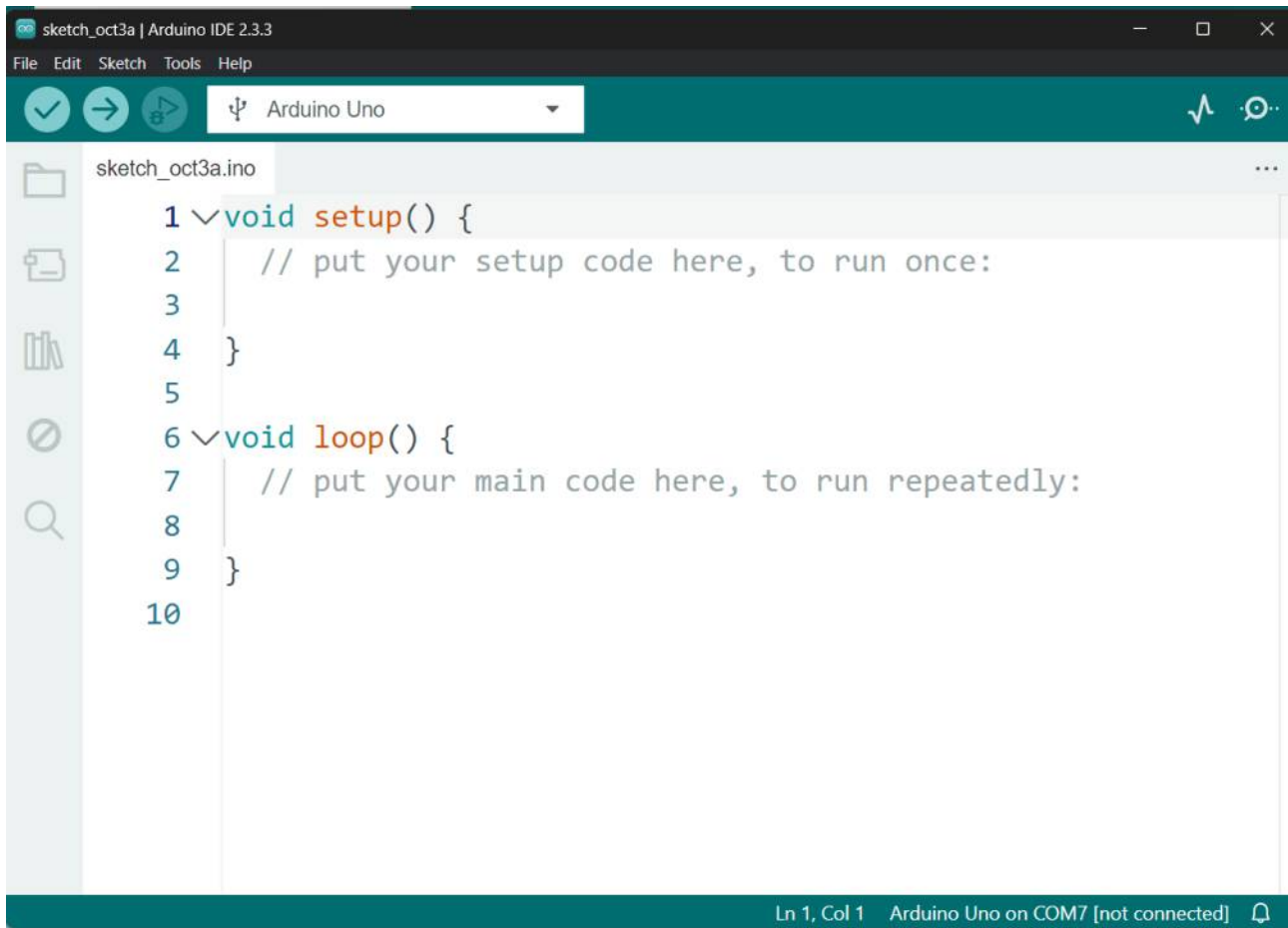


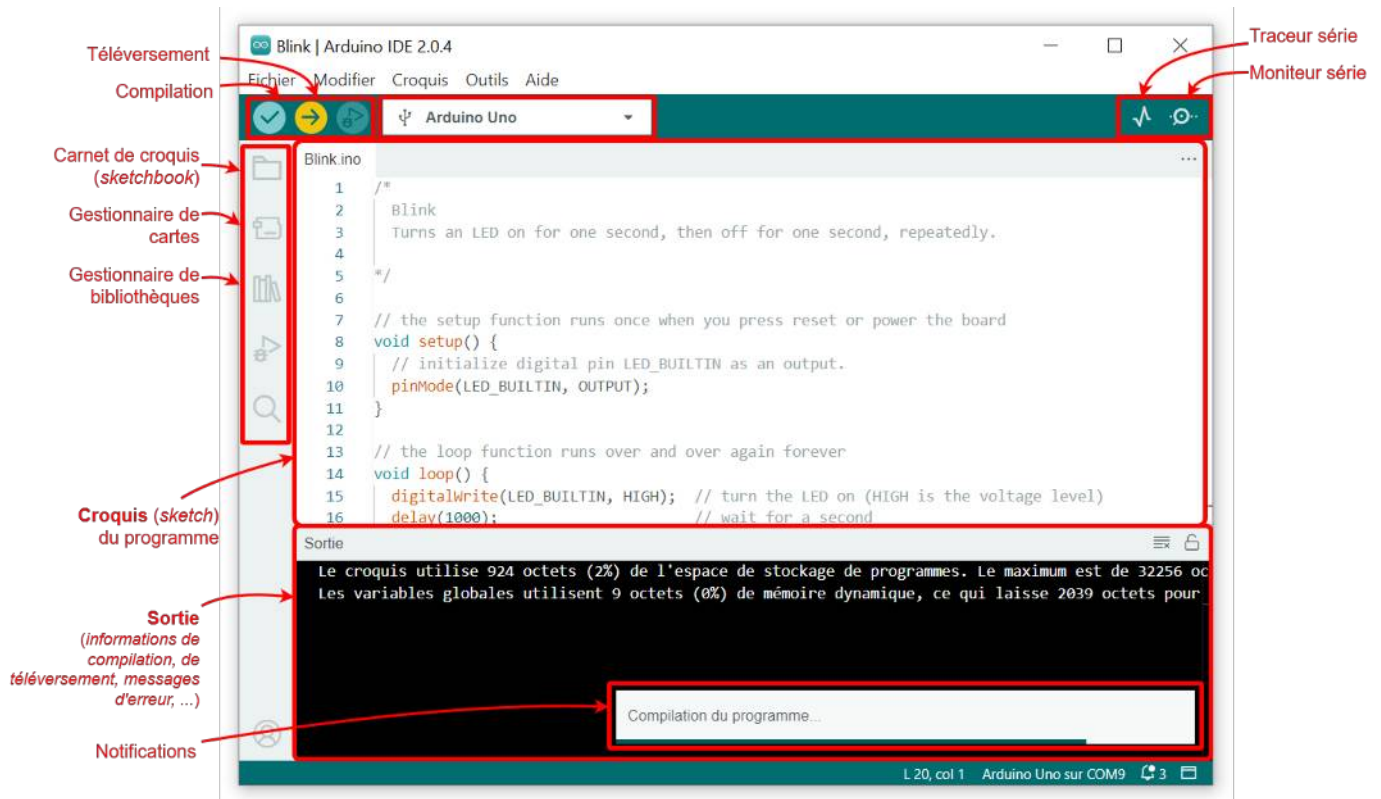
Arduino

Programmation

Logiciel IDE



```
sketch_oct3a | Arduino IDE 2.3.3
File Edit Sketch Tools Help
Arduino Uno
sketch_oct3a.ino
1 void setup() {
2   // put your setup code here, to run once:
3
4 }
5
6 void loop() {
7   // put your main code here, to run repeatedly:
8
9 }
10
Ln 1, Col 1  Arduino Uno on COM7 [not connected]
```



