Trabajo Práctico 4
Entrada/Salida

Tecnología Digital II

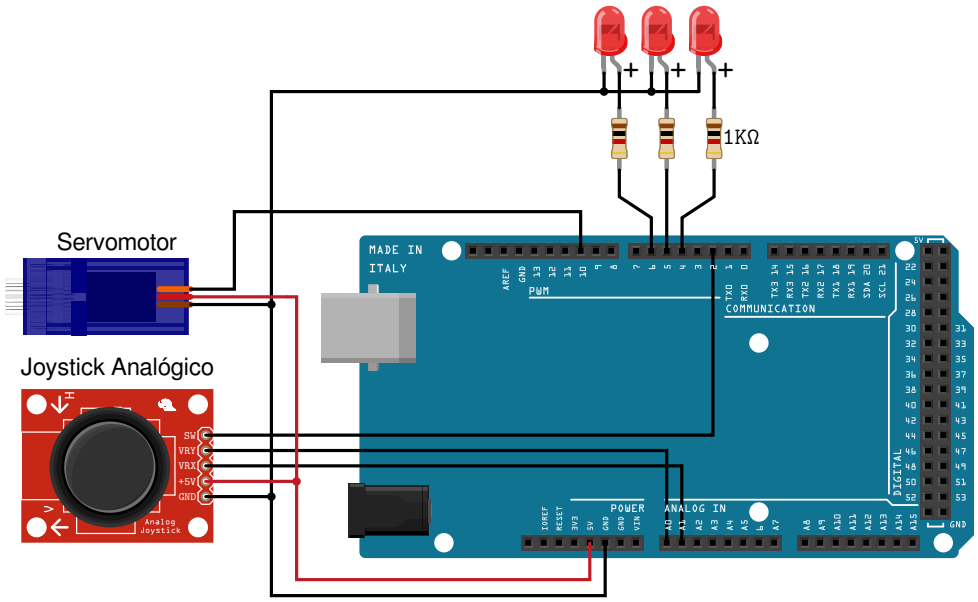
Introducción

Los mecanismos de entrada/salida que utilizan los sistemas de computo modernos requieren de una gran complejidad técnica para llevarlos a la práctica. Por esta razón se optó por utilizar dispositivos más simples basados en microcontroladores, para experimentar con mecanismos de entrada/salida.

La presente actividad consiste construir un circuito electrónico, conectando componentes como indicadores, sensores y actuadores, para luego utilizarlos en diferentes ejercicios. Los ejercicios están diseñados para ir incrementando su dificultad a medida que se incluyen distintos mecanismos de entrada/salida y componentes.

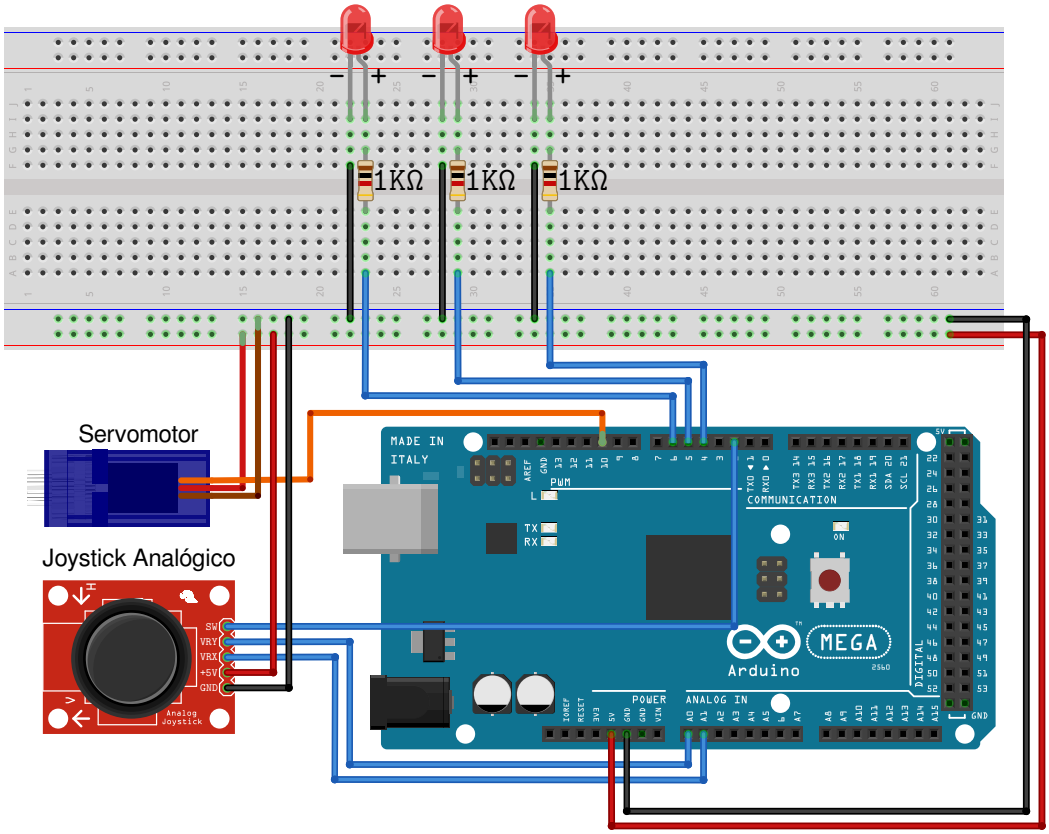
Para realizar el trabajo se utilizará un kit de Arduino basado en la placa Mega 2560 Rev3. Los componentes como servos, motores, joysticks no requieren mayor cuidado que conectar adecuadamente sus entradas, mientras que componentes como resistencias y leds requieren prestar mucha atención en el uso, para no manipularlos incorrectamente y dejarlos inutilizados.

