REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE LA FORMATION ET DE L'ENSEIGNEMENT PROFESSIONNELS

IFEP SBA

PROGRAMMATION DE ARDOUINO

Présenté par :

HALAILI Mohamed

LE LANGAGE ARDOUINO

1/ Syntaxe du langage Arduino:

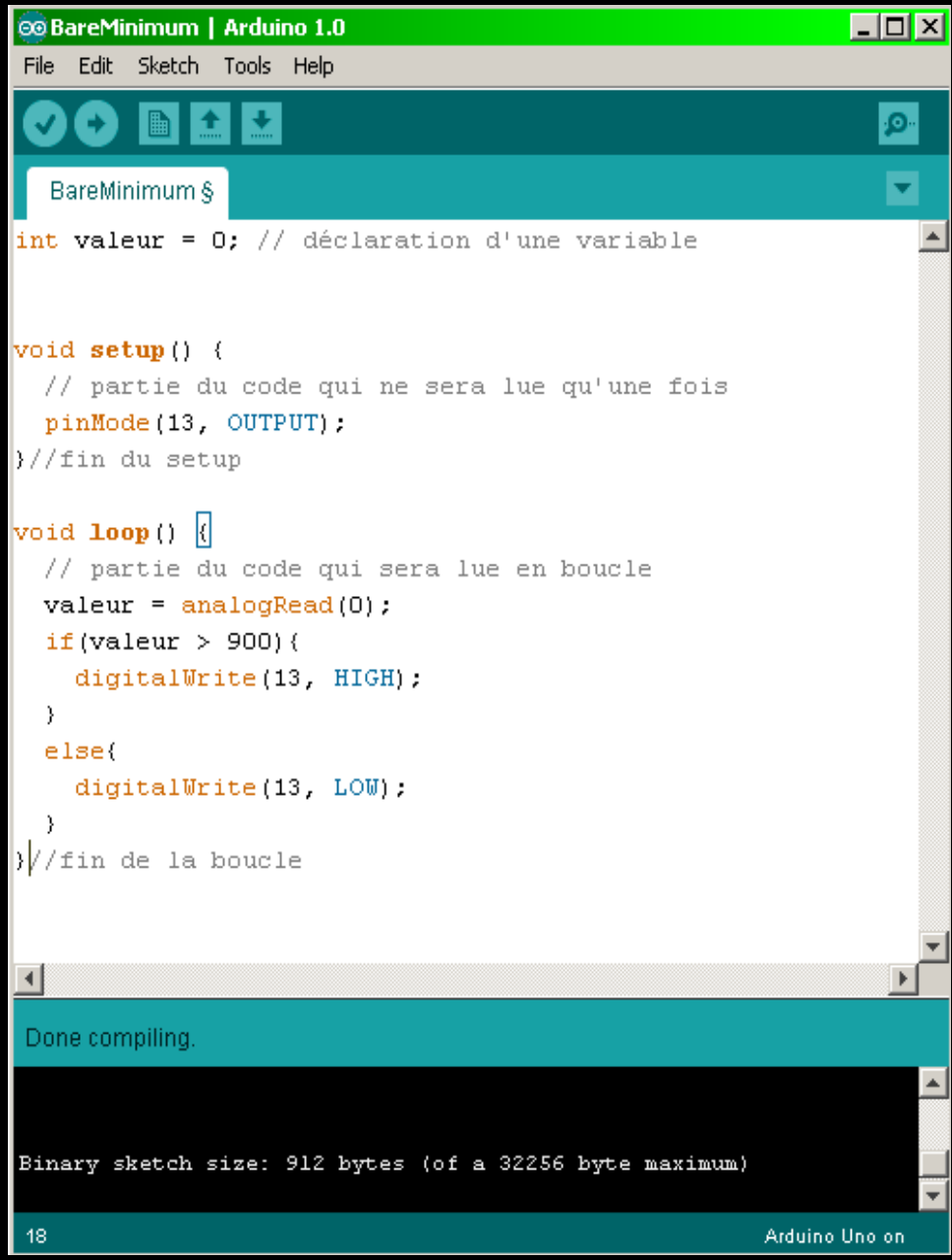
Le langage Arduino est très proche du C et du C++.

La syntaxe d'un langage de programmation est l'ensemble des règles d'écritures liées à ce langage. On va donc voir dans cette partie les règles qui régissent l'écriture du langage Arduino.

Syntaxe du langage Arduino

Le programme est lu par le micro-contrôleur de haut vers le bas.

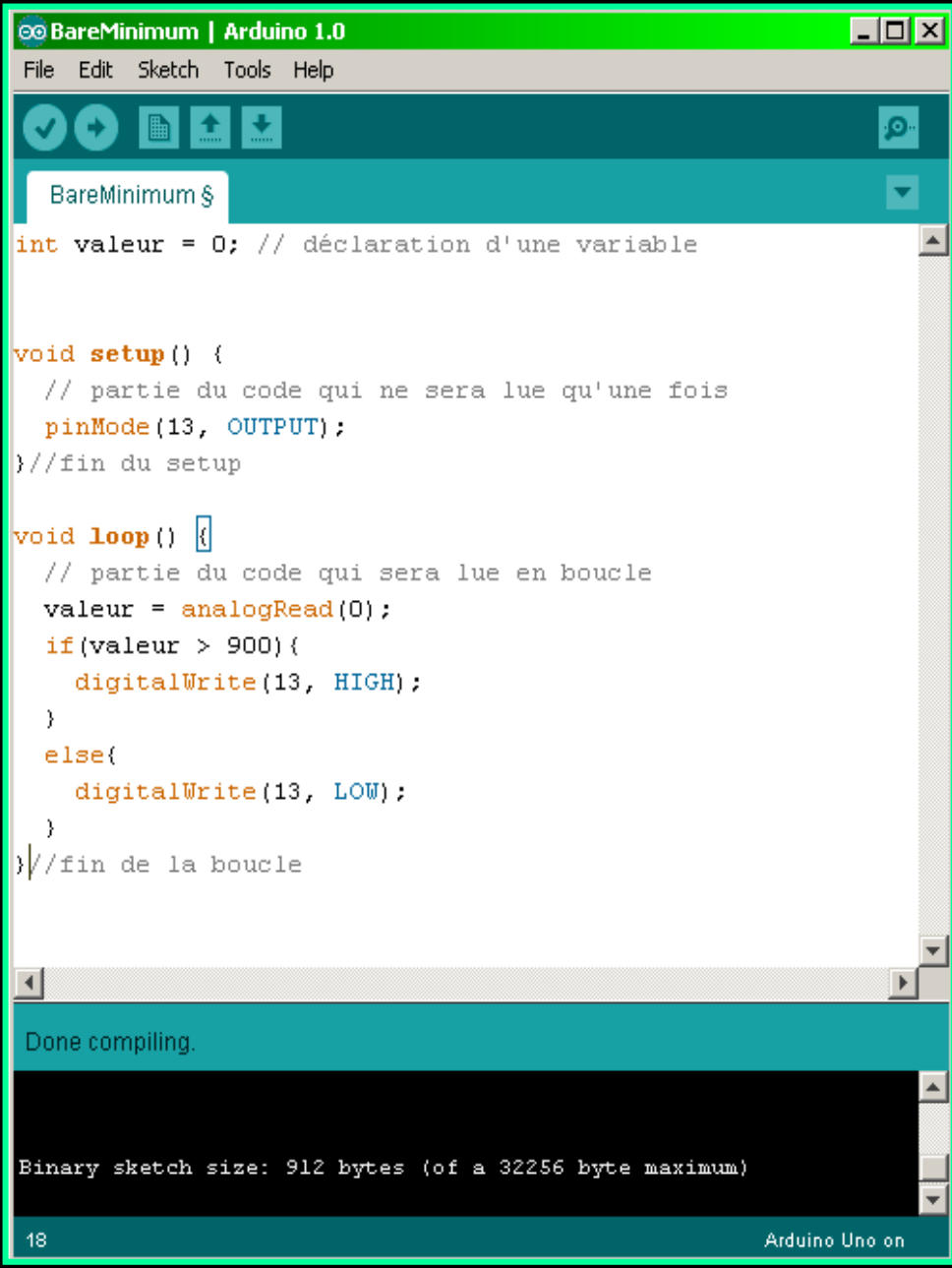
Une variable doit être déclarée avant d'être utilisée par une fonction.