Fonction *void loop*, *void setup*

Ces fonctions doivent figurer dans chaque programme et être appelés une seule fois, même si l'une des fonctions n'est pas utilisée. Le fait est que lorsque vous démarrez le microcontrôleur, le micrologiciel intégré commence à fonctionner et la première chose qu'il fait est de vérifier si un nouveau programme a commencé à être téléchargé depuis l'ordinateur. Si l'utilisateur n'a pas lancé le microprogramme, le contrôleur commence à exécuter le code chargé.

Les deux boucles sont appelées par la fonction intégrée `main()` du fichier `main.cpp`. La fonction `void setup()` est appelée une fois et `void loop()` est appelée un nombre infini de fois dans la boucle `for`. S'il y a plus d'une fonction `void setup()` ou `void loop()` dans le programme IDE Arduino, une redéfinition de '`void setup()`' ou une redéfinition de '`void loop()`' apparaîtra pendant la compilation du code dans l'IDE Arduino, respectivement.

Utiliser fonction *void setup* Arduino

Les accolades indiquent le début et la fin de la fonction. Toutes les commandes doivent donc être placées entre elles. Si vous supprimez ou placez accidentellement une accolade supplémentaire, vous obtiendrez une erreur lors de la compilation. La procédure `void setup` n'est appelée qu'une seule fois et sert à affecter le mode aux broches ou aux commandes qui doivent être exécutées uniquement au moment du chargement du programme.

Exemple : après avoir chargé ce code, la LED clignotera une fois (une seule exécution):

```
void setup() {
  pinMode(13, OUTPUT);
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}

void loop() {
}
```

Utiliser fonction *void loop* Arduino

Après avoir exécuté le cycle de configuration, le programme entre dans une boucle, qui se répète tant que la carte est sous tension. Si la boucle contient une seule instruction, elle sera exécutée des milliers de fois par seconde. Si vous décidez d'écrire un croquis pour le clignotement des LED sur l'Arduino, vous devez ajouter un délai au code pour l'exécution du programme, sinon le clignotement des LED ne sera pas perceptible.

Après le chargement de ce code, la LED clignote en continu:

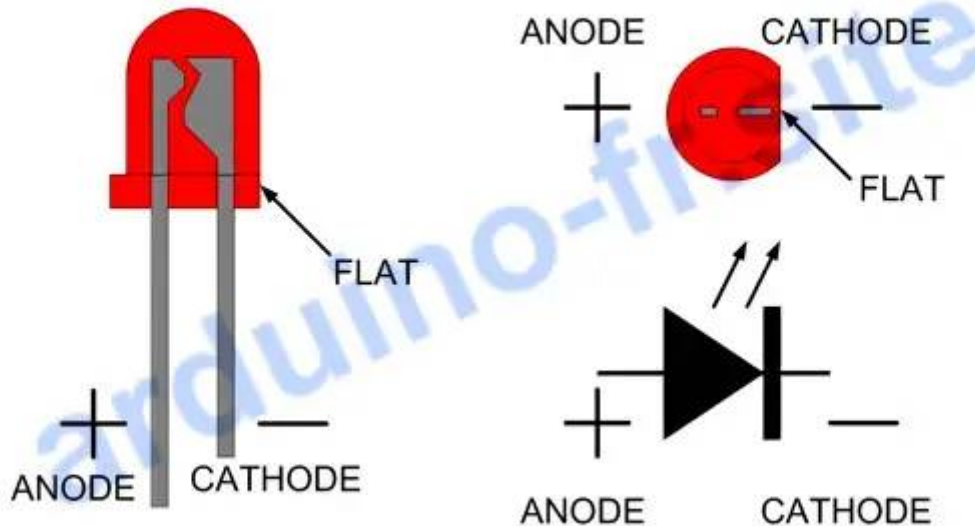
```
void setup() {
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}
```

Ainsi, si vous devez allumer la LED une fois pour indiquer le fonctionnement du dispositif sur le microcontrôleur Arduino Nano lors de l'exécution du programme, il est préférable d'écrire la commande dans la fonction `void setup()`. Si vous devez effectuer une action dans le programme de façon permanente, par exemple, pour afficher des informations sur l'écran LCD 1602, la commande doit être placée dans la fonction `void loop()`.

Fonctionnement de la LED Arduino

Fonctionnement de la LED



Fonctionnement de la LED Arduino: pinout, datasheet

Les LED sont des éléments semi-conducteurs utilisés pour l'indication et, plus récemment, pour l'éclairage. Elles ont une polarité, c'est-à-dire qu'elles sont sensibles au sens de circulation du courant continu (voir l'image ci-dessus). Si vous connectez la LED de manière incorrecte (l'élément ne brûlera pas), le courant ne passera pas dans le circuit et la LED ne s'allumera pas. L'anode (la longue branche de la LED) est toujours connectée au côté positif.

La luminosité de l'émission LED dépend directement du courant qui traverse l'élément. Cependant, le courant maximal autorisé ne doit pas être dépassé, sinon la LED tombera en panne. Pour éviter cela, la LED doit être connectée en série par l'intermédiaire d'une résistance de limitation de courant. La résistance peut être connectée à n'importe quelle branche de la LED (anode ou cathode). La résistance est calculée à l'aide de la formule suivante:

$$R = (V_A - V_L) / I_L \text{ (Loi d'Ohm)}$$

V_A – tension de la broche du microcontrôleur Arduino (5 Volts)

