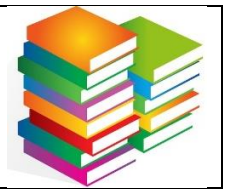
Les fonctions essentielles Arduino



Comment un programme Arduino peut-il lire les informations provenant de boutons ou de capteurs, puis commander les sorties d'un système électronique ?

Introduction

Le langage C++ permet d'organiser la logique du programme avec des variables, des conditions, des boucles et des fonctions.

L'environnement Arduino ajoute des fonctions spécifiques permettant au programme de communiquer avec les éléments matériels raccordés à la carte :

- ⇨ lire l'état d'un bouton
- ⇨ mesurer une tension
- ⇨ allumer une LED
- ⇨ produire un signal PWM
- ⇨ gérer le temps
- ⇨ échanger des informations avec un ordinateur.

La référence du langage Arduino distingue notamment les entrées-sorties numériques, les entrées-sorties analogiques, les fonctions temporelles et les fonctions de communication

Structure générale d'un programme Arduino

Un programme Arduino comporte obligatoirement deux fonctions principales :

```
void setup()
{
    // Initialisation du système
}

void loop()
{
    // Fonctionnement répété du système
}
```

La fonction `setup()`

La fonction `setup()` est exécutée une seule fois :

- ⇨ Après la mise sous tension
- ⇨ Après un appui sur le bouton Reset
- ⇨ Après le téléversement d'un programme

Elle permet notamment de :

- ⇨ Configurer les broches
- ⇨ Initialiser la communication série
- ⇨ Placer les sorties dans leur état initial

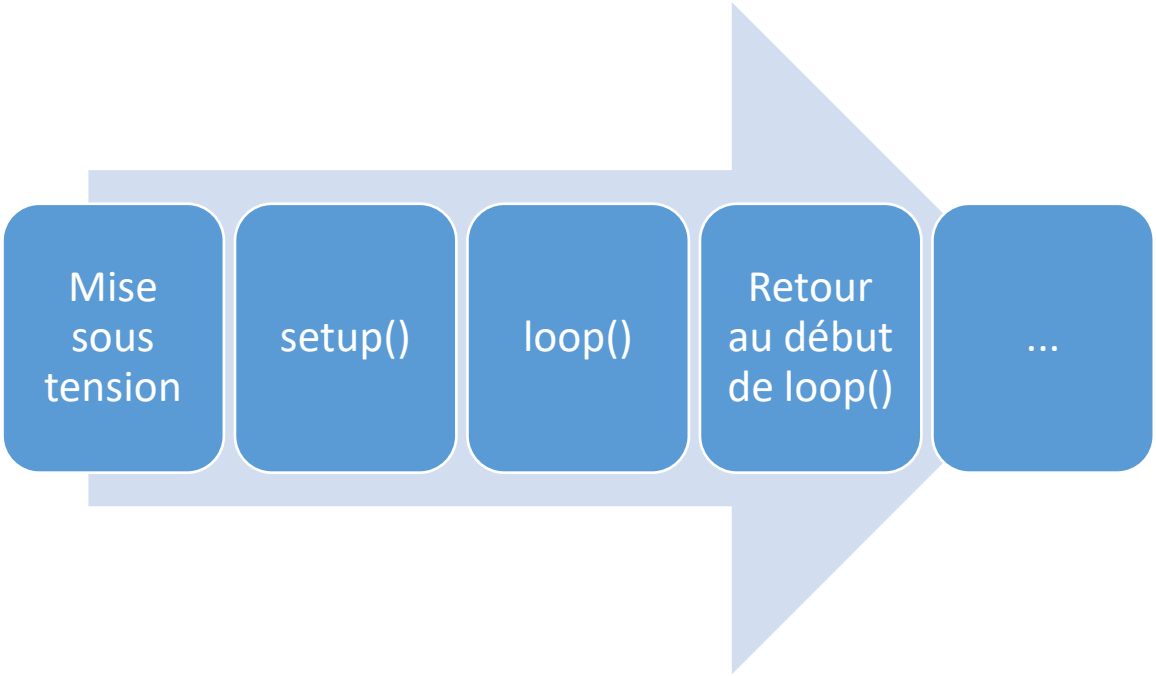
```
void setup()
{
    pinMode(8, OUTPUT);
    Serial.begin(9600);
}
```

La fonction `loop()`

Après l'exécution de `setup()`, la carte exécute continuellement la fonction `loop()`.

```
void loop()
{
    // Lecture des entrées
    // Traitement des informations
    // Commande des sorties
}
```

Le fonctionnement général est donc :



Configurer une broche avec `pinMode()`

Avant d'utiliser une broche numérique, il faut généralement préciser son rôle

```
pinMode(broche,mode);
```

Trois modes sont principalement utilisés :

Mode	Utilisation
`OUTPUT`	La broche est en sortie et commande un élément
`INPUT`	La broche est en entrée et reçoit une signal numérique
`INPUT_PULLUP`	La broche est en entrée et utilise une résistance interne de rappel forçant un niveau si elle est dans le « vide »

Configurer une sortie

```
const byte LED_VERTE = 8;

void setup()
{
    pinMode(LED_VERTE, OUTPUT);
}
```

La broche 8 pourra ensuite commander une LED, un relais ou l'entrée d'un autre circuit.

