

Arduino (C++) : Notions fondamentales

K. Boudjelaba



Table des matières

Généralités

Eléments du langage Arduino

Opérations de base

Les variables

Exemple dans un sketch

Exemples d'application

Conversion Analogique Numérique

Conditions et structures alternatives

 Elaboration

Les boucles

 La boucle FOR

 La boucle while

 Les boucles infinies

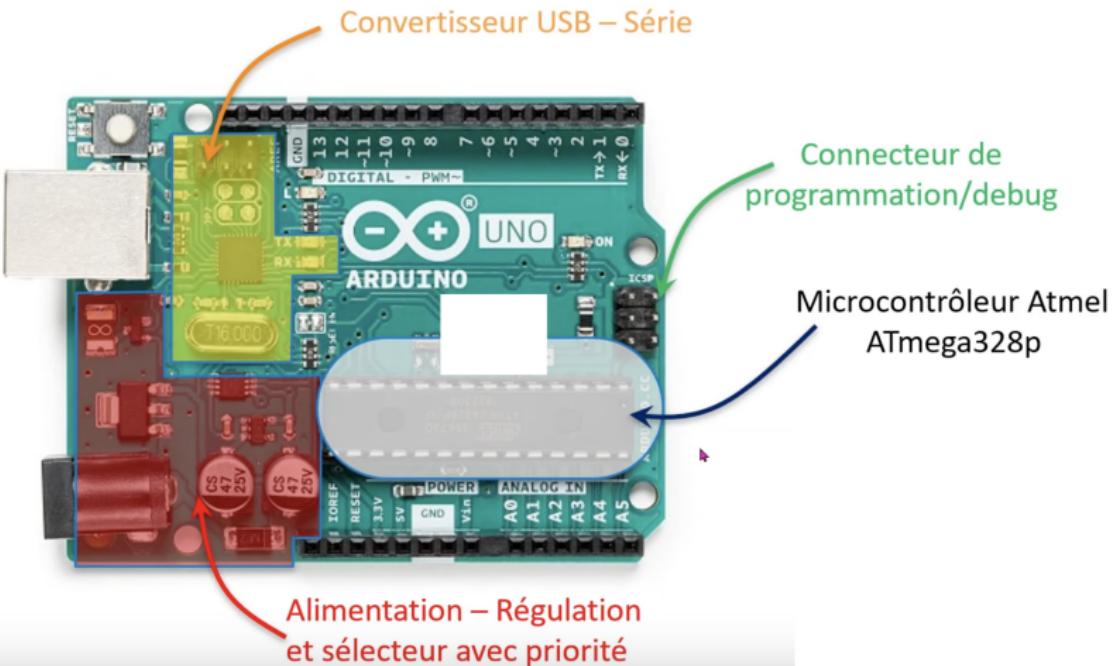
Les fonctions

 Fonction MAP

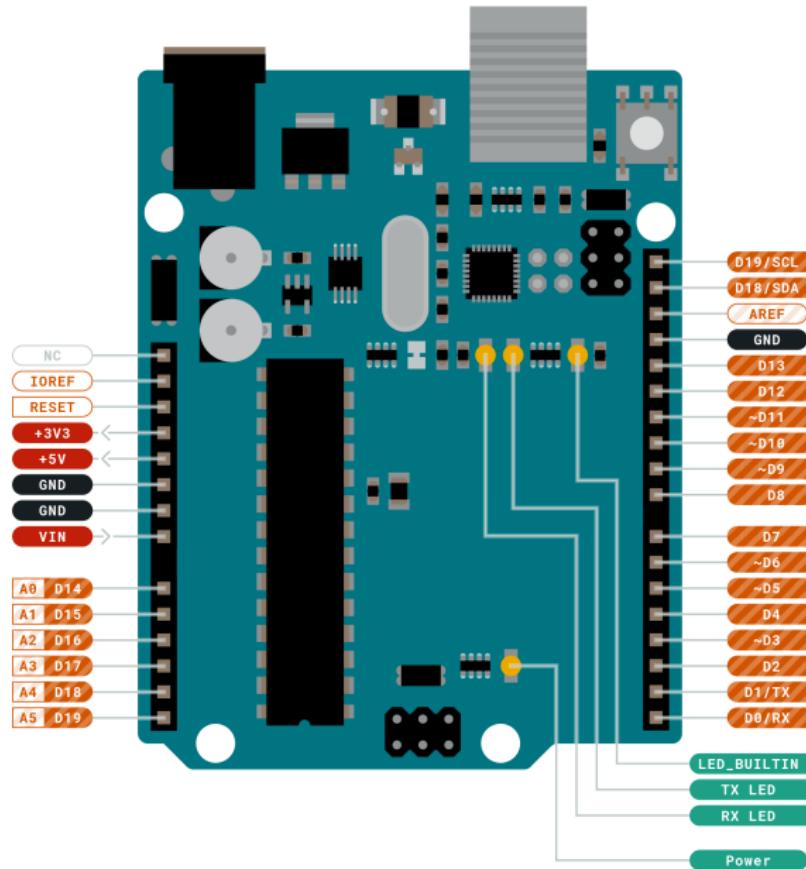
Exercices

Généralités

USB



Généralités



Eléments du langage Arduino

Dans le jargon **Arduino**, un fichier **Arduino** est appelé "Sketch" en anglais et "Croquis" en français.

Structure d'un sketch

```
1 void setup() {  
2     // put your setup  
3     // code here,  
4     // to run once:  
5 }  
6  
7  
8 void loop() {  
9     // put your main code  
10    // here, to run  
11    // repeatedly:  
12 }
```

La fonction `setup()` est appelée une seule fois au démarrage du programme **Arduino** (sketch). Elle contiendra le code nécessaire à l'initialisation de l'**Arduino**.

La fonction `loop()` est appelée en continu. Le code qu'elle contient est exécuté en boucle infinie, tant que l'**Arduino** est sous tension.

