# Práctica 9.2: Calculamos nuestras pulsaciones por minuto

Ahora, vamos a pasar a hacer algo más interesante. Está bien poder detectar nuestras pulsaciones, pero ¿por qué no contar el número de pulsaciones que tenemos por minuto o **BPM**?

Lo haremos con el siguiente código:

```
/*  Getting_BPM_to_Monitor prints the BPM to the Serial Monitor, using the least lines of code
and PulseSensor Library.
 *  Tutorial Webpage: https://pulsesensor.com/pages/getting-advanced
 *
--------Use This Sketch To-----------------------------------------
1) Displays user's live and changing BPM, Beats Per Minute, in Arduino's native Serial
Monitor.
2) Print: "♥  A HeartBeat Happened !" when a beat is detected, live.
2) Learn about using a PulseSensor Library "Object".
4) Blinks LED on PIN 13 with user's Heartbeat.
-----------------------------------------------------------------------*/

#define USE_ARDUINO_INTERRUPTS true    // Set-up low-level interrupts for most acurate BPM
math.
#include <PulseSensorPlayground.h>     // Includes the PulseSensorPlayground Library.

//  Variables
const int PulseWire = 0;       // PulseSensor PURPLE WIRE connected to ANALOG PIN 0
const int LED13 = 13;          // The on-board Arduino LED, close to PIN 13.
```

```
int Threshold = 550;           // Determine which Signal to "count as a beat" and which to
ignore.
                               // Use the "Gettting Started Project" to fine-tune Threshold
Value beyond default setting.
                               // Otherwise leave the default "550" value.

PulseSensorPlayground pulseSensor;  // Creates an instance of the PulseSensorPlayground object
called "pulseSensor"


void setup() {

  Serial.begin(9600);          // For Serial Monitor

  // Configure the PulseSensor object, by assigning our variables to it.
  pulseSensor.analogInput(PulseWire);
  pulseSensor.blinkOnPulse(LED13);       //auto-magically blink Arduino's LED with heartbeat.
  pulseSensor.setThreshold(Threshold);

  // Double-check the "pulseSensor" object was created and "began" seeing a signal.
   if (pulseSensor.begin()) {
    Serial.println("We created a pulseSensor Object !");  //This prints one time at Arduino
power-up,  or on Arduino reset.
  }
}



void loop() {

 int myBPM = pulseSensor.getBeatsPerMinute();  // Calls function on our pulseSensor object
that returns BPM as an "int".
                                               // "myBPM" hold this BPM value now.

if (pulseSensor.sawStartOfBeat()) {            // Constantly test to see if "a beat happened".
```

```
 Serial.println("♥  A HeartBeat Happened ! "); // If test is "true", print a message "a
heartbeat happened".
 Serial.print("BPM: ");                        // Print phrase "BPM: "
 Serial.println(myBPM);                       // Print the value inside of myBPM.
}


  delay(20);                    // considered best practice in a simple sketch.


}
```

Como puedes ver, no tiene muchas líneas, ¿por qué? Pues porque la mayor parte de procesos los realizan funciones que nos vienen preprogramadas en la librería PulseSensor Playground. Así, nosotras no tenemos que preocuparnos demasiado de esa parte y nos resultará más sencillo integrar nuestro pulsómetro con otros sensores o actuadores, haciendo que no sean necesarias tantas líneas de código.

## Desmenuzando el código

Como ya he hecho en anteriores páginas, no voy a volver a explicarte la parte de los **comentarios** , aunque te invito a leerlos, porque de esta manera entenderás mejor el propósito del código y las partes de este.

En esta práctica, vamos a ver un concepto nuevo en el que no vamos a ahondar demasiado: las **interrupciones**.

```
#define USE_ARDUINO_INTERRUPTS true    // Set-up low-level interrupts for most acurate BPM
math.
#include <PulseSensorPlayground.h>     // Includes the PulseSensorPlayground Library.
```

### Las interrupciones

Las interrupciones se encargan de indicarle a nuestro Arduino que cierto proceso ha de ser atendido con una prioridad mayor a cualquier otro. De esta manera se interrumpe cualquier actividad del Arduino para atender la información recibida a través de alguno de los pines.

En nuestro ejemplo concreto, para asegurar una correcta lectura de nuestras pulsaciones, usamos la orden **#define USE_ARDUINO_INTERRUPTS true.** Todo el manejo de las interrupciones permanece oculto, pero es necesario para la correcta lectura de nuestras pulsaciones.

Copyright 2026

Si quieres saber más sobre interrupciones, puedes echarle un vistazo aeste enlace.

## Las librerías

La siguiente línea que encontramos importa la **librería PulseSensorPlayground.h.** A Arduino necesitaremos indicárselo usando la palabra reservada **#include** seguida de la librería que queramos incluir, entre **< >**. Las librerías, son archivos con la terminación **.h**. Estos documentos pueden ser abiertos en un editor de texto y podemos ver su contenido y modificarlos, aunque esto último no te lo aconsejo... ya que si no estamos seguros de qué estamos modificando y borramos o editamos algo que no deberíamos, esta librería dejaría de funcionar y tendríamos que reinstalarla.

