

# [Arduino 1] Découverte de l'Arduino

Dans cette première partie, nous ferons nos premiers pas avec Arduino. Nous allons avant tout voir de quoi il s'agit exactement, essayer de comprendre comment cela fonctionne, puis installerons le matériel et le logiciel pour ensuite enchaîner sur l'apprentissage du langage de programmation nécessaire au bon fonctionnement de la carte Arduino. Soyez donc attentif afin de bien comprendre tout ce que je vais vous expliquer. Sans les bases, vous n'irez pas bien loin... 😊

---

## [Arduino 101] Présentation d'Arduino

Comment faire de l'électronique en utilisant un langage de programmation ? La réponse, c'est le projet Arduino qui l'apporte. Vous allez le voir, celui-ci a été conçu pour être accessible à tous par sa simplicité. Mais il peut également être d'usage professionnel, tant les possibilités d'applications sont nombreuses.

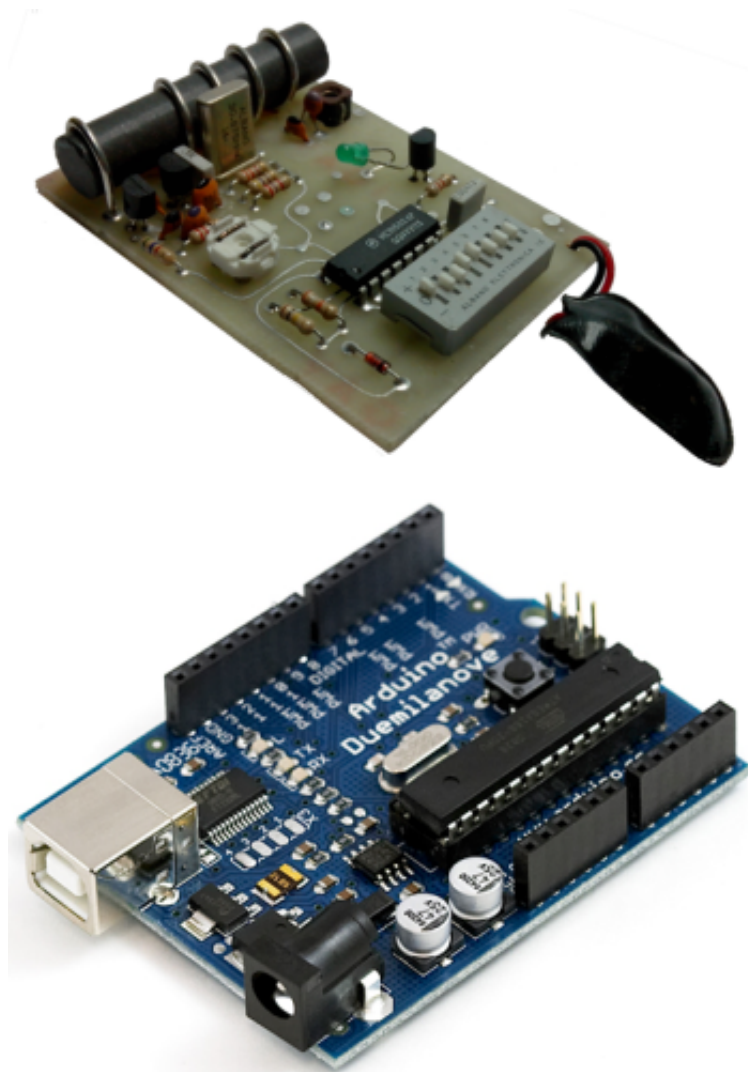
---

### Qu'est-ce que c'est ?

Une équipe de développeurs composée de *Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, David Mellis et Nicholas Zambetti* a imaginé un projet répondant au doux nom de **Arduino** et mettant en œuvre une petite carte électronique programmable et un logiciel multiplateforme, qui puisse être accessible à tout un chacun dans le but de créer facilement des systèmes électroniques. Étant donné qu'il y a des débutants parmi nous, commençons par voir un peu le vocabulaire commun propre au domaine de l'électronique et de l'informatique.

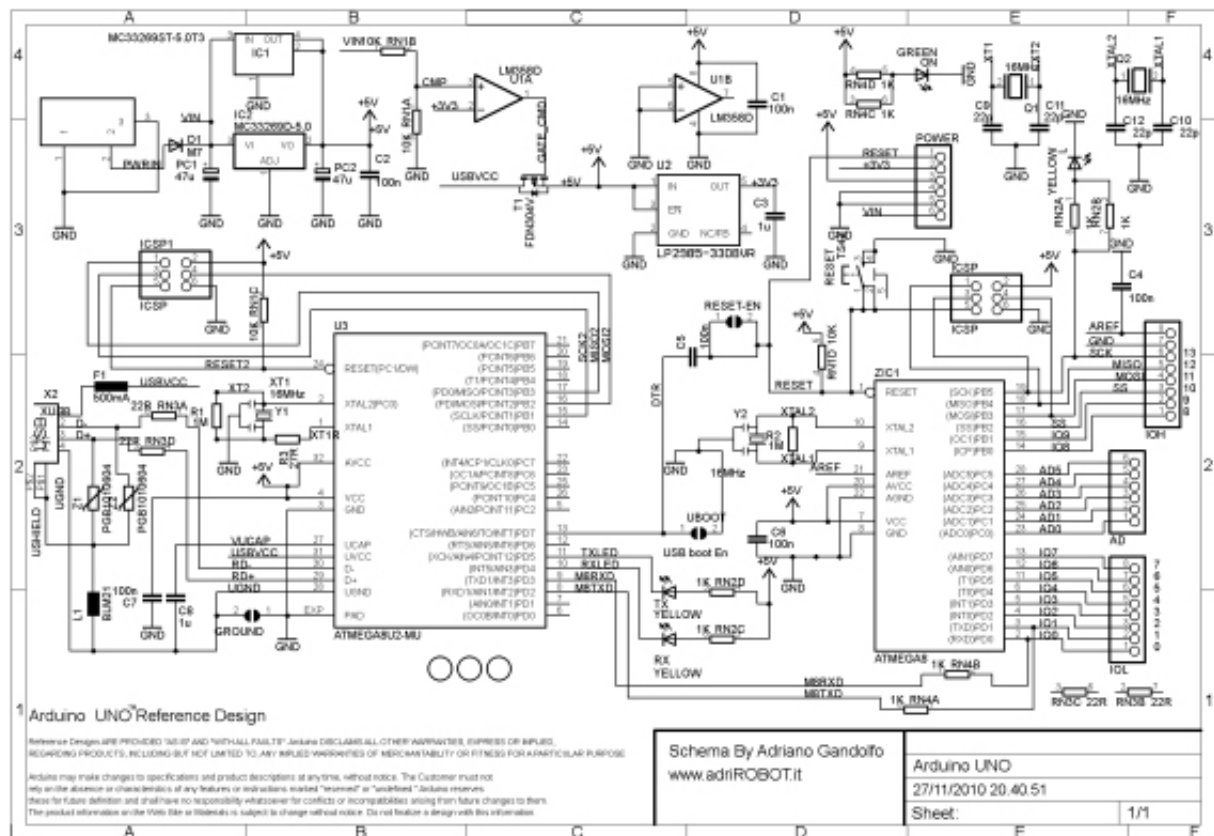
### Une carte électronique

Une **carte électronique** est un support plan, flexible ou rigide, généralement composé d'epoxy ou de fibre de verre. Elle possède des pistes électriques disposées sur une, deux ou plusieurs couches (en surface et/ou en interne) qui permettent la mise en relation électrique des composants électroniques. Chaque piste relie tel composant à tel autre, de façon à créer un système électronique qui fonctionne et qui réalise les opérations demandées.



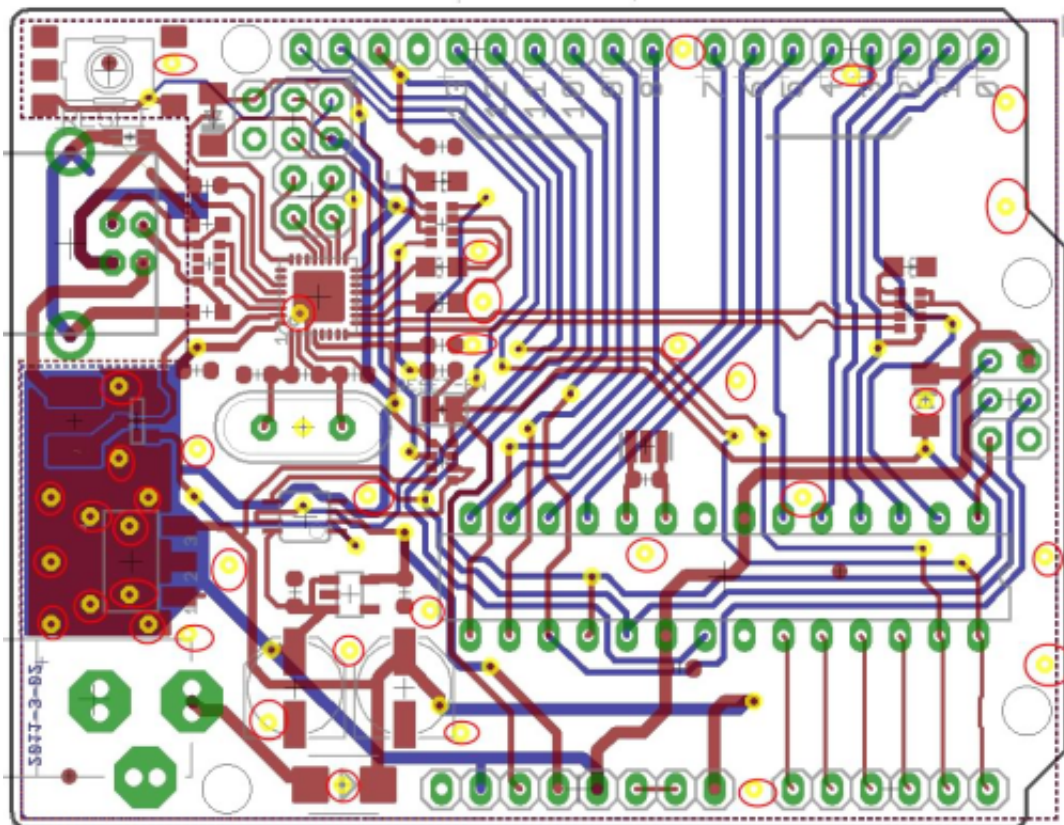
*Exemples de cartes électroniques*

Évidemment, tous les composants d'une carte électronique ne sont pas forcément reliés entre eux. Le câblage des composants suit un plan spécifique à chaque carte électronique, qui se nomme le **schéma électronique**.



Exemple de schéma électronique – carte Arduino Uno

Enfin, avant de passer à la réalisation d'une carte électronique, il est nécessaire de transformer le schéma électronique en un **schéma de câblage**, appelé **typon**.



Exemple de typon – carte Arduino

Une fois que l'on a une carte électronique, on fait quoi avec ?

Eh bien une fois que la carte électronique est faite, nous n'avons plus qu'à la tester et l'utiliser ! Dans notre cas, avec Arduino, nous n'aurons pas à fabriquer la carte et encore moins à la concevoir. Elle existe, elle est déjà prête à l'emploi et nous n'avons plus qu'à l'utiliser. Et pour cela, vous allez devoir apprendre comment l'utiliser, ce que je vais vous montrer dans ce tutoriel.

## Programmable ?

J'ai parlé de **carte électronique programmable** au début de ce chapitre. Mais savez-vous ce que c'est exactement ? Non ? pas vraiment ? Alors voyons ensemble de quoi il s'agit. La carte Arduino est une carte électronique qui ne sait rien faire sans qu'on lui dise quoi faire. Pourquoi ? Eh bien c'est du au fait qu'elle est **programmable**. Cela signifie qu'elle a besoin d'un **programme** pour fonctionner.

### Un programme

Un programme est une liste d'instructions qui est exécutée par un système. Par exemple votre navigateur internet, avec lequel vous lisez probablement ce cours, est un programme. On peut analogiquement faire référence à une liste de course :

- Lait
- Pain
- Steak
- Epinards
- ...



Chaque élément de cette liste est une **instruction** qui vous dit : “Va chercher le lait” ou “Va chercher le pain”, etc. Dans un programme le fonctionnement est similaire :

- Attendre que l'utilisateur rentre un site internet à consulter
- Rechercher sur internet la page demandée
- Afficher le résultat

Tel pourrait être le fonctionnement de votre navigateur internet. Il va attendre que vous lui demandiez quelque chose pour aller le chercher et ensuite vous le montrer. Eh bien, tout aussi simplement que ces deux cas, une carte électronique programmable suit une liste d'instructions pour effectuer les opérations demandées par le programme.

Et on les trouve où ces programmes ? Comment on fait pour le mettre dans la carte ? o\_O

Des programmes, on peut en trouver de partout. Mais restons concentré sur Arduino. Le programme que nous allons mettre dans la carte Arduino, c'est nous qui allons le réaliser. Oui, vous avez bien lu. Nous allons programmer cette carte Arduino. Bien sûr,

