzar per invertir el valor booleà.

```
x = !y;
```

&&

logical AND

Dona com a resultat `true` si ambdós operands són `true`.

```
if (operand1 == x &&
operand2 == x) {
// si ambdós operands tenen
el mateix val
// declaració o declaracions
}
```

||

logical OR

Dona com a resultat `true` si cap dels dos operands és `true`.

Aquest operador es pot utilitzar dins d'una condició `if`.

```
if (x > 0 || y > 0) {
// si ni x ni y són més
grans que 0
// declaració o declaracions
}
```

Sintaxi addicional

#include

`#include` s'utilitza per incloure biblioteques de fora en el programa. `#include` no duu el símbol ";" darrere seu.

```
#include <Arxiu.h>
#include "Arxiu.h"
```

;

semicolon

S'utilitza per finalitzar una declaració.

SHERPA

//

single line comment

Un comentari d'una sola línia comença amb el símbol //. Aquest comentari acaba automàticament al final de la línia on es troba. Tot el que segueix // fins al final d'una línia serà ignorat pel compilador.

/*...*/

block comment

Bloc de comentaris o comentari de múltiples línies.

Els comentaris són línies del programa que s'utilitzen per donar informació sobre com funciona el programa. El compilador els ignora i no s'exporten al processador, de manera que no ocupen espai en la memòria del microcontrolador.

El símbol /* marca l'inici d'un bloc de comentaris o d'un comentari de múltiples línies i el símbol / indica el seu final; una vegada que el compilador llegeixi el /* ignora el següent fins que troba un /.

REFERÈNCIES EXTERNES

Documentació oficial

- Referència Arduino: Guia de funcions, operadors i variables.
- Guia d'iniciació: Pas a pas per començar amb Arduino.
- Llibreries d'Arduino: Llista de llibreries oficials.

Tutorials i cursos de formació

- Tutorials oficials d'Arduino: Tutorials de temes bàsics i avançats relacionats amb Arduino.
- Tutorials d'Arduino de *Programming Electronics Academy*: Tutorials de programació i desenvolupament de projectes amb Arduino.
- Curs d'Arduino de la Universitat de Califòrnia: Curs gratuït que cobreix els fonaments de la plataforma Arduino i la programació en C.

Projectes i idees

- Arduino Project Hub: Projectes creats per la comunitat *Arduino*.
- Comunitat *Instructables* d'*Autodesk* - Arduino: Milers de projectes detallats i tutorials.