Syntaxe du langage Arduino

La structure minimale d'un programme Ardouino est constituée de :

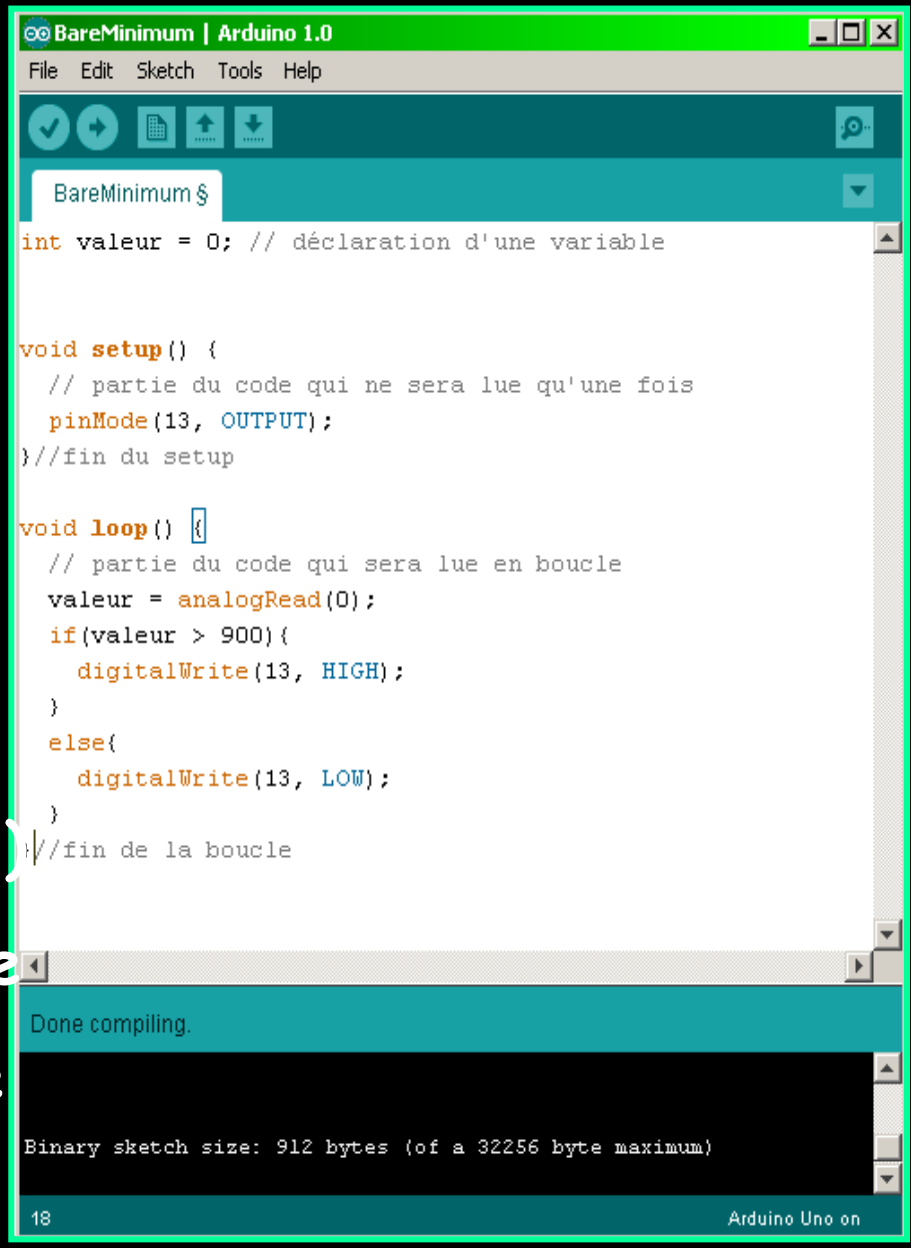
En tête : déclaration des variables, des constantes, indication de l'utilisation de bibliothèques etc...



Syntaxe du langage Arduino

void setup (= initialisation) : cette partie n'est lue qu'une seule fois, elle comprend les fonctions devant être réalisées au démarrage (utilisation des broches en entrées ou en sortie, mise en marche du midi, du port série de l' I2C etc.....)

Void loop (boucle) : cette partie est lue en boucle ! C'est ici que les fonctions sont réalisées.



Les variables

Une variable est un nombre . Ce nombre est stocké dans un espace de la mémoire vive (RAM) du microcontrôleur. C'est comme un compartiment dont la taille n'est adéquate que pour un seul type d'information. Elle est caractérisée par un nom qui permet d'y accéder facilement.

Les variables

Ce nombre a la particularité de changer de valeur.

Le symbole "=>" signifiant : "est contenu dans..."

Le nom de variable accepte quasiment tous les caractères sauf :

- Le point(.)
- La virgule(,)
- Les accents (é,à,ç,è)

Les variables

Voilà les types de variables les plus répandus :

Type	Quel nombre il stocke ?	Valeurs max du nombre stocké	Nombre sur x bits	Nombre d'octets
Int	entier	-32 768 à +32 767	16	2
Long	entier	-2 147 483 648 à +2 147 483 647	32	4
Char	entier	-128 à +127	8	1
Float	décimale	$-3{,}4\times10^{38}$ à $+3{,}4\times10^{38}$	32	4
double	décimale	$-3{,}4\times10^{38}$ à $+3{,}4\times10^{38}$	32	4

Par exemple, si notre variable "x" ne prend que des valeurs décimales, on utilisera les types int , long , ou char . Si maintenant la variable "x" ne dépasse pas la valeur 64 ou 87, alors on utilisera le type char .

Les variables

Si à présent notre variable "x" ne prend jamais une valeur négative (-20, -78, ...), alors on utilisera un type non-signé. C'est à dire, dans notre cas, un char dont la valeur n'est plus de -128 à +127, mais de 0 à 255.