ce ne sera pas aussi simple qu'une liste de course, mais rassurez-vous cependant car nous allons réussir quand même ! Je vous montrerai comment y parvenir, puisque avant tout c'est un des objectifs de ce tutoriel. Voici un exemple de programme :

```
// constants won't change. They're used here to
// set pin numbers:
const int buttonPin = 2;    // the number of the pushbutton pin
const int ledPin = 13;      // the number of the LED pin

// variables will change:
int buttonState = 0;        // variable for reading the pushbutton status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop(){
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  }
  else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

Vous le voyez comme moi, il s'agit de plusieurs lignes de texte, chacune étant une instruction. Ce langage ressemble à un véritable baragouin et ne semble vouloir à *a priori* rien dire du tout... Et pourtant, c'est ce que nous saurons faire dans quelques temps ! Car nous apprendrons le **langage informatique** utilisé pour programmer la carte Arduino. Je ne m'attarde pas sur les détails, nous aurons amplement le temps de revenir sur le sujet plus tard. Pour répondre à la deuxième question, nous allons avoir besoin d'un logiciel...

## Et un logiciel ?

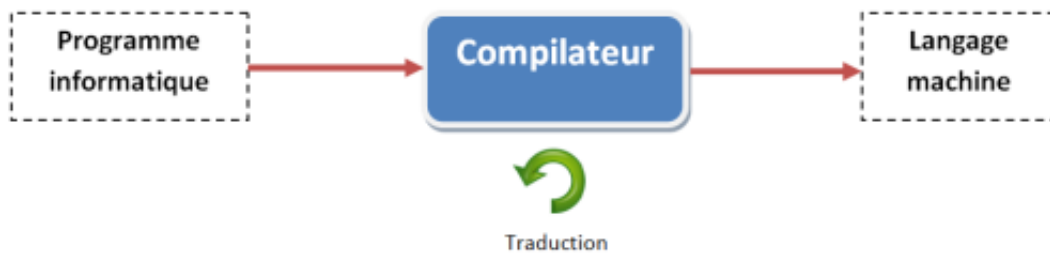
Bon, je ne vais pas vous faire le détail de ce qu'est un logiciel, vous savez sans aucun doute de quoi il s'agit. Ce n'est autre qu'un programme informatique exécuté sur un ordinateur. Oui, pour programmer la carte Arduino, nous allons utiliser un programme ! En fait, il va s'agir d'un **compilateur**. Alors qu'est-ce que c'est exactement ?

### Un compilateur

En informatique, ce terme désigne un logiciel qui est capable de traduire un langage informatique, ou plutôt un programme utilisant un langage informatique, vers un langage plus approprié afin que la machine qui va le lire puisse le comprendre. C'est un peu comme si le patron anglais d'une firme Chinoise donnait des instructions en anglais à l'un de ses ouvriers chinois. L'ouvrier ne pourrait comprendre ce qu'il doit faire. Pour cela, il a besoin que l'on traduise ce que lui dit son patron. C'est le rôle du



**traducteur.** Le compilateur va donc traduire les instructions du programme précédent, écrites en langage texte, vers un langage dit “machine”. Ce langage utilise uniquement des 0 et des 1. Nous verrons plus tard pourquoi. Cela pourrait être imagé de la façon suivante :



Donc, pour traduire le langage texte vers le langage machine (avec des 0 et des 1), nous aurons besoin de ce fameux compilateur. Et pas n'importe lequel, il faut celui qui soit capable de traduire le langage texte Arduino vers le langage machine Arduino. Et oui, sinon rien ne va fonctionner. Si vous mettez un traducteur Français vers Allemand entre notre patron anglais et son ouvrier chinois, ça ne fonctionnera pas mieux que s'ils discutaient directement. Vous comprenez ?

Et pourquoi on doit utiliser un traducteur, on peut pas simplement apprendre le langage machine directement ?

Comment dire... non ! Non parce que le langage machine est quasiment impossible à utiliser tel quel. Par exemple, comme il est composé de 0 et de 1, si je vous montre ça : “0001011100111010101000111”, vous serez incapable, tout comme moi, de dire ce que cela signifie ! Et même si je vous dis que la suite “01000001” correspond à la lettre “A”, je vous donne bien du courage pour coder rien qu’une phrase ! 😊 Bref, oubliez cette idée. C’est quand même plus facile d’utiliser des mots anglais (car oui nous allons être obligé de faire un peu d’anglais pour programmer, mais rien de bien compliqué rassurez-vous) que des suites de 0 et de 1. Vous ne croyez pas ?

### ***Envoyer le programme dans la carte***

Là, je ne vais pas vous dire grand chose car c’est l’environnement de développement qui va gérer tout ça. Nous n’aurons qu’à apprendre comment utiliser ce dernier et il se débrouillera tout seul pour envoyer le programme dans la carte. Nah ! Nous n’aurons donc qu’à créer le programme sans nous soucier du reste.

---

## **Pourquoi choisir Arduino ?**

### **Que va-t-on faire avec ?**

Avec Arduino, nous allons commencer par apprendre à programmer puis à utiliser des composants électroniques. Au final, nous saurons créer des systèmes électroniques plus ou moins complexes. Mais ce n’est pas tout...

### ***D’abord, Arduino c’est...***

... une carte électronique programmable et un logiciel gratuit :



### *Mais aussi*

- Un prix dérisoire étant donné l'étendue des applications possibles. On comptera 20 euros pour la carte que l'on va utiliser dans le cours. Le logiciel est fourni gratuitement !
- Une compatibilité sous toutes les plateformes, à savoir : Windows, Linux et MacOS.
- Une communauté ultra développée ! Des milliers de forums d'entraide, de présentations de projets, de propositions de programmes et de bibliothèques, ...
- Un site en anglais [arduino.cc](https://arduino.cc) et un autre en français [arduino.cc/fr](https://arduino.cc/fr) où vous trouverez tout de la référence Arduino, le matériel, des exemples d'utilisations, de l'aide pour débiter, des explications sur le logiciel et le matériel, etc.
- Une liberté quasi absolue. Elle constitue en elle-même deux choses :
  - Le logiciel : gratuit et open source, développé en Java, dont la simplicité d'utilisation relève du savoir cliquer sur la souris
  - Le matériel : cartes électroniques dont les schémas sont en libre circulation sur internet

Cette liberté a une condition : le nom « Arduino » ne doit être employé que pour les cartes « officielles ». En somme, vous ne pouvez pas fabriquer votre propre carte sur le modèle Arduino et lui assigner le nom « Arduino ».

### *Et enfin, les applications possibles*

Voici une liste non exhaustive des applications possibles réalisées grâce à Arduino :

- contrôler des appareils domestiques
- donner une "intelligence" à un robot
- réaliser des jeux de lumières
- permettre à un ordinateur de communiquer avec une carte électronique et différents capteurs
- télécommander un appareil mobile (modélisme)
- etc...

Il y a tellement d'autres infinités d'utilisations, vous pouvez simplement chercher sur votre moteur de recherche préféré ou sur Youtube le mot "Arduino" pour découvrir les milliers de projets réalisés avec !

### *Arduino dans ce tutoriel*

Je vais quand même rappeler les principaux objectifs de ce cours. Nous allons avant tout découvrir Arduino dans son ensemble et apprendre à l'utiliser. Dans un premier temps, il s'agira de vous présenter ce qu'est Arduino, comment cela fonctionne

globalement, pour ensuite entrer un peu plus dans le détail. Nous allons alors apprendre à utiliser le langage Arduino pour pouvoir créer des programmes très simple pour débiter. Nous enchaînerons ensuite avec les différentes fonctionnalités de la carte et ferons de petits TP qui vous permettront d'assimiler chaque notion abordée. Dès lors que vous serez plutôt à l'aise avec toutes les bases, nous nous rapprocherons de l'utilisation de composants électroniques plus ou moins complexes et finirons par un plus "gros" TP alliant la programmation et l'électronique. De quoi vous mettre de l'eau à la bouche ! 😊

## Arduino à l'école ?

Pédagogiquement, Arduino a aussi pas mal d'atout. En effet, ses créateurs ont d'abord pensé ce projet pour qu'il soit facile d'accès. Il permet ainsi une très bonne approche de nombreux domaines et ainsi d'apprendre plein de choses assez simplement.

### Des exemples

Voici quelques exemples d'utilisation possible :

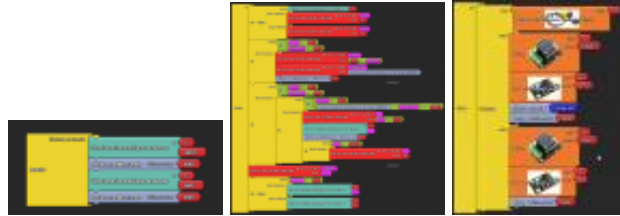
- Simuler le fonctionnement des portes logiques
- Permettre l'utilisation de différents capteurs
- Mettre en œuvre et faciliter la compréhension d'un réseau informatique
- Se servir d'Arduino pour créer des maquettes animées montrant le fonctionnement des collisions entre les plaques de la croûte terrestre, par exemple 🌐
- Donner un exemple concret d'utilisation des matrices avec un clavier alphanumérique 16 touches ou plus
- Être la base pour des élèves ayant un TPE à faire pour le BAC
- ...

De plus, énormément de ressources et tutoriels (mais souvent en anglais) se trouvent sur internet, ce qui offre une autonomie particulière à l'apprenant.

### Des outils existant !

Enfin, pour terminer de vous convaincre d'utiliser Arduino pour découvrir le monde merveilleux de l'embarqué, il existe différents outils qui puissent être utilisés avec Arduino. Je vais en citer deux qui me semblent être les principaux : [Ardublock](#) est un outil qui se greffe au logiciel Arduino et qui permet de programmer avec des blocs. Chaque bloc est une instruction. On peut aisément faire des programmes avec cet outil et même des plutôt complexes. Cela permet par exemple de se concentrer sur ce que l'on doit faire avec Arduino et non se concentrer sur Arduino pour ensuite ce que l'on doit comprendre avec. Citons entre autre la simulation de porte logique : plutôt créer des programmes rapidement sans connaître le langage pour comprendre plus facilement comment fonctionne une porte logique. Et ce n'est qu'un exemple. Car cela permet aussi de permettre à de jeunes enfants de commencer à programmer sans de trop grandes complications.





*Exemple de programmes réalisés avec Ardublock*

Processing est une autre plateforme en lien avec Arduino. Là il n'y a pas de matériel, uniquement un logiciel. Il permet entre autre de créer des interfaces graphiques avec un langage de programmation très similaire à celui d'Arduino. Par contre, cela demande un niveau un peu plus élevé pour pouvoir l'utiliser, même si cela reste simple dans l'ensemble.



*Voilà un exemple de ce que j'avais réalisé avec Processing pour faire communiquer mon ordinateur avec ma carte Arduino*

J'espère avoir été assez convaincant afin que vous franchissiez le pas et ayez du plaisir à apprendre ! 😊

## Les cartes Arduino

Le matériel que j'ai choisi d'utiliser tout au long de ce cours n'a pas un prix excessif et, je l'ai dit, tourne aux alentours de 25 € TTC. Il existe plusieurs magasins en lignes et en boutiques qui vendent des cartes Arduino. Je vais vous en donner quelques-uns, mais avant, il va falloir différencier certaines choses.

### Les fabricants

Le projet Arduino est libre et les schémas des cartes circulent librement sur internet. D'où la mise en garde que je vais faire : il se peut qu'un illustre inconnu fabrique lui même ses cartes Arduino. Cela n'a rien de mal en soi, s'il veut les commercialiser, il peut. Mais s'il est malhonnête, il peut vous vendre un produit défectueux. Bien sûr, tout le monde ne cherchera pas à vous arnaquer. Mais la prudence est de rigueur. Faites donc attention où vous achetez vos cartes.

### Les types de cartes

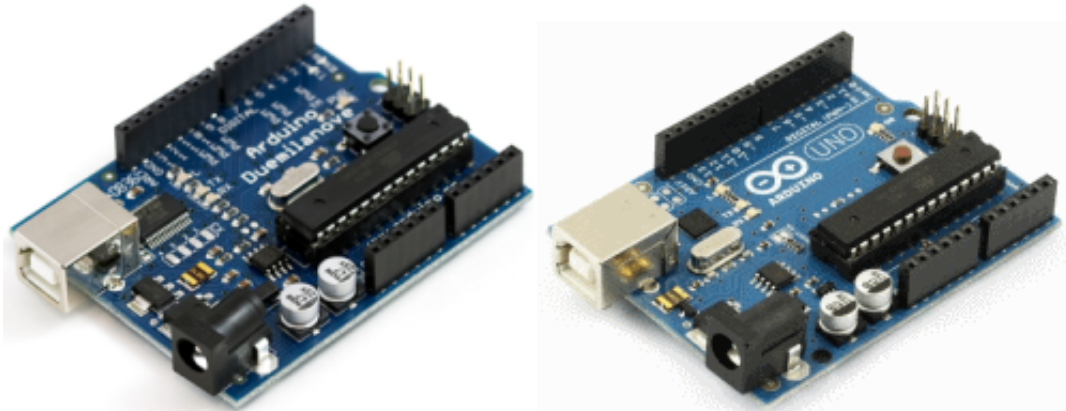
Il y a trois types de cartes :

- Lesdites « officielles » qui sont fabriquées en Italie par le fabricant officiel : *Smart Projects*
- Lesdits « compatibles » qui ne sont pas fabriqués par *Smart Projects*, mais qui sont totalement compatibles avec les Arduino officielles.
- Les « autres » fabriquées par diverse entreprise et commercialisées sous un nom différent (Freeduino, Seeduino, Femtoduino, ...).



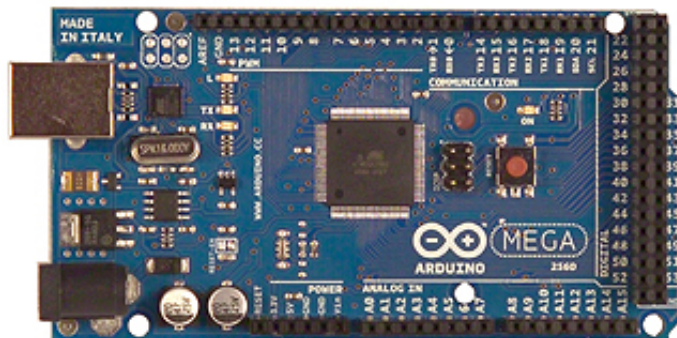
## Les différentes cartes

Des cartes Arduino il en existe beaucoup ! Voyons celles qui nous intéressent... **La carte Uno et Duemilanove** Nous choisirons d'utiliser la carte portant le nom de « Uno » ou « Duemilanove ». Ces deux versions sont presque identiques.



*Carte Arduino “Duemilanove” et “Uno” avec laquelle nous allons travailler*

**La carte Mega** La carte Arduino Mega est une autre carte qui offre toutes les fonctionnalités de la carte précédente, mais avec des fonctionnalités supplémentaires. On retrouve notamment un nombre d'entrées et de sorties plus important ainsi que plusieurs liaisons séries. Bien sûr, le prix est plus élevé : > 40 € !



*Carte Arduino “Mega”*

**Les autres cartes** Il existe encore beaucoup d'autres cartes, je vous laisse vous débrouiller pour trouver celle qui conviendra à vos projets. Cela dit, je vous conseil dans un premier temps d'utiliser la carte Arduino Uno ou Duemilanove d'une part car elle vous sera largement suffisante pour débiter et d'autre part car c'est avec celle-ci que nous présentons le cours.

## Où acheter ?

Il existe sur le net une multitude de magasins qui proposent des cartes Arduino. Pour consulter la liste de ces magasins, rien de plus simple, il suffit de vous rendre sur le forum dédié :

[Les meilleurs boutiques d'électronique](#) Cliquez-ici

J'ai vu des cartes officielles “édition SMD/CMS”. Ça à l'air bien aussi, c'est quoi la

différence ? Je peux m'en servir ?

Il n'y a pas de différence ! enfin presque... "SMD" signifie **Surface Mount Device**, en français on appelle ça des "CMS" pour **Composants Montés en Surface**. Ces composants sont soudés directement sur le cuivre de la carte, il ne la traverse pas comme les autres. Pour les cartes Arduino, on retrouve le composant principal en édition SMD dans ces cartes. La carte est donc la même, aucune différence pour le tuto. Les composants sont les mêmes, seule l'allure "physique" est différente. Par exemple, ci-dessus la "Mega" est en SMD et la Uno est "classique".

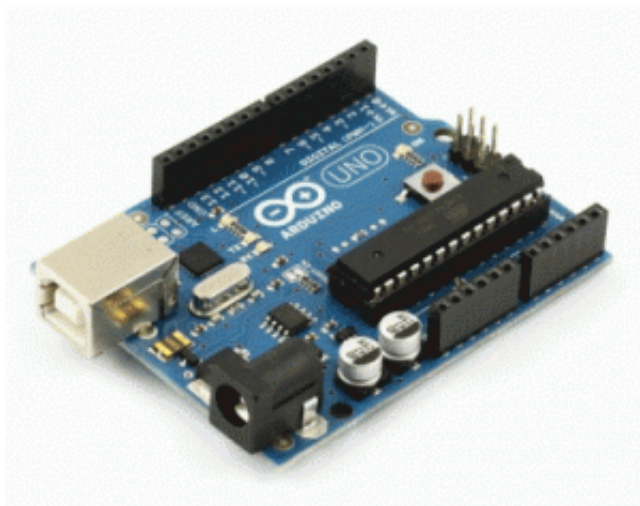
## Liste d'achat

Tout au long du cours, nous allons utiliser du matériel en supplément de la carte. Rassurez-vous le prix est bien moindre. Je vous donne cette liste, cela vous évitera d'acheter en plusieurs fois. Vous allez devoir me croire sur parole sur leur intérêt. Nous découvrirons comment chaque composant fonctionne et comment les utiliser tout au long du tutoriel. 😊

Attention, cette liste ne contient que les composants en quantités minimales strictes. Libre à vous de prendre plus de LED et de résistances par exemple (au cas où vous en perdriez ou détruisez...). Pour ce qui est des prix, j'ai regardé sur différents sites grands publics (donc pas Farnell par exemple), ils peuvent donc paraître plus élevé que la normale dans la mesure où ces sites amortissent moins sur des ventes à des clients fidèles qui prennent tout en grande quantité...

Avant que j'oublie, quatres éléments n'apparaîtront pas dans la liste et sont indispensables :

**Une Arduino Uno ou Duemilanove**

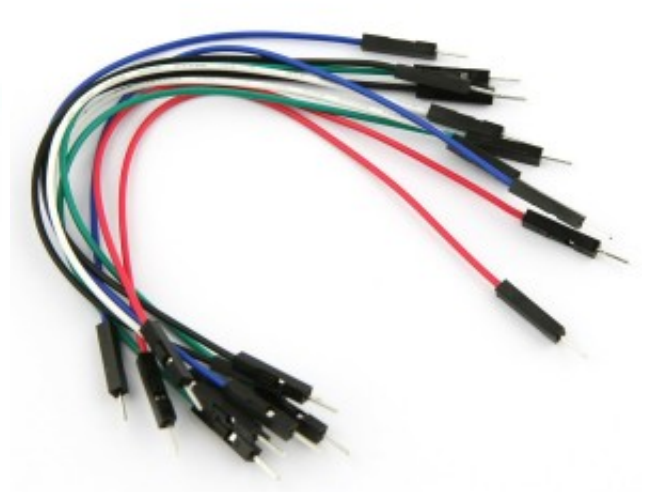
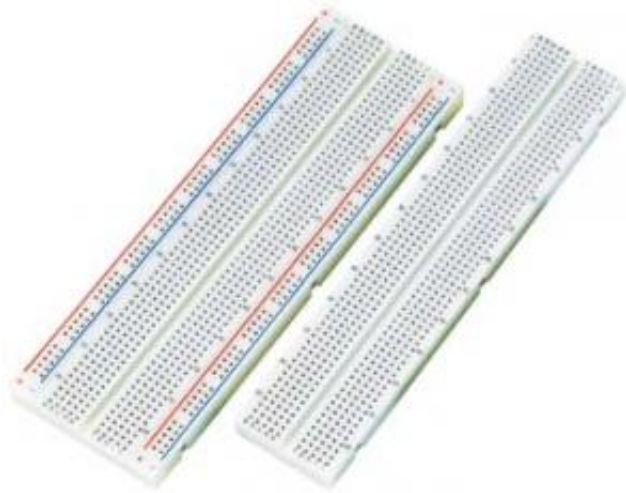


**Une BreadBoard (plaque d'essai)**

**Un câble USB A mâle/B mâle**








**Un lot de fils pour brancher le tout !**



## Liste Globale

Voici donc la liste du matériel nécessaire pour suivre le cours. Libre à vous de tout acheter ou non.

Liste incomplète, le tutoriel n'est pas terminé ! Mais elle suffit pour suivre les chapitres en ligne.

Désignation	Quantité	Photo	Description
LED rouge	7		Ce composant est une sorte de lampe un peu spécial. Nous nous en servons principalement pour faire de la signalisation.
LED verte	3		
LED jaune (ou orange)	2		
Résistance (entre 220 et 470 Ohm)	10		La résistance est un composant de base qui s'oppose au passage du courant. On s'en sert pour limiter des courants maximums mais aussi pour d'autres choses.
Résistance (entre 2.2 et 4.7 kOhm)	2		
Résistance (10 kOhm)	2		
Bouton Poussoir	2		Un bouton poussoir sert à faire passer le courant lorsqu'on appuie dessus ou au contraire garder le circuit "éteint" lorsqu'il est relâché.
Transistor (2N2222 ou BC547)	2		Le transistor sert à plein de chose. Il peut être utilisé pour faire de l'amplification (de courant ou de tension) mais aussi comme un interrupteur commandé électriquement.
Afficheur 7 segments (anode commune)	2		Un afficheur 7 segments est un ensemble de LEDs (cf. ci-dessus) disposées géométriquement pour afficher des chiffres.



Décodeur BCD  
(MC14543) 1



Le décodeur BCD (Binaire Codé Décimal) permet piloter des afficheurs 7 segments en limitant le nombre de fils de données (4 au lieu de 7).

Condensateur  
(10 nF) 2



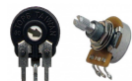
Le condensateur est un composant de base. Il sert à plein de chose. On peut se le représenter comme un petit réservoir à électricité.

Condensateur  
1000  $\mu$ F 1



Celui-ci est un plus gros réservoir que le précédent

Potentiomètre  
linéaire (10  
kOhm) 1



Le potentiomètre est une résistance que l'on peut faire varier manuellement.

LED RVB 1



Une LED RVB (Rouge Vert Bleu) est une LED permettant de mélanger les couleurs de bases pour en créer d'autres.

Écran LCD  
alphanumérique 1



L'écran LCD alphanumérique permet d'afficher des caractères tels que les chiffres et les lettres. Il va apporter de l'interactivité à vos projets les plus fous !

## Les revendeurs

Je vous ai déjà donné le lien, vous pourrez trouver ces composants chez les revendeurs listés dans ce sujet du forum :

[Les meilleurs boutiques d'électronique](#) Cliquez-ici

## Les kits

Enfin, il existe des kits tout prêts chez certains revendeurs. Nous n'en conseillerons aucun pour plusieurs raisons. Tout d'abord, pour ne pas faire trop de publicité et rester conforme avec la charte du site. Ensuite, car il est difficile de trouver un kit "complet". Ils ont tous des avantages et des inconvénients mais aucun (au moment de la publication de ces lignes) ne propose absolument tous les composants que nous allons utiliser. Nous ne voulons donc pas que vous reveniez vous plaindre sur les forums car nous vous aurions fait dépenser votre argent inutilement !

Cela étant dit, merci de **ne pas nous spammer de MP** pour que l'on donne notre avis sur tel ou tel kit ! Usez des forums pour cela, il y a toujours quelqu'un qui sera là pour vous aider. Et puis nous n'avons pas les moyens de tous les acheter et tester leur qualité !

---

## [Arduino 102] Quelques bases élémentaires

En attendant que vous achetiez votre matériel, je vais vous présenter les bases de



l'électronique et de la programmation. Cela vous demandera tout de même une bonne concentration pour essayer de comprendre des concepts pas évidents en soit.

La première partie de ce chapitre ne fait que reprendre quelques éléments du [cours sur l'électronique](#), que vous pouvez consulter pour de plus amples explications. 😊

---

## Le courant, la tension et la masse

Pour faire de l'électronique, il est indispensable de connaître sur le bout des doigts ce que sont les grandeurs physiques. Alors, avant de commencer à voir lesquelles on va manipuler, voyons un peu ce qu'est une grandeur physique. Une **grandeur physique** est quelque chose qui se mesure. Plus précisément il s'agit d'un élément mesurable, grâce à un appareil ou dispositif de mesure, régit par les lois de la physique. Par exemple, la pression atmosphérique est une grandeur physique, ou bien la vitesse à laquelle circule une voiture en est aussi une. En électronique cependant, nous ne mesurons pas ces grandeurs-là, nous avons nos propres grandeurs, qui sont : **le courant et la tension**.

### La source d'énergie

L'énergie que l'on va manipuler (courant et tension) provient d'un **générateur**. Par exemple, on peut citer : la pile électrique, la batterie électrique, le secteur électrique. Cette énergie qui est fournie par le générateur est restituée à un ou plusieurs **récepteurs**. Le récepteur, d'après son nom, reçoit de l'énergie. On dit qu'il la **consomme**. On peut citer pour exemples : un chauffage d'appoint, un sèche-cheveux, une perceuse.

### Le courant électrique

#### Charges électriques

Les charges électriques sont des grandeurs physiques mesurables. Elles constituent la matière en elle même. Dans un atome, qui est élément primaire de la matière, il y a trois charges électriques différentes : les charges **positives**, **négatives** et **neutres** appelées respectivement **protons**, **électrons** et **neutrons**. Bien, maintenant nous pouvons définir le courant qui est **un déplacement ordonné de charges électriques**.

### Conductibilité des matériaux

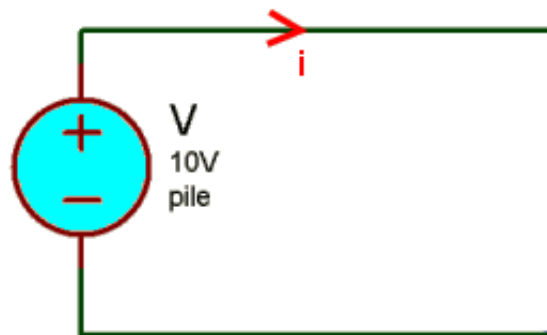
La notion de conductibilité est importante à connaître, car elle permet de comprendre pas mal de phénomènes. On peut définir la **conductibilité** comme étant la capacité d'un matériau à se laisser traverser par un courant électrique. De ces matériaux, on peut distinguer quatre grandes familles :

- les isolants : leurs propriétés empêchent le passage d'un courant électrique (plastique, bois, verre)
- les semi-conducteurs : ce sont des isolants, mais qui laissent passer le courant dès lors que l'on modifie légèrement leur structure interne (diode, transistor, LED)

- les conducteurs : pour eux, le courant peut passer librement à travers tout en opposant une faible résistance selon le matériau utilisé (or, cuivre, métal en général)
- les supraconducteurs : ce sont des types bien particuliers qui, à une température extrêmement basse, n'opposent quasiment aucune résistance au passage d'un courant électrique

## Sens du courant

Le courant électrique se déplace selon un sens de circulation. Un générateur électrique, par exemple une pile, produit un courant. Et bien ce courant va circuler du pôle positif vers le pôle négatif de la pile, si et seulement si ces deux pôles sont reliés entre eux par un fil métallique ou un autre conducteur. Ceci, c'est le **sens conventionnel** du courant. On note le courant par une flèche qui indique le sens conventionnel de circulation du courant :