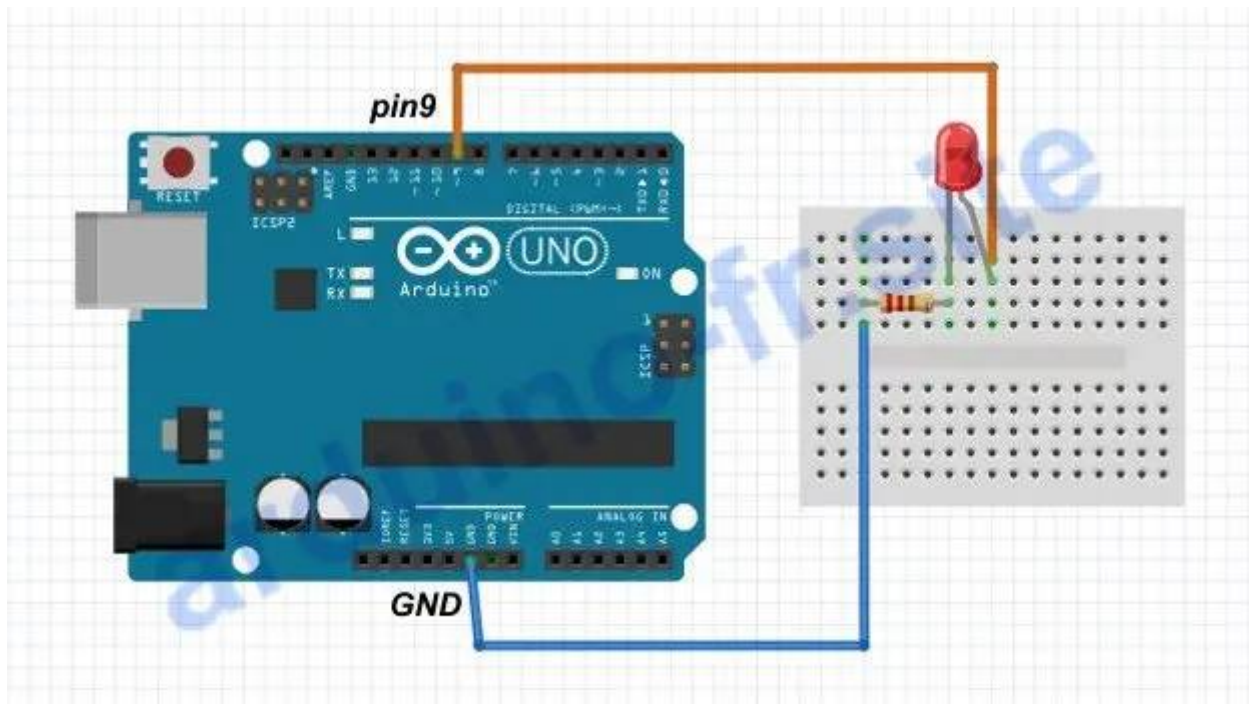
V_L – chute de tension sur la LED (spécifiée dans les caractéristiques)

IL – intensité de la LED, spécifiée dans les caractéristiques (10-20 mAmp)

Pour les LED conventionnelles, la chute de tension est d'environ 2 volts et le courant d'incandescence recommandé est de 15 mA. Ainsi, lorsqu'elle est connectée à la broche 5V du microcontrôleur Arduino, la résistance du circuit doit être d'au moins:

$$R = (5 - 2) / 0,015 = 200 \text{ ohms}$$

Nous pouvons donc utiliser des résistances de 220 Ohm.



Après avoir installé l'IDE Arduino, connectez l'Arduino à l'ordinateur via USB. Ensuite, dans les paramètres, sélectionnez la carte que vous utilisez et spécifiez le port COM sur lequel la carte est détectée. Dans le schéma ci-dessus, la LED de gauche sera allumée en permanence, et la LED de droite sera allumée à l'aide d'un bouton. Les ports d'alimentation (5V et GND) sont utilisés pour l'alimentation et ne sont pas contrôlés par le programme.

Les broches numériques doivent être utilisées pour allumer la LED à partir de l'Arduino. Dans cet exemple, la broche 9 est utilisée, mais n'importe quelle broche peut être utilisée en la spécifiant dans le programme. Connectez la LED à l'Arduino via la planche à pain comme indiqué sur l'image et chargez le programme suivant. La broche 9 du microcontrôleur est alimentée en +5V (HIGH) ou 0V (LOW) à l'aide de la commande digitalWrite.

Programme Arduino pour connecter et allumer une LED

```
const int LED = 9 ;
```

```
void setup() {  
  pinMode(LED, OUTPUT);  
}  
  
void loop() {  
  digitalWrite(LED, HIGH);  
  delay(1000);  
  
  digitalWrite(LED, LOW);  
  delay(1000);  
}
```

Explication du code pour allumer une LED à partir d'Arduino:

- pour allumer la LED, mettez la broche Arduino en mode OUTPUT à l'aide fonction pinMode;
- la vitesse de clignotement de la LED dépend du délai spécifié dans la fonction delay.

Vous pouvez raccourcir le délai dans le programme, par exemple en spécifiant delay(100); la LED connectée à la carte Arduino clignotera alors plus souvent.

Qu'est-ce que le moniteur série Arduino, signification?

Le moniteur série consiste en une fenêtre divisée en deux parties. Au centre se trouvent les données reçues du port série. La partie supérieure, à gauche, est le champ de saisie où vous pouvez envoyer des données depuis votre ordinateur. À droite se trouve le menu des paramètres. Afin d'éviter une erreur lors du chargement du croquis et de l'ouverture du moniteur série, vous devez sélectionner le port COM sur lequel la carte Arduino UNO est détectée.



Pour ouvrir le moniteur série, cliquez sur l'icône dans le coin supérieur droit de l'IDE Arduino ou utilisez la combinaison de touches Ctrl+Shift+M. Par défaut, le moniteur série Arduino est réglé sur 9600 bps – Serial.begin(9600), donc beaucoup de programme utilisent cette vitesse. Si le débit en bauds du code et les paramètres du moniteur série sont différents, toutes les informations seront affichées sous forme de hiéroglyphes et de carrés.

Envoyer et recevoir des données au moniteur série

Pour faire fonctionner l'utilitaire, utilisez les commandes suivantes:

Serial.begin(); – la commande démarre le moniteur série

Serial.end(); – arrête et efface le moniteur série

Serial.print(); – envoie des données au moniteur série

Serial.println(); – envoie des données avec un saut de ligne

Serial.read(); – reçoit des données du moniteur série

Serial.parseInt(); – lit les grands nombres à partir du moniteur série

Programme pour envoyer de texte sur moniteur série

```
void setup(){
  Serial.begin(9600); // initialiser le moniteur série
  Serial.print("Hello!"); // afficher Hello! sur le moniteur série
}

void loop(){
}
```

Programme pour envoyer de variable / Serial.println Arduino

```
int x;

void setup(){
  Serial.begin(9600); // initialiser le moniteur série
}

void loop(){
  x = x + 1;
  Serial.print("x = ");
  Serial.println(x); // afficher variable sur le moniteur série

  delay(1000);
}
```

Explication du code pour moniteur série Arduino:

les guillemets sont utilisées pour la sortie de texte dans les fonctions Serial.print(), aucune guillemet n'est utilisée pour les variables.