Eléments du langage Arduino

Liste (non-exhaustive) des commandes

<code>setup()</code>	<code>loop()</code>	
<code>Serial.begin()</code>	<code>Serial.print()</code>	<code>Serial.println()</code>
<code>Serial.write()</code>	<code>#define</code>	<code>#include</code>
<code>delay()</code>	<code>delayMicroseconds()</code>	<code>millis()</code>
<code>micros()</code>	<code>pulseIn()</code>	<code>random()</code>
<code>randomSeed()</code>	<code>size_t</code>	<code>sizeof()</code>
<code>pinMode()</code>	<code>digitalWrite()</code>	<code>digitalRead()</code>
<code>analogRead()</code>	<code>analogWrite()</code>	
<code>Serial.available()</code>	<code>Serial.read()</code>	

Serial.begin()

Définit le débit de données en bits par seconde (baud) pour la transmission de données en série. Pour communiquer avec le moniteur série (Serial Monitor), assurez-vous d'utiliser l'un des débits en bauds répertoriés dans le menu dans le coin inférieur droit de son écran.

```
1 Serial.begin(speed)
```

speed : en bits par seconde (baud).

```
1 void setup() {
2     Serial.begin(9600); // ouvre le port série, définit le
3                             // débit de données à 9600 bps
4 }
5 void loop() {}
```

L'instruction : `Serial.begin(9600);` démarre une communication série à la vitesse de 9600 bauds. Il faut donc que le moniteur série soit configuré sur ce débit, sinon le moniteur affiche des caractères bizarres.

Eléments du langage Arduino

Serial.print()

Imprime (affiche) les données sur le port série sous forme de texte ASCII lisible par l'homme. Cette commande peut prendre plusieurs formes. Les nombres sont imprimés en utilisant un caractère ASCII pour chaque chiffre. Les flottants sont imprimés de la même manière sous forme de chiffres ASCII, avec deux décimales par défaut. Les octets sont envoyés sous la forme d'un seul caractère. Les caractères et les chaînes sont envoyés tels quels.

```
1 Serial.print(val)
2 Serial.print(val, format)
```

- ▶ val : la valeur à afficher. Types de données autorisés : tout type de données.
- ▶ format : spécifie la base numérique (pour les entiers) ou le nombre de décimales (pour les float).

Serial.println()

Imprime (affiche) les données sur le port série sous forme de texte ASCII lisible suivi d'un caractère de retour chariot (ASCII 13 ou '\r') et d'un caractère de saut de ligne (ASCII 10 ou '\n'). Cette commande prend les mêmes formes que `Serial.print()`.

```
1 Serial.println(val)
2 Serial.println(val, format)
```

- ▶ val : la valeur à afficher. Types de données autorisés : tout type de données.
- ▶ format : spécifie la base numérique (pour les entiers) ou le nombre de décimales (pour les réels (float)).

Eléments du langage Arduino

delay()

Suspend le programme pendant la durée (en millisecondes) spécifiée en paramètre.

```
1 delay(ms)
```

ms : le nombre de millisecondes de pause.

delayMicroseconds()

Met le programme en pause pendant la durée (en microsecondes) spécifiée par le paramètre.

```
1 delayMicroseconds(us)
```

us : le nombre de microsecondes de pause.

Actuellement, la valeur la plus élevée qui produira un délai précis est 16383. Pour les délais supérieurs à quelques milliers de microsecondes, vous devez utiliser **delay()** à la place.

Eléments du langage Arduino

millis()

Renvoie le nombre de millisecondes (ms) écoulées depuis que la carte Arduino a commencé à exécuter le programme en cours. Ce nombre va déborder (revenir à zéro), après environ 50 jours.

```
1 time = millis()
```

```
1 unsigned long time;
2 void setup() {
3     Serial.begin(9600);
4 }
5 void loop() {
6     Serial.print("Temps : ");
7     time = millis();
8     Serial.println(time); // imprime le temps depuis le
                           // démarrage du programme
9
10    delay(1000);        // attendre une seconde pour ne
                           // pas envoyer des quantités
                           // massives de données
11
12
13 }
```

Eléments du langage Arduino

micros()

Renvoie le nombre de microsecondes depuis que la carte [Arduino](#) a commencé à exécuter le programme en cours. Ce nombre va déborder (revenir à zéro), après environ 70 minutes. Sur les cartes [Arduino](#) 16 MHz, cette fonction a une résolution de $4 \mu s$ (c'est-à-dire que la valeur renvoyée est toujours un multiple de 4). Sur les cartes [Arduino](#) 8 MHz, cette fonction a une résolution de $8 \mu s$.

```
1 time = micros()
```

Eléments du langage Arduino

pinMode()

Configure la broche spécifiée pour qu'elle se comporte soit comme une entrée, soit comme une sortie.

Syntaxe : `pinMode(pin, mode);`

- ▶ **pin** : le numéro de broche **Arduino** pour définir le mode
- ▶ **mode** : **INPUT**, **OUTPUT**.

Exemple : Configuration de la broche 13 en sortie

BTS SN

```
1 void setup() {  
2     pinMode(13, OUTPUT);      // configure la pin numérique 13  
3                                         // comme une sortie  
4 }  
5  
6 void loop() {  
7     digitalWrite(13, HIGH);   // mettre pin 13 à l'état "on"  
8     delay(1000);            // attendre 1 seconde  
9     digitalWrite(13, LOW);   // mettre pin 13 à l'état "off"  
10    delay(1000);           // attendre 1 seconde  
11 }
```

Eléments du langage Arduino

digitalWrite()

Ecrire une valeur **HIGH** ou **LOW** sur une broche numérique.

Si la broche a été configurée en **OUTPUT** avec **pinMode()**, sa tension sera réglée sur la valeur correspondante : 5V pour **HIGH** et 0V (GND) pour **LOW**.

Si on ne définit pas **pinMode()** sur **OUTPUT** et qu'on connecte une LED à une broche, lors de l'appel de **digitalWrite(HIGH)**, l'éclairage de la LED peut être faible. Sans définir explicitement **pinMode()**, **digitalWrite()** aura activé la résistance de rappel interne (pullup), qui agit comme une grande résistance de limitation de courant.

Syntaxe : **digitalWrite(pin, value);**

- ▶ **pin** : le numéro de broche **Arduino**
- ▶ **value** : **HIGH** ou **LOW**. Voir l'exemple précédent.

digitalRead()

Lit la valeur d'une broche numérique spécifiée, HIGH ou LOW.

Syntaxe : `digitalRead(pin);`

- ▶ `pin` : le numéro de broche Arduino

Retour : HIGH ou LOW.

