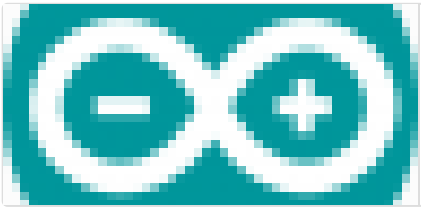
Arduino - Les bases

Arduino



Arduino - Home

Open-source electronic prototyping platform enabling users to create interactive electronic objects.
https://www.arduino.cc/

Arduino est une plateforme de prototypage électronique open-source basée sur du matériel et un logiciel faciles à utiliser. Cela permet aux amateurs, aux étudiants, aux artistes et aux professionnels de créer des projets interactifs en utilisant des microcontrôleurs.

Un Arduino est une carte de développement qui intègre un microcontrôleur, des broches d'entrée/sortie (E/S), et un environnement de programmation pour télécharger et exécuter du code. Les microcontrôleurs utilisés dans les cartes Arduino sont généralement basés sur l'architecture AVR, bien que certaines variantes utilisent des microcontrôleurs ARM.

L' Arduino pour fonctionner a besoin des deux boucles suivante (fonctions):

Boucle Setup

La boucle Setup est la première à être exécuté par le microcontrôleur , elle permet de réaliser des actions qui ne seront nécessaire qu'une seule fois comme le mode des pins .

```c
void setup() {
// put your setup code here, to run once:
}
```

Boucle Loop

Cette boucle est le cœur de votre programme, c'est ici que vous devez mettre le code qui sera exécuté.
À la fin de la boucle, celle si se répète indéfiniment tant que l'Arduino reste sous tension.

```c
void loop() {

// put your main code here, to run repeatedly:
}
```

Variables

Comme dans les autres langages de programmation vous serez amené à utilisé des variables.

Déclaration

Pour déclaré une variable ceci se fait sous la forme **type var_name**

```c
typeDeDonnées nomDeVariable;
```

Voici quelques exemples de déclarations de variables en Arduino :

1. Entier (int) :

```c
int monEntier;
```

2. Flottant (float) :

```c
float monFlottant;
```

3. Caractère (char) :

```c
char monCaractere;
```

4. Booléen (boolean) :

```c
boolean monBool;
```

5. Chaîne de caractères (String) :

```c
String maChaine;
```

Vous pouvez également initialiser une variable lors de sa déclaration en lui attribuant une valeur :

```c
typeDeDonnées nomDeVariable = valeur;
```

Exemple :

```c
int monEntier = 10;
float monFlottant = 3.14;
char monCaractere = 'A';
boolean monBool = true;
String maChaine = "Hello";
```

Type

Le langage Arduino étant un dérivé du C++, il utilise un typage strict qui doit être explicitement spécifié lors de la déclaration d'une variable.
les principaux types sont les suivants:

Type de variable	Description	Plage de valeurs
int	Entier signé	-32,768 à 32,767
unsigned int	Entier non signé	0 à 65,535
long	Entier signé long	-2,147,483,648 à 2,147,483,647
unsigned long	Entier non signé long	0 à 4,294,967,295
byte	Entier non signé sur 8 bits	0 à 255

Type de variable	Description	Plage de valeurs
float	Nombre décimal à virgule flottante	±1.17549e-38 à ±3.40282e+38
double	Nombre décimal à virgule flottante double	±2.22507e-308 à ±1.79769e+308
boolean	Valeur booléenne (vrai/faux)	Vrai (true) ou faux (false)
char	Caractère ASCII	-128 à 127
String	Chaîne de caractères	-
void	Type spécial utilisé pour les fonctions void	-

Les tableaux

Pour déclarer un tableau dans Arduino, vous pouvez utiliser la syntaxe suivante :

```c
type nomTableau[taille];
```

Voici quelques exemples de déclarations de tableaux dans Arduino :