Saut de ligne et tabulation / Serial.begin Arduino

Si vous voulez désactiver le moniteur série ou effacer le presse-papiers, Serial.end() est utilisé. Pour relancer l'échange d'informations entre l'Arduino et l'ordinateur, la commande Serial.begin() est exécutée à nouveau. Et pour afficher les messages sur le moniteur série du microcontrôleur Arduino Uno, vous pouvez utiliser des tabulations et des sauts de ligne pour rendre les informations plus lisibles. Voir l'exemple programme suivant.

```
int x;

void setup(){
  Serial.begin(9600); // initialiser le moniteur série
}

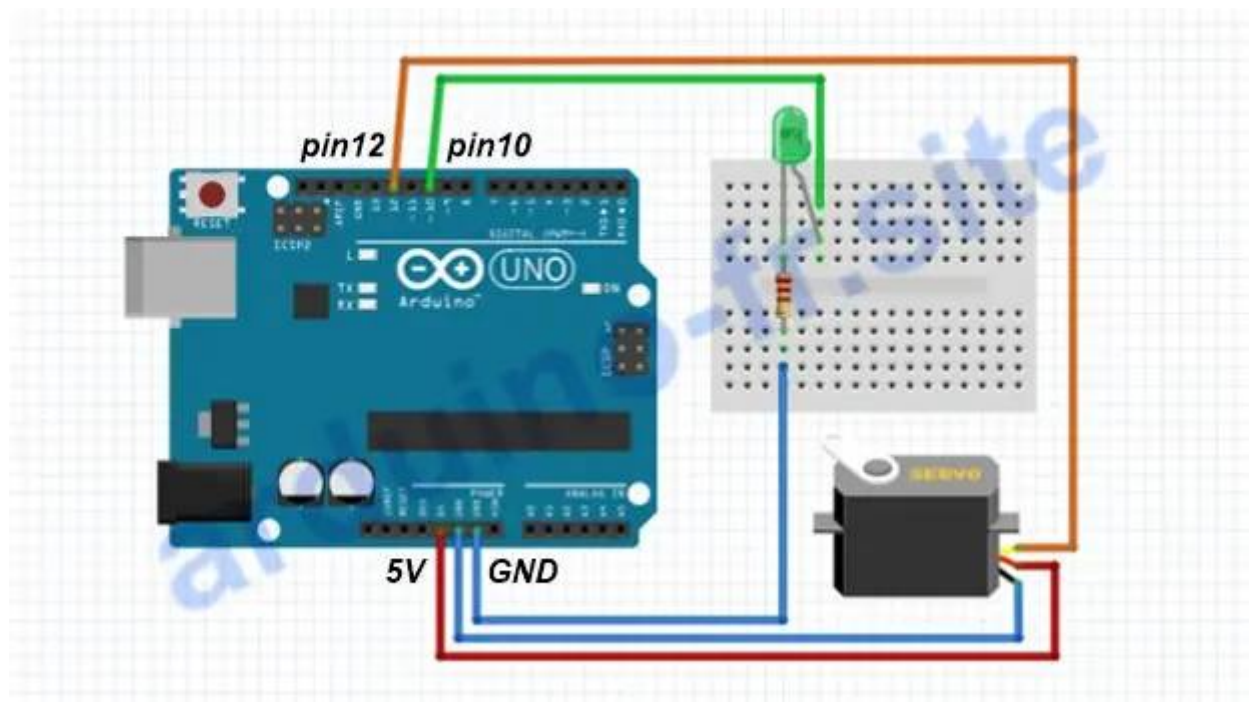
void loop(){
  x = x + 1;

  Serial.print("x");
  Serial.print("\t"); // tabulation
  Serial.print("=");
  Serial.print("\t"); // tabulation
  Serial.println(x); // afficher variable

  delay(1000);
}
```

Voyons comment contrôler Arduino depuis un ordinateur à travers le moniteur de port en utilisant la fonction **Serial.available**. Cette fonction est utilisée pour obtenir le nombre d'octets disponibles pour la lecture dans le tampon du moniteur Arduino IDE. La mémoire tampon du moniteur série peut stocker un maximum de 64 octets. La fonction Serial.available() est également utilisée lors de la communication entre le module bluetooth hc-05 et Arduino.

Interface pour contrôler son Arduino depuis un PC



Contrôler LED Arduino à partir du clavier ordinateur

La fonction Serial.available du langage Arduino vérifie les informations qui arrivent au microcontrôleur par le port série UART. Dans cet exemple, nous contrôlerons la LED et le servomoteur connectés à la carte Arduino via le clavier de l'ordinateur. Vous pouvez envoyer des commandes au microcontrôleur via UART non seulement avec des chiffres, mais aussi avec des lettres de l'alphabet latin et des caractères spéciaux.

Interface pour contrôler son Arduino depuis un PC

```
#include "Servo.h"
Servo servo;
int val;

const int LED = 10 ; // broche de connexion LED
#define SERVO 12 // broche de connexion servo

void setup() {
  Serial.begin(9600);
  pinMode(LED, OUTPUT);
  servo.attach(SERVO);
}

void loop() {
  // attente d'une commande du moniteur de port
  if (Serial.available()) {
    val = Serial.read();

    // allumer ou éteindre la LED
    if (val == '1') { digitalWrite(LED, HIGH); }
    if (val == '0') { digitalWrite(LED, LOW); }

    // tourner le servo en fonction de la commande
    if (val == '2') { servo.write(0); }
    if (val == '3') { servo.write(90); }
```

```
    if (val == '4') { servo.write(180); }  
  }  
}
```

Explication du code pour contrôler Arduino depuis ordinateur:

ouvrez le moniteur de port et envoyez la commande sous forme de chiffre;

il est possible de ne pas connecter de servo, mais de contrôler uniquement l'allumage LED.

IF ... ELSE Arduino (opérateurs logiques de comparaison)

Référence du langage Arduino if ... else

Arduino if est utilisé avec des opérateurs de comparaison pour vérifier la vérité des conditions. Les parenthèses regroupent logiquement plusieurs lignes en un seul bloc. Exemple de code (arduino if 2 conditions) qui vérifie si la valeur d'une variable dépasse un nombre donné:

```
if (water > 100) { digitalWrite(12, HIGH); digitalWrite(10, LOW); }  
if (water < 100) { digitalWrite(12, LOW); digitalWrite(10, HIGH); }
```

La construction if..else permet de mieux contrôler le programme qu'une simple fonction if Arduino. Il vous permet de définir des actions non seulement pour le cas où l'expression est vraie, mais aussi pour le cas contraire (lorsque la valeur de l'expression est fausse):

```
if (water > 100) { digitalWrite(12, HIGH); digitalWrite(10, LOW); }  
else { digitalWrite(12, LOW); digitalWrite(10, HIGH); }
```

Plusieurs conditions logiques if else, else if Arduino

Dans l'IDE Arduino, else peut être suivi d'un autre fonction if, créant ainsi toute une chaîne de contrôles. Les contrôles seront exécutés l'un après l'autre jusqu'à ce qu'une expression vraie soit rencontrée. Dans ce cas, le bloc de code qui suit la condition sera exécuté:

```
if (water > 100) { digitalWrite(12, HIGH); digitalWrite(10, LOW); }  
else if (water > 200) { digitalWrite(12, LOW); digitalWrite(10, HIGH); }
```

```
else if (water > 300) { digitalWrite(12, HIGH); digitalWrite(10, HIGH); }  
else { digitalWrite(12, LOW); digitalWrite(10, LOW); }
```

Si aucune expression vraie n'est trouvée, le bloc else final (s'il y en a un dans le programme) sera exécuté. Une construction else if peut être sans un bloc else final et vice versa. Il n'y a pas de limite au nombre de branches else if dans le code – arduino if 2 conditions et plus.