Exemple : Définir la broche 13 à la même valeur que la broche 7

```
1 int ledPin = 13; // LED connectée à la pin 13
2 int inPin = 7; // Bouton poussoir connecté à la pin 7
3 int val = 0; // Variable pour sauvegarder la valeur lue
4 void setup() {
5     pinMode(ledPin, OUTPUT); // configure la pin 13 en sortie
6     pinMode(inPin, INPUT); // configure la pin 7 en entrée
7 }
8 void loop() {
9     val = digitalRead(inPin); // Lire la pin configurée en
10                                // entrée (pin 7)
11     digitalWrite(ledPin, val); // Régler l'état de la LED
12                                // en fonction du l'état
13                                // du bouton
14 }
```

Eléments du langage Arduino

analogRead()

Lit la valeur de la broche analogique spécifiée.

Les cartes **Arduino** contiennent un convertisseur analogique-numérique 10 bits. Cela signifie qu'il "mappera" les tensions d'entrée entre 0 et la tension de fonctionnement (5 V ou 3,3 V) en valeurs entières entre 0 et 1023. Sur un **Arduino UNO**, par exemple, cela donne une résolution entre les lectures de : 5 volts / 1024 unités ou , 0,0049 volt (4,9 mV) par unité.

Sur les cartes basées sur ATmega (UNO, Nano, Mini, Mega), il faut environ 100 microsecondes (0,0001 s) pour lire une entrée analogique, donc le taux de lecture maximum est d'environ 10 000 fois par seconde.

Syntaxe : `analogRead(pin);`

- ▶ **pin** : le nom de l'entrée analogique à lire (A0 à A5).

Retour : La valeur analogique lue sur la pin. Type de données : **int** entre 0 et 1023.

Eléments du langage Arduino

analogRead()

Exemple : Lecture et affichage de la tension d'une broche analogique

```
1 int analogPin = A3; /* potentiomètre : la broche centrale
2                      connectée à la pin 3 (analogique)
3                      les deux broches externes
4                      reliées à GND et +5V */
5 int val = 0; // variable pour sauvegarder la valeur lue
6
7 void setup() {
8     Serial.begin(9600); // Réglage de la vitesse "serial"
9 }
10
11 void loop() {
12     val = analogRead(analogPin); // Lire la pin d'entrée
13     Serial.println(val); // Affichage de la valeur lue
14 }
```

analogWrite()

Écrit une valeur analogique (signal PWM) sur une broche.

Peut être utilisé pour allumer une LED à différentes luminosités.

Après un appel à `analogWrite()`, la broche générera un signal rectangulaire du rapport cyclique spécifié jusqu'au prochain appel à `analogWrite()` (ou un appel à `digitalRead()` ou `digitalWrite()`) sur la même broche.

Syntaxe : `analogWrite(pin, value);`

- ▶ `pin` : La broche `Arduino` sur laquelle écrire.
- ▶ `value` : Le rapport cyclique : entre 0 (éteint) et 255 (allumé).

analogWrite()

Tension de la LED proportionnelle à la valeur lue sur le potentiomètre

```
1 int ledPin = 9;          // LED connectée à la pin 9
2 int analogPin = 3;        // potentiometer connecté à la
                           // pin analogique 3
3
4 int val = 0;              // variable pour stocker la valeur lue
5
6 void setup() {
7   pinMode(ledPin, OUTPUT);
8 }
9
10 void loop() {
11   val = analogRead(analogPin); // Lecture de la pin
                                // d'entrée
12
13   analogWrite(ledPin, val/4); // analogRead (valeurs
                                // comprises entre 0 et 1023)
14
                                // analogWrite (valeurs
15                                // comprises entre 0 et 255)
16
17 }
```

Opérations de base

Opérateur d'affectation : =

Exemples

```
1 unsigned int a = 0b11001100;
2 unsigned int b = 0xcc;
3 char ch = 'a';
4 float x = 0.1;
```

Usuels : +, -, /, *

Modulo : %

% = reste de la division entière.

Exp : $7 \% 5 = 2$

Exemples

```
1 unsigned int a, b = 6;
2 float y, x = 0.1;
3
4 a = 3*b + b%2;
5 y = 2.2*x-5;
```

Opérations de base

Incrémantation et décrémentation : ++ et --

$i++ ; \iff i = i + 1;$

Affectation avec calcul :

$+ =, - =, / =, * =$

$i += 3; \iff i = i + 3;$

Opérateurs logiques bit à bit :

- ▶ & = ET bit à bit
- ▶ | = OU bit à bit
- ▶ ^ = OU exclusif bit à bit

Exemples

```
1 unsigned int a = 0b11001100;
2 unsigned int b = 0b01001001;
3 unsigned int ou, et, ouex;
4
5 ou = a|b;
6 // donne ou vaut 0b11001101
7 et = a&b;
8 // donne et vaut 0b01001000
9 ouex = a^b;
10 // donne ouex vaut 0b10000101
11
12 Serial.print("ou = ");
13 Serial.println(ou, BIN);
```

Les variables

Variable = information typée stockée dans la SRAM (Static Random Access Memory) et utilisable pour faire des calculs ...
SRAM : 2kOctets – 256 pour tous les registres de périphérique

Les entiers

- ▶ Données codées sur 8 bits :
 - ▶ `char` : valeurs de -128 à +127
 - ▶ `unsigned char` : valeurs de 0 à 255
- ▶ Données codées sur 16 bits :
 - ▶ `int` : valeurs de -32768 à +32767
 - ▶ `unsigned int` : valeurs de 0 à 65535
- ▶ Données sur 32 bits :
 - ▶ `long` : valeurs de -2^{31} à $2^{31} - 1$
 - ▶ `unsigned long` : valeurs de 0 à $2^{32} - 1$

Les réels

- ▶ simple précision (32 bits) : `float` de $\pm 3.4 \times 10^{-38}$ à $\pm 3.4 \times 10^{38}$

Les variables

Déclaration de variables

Forme générale ([…] pour ce qui est facultatif) :

type nom[=valeur];

type nom1[=valeur1], nom2[=valeur2], ...;

Exemples :

```
1 int i = 0; j = 2;
2
3 unsigned int indice;
4
5 float a;
6 float b = 1e5, c = 0.01;
7
8 char ch1 = 'a';
9 char ch2;
```