Type	Quel nombre il stocke ?	Valeurs max du nombre stocké	Nombre sur x bits	Nombre d'octets
unsigned char	Entier positif	0 à 255	8	1
unsigned int	Entier positif	0 à 65 535	16	2
unsigned long	Entier positif	0 à 4 294 967 295	32	4

Une des particularités du langage Arduino est qu'il accepte un nombre plus important de types de variables. Voir le tableau suivant :

Type	Quel nombre il stocke ?	Valeurs max du nombre stocké	Nombre sur x bits	Nombre d'octets
byte	Entier positif	0 à 255	8	1
word	Entier positif	0 à 65 535	16	2
boolean	Entier positif	0 à 1	1	1

Les variables booléennes

Les variables **booléennes** sont des variables qui ne peuvent prendre que deux valeurs : ou **VRAI** ou **FAUX**.
Elles sont utilisées notamment dans les boucles et les conditions. Une variable booléenne peut être définie de plusieurs manières :

```
boolean variable = FALSE; //variable est fausse car elle vaut FALSE, du terme anglais "faux«

boolean variable = TRUE; //variable est vraie car elle vaut TRUE, du terme anglais "vrai"
```

```
int variable = 0 ; //variable est fausse car elle vaut 0
int variable = 1 ; //variable est vraie car elle vaut 1
int variable = 42 ; //variable est vraie car sa valeur est différente de 0
```

ou plus particulièrement dans le langage Arduino

```
int variable =LOW; //variable est à l'état logique bas (= traduction de "low"), donc 0
int variable =HIGH;//variable est à l'état logique haut (= traduction de "high"),donc1
```

Les opérations simples

L'addition

```
int x = 0 ;        //définition de la variable x
x = 12 + 3 ;       //on change la valeur de x par une opération simple et x vaut maintenant 12+3=15
```

Exemple:

```
int x = 38 ; //définition de la variable x et assignation à la valeur 38
int y = 10 ;
int z = 0 ;
//faisons une addition avec un nombre choisi au hasard
z = x + y; // on a donc z = 38 + 10 = 48
```

Le principe est le même pour la soustraction, la multiplication et la division.

Les opérations simples

Le modulo

Cette opération permet d'obtenir le reste d'une division.

```
18 % 6 // le reste de l'opération est 0, car il y a 3*6 dans 18 donc 18 - 18 = 0
18 % 5 // le reste de l'opération est 3, car il y a 3*5 dans 18 donc 18 - 15 = 3
```

L'incrémentation et la décrémentation

```
var = 0 ;
var++ ; //c'est cette ligne de code qui nous intéresse qui revient à écrire var=var+1
var +=6 ; // ici var=var+6
var-- ; // ici var=var-1
var -=6; // ici var=var-6
var /=5 ; // ici var=var/5
var %=6 ; // ici var=var % 6
```

Les conditions

C'est un choix que l'on fait entre plusieurs propositions. En informatique, les conditions servent à tester des variables.

Opérateurs logiques de comparaison

Symbole	A quoi il sert	Signifiaction
==	Ce symbole, composé de deux égales, permet de tester l'égalité entre deux variables	... est égale à ...
<	Celui-ci teste l'infériorité d'une variable par rapport à une autre	...est inférieur à...
>	Là c'est la supériorité d'une variable par rapport à une autre	...est supérieur à...
<=	teste l'infériorité ou l'égalité d'une variable par rapport à une autre	...est inférieur ou égale à...
>=	teste la supériorité ou l'égalité d'une variable par rapport à une autre	...est supérieur ou égal à...
!=	teste la différence entre deux variables	...est différent de...

L'instruction if

L'instruction if ("si" en français), utilisée avec un opérateur logique de comparaison, permet de tester si une condition est vraie, par exemple si la mesure d'une entrée analogique est bien supérieure à une certaine valeur.

```
if (uneVariable > 50)
{
// faire quelque chose
}
```

Dans cet exemple, le programme va tester si la variable uneVariable est supérieure à 50. Si c'est le cas, le programme va réaliser une action particulière. Autrement dit, si l'état du test entre les parenthèses est vrai, les instructions comprises entre les accolades sont exécutées. Sinon, le programme se poursuit sans exécuter ces instructions.

L'instruction if

Les accolades peuvent être omises après une instruction if. Dans ce cas, la suite de la ligne (qui se termine par un point-virgule) devient la seule instruction de la condition. Tous les exemples suivants sont corrects :

```
if (x > 120) digitalWrite (LEDpin, HIGH);

if (x > 120)
digitalWrite (LEDpin, HIGH);

if (x > 120) { digitalWrite (LEDpin, HIGH); } // toutes ces formes sont correctes
```

Prenez garde à ne pas utiliser accidentellement le signe = unique (par exemple if (x=10)). Le signe égal unique est l'opérateur d'attribution d'une valeur, et fixe la valeur de x à 10

L'instruction if/else

L'instruction if/else (si/sinon en français) permet un meilleur contrôle du déroulement du programme que la simple instruction if, en permettant de grouper plusieurs tests ensemble. Par exemple, une entrée analogique peut-être testée et une action réalisée si l'entrée est inférieure à 500, et une autre action réalisée si l'entrée est supérieure ou égale à 500. Le code ressemblera à cela :

```
if (brocheCinqEntree < 500)
{
 // action A
}
else
{
// action B
}
```

L'instruction if/else

```
if (brocheCinqEntree < 500)
{
 // action A si la condition est vraie
}
else
{
// action B si la condition est fausse
}
```

```
if (boutonPoussoir1 = 1)
{
// faire l'action A si la condition1 est vraie
}
else if (boutonPoussoir2 = 1)
{
// faire l'action B si la condition1 est Fausse et la si la condition2 est vraie
} else
{
// faire l'action C si les 2 conditions sont fausses
}
```

L'instruction if/else

Exemple 01

```
if (var1 > var2) //Contrôle de la condition d'exécution.
{
  //Actions programmées.
  Serial.println("if => var1 > var2");//Affichage d'un message.
  Serial.println();//Saut de ligne.
}//fin de if.
```

Dans l'exemple ci-dessus. Si la condition d'exécution est **vraie** "true", soit **var1 supérieur à var2** le programme exécute les instructions pour afficher le message présent dans le bloc { }. Lorsque la condition d'exécution sera **fausse** "false" soit **var1<var2** le bloc d'instructions ne sera pas exécuté.

L'instruction if/else

Exemple 02

```
void setup(){
        int a = 10;
        int b = 10;
        Serial.begin(9600);
}
void loop(){
        if(a == b){
           Serial.print("a est egale à b");
   }
       if(a < b){
          Serial.print("a est inférieure à b");
   } else
          {
          Serial.print("a est superieure à b");
           }
     }
```

Les Instructions conditionnelles : le if … else

TP N°05

Pour prendre un exemple concret, supposons que vous vouliez que l'Arduino signale sur une buzzer la présence d'un train dans une gare cachée. Les constituants de ce système sont :

1. Un Arduino ;
2. Un capteur qui va donner une information de nature tout ou rien : le train est présent ou il ne l'est pas ;(Capteur infrarouge)
3. La buzzer qui permet de signaler la présence du train.