Opérateurs logiques de comparaison ==, !=, <, >

Ne confondez pas le signe égal « = » avec l'opérateur de comparaison Arduino « == ». L'utilisation d'un signe égal dans une instruction conditionnelle if peut donner un résultat différent lors de l'exécution du programme. Par exemple, fragment de code if (y = 100) n'est pas la même chose que boucle if (y==100). Le signe égal est un opérateur d'affectation qui fixe la valeur d'une variable à 40 et ne vérifie pas si la variable est 100.

Opérateurs de comparaison Arduino

x == y (x égal y)

x != y (x n'est pas égal à y)

x < y (x est inférieur à y)

x > y (x est plus grand que y)

x <= y (x est inférieur ou égal à y)

x >= y (x est supérieur ou égal à y)

Opérateurs logiques Arduino (not, and, or)

Les opérateurs booléens sont utilisés pour relier plusieurs valeurs logiques:

! (not) logique NOT, une négation.

&& (and) logique ET

|| (or) logique OU

```
if (water > 100 and value > 100) {digitalWrite(12,HIGH);digitalWrite(10,LOW);}  
else { digitalWrite(12, LOW); digitalWrite(10, HIGH); }
```

L'ordre des conditions joue un rôle important lorsqu'il s'agit d'optimiser le code et d'essayer de rendre un programme plus rapide. L'idée est que les expressions / valeurs logiques sont vérifiées de gauche à droite, et que si la première vérification rend l'expression invalide, la poursuite de la

vérification des conditions s'arrête. Dans le dernier exemple, si la condition `water > 100` est fausse, la valeur de l'expression `value > 100` n'est plus vérifiée.

la boucle for et while

Nous piloterons une LED via une sortie analogique de façon à la faire s'allumer et s'éteindre progressivement. Comment fonctionnent **while**, comment utiliser correctement les boucles dans vos programmes et quelles sont les erreurs à éviter. À l'aide d'exemples simples, nous montrons comment arrêter une boucle ou passer d'une boucle à une autre.

Description de boucle WHILE / DO WHILE Arduino

La boucle `while` s'exécute de manière continue et indéfinie tant que la condition entre parenthèses est vraie. La sortie de la boucle sera atteinte lorsque la variable de la condition `while` changera, donc quelque chose doit changer sa valeur. Un changement de variable peut se produire dans le code du programme à l'intérieur d'une boucle ou lors de la lecture des valeurs d'un capteur quelconque, tel qu'un télémètre à ultrasons HC-SR04.

```
while (condition) {  
    // instruction(s) à exécuter;  
}
```

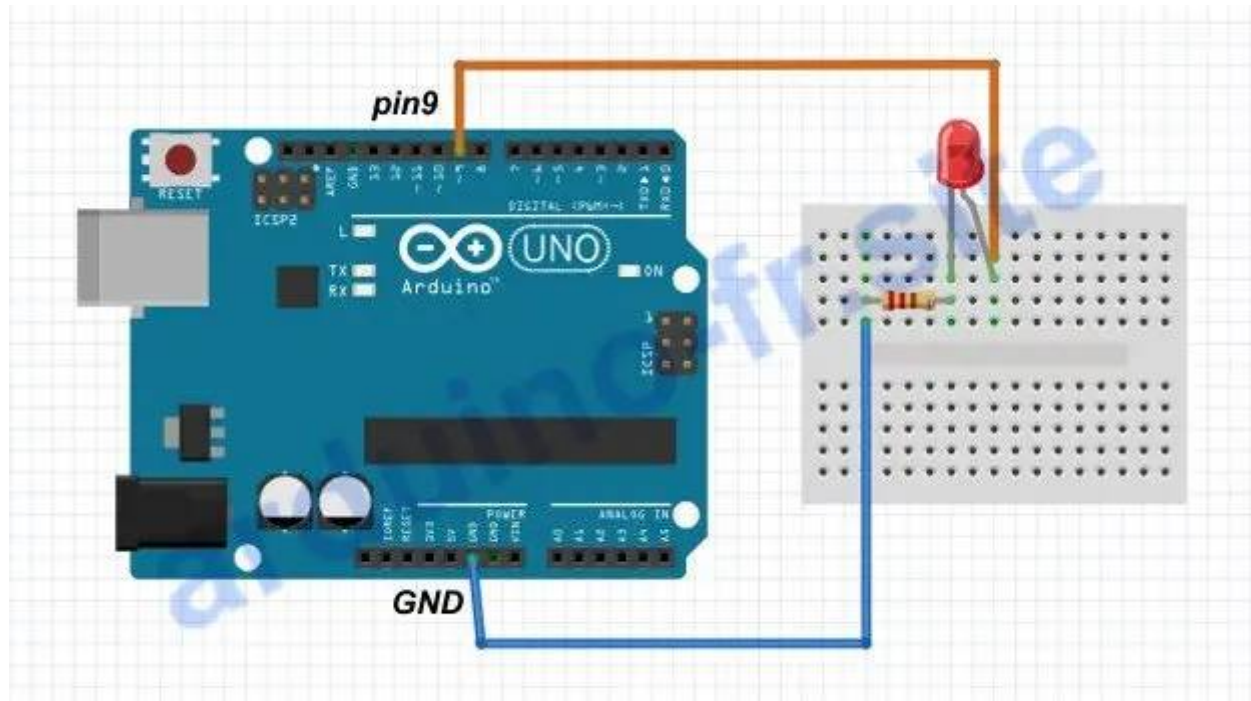
Une autre boucle qui peut être utilisée dans l'IDE Arduino : c'est une boucle avec une postcondition `do ... alors que`. En utilisant cette construction, les commandes de la boucle seront exécutées au moins une fois, quelle que soit la condition, car la condition est vérifiée après l'exécution du corps de la boucle. Dans l'exemple de code suivant, la DEL s'allume indépendamment du capteur et ce n'est qu'ensuite que la postcondition est vérifiée.

```
do {  
    // instruction(s) à exécuter;  
}  
while (condition);
```

`do` est un bloc qui est exécuté au moins une fois et répété en boucle tant que la condition d'exécution `while`, sera vraie « true ». Une fois que la condition d'exécution sera fausse « false », la mise en boucle du bloc ne sera plus exécutée mais seulement exécutée une seule fois à chaque

lecture du programme void loop(). Ne pas oublier le point virgule « ; » à la suite de while() quand ce dernier succède au bloc do{ }.