Portée des variables

- ▶ Globales :
 - ▶ Déclarées en dehors de toutes fonctions : `setup()` , `loop()` , fonctions de l'utilisateur
 - ▶ Visibles dans toutes les fonctions
- ▶ Locales :
 - ▶ Déclarées dans une fonction
 - ▶ Visibles uniquement dans cette fonction

Les variables - Exemple dans un sketch

Les variables - Exemple dans un sketch

Les variables - Exemples d'application

Exemple 1

Exemple 1 : Arduino :

BTS SN

```
1 long duree = 3;
2
3 void setup() {
4     Serial.begin(115200);
5     int i; Serial.print("setup.i = "); Serial.print(i);
6     Serial.print("\t"); delay(1000);
7 }
8
9 void loop() {
10    int i = 25; Serial.print("loop.i = "); Serial.print(i);
11    Serial.print("\t");
12    float x = 0.1; Serial.print("x = "); Serial.print(x);
13    Serial.print("\t");
14
15    i = i * 2; Serial.print("loop.i2 = "); Serial.print(i);
16    Serial.print("\t");
17    duree = 3*duree; Serial.print("duree = ");
18    Serial.println(duree); delay(3000);
19    Serial.println("===== Fin boucle =====");
20 }
```

Les variables - Exemples d'application

Exemple 2

Exemple 2 : Arduino :

BTS SN

```

1 long duree = 3;
2
3 void setup() {
4     Serial.begin(115200);
5     int i; Serial.print("setup.i = "); Serial.print(i);
6     Serial.print("\t"); delay(1000);
7 }
8
9 void loop() {
10    static int i = 25; Serial.print("loop.i = ");
11    Serial.print(i); Serial.print("\t");
12    float x = 0.1; Serial.print("x = "); Serial.print(x);
13    Serial.print("\t");
14
15    i = i * 2; Serial.print("loop.i2 = "); Serial.print(i);
16    Serial.print("\t");
17    duree = 3*duree; Serial.print("duree = ");
18    Serial.println(duree); delay(3000);
19    Serial.println("===== Fin boucle =====");
20 }
```

Les variables - Exemples d'application

Exemple 3

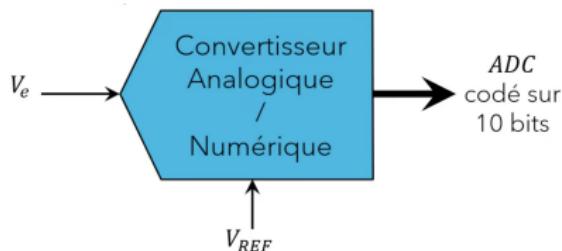
Exemple 3 : Arduino :

BTS SN

```
1 void setup() {
2     Serial.begin(115200);
3     Serial.println("Liaison série initialisée.");
4 }
5
6 void loop() {
7     static unsigned temps = 0;
8     delay(1000);
9     Serial.print("Temps = ");
10    Serial.println(temps);
11    temps = temps+1;
12 }
```

Conversion Analogique Numérique

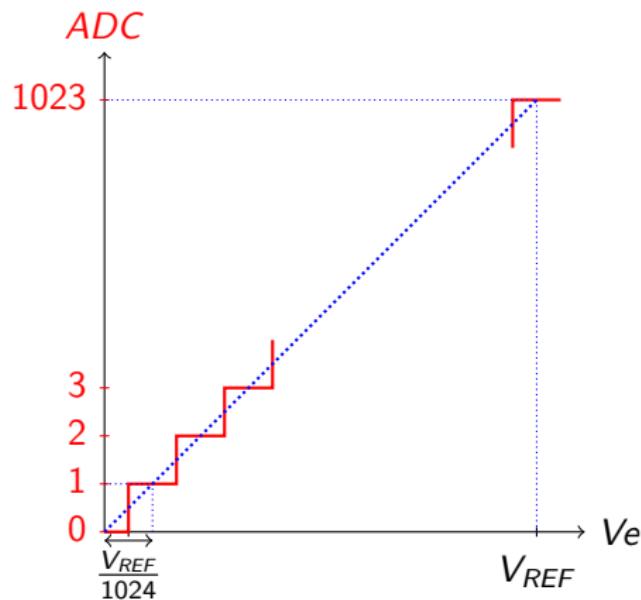
Transformation d'une grandeur physique en grandeur numérique



Lien entre *ADC* et V_e

$$ADC = \frac{1024}{V_{REF}} \cdot V_e$$

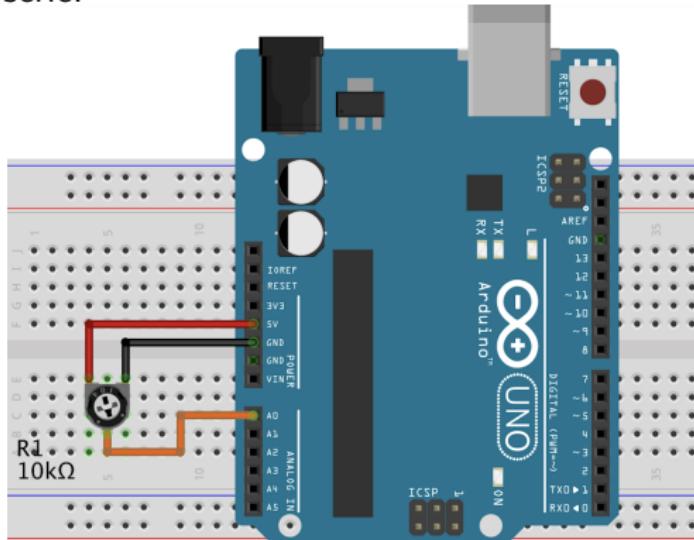
$$V_e = \frac{V_{REF}}{1024} \cdot ADC$$



Conversion Analogique Numérique

Application

- ▶ Réaliser le montage ci-dessous,
- ▶ Ecrire un programme **Arduino** pour lire la tension (numérique) de la pin A0 et l'afficher dans le moniteur série,
- ▶ Convertir cette tension en tension analogique et l'afficher dans le moniteur série.