*Indication du sens du courant*

## Intensité du courant

L'intensité du courant est la vitesse à laquelle circule ce courant. Tandis que le courant est un déplacement ordonné de charges électriques. Voilà un point à ne pas confondre.

On mesure la vitesse du courant, appelée **intensité**, en **Ampères** (noté **A**) avec un Ampèremètre. En général, en électronique de faible puissance, on utilise principalement le milli-Ampère (**mA**) et le micro-Ampère (**μA**), mais jamais bien au-delà. C'est tout ce qu'il faut savoir sur le courant, pour l'instant.

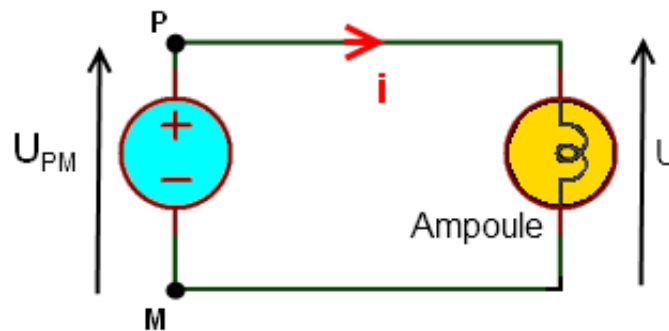
## Tension

Autant le courant se déplace, ou du moins est un déplacement de charges électriques, autant la **tension** est quelque chose de **statique**. Pour bien définir ce qu'est la tension, sachez qu'on la compare à la pression d'un fluide. Par exemple, lorsque vous arrosez votre jardin (ou une plante, comme vous préférez) avec un tuyau d'arrosage et bien dans ce tuyau, il y a une certaine pression exercée par l'eau fournie par le robinet. Cette pression permet le déplacement de l'eau dans le tuyau, donc créer un courant. Mais si la pression n'est pas assez forte, le courant ne sera lui non plus pas assez fort. Pour preuve, vous n'avez qu'à pincer le tuyau pour constater que le courant ne circule plus. On appelle ce "phénomène de pression" : la **tension**. Je n'en dis pas plus car se serait

vous embrouiller. 😊

## Notation et unité

La tension est mesurée en **Volts** (notée **V**) par un Voltmètre. On utilise principalement le Volt, mais aussi son sous-multiple qui est le milli-Volt (**mV**). On représente la tension, d'une pile par exemple, grâce à une flèche orientée toujours dans le sens du courant aux bornes d'un générateur et toujours opposée au courant, aux bornes d'un récepteur :



*Fléchage de la tension*

## La différence de potentiel

Sur le schéma précédent, on a au point M une tension de 0V et au point P, une tension de 5V. Prenons notre Voltmètre et mesurons la tension aux bornes du générateur. La borne COM du Voltmètre doit être reliée au point M et la borne “+” au point P. Le potentiel au point P, soustrait par le potentiel au point M vaut :  $U_P - U_M = 5 - 0 = 5V$ . On dit que la **différence de potentiel** entre ces deux points est de 5V. Cette mesure se note donc :  $U_{PM}$ . Si on inverse le sens de branchement du Voltmètre, la borne “+” est reliée au point M et la borne COM au point P. La mesure que l'on prend est la différence de tension (= potentiel) entre le point M et le point P :  $U_M - U_P = 0 - 5 = -5V$ . Cette démonstration un peu surprenante vient du fait que la masse est arbitraire.

## La masse

Justement, parlons-en ! La **masse** est, en électronique, un point de référence.

## Notion de référentiel

Quand on prend une mesure, en général, on la prend entre deux points bien définis. Par exemple, si vous vous mesurez, vous prenez la mesure de la plante de vos pieds jusqu'au sommet de votre tête. Si vous prenez la plante de vos pieds pour référence (c'est-à-dire le chiffre zéro inscrit sur le mètre), vous lirez 1m70 (par exemple). Si vous inversez, non pas la tête, mais le mètre et que le chiffre zéro de celui-ci se retrouve donc au sommet de votre tête, vous serez obligé de lire la mesure à -1m70. Et bien, ce chiffre zéro est la référence qui vous permet de vous mesurer. En électronique, cette référence existe, on l'appelle la **masse**.

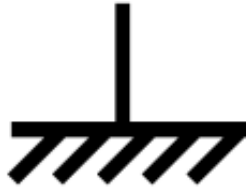
## Qu'est ce que c'est ?

La masse, et bien c'est un référentiel. En électronique on voit la masse d'un montage

comme étant le zéro Volt (0V). C'est le point qui permet de mesurer une bonne partie des tensions présentes dans un montage.

## Représentation et notation

Elle se représente par ce symbole, sur un schéma électronique :



*Symbole de la masse*

Vous ne le verrez pas souvent dans les schémas de ce cours, pour la simple raison qu'elle est présente sur la carte que l'on va utiliser sous un autre nom : **GND**. GND est un diminutif du terme anglais "Ground" qui veut dire terre/sol. Donc, pour nous et tous les montages que l'on réalisera, ce sera le point de référence pour la mesure des tensions présentes sur nos circuits et le zéro Volt de tous nos circuits.

## Une référence arbitraire

Pour votre culture, sachez que la masse est quelque chose d'arbitraire. Je l'ai bien montré dans l'exemple au début de ce paragraphe. On peut changer l'emplacement de cette référence et, par exemple, très bien dire que le 5V est la masse. Ce qui aura pour conséquence de modifier l'ancienne masse en -5V.

---

## La résistance et sa loi !

En électronique il existe plein de composants qui ont chacun une ou plusieurs fonctions. Nous allons voir quels sont ces composants dans le cours, mais pas tout de suite. Car, maintenant, on va aborder la résistance qui est LE composant de base en électronique.

## Présentation

C'est le composant le plus utilisé en électronique. Sa principale fonction est de réduire l'intensité du courant (mais pas uniquement). Ce composant se présente sous la forme d'un petit boîtier fait de divers matériaux et repéré par des anneaux de couleur indiquant la valeur de cette dernière. Photo de résistance :



*Photo de résistance*

## Symbole

Le symbole de la résistance ressemble étrangement à la forme de son boîtier :



Symbole de la résistance

## Loi d'ohm

Le courant traversant une résistance est régi par une formule assez simple, qui se nomme **la loi d'ohm** :

$$I = \frac{U}{R}$$

- **I** : intensité qui traverse la résistance en Ampères, notée *A*
- **U** : tension aux bornes de la résistance en Volts, notée *V*
- **R** : valeur de la résistance en Ohms, notée  $\Omega$

En général, on retient mieux la formule sous cette forme :  $U = R * I$

## Unité

L'unité de la résistance est l'**ohm**. On le note avec le symbole grec oméga majuscule :  $\Omega$  .

## Le code couleur

La résistance possède une suite d'anneaux de couleurs différentes sur son boîtier. Ces couleurs servent à expliciter la valeur de la résistance sans avoir besoin d'écrire en chiffre dessus (car vous avez déjà essayé d'écrire sur un cylindre 😊 ?) Le premier anneau représente le chiffre des centaines, le second celui des dizaines et le troisième celui des unités. Enfin, après un petit espace vient celui du coefficient multiplicateur. Avec ses quatre anneaux et un peu d'entraînement vous pouvez alors deviner la valeur de la résistance en un clin d'oeil 😊 . Ce tableau vous permettra de lire ce code qui correspond à la valeur de la résistance :

### Couleur Chiffre Coefficient multiplicateur Puissance Tolérance

Noir	0	1	$10^0$	-
Brun	1	10	$10^1$	$\pm 1 \%$
Rouge	2	100	$10^2$	$\pm 2 \%$
Orange	3	1000	$10^3$	-
Jaune	4	10 000	$10^4$	-
Vert	5	100 000	$10^5$	$\pm 0.5 \%$
Bleu	6	1 000 000	$10^6$	$\pm 0.25 \%$
Violet	7	10 000 000	$10^7$	$\pm 0.10 \%$
Gris	8	100 000 000	$10^8$	$\pm 0.05 \%$
	9	1 000 000 000	$10^9$	-
-	-	-	-	-
Or	0.1	0.1	$10^{-1}$	$\pm 5 \%$
Argent	0.01	0.01	$10^{-2}$	$\pm 10 \%$
(absent)	-	-	-	$\pm 20 \%$



---

# Le microcontrôleur

Nous avons déjà un peu abordé le sujet dans la présentation du cours. Je vous ai expliqué “brièvement” comment fonctionnait un programme et surtout ce que c’était ! 🤖  
Bon, dès à présent je vais rentrer un petit peu plus dans le détail en vous introduisant des notions basées sur le matériel étroitement lié à la programmation. Nous allons en effet aborder le microcontrôleur dans un niveau de complexité supérieur à ce que je vous avais introduit tout à l’heure. Ho, rien de bien insurmontable, soyez sans craintes.  
😊

## La programmation en électronique

Aujourd’hui, l’électronique est de plus en plus composée de composants numériques programmables. Leur utilisation permet de simplifier les schémas électroniques et par conséquent réduire le coût de fabrication d’un produit. Il en résulte des systèmes plus complexes et performants pour un espace réduit.

### *Comment programmer de l’électronique ?*

Pour faire de l’électronique programmée, il faut un ordinateur et un **composant programmable**. Il existe tout plein de variétés différentes de composants programmables, à noter : les microcontrôleurs, les circuits logiques programmables, ... Nous, nous allons programmer des microcontrôleurs. Mais à ce propos, vous ai-je dit qu’est ce que c’était qu’un microcontrôleur ?

Qu’est ce que c’est ?

Je l’ai dit à l’instant, le microcontrôleur est un composant électronique programmable. On le programme par le biais d’un ordinateur grâce à un langage informatique, souvent propre au type de microcontrôleur utilisé. Je n’entrerais pas dans l’utilisation poussée de ces derniers car le niveau est rudement élevé et la compréhension difficile. Voici une photo d’un microcontrôleur :



*Photo de microcontrôleur*

C'est donc le microcontrôleur qui va être le cerveau de la carte Arduino, pour en revenir à nos moutons. C'est lui que nous allons programmer. On aura le temps d'en rediscuter. Pour l'instant je veux uniquement vous présenter les éléments principaux qui le composent.

### ***Composition des éléments internes d'un micro-contrôleur***

Un microcontrôleur est constitué par un ensemble d'éléments qui ont chacun une fonction bien déterminée. Il est en fait constitué des mêmes éléments que sur la carte mère d'un ordinateur. Si l'on veut, c'est un ordinateur (sans écran, sans disque dur, sans lecteur de disque) dans un espace très restreint. Je vais vous présenter les différents éléments qui composent un microcontrôleur typique et uniquement ceux qui vont nous être utiles. **La mémoire** La mémoire du microcontrôleur sert à plusieurs choses. On peut aisément citer le stockage du programme et de données autres que le programme. Il en possède 5 types :

- La mémoire Flash: C'est celle qui contiendra le programme à exécuter (celui que vous allez créer!). Cette mémoire est effaçable et ré-inscriptible (c'est la même qu'une clé USB par exemple)
- RAM : c'est la mémoire dite "vive", elle va contenir les variables de votre programme. Elle est dite "volatile" car elle s'efface si on coupe l'alimentation du micro-contrôleur (comme sur un ordinateur).
- EEPROM : C'est le "disque dur" du microcontrôleur. Vous pourrez y enregistrer des infos qui ont besoin de survivre dans le temps, même si la carte doit être arrêtée et coupée de son alimentation. Cette mémoire ne s'efface pas lorsque l'on éteint le microcontrôleur ou lorsqu'on le reprogramme.
- Les registres : c'est un type particulier de mémoire utilisé par le processeur. Nous n'en parlerons pas tout de suite.
- La mémoire cache : c'est une mémoire qui fait la liaison entre les registres et la RAM. Nous n'en parlerons également pas tout de suite.

Pour plus de détails sur les mémoires utilisables dans vos programmes, une annexe en fin de tutoriel s'occupe de cela 😊

**Le processeur** C'est le composant principal du micro-contrôleur. C'est lui qui va exécuter le programme que nous lui donnerons à traiter. On le nomme souvent le CPU.  
**Diverses choses** Nous verrons plus en détail l'intérieur d'un micro-contrôleur, mais pas tout de suite, c'est bien trop compliqué. Je ne voudrais pas perdre la moitié des visiteurs en un instant ! 😊

## Fonctionnement

Avant tout, pour que le microcontrôleur fonctionne, il lui faut une alimentation ! Cette alimentation se fait en générale par du +5V. D'autres ont besoin d'une tension plus faible, du +3,3V (c'est le cas de la Arduino Due par exemple). En plus d'une alimentation, il a besoin d'un signal d'horloge. C'est en fait une succession de 0 et de 1 ou plutôt une succession de tension 0V et 5V. Elle permet en outre de cadencer le fonctionnement du microcontrôleur à un rythme régulier. Grâce à elle, il peut introduire la notion de temps en programmation. Nous le verrons plus loin. Bon, pour le moment, vous n'avez pas besoin d'en savoir plus. Passons à autre chose.

## Les bases de comptage (2,10 et 16)

### Les bases de comptage

On va apprendre à compter ? o\_O

Non, je vais simplement vous expliquer ce que sont les **bases de comptage**. C'est en fait un **système de numération** qui permet de compter en utilisant des caractères de numérations, on appelle ça des **chiffres**.

### Cas simple, la base 10

La base 10, vous la connaissez bien, c'est celle que l'on utilise tous les jours pour compter. Elle regroupe un ensemble de 10 chiffres : 0,1,2,3,4,5,6,7,8,9. Avec ces chiffres, on peut créer une infinité de nombres (ex : 42, 89, 12872, 14.56, 9.3, etc...). Cependant, voyons cela d'un autre œil...

- L'unité sera représenté par un chiffre multiplié par 10 à la puissance 0.
- La dizaine sera représenté par un chiffre multiplié par 10 à la puissance 1.
- La centaine sera représenté par un chiffre multiplié par 10 à la puissance 2.
- [...]
- Le million sera représenté par un chiffre multiplié par 10 à la puissance 6.
- etc...

En généralisant, on peut donc dire qu'un nombre (composé de chiffres) est la somme des chiffres multipliés par 10 à une certaine puissance. Par exemple, si on veut écrire 1024, on peut l'écrire :  $1 \times 1000 + 0 \times 100 + 2 \times 10 + 4 \times 1 = 1024$  ce qui est équivalent à écrire :  $1 \times 10^3 + 0 \times 10^2 + 2 \times 10^1 + 4 \times 10^0 = 1024$  Et bien c'est

ça, compter en base 10 ! Vous allez mieux comprendre avec la partie suivante.

## Cas informatique, la base 2 et la base 16

En informatique, on utilise beaucoup les bases 2 et 16. Elles sont composées des chiffres suivants :

- pour la **base 2** : les chiffres 0 et 1.
- pour la **base 16** : on retrouve les chiffres de la base 10, plus quelques lettres : 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

On appelle la base 2, la **base binaire**. Elle représente des états logiques 0 ou 1. Dans un signal numérique, ces états correspondent à des niveaux de tension. En électronique numérique, très souvent il s'agira d'une tension de 0V pour un état logique 0 ; d'une tension de 5V pour un état logique 1. On parle aussi de niveau HAUT ou BAS (in english : HIGH or LOW). Elle existe à cause de la conception physique des ordinateurs. En effet, ces derniers utilisent des millions de transistors, utilisés pour traiter des données binaires, donc deux états distincts uniquement (0 ou 1). Pour compter en base 2, ce n'est pas très difficile si vous avez saisi ce qu'est une base. Dans le cas de la base 10, chaque chiffre était multiplié par 10 à une certaine puissance en partant de la puissance 0. Et bien en base 2, plutôt que d'utiliser 10, on utilise 2. Par exemple, pour obtenir 11 en base 2 on écrira : 1011... En effet, cela équivaut à faire :  $1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$  soit :  $1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1$

Un chiffre en base 2 s'appelle un **bit**. Un regroupement de 8 bits s'appelle un **octet**. Ce vocabulaire est très important donc retenez-le !

La base 16, ou **base hexadécimale** est utilisée en programmation, notamment pour représenter des octets facilement. Reprenons nos bits. Si on en utilise quatre, on peut représenter des nombres de 0 (0000) à 15 (1111). Ça tombe bien, c'est justement la portée d'un nombre hexadécimal ! En effet, comme dit plus haut il va de 0 (0000 ou 0) à F (1111 ou 15), ce qui représente 16 "chiffres" en hexadécimal. Grâce à cela, on peut représenter "simplement" des octets, en utilisant juste deux chiffres hexadécimaux.

## Les notations

Ici, rien de très compliqué, je vais simplement vous montrer comment on peut noter un nombre en disant à quelle base il appartient.

- Base binaire : (10100010)<sub>2</sub>
- Base décimale : (162)<sub>10</sub>
- Base hexadécimale : (A2)<sub>16</sub>

A présent, voyons les différentes méthodes pour passer d'une base à l'autre grâce aux **conversions**.

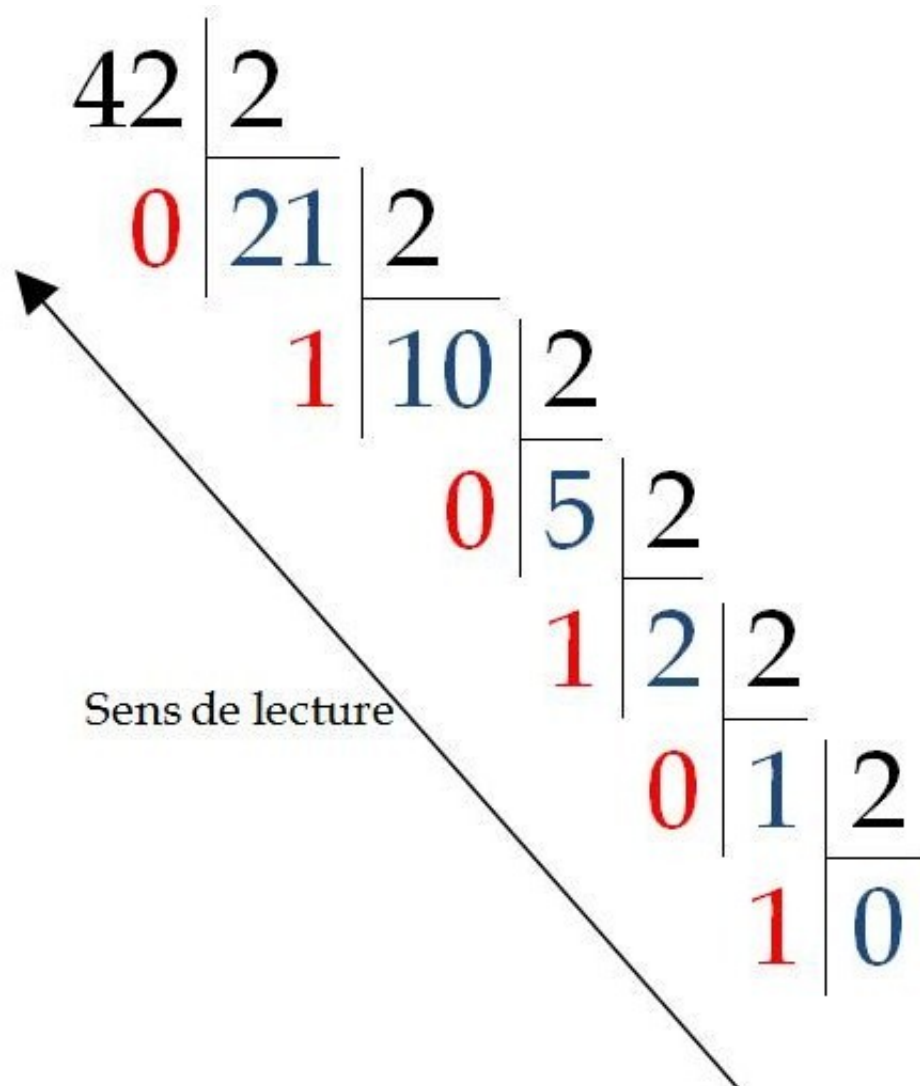
## Conversions

Souvent, on a besoin de convertir les nombres dans des bases différentes. On

retrouvera deux méthodes, bonnes à savoir l'une comme l'autre. La première vous apprendra à faire les conversions "à la main", vous permettant de bien comprendre les choses. La seconde, celle de la calculatrice, vous permettra de faire des conversions sans vous fatiguer.

### Décimale – Binaire

Pour convertir un nombre décimal (en base 10) vers un nombre binaire (en base 2, vous suivez c'est bien !), il suffit de savoir diviser par ... 2 ! Ça ira ? Prenez votre nombre, puis divisez le par 2. Divisez ensuite le quotient obtenu par 2... puis ainsi de suite jusqu'à avoir un quotient nul. Il vous suffit alors de lire les restes de bas en haut pour obtenir votre nombre binaire... Par exemple le nombre **42** s'écrira **101010** en binaire. Voilà un schéma de démonstration de cette méthode :



On garde les restes (en rouge) et on li le résultat de bas en haut.

### Binaire – Hexadécimal

La conversion de binaire à l'hexadécimal est la plus simple à réaliser. Tout d'abord, commencez à regrouper les bits par blocs de quatre en commençant par la droite. Si il n'y a pas assez de bits à gauche pour faire le dernier groupe de quatre, on rajoute des



zéros. Prenons le nombre 42, qui s'écrit en binaire, on l'a vu, **101010**, on obtiendra deux groupes de 4 bits qui seront **0010 1010**. Ensuite, il suffit de calculer bloc par bloc pour obtenir un chiffre hexadécimal en prenant en compte la valeur de chaque bit. Le premier bit, de poids faible (tout à droite), vaudra par exemple A ( $1 \times 8 + 0 \times 4 + 1 \times 2 + 0 \times 1 = 10$  : A en hexadécimal). Ensuite, l'autre bloc vaudra simplement 2 ( $0 \times 8 + 0 \times 4 + 1 \times 2 + 0 \times 1 = 2$ ). Donc 42 en base décimale vaut 2A en base hexadécimale, ce qui s'écrit aussi  $(42)_{10} = (2A)_{16}$ . Pour passer de hexadécimal à binaire, il suffit de faire le fonctionnement inverse en s'aidant de la base décimale de temps en temps. La démarche à suivre est la suivante :

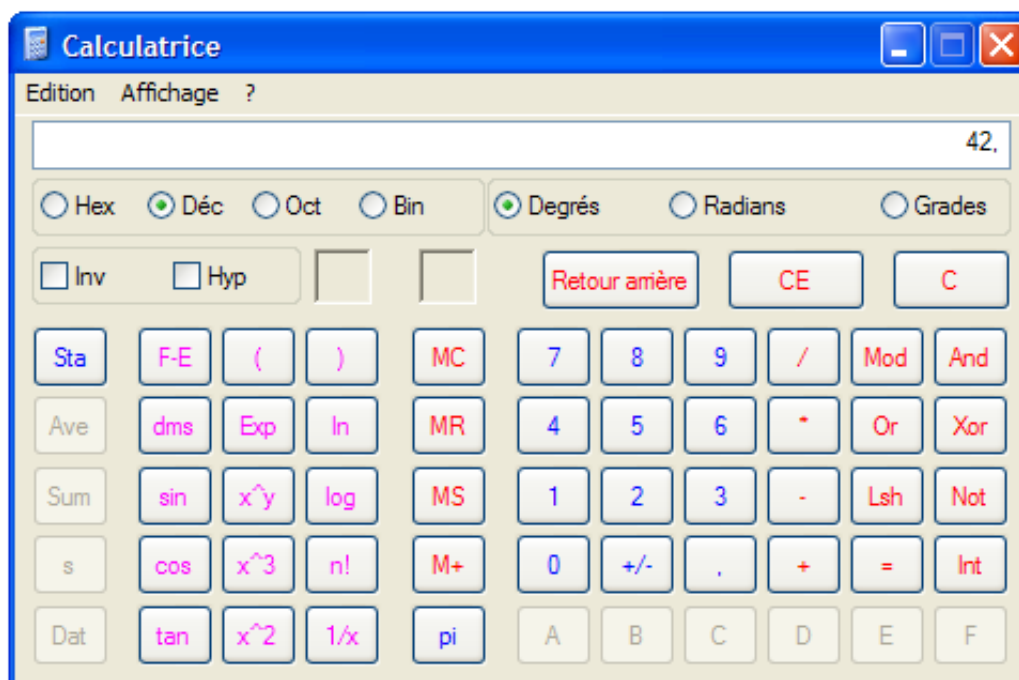
- Je sépare les chiffres un par un (on obtient 2 et A)
- Je "convertis" leurs valeurs en décimal (ce qui nous fait 2 et 10)
- Je met ces valeurs en binaire (et on a donc 0010 1010)

## Décimal – Hexadécimal

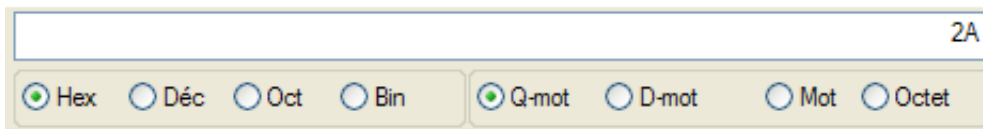
Ce cas est plus délicat à traiter, car il nécessite de bien connaître la table de multiplication par 16. 😊 Comme vous avez bien suivi les explications précédentes, vous comprenez comment faire ici... Mais comme je suis nul en math, je vous conseillerais de faire un passage par la base binaire pour faire les conversions !

## Méthode rapide

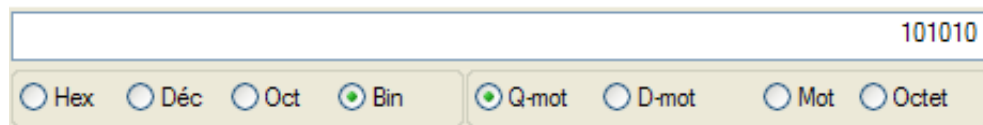
Pour cela, je vais dans *Démarrer / Tous les programmes / Accessoires / Calculatrice*. Qui a dit que j'étais fainéant ? :colere2:



Vous voyez en haut qu'il y a des options à cocher pour afficher le nombre entré dans la base que l'on veut. Présentement, je suis en base 10 (décimale – bouton *Déc*). Si je clique sur *Hex* :



Je vois que mon nombre **42** a été converti en : **2A**. Et maintenant, si je clique sur *Bin* :



Notre nombre a été converti en : **00101010** Oui, c'est vrai ça. Pour quoi on a pas commencé par expliquer ça ? Qui sait. 🤖

---

## [Arduino 103] Le logiciel

Afin de vous laisser un léger temps de plus pour vous procurer votre carte Arduino, je vais vous montrer brièvement comment se présente le logiciel Arduino.

---

### Installation

Il n'y a pas besoin d'installer le logiciel Arduino sur votre ordinateur puisque ce dernier est une version portable. Regardons ensemble les étapes pour préparer votre ordinateur à l'utilisation de la carte Arduino.

### Téléchargement

Pour télécharger le logiciel, il faut se rendre sur [la page de téléchargement du site arduino.cc](https://www.arduino.cc/en/software). Vous avez deux catégories :

- **Download** : Dans cette catégorie, vous pouvez télécharger la dernière version du logiciel. Les plateformes Windows, Linux et Mac sont supportées par le logiciel. **C'est donc ici que vous allez télécharger le logiciel.**
- **Previous IDE Versions** : Dans cette catégorie-là, vous avez toutes les versions du logiciel, sous les plateformes précédemment citées, depuis le début de sa création.

### Sous Windows

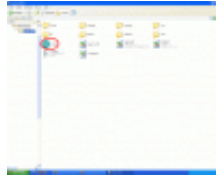
Pour moi ce sera sous Windows. Je clique sur le lien Windows et le fichier apparaît :



*Téléchargement du logiciel Arduino*

Une fois que le téléchargement est terminé, vous n'avez plus qu'à décompresser le

fichier avec un utilitaire de décompression (7-zip, WinRar, ...). A l'intérieur du dossier se trouvent quelques fichiers et l'exécutable du logiciel :



*Exécutable du logiciel Arduino*

## Mac os

Cliquez sur le lien Mac OS. Un fichier **.dmg** apparaît. Enregistrez-le.



*Téléchargement sous Mac os*

Double-cliquez sur le fichier **.dmg** :



*Contenu du téléchargement*

On y trouve l'application Arduino (**.app**), mais aussi le driver à installer (**.mpkg**). Procédez à l'installation du driver puis installez l'application en la glissant dans le raccourci du dossier "Applications" qui est normalement présent sur votre ordinateur.

## Sous Linux

Rien de plus simple, en allant dans la logithèque, recherchez le logiciel "Arduino". Sinon vous pouvez aussi passer par la ligne de commande:

```
1 $ sudo apt-get install arduino
```

Plusieurs dépendances seront installées en même temps.

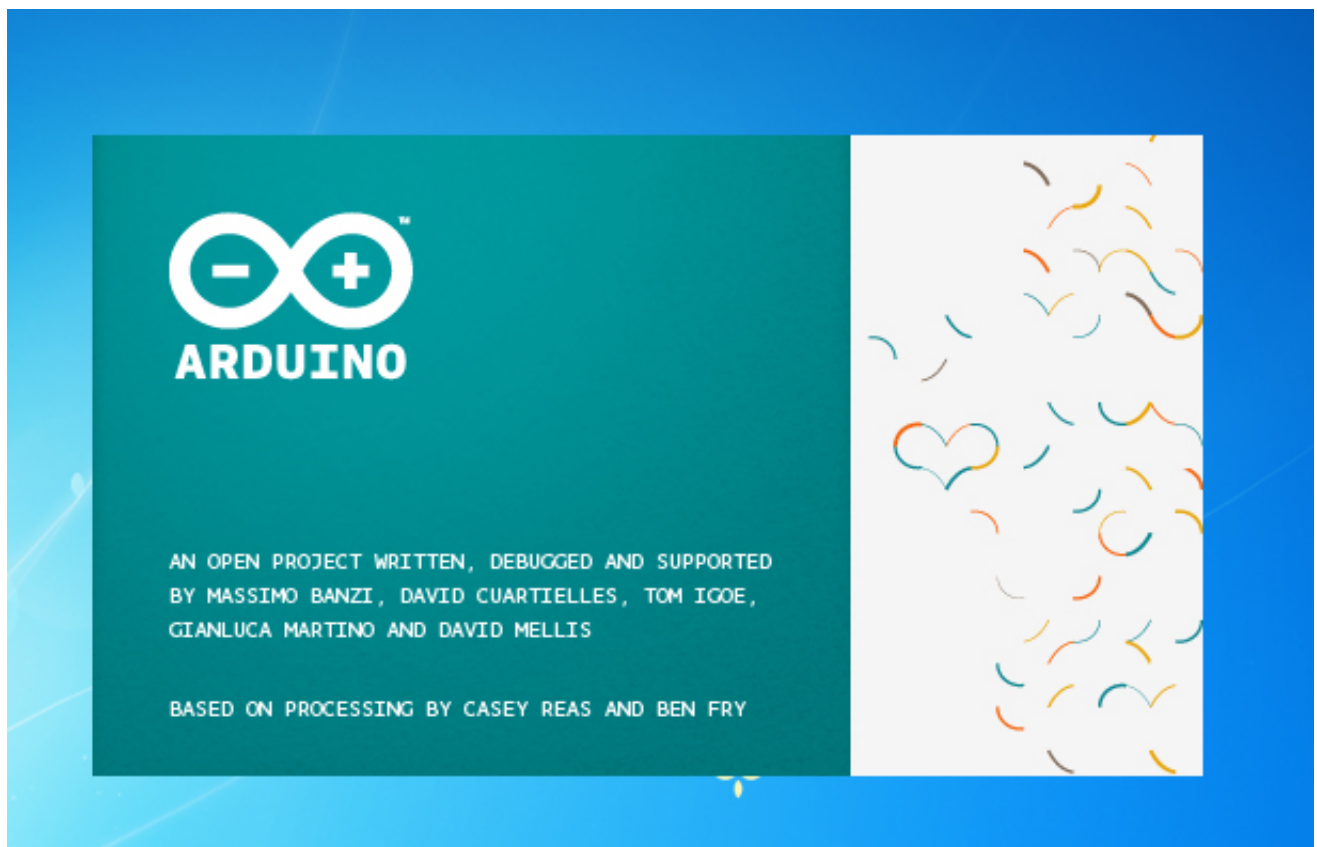
Je rajoute [un lien](#) qui vous mènera vers la page officielle.

---

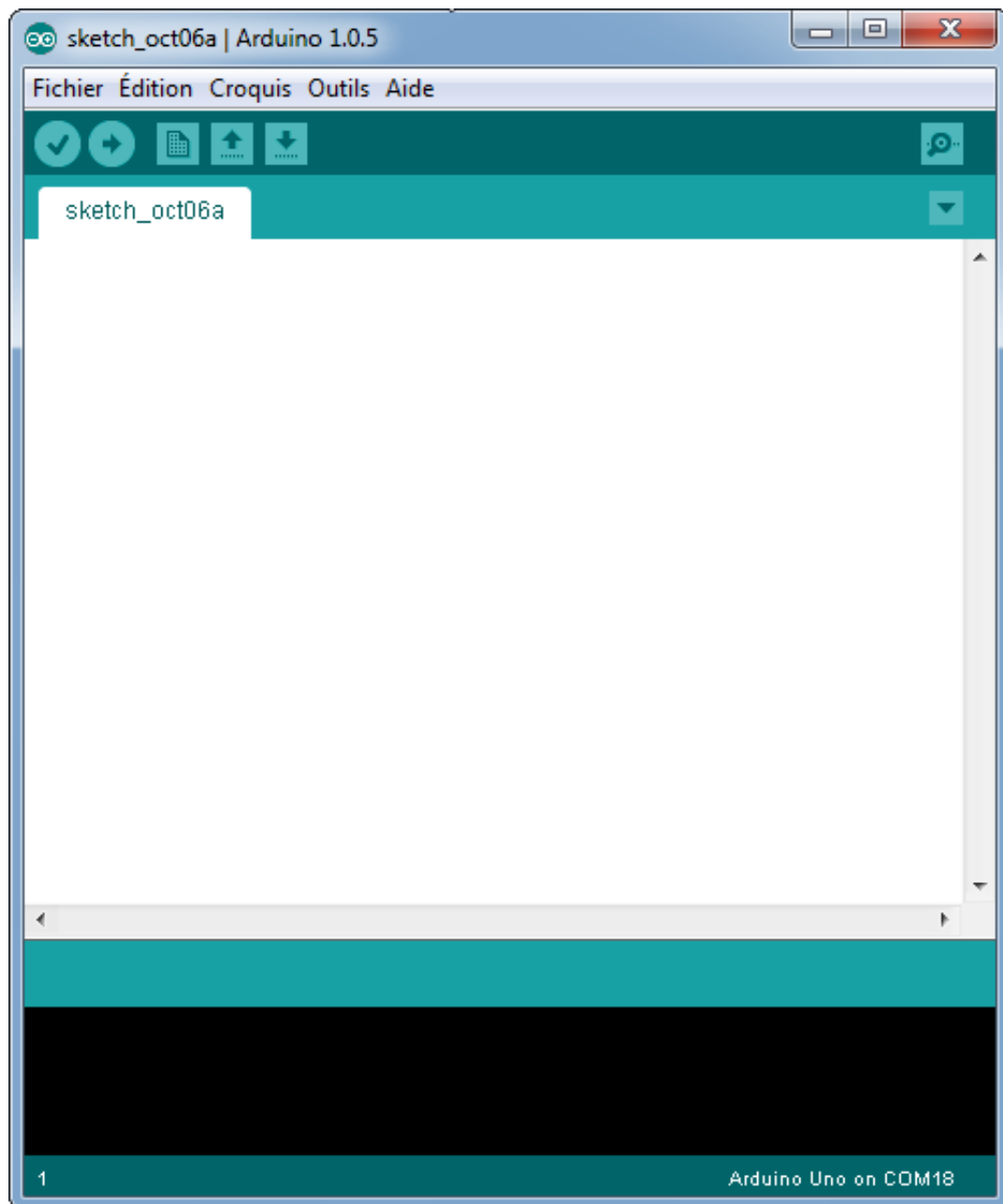
# Interface du logiciel

## Lancement du logiciel

Lançons le logiciel en double-cliquant sur l'icône avec le symbole "infinie" en vert. C'est l'exécutable du logiciel. Après un léger temps de réflexion, une image s'affiche :



Cette fois, après quelques secondes, le logiciel s'ouvre. Une fenêtre se présente à nous :

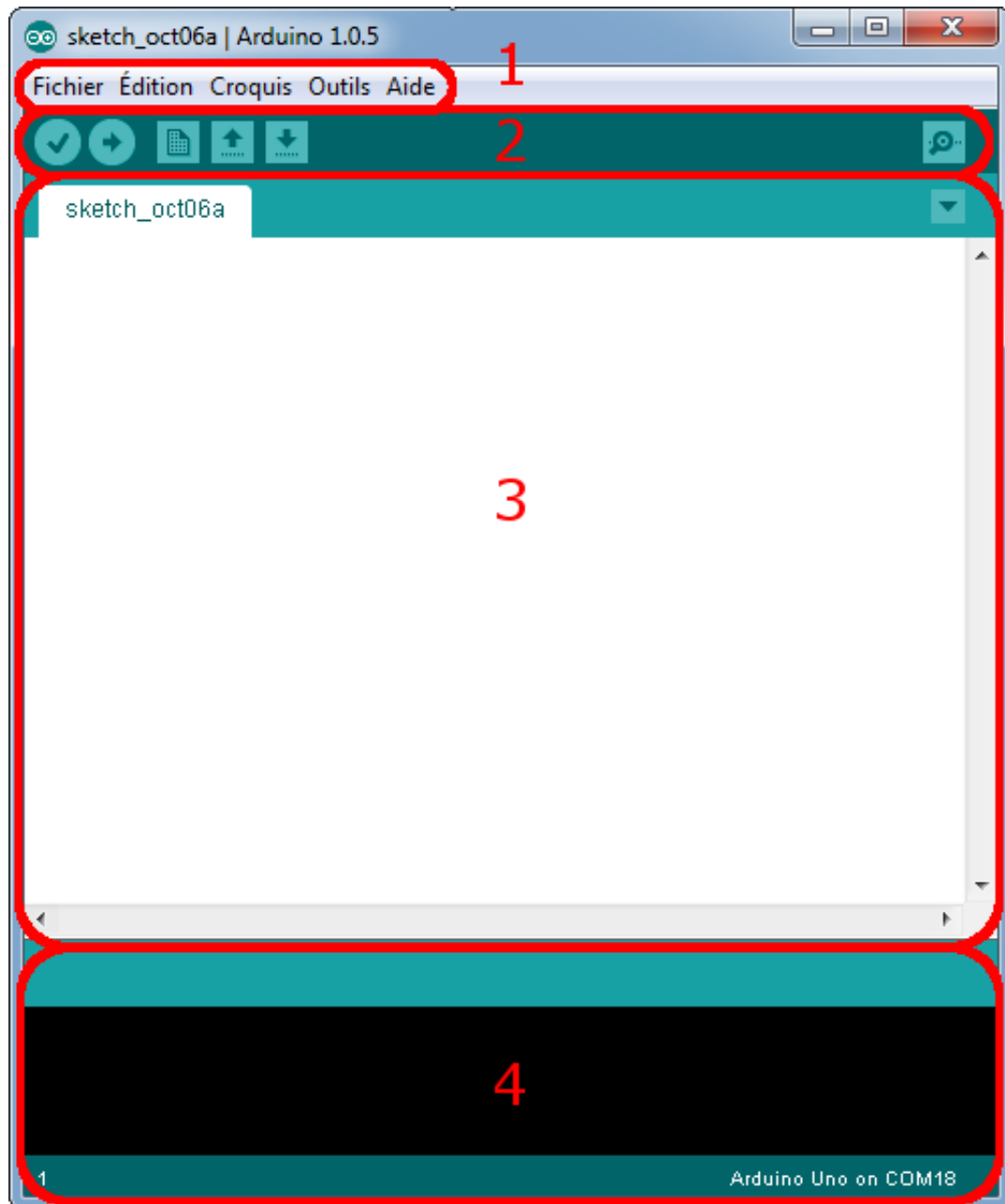


Ce qui saute aux yeux en premier, c'est la clarté de présentation du logiciel. On voit tout de suite son interface intuitive. Voyons comment se compose cette interface.

## Présentation du logiciel

J'ai découpé, grâce à mon ami paint.net, l'image précédente en plusieurs parties :





### Correspondance

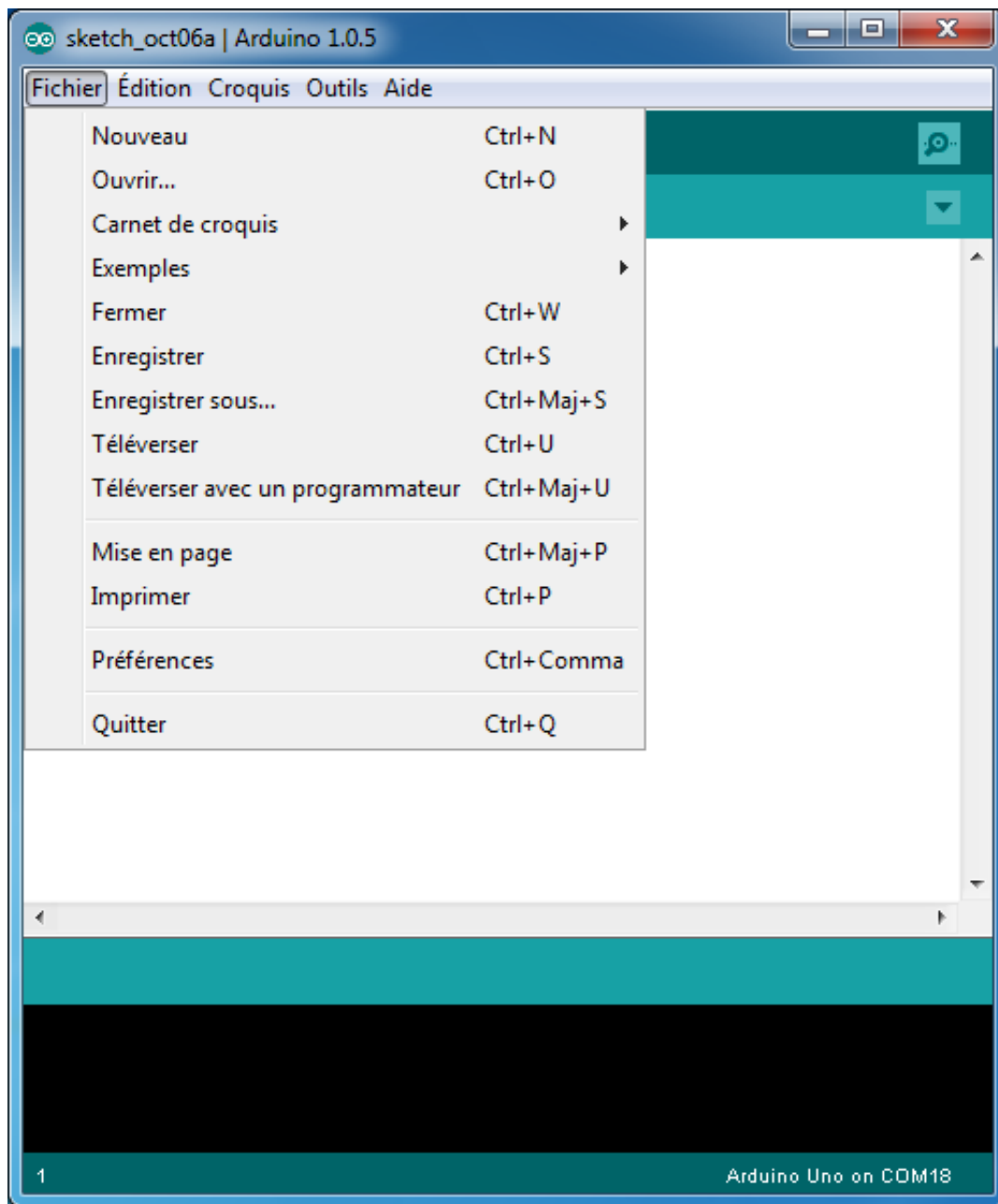
- Le cadre numéro 1 : ce sont les options de configuration du logiciel
- Le cadre numéro 2 : il contient les boutons qui vont nous servir lorsque l'on va programmer nos cartes
- Le cadre numéro 3 : ce bloc va contenir le programme que nous allons créer
- Le cadre numéro 4 : celui-ci est important, car il va nous aider à corriger les fautes dans notre programme. C'est le **débogueur**.

## Approche et utilisation du logiciel

Attaquons-nous plus sérieusement à l'utilisation du logiciel. La barre des menus est entourée en rouge et numérotée par le chiffre 1.

### Le menu *File*

C'est principalement ce menu que l'on va utiliser le plus. Il dispose d'un certain nombre de choses qui vont nous être très utiles. Il a été traduit en français progressivement, nous allons donc voir les quelques options qui sortent de l'ordinaire :



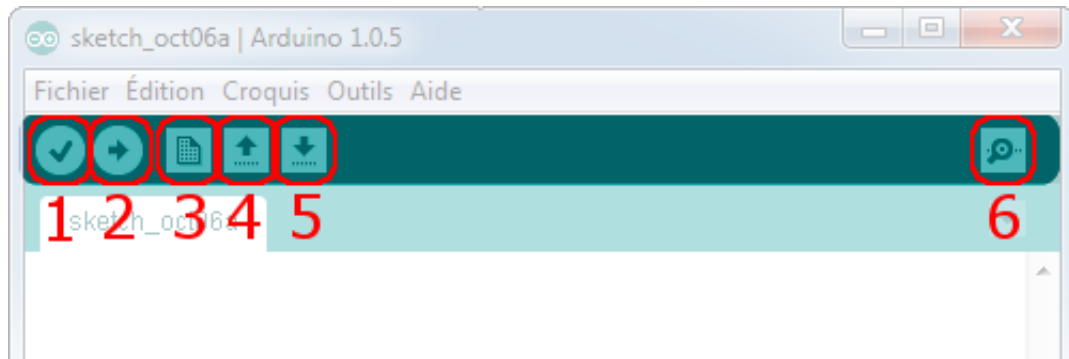
- *Carnet de croquis* : CE menu regroupe les fichiers que vous avez pu faire jusqu'à maintenant (et s'ils sont enregistré dans le dossier par défaut du logiciel)
- *Exemples* (exemples) : ceci est important, toute une liste se déroule pour afficher les noms d'exemples de programmes existants ; avec ça, vous pourrez vous aider/inspirer pour créer vos propres programmes ou tester de nouveaux composants
- *Téléverser* : Permet d'envoyer le programme sur la carte Arduino. Nous y reviendrons 😊.
- *Téléverser avec un programmeur* : Idem que si dessus, mais avec l'utilisation d'un programmeur (vous n'en n'aurez que très rarement besoin).
- *Préférences* : Vous pourrez régler ici quelques paramètres du logiciel

Le reste des menus n'est pas intéressant pour l'instant, on y reviendra plus tard, avant

de commencer à programmer.

## Les boutons

Voyons à présent à quoi servent les boutons, encadrés en rouge et numérotés par le chiffre 2.



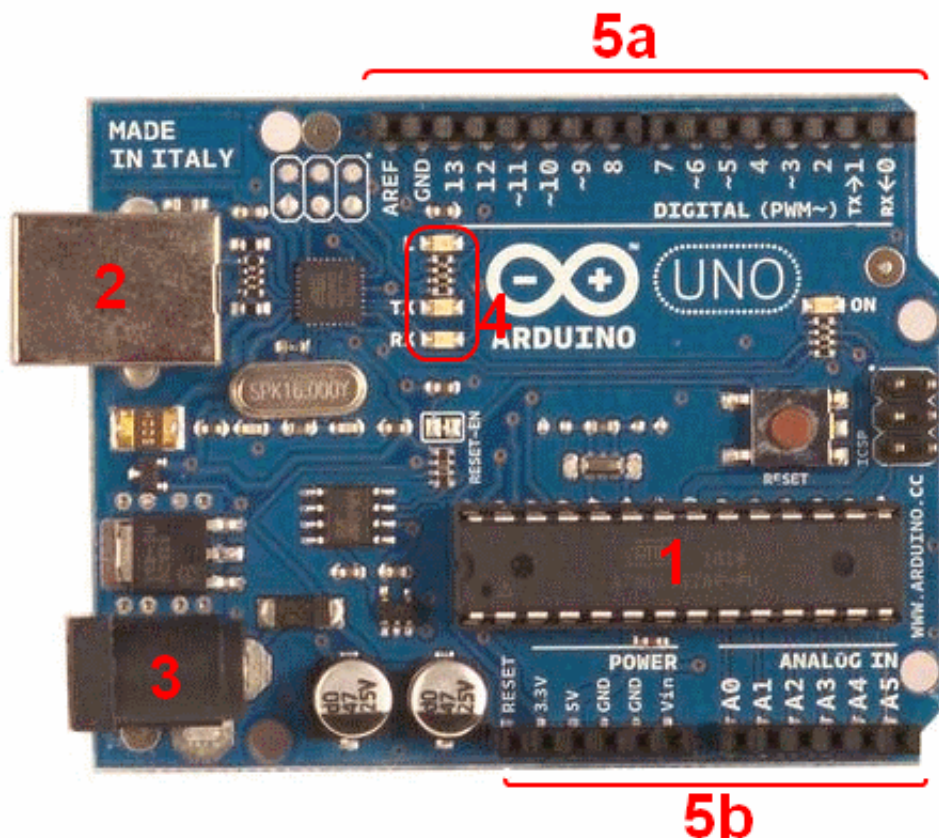
- Bouton 1 : Ce bouton permet de vérifier le programme, il actionne un module qui cherche les erreurs dans votre programme
- Bouton 2 : Charge (téléverse) le programme dans la carte Arduino
- Bouton 3 : Crée un nouveau fichier
- Bouton 4 : Ouvre un fichier
- Bouton 5 : Enregistre le fichier
- Bouton 6 : Ouvre le moniteur série (on verra plus tard ce que c'est 😊)

Enfin, on va pouvoir s'occuper du matériel que vous devriez tous posséder en ce moment même : la carte Arduino !

## [Arduino 104] Le matériel

### Présentation de la carte

Pour commencer notre découverte de la carte Arduino, je vais vous présenter la carte en elle-même. Nous allons voir comment s'en servir et avec quoi. J'ai représenté en rouge sur cette photo les points importants de la carte.



*Présentation de la carte Arduino*

## Constitution de la carte

Voyons quels sont ces points importants et à quoi ils servent.

### *Le micro-contrôleur*

Voilà le cerveau de notre carte (en **1**). C'est lui qui va recevoir le programme que vous aurez créé et qui va le stocker dans sa mémoire puis l'exécuter. Grâce à ce programme, il va savoir faire des choses, qui peuvent être : faire clignoter une LED, afficher des caractères sur un écran, envoyer des données à un ordinateur, ...

### *Alimentation*

Pour fonctionner, la carte a besoin d'une alimentation. Le microcontrôleur fonctionnant sous 5V, la carte peut être alimentée en 5V par le port USB (en **2**) ou bien par une alimentation externe (en **3**) qui est comprise entre 7V et 12V. Cette tension doit être continue et peut par exemple être fournie par une pile 9V. Un régulateur se charge ensuite de réduire la tension à 5V pour le bon fonctionnement de la carte. Pas de danger de tout griller donc! Veuillez seulement à respecter l'intervalle de 7V à 15V (même si le régulateur peut supporter plus, pas la peine de le retrancher dans ses limites)

### *Visualisation*

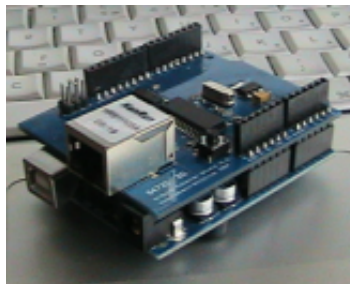
Les trois "points blancs" entourés en rouge (**4**) sont en fait des LED dont la taille est de

l'ordre du millimètre. Ces LED servent à deux choses :

- Celle tout en haut du cadre : elle est connectée à une broche du microcontrôleur et va servir pour tester le matériel. *Nota* : Quand on branche la carte au PC, elle clignote quelques secondes.
- Les deux LED du bas du cadre : servent à visualiser l'activité sur la voie série (une pour l'émission et l'autre pour la réception). Le téléchargement du programme dans le micro-contrôleur se faisant par cette voie, on peut les voir clignoter lors du chargement.

## La connectique

La carte Arduino ne possédant pas de composants qui peuvent être utilisés pour un programme, mis à part la LED connectée à la broche 13 du microcontrôleur, il est nécessaire de les rajouter. Mais pour ce faire, il faut les connecter à la carte. C'est là qu'intervient la connectique de la carte (en **5a** et **5b**). Par exemple, on veut connecter une LED sur une sortie du microcontrôleur. Il suffit juste de la connecter, avec une résistance en série, à la carte, sur les fiches de connections de la carte. Cette connectique est importante et a un brochage qu'il faudra respecter. Nous le verrons quand nous apprendrons à faire notre premier programme. C'est avec cette connectique que la carte est "extensible", car l'on peut y brancher tous types de montages et modules ! Par exemple, la carte Arduino Uno peut être étendue avec des shields, comme le « **Shield Ethernet** » qui permet de connecter cette dernière à internet.



*Une carte Arduino étendue avec un Ethernet Shield*

---

## Installation

Afin d'utiliser la carte, il faut l'installer. Normalement, les drivers sont déjà installés sous GNU/Linux. Sous mac, il suffit de double cliquer sur le fichier *.mkpg* inclus dans le téléchargement de l'application Arduino et l'installation des drivers s'exécute de façon automatique.

### Sous Windows

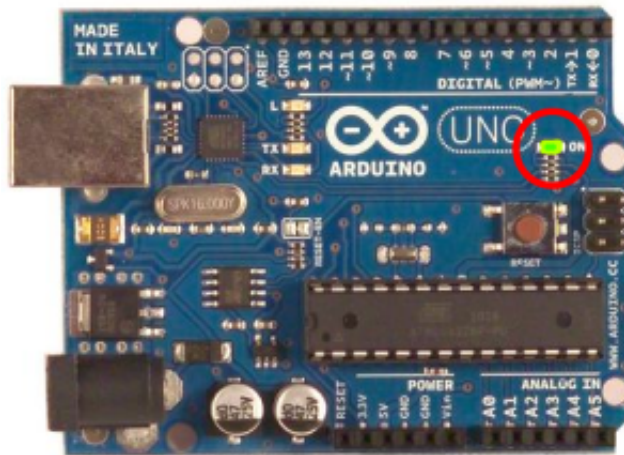
Lorsque vous connectez la carte à votre ordinateur sur le port USB, un petit message en bas de l'écran apparaît. Théoriquement, la carte que vous utilisez doit s'installer toute seule. Cependant, si vous êtes sous Win 7 comme moi, il se peut que ça ne marche pas du premier coup. Dans ce cas, laissez la carte branchée puis ensuite allez dans le panneau de configuration. Une fois là, cliquez sur "système" puis dans le panneau de gauche sélectionnez "gestionnaire de périphériques". Une fois ce menu ouvert, vous

devriez voir un composant avec un panneau "attention" jaune. Faites un clic droit sur le composant et cliquez sur "Mettre à jour les pilotes". Dans le nouveau menu, sélectionnez l'option "Rechercher le pilote moi-même". Enfin, il ne vous reste plus qu'à aller sélectionner le bon dossier contenant le driver. Il se trouve dans le dossier d'Arduino que vous avez du décompresser un peu plus tôt et se nomme "drivers" (attention, ne descendez pas jusqu'au dossier "FTDI"). Par exemple, pour moi le chemin sera:

*[le-chemin-jusqu'au-dossier]arduino-0022\arduino-0022\drivers*

Il semblerait qu'il y est des problèmes en utilisant la version française d'Arduino (les drivers sont absents du dossier). Si c'est le cas, il vous faudra télécharger la version originale (anglaise) pour pouvoir installer les drivers.

Après l'installation et une suite de clignotement sur les micro-LED de la carte, celle-ci devrait être fonctionnelle; une petite LED verte témoigne de la bonne alimentation de la carte :



*Carte connectée et alimentée*

## Tester son matériel

Avant de commencer à programmer la tête baissée, il faut, avant toutes choses, tester le bon fonctionnement de la carte. Car ce serait idiot de programmer la carte et chercher les erreurs dans le programme alors que le problème vient de la carte ! 🤖 Nous allons tester notre matériel en chargeant un programme qui fonctionne dans la carte.

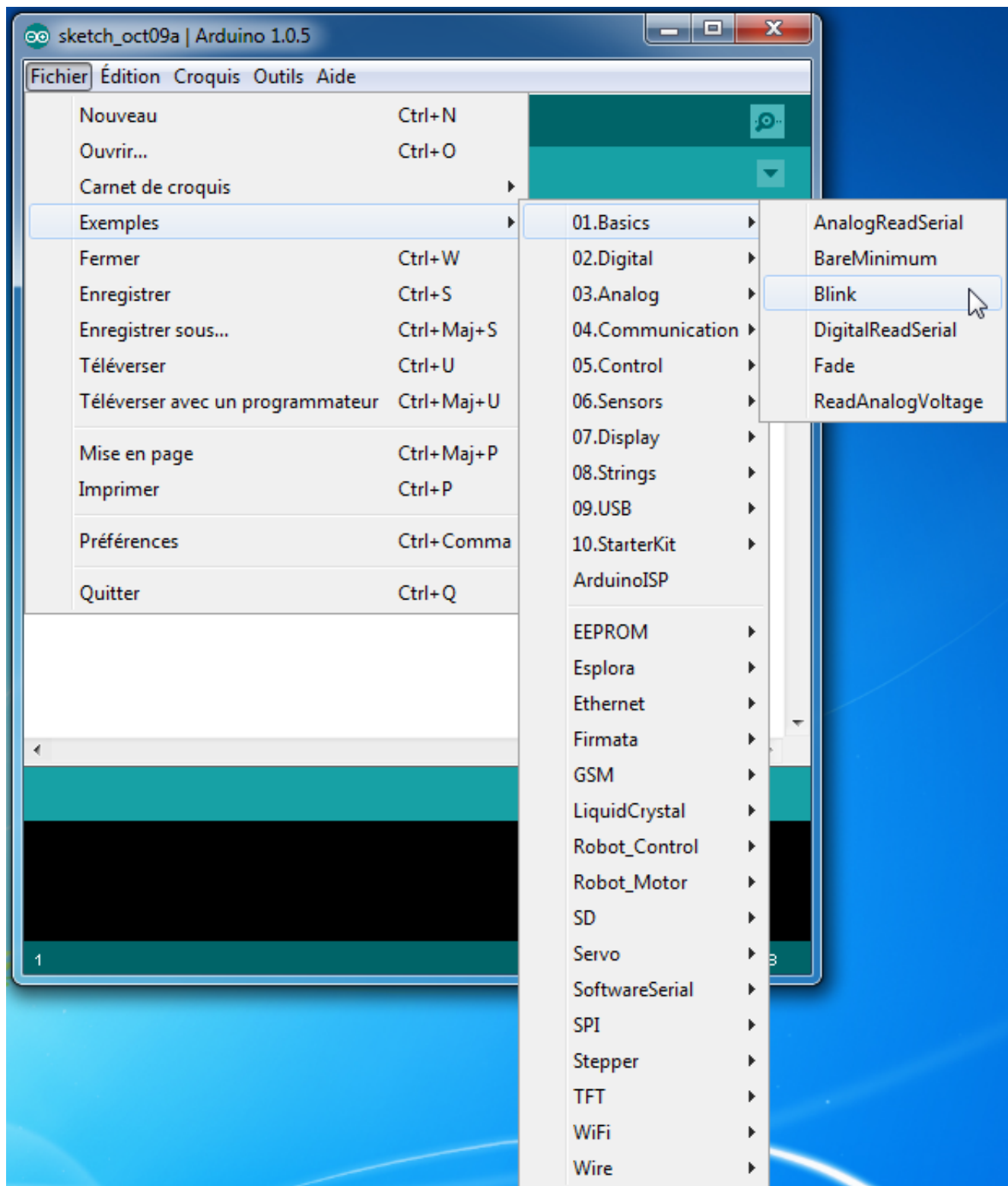
Mais, on n'en a pas encore fait de programmes ? o\_O

Tout juste ! Mais le logiciel Arduino contient des exemples de programmes. Et bien ce sont ces exemples que nous allons utiliser pour tester la carte.

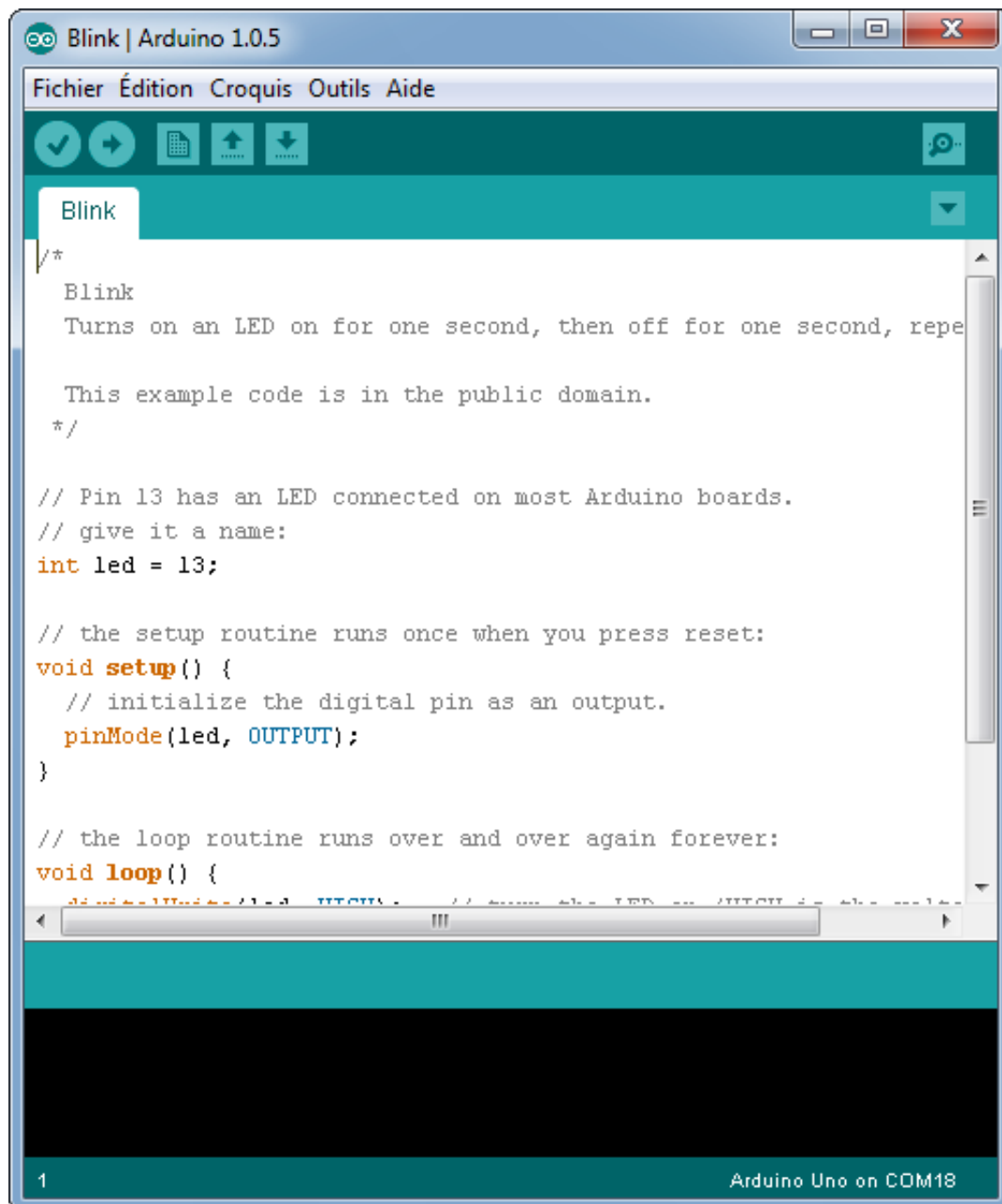
### **1ère étape : ouvrir un programme**

Nous allons choisir un exemple tout simple qui consiste à faire clignoter une LED. Son nom est *Blink* et vous le trouverez dans la catégorie *Basics* :



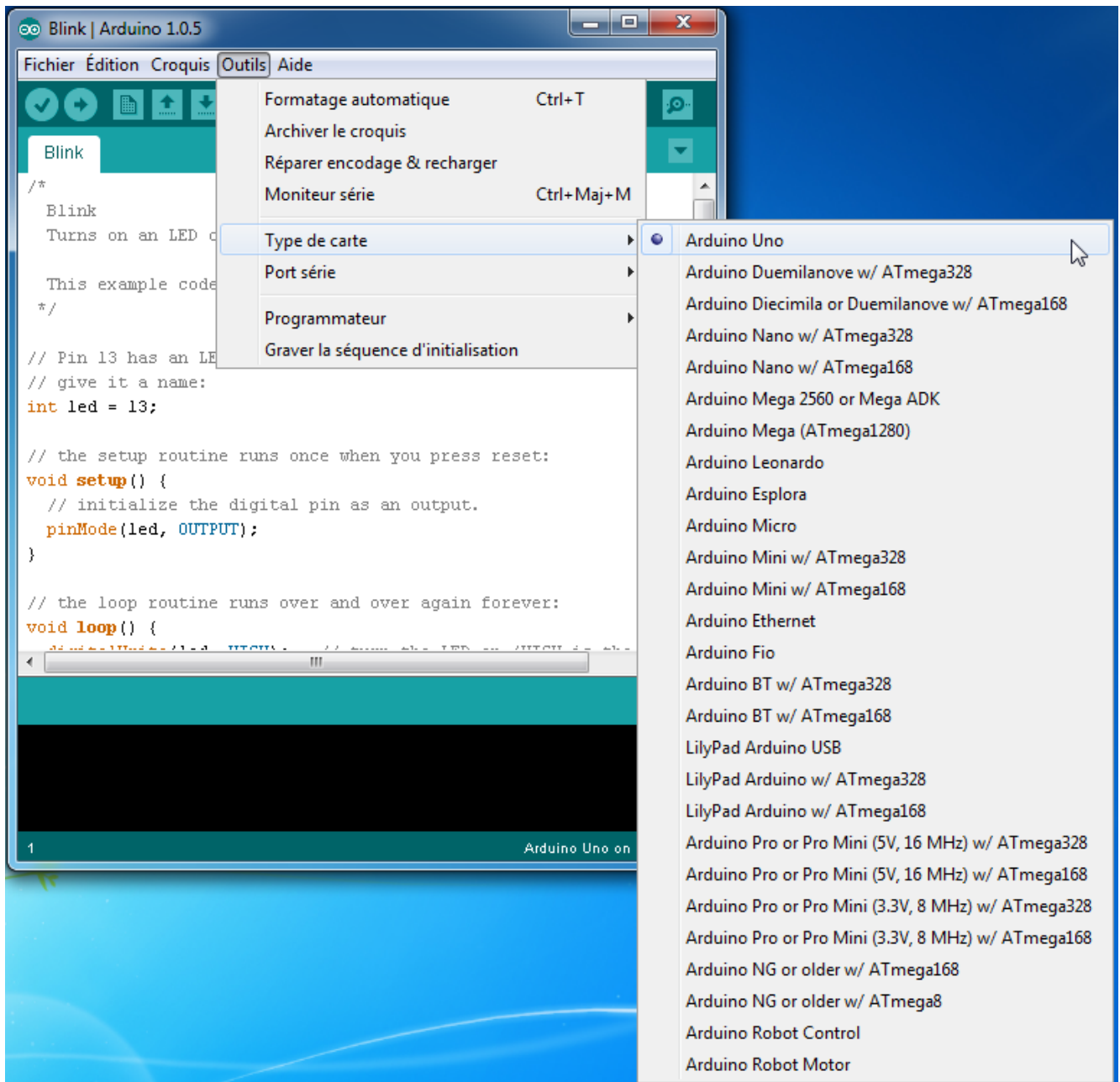


Une fois que vous avez cliqué sur *Blink*, une nouvelle fenêtre va apparaître. Elle va contenir le programme *Blink*. Vous pouvez fermer l'ancienne fenêtre qui va ne nous servir plus à rien.

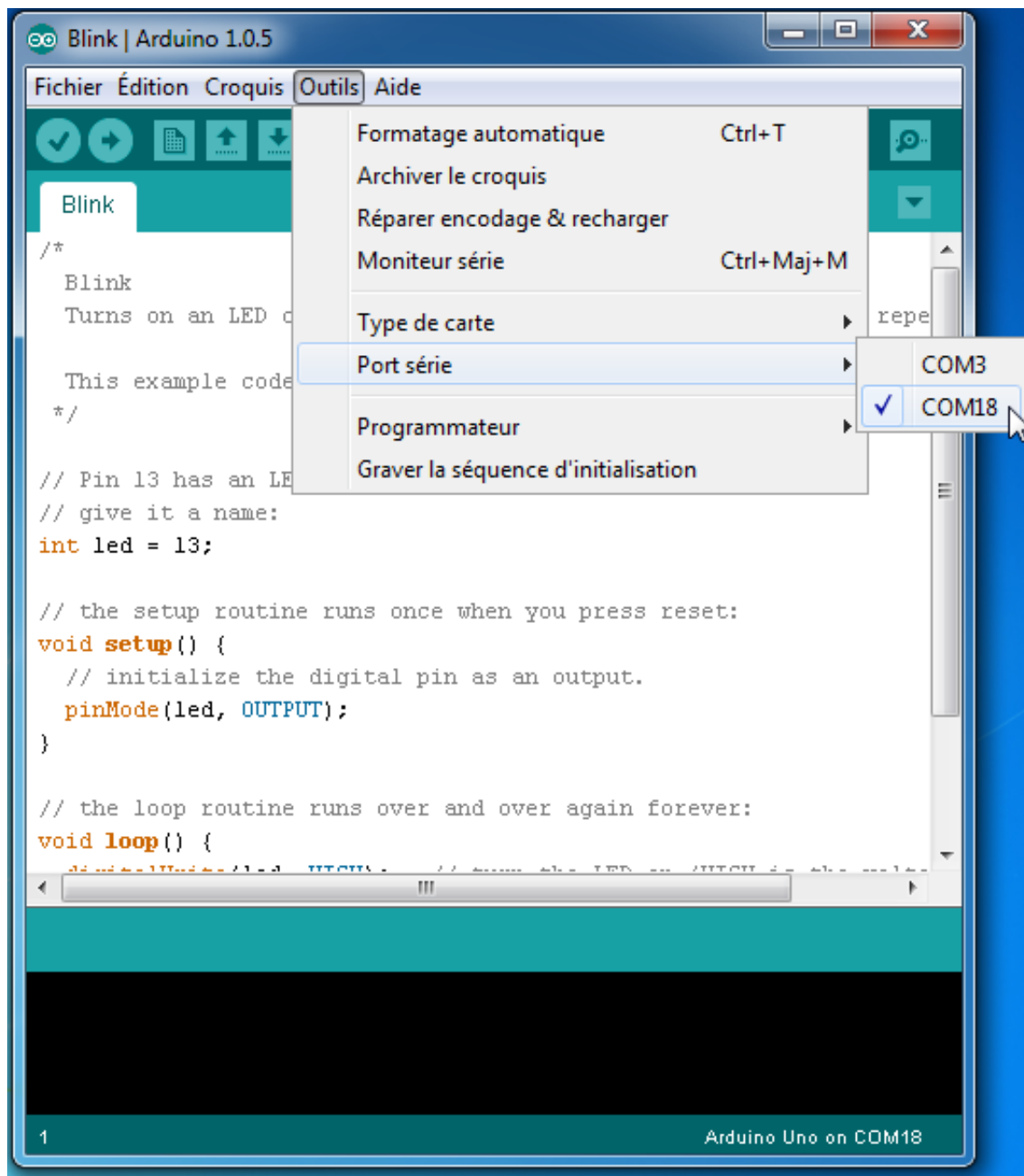


## 2e étape

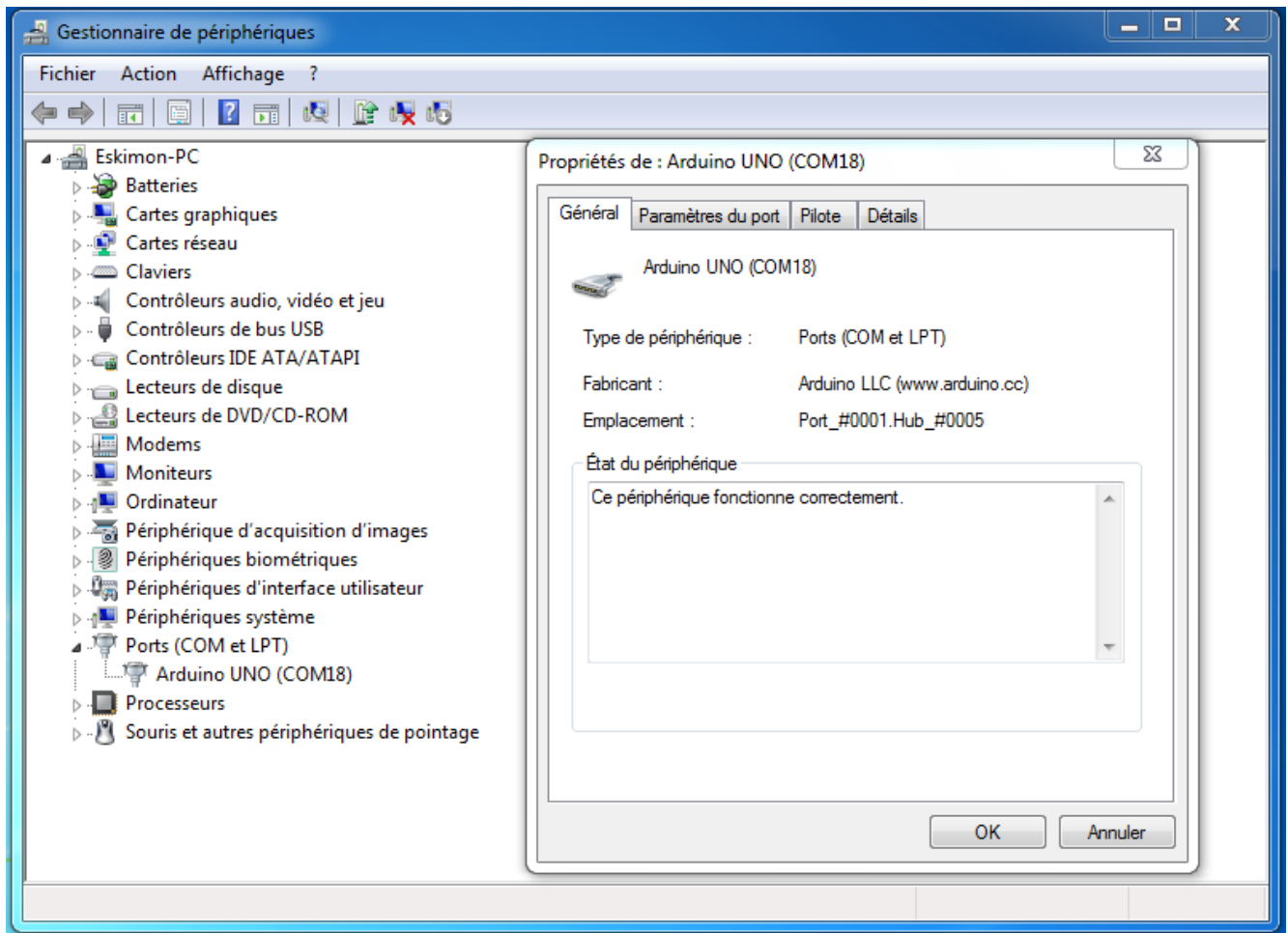
Avant d'envoyer le programme *Blink* vers la carte, il faut dire au logiciel quel est le nom de la carte et sur quel port elle est branchée. **Choisir la carte que l'on va programmer.** Ce n'est pas très compliqué, le nom de votre carte est indiqué sur elle. Pour nous, il s'agit de la carte "Uno". Allez dans le menu "Tools" ("outils" en français) puis dans "Board" ("carte" en français). Vérifiez que c'est bien le nom "Arduin Uno" qui est coché. Si ce n'est pas le cas, cochez-le.



**Choisissez le port de connexion de la carte.** Allez dans le menu *Tools*, puis *Serial port*. Là, vous choisissez le port COMX, X étant le numéro du port qui est affiché. Ne choisissez pas COM1 car il n'est quasiment jamais connecté à la carte. Dans mon cas, il s'agit de COM5 :

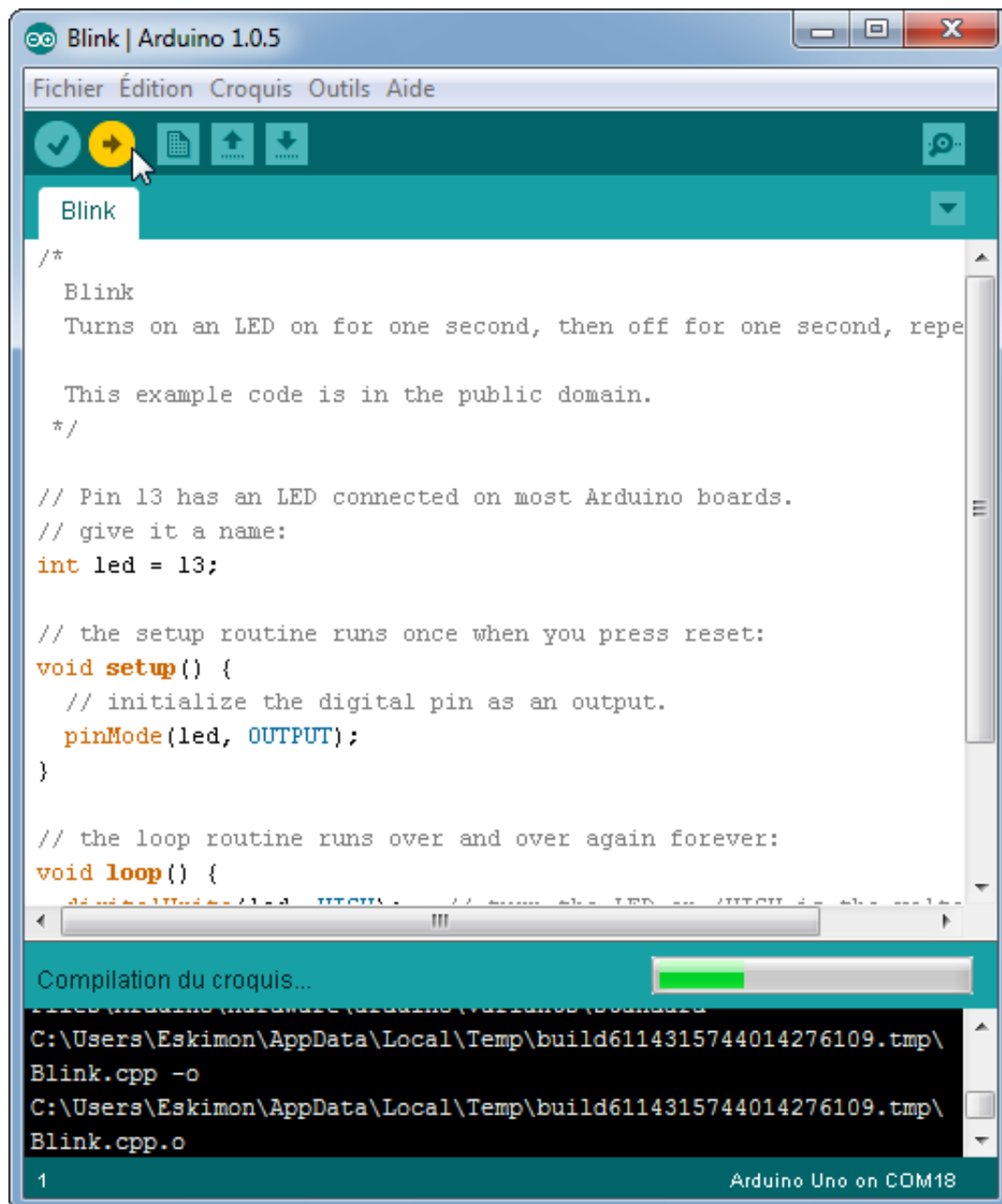


Pour trouver le port de connexion de la carte, vous pouvez aller dans le **gestionnaire de périphérique** qui se trouve dans le **panneau de configuration**. Regardez à la ligne **Ports (COM et LPT)** et là, vous devriez avoir **Arduino Uno (COMX)**. Aller, une image pour le plaisir :



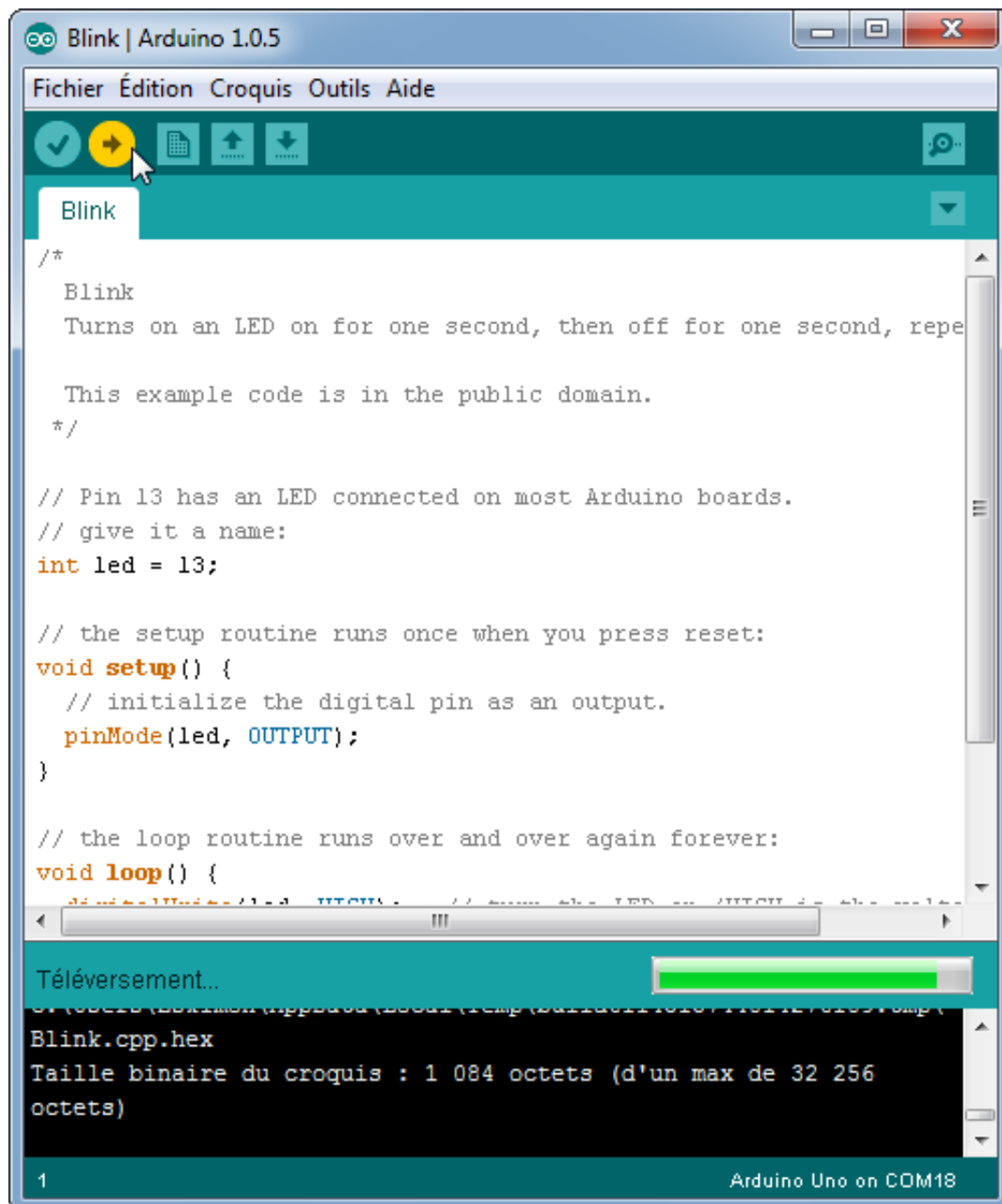
### *Dernière étape*

Très bien. Maintenant, il va falloir envoyer le programme dans la carte. Pour ce faire, il suffit de cliquer sur le bouton *Téléverser*, en jaune-orangé sur la photo :

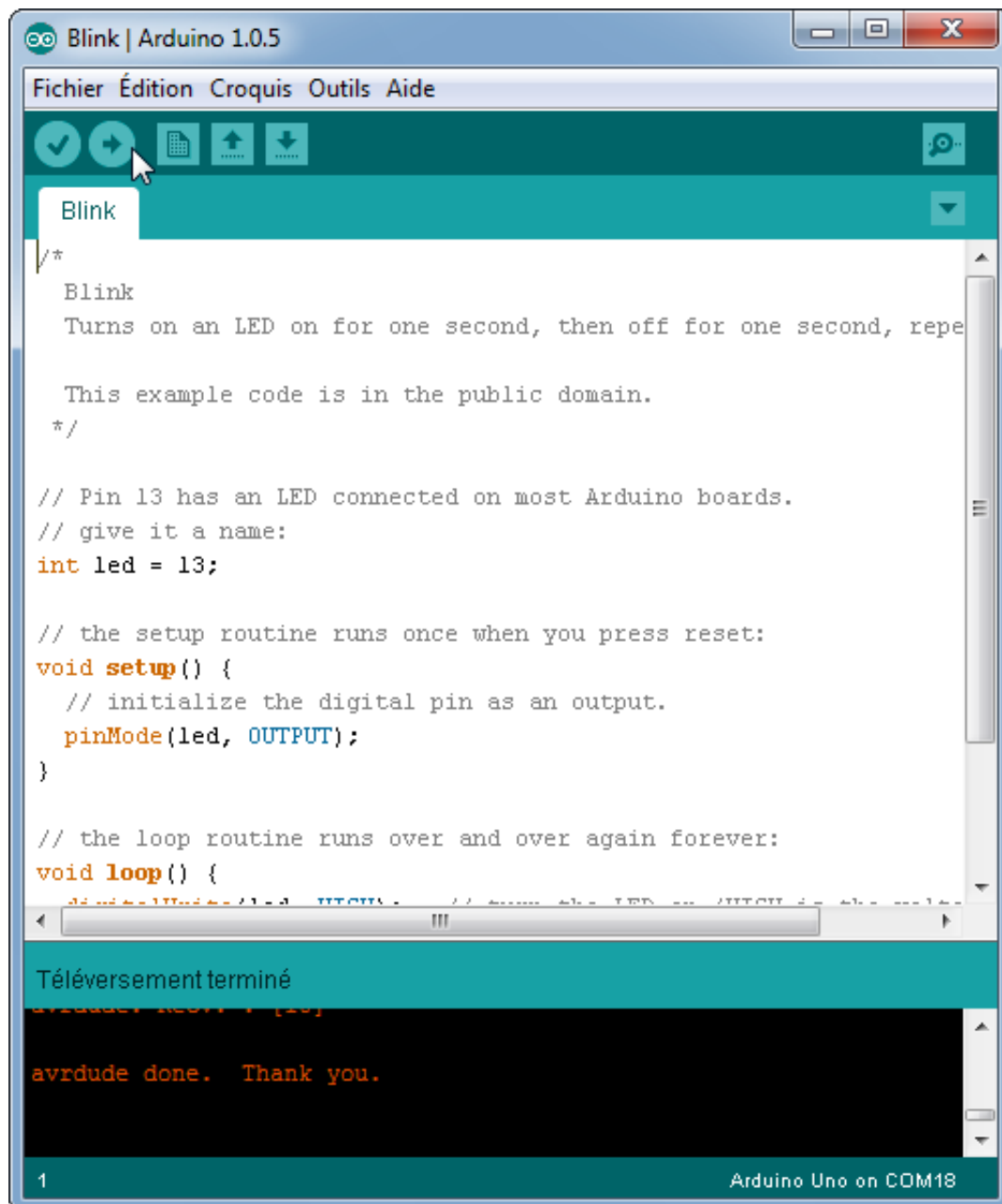


Vous verrez tout d'abord le message "Compilation du croquis en cours..." pour vous informer que le programme est en train d'être compilé en langage machine avant d'être envoyé. Ensuite vous aurez ceci :





En bas dans l'image, vous voyez le texte : "Téléversement...", cela signifie que le logiciel est en train d'envoyer le programme dans la carte. Une fois qu'il a fini, il affiche un autre message :



Le message afficher : “*Téléversement terminé*” signale que le programme à bien été chargé dans la carte. Si votre matériel fonctionne, vous devriez avoir une LED sur la carte qui clignote :

Si vous n’obtenez pas ce message mais plutôt un truc en rouge, pas d’inquiétude, le matériel n’est pas forcément défectueux!

En effet, plusieurs erreurs sont possibles:

- - l’IDE recompile avant d’envoyer le code, vérifier la présence d’erreur
- - La voie série est peut-être mal choisi, vérifier les branchements et le choix de la voie série
- - l’IDE est codé en JAVA, il peut-être capricieux et bugger de temps en temps (surtout avec la voie série...) : réessayez l’envoi!



*LED sur la carte qui clignote*

## Fonctionnement global

Nous avons vu précédemment ce qu'était une carte électronique programmable. Nous avons également vu de quels éléments se basait une carte électronique pour fonctionner (schéma électronique, schéma de câblage). Je viens de vous présenter la carte, de quoi elle est principalement constituée. Enfin, je vous ai montré comment l'utiliser de manière à faire clignoter une petite lumière. Dorénavant, nous allons voir comment elle fonctionne de façon globale et répondre à quelques questions qui pourraient vous trotter dans la tête : "Comment la carte sait qu'il y a une LED de connectée ?", "Et comment sait-elle que c'est sur telle broche ?", "Et le programme, où est-ce qu'il se trouve et sous quelle forme ?", "Comment la carte fait pour comprendre ce qu'elle doit faire ?", ... De nombreuses questions, effectivement ! 😊

## Partons du programme

### *Le contenu*

Le contenu du programme, donc le programme en lui-même, est ce qui va définir chaque action que va exécuter la carte Arduino. Mais ce n'est pas tout ! Dans le programme il y a plusieurs zones, que nous verrons plus en détail tout au long de la lecture de ce cours, qui ont chacune un rôle particulier.

- La première zone sert principalement (je ne vais pas m'étendre) à dire à la carte de **garder en mémoire quelques informations** qui peuvent être : l'emplacement d'un élément connecté à la carte, par exemple une LED en broche 13, ou bien une valeur quelconque qui sera utile dans le programme :

```
// constants won't change. They're used here to
// set pin numbers:
const int ledPin = 13;      // the number of the LED pin
```

- La zone secondaire est l'endroit où l'on va **initialiser certains paramètres** du

programme. Par exemple, on pourra dire à la carte qu'elle devra communiquer avec l'ordinateur ou simplement lui dire ce qu'elle devra faire de la LED qui est connectée sur sa broche 13. On peut encore faire d'autres choses, mais nous le verrons plus tard.


```
void setup() {  
  // initialize the LED pin as an output:  
  pinMode(ledPin, OUTPUT);  
}
```

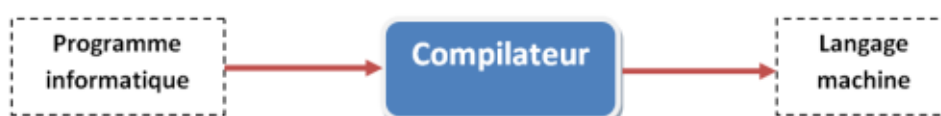
- La dernière zone est la **zone principale où se déroulera le programme**. Tout ce qui va être écrit dans cette zone sera exécuté par la carte, se sont les actions que la carte fera. Par exemple, c'est ici qu'on pourra lui dire de faire clignoter la LED sur sa broche 13. On pourra également lui demander de faire une opération telle que  $2+2$  ou bien d'autres choses encore !