Le capteur fournit la condition : le train est présent. La série d'instructions à exécuter correspond à on allume le Buzzer.

Les Instructions conditionnelles : le if … else

L'instruction permettant d'allumer la DEL si le train est présent est le if ... else, « si … sinon » en français. La syntaxe générale est la suivante :

```
1. if (condition) {
2.     // instruction(s) exécutée(s) si la condition est vraie
3. }
4. else {
5.     // instruction(s) exécutée(s) si la condition est fausse
6. }
```

Les Instructions conditionnelles : le if … else

Si il n'est pas nécessaire d'exécuter une série d'instruction si la condition est fausse, il suffit d'omettre la partie else comme ceci :

```
1. if (condition) {
2.     // instruction(s) exécutée(s) si la condition est vraie
3. }
```

Pour revenir à notre système, supposons que le capteur soit connecté à la broche 2 de l'Arduino et qu'il fournisse HIGH, c'est à dire 5V, si le train est présent et LOW, c'est à dire 0V, si il n'est pas présent. La condition pour allumer le buzzer est donc que le capteur fournisse HIGH.

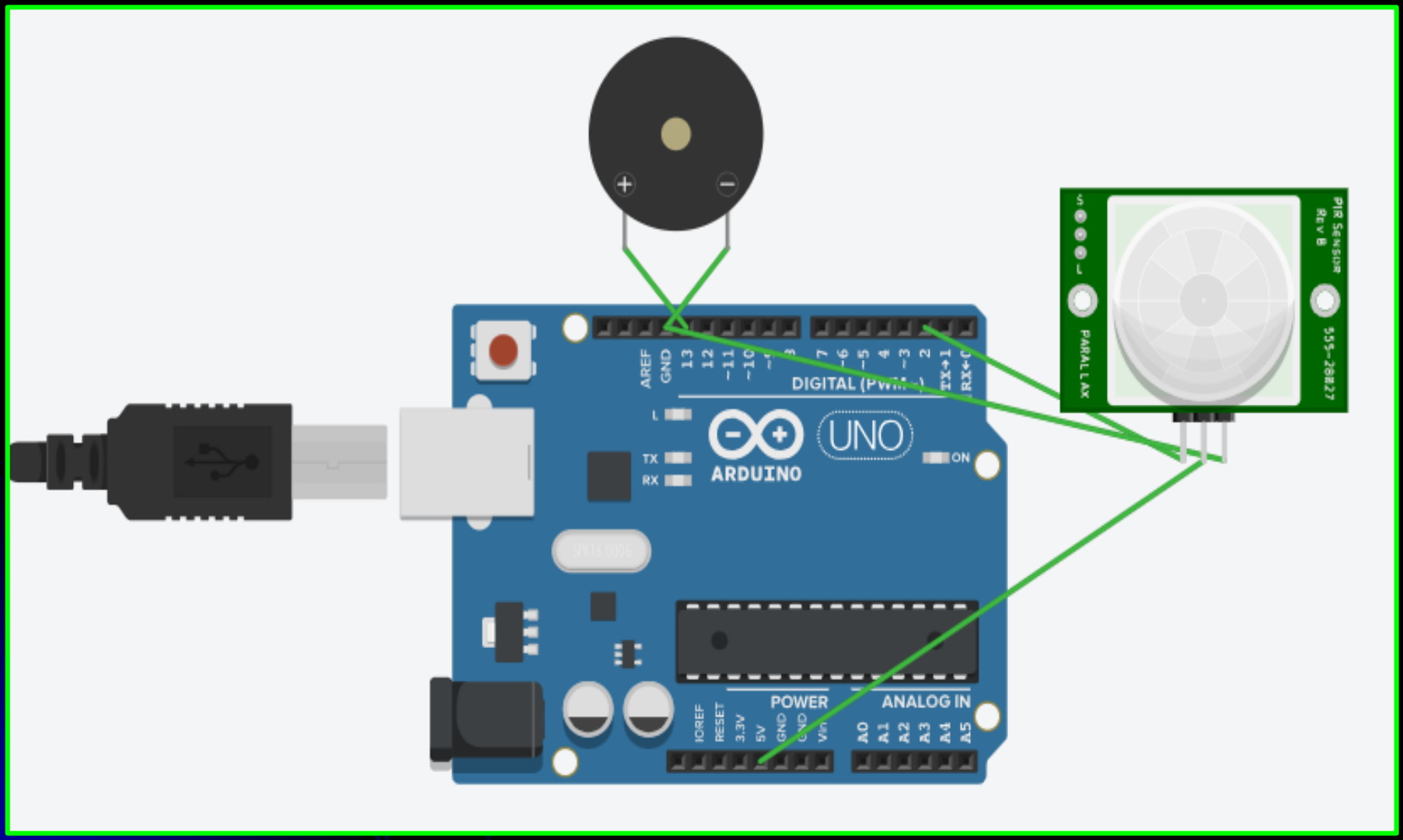
Les Instructions conditionnelles : le if … else

```
void setup() {
    pinMode(2, INPUT); // on veut lire le capteur
    pinMode(13, OUTPUT);  // on veut piloter le Buzzer
}

void loop() {
    if (digitalRead(2 == HIGH) { // si un train est d'étecté
        digitalWrite(13, HIGH);   // allume le Buzzer
    }
    else { // sinon
        digitalWrite(13, LOW);   // Buzzer éteint
    }
}
```

```
const byte pinCapteur = 2; // le capteur est connecté sur la broche 2
const byte pinBuzzer = 13;    // on utilise le Buzzer de la carte sur la broche 13

void setup() {
 pinMode(pinCapteur, INPUT); // on veut lire le capteur
  pinMode(pinBuzzer, OUTPUT);  // on veut piloter la Buzzer
}
void loop() {
  if (digitalRead(pinCapteur) == HIGH) { // si un train est détecté
        digitalWrite(pinBuzzer, HIGH);   // allume la DEL
    }
    else { // sinon
        digitalWrite(pinBuzzer, LOW);   // Buzzer éteint
    }
}
```

L'instruction if/else

TP : Détecteur de train



if imbriqués

TP : Détecteur de train

```
const byte pinCapteur = 2; // le capteur est connecté sur la broche 2
const byte pinDEL = 13;    // on utilise la DEL de la carte sur la broche 13
void setup() {
   pinMode(pinCapteur, INPUT); // on veut lire le capteur
   pinMode(pinBuzzer, OUTPUT);  // on veut piloter la DEL
}
```

if imbriqués

TP : Détecteur de train

```
void loop() {
    if (digitalRead(pinCapteur) == HIGH) {
        // si un train est détecté
        // on attend 10 ms avant de vérifier que
        // ce n'est pas une détection erronée
        delay(10);
        if (digitalRead(pinCapteur) == HIGH) {
            // le train est toujours là, allume la DEL
            digitalWrite(pinDEL, HIGH);
        }
        else {
            // sinon, éteint la DEL
            digitalWrite(pinDEL, LOW);
        }
    }
    else {
        // sinon, éteint la DEL
        digitalWrite(pinDEL, LOW);
    }
}
```