Application

Code Arduino :

BTS SN

```
1 void setup() {
2     Serial.begin(9600);
3     delay(100);
4 }
5
6 void loop() {
7     static uint16_t Ve_N;
8
9     Ve_N = analogRead(A0);
10    Serial.print("Ve_N = ");
11    Serial.print(Ve_N);
12    Serial.print(" Ve_A = ");
13    Serial.print(Ve_N*5.0/1023.0);
14
15    delay(100);
16 }
```

Conditions et structures alternatives

Elaboration

Avec des opérateurs relationnels

- ▶ Egalité : ==
- ▶ Différent de : !=
- ▶ Supérieur : >
- ▶ Inférieur : <
- ▶ Supérieur ou égal : >=
- ▶ Inférieur ou égal : <=

Conditions et structures alternatives

Elaboration

Avec des opérateurs relationnels

- ▶ Egalité : ==
- ▶ Différent de : !=
- ▶ Supérieur : >
- ▶ Inférieur : <
- ▶ Supérieur ou égal : >=
- ▶ Inférieur ou égal : <=

et des opérateurs logiques

- ▶ ET logique : &&
- ▶ OU logique : ||
- ▶ NON logique : !

Conditions et structures alternatives

Elaboration

Avec des opérateurs relationnels

- ▶ Egalité : ==
- ▶ Différent de : !=
- ▶ Supérieur : >
- ▶ Inférieur : <
- ▶ Supérieur ou égal : >=
- ▶ Inférieur ou égal : <=

et des opérateurs logiques

- ▶ ET logique : &&
- ▶ OU logique : ||
- ▶ NON logique : !

Résultat : VRAI ou FAUX

- Pas de type booléen en C et C ++
 - ▶ Condition renvoie 1 si elle est VRAIE
 - ▶ Condition renvoie 0 si elle est FAUSSE
 - ▶ Si remplacement d'une condition par une expression qui n'est pas une condition
 - ▶ Valeur nulle considérée comme condition FAUSSE
 - ▶ Toute autre valeur considérée comme une condition VRAIE

Conditions et structures alternatives

Instruction if

L'instruction **if** vérifie une condition et exécute l'instruction ou l'ensemble d'instructions suivant si la condition est "vraie" ou "fausse".

Syntaxe

```
1 if (condition) {  
2   // action(s);  
3 }
```

Exemples

```
1 if (x > 10) digitalWrite(LEDpin, HIGH);  
2  
3 if (x > 10)  
4   digitalWrite(LEDpin, HIGH);  
5  
6 if (x > 10) {digitalWrite(LEDpin, HIGH);}  
7  
8 if (x > 10) {  
9   digitalWrite(LEDpin1, HIGH);  
10  digitalWrite(LEDpin2, HIGH);  
11 }
```

Conditions et structures alternatives

Opérateur conditionnel

Opérateur : ?

(condition) ? valeur si condition vraie : valeur si condition fausse

Exemple

```
1 Serial.println ((a==3)?"a est = à 3":"a est # de 3");
```

Conditions et structures alternatives

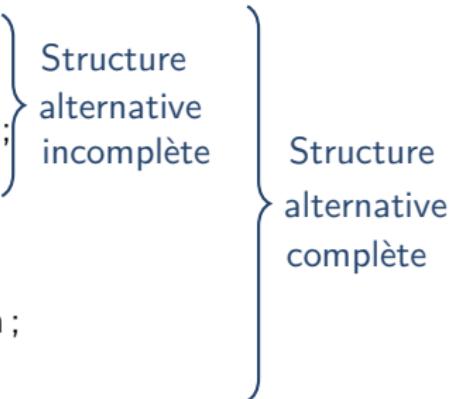
Structure alternative

Actions multiples

```
si condition alors
    actions_alors;
fin
sinon
    actions_sinon;
fin
```

Structure alternative incomplète

Structure alternative complète



Conditions et structures alternatives

Elaboration

Actions multiples (partie entre [] facultative)

```
if (condition) {  
    actions_alors;  
}  
[ else {  
    actions_sinon;  
} ]
```

Conditions et structures alternatives

Elaboration

Actions multiples (partie entre [] facultative)

```
if (condition) {  
    actions_alors;  
}  
[ else {  
    actions_sinon;  
} ]
```

Forme synthétique

```
if (condition) actions_alors; [ else actions_sinon; ]
```

Conditions

Syntaxe (début)

```
1 void setup() {
2     int a = 3, b = 4;
3     Serial.begin(115200);
4     Serial.println((a==3)?"a est égal à 3":"a est différent
5                     de 3");
6
7     Serial.println("Avant : a =" + String(a) + " et b =" +
8                   + String(b));
9     if (a == b)
10        Serial.println("La condition " + String(a) + "==" +
11                      + String(b) + " est vraie");
12    else
13        Serial.println("La condition " + String(a) + "==" +
14                      + String(b) + " est fausse");
15    Serial.println("Après : a =" + String(a) + " et b =" +
16                      + String(b));
17
18    int a_avant = a;
```

Conditions et structures alternatives

Conditions

Syntaxe (suite)

```
1 // int a_avant = a;
2 if (a == b)
3     Serial.println("La condition " + String(a_avant)
4         + "==" + String(b) + " est vraie");
5 else
6     Serial.println("La condition " + String(a_avant)
7         + "==" + String(b) + " est fausse");
8 Serial.println("Après : a =" + String(a) + " et b =" +
9     String(b));
10
11 if (1) Serial.println("1 est vu comme une condition
12     vraie");
13 else Serial.println("1 est vu comme une condition
14     fausse");
15
16 if (-1) Serial.println("-1 est vu comme une condition
17     vraie");
18 else Serial.println("-1 est vu comme une condition
19     fausse");
```

Conditions et structures alternatives

Conditions

Syntaxe (fin)

```
1  if (0) Serial.println("0 est vu comme une condition
2          vraie");
3  else Serial.println("0 est vu comme une condition
4          fausse");

5
6  if (10) Serial.println("10 est vu comme une condition
7          vraie");
8  else Serial.println("10 est vu comme une condition
9          fausse");
10 }
11 void loop() {
12 }
```