Configurer une entrée

```
const byte BP_MARCHE = 2;

void setup()
{
    pinMode(BP_MARCHE, INPUT);
}
```

Une entrée ne doit jamais rester électriquement « en l'air », car son état risquerait de devenir instable.

Utiliser la résistance interne de rappel

```
pinMode(BP_MARCHE, INPUT_PULLUP);
```

Avec `INPUT_PULLUP`, le bouton est généralement connecté entre la broche et la masse (GND).

La logique est alors inversée :

Etat du bouton	Valeur électrique lue
Bouton relâché	`HIGH`
Bouton appuyé	`LOW`

Commander une sortie avec `digitalWrite()`

La fonction `digitalWrite()` place une broche numérique dans l'un des deux états logiques possibles.

```
digitalWrite(broche, etat);
```

Les deux états sont :

Etat	Signification
`HIGH`	Niveau logique haut
`LOW`	Niveau logique bas

Allumer une LED

```
const byte LED_VERTE = 8;

void setup()
{
    pinMode(LED_VERTE, OUTPUT);
}

void loop()
{
    digitalWrite(LED_VERTE, HIGH);
}
```

Si la LED est câblée entre la broche est la masse, `HIGH` provoque son allumage.

Eteindre la LED

```
digitalWrite(LED_VERTE, LOW);
```

Exemple avec deux LED

```
digitalWrite(LED_VERTE, HIGH);
digitalWrite(LED_ROUGE, LOW);
```

La LED verte est allumée tandis que la LED rouge est éteinte.

Lire une entrée avec `digitalRead()`

La fonction `digitalRead()` lit l'état logique d'une broche numérique.

```
digitalRead(broche);
```

La fonction renvoie soit `HIGH` soit `LOW`.

Enregistrer l'état d'un bouton

```
bool etatBouton;

etatBouton = digitalRead(BP_MARCHE);
```

La valeur retournée est enregistrée dans la variable `etatBouton`.

Utilisation directe dans une condition

```
if (digitalRead(BP_MARCHE) == LOW)
{
    digitalWrite(LED_VERTE, HIGH);
}
else
{
    digitalWrite(LED_VERTE, LOW);
}
```

Avec un bouton configuré en `INPUT_PULLUP`, la condition est vraie lorsque le bouton est appuyé.

Programme complet

```
const byte BP_MARCHE = 2;
const byte LED_VERTE = 8;

void setup()
{
    pinMode(BP_MARCHE, INPUT_PULLUP);
    pinMode(LED_VERTE, OUTPUT);
}

void loop()
{
    if (digitalRead(BP_MARCHE) == LOW)
    {
        digitalWrite(LED_VERTE, HIGH);
    }
    else
    {
        digitalWrite(LED_VERTE, LOW);
    }
}
```

Lire une entrée analogique avec `analogRead()`

Une entrée numérique ne reconnaît que deux états : `HIGH` ou `LOW`.

Une entrée analogique permet de mesurer une tension variable, par exemple celle produite par :

- ⇨ Un potentiomètre
- ⇨ Une photorésistance
- ⇨ Une thermistance
- ⇨ Un capteur analogique

```
analogRead(broche);
```

Exemple

```
int mesure;

mesure = analogRead(A0);
```

Sur une Arduino Uno R3, la conversion analogique fournit une valeur entière comprise entre 0 et 1023.

Tension approximative	Valeur obtenue
0V	0
2.5V	Environ 512
5V	Environ 1023

Programme simple

```
const byte CAPTEUR = A0;

int mesure = 0;

void setup()
{
}

void loop()
{
    mesure = analogRead(CAPTEUR);
}
```

Produire un signal PWM avec `analogWrite()`

La fonction `analogWrite()` permet notamment de faire varier :

- ⇨ L'intensité lumineuse d'une LED
- ⇨ La vitesse d'un moteur à courant continu
- ⇨ La puissance moyenne fournie à une charge

```
analogWrite(broche, valeur);
```

Sur une Arduino Uno R3, la valeur est comprise entre 0 et 255.

Valeur	Comportement
`0`	Sortie toujours à l'état bas
`128`	Rapport cyclique proche de 50%
`255`	Sortie toujours à l'état haut

La fonction ne produit pas une tension analogique constante : elle génère un signal PWM, c'est-à-dire une succession rapide d'états `HIGH` et `LOW`.