1. Tableau d'entiers :

```c
int monTableau[5];  // Déclare un tableau d'entiers de taille 5
```

2. Tableau de caractères (chaîne de caractères) :

```c
char monMessage[] = "Hello";  // Déclare un tableau de caractères avec une chaîne de caractères
```

3. Tableau de flottants :

```c
float mesDonnees[10];  // Déclare un tableau de flottants de taille 10
```

4. Tableau de valeurs booléennes :

```c
boolean etatLEDs[8];  // Déclare un tableau de valeurs booléennes de taille 8
```

Lors de la déclaration d'un tableau, vous spécifiez le **type** de données des éléments du tableau, suivi du **nom** du tableau, puis entre crochets la **taille** du tableau (c'est-à-dire le nombre d'éléments qu'il peut contenir).

Notez que dans Arduino, les tableaux sont indexés à partir de zéro. Cela signifie que le premier élément d'un tableau est référencé par l'indice 0, le deuxième élément par l'indice 1, et ainsi de suite. Vous pouvez accéder et manipuler les éléments d'un tableau à l'aide de l'indice correspondant. Par exemple, `monTableau[0]` fait référence au premier élément du tableau `monTableau`.

pour remplir un tableau dés la déclaration, la syntax est la suivante:

Pour déclarer un tableau avec des valeurs initiales en Arduino, vous pouvez utiliser la syntaxe suivante :

```c
type nomTableau[] = {valeur1, valeur2, valeur3, ...};
```

Voici quelques exemples de déclarations de tableaux avec des valeurs initiales en Arduino :

1. Tableau d'entiers :

```c
int monTableau[] = {1, 2, 3, 4, 5};  // Déclare un tableau d'entiers avec des valeurs initiales
```

2. Tableau de caractères (chaîne de caractères) :

```c
char monMessage[] = "Hello";  // Déclare un tableau de caractères avec une chaîne de caractères initiale
```

Dalns ces exemples, les tableaux sont déclarés avec des crochets vides `[]` pour permettre à l'Arduino de déterminer automatiquement la taille du tableau en fonction du nombre de valeurs initiales fournies.

Vous pouvez également spécifier la taille du tableau lors de la déclaration en utilisant un nombre entre les crochets `[]`. Par exemple :

```c
int monTableau[5] = {1, 2, 3, 4, 5};  // Déclare un tableau d'entiers de taille 5 avec des valeurs initiales
```

Assurez-vous que le nombre de valeurs initiales correspond à la taille déclarée du tableau pour éviter tout dépassement de mémoire ou toute erreur de compilation.

Tableaux à plusieurs dimensions

Les tableaux peuvent avoir plusieurs dimensions, le plus souvent deux mais nous pouvons en mettre beaucoup plus.

```c
// Déclaration et initialisation d'un tableau à deux dimensions de taille 3x3
int tableau[3][3] = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9}
};
```

exemple d'affichage:

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

  int tableau[3][3] = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
  };

  // Affichage des éléments du tableau
  for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
      Serial.print(tableau[i][j]);
      Serial.print(" ");
    }
    Serial.println();
  }
}