Seuil allumage 1

Syntaxe

```
1 #define PIN_LED 3
2
3 void setup() {
4     pinMode(PIN_LED, OUTPUT);
5     Serial.begin(9600);
6     delay(100);
7 }
8 void loop() {
9     const unsigned int SEUIL_ALLUMAGE = 512;
10    unsigned int mesure = analogRead(A0);
11    Serial.println(mesure);
12    if (mesure < SEUIL_ALLUMAGE)
13        digitalWrite(PIN_LED, HIGH);
14    else
15        digitalWrite(PIN_LED, LOW);
16 }
```

Conditions et structures alternatives

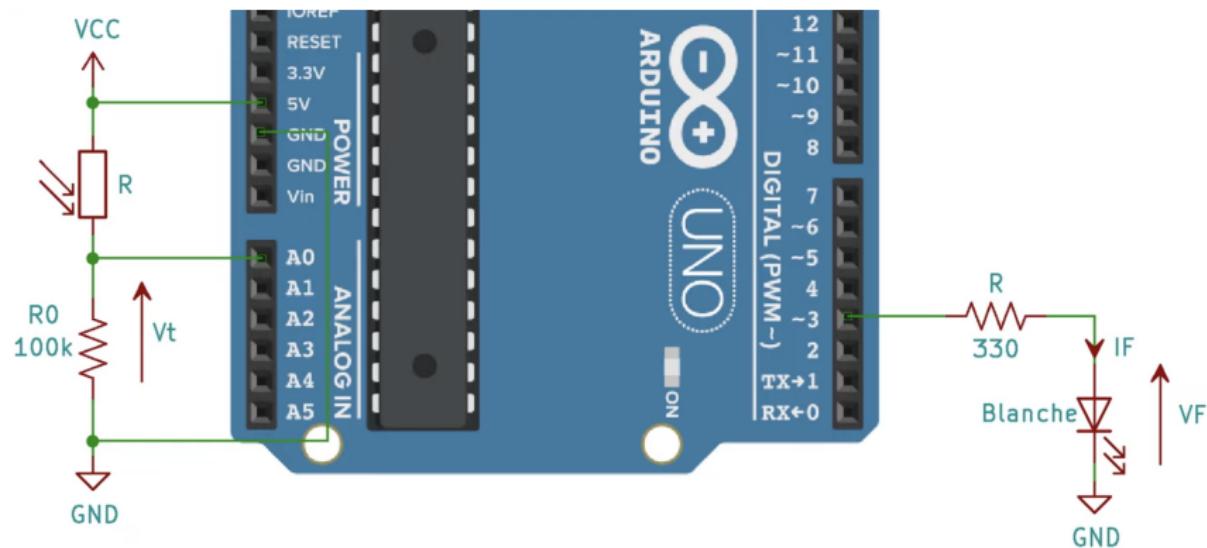
Seuil allumage 2

Syntaxe

```
1 #define PIN_LED 3
2
3 void setup() {
4     pinMode(PIN_LED, OUTPUT);
5     Serial.begin(9600);
6     delay(100);
7 }
8 void loop() {
9     const unsigned int SEUIL_ALLUMAGE = 400;
10    const unsigned int SEUIL_EXTINCTION = 800;
11
12    unsigned int mesure = analogRead(A0);
13    Serial.println(mesure);
14    if (mesure < SEUIL_ALLUMAGE)
15        digitalWrite(PIN_LED, HIGH);
16    if (mesure > Seuil_EXTINCTION)
17        digitalWrite(PIN_LED, LOW);
18 }
```

Conditions et structures alternatives

Seuil allumage 3



Conditions et structures alternatives

Structure au cas par cas

```
switch (expression) {  
    case expression_1 :  
        actions;  
        [ break; ]  
    case expression_2 :  
        actions;  
        [ break; ]  
    case expression_n :  
        actions;  
        [ break; ]  
    default :  
        actions; };
```

```
1 switch (couleur) {  
2     case VERT:  
3         Serial.println("Vert");  
4         break;  
5     case BLEU:  
6         Serial.println("Bleu");  
7         break;  
8     case ROUGE:  
9         Serial.println("Rouge");  
10        break;  
11    default:  
12        Serial.println("C. inconnue");  
13    }
```

Les boucles - La boucle FOR

La boucle **for** est utilisée pour répéter un bloc d'instructions entouré d'accolades (**{ }**). Un compteur d'incrémentation est utilisé pour incrémenter et terminer la boucle.

Syntaxe

```
1 for (initialisation; condition; incrément) {  
2     // action(s);  
3 }
```

Exemple

```
1 void setup() {  
2     // setup  
3 }  
4  
5 void loop() {  
6     for (int i = 0; i <= 10; i++) {  
7         Serial.println(i); delay(100);  
8     }  
9 }
```

Les boucles - La boucle while

Une boucle **while** tournera continuellement, et indéfiniment, jusqu'à ce que l'expression entre parenthèses () devienne fausse. On doit changer la variable de test (variable à incrémenter ou condition externe : test d'un capteur) sinon la boucle **while** ne se terminera jamais.

Syntaxe

```
1 while (condition) {  
2     // action(s);  
3 }
```

Exemple

```
1 int i = 0;  
2 while (i < 20) {  
3     // action(s) à répéter 20 fois  
4     i++;  
5 }
```

Boucle for : Voir page 5

```
1 // Boucle infinie avec for
2 for (;;) {
3     // code à exécuter à l'infinie
4 }
```

Boucle while

```
1 // Boucle infinie avec while
2 while(1) {
3     // code à exécuter à l'infinie
4 }
```

Boucle while

```
1 // Boucle infinie avec while
2 while(true) {
3     // code à exécuter à l'infinie
4 }
```

```
1 void loop() {
2 }
```

Dans **Arduino**,
`loop()` est une boucle
infinie

Les fonctions

La segmentation du code en fonctions permet à un programmeur de créer des morceaux de code modulaires qui effectuent une tâche définie. Le cas typique de création d'une fonction est celui où l'on doit effectuer plusieurs fois la même action dans un programme.