La infraestructura de Arduino provee todo el software necesario para compilar y ejecutar ejemplos sobre estas placas, además provee bibliotecas de funciones utilitarias para controlar dispositivos. Al final de este enunciado se presentan algunas funciones básicas como referencia. Es importante que utilicen la documentación oficial para resolver dudas y consultar otras funciones que gusten usar para resolver los ejercicios solicitados.



Actividades preliminares

Este trabajo práctico está diseñado para ser realizado durante dos jornadas de clase, para esto se debe previamente tener instalado todo el **Arduino IDE**. El software se debe descargar de `https://www.arduino.cc/en/software`, donde se presenta tanto para sistemas Windows, Linux o Mac. Es fundamental que previamente a iniciado el primer día del taller tengan instalado todo el software de Arduino. Además deben leer toda la documentación provista y estar familiarizados con el enunciado del taller, buscando analizar los códigos presentados y las posibles formas de solucionar los ejercicios pedidos.



El segundo paso, ya en clase, será armar el circuito presentado anteriormente. Deben conectar todos los cables y conexiones a componentes como se indica en la figura. Puede que para esto, tengan que usar más cables de los indicados, lo importante es respetar las conexiones del circuito.

Ejercicios

Algunos de los items de los ejercicios estás marcados con la etiqueta **[Trabajo Previo]**, esto indica, tal como su nombre lo indica, que es tarea que se debe realizar previamente al taller. En todos los casos se solicitará que tengan completos estos puntos antes de comenzar el taller. El 30 % de la calificación del taller corresponde a que tengan realizada está tarea, la cual será verificada al comienzo del taller.
Por otro lado, otros items pueden tener las etiquetas **[Opción A]** o **[Opción B]**, esto quiere decir que a la hora de realizar el ejercicio, pueden elegir entre hacer uno u otro de los items.

1. **Introducción**

   Leer la documentación adjunta, complementando el contenido con la documentación oficial provista en `https://www.arduino.cc/reference/es/` y responder las siguientes preguntas:

   a) **[Trabajo Previo]** ¿Cual es la diferencia entre un pin digital y un pin analógico?
   b) **[Trabajo Previo]** ¿Qué hace la función `digitalPinToInterrupt`?
   c) **[Trabajo Previo]** ¿Para que sirve la función `map`?
   d) **[Trabajo Previo]** ¿Para que sirve habilitar la conexión serial?