Faire varier la luminosité d'une LED

```
const byte LED = 9;

void setup()
{
    pinMode(LED, OUTPUT);
}

void loop()
{
    analogWrite(LED, 50);
}
```

Quelques valeurs

```
analogWrite(LED, 0);      // LED éteinte
analogWrite(LED, 64);     // Faible luminosité
analogWrite(LED, 128);    // Luminosité moyenne
analogWrite(LED, 255);    // Luminosité maximale
```

Créer une temporisation avec `delay()`

La fonction `delay()` suspend l'exécution du programme pendant une durée exprimée en millisecondes.

```
delay(duree);
```

Correspondances

Valeur utilisée	Durée
`delay(1);`	1 ms
`delay(100);`	100 ms
`delay(1000);`	1 s
`delay(2000);`	2 s

Faire clignoter une LED

```
void loop()
{
    digitalWrite(LED_VERTE, HIGH);
    delay(1000);

    digitalWrite(LED_VERTE, LOW);
    delay(1000);
}
```

La LED reste allumée pendant une seconde, puis éteinte pendant une seconde.

Limite de `delay()`

Pendant l'exécution de `delay()`, le programme reste bloqué sur cette instruction.

```
delay(5000);
```

Pendant ces cinq secondes, le programme ne peut pas effectuer les instructions suivantes, comme lire un bouton ou actualiser un affichage.

Mesurer le temps avec `millis()`

La fonction `millis()` renvoie le nombre de millisecondes écoulées depuis le démarrage du programme.

```
millis();
```

La valeur est généralement enregistrée dans une variable de type `unsigned long`.

Exemple :

```
unsigned long tempsActuel;

tempsActuel = millis();
```

Réaliser une action sans bloquer le programme

```
const byte LED_VERTE = 8;

bool etatLed = false;

unsigned long ancienTemps = 0;
const unsigned long PERIODE = 1000;

void setup()
{
    pinMode(LED_VERTE, OUTPUT);
}

void loop()
{
    unsigned long tempsActuel = millis();

    if (tempsActuel - ancienTemps >= PERIODE)
    {
        ancienTemps = tempsActuel;

        etatLed = !etatLed;

        digitalWrite(LED_VERTE, etatLed);
    }
}
```

Contrairement à `delay()`, le programme peut continuer à lire des boutons ou à effectuer d'autres traitements entre deux changements d'état de la LED.

Communiquer avec l'ordinateur ou un autre module électronique en UART

La communication série permet d'échanger des informations entre la carte Arduino et l'ordinateur ou tout autre module possédant un bus UART (module Bluetooth, module GSM, module GPS, ...).

Elle est notamment utile pour :

- ⇨ Observer la valeur d'un capteur
- ⇨ Suivre l'évolution d'une variable
- ⇨ Vérifier une condition
- ⇨ Rechercher une erreur dans un programme

Initialiser la communication

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

La valuer 9600 représente le débit de transmission en bits par seconde (bps). Le moniteur série (putty, teraterm, Arduino IDE, ...) ou le module (GPS, GSM, BT, ...) doit être configuré avec le même débit.

Envoyer une information avec `Serial.print()`

```
Serial.print("Temperature : ");
```

La prochaine information est affichée sur la même ligne.

Envoyer une information avec un retour à la ligne

```
Serial.println("Système actif");
```

La prochaine information sera affichée sur une nouvelle ligne.

Afficher une variable

```
int mesure = analogRead(A0);

Serial.print("Mesure : ");
Serial.println(mesure);
```

Résultat possible :

```
Mesure : 487
```

Programme complet :

```
const byte CAPTEUR = A0;

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

void loop()
{
    int mesure = analogRead(CAPTEUR);

    Serial.print("Valeur analogique : ");
    Serial.println(mesure);

    delay(500);
}
```

Erreurs fréquentes

Oublier de configurer une broche

```
digitalWrite(LED_VERTE,HIGH);
```

Il manque généralement :

```
pinMode(LED_VERTE,OUTPUT);
```

Oublier la logique inversée de INPUT_PULLUP

Incorrect :

```
if (digitalRead(BP_MARCHE) == HIGH)
{
    // Le bouton est appuyé
}
```

Avec un bouton relié à la masse, la bonne condition est :

```
if (digitalRead(BP_MARCHE) == LOW)
{
    // Le bouton est appuyé
}
```

Utiliser analogWrite() sur une broche non PWM

```
analogWrite(8, 128);
```

Sur une Uno R3, la broche 8 ne fait pas partie des sorties PWM.

Confondre entrée analogique et sortie PWM

```
analogRead(A0);
```

Sert à effectuer une mesure.

```
analogWrite(9, 128);
```

Sert à produire un signal PWM.

Synthèse

Fonction	Utilisation
`setup()`	Initialiser le système
`loop()`	Répéter le fonctionnement principal
`pinMode()`	Configurer une broche
`digitalWrite()`	Commander uen sortie numérique
`digitalRead()`	Lire une entrée numérique
`analogRead()`	Lire une entrée analogique
`analogWrite()`	Produire un signal PWM
`delay()`	Créer une temporisation bloquante
`millis()`	Mesurer le temps sans bloquer
`Serial.begin()`	Initialiser la communication série
`Serial.print()`	Afficher sans retour à la ligne
`Serial.println()`	Afficher avec retour à la ligne