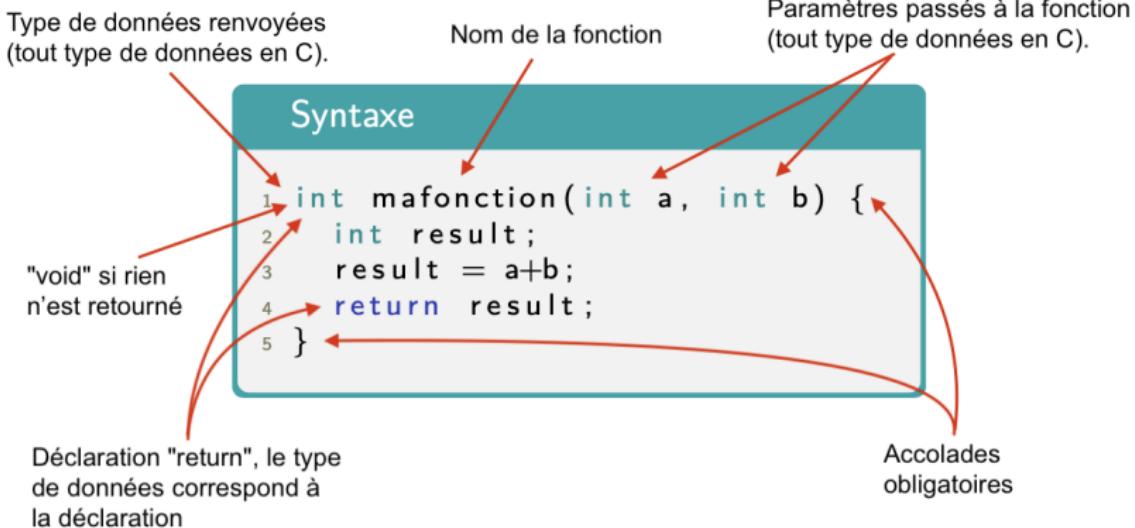
La standardisation des morceaux de code en fonctions présente plusieurs avantages :

- ▶ Les fonctions aident le programmeur à rester organisé. Cela aide souvent à conceptualiser le programme.
- ▶ Les fonctions codifient une action en un seul endroit de sorte que la fonction n'a besoin d'être pensée et déboguée qu'une seule fois.
- ▶ Cela réduit également les risques d'erreurs de modification, si le code doit être modifié.
- ▶ Les fonctions rendent l'ensemble du programme plus compact car les sections de code sont réutilisées plusieurs fois.
- ▶ Les fonctions facilitent la réutilisation du code dans d'autres programmes en le rendant plus modulaire, et rendent aussi souvent le code plus lisible.

Les fonctions

Remarque

Il y a deux fonctions requises dans un sketch Arduino, `setup()` et `loop()`. D'autres fonctions doivent être créées en dehors de ces deux fonctions.



Les fonctions

Pour "appeler" la fonction d'addition (mafondction), on lui passe des paramètres du type de données qu'elle attend :

Appel de la fonction

```
1 void loop(){
2     int i = 2;
3     int j = 3;
4     int k;
5
6     k = mafondction(i, j); // k = 5
7 }
```

Les fonctions

La fonction doit être déclarée en dehors de toute autre fonction, donc **mafondction()** peut être déclarée au-dessus ou en dessous de la fonction **loop()**.

Programme

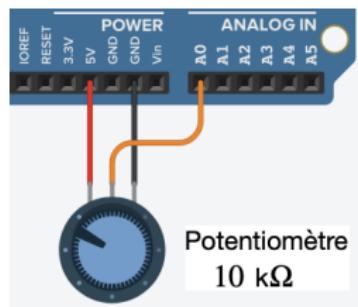
```
1 void setup(){
2     Serial.begin(9600);
3 }
4 void loop() {
5     int i = 2;
6     int j = 3;
7     int k;
8
9     k = mafondction(i, j); // k = 5
10    Serial.println(k);
11    delay(500);
12 }
13
14 int mafondction(int a, int b){
15     int result;
16     result = a+b;
17     return result;
18 }
```

Exemple

Cette fonction lira une tension avec `analogRead()`.

Fonction

```
1 float Lecture_Tension(){
2     float V;
3     V = analogRead(0)*5.0/1024.0; // pin 0
4     return V;
5 }
```



Pour appeler la fonction, on l'assigne simplement à une variable.

```
1 float tension;
2 tension = Lecture_Tension();
```

Comme on peut le voir, même si une fonction n'a pas de paramètres et qu'aucun retour n'est attendu : `()` et `;` doivent être donnés.

Les fonctions

Exemple complet

Exemple complet

```
1 void setup() {
2     Serial.begin(9600);
3 }
4
5 void loop() {
6     Serial.print("TensionLue = ");
7     Serial.print(analogRead(0)*5.0/1024.0);
8     float tension;
9     tension = Lecture_Tension();
10    Serial.print("\t TensionFonc = ");
11    Serial.println(tension);
12    delay(1000);
13 }
14
15 float Lecture_Tension(){
16     float V;
17     V = analogRead(0)*5.0/1024.0; // analog pin 0 (A0)
18     return V;
19 }
```

Les fonctions - Fonction MAP

Mappe un nombre d'une plage à une autre. C'est-à-dire qu'une valeur de fromLow serait mappée à toLow, une valeur de fromHigh à toHigh, des valeurs intermédiaires à des valeurs intermédiaires, etc.

Syntaxe

```
1 map(value, fromLow, fromHigh, toLow, toHigh);
```

- ▶ value : le nombre à mapper.
- ▶ fromLow : la limite inférieure de la plage actuelle de la valeur.
- ▶ fromHigh : la limite supérieure de la plage actuelle de la valeur.
- ▶ toLow : la limite inférieure de la plage cible de la valeur.
- ▶ toHigh : la limite supérieure de la plage cible de la valeur.