2. **Uso de Leds**

   a) Cargar el siguiente programa y ejecutarlo.

```
const int ledPin0 = 4;
const int ledPin1 = 5;
const int ledPin2 = 6;

void setup() {
```

```
  pinMode(ledPin0, OUTPUT);
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);
}

void loop() {
    digitalWrite(ledPin0, 1);
    digitalWrite(ledPin1, 1);
    digitalWrite(ledPin2, 1);
    delay(1000);
    digitalWrite(ledPin0, 0);
    digitalWrite(ledPin1, 0);
    digitalWrite(ledPin2, 0);
    delay(1000);
}
```

b) **[Trabajo Previo]** Explicar que realiza el programa.

c) **[Opción A]** Modificar el programa para que presente la siguiente secuencia de encendido de leds: `000`, `001`, `011`, `111`, `110`, `100`. La misma se debe repetir todo el tiempo en intervalos regulares de 1 segundo entre cada valor.

d) **[Opción B]** Modificar el programa anterior para que por cada iteración de la secuencia, el tiempo entre cada valor se reduzca en 0,05 segundos. Una vez que llegue a 0 se debe volver a reiniciar a 1 segundo.

3. **Uso de Joystick**

a) Cargar el siguiente programa y ejecutarlo.

```
const int ledPin0 = 4;
const int VRy = A0;
const int VRx = A1;
const int SW = 2;

void setup() {
  Serial.begin(9600);
  pinMode(ledPin0, OUTPUT);
  pinMode(VRx, INPUT);
  pinMode(VRy, INPUT);
  pinMode(SW, INPUT_PULLUP);
}

void loop() {
  int xPosition = analogRead(VRx);
  int yPosition = analogRead(VRy);
  int SW_state = digitalRead(SW);

  int mapX = map(xPosition, 0, 1023, -512, 512);
  int mapY = map(yPosition, 0, 1023, -512, 512);

  Serial.println(mapX);
  Serial.println(mapY);
  Serial.println(SW_state);

  digitalWrite(ledPin0, SW_state);
  delay(10);
}
```

b) **[Trabajo Previo]** Explicar que realiza el programa.

c) **[Opción A]** Modificar el programa de forma que se enciendan los leds respetando el siguiente patron:

Posición de Joystick	Leds
↑	`010`
↓	`101`
←	`110`
→	`011`

El umbral de encendido debe ser sobre el 10 % del recorrido del Joystick. En el caso de detectar una esquina, se deben colocar todos los leds en cero.

d) **[Opción B]** Modificar el programa de forma que los leds se enciendan gradualmente desde `000` a `111` indicando la posición del Joystick, tanto del centro hacia arriba como del centro hacia abajo.

4. **Uso de Servo**

a) Cargar el siguiente programa y ejecutarlo.

```
#include <Servo.h>
const int servoPin = 10;
Servo servo1;

void setup() {
  servo1.attach(servoPin);
}

void loop() {
  servo1.write(0);
  delay(1000);
  servo1.write(30);
  delay(1000);
  servo1.write(60);
  delay(1000);
  servo1.write(90);
  delay(1000);
  servo1.write(180);
  delay(1000);
}
```

b) **[Trabajo Previo]** Explicar que realiza el programa.

c) **[Opción A]** Modificar el programa para que el Servo reproduzca el siguiente patrón de comportamiento una y otra vez, cambiando de posición cada 2 segundos.

Tiempo	Posición del Servo
1	0 grados
2	90 grados
3	45 grados
4	180 grados

d) **[Opción B]** Modificar el programa para que el movimiento de izquierda a derecha del Joystick, reproduzca el mismo comportamiento que el Servo de 0 a 180 grados.

5. **Interrupciones**

a) Cargar el siguiente programa y ejecutarlo.

```
#include <Servo.h>
Servo servo1;
const int servoPin = 10;

const byte interruptPin = 2;
volatile byte button = 0;

const int ledPin0 = 4;
const int ledPin1 = 5;

const int VRy = A0;
const int VRx = A1;
const int SW = 2;

void setup() {
  servo1.attach(servoPin);
  pinMode(ledPin0, OUTPUT);
  pinMode(ledPin1, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin), click, RISING);
  pinMode(VRx, INPUT);
  pinMode(VRy, INPUT);
  pinMode(SW, INPUT_PULLUP);
}

void loop() {
  int xPosition = analogRead(VRx);
  int yPosition = analogRead(VRy);
  int SW_state = digitalRead(SW);

  int angulo1 = map(xPosition, 0, 1023, 0, 180);
  servo1.write(angulo1);

  int brillo = map(yPosition, 0, 1023, -255, 255);
  analogWrite(ledPin1, abs(brillo));

  if( button ) {
    digitalWrite(ledPin0, 1);
  } else {
    digitalWrite(ledPin0, 0);
  }
}

void click() {
  if( button == 0 ) {
    button = 1;
  } else {
    button = 0;
  }
}
```

b) **[Trabajo Previo]** Explicar que realiza el programa.

c) **[Opción A]** Modificar el programa para que cada vez que se presiona el botón del Joystick se prendan todos los leds y estos queden prendidos. Cuando se presione nuevamente, estos se deben apagar.

d) **[Opción B]** Modificar el programa para que cada vez que se presiona el botón del Joystick

se intercambie el sentido de rotación del Servo.