void loop() {
  // Votre code principal
}

//Affiche sur le moniteur série:
// 1 2 3
// 4 5 6
// 7 8 9
```

Variables globale et locale

En Arduino, comme dans la plupart des langages de programmation, il existe deux types de variables : les variables **globales** et les variables **locales**.

	Variables globales	Variables locales
Déclaration	Déclarées en dehors de toutes les fonctions	Déclarées à l'intérieur d'une fonction
Accessibilité	Accessibles depuis n'importe quelle partie du code	Accessibles uniquement à l'intérieur de la fonction
Portée	Tout le programme	La fonction dans laquelle elles sont déclarées

	Variables globales	Variables locales
Durée de vie	Tout au long de l'exécution du programme	Créées lors de l'appel de la fonction et détruites à la fin de celle-ci
Conservation de la valeur	Conserve leur valeur tant qu'elle n'est pas explicitement modifiée	Créée à chaque appel de la fonction avec une nouvelle valeur
Occupation de la mémoire	Occupe de l'espace mémoire pendant toute l'exécution du programme	Occupe de l'espace mémoire uniquement pendant l'exécution de la fonction
Utilisation typique	Stockage de valeurs partagées ou maintien d'un état global du programme	Stockage de valeurs temporaires ou de données spécifiques à une fonction

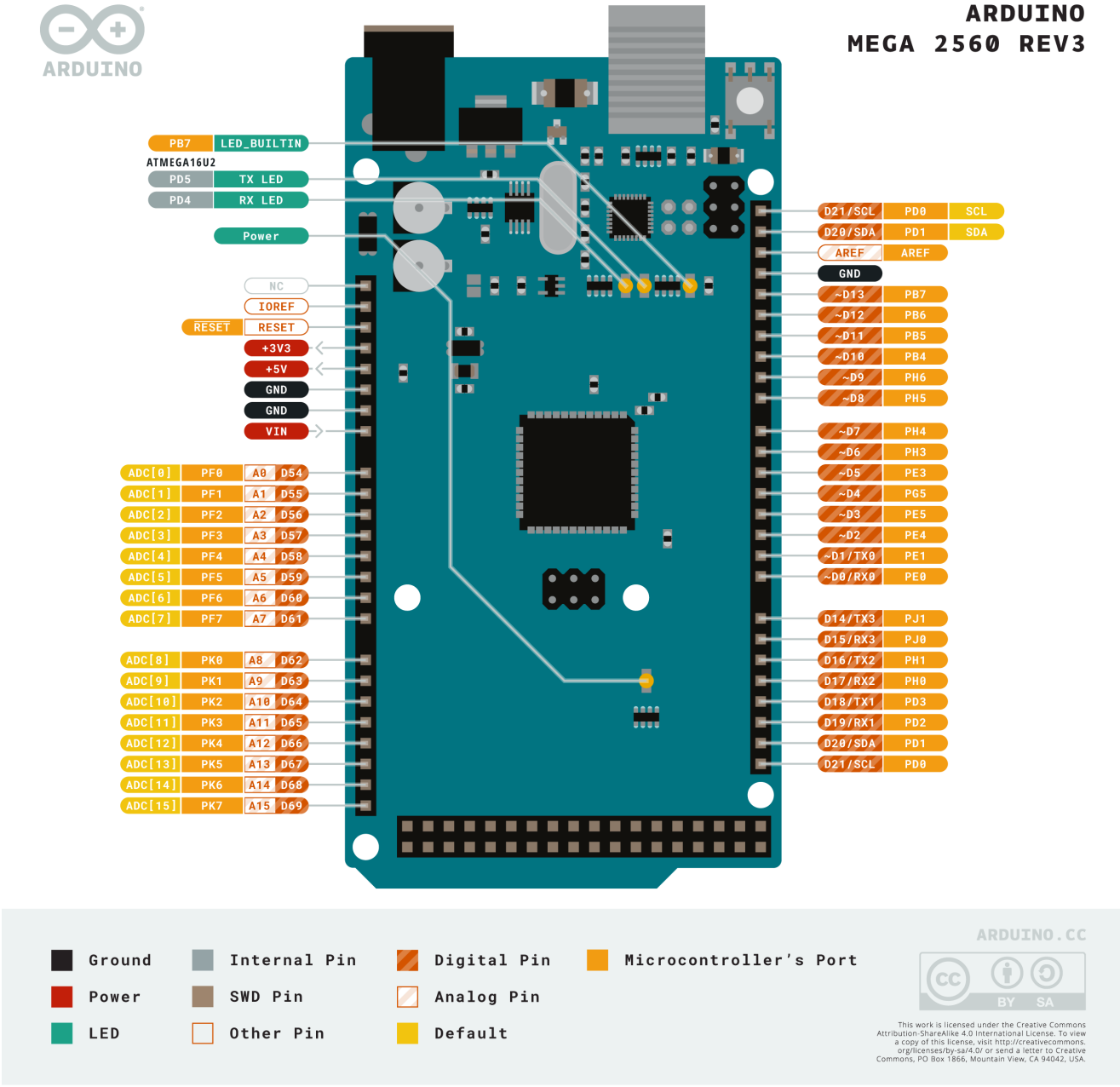
Il est recommandé d'utiliser les variables globales avec prudence, car elles peuvent rendre le code plus complexe et moins modulaire en plus d'utilisé de la mémoire en permanence.

E/S

Pour interagir avec utilisateur au travers de bouton par exemple ou récupérer des donnée provenant de capteurs et agir sur des composant comme des DELs ou des écrans, l'Arduino possède des broches qui permettent diverses connections, ce sont des **entrées/sorties**.

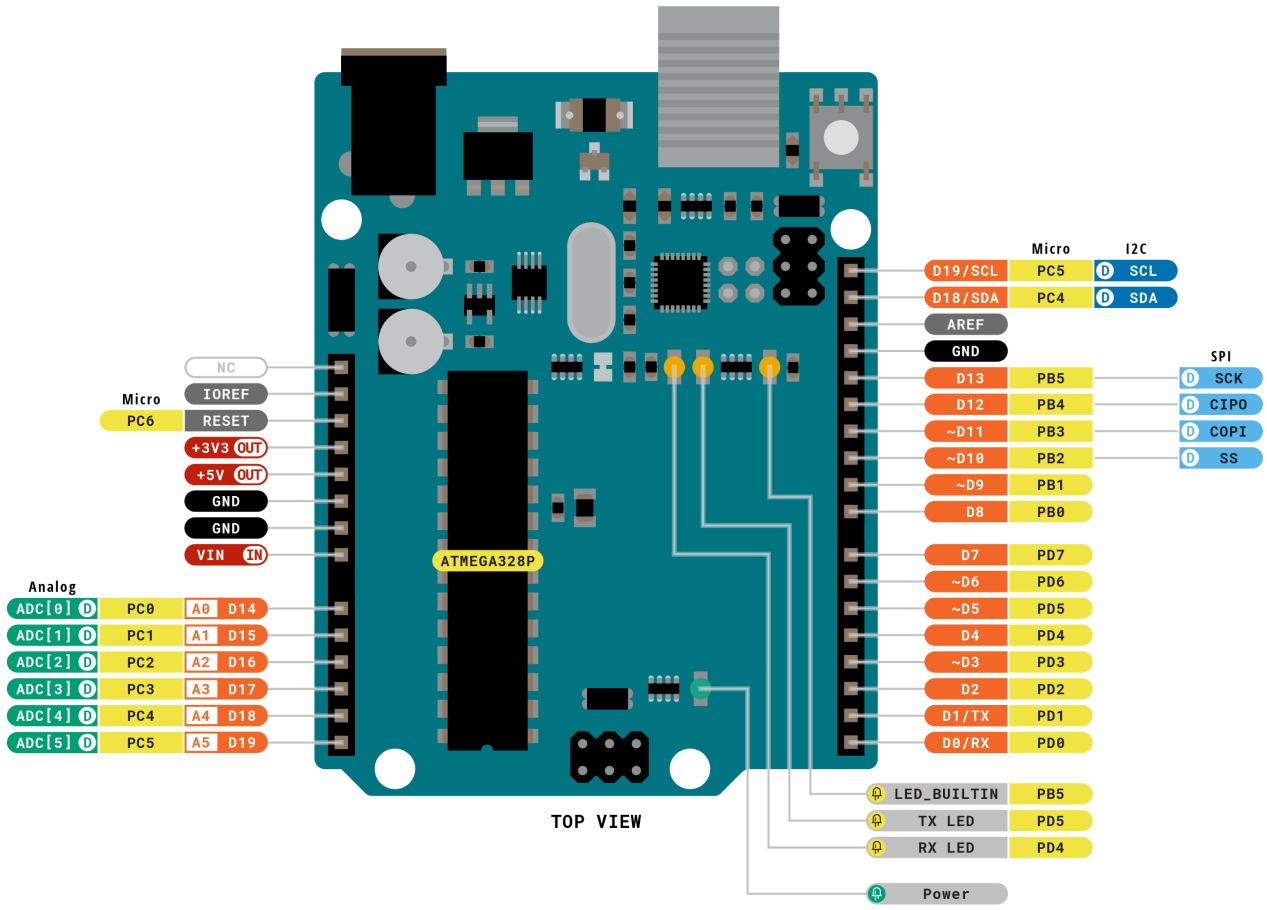
Des exemples d'utilisation peuvent être trouvés sur la page: Arduino - Les composants

Broches Mega:



source: https://docs.arduino.cc/hardware/uno-rev3

Borches UNO:



source: https://docs.arduino.cc/

Mode de fonctionnement

Pour que le microcontrôleur puisse savoir si une pin est une entrée ou une sortie, vous devez préalablement l'indiquer dans le *setup* sous la forme : **pinMode(pin, mode)** par exemple:

```c
void setup() {
pinMode(1,OUTPUT); //la pin 1 est un sortie
pinMode(2,INPUT); //la pin 2 est une entrée
}
```

Lecture et écriture

pour lire ou écrire une valeur sur les pins vous aurez besoin de certaines fonctions.

E/S numérique

Toutes les pins avec marqué **Dx** peuvent être utilisées aussi n=bien en entrée que en sortie

Fonction	Description
digitalWrite(pin, value)	Écrit une valeur (HIGH ou LOW) sur une broche numérique configurée en mode OUTPUT.
digitalRead(pin)	Lit la valeur (HIGH ou LOW) d'une broche numérique configurée en mode INPUT.

HIGH et LOW peuvent être remplacés par 1 et 0

exemple avec le clignotement d'une DEL:

```c
const int ledPin = 13;  // Broche numérique utilisée pour la LED

void setup() {
  pinMode(ledPin, OUTPUT);  // Configure la broche de la LED en sortie
}

void loop() {
  digitalWrite(ledPin, HIGH);  // Allume la LED en mettant la broche à l'état HIGH (5V)
  delay(1000);  // Attend 1 seconde

  digitalWrite(ledPin, LOW);  // Éteint la LED en mettant la broche à l'état LOW (0V)
  delay(1000);  // Attend 1 seconde
}
```

exemple de lecture:

```c
const int boutonPin = 2;  // Broche numérique utilisée pour la lecture du bouton

void setup() {
  pinMode(boutonPin, INPUT);  // Configure la broche comme une entrée
  Serial.begin(9600);  // Initialise la communication série
}

void loop() {
  int etatBouton = digitalRead(boutonPin);  // Lit l'état du bouton (HIGH ou LOW)

  if (etatBouton == HIGH) {
    Serial.println("Bouton appuyé");
  } else {
    Serial.println("Bouton relâché");
  }

  delay(100);  // Délai pour éviter une lecture trop rapide
}
```

E/S analogique

Pour les pins analogiques cela se complique un peu, seule les pin **Ax** peuvent être utilisées en **entrée**, pour les **sorties** seulement les pins **PWM** peuvent l'être, elle sont reconnaissable par le symbole **~**

Fonction	Description
analogRead(pin)	Lit la valeur analogique (de 0 à 1023) d'une broche analogique.
analogWrite(pin, value)	Écrit une valeur analogique (de 0 à 255) sur une broche PWM (Pulse Width Modulation).

exemple:

```c
const int pinCapteur = A0;  // Définit la broche analogique à laquelle est connecté le capteur

void setup() {
  Serial.begin(9600);  // Initialise la communication série avec une vitesse de 9600 bauds
}

void loop() {
  int valeurCapteur = analogRead(pinCapteur);  // Lit la valeur du capteur analogique

  // Affiche la valeur du capteur dans le moniteur série
  Serial.print("Valeur capteur : ");
  Serial.println(valeurCapteur);

  delay(1000);  // Attend une seconde avant la prochaine lecture
}
```

les opérateurs

1. Opérateurs arithmétiques :

- `+` : Addition
- `-` : Soustraction
- `*` : Multiplication
- `/` : Division
- `%` : Modulo (reste de la division)

2. Opérateurs d'assignation :

- `=` : Assignation simple
- `+=` : Addition et assignation
- `-=` : Soustraction et assignation
- `*=` : Multiplication et assignation
- `/=` : Division et assignation
- `%=` : Modulo et assignation

3. Opérateurs de comparaison :

- `==` : Égal à
- `!=` : Différent de
- `<` : Inférieur à
- `>` : Supérieur à
- `<=` : Inférieur ou égal à
- `>=` : Supérieur ou égal à

4. Opérateurs logiques :

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

5. Opérateurs d'incrémentation et de décrémentation :

- `++` : Incrémentation de 1
- `--` : Décrémentation de 1

6. Opérateurs bit à bit :

- `&` : ET bit à bit
- `|` : OU bit à bit
- `^` : OU exclusif bit à bit
- `~` : Complément bit à bit (inverse les bits)
- `<<` : Décalage à gauche
- `>>` : Décalage à droite

7. Autres opérateurs :

- `sizeof()` : Renvoie la taille d'un type ou d'une variable en octets
- `? :` : Opérateur ternaire (condition ? valeurSiVrai : valeurSiFaux)

Il est important de noter que les opérateurs en Arduino sont similaires à ceux utilisés dans d'autres langages de programmation tels que le C/C++. Cependant, la liste ci-dessus couvre les opérateurs de base que vous rencontrerez fréquemment lors de la programmation en Arduino.

Catégorie	Opérateurs
Opérateurs arithmétiques	`+`, `-`, `*`, `/`, `%`
Opérateurs d'assignation	`=`, `+=`, `-=`, `*=`, `/=`, `%=`
Opérateurs de comparaison	`==`, `!=`, `<`, `>`, `<=`, `>=`
Opérateurs logiques	`&&`, `

Catégorie	Opérateurs
Incrémentation / Décrémentation	`++`, `--`
Opérateurs bit à bit	`&`, `
Autres opérateurs	`sizeof()`, `? :`

Les conditions

Voici les structures conditionnelles les plus couramment utilisées dans la programmation Arduino :

1. Structure `if` :

La structure `if` permet d'exécuter un bloc de code si une condition est vraie.

```c
if (condition) {
  // Code à exécuter si la condition est vraie
}
```

2. Structure `if...else` :

La structure `if...else` permet d'exécuter un bloc de code si une condition est vraie, et un autre bloc de code si la condition est fausse.

```c
if (condition) {
  // Code à exécuter si la condition est vraie
} else {
  // Code à exécuter si la condition est fausse
}
```

3. Structure `if...else if...else` :

La structure `if...else if...else` permet d'évaluer plusieurs conditions successives et d'exécuter le bloc de code correspondant à la première condition vraie. Si aucune condition n'est vraie, le bloc de code dans la section `else` est exécuté.

```c
if (condition1) {
  // Code à exécuter si la condition1 est vraie
} else if (condition2) {
  // Code à exécuter si la condition2 est vraie
} else {
  // Code à exécuter si aucune condition n'est vraie
}
```

4. Structure `switch...case` :

La structure `switch...case` permet d'évaluer une expression et d'exécuter le bloc de code correspondant à la valeur de cette expression.

```c
switch (expression) {
  case valeur1:
    // Code à exécuter si l'expression a la valeur1
    break;
  case valeur2:
    // Code à exécuter si l'expression a la valeur2
    break;
  default:
    // Code à exécuter si aucune des valeurs ne correspond
    break;
}
```

Ces structures conditionnelles vous permettent de prendre des décisions et d'exécuter différents blocs de code en fonction des conditions spécifiées, ce qui est essentiel pour contrôler le comportement de votre programme Arduino.

Les Boucles

Voici les boucles les plus couramment utilisées dans la programmation Arduino :

1. Boucle `for` :

La boucle `for` permet d'exécuter un bloc de code un certain nombre de fois, en utilisant une variable d'itération.