L'instruction For

La structure for (pour) est utilisée afin de répéter une série d'instructions (comprise entre les accolades) le nombre de fois voulu. Un compteur à incrémenter est souvent utilisé afin de sortir de la boucle une fois le nombre de tour fait. Cette structure est composée de trois parties séparées par un point-virgule (;) dans l'entête de la boucle :

```
For (initialisation; condition; expression d'incrementation)
{
 // action A
}
```

```
For (int i=0 ; i<20 ; i++) {
digitalWrite(13, HIGH) ;
delay(250) ;
digitalWrite(13,LOW);
delay(250) ;
}
```

L'instruction For

La structure for (pour) est utilisée afin de répéter une série d'instructions (comprise entre les accolades) le nombre de fois voulu. Un compteur à incrémenter est souvent utilisé afin de sortir de la boucle une fois le nombre de tour fait. Cette structure est composée de trois parties séparées par un point-virgule (;) dans l'entête de la boucle :

TPN°04: Chenillard à 4 LEDs.

Programme : Deuxième Méthode

```
// Initialisation des lignes 4 à  7 en sortie
void setup ()
{
  pinMode (4, OUTPUT) ;
  pinMode (5, OUTPUT) ;
  pinMode (6, OUTPUT) ;
  pinMode (7, OUTPUT) ;
}
```

TPN°04: Chenillard à 4 LEDs.

Programme : Deuxième Méthode

```
// Fonction loop
void loop ()
{    // Extinction de toutes les DEL au départ du programme
  for (byte i = 4 ; i <= 7 ; i++) {
    digitalWrite (i, LOW) ; // éteint la DEL reliée a  la broche i
  }
  // Boucle pour faire flasher les DEL
  for (byte i = 4 ; i <= 7 ; i++) {
    digitalWrite (i, HIGH) ; // allume la DEL sur broche i
    delay (50) ; // durée du flash 50 millisecondes
    digitalWrite (i, LOW) ; // éteint la DEL
  }
  delay (500) ; // délai de 500 millisecondes
  // Recommence la séquence
}
```

L'instruction While

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

```
while (condition) {
  // à remplacer par les instructions qui seront
  // exécutées tant que la condition est vraie
}
```

L'instruction While

```
while (condition) {
  // à remplacer par les instructions qui seront
  // exécutées tant que la condition est vraie
}
```

EXEMPLE

```
void loop() {
   unsigned long dureePression = 0; // variable pour compter la durée

   while (digitalRead(4) == HIGH) {
      dureePression++;
      delay(10);
   }
   Serial.println(dureePression);
}
```

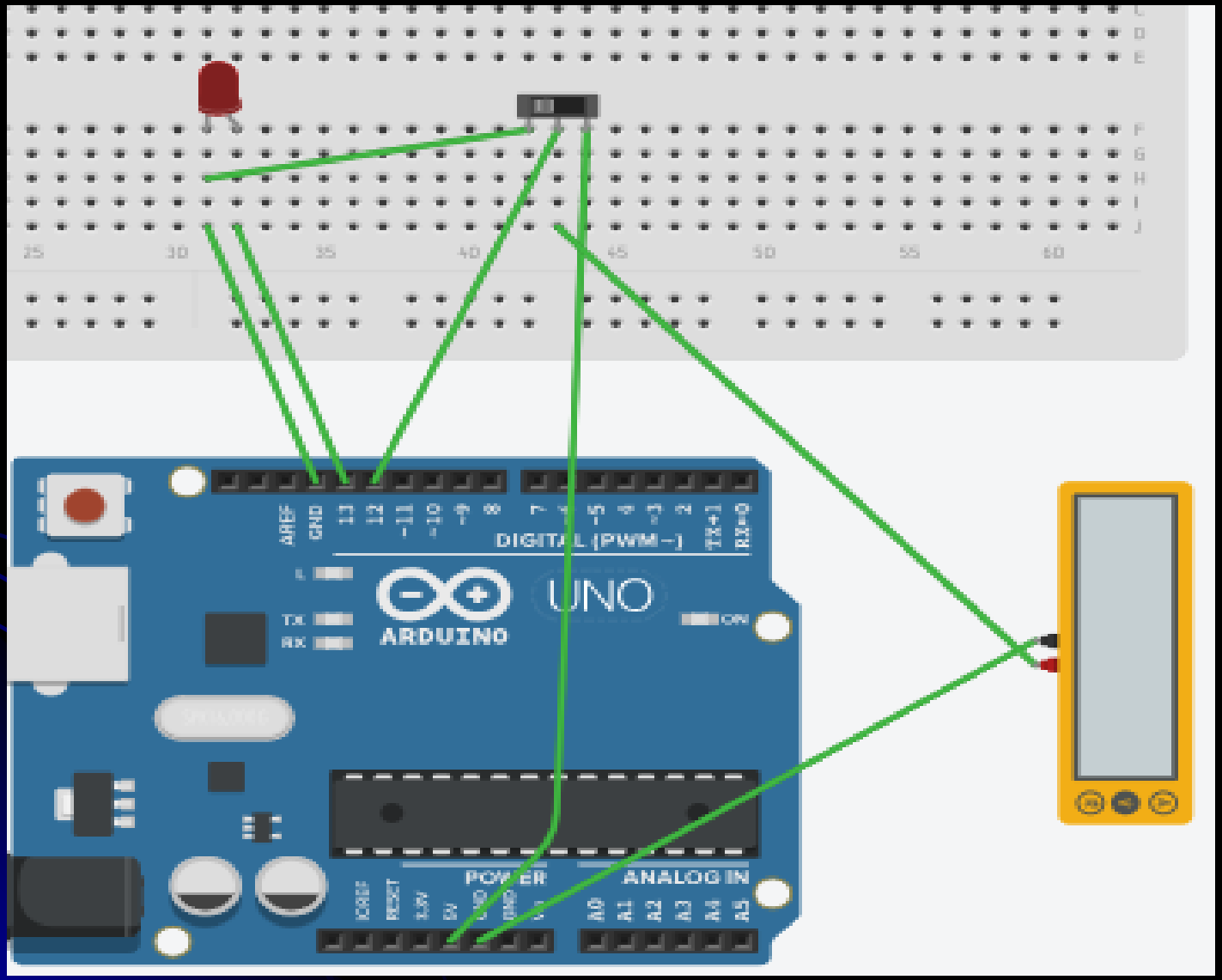
L'instruction While

TP N°06

Modifier le TPN°01 « clignoter une LED »
C’est-à-dire faire clignoter la LED tanque l’interrupteur est au niveau 1
Utiliser pour ce la l’instruction While

L'instruction While

TP N°06



L'instruction While

```
void setup() {
pinMode(13, OUTPUT); // Initialise la broche 13 comme sortie
pinMode(12, INPUT); // Initialise la broche 13 comme entrée
}
// le code dans cette fonction est exécuté en boucle
void loop() {
digitalWrite(13, LOW); // allumer la LED, tension 5V sur la broche13

  while (digitalRead(12) == HIGH) {
  digitalWrite(13,HIGH);
  delay(1000); // attendre 1000ms=1s
   digitalWrite(13, LOW); // éteindre la LED, tension 0V sur la broche 13
   delay(1000); // attendre 1 seconde
}
}
```

L'instruction Switch...case

La structure switch... case (commutateur de cas) permet de dresser une liste de tests ou de cas (comme le "if") et d'exécuter le code correspondant si un des cas du test est vrai. La principale différence entre le switch... case et le if...else est que le switch... case permet de continuer les tests même si un des cas est vrai, contrairement au if...else qui quittera dans le même cas.