6. **Resumen**

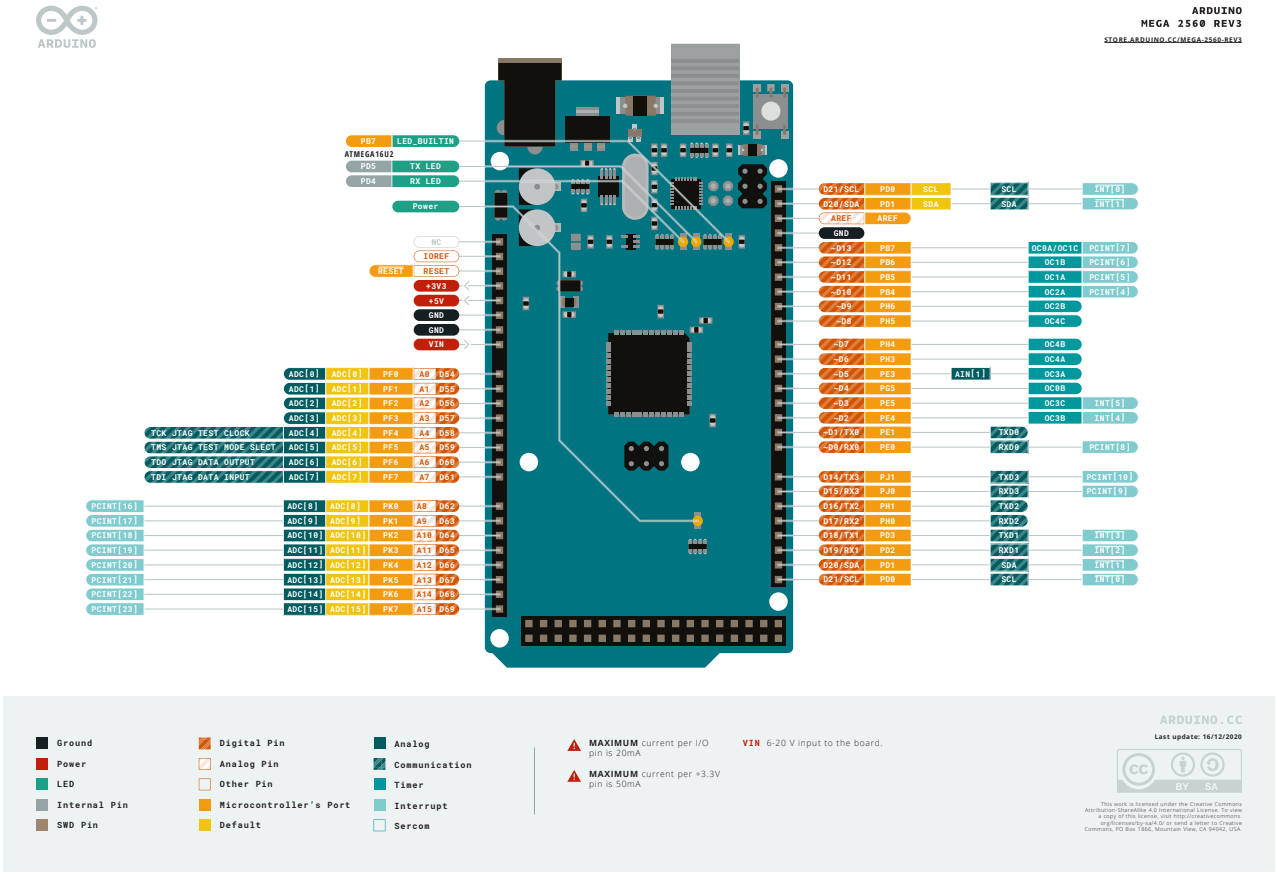
   Utilizado lo aprendido en los puntos anteriores construir un programa que respete el siguiente comportamiento:

   - Los leds van a mostrar contando del `000` al `111` la posición del Joystick en sentido vertical.
   - El movimiento del Joystick en sentido horizontal deberá mover el Servo en todo su rango.
   - Si se presiona el botón del Joystick se intercambiará el comportamiento de los ejes vertical y horizontal.