```c
for (initialisation; condition; incrémentation) {
  // Code à exécuter
}
```

Exemple :

```c
for (int i = 0; i < 10; i++) {
  // Code à exécuter 10 fois
}
```

2. Boucle `while` :

La boucle `while` permet d'exécuter un bloc de code tant qu'une condition spécifiée est vraie.

```c
while (condition) {
  // Code à exécuter
}
```

Exemple :

```c
int i = 0;
while (i < 10) {
  // Code à exécuter tant que i est inférieur à 10
  i++;
}
```

3. Boucle `do...while` :

La boucle `do...while` est similaire à la boucle `while`, mais elle exécute le bloc de code au moins une fois, puis répète l'exécution tant qu'une condition spécifiée est vraie.

```c
do {
  // Code à exécuter
} while (condition);
```

Exemple :

```c
int i = 0;
do {
  // Code à exécuter au moins une fois, puis tant que i est inférieur à 10
  i++;
} while (i < 10);
```

Ces boucles vous permettent d'itérer et de répéter des sections de code, ce qui est essentiel pour effectuer des opérations répétitives ou contrôler le flux de votre programme Arduino.

Les fonctions

Une Fonctions est un morceau de code qui exécute une tâche précise quand elle est **appelée**, avant de pouvoir l'utiliser elle doit être **déclarée**.

Certaine existe de base (ou sont présente dans des librairies), par exemple la fonction ***random*** (génère un nombre aléatoire), est déclaré dans le code source arduino, il suffit donc simplement de l'appeler.

Pour **déclarer** une fonction en Arduino, vous pouvez suivre la syntaxe suivante :

```c
typeDeRetour nomDeFonction(paramètres) {
  // Corps de la fonction
}
```

- `typeDeRetour` : spécifie le type de données renvoyé par la fonction. Cela peut être `void` si la fonction ne renvoie aucune valeur, ou un autre type de données (par exemple `int`, `float`, `void`, etc.) si la fonction renvoie une valeur.
- `nomDeFonction` : le nom que vous donnez à votre fonction. Choisissez un nom significatif qui décrit l'action que la fonction effectue. Celui-ci servira aussi à l'appeler

- `paramètres` : les variables qui sont passées à la fonction pour être utilisées à l'intérieur de celle-ci. Les paramètres sont optionnels et peuvent être de n'importe quel type de données. Si vous avez plusieurs paramètres, vous pouvez les séparer par des virgules.
- `corps de la fonction` : c'est l'ensemble des instructions qui seront exécutées lorsque la fonction est appelée. Les instructions à l'intérieur des accolades `{}` constituent le corps de la fonction.

Voici un exemple de **déclaration** d'une fonction en Arduino :

```c
// Déclaration d'une fonction sans paramètre et sans type de retour
void direBonjour() {
  Serial.println("Bonjour !");
}

// Déclaration d'une fonction avec un paramètre et un type de retour
int additionner(int a, int b) {
  int resultat = a + b;
  return resultat;
}
```

Dans cet exemple, la fonction `direBonjour()` ne prend aucun paramètre et ne renvoie aucune valeur. Elle affiche simplement "Bonjour !" via la communication série.

La fonction `additionner()` prend deux paramètres de type `int` et renvoie leur somme.

Une fois que vous avez déclaré une fonction, vous pouvez l'appeler dans votre code en utilisant son nom suivi de parenthèses. Par exemple :

```c
direBonjour();  // Appel de la fonction direBonjour()

int resultatAddition = additionner(5, 3);  // Appel de la fonction additionner() avec les paramètres 5 et 3
```

Liaison série

La liaison série est une fonctionnalité très importante car elle vous permet d'afficher des informations pour comprendre ce que fait votre arduino pour du débogage.

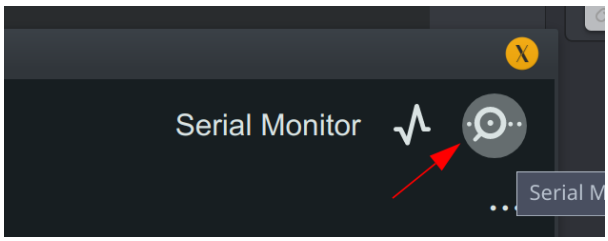
Dans votre code Arduino, utilisez la fonction `Serial.begin()` pour initialiser la communication série avec une vitesse de transmission spécifiée. Par exemple :

```c
void setup() {
  Serial.begin(9600);  // Initialise la communication série avec une vitesse de 9600 bauds
}
```

5. Utilisez les fonctions `Serial.print()` ou `Serial.println()` pour envoyer des données vers le moniteur série. Par exemple :

```c
void loop() {
  int valeurCapteur = analogRead(A0);  // Lit la valeur d'un capteur analogique
  Serial.print("Valeur capteur : ");
  Serial.println(valeurCapteur);  // Affiche la valeur du capteur dans le moniteur série
  delay(1000);  // Attend une seconde
}
```

Depuis l'IDE Arduino vous pouvez lire les données depuis le moniteur



La gestion du temps

En Arduino, vous pouvez utiliser plusieurs fonctions et bibliothèques pour gérer le temps. Voici quelques-unes des fonctions couramment utilisées pour la gestion du temps :

- `delay(ms)` :
  La fonction `delay(ms)` suspend l'exécution du programme pendant le nombre de millisecondes spécifié.

> ⚠ **Attention**
>
> L'Arduino attend à cette endroit et ne fera rien d'autre jusqu'à ce que le temps soit écoulé,
> si vous avez besoin de continué l'éxécution de certainne partie du code, c'est la fonction **millis()** qui devra être employée

Exemple d'utilisation :

```c
delay(1000);  // Attend une seconde (1000 millisecondes)
```

- `millis()` :
  La fonction `millis()` renvoie le nombre de millisecondes écoulées depuis le démarrage de votre programme Arduino. Elle est souvent utilisée pour mesurer des intervalles de temps ou pour créer des délais sans bloquer l'exécution du code.

  Exemple d'utilisation :

  ```c
  unsigned long tempsActuel = millis();
  ```

exemple:

```c
void setup() {
  Serial.begin(9600);  // Initialise la communication série
}

void loop() {
  static unsigned long tempsPrecedent = 0;  // Variable pour stocker le temps précédent
  unsigned long tempsActuel = millis();  // Récupère le temps actuel

  if (tempsActuel - tempsPrecedent >= 1000) {
    Serial.println("1 seconde écoulée");
    tempsPrecedent = tempsActuel;  // Met à jour le temps précédent
  }

  // Autres actions à effectuer dans la boucle loop()
}
```

- Bibliothèque `Time` :
  La bibliothèque `Time` fournit des fonctions et des structures pour manipuler les dates et les heures. Elle permet de définir des horloges, des minuteries et de réaliser des opérations sur les dates et les heures.

  Pour utiliser la bibliothèque `Time`, vous devez l'importer dans votre programme Arduino :

  ```c
  #include <TimeLib.h>
  ```

Exemple d'utilisation :

```c
#include <TimeLib.h>

void setup() {
  // Configurez votre horloge ici
  setTime(12, 34, 0, 1, 1, 2023);  // Définit l'heure à 12:34:00 le 1er janvier 2023
}

void loop() {
  // Lisez l'heure actuelle
  int heure = hour();
  int minute = minute();
  int seconde = second();

  // Faites quelque chose en fonction de l'heure actuelle
  // ...

  delay(1000);  // Attendez une seconde
}
```

Ces fonctions et bibliothèques vous permettent de mesurer le temps écoulé, de créer des délais, de manipuler des horloges et de réaliser des opérations sur les dates et les heures dans vos programmes Arduino.