L'instruction Switch...case

```
switch(Variable)
      {
       case A: inst 1;
       case B: inst 2;
       case C: inst 3;
       case D: inst 4;
      }
```

Variable sera comparée à A, si le test est vrai, inst 1 sera exécuté. Une fois le code exécuté (ou si le test est faux), " Variable " sera comparé à B et ainsi de suite. Dans le cas où on voudrait s'arrêter à un des cas, on ajoutera l'instruction break. De plus, si aucun des cas n'est vérifié, on peut ajouter le cas default qui sera exécuté si aucun test n'est vrai.

L'instruction break

L'instruction **break** est utilisée pour sortir d'une boucle **do**, **for** ou **while**, en passant outre le déroulement normal de la boucle. Cette instruction est également utilisée pour sortir d'une instruction **switch**.

```
byte x=50;
void setup(){
Serial.begin(9600);
}
void loop(){
switch (x) {
      case 20:
            Serial.println("Vaut 20 exactament");
            break;
      case 50:
            Serial.println("Vaut 50 exactament");
            break;
      default:
            Serial.println("Ne vaut pas les valeurs antérieures");
            }
delay(1000);
}
```

Définir les broches du microcontrôleur

Broche non-définie

```
void setup() {
  pinMode(13, OUTPUT);
// Initialise la broche 13 comme sortie
  Serial.begin(9600);
}

void loop() {
  digitalWrite(13, HIGH);
delay(500);
  digitalWrite(13, LOW);
delay(500);
}
```

Broche définie

```
const int led1= 13;
//La broche 13 devient led1

void setup() {
  pinMode(led1, OUTPUT);
// Initialise led1 comme broche de sortie
  Serial.begin(9600);
}

void loop() {
  digitalWrite(led1, HIGH);
delay(500);
digitalWrite(led1, LOW);
delay(500);
}
```

Jusqu'à maintenant, nous avons identifié les broches du microcontrôleur à l'aide de leurs numéros, comme dans l'exemple suivant: pinMode(13, OUTPUT);. Cela ne pose pas de problème quand on a une ou deux LEDs connectées. Mais dès qu'on a des montages plus compliqués, cela devient difficile de savoir qui fait quoi. Il est donc possible de renommer chaque broche du microcontrôleur.

Définir les broches du microcontrôleur

Broche non-définie

```
void setup() {
  pinMode(13, OUTPUT);
// Initialise la broche 13 comme sortie
  Serial.begin(9600);
}

void loop() {
  digitalWrite(13, HIGH);
delay(500);
  digitalWrite(13, LOW);
delay(500);
}
```

Broche définie

```
const int led1= 13;
//La broche 13 devient led1

void setup() {
  pinMode(led1, OUTPUT);
// Initialise led1 comme broche de sortie
  Serial.begin(9600);
}

void loop() {
  digitalWrite(led1, HIGH);
delay(500);
digitalWrite(led1, LOW);
delay(500);
}
```

Premièrement, définissons la broche utilisée du microcontrôleur en tant que variable. `const int led1 = 13;`
Le terme *const* signifie que l'on définit la variable comme étant constante . Par conséquent, on change la nature de la variable qui devient alors constante.

Définir les broches du microcontrôleur

Broche non-définie

```
void setup() {
  pinMode(13, OUTPUT);
// Initialise la broche 13 comme sortie
  Serial.begin(9600);
}

void loop() {
  digitalWrite(13, HIGH);
delay(500);
  digitalWrite(13, LOW);
delay(500);
}
```

Broche définie

```
const int led1= 13;
//La broche 13 devient led1

void setup() {
  pinMode(led1, OUTPUT);
// Initialise led1 comme broche de sortie
  Serial.begin(9600);
}

void loop() {
  digitalWrite(led1, HIGH);
delay(500);
digitalWrite(led1, LOW);
delay(500);
}
```

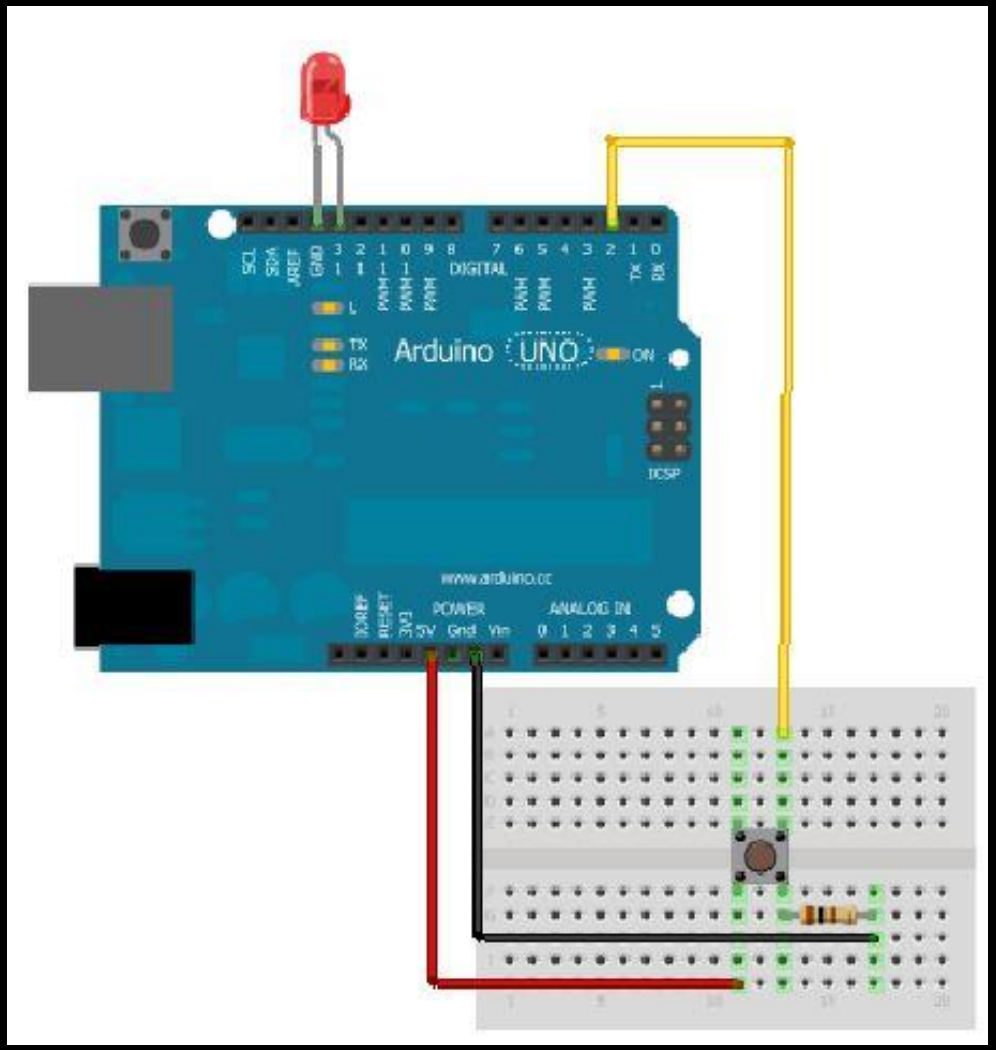
Le terme *int* correspond à un type de variable. Dans une variable de ce type, on peut stocker un nombre allant de -2147483648 à +2147483647, ce qui sera suffisant! Ainsi, la broche *13* s'appellera *led1*.
Nous sommes donc en présence d'une variable, nommée *led1*, qui est en fait une constante, qui peut prendre une valeur allant de -2147483648 à +2147483647. Dans notre cas, cette constante est assignée au nombre13.

TP N°03

Objectif : Détecter quand un bouton est appuyé pour allumer une LED.

Matériel

- ❑ une LED
- ❑ un bouton poussoir
- ❑ plaque d'essaie
- ❑ une résistance de 10 kΩ



TP N°03

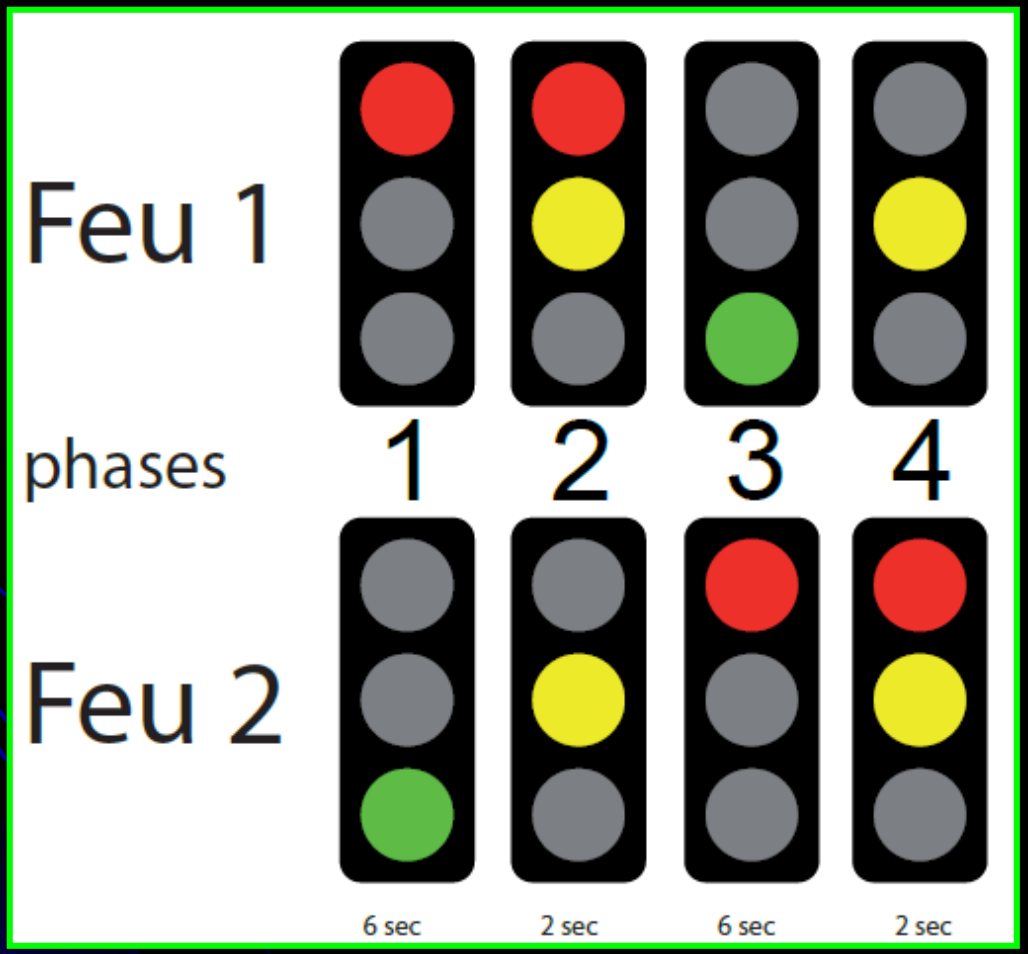
```
/*
Allume la LED quand on appuie sur le bouton.
*/
// Numero de la broche a laquelle est connecte le bouton
const int buttonPin = 2;
// Numero de la broche a laquelle est connectee la LED
const int ledPin = 13;
void setup() {
    // indique que la broche de la LED est utilisee en sortie :
    // on va modifier sa tension
    pinMode(ledPin, OUTPUT);
    // indique que la broche du bouton est utilisee en entree :
    // on va lire sa tension
    pinMode(buttonPin, INPUT);
    }
```

TP N°03

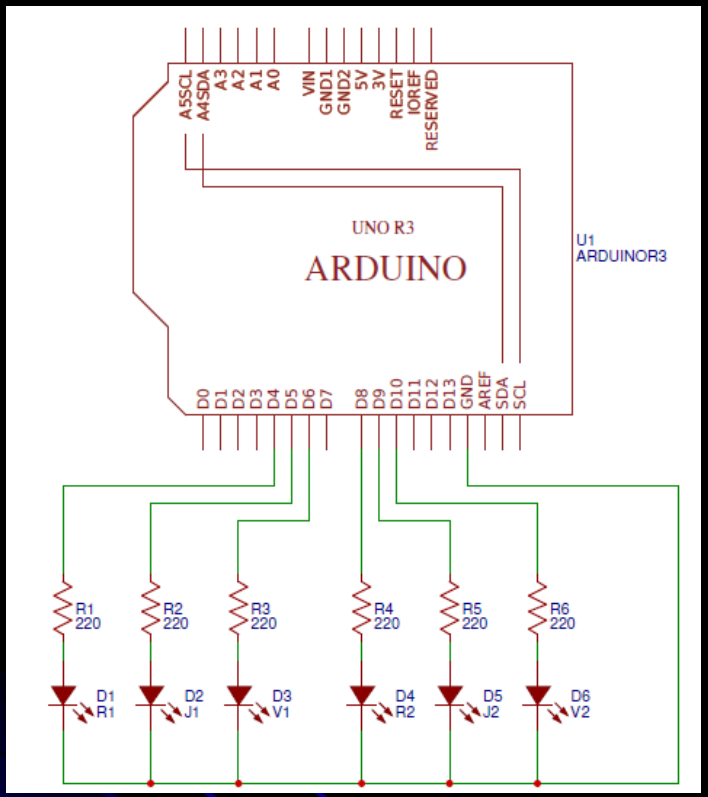
```
void loop() {
      // lit l'etat du bouton : appuye si on lit 5V
      // eteint si on lit 0 V
      int buttonState = digitalRead(buttonPin);
      delay(20); // anti rebond
      // Condition sur l'etat du bouton
      if (buttonState == HIGH) {
      // s'il est a 5V (HIGH)
      // on allume la LED
      digitalWrite(ledPin, HIGH);
      } else {
      // sinon
      // on eteint la LED
      digitalWrite(ledPin, LOW);
      }
}
```