# Las variables y constantes

Si avanzamos, encontramos las siguientes líneas, las cuales aparecen explicadas en los comentarios que las acompañan a la derecha:

```
const int PulseWire = 0;       // PulseSensor PURPLE WIRE connected to ANALOG PIN 0
const int LED13 = 13;          // The on-board Arduino LED, close to PIN 13.
int Threshold = 550;           // Determine which Signal to "count as a beat" and which to
ignore.
                               // Use the "Gettting Started Project" to fine-tune Threshold
Value beyond default setting.
                               // Otherwise leave the default "550" value.
```

En Arduino, habíamos visto que existen distintos tipos de datos para almacenar información. Por ejemplo, si estamos trabajando con números, podíamos usar, entre otros, números enteros o decimales, int o float respectivamente. A parte de elegir el tipo de datos, podemos indicarle a nuestro Arduino si el valor que vamos a almacenar va a variar o siempre va a ser el mismo. Si el valor puede ser modificado a lo largo del código, escribiremos las variables como hemos hecho hasta ahora, pero si no queremos que ese valor cambia, si queremos que sean constantes, tendremos que escribir delante la palabra **const**.

# El objeto pulseSensor

Copyright 2026

Para hacer uso de las funciones preprogramadas en la librería que queremos utilizar, tenemos que crear "una instancia del objeto PulseSensorPlayground". Eso es lo que nos pone en el comentario que acompaña a la siguiente línea de código. Pero, ¿qué significa eso?

```
PulseSensorPlayground pulseSensor;  // Creates an instance of the PulseSensorPlayground object called "pulseSensor"
```

Bueno, si te suena raro, voy a explicártelo con coches. Por un lado existe la idea abstracta de coche y, por otro, los coches reales que podemos conducir con unas características que los diferencian según la marca, el año, etc. Bueno, hasta aquí todo bien. Pues nuestra librería es como la idea de coche, también tiene una serie de características y para usarlas, tendremos que crear un objeto que funcione como un pulsómetro.

En el caso de los coches, una instancia de la idea de coche podría ser un ALFA ROMEO Giulietta 1.6 JTD 120CV. En el caso de nuestra librería, podemos crear una instancia de un pulsómetro con la línea de código: **PulseSensorPlayground pulseSensor;**

Ahí, lo que estamos diciendo es que a partir de la **idea de coche/librería** PulseSensorPlayground, vamos a obtener un **coche/pulsómetro** que se va a llamar pulseSensor. Una vez hayamos creado este objeto, podremos utilizar las funciones que en la librería han sido programadas.

## void setup(): lo que se ejecuta una sola vez

Una vez tenemos  todo preparado, será necesario comenzar la comunicación a través del puerto serie y configurar nuestro objeto pulseSensor. Para ello, usaremos funciones vinculadas a él. Estas funciones tendremos que escribirlas a continuación de nuestro objeto y separadas de este por un **'.'** Una de ellas es **pulseSensor.analogInput(PulseWire).** Ahí, lo que estamos diciendo es que para nuestro pulsómetro, el pin de entrada va a ser el **pin A0**. Si recuerdas, la constante **pulseWire** ya la hemos definido antes.

Las otras dos funciones que la siguen, lo que dicen es que el LED que parpadeará será el 13 ( **pulseSensor.blinkOnPulse(LED13))** y que el umbral para detectar la pulsación sea el que hemos definido antes con la constante Threshold (**pulseSensor.setThreshold(Threshold))** .

```
void setup() {

  Serial.begin(9600);          // For Serial Monitor

  // Configure the PulseSensor object, by assigning our variables to it.
```

```
  pulseSensor.analogInput(PulseWire);
  pulseSensor.blinkOnPulse(LED13);       //auto-magically blink Arduino's LED with heartbeat.
  pulseSensor.setThreshold(Threshold);

  // Double-check the "pulseSensor" object was created and "began" seeing a signal.
   if (pulseSensor.begin()) {
    Serial.println("We created a pulseSensor Object !");  //This prints one time at Arduino
power-up,  or on Arduino reset.
  }
}
```

A continuación, encontramos un condicional en el que, si hemos comenzado la comunicación por el puerto serie, imprimiremos: **We created a pulseSensor object!**.

## void loop(): lo que se ejecuta todo el rato

A continuación, vamos a guardar en una variable las pulsaciones por minuto que va a calcular la función **.getBeatsPerMinute()**. Esto lo va a calcular la función directamente, así que no tendremos que preocuparnos de cómo se hace.

En el siguiente condicional, lo que vamos a hacer es que cada vez que se detecte una pulsación, con la función **.sawStartOfBeat()**, se va a imprimir por el puerto serie un aviso de que ha ocurrido una pulsación e imprimirá las pulsaciones por minuto actuales.

Para finalizar, se añade un pequeño **delay** para que la lectura de pulsaciones se realice correctamente.

```
void loop() {

 int myBPM = pulseSensor.getBeatsPerMinute();  // Calls function on our pulseSensor object
that returns BPM as an "int".
                                               // "myBPM" hold this BPM value now.

if (pulseSensor.sawStartOfBeat()) {            // Constantly test to see if "a beat happened".

 Serial.println("♥  A HeartBeat Happened ! "); // If test is "true", print a message "a
heartbeat happened".
```

```
  Serial.print("BPM: ");                        // Print phrase "BPM: "
  Serial.println(myBPM);                       // Print the value inside of myBPM.
}

   delay(20);                     // considered best practice in a simple sketch.

}
```

Ahora que ya entendemos el código, conectamos nuestro Arduino al ordenador y subimos el código.

En este .gif podemos verlo en acción:

