Montage 5

Le séquenceur de lumière

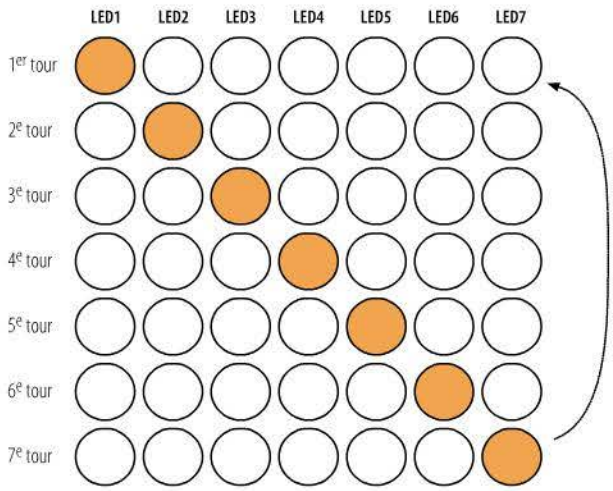
Au sommaire :

- la déclaration et l'initialisation d'un tableau (`array`) ;
- la programmation de plusieurs broches comme sortie (`OUTPUT`) ;
- l'utilisation d'une boucle `for` ;
- le sketch complet ;
- l'analyse du schéma ;
- la réalisation du circuit ;
- un exercice complémentaire.

Qu'est-ce qu'un séquenceur de lumière ?

Vous maîtrisez maintenant suffisamment les LED pour être en mesure de réaliser des montages où clignotent plusieurs diodes électroluminescentes. Ça n'a l'air de rien dit comme ça, mais ce n'est pas si simple. Nous allons commencer par un séquenceur de lumière, qui commande une par une différentes LED. Dans ce montage, les LED branchées sur les broches numériques devront s'allumer conformément au modèle présenté sur la figure 5-1.

Figure 5-1 ▶
Séquence d'allumage des 7 LED



À chaque tour, la LED s'allume une position plus à droite. Arrivé à la fin, le cycle reprend au début. Vous pouvez programmer les diverses broches, qui toutes sont censées servir de sortie, de différentes manières. Dans l'état actuel de vos connaissances, vous devez déclarer sept variables et les initialiser avec les valeurs de broche correspondantes. Ce qui pourrait donner ceci :

```
int ledPin1 = 7;
int ledPin2 = 8;
int ledPin3 = 9;
…
```

Chaque broche doit être ensuite programmée dans la fonction `setup` avec `pinMode` comme sortie, ce qui représente aussi un travail de saisie considérable :

```
pinMode(ledPin1, OUTPUT);
pinMode(ledPin2, OUTPUT);
pinMode(ledPin3, OUTPUT);
…
```

Voici donc la solution. Je voudrais vous présenter un type intéressant de variable, capable de mémoriser plusieurs valeurs du même type de donnée sous un même nom.

Vous rigolez ! Comment une variable peut-elle mémoriser plusieurs valeurs sous un seul et même nom ? Et comment dois-je faire pour sauvegarder ou appeler les différentes valeurs ?

Patience ! C'est possible. Cette forme spéciale de variable est appelée tableau (`array`). On n'y accède pas seulement par son nom évocateur, car une telle variable possède aussi un index. Cet index est un nombre entier incrémenté. Ainsi, les différents éléments du tableau – c'est le nom donné aux valeurs stockées – peuvent être lus ou modifiés. Vous allez voir comment dans le code du sketch ci-après.

Composants nécessaires

	7 LED rouges
	7 résistances de 330 Ω
	Plusieurs cavaliers flexibles de couleurs et de longueurs diverses

Code du sketch

Voici le code du sketch pour commander le séquenceur de lumière à sept LED :

```
int ledPin[] = {7, 8, 9, 10, 11, 12, 13}; //Tableau de LED avec
                                          //numéros des broches
int waitTime = 200; // Pause entre les changements en ms
void setup()
{
  for(int i = 0; i < 7; i++)
    pinMode(ledPin[i], OUTPUT); //Toutes les broches du tableau comme
                                //sorties
}

void loop()
{
  for(int i = 0; i < 7; i++)
  {
    digitalWrite(ledPin[i], HIGH); //Élément du tableau au niveau HIGH
    delay(waitTime);
    digitalWrite(ledPin[i], LOW); //Élément du tableau au niveau LOW
  }
}
```

Revue de code

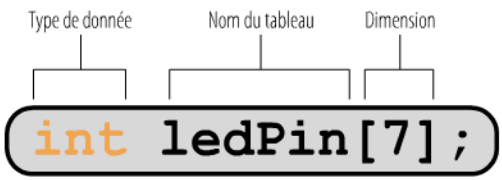
Les variables suivantes sont techniquement nécessaires à notre programmation expérimentale.

Tableau 5-1 ▶
Variables nécessaires et leur objet

Variable	Objet
`ledPin`	Tableau pour enregistrer les différentes broches sur lesquelles les LED sont branchées.
`waitTime`	Contient le temps d'attente entre les changements de LED (en ms).

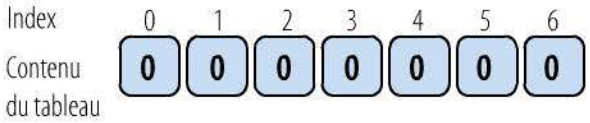
Dans le sketch du séquenceur de lumière, vous rencontrez pour la première fois un tableau et une boucle. Cette dernière est nécessaire pour accéder confortablement aux différents éléments du tableau par le biais des numéros de broche. D'une part les broches sont toutes programmées en tant que sorties, et d'autre part les sorties numériques sont sélectionnées. L'accès à chaque élément se fait par un index et comme la boucle utilisée ici dessert automatiquement un certain domaine de valeurs, cette construction est idéale pour nous. Commençons par la variable de type `array`. La déclaration ressemble à celle d'une variable normale, à ceci près que le nom doit être suivi d'une paire de crochets.

Figure 5-2 ▶
Déclaration du tableau



- Le type de donnée définit quel type les différents éléments du tableau doivent avoir.
- Le nom du tableau est un nom évocateur pour accéder à la variable.
- Le nombre entre les crochets indique combien d'éléments le tableau doit contenir.

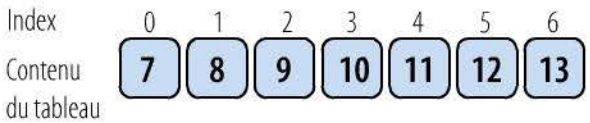
Vous pouvez imaginer un tableau comme un meuble à plusieurs tiroirs. Chaque tiroir est surmonté d'une étiquette portant un numéro d'ordre. Si je vous donne par exemple pour instruction d'ouvrir le tiroir numéro 3 et de regarder ce qu'il y a dedans, les choses sont plutôt claires non ? Il en va de même pour le tableau.



Tous les éléments de ce tableau ont été implicitement initialisés avec la valeur `0` après la déclaration. L'initialisation peut toutefois être faite de deux manières différentes. Nous avons choisi la manière facile et les valeurs, dont le tableau est censé être pourvu, sont énumérées derrière la déclaration entre deux accolades et séparées par des virgules :

```
int ledPin[] = {7, 8, 9, 10, 11, 12, 13};
```

Sur la base de cette ligne d'instruction, le contenu du tableau est le suivant.



N'avons-nous pas oublié quelque chose d'important ? Dans la déclaration du tableau, il n'y a rien entre les crochets. La taille du tableau devrait pourtant y être indiquée.

C'est vrai, mais le compilateur connaît déjà dans le cas présent – par les informations fournies pour l'initialisation faite dans la même ligne – le nombre d'éléments. Aussi la dimension du tableau n'a-t-elle pas besoin d'être indiquée. L'initialisation, quelque peu fastidieuse, consiste à affecter explicitement les différentes valeurs à chaque élément du tableau :

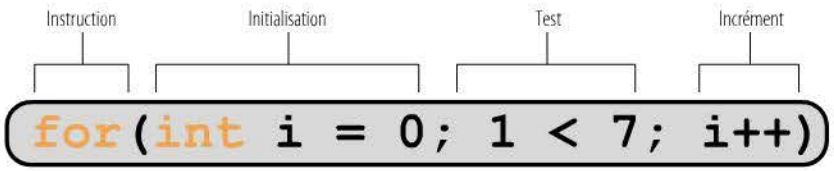
```
int ledPin[7];    //Déclaration du tableau avec 7 éléments
void setup()
{
   ledPin[0] = 7;
   ledPin[1] = 8;
   ledPin[2] = 9;
   ledPin[3] = 10;
   ledPin[4] = 11;
   ledPin[5] = 12;
   ledPin[6] = 13;
   // …
}
```

Attention !
L'index du premier élément du tableau est toujours le chiffre 0. Si, par exemple, vous déclarez un tableau de 10 éléments, l'index admis le plus élevé sera le chiffre 9 – soit toujours un de moins que le nombre d'éléments.

Si vous ne vous en tenez pas à cette règle, vous pouvez provoquer une erreur à l'exécution que le compilateur, qui se cache derrière l'environnement de développement, ne détecte ni au moment du développement ni plus tard pendant l'exécution, c'est pourquoi vous devez redoubler d'attention.

Venons-en maintenant à la boucle et regardons la syntaxe de plus près.

Figure 5-3 ▶
Boucle `for`



La boucle introduite par le mot-clé `for` est appelée boucle `for`. Suivent entre parenthèses certaines informations sur les caractéristiques-clés.

- Initialisation : à partir de quelle valeur la boucle doit-elle commencer à compter ?
- Test : jusqu'à combien doit-elle compter ?
- Incrément : de combien la valeur initiale doit-elle être modifiée ?

Ces trois informations déterminent le comportement de la boucle `for` et définissent son comportement au moment de l'appel.

Pour aller plus loin
Une boucle `for` est utilisée la plupart du temps quand on connaît au départ le nombre de fois que certaines instructions doivent être exécutées. Ces caractéristiques sont définies dans ce que l'on appelle l'en-tête de boucle, qui correspond à ce qui est entre parenthèses.

Mais soyons plus concrets. La ligne de code suivante :

```
for(int i = 0; i < 7; i++)
```

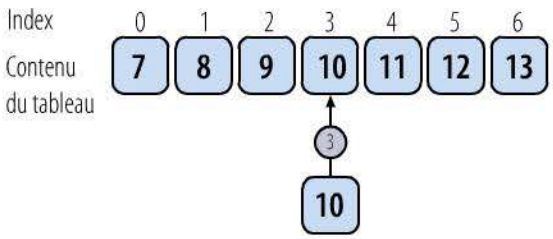
déclare et initialise une variable `i` du type `int` avec la valeur `0`. L'indication du type de donnée dans la boucle stipule qu'il s'agit d'une variable locale qui n'existe que tant que la boucle `for` itère, c'est-à-dire suit son cours. La variable `i` est effacée de la mémoire à la sortie de la boucle.

Le nom exact d'une telle variable dans une boucle est « variable de contrôle ». Elle parcourt une certaine zone tant que la condition (`i < 7`) – désignée ici sous le nom de « test » – est remplie. Une mise à jour de la variable est ensuite effectuée selon l'expression de l'incrément. L'expression `i++` ajoute la valeur `1` à la variable `i`.

> Vous avez utilisé l'expression `i++`. Pouvez-vous m'expliquer exactement ce qu'elle signifie ? Elle doit augmenter la valeur de `1`, mais son écriture est étrange.

Les signes `++` sont un opérateur qui ajoute la valeur `1` au contenu de l'opérande, donc à la variable. Les programmeurs sont paresseux de naissance et font tout pour formuler au plus court ce qui doit être tapé. Quand on pense au nombre de lignes de code qu'un programmeur doit taper dans sa vie, moins il y a de caractères et mieux c'est. Il s'agit aussi à terme de consacrer plus de temps à des choses plus importantes – par exemple encore plus de code – en adoptant un mode d'écriture plus court. Toujours est-il que les deux expressions suivantes ont exactement le même effet : `i++;` et `i = i + 1;`

Deux caractères de moins ont été utilisés, ce qui représente tout de même une économie de 40 %. Mais revenons-en au texte. La variable de contrôle `i` sert ensuite de variable d'index dans le tableau et traite ainsi l'un après l'autre les différents éléments de ce tableau.



Sur cette capture d'écran d'une itération de la boucle, la variable `i` présente la valeur `3` et a donc accès au 4e élément dont le contenu est `10`. Autrement dit, toutes les broches consignées dans le tableau `ledPin` sont programmées en tant que sorties dans la fonction `setup` au moyen des deux lignes suivantes :

```
for(int i = 0; i < 7; i++)
  pinMode(ledPin[i], OUTPUT);
```

Une chose importante encore : si, dans une boucle `for`, il n'y a aucun bloc d'instructions, formé au moyen d'accolades (comme nous en verrons un bientôt dans la fonction `loop`), seule la ligne venant immé-

diatement après la boucle `for` est prise en compte par cette dernière. Le code de la fonction `loop` contient seulement une boucle `for` dont la structure de bloc donne cependant accès à plusieurs instructions :

```
for(int i = 0; i < 7; i++)
{
  digitalWrite(ledPin[i], HIGH);  //Élément de tableau au niveau HIGH
  delay(waitTime);
  digitalWrite(ledPin[i], LOW);   //Élément de tableau au niveau LOW
}
```

Je voudrais vous montrer dans un court sketch comment la variable de contrôle `i` est augmentée (incrémentée) :

```
void setup(){
  Serial.begin(9600);         //Configuration de l'interface série
  for(int i = 0; i < 7; i++)
    Serial.println(i);        //Impression sur l'interface série
}

void loop(){/*   vide   */}
```

Puisque notre Arduino n'a pas de fenêtre d'affichage, nous devons trouver autre chose. L'interface série sur laquelle il est quasiment branché peut nous servir à envoyer des données. L'environnement de développement dispose d'un Serial Monitor capable de recevoir et d'afficher ces données sans problème. Vous pouvez même l'utiliser pour envoyer des données à la carte Arduino. Vous en saurez plus bientôt. Le code initialise par l'instruction suivante :

```
Serial.begin(9600);
```

l'interface série avec une vitesse de transmission de 9 600 bauds.

La ligne suivante :

```
Serial.println(i);
```

envoie ensuite au moyen de la fonction `println` la valeur de la variable `i` à l'interface. Il ne vous reste plus qu'à ouvrir le Serial Monitor pour afficher les valeurs de la figure 5-4.

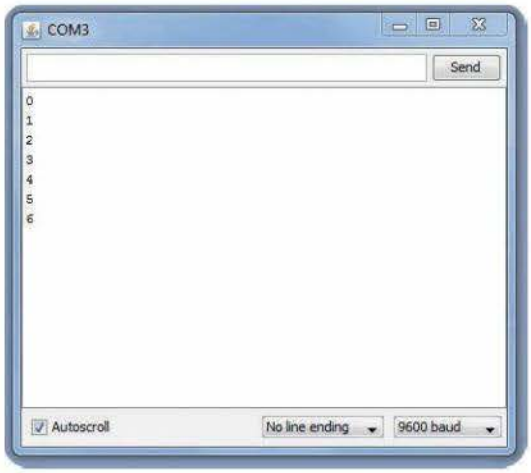


Figure 5-4
Impression des valeurs dans le Serial Monitor

On voit ici comment les valeurs de la variable de contrôle `i`, dont nous avons besoin dans notre sketch pour sélectionner les éléments du tableau, sont imprimées de `0` à `6`. J'ai placé le code dans la fonction `setup` pour que la boucle `for` ne soit exécutée qu'une fois et ne s'affiche pas constamment. La figure 5-5 montre de plus près les différents passages de la boucle `for`.

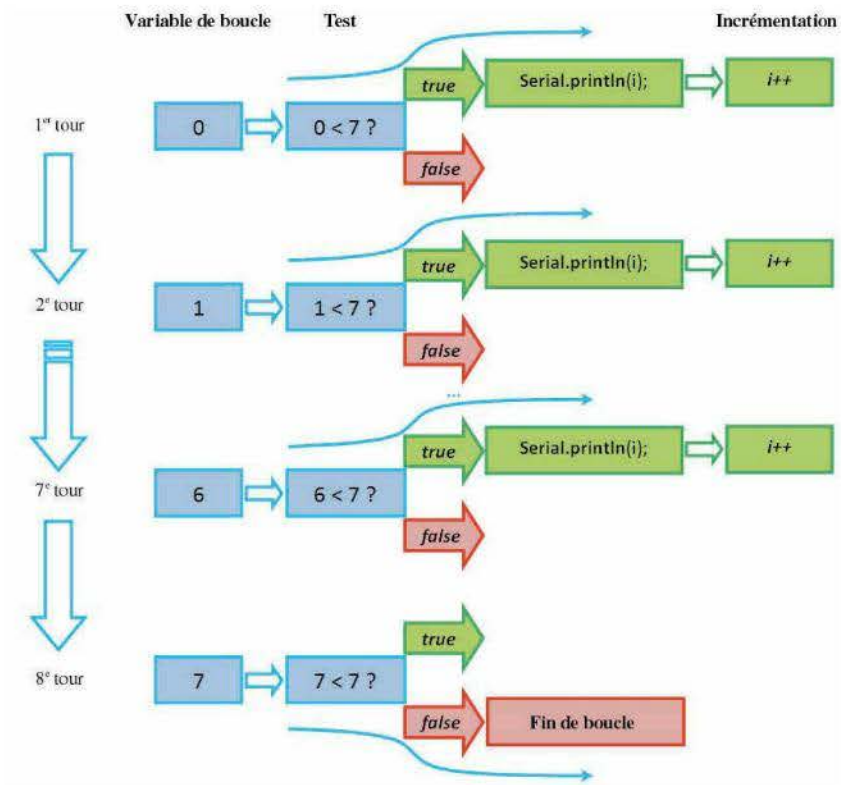


Figure 5-5
Comportement de la boucle `for`

Eh là, pas si vite ! Le code de programmation de l'interface série c'est du chinois pour moi. On y trouve `Serial`, `begin` ou encore `println` avec un point entre les deux. Qu'est-ce que ça veut dire ?

Vous aimez bien tout comprendre et ce n'est pas pour me déplaire ! Très bien ! Il me faut maintenant parler de la programmation orientée objet, car elle va me servir à vous expliquer la syntaxe. Nous reviendrons plus tard sur ce mode de programmation puisque C++ est un langage orienté objet (ou OOP sous sa forme abrégée). Ce langage est tourné vers la réalité constituée d'objets réels tels que par exemple table, lampe, ordinateur, barre de céréales, etc. Aussi les programmeurs ont-ils défini un « objet » représentant l'interface série. Ils ont donné à cet objet le nom de `Serial`, et il est utilisé à l'intérieur d'un sketch. Chaque objet possède cependant d'une part certaines caractéristiques (telles que la couleur ou la taille) et d'autre part un ou plusieurs comportements qui définissent ce qu'on peut faire avec cet objet. Dans le cas d'une lampe, le comportement serait par exemple le fait de s'allumer ou de s'éteindre. Mais revenons à notre objet `Serial`. Le comportement de cet objet est géré par de nombreuses fonctions qui sont appelées méthodes en programmation orientée objet (OOP). Deux de ces méthodes vous sont déjà familières : la méthode `begin` qui initialise l'objet `Serial` avec le taux de transmission voulu, et la méthode `println` (*print line* signifie en quelque sorte imprimer et faire un saut de ligne) qui envoie quelque chose sur l'interface série. Le lien entre objet et méthode est assuré par l'opérateur point (`.`) qui les relie ensemble. Quand je dis par conséquent que `setup` et `loop` sont des fonctions, ce n'est qu'une demi-vérité car il s'agit, à bien y regarder, de méthodes.