Referencia Arduino

El proyecto Arduino consiste en una familia de placas para realizar proyectos simples de electrónica. Compuestas por un microcontrolador programable con una configuración mínima y una serie de pines que permiten establecer conexiones entre el microcontrolador y dispositivos externos, como sensores, actuadores o incluso displays.

PinOut



La placa Arduino se conecta a la computadora por medio de un conector USB tipo A, por lo tanto se debe contar con tal puerto para poder programar la placa. *Recuerden traer adaptadores en caso de ser necesario.*

Estructura

La estructura de un programa para un Arduino consiste en dos funciones.

- **`setup()`** Esta función es llamada al comienzo del programa, cuando inicia la placa y es ejecutada solo una sola vez. Se ocupa de inicializar variables, el modo de los pines, bibliotecas de funciones, etc.
- **`loop()`** Luego de la función `setup()` se ejecuta la función `loop`. Esta función es ejecutada, como su nombre lo indica, es ejecutada una y otra vez en un ciclo infinito. Este ciclo permite al sistema Arduino tomar el control de forma activa entre cada iteración del ciclo.

Funciones básicas

`digitalRead(pin)`

Lee el estado de un pin digital, este puede ser `HIGH` o `LOW`.
Parámetros

- `pin`: El número del pin digital a leer.

digitalWrite(pin, value)
Escribe un valor `HIGH` o `LOW` en un pin digital seleccionado.
Parámetros
- `pin`: El número de pin digital a escribir.
- `value`: `HIGH` o `LOW`.

pinMode(pin, mode)
Configura el comportamiento de un pin, tanto como entrada o como salida. Parámetros
- `pin`: El número de pin a configurar.
- `mode`: `INPUT`, `OUTPUT`, o `INPUT_PULLUP`.

analogRead(pin)
Lee el valor de un pin analógico. El conversor de analógico a digital es de 10-bits, por lo tanto retorna valores entre 0 y 1023. La resolución es de 4.9 mV por unidad (5 vols / 1024 unidades). El conversor demora aproximadamente 100 microsegundos (0.0001 S) en convertir la magnitud analogica en un valor digital, pudiendo llegar a un máximo de 10000 lecturas por segundo.
Parámetros
- `pin`: El número del pin analogico a leer.

analogWrite(pin, value)
Escribe un valor analógico en el pin como una onda *PWM* (Modulación por ancho de pulsos). La frecuencia de la onda es de 490 Hz o 980 Hz dependiendo del pin que se este utilizando. Esta característica puede ser utilizada para variar la intensidad de un led o la velocidad de un motor de continua.
Parámetros
- `pin`: Número de pin donde escribir.
- `value`: Valor del *duty cycle*: entre 0 (siempre apagado) y 255 (siempre encendido).

map(value, fromLow, fromHigh, toLow, toHigh)
Remapea un número entero desde un rango a otro rango. Esto quiere decir que mapea la magnitud `fromLow` a `toLow` y `fromHigh` a `toHigh`, junto con todos los valores intermedios. Esta función opera en enteros, resultando truncados los valores fraccionarios en la operación. La función permite operar tanto con rangos negativos, como con rangos ascendentes o descendentes. Parámetros
- `value`: El número a mapear.
- `fromLow`: El límite inferior del valor actual.
- `fromHigh`: El límite superior del valor actual.
- `toLow`: El límite inferior del valor destino.
- `toHigh`: El límite superior del valor destino.

delay(ms)
Detiene el programa durante un tiempo especificado en milisegundos (1000 milisegundos = 1 segundo).
Parámetros
- `ms`: Cantidad de milisegundos de pausa.

Funciones de Comunicación

Serial.begin(speed)
Inicializa la comunicación serial, configurando la cantidad de bits por segundo que debe enviar. Por default envia 8 bits, sin bit de paridad y sin bit de *stop*.
Parámetros
- `speed`: cantidad de bits por segundo (baudios).

Serial.print(val)
Imprime en el canal serial el dato pasado por parámetro.
Parámetros
- `val`: Dato a imprimir, puede ser tanto un número entero, en punto flotante o una *string*.