Référence boucle WHILE Arduino – exemple, syntaxe



```
#define RED_LED 9
int i = 0;
void setup() {
  // initialise les broches
  pinMode(RED_LED, OUTPUT);
}

void loop() {
  while(i<255) {
    analogWrite(RED_LED, i);
    delay(10);
    i++;
  }

  while(i>0) {
    analogWrite(RED_LED, i);
    delay(10);
    i--;
  }
}
```

Si la condition d'exécution () du premier cycle est vraie « true », soit la variable de contrôle « i » inférieure à 255. L'action programmée sera exécutée. Le bloc d'actions programmées entre accolades {} – réalise le calcul et résultat (allume en douceur la LED) tant que la condition d'exécution est vraie « true ». Dans le deuxième cycle while, il y aura une extinction progressive de la LED tant que la variable « i » est supérieure à zéro.

L'instruction break pour sortir d'une boucle WHILE

Si vous souhaitez quitter le corps de la boucle, quelles que soient les conditions spécifiées, utilisez l'instruction break ou goto. Une instruction break vous permet de sortir de la boucle et le programme continuera à exécuter les commandes suivantes. L'instruction goto vous permet non seulement de sortir de la boucle, mais aussi de poursuivre l'exécution du programme à partir de l'endroit où vous le souhaitez, par exemple, vous pouvez passer à une autre boucle.

Chaque boucle en C++ et dans le langage de programmation Arduino est une action qui est répétée plusieurs fois ou infiniment de fois. Aucun programme pour l'Arduino n'est sans boucles, par exemple, la boucle void loop est appelée dans une boucle infinie. Il existe trois types de boucles: for, while et do while.

Description de la boucle FOR Arduino IDE

La boucle for est utilisée pour répéter certaines commandes placées entre accolades. Cette boucle est adaptée à l'exécution de toute action répétitive. Le but étant donc d'initialiser une variable à une certaine valeur, la faire évoluer d'un pas fixe à chaque tour pour sortir de la boucle une fois une condition atteinte, voilà la syntaxe fournie par le site officiel arduino:

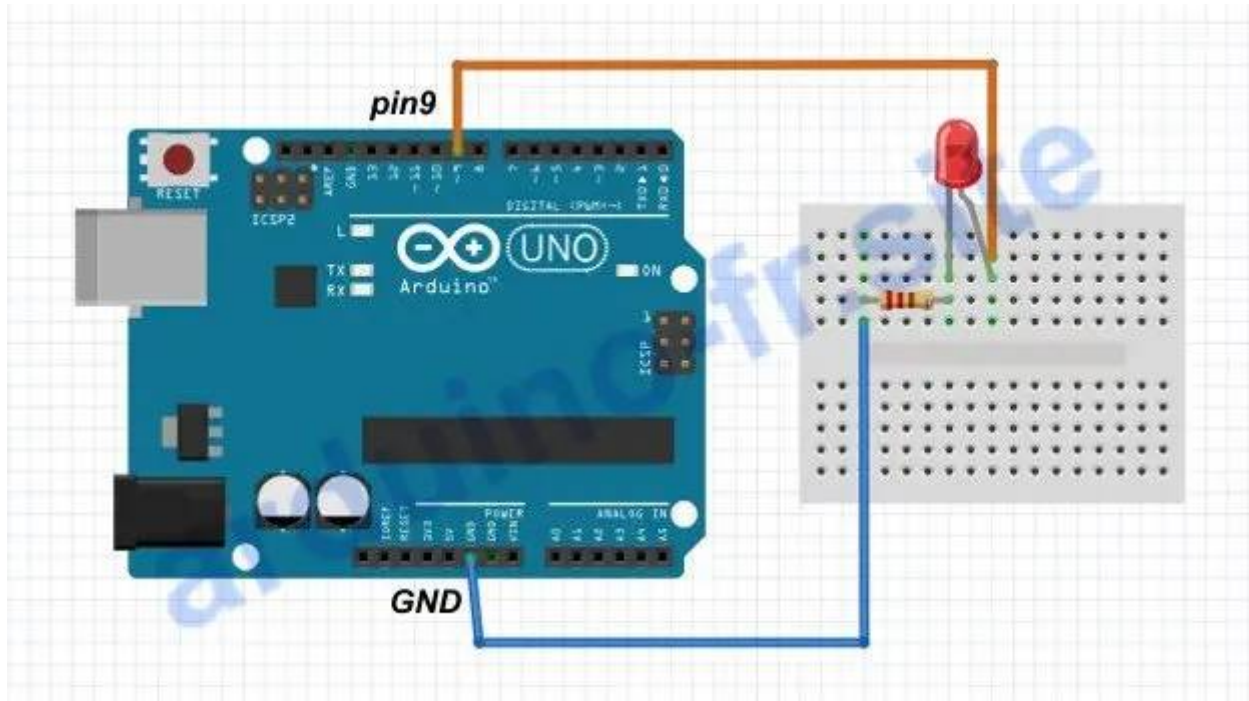
```
for (initialization; condition; incrementation) {  
    // instruction(s) à exécuter;  
}
```

Initialisation – nous assignons la valeur de départ de la variable qui évoluera dans notre boucle. Notez qu'il est possible de déclarer la variable en question à cet endroit (il suffira de rajouter le type de la variable devant celle ci, exemple pour une variable nommé indexBoucle qui commencera à 0, vous mettrez à cet endroit int indexBoucle = 0)

Condition – la boucle s'exécutera tant que cette condition sera vrai (souvent une comparaison)

Incrément – la valeur de laquelle évoluera la variable à chaque passage dans la boucle for

Référence boucle FOR – exemple, syntaxe



Pour générer une tension variable ou pseudo analogique en sortie d'une broche digitale de l'Arduino, il va falloir changer très rapidement l'état de la sortie. 6 pins d'une Arduino Uno or Nano fournissent la sortie signal PWM – 3, 5, 6, 9, 10, 11. Utilise une boucle for pour allumer et éteindre une LED en douceur, on doit lui donner trois paramètres entre les parenthèses (le code qui se trouve à l'intérieur de la boucle sera exécuté 255 fois):

```
#define RED_LED 9

void setup() {
  // initialise les broches
  pinMode(RED_LED, OUTPUT);
}

void loop() {
  for(int i=0; i<=255; i++) {
    analogWrite(RED_LED, i);
    delay(10);
  }

  for(int i=255; i>=0; i--) {
    analogWrite(RED_LED, i);
    delay(10);
  }
}
```

Dans l'exemple de code, la première boucle commence avec une valeur initiale de la variable $i=0$, la condition stipule que la boucle s'exécutera jusqu'à ce que la variable soit égale ou supérieure à cinq i . La modification spécifie que la variable sera incrémentée de un à chaque étape de la boucle. Finalement, la sortie de la boucle `for` se produira lorsque la variable deviendra 255, de sorte que la LED s'allumera en douceur avant la fin de la boucle.

L'instruction `break` pour sortir d'une boucle `FOR`

Si vous souhaitez quitter le corps de la boucle, quelles que soient les conditions spécifiées, utilisez l'instruction `break` ou `goto`. Une instruction `break` vous permet de sortir de la boucle et le programme continuera à exécuter les commandes suivantes. L'instruction `goto` vous permet non seulement de sortir de la boucle, mais aussi de poursuivre l'exécution du programme à partir de l'endroit où vous le souhaitez, par exemple, vous pouvez passer à une autre boucle.

Fonction `map` Arduino IDE description

Syntaxe `map`

```
map(valeur_entrée, inférieur_entrée, supérieur_entrée, inférieur_sortie, supérieur_sortie);
```

On observe cinq paramètres:

`valeur_entrée` -> c'est le nombre à mapper (à transformer)

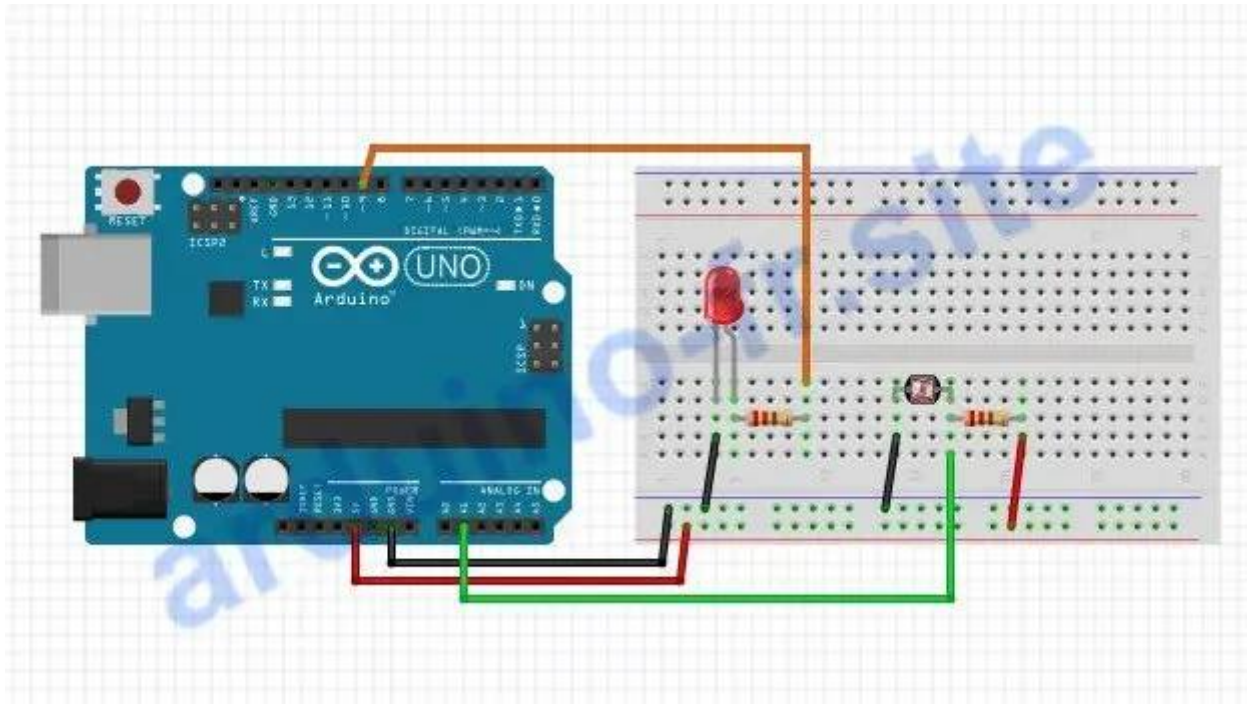
`inférieur_entrée` -> c'est la limite inférieure de l'intervalle actuel

`supérieur_entrée` -> c'est la limite supérieure de l'intervalle actuel

`inférieur_sortie` -> c'est la limite inférieure de l'intervalle résultat du changement

`supérieur_sortie` -> c'est la limite supérieure de l'intervalle résultat du changement

Function map Arduino exemple, syntaxe



Parfois, les valeurs que nous obtenons de la lecture d'un pin, comme par exemple un capteur, peuvent être hors d'une portée déterminée, c'est pour cela que nous devons les transférer à une nouvelle portée afin de pouvoir les utiliser correctement. Notez que fonction `map()` de la Arduino IDE peut inverser la plage, c'est-à-dire que la « limite inférieure » peut être plus grande que la « limite supérieure ». Exemple de programme avec LDR et la LED:

```
int value;

void setup() {
  Serial.begin(9600); // initialise la communication
  pinMode(A1, INPUT); // composante photorésistance sur la pin A1
  pinMode(9, OUTPUT); // composante la LED sur la pin A1
}

void loop() {
  // mesure la tension sur la broche A1
  value = analogRead(A1);
  Serial.println(value);

  /* la LED allumera plus fort si l'entourage est plus sombre.
  c'est à dire, "analogWrite()" est inversement proportionnelle à "value".
  On doit inverser "value" pour que sa valeur change de 0/1023 à 255/0 */

  value = map(value, 0, 1023, 255, 0);
  analogWrite(9, value);
}
```

Fonction map float Arduino, nombres fractionnaires

Dans l'exemple suivant, nous allons convertir les données de la photorésistance (valeurs de 0 à 1023) en volts (valeurs de 0 à 5) et les afficher sur le moniteur du port. Pour obtenir des valeurs avec des nombres fractionnaires, vous devez utiliser la fonction float mapf dans le croquis pour mesurer plus précisément la tension à l'entrée analogique. Sans modifier le circuit du premier exemple, chargez le programme suivant dans le microcontrôleur.

```
int val, led;

void setup() {
  Serial.begin(9600);
  pinMode(A1, INPUT);
  pinMode(9, OUTPUT);
}

void loop() {
  val = analogRead(A1);
  float data = mapf(val, 0, 1023, 0, 5);

  Serial.print(data);
  Serial.println(" V");

  led = map(val, 0, 1023, 255, 0);
  analogWrite(9, led);
  delay(100);
}

float mapf(float value, float fromLow, float fromHigh, float toLow, float
toHigh) {
  float result;
  result = (value - fromLow) * (toHigh - toLow) / (fromHigh - fromLow) + toLow;
  return result;
}
```

Lors de la conversion à l'aide de map(), la partie fractionnaire du nombre n'est pas arrondie, mais simplement écartée (non prise en compte). Ce fait doit être pris en compte lorsque vous travaillez avec le type de données float dans le langage Arduino. Notez que la fonction map utilise une division par 2, ce qui est une opération très coûteuse pour les microcontrôleurs. La fonction est donc très lente et ne doit pas être interrompue.