Pour aller plus loin

Vous savez maintenant comment envoyer quelque chose à l'interface série. Vous pouvez vous en servir pour trouver une ou plusieurs erreurs dans un sketch. Si le sketch ne fonctionne pas comme prévu, placez des instructions d'écriture sous forme de `Serial.println()` à divers endroits qui vous paraissent importants dans le code et imprimez certains contenus de variable ou encore des textes. Vous pouvez ainsi savoir ce qui se passe dans votre sketch et pourquoi il ne marche pas bien. Vous devez seulement apprendre à interpréter les données imprimées. Ce n'est pas toujours facile et il faut un peu d'entraînement.

Schéma

Le schéma montre les différentes LED avec leur résistance série de 330 ohms.

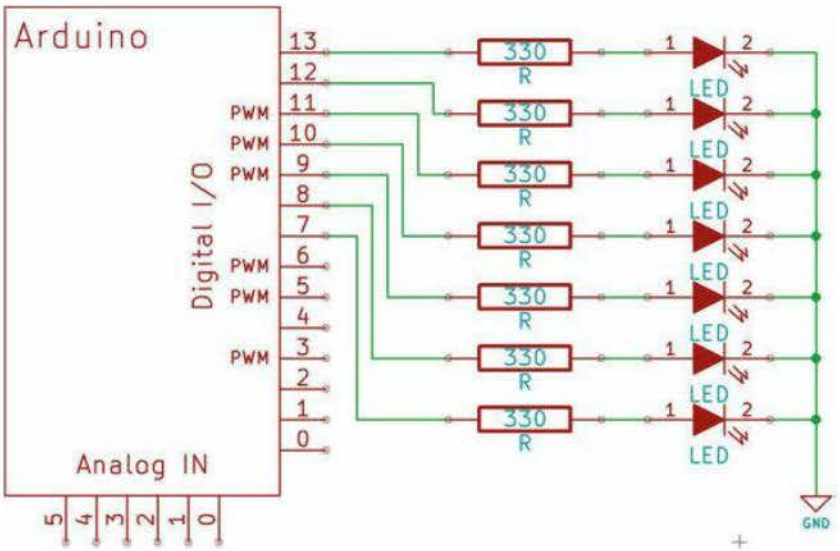


Figure 5-6
Carte Arduino commandant 7 LED pour un séquenceur de lumière

Réalisation du circuit

Votre plaque d'essais accueille toujours plus de composants électroniques sous forme de résistances et diodes électroluminescentes.

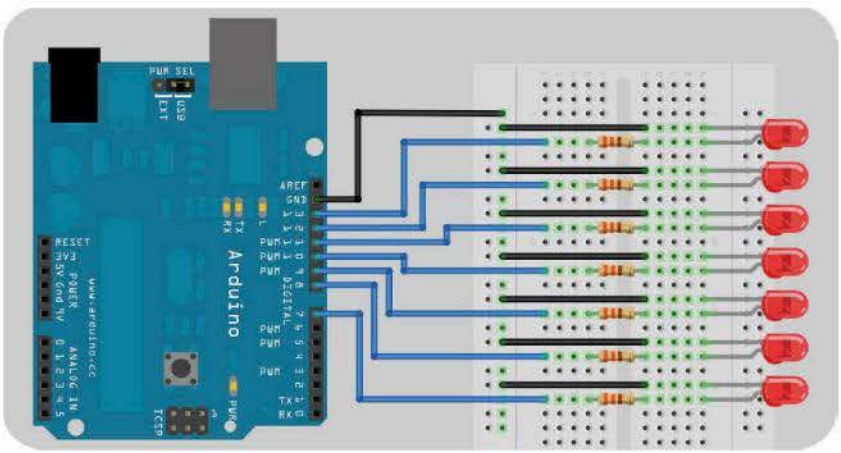


Figure 5-7
Réalisation du circuit de séquenceur de lumière avec Fritzing

Attention !
Quand vous branchez des composants électroniques tout près les uns des autres, comme c'est ici le cas, soyez très attentif car il arrive souvent de se tromper et d'occuper le trou voisin sur la plaque, si bien que le circuit ne fonctionne qu'en partie, voire pas du tout. Cela devient sérieux si vous travaillez avec les lignes d'alimentation et de masse placées l'une à côté de l'autre. Des problèmes peuvent aussi résulter de cavaliers flexibles mal enfoncés dans leur trou, dont les fils conducteurs dénudés ressortent en partie. Des courts-circuits peuvent se produire quand on bouge ces cavaliers, lesquels peuvent tout abîmer. Il faut donc se monter soigneux.