Interrupciones

Las interrupciones son utilizadas en el sistema Arduino para el control de múltiples dispositivos y temporizadores. Parte de esta lógica puede ser delegada al desarrollador por medio de asociar rutinas a interrupciones. Las interrupciones son disparadas por el cambio de estado en pines específicos de la placa.

`attachInterrupt(interrupt, ISR, mode)`

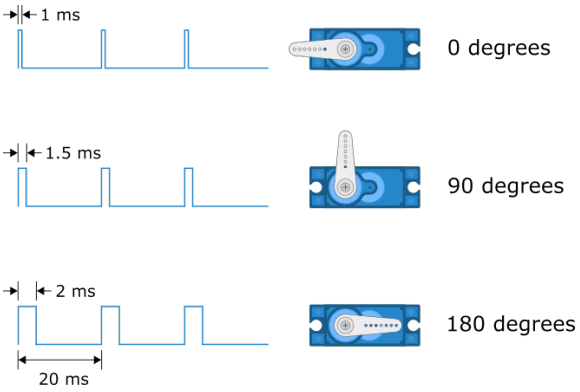
Asocia una función a una interrupción que responde al cambio de estado en un pin específico. La placa puede escuchar interrupciones sobre los pines 2, 3, 18, 19, 20 y 21. Para traducir el número de pin al número de interrupción se usa el resultado de la función `digitalPinToInterrupt(pin)` como primer parámetro para `attachInterrupt`.

Parámetros

- `interrupt`: Número de interrupción (int)
- `ISR`: La función o rutina de atención de interrupciones que se ejecutará cuando se genere la interrupción. No toma ningún parámetro ni retorna ningún resultado.
- `mode`: Define cuando la interrupción debe ser generada. Los modos validos son:
  - `LOW`: Cuando el pin está en estado bajo.
  - `CHANGE`: Cuando el pin cambia de valor de bajo a alto o a la inversa.
  - `RISING`: Cuando el pin cambia de estado bajo a estado alto.
  - `FALLING`: Cuando el pin cambia de estado alto a estado bajo.

Servo

Los servos son dispositivos actuadores compuestos por un motor, una caja reductora, un potenciómetro y una placa de control. El motor puede moverse mediante la caja reductora a una posición determinada controlada por la placa de control y el potenciómetro. Gracias al potenciómetro el servo puede medir en que posición se encuentra, y mantener está posición mediante la placa de control que aplica la energía necesaria al motor.



Los servos son controlados mediante una señal de pulsos, dependiendo del ancho del pulso, se controla su posición. Este control es generado por la biblioteca de funciones `Servo.h`. Las siguientes son las funciones básicas para utilizar está biblioteca.

`servo.attach(pin)`

Asocia una variable de tipo servo a un pin. Esta libreria está limitada para controlar servos solo sobre los pines 9 y 10.

Parámetros

- `servo`: Variable de tipo servo.
- `pin`: Número de pin al que asociar la variable servo.

`servo.write(angle)`

Escribe en una variable servo la nueva posición donde mover el servo. La posición corresponde a un angulo en grados de orientación donde moverse desde la posición anterior.

Parámetros

- `servo`: Variable de tipo servo.
- `angle`: Valor del angulo en grados para posicionar el servo. Número entre 0 y 180.

Notas complementarias

Detalle del conexionado interno del *protoboard*:

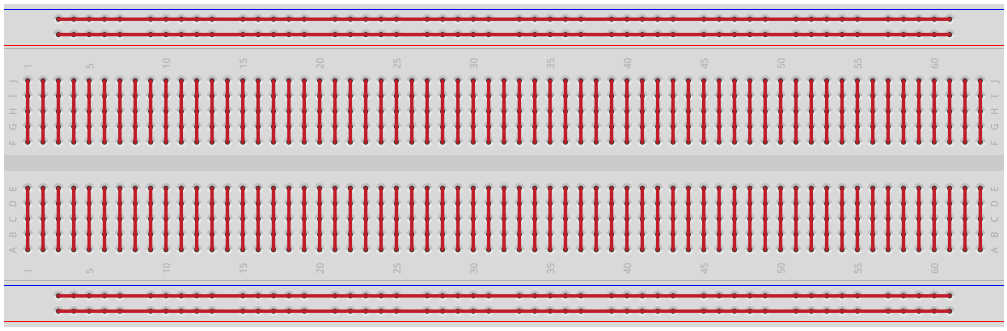


Tabla de colores de resistencias: