Montage 10

Des détecteurs de lumière

Au sommaire :

- la mesure d'une quantité de lumière par une photorésistance (LDR) ;
- la programmation de plusieurs broches comme sortie (`OUTPUT`) ;
- l'interrogation d'une entrée analogique avec l'instruction `analogRead()` ;
- le sketch complet ;
- l'analyse du schéma ;
- la réalisation du circuit ;
- la communication avec Processing ;
- un exercice complémentaire.

Comment fonctionne un détecteur de lumière ?

Dans ce projet, nous allons construire un circuit capable de réagir à des influences ou réalités extérieures. Les valeurs environnementales vraiment marquantes qui agissent sur nous sont la température et la luminosité. Toutes deux peuvent être ressenties différemment par les êtres humains et sont des impressions subjectives. L'un aura bien chaud tandis que l'autre aura la chair de poule. Il existe bien entendu des appareils ou détecteurs qui mesurent la température et la luminosité de manière objective. Notre prochain circuit sera consacré à la luminosité, que nous entendons mesurer au moyen d'une photorésis-

tance ou résistance photosensible, également appelée LDR *(Light Dependent Resistor)*.

Il s'agit d'un semi-conducteur, dont la résistance dépend de la lumière. Plus la quantité de lumière atteignant la LDR est élevée, plus la résistance est faible. Notre circuit doit commander, en fonction de la valeur de luminosité, une rangée de LED qui éclaire alors plus ou moins. Le circuit ressemble à celui de notre séquenceur de lumière, à ceci près que les différentes LED ne sont pas commandées l'une après l'autre par une boucle mais par une logique qui évalue la luminosité sur la résistance photosensible.

Composants nécessaires

	1 LDR
	10 LED rouges
	10 résistances de 330 Ω
	1 résistance de 10 kΩ
	Plusieurs cavaliers flexibles de couleurs et de longueurs diverses

Code du sketch

```
int pin[] = {2, 3, 4, 5, 6, 7, 8, 9, 10, 11};  //Tableau des broches
int analogPin = 0;   //Broche de l'entrée analogique
int analogValue = 0;  //Stocke la valeur analogique mesurée

void setup(){
  for(int i = 0; i < 10; i++)
    pinMode(pin[i], OUTPUT);
}
```

```
void loop(){
     analogValue = analogRead(analogPin) ;
     controlLEDs(analogValue);
}

//Fonction pour commander les LED
void controlLEDs(int value){
  int bargraphValue = map(value, 0, 1023, 0, 9);
  for(int i = 0; i < 10; i++)
    digitalWrite(pin[i], (bargraphValue >= i)?HIGH:LOW);
}
```

Revue de code

Du point de vue logiciel, les variables du tableau 10-1 sont nécessaires à notre atelier.

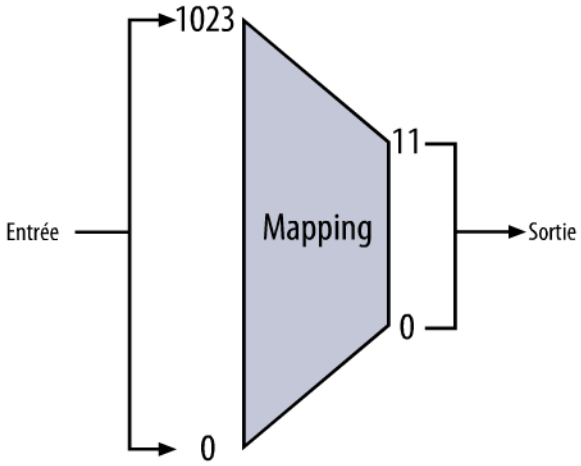
Tableau 10-1
Variables nécessaires et leur objet

Variable	Objet
`pin[]`	Stocke les numéros des broches pour commander les 10 LED.
`analogPin`	Numéro de broche pour l'entrée analogique
`analogValue`	Stocke la valeur analogique mesurée.

Le grand nombre de LED impose d'utiliser un tableau qui est stocké dans `pin[]`. La fonction `loop` lit continuellement la valeur à l'entrée analogique broche A0. La fonction `controlLEDs` est ensuite appelée avec la valeur mesurée, et se charge de commander les différentes LED. Chaque entrée analogique a une résolution de 10 bits et des valeurs comprises entre 0 et 1023 peuvent donc y être mesurées. N'utilisant cependant que 10 LED pour notre exemple, nous devons convertir le domaine des valeurs d'entrée trop grand en un domaine des valeurs de sortie adapté allant de 0 à 9 (10 LED).

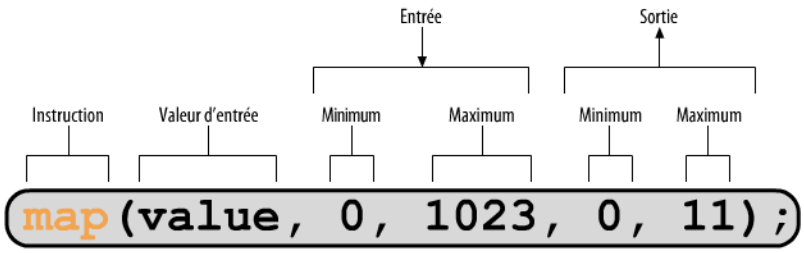
Ce code me donne l'occasion d'introduire une nouvelle instruction très intéressante qui permet de transposer un domaine de valeurs dans un autre domaine. Cette instruction est notée `map` (abréviation de *mapping* : transposition). L'image 10-1 illustre le processus de mapping réalisé par cette instruction. Il doit convertir le domaine des valeurs d'entrée, qui s'étend de 0 à 1023, en un domaine de valeurs de sortie allant de 0 à 11.

Figure 10-1 ▶
Principe du mapping



La syntaxe de l'instruction `map` est la suivante :

Figure 10-2 ▶
Instruction `map`

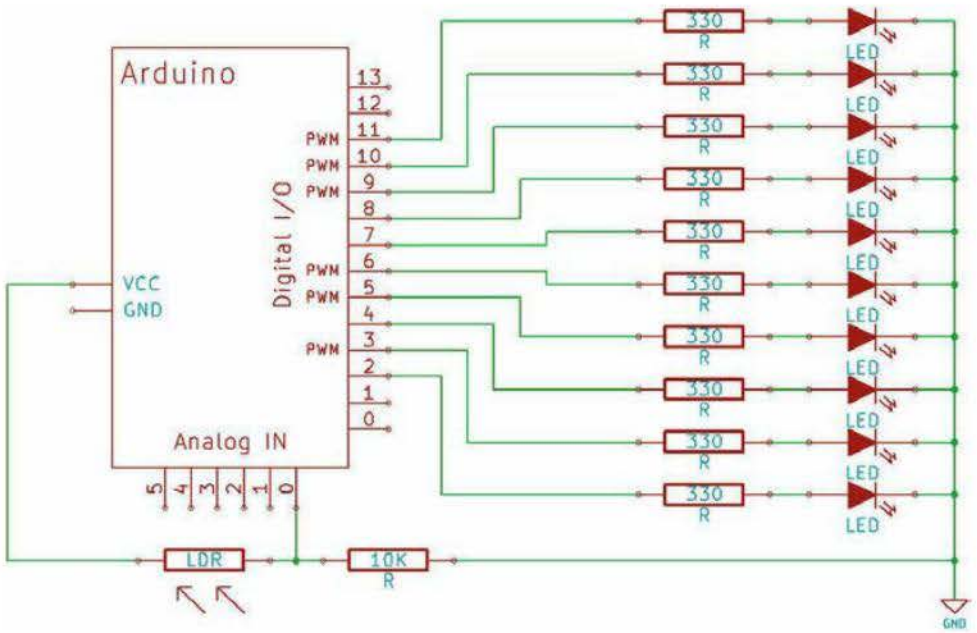


La valeur renvoyée par l'instruction `map` appartient au nouveau domaine de valeurs défini.

Ici, le résultat est consigné dans la variable `bargraphValue`. Chaque LED est ensuite commandée en la comparant à la valeur `bargraphValue` alors déterminée. Si cette valeur est supérieure ou égale au numéro de la LED en question, elle est aussitôt mise sur `HIGH` ; sinon, elle est mise sur `LOW`. Plus la valeur est élevée, plus le nombre de LED allumées est important.

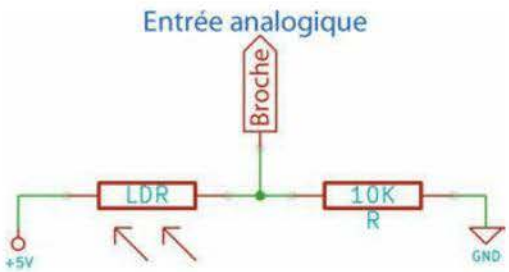
Schéma

Le schéma ressemble à s'y méprendre à celui du séquenceur de lumière, à ceci près qu'il dispose en plus d'une extension lui permettant de mesurer l'intensité lumineuse.



Vous savez déjà ce qu'est un diviseur de tension.

▲ **Figure 10-3**
Circuit pour mesurer l'énergie lumineuse



Les deux résistances (LDR et 10K) forment un diviseur de tension, la prise de tension intermédiaire est raccordée à l'entrée analogique de la carte Arduino. Le rapport des résistances et donc des tensions varie en fonction de la luminosité existante sur la LDR. La tension la plus élevée se trouve naturellement aux bornes de la plus grande résistance. Si la résistance de la LDR diminue sous l'influence d'une augmentation de la lumière, la tension à ses bornes diminue. Cela veut donc dire que la tension aux bornes de la résistance de 10K est plus élevée, laquelle se retrouve sur l'entrée analogique. Une valeur plus élevée est mesurée sur cette broche, autrement dit plus de LED s'allument. Ce processus s'inverse si moins de lumière parvient à la LDR.

Encore une fois, s'il vous plaît. Si la tension la plus élevée se trouve aux bornes de la plus grande résistance, la tension la plus élevée doit se trouver à l'entrée analogique quand la LDR s'assombrit.

Je comprends votre problème, Ardus. Je vais me servir de la figure suivante pour vous expliquer.

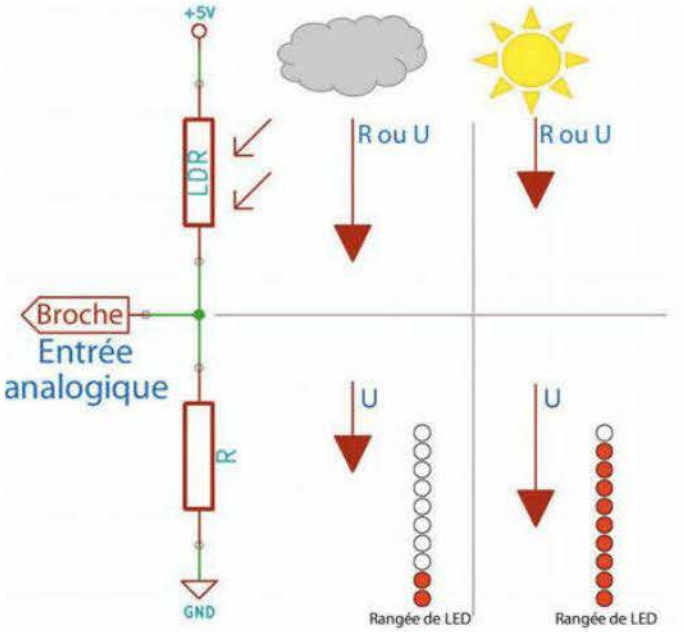


Figure 10-4 ▶
Résistances et tensions selon l'ensoleillement

La longueur des flèches indique la grandeur de la tension. Si le ciel est couvert, la résistance ou plutôt la tension est élevée sur la LDR. Mais si le soleil brille, résistance et tension sont faibles. Comme le diviseur de tension est alimenté sous 5 V, il ne reste que la différence de tension aux bornes de la résistance située à la partie inférieure. Cette tension est mesurée entre l'entrée analogique et la masse.

Pour aller plus loin

Pour compléter ce chapitre, vous pouvez effectuer une recherche sur Internet sur les mots-clés :

- LDR ;
- photorésistance ;
- résistance photosensible.

Réalisation du circuit

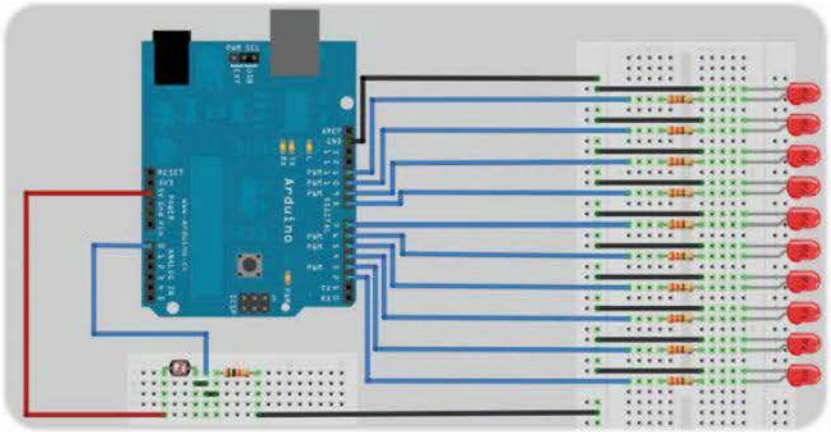


Figure 10-5
Réalisation du circuit avec Fritzing

À droite de la plaque d'essais se trouvent les 10 LED pour afficher l'intensité lumineuse, le diviseur de tension se trouve quant à lui, avec sa LDR et sa résistance fixe, sous la carte Arduino.

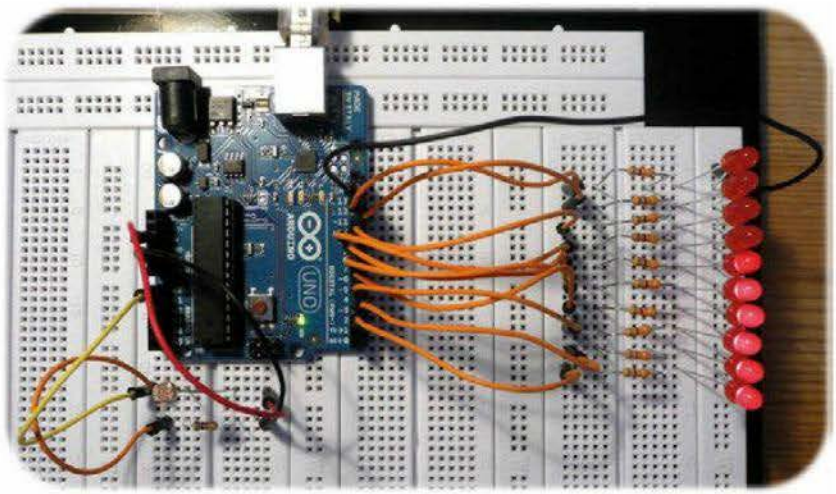


Figure 10-6
Réalisation du circuit détecteur de lumière

Nous devenons communicatifs

Il est certes intéressant à mes yeux d'observer le cycle des LED sous différentes conditions de lumière, mais le déroulement chronologique reste difficile à voir sur une longue période. Aussi voudrais-je vous présenter ici un projet qui vous plaira sûrement, puisque agréable à regarder.

Le langage de programmation Processing s'impose quand il s'agit de générer des graphiques. Vous trouverez l'environnement de développement pour le langage de programmation Processing sur le site Internet *http://processing.org*. Le côté pratique de la chose est que, tout comme pour Arduino, il suffit de décompresser le fichier téléchargé dans un répertoire. Aucune installation n'est à faire. Ce qui prendrait un temps fou avec des langages de programmation tels que C++ ou C# sera plus vite fait et sera moins fastidieux avec Processing. Je vous montre d'emblée le résultat dans la fenêtre graphique de Processing pour que vous sachiez à quoi vous attendre.

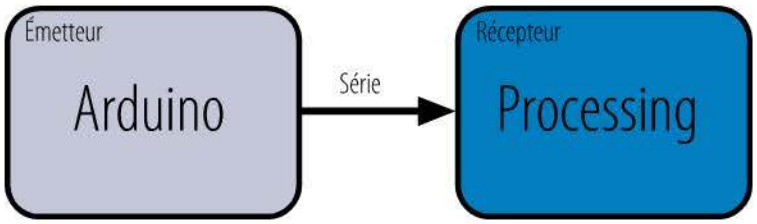
Figure 10-7 ▶
Courbe des valeurs de l'énergie lumineuse dans la fenêtre graphique de Processing

Les valeurs affichées sont actualisées en permanence, la courbe évoluant de droite à gauche dans la fenêtre. Les nouvelles valeurs apparaissent à droite et les anciennes valeurs disparaissent à gauche de la fenêtre.

Pouvez-vous me dire comment deux langages de programmation différents font pour échanger des données ?

J'allais y venir, Ardus. Une base commune leur permettant de se comprendre doit être définie. L'interface série vous dit sûrement déjà quelque chose. Tout langage de programmation ou presque a dans son vocabulaire des instructions pour envoyer ou recevoir par cette interface.

Notre exemple montre un émetteur et un récepteur. La communication est unidirectionnelle, autrement dit se fait dans un seul sens. Certes, l'interface est capable de communiquer presque simultanément dans les deux sens, mais nous nous contenterons d'un seul.



Tout ce que votre carte Arduino doit faire c'est enregistrer les valeurs mesurées et envoyer les données via l'interface série. Plus vite à dire qu'à faire ?! Non, pas du tout car la plupart du travail de calcul se fait du côté de Processing. Voyons d'abord du côté de l'émetteur ce qu'Arduino doit faire.

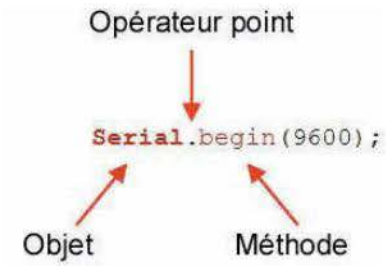
Arduino l'émetteur

Côté matériel, il ne vous faut que le diviseur de tension avec sa LDR et sa résistance de 10 kΩ fixe, connecté à l'entrée analogique broche 0, pour envoyer les valeurs de luminosité à l'interface série.

```
void setup(){
    Serial.begin(9600);
}

void loop(){
   Serial.println(analogRead(0));
}
```

Voyons maintenant ce que ce bout de code donne. Dans la fonction `setup`, l'interface série est préparée pour la transmission. Vous avez eu vos premiers contacts avec la programmation orientée objet lors de la création de votre propre bibliothèque dans le montage n° 9. L'interface série est considérée comme étant un objet logiciel nommé `Serial`. Vous disposez de quelques méthodes, que nous entendons maintenant utiliser.



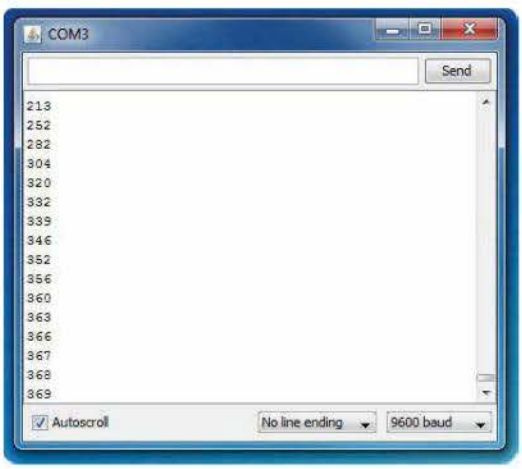
La méthode pour initialiser l'interface a pour nom `begin` et reçoit une valeur qui détermine la vitesse de la transmission. Il s'agit dans notre cas de `9600`. L'unité de mesure est ici le baud, le nombre indiquant la cadence. 1 baud signifie 1 changement d'état par seconde. Lisez la littérature technique ou allez sur Internet pour d'autres informations. La deuxième méthode que nous utiliserons se nomme `println`. Elle envoie la valeur qui lui a été transmise à l'interface série. Il s'agit de la valeur mesurée sur la broche analogique `0` dans notre bout de sketch. L'interrogation de la broche analogique et la transmission à l'interface ont lieu continuellement dans la fonction `loop`.

Attention !

Pour que la communication entre émetteur et récepteur puisse se faire, le réglage du taux de transfert doit être le même sur les deux stations.

Vous pouvez suivre la transmission des valeurs déterminées en temps réel, de préférence en ouvrant le Serial Monitor de l'environnement de développement.

Figure 10-8 ▶
Édition des données dans le Serial Monitor



Cela fonctionne évidemment aussi avec n'importe quel autre programme de terminal ayant accès à l'interface série. Pensez ici aussi à régler le taux de transfert tout en bas à droite de la fenêtre.

Processing le récepteur

Venons en maintenant au programme en question, chargé principalement de représenter graphiquement les valeurs reçues. Le code est assez conséquent, mais voici quand-même une courte description pour que vous ne soyez pas perdu.

```
import processing.serial.*;

Serial mySerialPort;
int xPos = 1;
int serialValue;
int[] yPos;

void setup(){
  size(400, 300);
  println(Serial.list());
  mySerialPort = new Serial(this, Serial.list()[0], 9600);
  mySerialPort.bufferUntil ('\n');
  //Set initial background
  background(0);
  yPos = new int[width];
}

void draw(){
  background(0);
  stroke(255, 255, 0, 120);
  for(int i=0; i < width; i+=50)
    line(i, 0, i, height);
  for(int i=0; i < height; i+=50)
    line(0, i, width, i);

  stroke(255, 0, 0);
  strokeWeight(1);
  int yPosPrev = 0, xPosPrev = 0;
  println(serialValue);
  //Décaler les valeurs de l'array vers la gauche
  for(int x = 1; x < width; x++)
  yPos[x-1] = yPos[x];
  //Joindre les nouvelles coordonnées de la souris
  //à l'extrémité droite de l'array
  yPos[width - 1] = serialValue;
  //Affichage de l'array
   for(int x = 0; x < width; x++){
     if(x > 0)
       line(xPosPrev, yPosPrev, x, yPos[x]);
     xPosPrev = x;        //Stockage de la dernière position x
     yPosPrev = yPos[x];  //Stockage de la dernière position y
   }
}

void serialEvent(Serial mySerialPort){
```

```
  String portStream = mySerialPort.readString();
  float data = float(portStream);
  serialValue = height - (int)map(data, 0, 1023, 0, height);
}
```

Processing comprend également deux fonctions principales, qui ressemblent à celles d'Arduino.

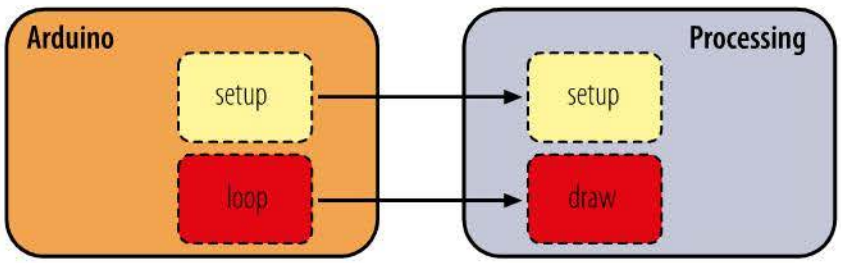


Figure 10-9 ▶ Correspondance entre deux fonctions principales

La fonction `setup` est appelée une seule fois elle aussi en début de sketch et sert à initialiser des variables. La fonction `draw` est une boucle sans fin semblable à la fonction `loop` dans Arduino, qui doit son nom au fait qu'elle sert à dessiner (en anglais : *to draw*) les éléments graphiques dans la fenêtre d'édition. Pour pouvoir traiter l'interface série dans Processing, vous devez écrire la ligne :

```
import processing.serial.*;
```

qui permet d'importer un paquet en langage Java. Eh oui, Processing est un langage basé sur Java, contrairement à Arduino qui, lui, utilise C ou C++.

Les deux langages ont une syntaxe très similaire, aussi la programmation dans Processing ne semble-t-elle pas compliquée à ceux qui connaissent bien C ou C++. Si vous écrivez la ligne :

```
println(Serial.list());
```

Processing vous donne une liste de toutes les interfaces série disponibles. L'affichage dans la fenêtre de messagerie ressemble alors à ceci.

```
Native lib Version = RXTX-2.1-7
Java lib Version   = RXTX-2.1-7
[0] "COM3"
[1] "COM4"
```

Elle indique que deux ports sériels sont disponibles. Arduino utilisant le premier port COM3 correspondant à la première entrée de la liste `[0]`, cet index est reporté dans la ligne ci-après :

```
mySerialPort = new Serial(this, Serial.list()[0], 9600);
```

Attention !
Si, sur votre ordinateur, le port série utilisé n'est pas le premier de la liste, il faut alors remplacer dans la ligne précédente `Serial.list()[0]` par `"COMn"`, où n est le numéro du port série effectivement utilisé par Arduino.

Un nouvel objet sériel est ainsi généré, et instancié avec le mot-clé `new` dans Processing. La valeur `9600` apparaît une fois de plus, elle doit correspondre à celle qui est dans le sketch Arduino. Tous les éléments graphiques tels que trame de fond et courbe sont alors dessinés dans la fonction `draw`. Les valeurs Arduino transmises s'accumulent dans la fonction `serialEvent` et sont stockées dans la variable `serialValue`. Cette variable sert à dessiner la courbe dans la fonction `draw`.

Attention !
Si vous avez ouvert un programme de terminal, par exemple Serial Monitor, pour visualiser les valeurs envoyées par Arduino, vous aurez des problèmes si vous démarrez en même temps l'affichage graphique des valeurs dans Processing. Le port COM concerné est en effet exclusivement utilisé par le programme de terminal, interdisant tout accès supplémentaire par un autre programme. Fermez par conséquent le programme de terminal avant de démarrer Processing pour évaluer les données.

Problèmes courants

Si les différentes LED ne réagissent pas aux modifications des conditions lumineuses ou si aucun changement du tracé de la courbe n'est à observer par la suite dans la fenêtre d'édition, il peut y avoir plusieurs raisons.

- Vérifiez que vos fiches de raccordement sur la plaque d'essais correspondent bien au circuit.
- Vérifier qu'il n'y a pas de court-circuit entre elles.
- Les résistances ont-elles bien les bonnes valeurs ?
- Toutes les LED sont-elles correctement polarisées ?
- Vérifiez encore une fois l'exactitude du code du sketch côté Arduino et côté Processing.
- Ouvrez le Serial Monitor dans l'IDE Arduino pour vous assurer que des valeurs différentes sont transmises à l'interface série, lorsque les conditions lumineuses varient. Dans le code de Processing, vous pouvez ajouter la ligne `println(serialValue)` de manière à ce que les valeurs transmises (pour peu qu'elles le soient) s'affichent également dans la fenêtre de messagerie.
- Vérifiez que l'interface série utilisée n'est pas bloquée par un autre processus et que seul Processing y accède.

Qu'avez-vous appris ?

- Vous savez comment interroger une entrée analogique à laquelle est reliée une résistance photosensible (LDR) avec l'instruction `analogRead`.
- Un diviseur de tension sert à diviser la tension appliquée à ses bornes dans un rapport déterminé. Nous avons utilisé cette propriété pour amener à l'entrée analogique une tension fonction de l'intensité lumineuse.
- Vous savez comment échanger des données entre deux programmes au moyen de l'interface série. L'émetteur est ici la carte Arduino et le récepteur un sketch Processing reproduisant visuellement les données reçues sous forme de courbe.

Exercice complémentaire

Créez un sketch Arduino faisant clignoter régulièrement toutes les LED concernées quand par exemple un certain seuil de flux lumineux est franchi, pour vous avertir qu'un état critique est maintenant atteint et qu'une crème solaire avec indice de protection 75 + doit être utilisée.