Problèmes courants

Si les LED ne s'allument pas l'une après l'autre, débranchez le port USB de la carte pour plus de sécurité et vérifiez ce qui suit.

- Vos fiches de raccordement sur la plaque correspondent-elles vraiment au circuit ?
- Pas de court-circuit éventuel entre elles ?
- Les LED ont-elles été mises dans le bon sens ? Autrement dit, la polarité est-elle correcte ?
- Les résistances ont-elles bien les bonnes valeurs ?
- Le code du sketch est-il correct ?

Qu'avez-vous appris ?

- Vous avez fait la connaissance d'une forme spéciale de variable vous permettant d'enregistrer plusieurs valeurs d'un même type de donnée. Elle est appelée tableau (`array`). On accède à ses différents éléments au moyen d'un index.
- La boucle `for` vous permet d'exécuter plusieurs fois une ou plusieurs lignes de code. Elle est gérée par une variable de contrôle, active dans la boucle et initialisée avec une certaine valeur initiale. Une condition vous a permis de définir pendant combien de temps la boucle doit s'exécuter. Vous contrôlez ainsi quel domaine de valeurs la variable traite.
- Vous pouvez réunir plusieurs instructions, qui sont ensuite toutes exécutées par exemple dans le cas d'une boucle `for`, en constituant un bloc au moyen de la paire d'accolades.
- La variable de contrôle, dont nous venons de parler, est utilisée pour modifier l'index d'un tableau et accéder ainsi à ses différents éléments.

Exercice complémentaire

Dans cet exercice, je souhaiterais que vous fassiez clignoter le séquenceur de lumière de différentes manières. Je vous propose :

- toujours dans le même sens, une LED s'allumant à tour de rôle (c'est le montage que vous venez de voir) ;
- dans un sens puis dans l'autre, une ou plusieurs LED s'allumant à tour de rôle ;
- dans les deux sens en même temps (la LED 1 s'allumant en même temps que la LED 7 au premier tour, puis la LED 2 s'allumant au même moment que la LED 6 au deuxième tour et ainsi de suite) ;
- à chaque tour, une LED s'allume de manière aléatoire.

Pour commander une LED au hasard, vous aurez besoin d'une autre fonction que vous ne connaissez pas encore. Elle se nomme `random`, ce qui signifie « aléatoire » ou « au hasard ». Il existe deux syntaxes possibles pour cette fonction.

1re syntaxe

Vous utiliserez la syntaxe suivante pour générer une valeur au hasard dans un domaine compris entre `0` et une limite supérieure :

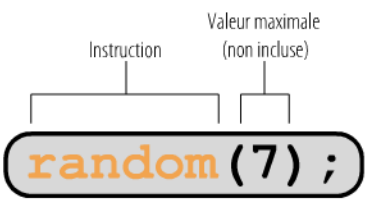


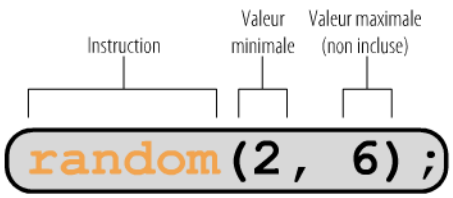
Figure 5-8
Instruction `random` (avec un argument)

Attention, la valeur maximale que vous indiquez sera toujours non incluse. Dans cet exemple, vous générerez ainsi des chiffres au hasard entre `0` et `6` inclus.

2e syntaxe

Vous utiliserez la syntaxe suivante pour générer une valeur au hasard comprise entre une limite inférieure et une limite supérieure.

Figure 5-9 ▶
Instruction `random` (avec deux arguments)



Cette instruction générera des valeurs comprises entre 2 et 5 inclus, la valeur la plus élevée étant ici aussi exclue. Cette particularité pourra surprendre certains, mais il n'est pas possible de faire autrement.