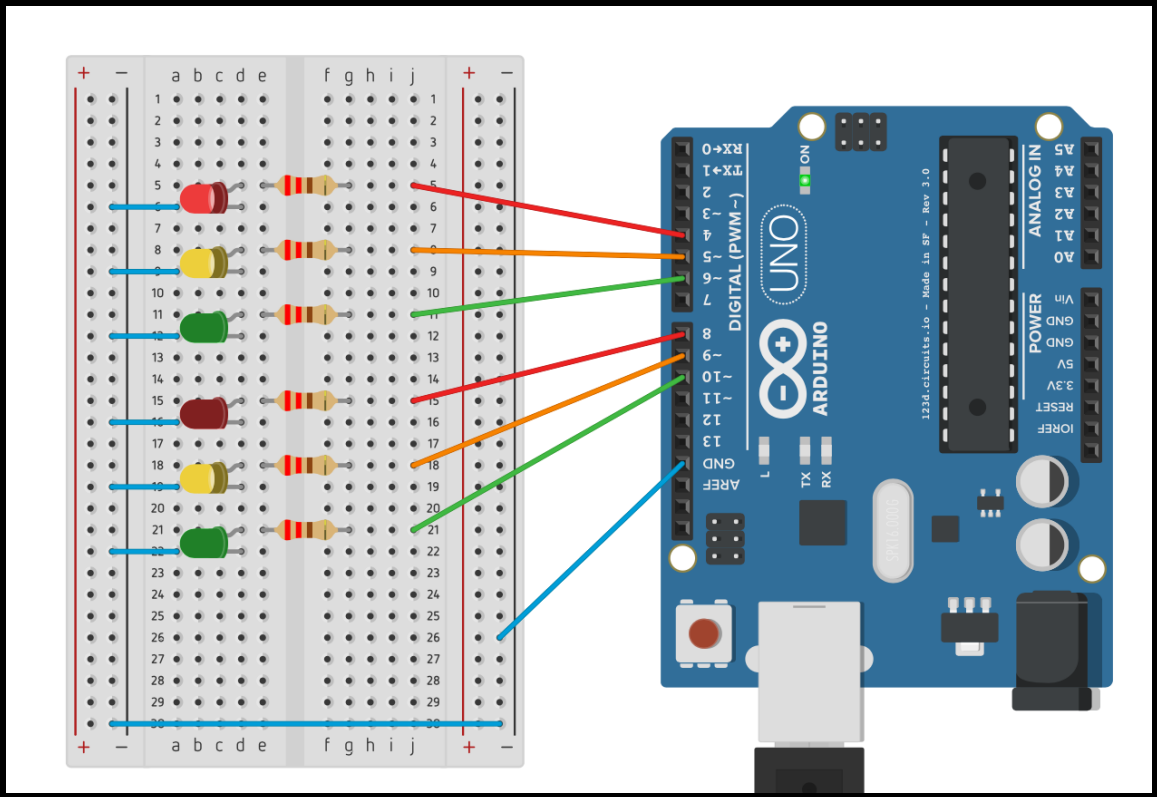
TP N°04: Les feux de circulation

L'objectif de ce TP est de créer deux feux de circulation et de les faire fonctionner de manière synchrone.
Voici les phases de deux feux de circulation que tu dois recréer:



TP N°04: Les feux de circulation





TP N°04: Les feux de circulation

Liste des composants

2 LEDs rouges
2 LEDs jaunes ou oranges
2 LEDs vertes
6 résistances de 220 à 470Ω

Afin de faciliter l'identification de chaque LED, nous allons renommer les broches comme suit:

Feu 1:

LED rouge connectée sur la broche 4 et renommée R1
LED jaune connectée sur la broche 5 et renommée J1
LED verte connectée sur la broche 6 et renommée V1

Feu 2:

LED rouge connectée sur la broche 8 et renommée R2
LED jaune connectée sur la broche 9 et renommée J2
LED verte connectée sur la broche 10 et renommée V2

Enfin, nous utiliserons deux variables timer1 et timer2 pour définir les temps d'allumages.

TP N°04: Les feux de circulation

```
/*
 Feux de circulation
*/
// On définit les variables pour chaque broche
//FEU 1
const int R1 = 4; //La broche 4 devient le feu rouge 1
const int J1 = 5; //La broche 3 devient le feu jaune 1
const int V1 = 6; //La broche 2 devient le feu vert 1
//FEU2
const int R2 = 8; //La broche 8 devient le feu rouge 2
const int J2 = 9; //La broche 9 devient le feu jaune 2
const int V2 = 10; //La broche 10 devient le feu vert 2
//TEMPS
int timer1 = 2000; //Le temps est fixé à 2 secondes
int timer2 = 6000; //Le temps est fixé à 6 secondes
```

TP N°04: Les feux de circulation

```
void setup() {
//On initialise les sorties
pinMode(R1, OUTPUT);
pinMode(J1, OUTPUT);
pinMode(V1, OUTPUT);
pinMode(R2, OUTPUT);
pinMode(J2, OUTPUT);
pinMode(V2, OUTPUT);
}
```

TP N°04: Les feux de circulation

```
void loop() {
//Phase 1: R1 et V2 sont allumés, R2, J1 et J2 sont éteints
digitalWrite(R2, LOW); //R2 éteint
digitalWrite(J1, LOW); //J1 éteint
digitalWrite(J2, LOW); //J2 éteint
digitalWrite(R1, HIGH); //R1 allumé
digitalWrite(V2, HIGH); //V2 allumé
delay(timer2); //durée: 6'000 millisecondes, soit 6 secondes
//Phase 2: R1, J1, J2 allumés et V2 éteint
digitalWrite(V2, LOW); //V2 éteint
digitalWrite(J1, HIGH); //J1 allumé
digitalWrite(J2, HIGH); //J2 allumé
delay(timer1); //durée: 2'000 millisecondes, soit 2 secondes
```

TP N°04: Les feux de circulation

```
//Phase 3: R1, J1, J2 éteints et V1 et R2 allumés
digitalWrite(R1, LOW); //R1 éteint
digitalWrite(J1, LOW); //J1 éteint
digitalWrite(J2, LOW); //J2 éteint
digitalWrite(V1, HIGH); //V1 allumé
digitalWrite(R2, HIGH); //R2 allumé
delay(timer2);
//Phase 4: V1 éteint et J1, J2 et R2 allumés
digitalWrite(V1, LOW); //V1 éteint
digitalWrite(J1, HIGH); //J1 allumé
digitalWrite(J2, HIGH); //J2 allumé
digitalWrite(R2, HIGH); //R2 allumé
delay(timer1);
}
```