Chapitre 3 : Les Fonctions dans le Langage Arduino

I. Le concept de fonction

Une fonction est en quelque sorte un programme dans le programme, et on l'appelle d'ailleurs également sous-programme. Lorsque vous avez défini une fonction, vous pouvez l'appeler ensuite depuis n'importe quel endroit dans le programme. Une fonction peut posséder ses propres variables et ses propres instructions. Une fois ces instructions exécutées, le flux d'exécution revient à l'endroit qui suit immédiatement la ligne qui appelait la fonction.

Modifions notre programme élémentaire qui provoquait 20 clignotements pour définir une nouvelle fonction que nous choisissons de baptiser clignoter() :

```
int brocheLED = 13;
int dureePause = 250;
void setup()
{
    pinMode(brocheLED, OUTPUT);
}
void loop()
{
    for (int i = 0; i < 20; i ++)
    {
        clignoter();
    }
    delay(3000);
 }
void clignoter() // Nouvelle fonction
{
    digitalWrite(brocheLED, HIGH);
    delay(dureePause);
    digitalWrite(brocheLED, LOW);
    delay(dureePause);
}
```

Vous remarquez les parenthèses vides juste après le nom de la fonction. Elles permettent de confirmer que cette fonction n'attend aucun paramètre d'entrée. La durée de clignotement qu'elle utilise est définie par la variable dureePause que nous avons laissée en début de programme.

II. Paramètres d'une fonction

Voyons ici comment doter la fonction clignoter() de plusieurs paramètres qui lui permettront de savoir combien de fois elle doit clignoter et quelle est la durée de chaque clignotement. Étudions d'abord l'exemple suivant.

```
int brocheLED = 13;
int dureePause = 250;
void setup()
{
    pinMode(brocheLED, OUTPUT);
}
void loop()
{
    clignoter(20, dureePause);
    delay(3000);
}
void clignoter(int nbrEclairs, int d)
{
    for (int i = 0; i < nbrEclairs; i ++)
    {
        digitalWrite(brocheLED, HIGH);
        delay(d);
        digitalWrite(brocheLED, LOW);
        delay(d);
    }
}
```

Dans la définition de fonction ci-dessus, nous avons déclaré entre parenthèses les types des deux valeurs acceptées comme paramètres. Ces deux paramètres sont de type entier int. Cela correspond à la déclaration de deux nouvelles variables, mais ce sont des variables locales (nbrEclairs et d) qui ne sont utilisables que dans le corps de la fonction clignoter().

III. Variables globales et locales

Nous avons dit que les paramètres d'une fonction ne pouvaient être utilisés qu'à l'intérieur de cette fonction. L'extrait suivant provoquerait une erreur :

```
void informer(int monX)
{
    clignoter(monX, 10);
}
monX = 15;
```

En revanche, cet exemple-ci serait accepté par le compilateur :

```
int x;
void informer(int x)
{
    clignoter(x, 10);
}
x = 15;
```

Ce dernier exemple ne provoque pas de compilation, mais il peut constituer un piège.
En effet, vous utilisez ici deux variables portant le même nom x et pouvant donc avoir des valeurs différentes. Celle qui est déclarée sur la première ligne est une variable globale, ce qui signifie qu'elle peut être utilisée de n'importe où dans le programme, y compris dans les fonctions.

Dans une fonction, vous pouvez définir non seulement des paramètres, mais également des variables qui ne pourront être utilisées que dans la fonction. Ces variables sont locales :

```
void informer(int x)
{
    int nombreEclairs = x * 2;
    clignoter(nombreEclairs, 10);
}
```

La variable locale nommée nombreEclairs n'occupera de l'espace en mémoire que pendant l'exécution de la fonction. Dès que la fonction se termine, la variable disparaît. Voilà pourquoi les variables locales ne sont pas utilisables de l'extérieur de la fonction.

IV. Valeur renvoyée

Nous avons déjà écrit des fonctions qui utilisent une valeur d'entrée (un argument) mais aucune n'a encore renvoyé une valeur (son résultat). Autrement dit, nos fonctions ont toutes été jusqu'ici de type "void" (vide). Lorsqu'une fonction doit renvoyer une valeur, il faut déclarer le type de données de cette valeur.

Voyons comment écrire une fonction qui reçoit en entrée une température exprimée en degrés Celsius, puis renvoie la température après conversion en degrés Fahrenheit :

```
int convertirCelsFahr(int c)
{
    int f = c * 9 / 5 + 32;
    return f;
}
```

La définition de la fonction commence par le mot-clé int au lieu du mot void, ce qui indique que la fonction va renvoyer une valeur de type int dans la ligne où a été appelée cette fonction. Voici un exemple d'appel réel à la fonction :

```
int beauTemps = convertirCelsFahr(20);
```

Ce qui suit le mot-clé return peut être non seulement le nom d'une variable, mais aussi une expression. Cela permet d'abréger l'exemple précédent de la façon suivante :

```
int convertirCelsFahr(int c)
{
    return (f = c * 9 / 5 + 32);
}
```

V. Types de données des variables

Toutes les variables utilisées jusqu'à présent dans nos exemples étaient du type numérique entier int qui est le type de variable le plus utilisé. Il existe néanmoins d'autres types de données que vous devez connaître.

V.1. Numérique flottant : float

Un type de données qui va nous être indispensable dans l'exemple de conversion de température précédent est le type numérique à virgule flottante, qui correspond au mot-clé float. Revoyons la formule de conversion : Si nous donnons à la variable c la valeur 17, alors f vaudra 17 * 9 / 5 + 32, soit 62.6.

Si la variable f est de type int, cette valeur sera tronquée à 62 .

Dans de telles circonstances, nous utilisons le type float:

```
float convertirCelsFahr(float c)
{
    float f = c * 9.0 / 5.0 + 32.0;
    return f;
}
```

Remarquez que nous avons ajouté la mention .0 à la fin de toutes les valeurs littérales constantes. Cela permet d'informer le compilateur qu'il s'agit de valeurs de type float et non int.

V.2. Le type booléen (boolean)

Les valeurs booléennes sont des valeurs binaires, logiques. Un booléen ne peut être que vrai ou faux (1 ou 0).

La condition de l'instruction if telle que (compteur == 20) est une expression qui renvoie un résultat de type boolean. Voici un exemple de définition et d'utilisation d'une variable booléenne :

```
boolean tooBig = (x > 10);
if (tooBig)
{
x = 5;
}
```

Vous pouvez modifier des valeurs booléennes grâce aux opérateurs booléens. Les deux plus répandus de ces opérateurs étant and (&&) et or (||).

Un autre opérateur est l'inverseur not qui s'écrit !. Sans surprise, cet opérateur renvoie le résultat non vrai (faux) si l'entrée est vraie, et inversement. Voici un exemple de combinaison de ces opérateurs booléens dans la condition d'une instruction if satisfaite si x vaut entre 11 et 49 :

```
if ((x > 10) && (x < 50))
```

V.3. Autres types de données

Les deux types int et float seront suffisants pour la plupart des besoins, mais d'autres types numériques pourront s'avérer précieux de temps à autre (Tableau).

Vous devez tenir compte des risques de débordement et de rebouclage dans les données numériques. Si vous stockez dans une variable de type byte la valeur maximale 255 puis demandez d'y ajouter 1, la valeur passe à 0. Avec une variable int qui contient 32 767, le fait d'ajouter 1 donne –32 768.

VI. Commentaires

Les commentaires sont tous les textes dans le code source qui n'ont aucun effet fonctionnel, mais uniquement un rôle descriptif. Le compilateur ignore totalement tout le texte considéré comme commentaire. Les commentaires peuvent être définis de deux manières :

- Sur une seule ligne, le commentaire commence par // et se termine à la fin de la ligne.
- Le commentaire multiligne commence par /* et se termine par */.

L'exemple suivant utilise les deux formats de commentaires :

```
/* Une fonction loop() pas très utile.
Pour illustrer le concept de commentaires
*/
void loop() {
static int compteur = 0;
compteur ++;  // Un commentaire monoligne
if (compteur == 20) {
```

```
compteur = 0;
}
```

Type	*Mémoire (octets)*	*Plage de valeur*	*Notes*
boolean	1	true ou false (0 ou 1)	Type booléen.
char	1	–128 à +128	Sert à stocker le code numérique d'un caractère au format ASCII. Par exemple, le A correspond à la valeur 65. Normalement, les valeurs négatives ne sont pas utilisées.
byte	1	0 à 255	Souvent utilisé pour transférer des données par une liaison série, octet par octet. Voir le Chapitre 9.
int	2	–32 768 à +32 767	Type entier signé.
unsigned int	2	0 à 65 536	Permet d'augmenter la plage de valeurs lorsque les valeurs négatives ne sont pas utiles. À utiliser avec précaution, car la combinaison d'entiers signés et non signés peut donner des résultats inattendus.
long	4	-2 147 483 648 à 2 147 483 647	Seulement utile pour les très grandes valeurs.
unsigned long	4	0 à 4 294 967 295	Même principe que **unsigned int**.
float	4	–3.4028235E+38 à + 3.4028235E+38	Nombre à virgule flottante.
double	4	Comme **float**	Sur les processeurs des PC de bureau, ce type occupe 8 octets, ce qui offre une précision double par rapport à **float**. Sur le circuit Arduino, le type équivaut à **float**.