Utilisation de la fonction millis()

Pour pallier aux problèmes générés par l'utilisation de la fonction delay(), une solution possible est d'utiliser la fonction millis(). Dès la première utilisation de l'Arduino, la fonction delay() est utilisée afin de gérer les instructions en fonction du temps. Le problème majeur de la fonction delay() est qu'elle bloque l'exécution de la suite du code. Ceci devient très limitant lorsqu'on

travaille avec plusieurs composants (gestions de plusieurs LEDs ou capteurs). Nous allons voir dans ce tutoriel comment utiliser la fonction `millis()` pour remplacer la fonction `delay()`.

Description de la fonction `millis()`

La fonction `millis()` ne prend aucun paramètre et renvoie une valeur qui représente le nombre de millisecondes écoulées depuis la mise en tension de l'Arduino. La valeur est de type long non-signé (unsigned long, 4-bytes ou 32-bits). La valeur maximale qu'elle peut prendre est de 4,294,967,295 soit 49 jours.

N.B.: Il existe aussi la fonction `micros()` qui fonctionne sur le même principe mais renvoie des microsecondes.

```
void setup() {}

void loop() {
  Serial.print("Millisecons since ignition: ");
  Serial.println(millis())
  delay(200);
}
```

Remplacer `delay()`

Dans les premiers exemples d'Arduino, on utilise la fonction `delay()` pour exécuter un bloc de code périodiquement mais ce n'est pas sa fonction propre puisqu'elle bloque l'exécution du code. La fonction `millis()` est bien plus adaptée pour cette fonction.

Reprenons l'exemple Blink. Nous utilisons la fonction `delay` pour faire clignoter une LED toutes les 200 millisecondes.

```
void setup() {
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);
  delay(200);
  digitalWrite(13, LOW);
  delay(200);
}
```

Pour remplacer ce code par un structure avec `millis()`, l'idée est de créer une variable afin de conserver en mémoire à chaque itération la valeur de temps écoulé puis de la comparer à la valeur de temps actuelle. A chaque intervalle, nous inversons l'état de la led avec l'opérateur « ! ».

```

unsigned long currentTime=0;
unsigned long previousTime=0;
bool ledState=LOW;

void setup() {
  Serial.begin(9600);
  pinMode(13,OUTPUT);
}

void loop() {
  currentTime=millis();
  if((currentTime-previousTime)>200){
    previousTime=currentTime;
    ledState=!ledState;
    digitalWrite(13, ledState);
    Serial.print(F("LED State : "));Serial.println(ledState);
  }
}

```

type de données

Un type de données (variables Arduino) est une cellule de mémoire avec un nom qui contient une valeur numérique ou alphabétique. Avec les types de données numériques (variables), vous pouvez effectuer toutes les opérations mathématiques: multiplication, division, addition, soustraction, conversion vers un autre type de données, etc. Les variables sont écrites dans un programme sous la forme suivante: <type> <nom> = <valeur>;

Si la valeur d'une variable globale ou locale n'est pas définie, une valeur de 0 est attribuée. Les variables qui ne peuvent pas être modifiées dans le programme après avoir été déclarées sont appelées constantes. Pour définir une constante, ajoutez const devant le type de données de la variable. Le tableau suivant montre les types de données Arduino qui peuvent être utilisés lors de l'écriture d'un programme pour le microcontrôleur.

Type	Taille en mémoire	Valeurs min	Valeurs max
boolean	1 octet (8 bits)	false	true
byte	1 octet (8 bits)	0	255
char	1 octet (8 bits)	-128	127
int	2 octet (16 bits)	-32768	32767
unsigned int	2 octet (16 bits)	0	65535
long	4 octet (32 bits)	-2147483648	2147483647
unsigned long	4 octet (32 bits)	0	4294967295
float, double	4 octet (32 bits)	-3.4028235E+38	3.4028235E+38

Arduino: Boolean ou bool

Un “boolean” est un booléen. Il s’agit d’une donnée ne pouvant prendre que 2 valeurs: true (vrai) ou false (faux). Il est également possible de remplacer true par 1 et false par 0 pour un booléen. Chaque variable booléenne occupe un byte de mémoire.

Arduino Byte

Une donnée byte peut stocker un nombre non signé sur 8 bits, de 0 à 255.

```
byte val = 98; // 98 en décimal  
byte val = B00101111; // 47 en binaire (le B signifie binaire)
```

Arduino Char

Les données Char sont des données pouvant contenir un caractère. C’est un type de donnée qui occupe un octet de mémoire. caractère s’écrit dans une variable char entre guillemets simples comme ceci: ‘A’. Pour écrire plusieurs caractères, ce qu’on appelle aussi une chaîne de caractères (string), on utilisera les guillemets doubles comme ceci: “ABC”.

Arduino Int

Les données int représentent les données de stockage des nombres entiers. Il s’agit du type de donnée principal pour le stockage des nombres. Int stocke une valeur sur 16 bits, soit 2 octets. Cela donne une fourchette de -32 768 à 32 767 (soit une valeur minimale de -215 et une valeur maximale de 215-1).

Arduino Unsigned Int

Les Unsigned Int fonctionnent de la même manière que les Int. La différence ici, c’est que les nombres négatifs ne sont plus pris en compte. Le nombre est toujours encodé sur 2 bytes, soit 16 bits, soit 2 octets. La plage positive est donc plus grande : de 0 à 65 535 (216-1). Les Arduino Due, MKR1000, Zero étant sur des nombres en 32 bits, la plage de ce type de données se situe entre 0 et 4 294 967 295 (232-1).

Arduino Long

Les variables Long peuvent stocker des nombres entiers sur 4 bytes, soit 32 bits : de -2 147 483 648 (-231) à 2 147 483 647 (231+1).

Arduino Unsigned long

Les variables Unsigned long peuvent stocker des nombres non signés entiers sur 32 bits : de 0 à 4 294 967 295 (232-1).

Arduino Float

Le type de données float permet de stocker un nombre à virgule flottante. Les variables float sont souvent utilisées pour approcher les valeurs analogiques et continues car la donnée float possède une plus grande résolution que les nombres entiers.

Arduino Double

Sur Arduino UNO et autres cartes munies de processeur ATMEGA, une donnée double fonctionne exactement de la même manière qu'une donnée float. Elle occupe également 32 bits, sans gain de précision supplémentaire. Par contre sur Arduino Due, MKR1000 ou Zero, la donnée double occupe 64 bits, ce qui offre plus de précision.

Arduino conversion des types de variables (données)

Dans Arduino IDE, une conversion de variable est la conversion d'une valeur de variable à un type différent. Par exemple, vous souhaitez convertir un type de données byte en type de données int dans un programme. Pour ce faire, indiquez le type de variable souhaité entre parenthèses avant la variable à convertir – le résultat renverra une variable avec le nouveau type de données. Voici un exemple de conversion explicite de variables Arduino:

```
byte x = 150;  
int y = (int)x;  
  
float x = 15,4;  
int y = (int)x;
```