Exemple

```
1 /* Mapper une valeur analogique sur 8 bits (0 à 255) */
2 void loop() {
3     int val = analogRead(0);
4     val = map(val, 0, 1023, 0, 255);
5     Serial.println(val);
6 }
```

Exercices

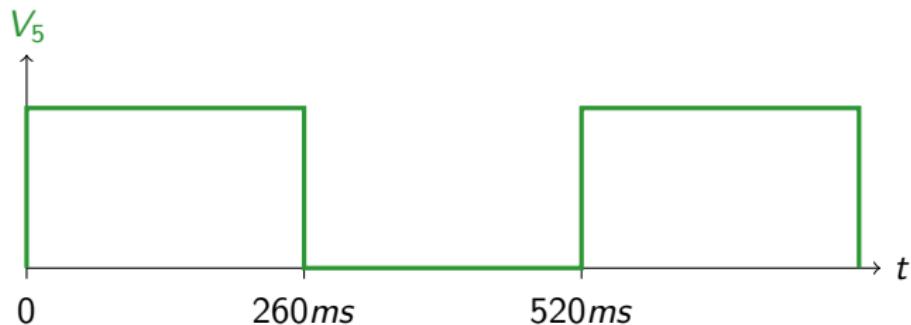
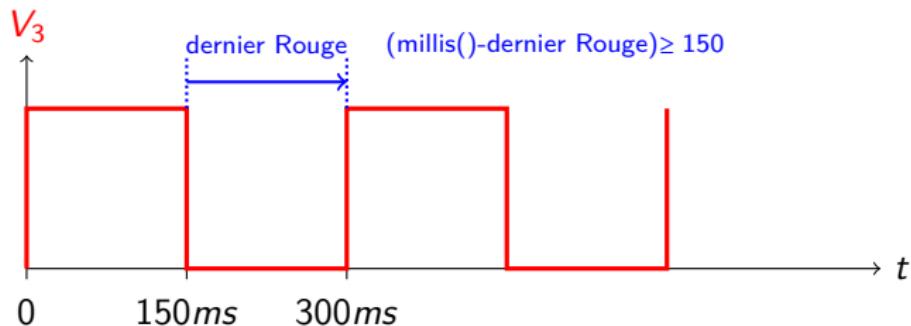
Exercice 1

Réaliser un programme **Arduino** pour faire clignoter simultanément 2 LEDs :

- ▶ Une LED rouge sur la broche numérique 3 avec une période de 300 ms
- ▶ Une LED jaune sur la broche numérique 5 avec une période de 520 ms
- ▶ Ne pas oublier de monter chaque LED avec une résistance en série de $330\ \Omega$ (ou $1\ k\Omega$)

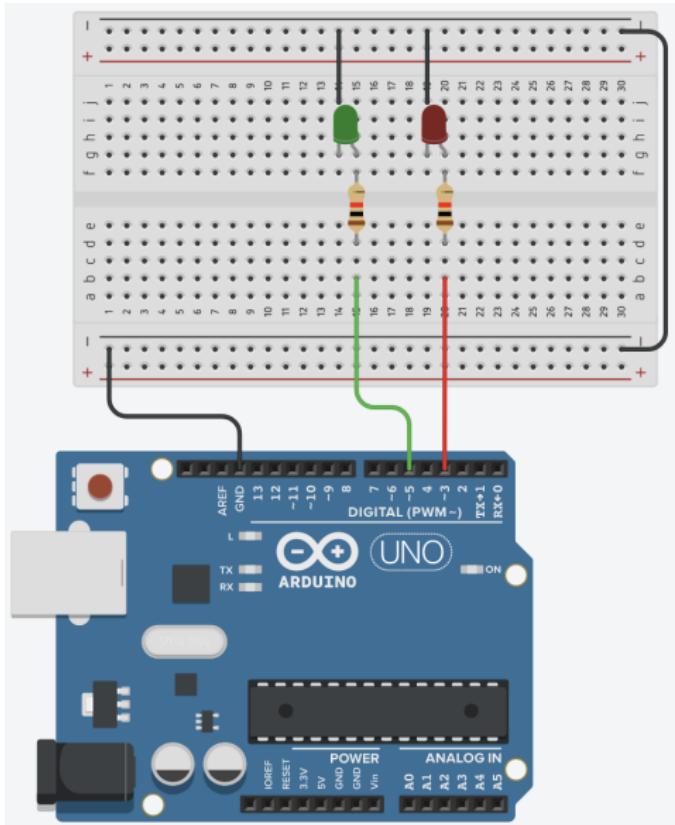
Exercices

Exercice 1 : Solution (Signaux à générer dans la boucle principale)



Exercices

Exercice 1 : Solution



Exercices

Exercice 1 : Solution

Programme complet

```
1 #define PIN_LED_ROUGE 3
2 #define DEMI_PERIODE_ROUGE 150
3 #define PIN_LED_Verte 5
4 #define DEMI_PERIODE_Verte 260
5 #define basculeLED(x) digitalWrite(x, !digitalRead(x))
6 void setup() {
7     // Configue les broches des LEDs en sortie
8     pinMode(PIN_LED_ROUGE, OUTPUT);
9     pinMode(PIN_LED_Verte, OUTPUT);
10 }
11 void loop() {
12     static unsigned long dernierRouge = 0, dernierVerte = 0;
13     if ((millis()-dernierRouge) >= DEMI_PERIODE_ROUGE) {
14         dernierRouge = dernierRouge+DEMI_PERIODE_ROUGE;
15         basculeLED(PIN_LED_ROUGE);
16     }
17     if ((millis()-dernierVerte) >= DEMI_PERIODE_Verte) {
18         dernierVerte = dernierVerte+DEMI_PERIODE_Verte;
19         basculeLED(PIN_LED_Verte);
20     }
21 }
```

Exercices

Exercice 2 : Envoyer l'alphabet sur le port série

L'objectif est d'envoyer l'ensemble des lettres de l'alphabet de manière la plus intelligente possible, c.à.d., sans écrire 26 fois `Serial.print()`;

La fonction `setup()` restera la même que celle vue précédemment. Un délai de 250 ms est attendu entre chaque envoi de lettre et un délai de 5 secondes est attendu entre l'envoi de deux alphabets.

Exercices

Exercice 2 : Solution

Programme complet

```
1 void setup() {
2     Serial.begin(9600);
3 }
4
5 void loop()
6 {
7     char i = 0;
8     char lettre = 'a'; // ou 'A' pour envoyer en majuscule
9     Serial.println("----- L'alphabet -----");
10    // on commence les envois
11    for(i=0; i<26; i++)
12    {
13        Serial.print(lettre); // on envoie la lettre
14        lettre = lettre + 1; // on passe à la lettre suivante
15        delay(250); // on attend 250ms avant de réenvoyer
16    }
17    Serial.println(""); // on fait un retour à la ligne
18    delay(5000); // on attend 5 secondes avant de renvoyer
19                // l'alphabet
20 }
```