

# ► Algo Applications

Application de l'algorithmique à la programmation  
Arduino

Cours II - Boutons pressoirs, entrées/sorties, introduction à  
la syntaxe Arduino

# Broches numériques

- ▶ Les broches numériques de l'Arduino ne connaissent que deux états : lorsqu'il y a une tension sur une entrée et lorsqu'il n'y en a pas. Par anglicisme on qualifie ces entrées de digitales.
- ▶ Ces états sont communément appelés **HIGH** (haut) et **LOW** (bas). **HIGH** équivaut à dire « il y'a une tension ici ! » et **LOW** signifie « il n'y a pas de tension sur cette broche ! ».
- ▶ Quand vous mettez une sortie (*OUTPUT*) à l'état haut (*HIGH*) en utilisant une commande appelée ***digitalWrite()*** vous l'activez.
- ▶ Les broches numériques de l'Arduino peuvent se comporter en entrée ou en sortie

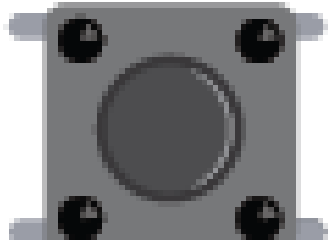
# Configuration des broches numériques

Dans votre code, vous configurerez les broches digitales en fonction de la manière dont vous voulez les utiliser.

Quand les broches sont définies en sortie, vous pouvez les utiliser pour contrôler des composants comme les LEDs.

Lorsqu'elles sont définies en entrées, vous pouvez vérifier si l'on appuie sur un bouton ou non. Etant donné que les broches 0 et 1 sont utilisées pour la communication avec l'ordinateur, il vaut mieux commencer par la broche 2.

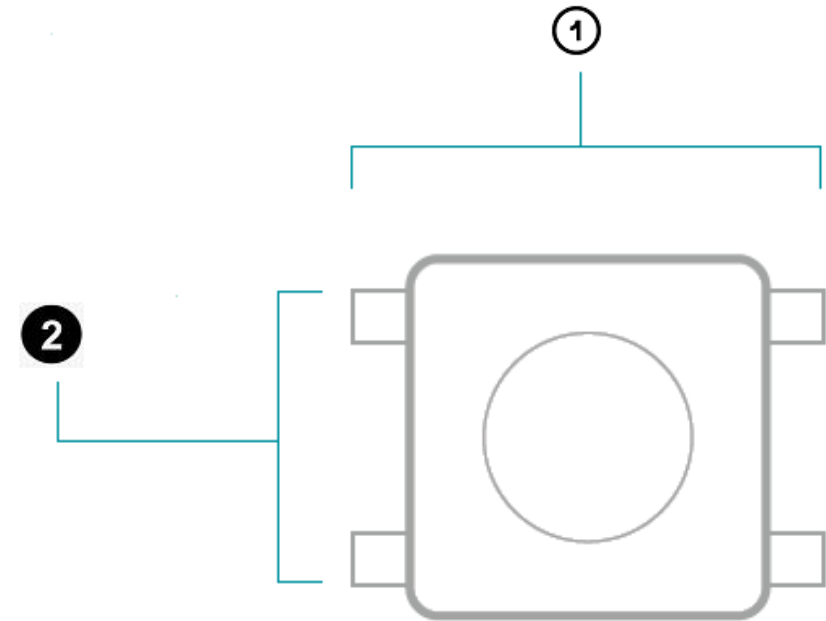
# Le bouton poussoir



- Pour illustrer l'utilisation d'une broche digitale en entrée on va écrire un programme Arduino basé sur l'utilisation d'un nouveau composant : **Les boutons poussoirs**.
- Un **interrupteur** interrompt le passage de l'électricité, coupant le circuit lorsqu'il est ouvert. Il existe plusieurs types d'interrupteurs dont les **boutons poussoirs** que nous allons utiliser par la suite.

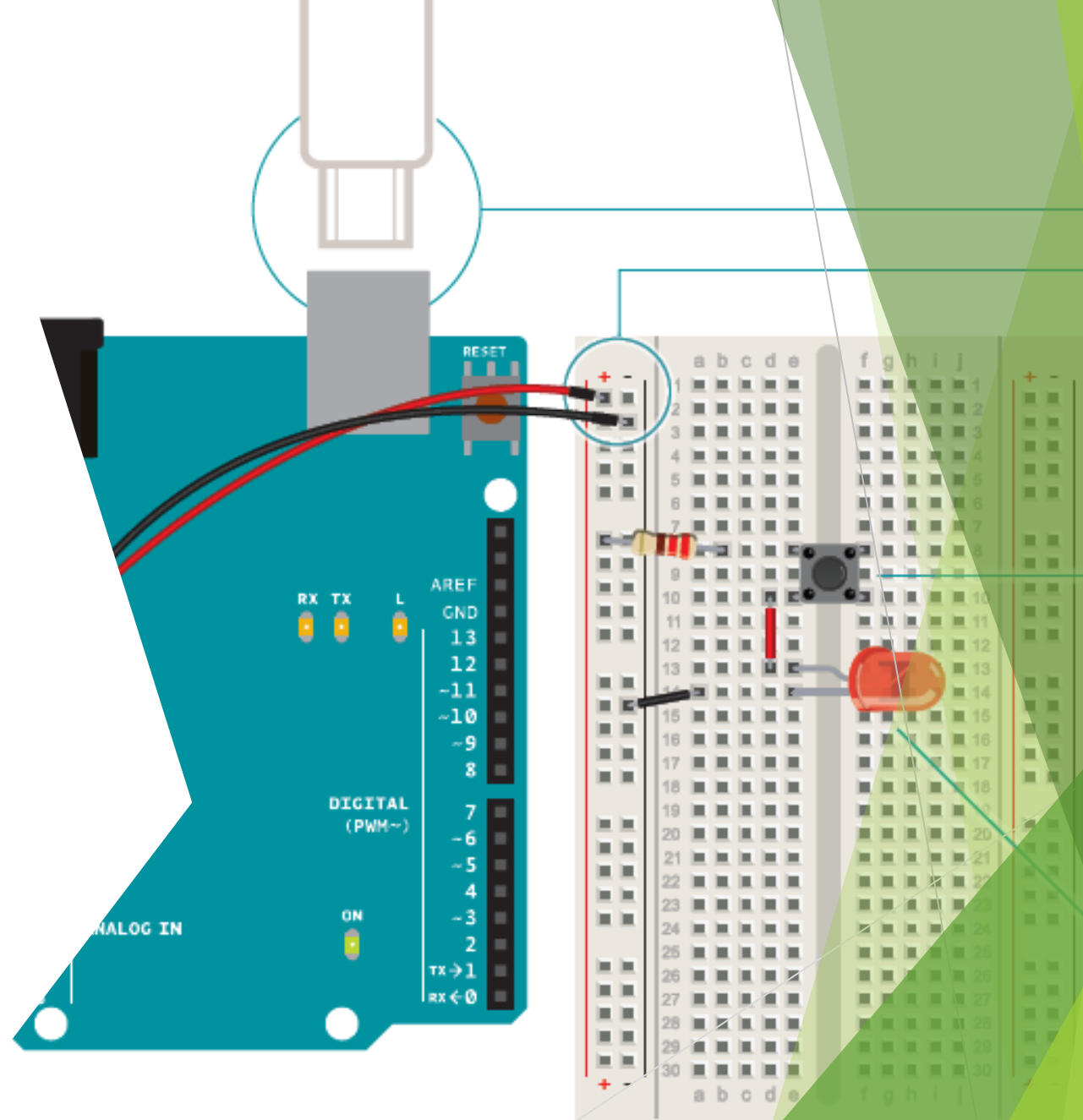
# Fonctionnement du bouton poussoir

- ▶ 1. Ces deux pattes du bouton sont connectées ensemble.
- ▶ 2. Ces deux pattes du bouton ne sont pas connectées ensemble. Il faut appuyer sur le bouton pour qu'elles le soient. Elles forment l'interrupteur.

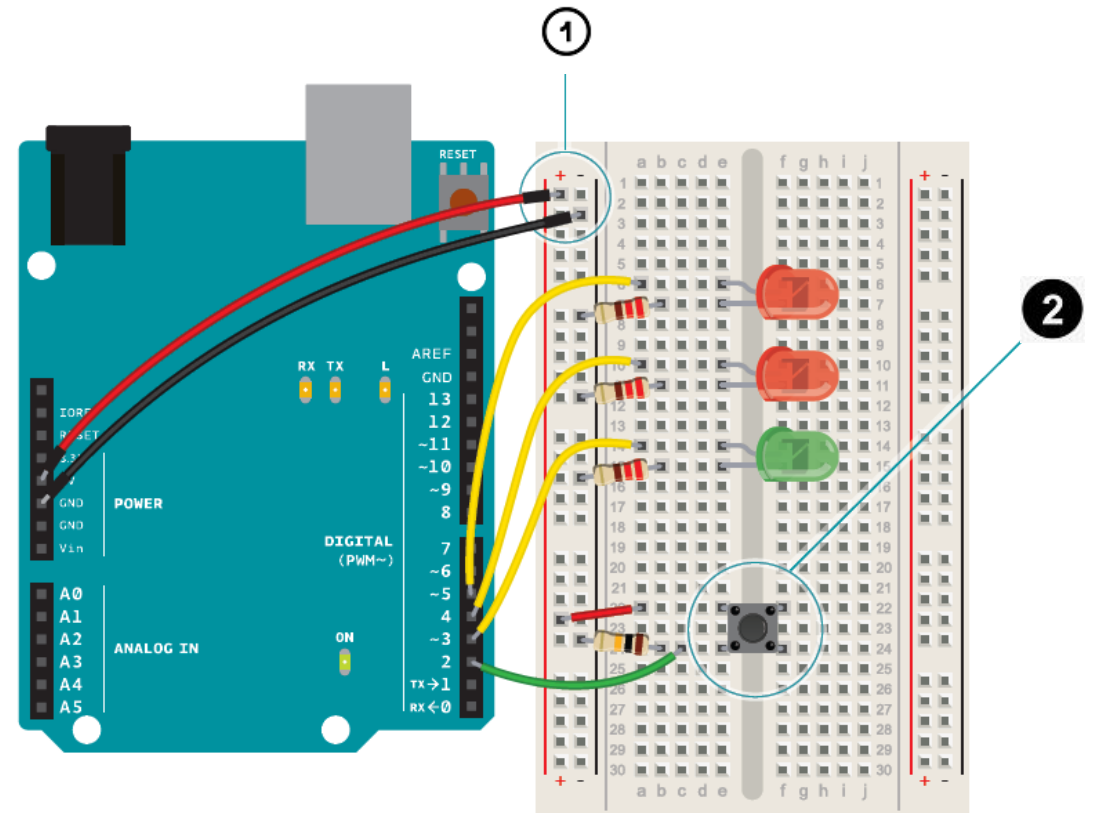
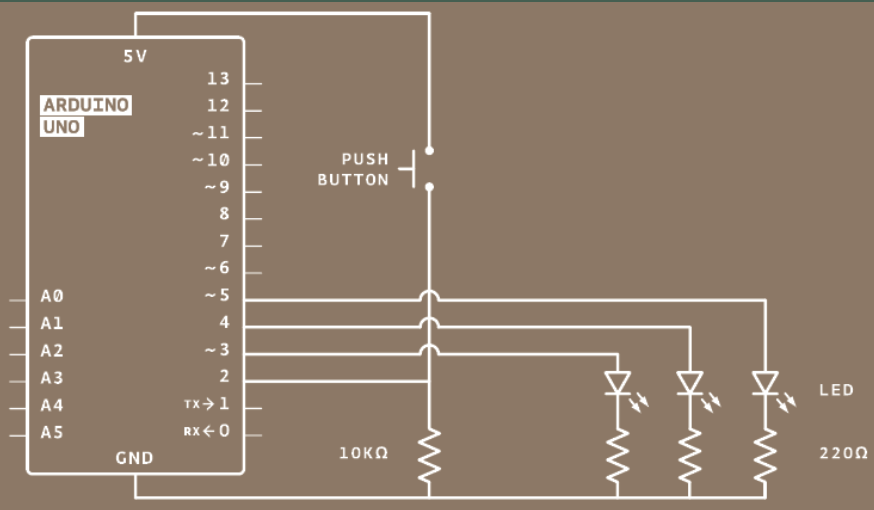


## Du mal à différencier les pattes connectées des pattes non connectées ?

- Il est commun (mais pas obligatoire) de placer le bouton poussoir au-dessus de la ligne centrale de la plaque d'essai. Quand vous la placez ainsi elle ne peut être posée que dans un seul sens.
- **La courbure des pattes pointe vers le centre de la plaque.** Lorsque le bouton est placé ainsi les pattes connectées vont de gauche à droite, les pattes non connectées vont de haut en bas.



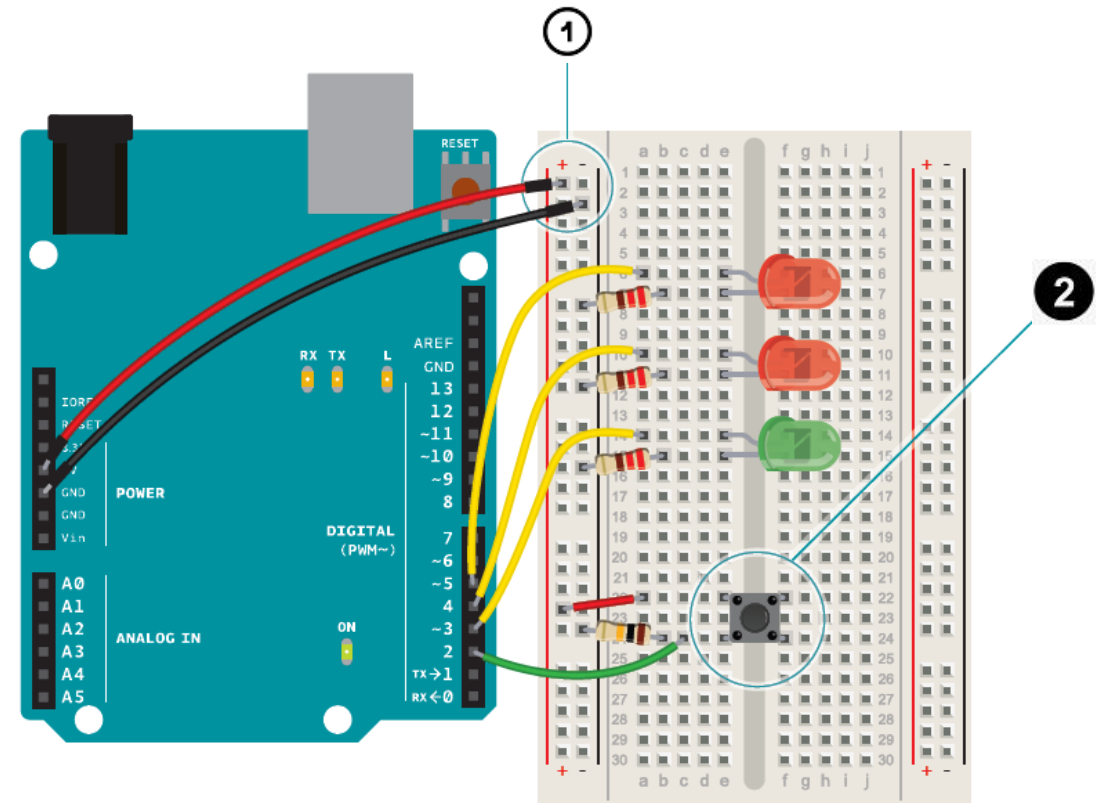
# Montage d'un nouveau circuit électronique



# Présentation d'un nouveau circuit (2)

1. On branche la plaque d'essai aux broches 5V et GND de l'Arduino. On place trois LEDs sur la breadboard.

La cathode (patte courte) est reliée à la masse via une résistance de 220 Ohms. L'anode (patte longue) est reliée à la borne digitale.

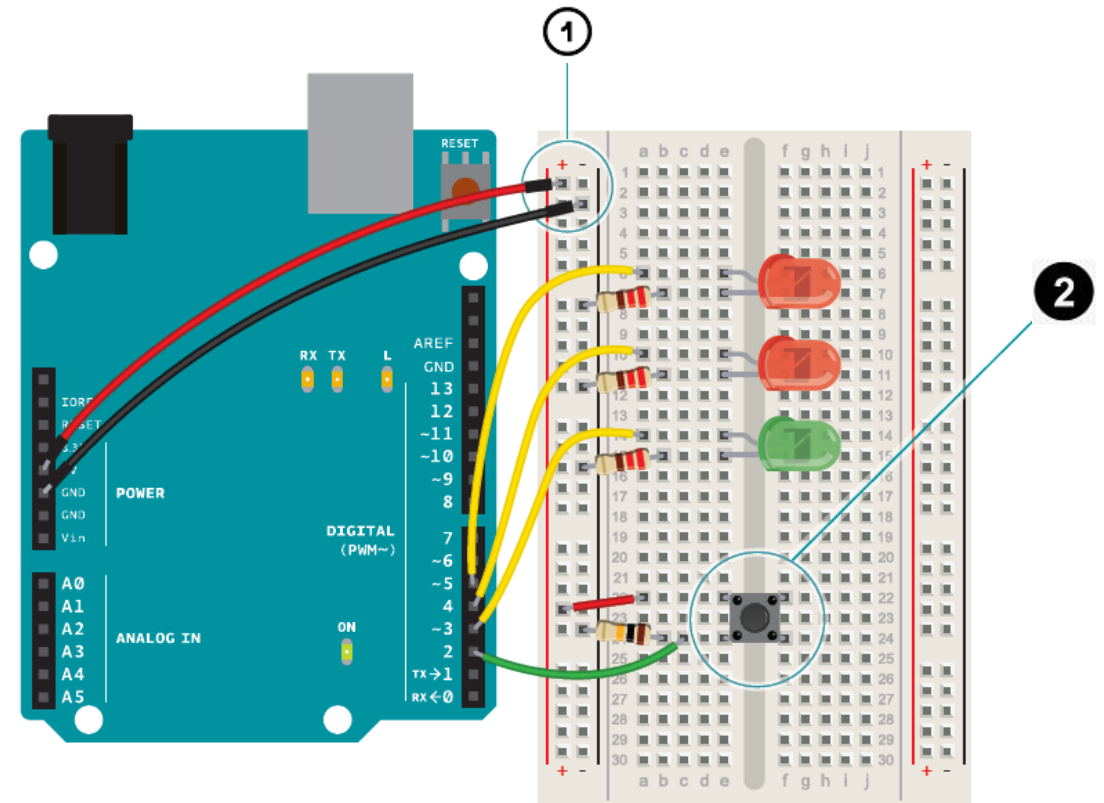




# Présentation d'un nouveau circuit (3)

2. On place le bouton poussoir sur la plaque d'essai, reliée d'un côté à l'alimentation et de l'autre côté à la broche numérique 2 de l'Arduino.

On prend le soin d'ajouter une résistance de 10 kiloOhms entre la masse et la patte du bouton qui communique avec l'Arduino, comme ça on s'assure que la borne lise un signal LOW lorsqu'il n'y a aucune tension.



# Mini projet : Panneau de contrôle avec LEDs

- ▶ Avec le montage illustré dans les précédents slides, nous allons monter un projet Arduino.
- ▶ But : Une LED verte est allumée jusqu'à ce que l'utilisatrice presse un bouton. Lorsque l'Arduino reçoit un signal du bouton, la lumière verte s'éteint et les deux autres lumières se mettent à clignoter.

# Le code : Rappel

- ▶ Chaque programme Arduino a deux fonctions principales ***setup()*** et ***loop()***. Dans un programme une **fonction** est une suite d'instructions que l'on déclare pour indiquer à l'Arduino quoi faire, puis que l'on exécute en l'appelant.
- ▶ Une variable est un nom donné à une case mémoire de l'Arduino, permettant de garder une trace des événements et dont la valeur peut changer suivant le déroulement du programme.

**CONSEIL** : Nommez vos variables en fonction de leur rôle.

```
void setup() {  
}  
  
void loop() {  
}
```

# Commençons à coder

- ▶ Le type de données `int` contiendra un entier (*integer* en anglais); Il s'agit de n'importe quel nombre entier.
- ▶ Quand on déclare une variable, on lui donne une valeur initiale. La déclaration de la variable, comme chaque commande, se termine avec un point-virgule (;).
- ▶ On nomme notre variable *switchState* car elle est censé représenter l'état du bouton (switch signifie interrupteur en anglais et state signifie état).

```
int switchState = 0;
```

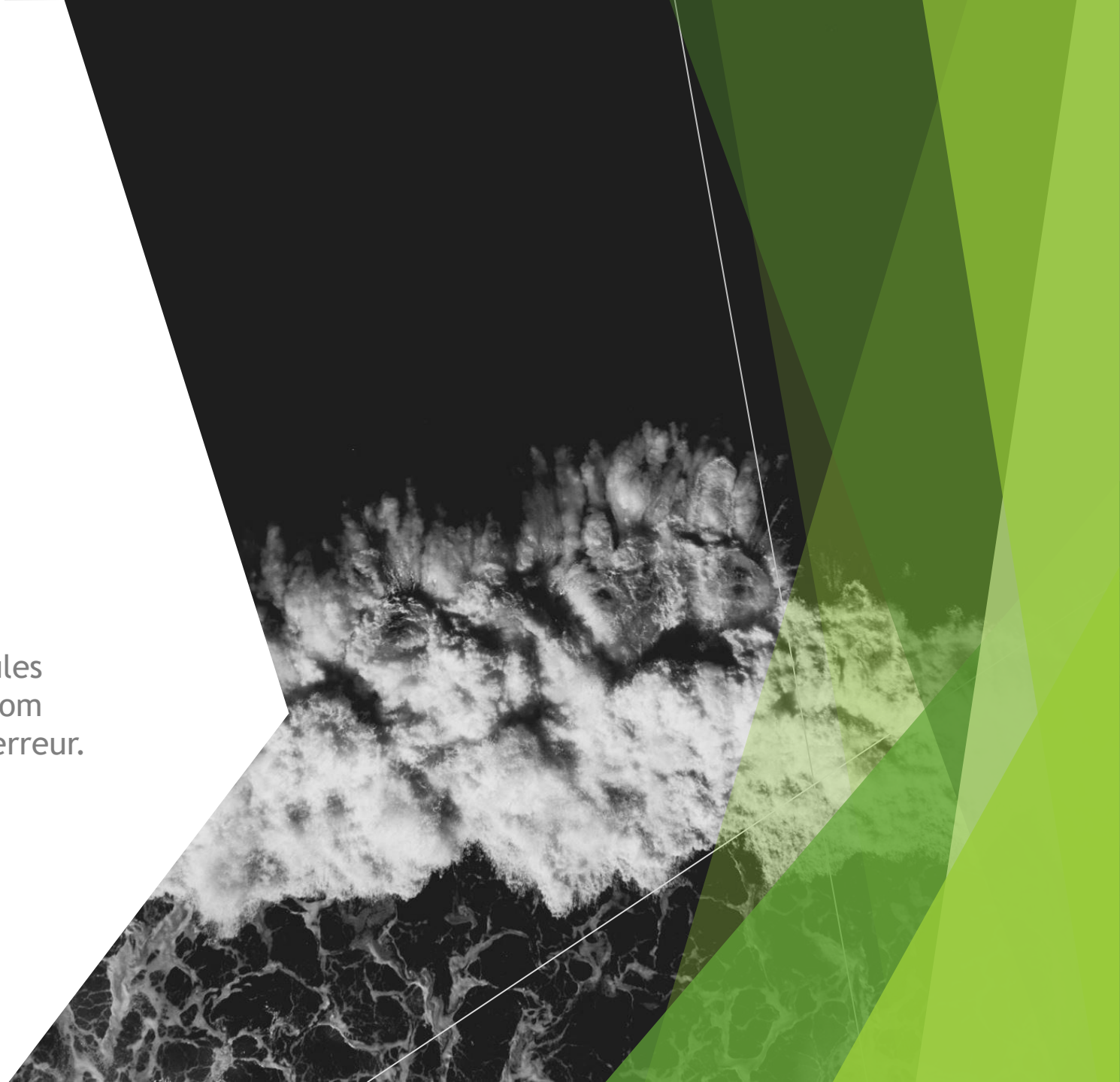
# Configurez les fonctions des broches

- ▶ Le ***setup()*** est exécuté une seule fois, au moment où l'Arduino s'allume. C'est à ce moment qu'on configure les broches numériques pour être soit des entrées, soit des sorties.
- ▶ Pour faire ça, on utilise une fonction appelée ***pinMode()***. Les broches connectées aux LEDs seront des sorties (***OUTPUT***) et l'interrupteur sera une entrée (***INPUT***).

```
void setup() {  
    pinMode(3, OUTPUT); // LED verte  
    pinMode(4, OUTPUT); // LED rouge  
    pinMode(5, OUTPUT); // LED rouge  
    pinMode(2, INPUT);  // Le bouton poussoir  
}  
  
/* Voir schéma du montage */
```

# Attention à la casse

Fâites attention aux majuscules et aux minuscules dans votre code. Par exemple **pinMode** est le nom d'une commande, mais *pinmode* produira une erreur.



# Créer la fonction loop

- ▶ La **loop()** (boucle) se répète continuellement après que le **setup()** se soit exécuté. La **loop()** est l'endroit où on regarde ce qui se passe sur les entrées et où on met les sorties à l'état haut ou bas.
- ▶ Pour vérifier le niveau de tension sur une broche, on utilise la fonction **digitalRead()**.
- ▶ Pour savoir quelle broche regarder, **digitalRead()** a besoin d'un argument. Les arguments sont des informations qu'on met dans des fonctions pour leur dire comment faire leur travail.

```
void loop() {  
    switchState = digitalRead(2);  
}
```

# Créer la fonction loop (2)

- ▶ La fonction, ***digitalRead()*** a besoin d'un argument : la broche à aller vérifier. Cette fonction va vérifier l'état de la broche 2 et stocker la valeur dans la variable *switchState*.
- ▶ S'il y'a une tension sur la broche lorsque ***digitalRead()*** est appelée, la variable *switchState* aura pour valeur **HIGH** (ou 1). S'il n'y a aucune tension sur la broche, *switchState* aura pour valeur **LOW** (ou 0).

```
void loop() {  
    switchState = digitalRead(2);  
}
```



# L'instruction *if*

- ▶ On utilise l'instruction *if* pour vérifier l'état de l'interrupteur. Cette instruction compare deux choses et détermine si la comparaison est vraie ou fausse. Si la comparaison est vraie les instructions du *if* sont exécutées.
- ▶ Lorsque l'on compare deux choses en programmation, on utilise deux signes égal ==. Si vous n'en mettez qu'un seul, vous attribuerez une valeur à une variable au lieu de la comparer.

```
if (switchState == LOW) {  
    // pas d'appui sur le bouton  
    digitalWrite(3, HIGH); // LED verte  
    digitalWrite(4, LOW);  // LED rouge  
    digitalWrite(5, LOW);  // LED rouge  
}
```

# L'instruction *else*

- ▶ L'instruction *if* peut être complétée par un *else* (sinon) qui permet qu'une autre chose se passe si la première condition n'est pas remplie.
- ▶ Comme vous venez de vérifier si l'interrupteur était fermé (**LOW**) on écrit du code pour l'autre cas (**HIGH**) à l'intérieur des accolades du *else*.


```
else {  
    // pas d'appui sur le bouton  
    digitalWrite(3, LOW); // LED verte  
    digitalWrite(4, LOW); // LED rouge  
    digitalWrite(5, HIGH); // LED rouge  
    delay(250);  
    digitalWrite(4, HIGH); // LED rouge  
    digitalWrite(5, LOW); // LED rouge  
    delay(250);  
}
```

# Résultat

- ▶ A présent notre programme fait clignoter les LEDs lorsqu'on appuie sur le bouton.
- ▶ La carte Arduino devra attendre un peu entre deux changements d'états des LEDs, sinon on aura pas le temps de voir le changement : Elles auront juste l'air d'être allumées faiblement. En réalité c'est parce que l'Arduino fait le tour de la *loop()* plus vite que nous ne pouvons le percevoir.
- ▶ La fonction *delay()* permet donc de mettre l'Arduino en pause pendant un laps de temps qu'on indique en millisecondes. Ici **250** correspond donc à un quart de seconde.

# Introduction à la syntaxe d'Arduino (suite)

A présent que vous avez une vision globale sur le fonctionnement de la carte Arduino, il est temps d'approfondir votre connaissance de la syntaxe Arduino



Le langage Arduino est très proche du C et du C++. C'est de ces deux langages (en particulier du C) que s'inspire le python.

# Quelques opérations bien pratiques

- Voyons un peu d'autres opérations qui facilitent parfois l'écriture du code.

- **L'incrémentation** : on ajoute le chiffre 1 à la valeur de *var*

```
var = 0;  
var++; // incrémentation, on ajoute une unité à var
```

- **La décrémentation** : C'est l'inverse de l'incrémentation. Autrement dit, on enlève le chiffre 1 à la valeur de *var*.

```
var = 0;  
var--; // décrémentation, on enlève une unité à var
```

# Quelques opérations bien pratiques (2)

- Il existe des raccourcis lorsque l'on veut effectuer une opération sur une même variable :

```
int var = 10;  
var = var + 6; // <=> var += 6;  
var = var - 6; // <=> var -= 6;  
var = var * 6; // <=> var *= 6;  
var = var / 5; // <=> var /= 5;
```

# Les booléens

- 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.

```
// variable est fausse car elle vaut FALSE
boolean_variable = FALSE; // <=> boolean_variable = 0 ou LOW

// variable est vraie car elle vaut TRUE,
boolean_variable = TRUE; // <=> boolean_variable = 1 ou HIGH
```

- Pour inverser la valeur d'un booléen on peut utiliser l'opérateur (!) qui l'équivalent du NON

```
bool x = TRUE;
x = !x; // x vaut maintenant FALSE
```

# Les opérateurs logiques

- Pour complexifier un peu nos comparaisons on peut faire appel aux opérateurs logiques traditionnels : et noté `&&`, ou noté `||`, non noté `!`

```
int prix_voiture = 5500;
int option_GPS = TRUE;

if((prix_voiture < 5000) || (prix_voiture == 5500 &&
option_GPS)) {
// la condition est vraie, donc j'achète la voiture
}
else {
// la condition est fausse, donc je n'achète pas la voiture
}
```



# Les boucles

- ▶ En programmation, une **boucle** est une instruction qui permet de répéter un bout de code. Cela va nous permettre de répéter les instructions d'une partie du programme. Il existe deux types principaux de boucles :
  - La **boucle conditionnelle** , qui teste une condition et qui exécute les instructions qu'elle contient tant que la condition testée est vraie.
  - La **boucle de répétition** , qui exécute les instructions qu'elle contient, un nombre de fois prédéterminé.

# La boucle *while*

## ► Problème :

- Je veux que le volet électrique de ma fenêtre se ferme automatiquement quand la nuit tombe (Nous ne nous occuperons pas de faire le système qui ferme le volet à l'arrivée de la nuit). La carte Arduino dispose d'un capteur qui indique la position du volet (ouvert ou fermé).

Ce que nous cherchons à faire : c'est créer un bout de code qui fait descendre le volet tant qu'il n'est pas fermé .

```
while(position_volet == "ouvert") {  
  // instructions qui font descendre le volet  
}
```

# La boucle *while* (2)

- ▶ Voilà donc la syntaxe de cette boucle qu'il faut retenir :

```
while(/* condition à tester */) {  
  // les instructions entre ces accolades sont répétées  
  // tant que la condition est vraie  
}
```

# La boucle *for*

- ▶ Cette boucle est exécutée X fois. Contrairement à la boucle précédente, on doit lui donner trois paramètres.

```
for(int compteur = 0; compteur < 5; compteur++) {  
    // code à exécuter  
}
```

D'abord, on crée la boucle avec le terme for (signifie "pour que"). Ensuite, entre les parenthèses, on doit donner trois paramètres qui sont :

- ✓ la création et l'assignation de la variable à une valeur de départ
- ✓ suivi de la définition de la condition à tester
- ✓ suivit de l'instruction à exécuter