```
void loop() {  
  digitalWrite(13, HIGH); // set the LED on  
  delay(1000);            // wait for a second  
  digitalWrite(13, LOW);  // set the LED off  
  delay(1000);            // wait for a second  
}
```

En conclusion, tout (vraiment tout !) ce que va faire la carte est inscrit dans le programme. Sans programme, la carte ne sert à rien ! C'est grâce au programme que la carte Arduino va savoir qu'une LED est connectée sur sa broche 13 et ce qu'elle va devoir faire avec, allumer et éteindre la LED alternativement pour la faire clignoter.

## Et l'envoi

Le programme est envoyé dans la carte lorsque vous cliquez sur le bouton . Le logiciel Arduino va alors vérifier si le programme ne contient pas d'erreur et ensuite le compiler (le traduire) pour l'envoyer dans la carte :



```
// constants won't change. They're used here to  
// set pin numbers:  
const int buttonPin = 2; // the number of the pushbutton pin  
const int ledPin = 13;   // the number of the LED pin  
  
// variables will change:  
int buttonState = 0;      // variable for reading the pushbutton status  
  
void setup() {  
  // initialize the LED pin as an output:  
  pinMode(ledPin, OUTPUT);  
  // initialize the pushbutton pin as an input:  
  pinMode(buttonPin, INPUT);  
}  
  
void loop(){  
  // read the state of the pushbutton value:  
  buttonState = digitalRead(buttonPin);  
  
  // check if the pushbutton is pressed.  
  // if it is, the buttonState is HIGH:  
  if (buttonState == HIGH) {  
    // turn LED on:  
    digitalWrite(ledPin, HIGH);  
  }  
  else {  
    // turn LED off:  
    digitalWrite(ledPin, LOW);  
  }  
}
```



Traduction



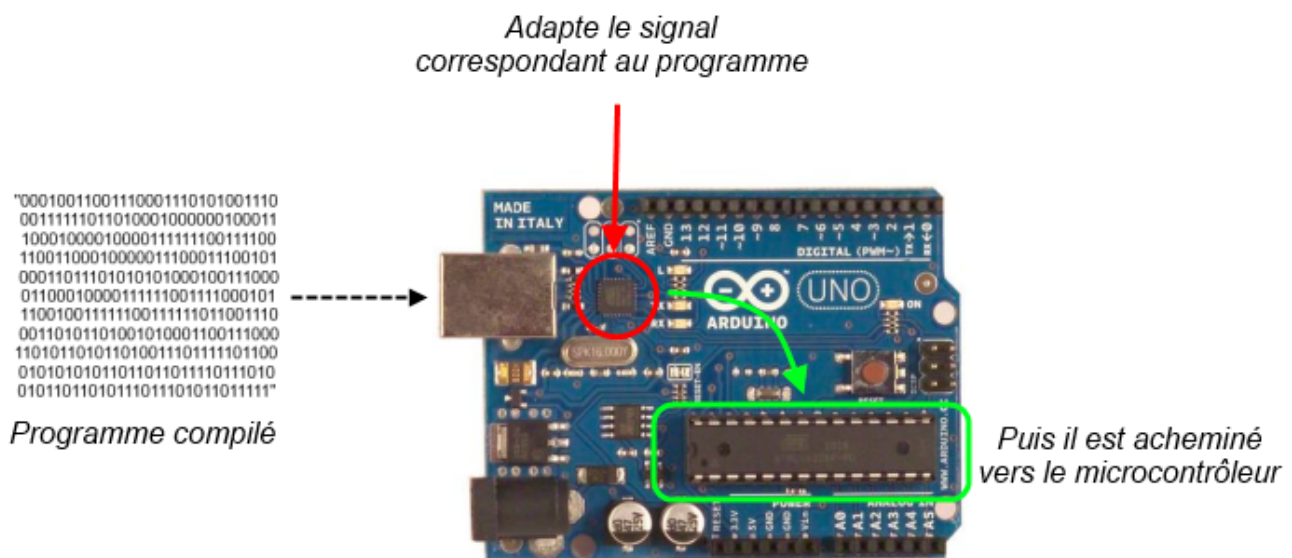
```
"00010011001110001110101001110  
00111111011010001000000100011  
10001000010000111111100111100  
11001100010000011100011100101  
00011011101010101000100111000  
01100010000111111001111000101  
110010011111100111111011001110  
00110101101001010001100111000  
110101101011010011101111101100  
0101010101101101101111011010  
0101101101011101110101101111"
```

Au départ, le programme est sous forme de texte, puis il est transformé en un langage composé uniquement de 0 et de 1 (ce qui est absolument illisible en soi ! 😊).

L'envoi du programme est géré par votre ordinateur : le programme passe, sous forme de 0 et de 1, dans le câble USB qui relie votre ordinateur à votre carte et arrive dans la carte. Le reste se passe dans la carte elle-même...

## Réception du programme

Le programme rentre donc dans la carte en passant en premier par le connecteur USB de celle-ci. Il va alors subir une petite transformation qui permet d'adapter le signal électrique correspondant au programme (oui car le programme transite dans le câble USB sous forme de signal électrique) vers un signal plus approprié pour le microcontrôleur. On passe ainsi d'un signal codé pour la norme USB à un signal codé pour une simple voie série (que l'on étudiera plus tard d'ailleurs). Puis ce "nouveau" signal est alors intercepté par le microcontrôleur.



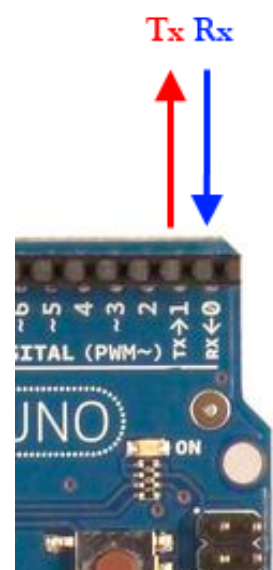
Tout le reste se passe alors...

## A l'intérieur du microcontrôleur

### *L'emplacement du programme*

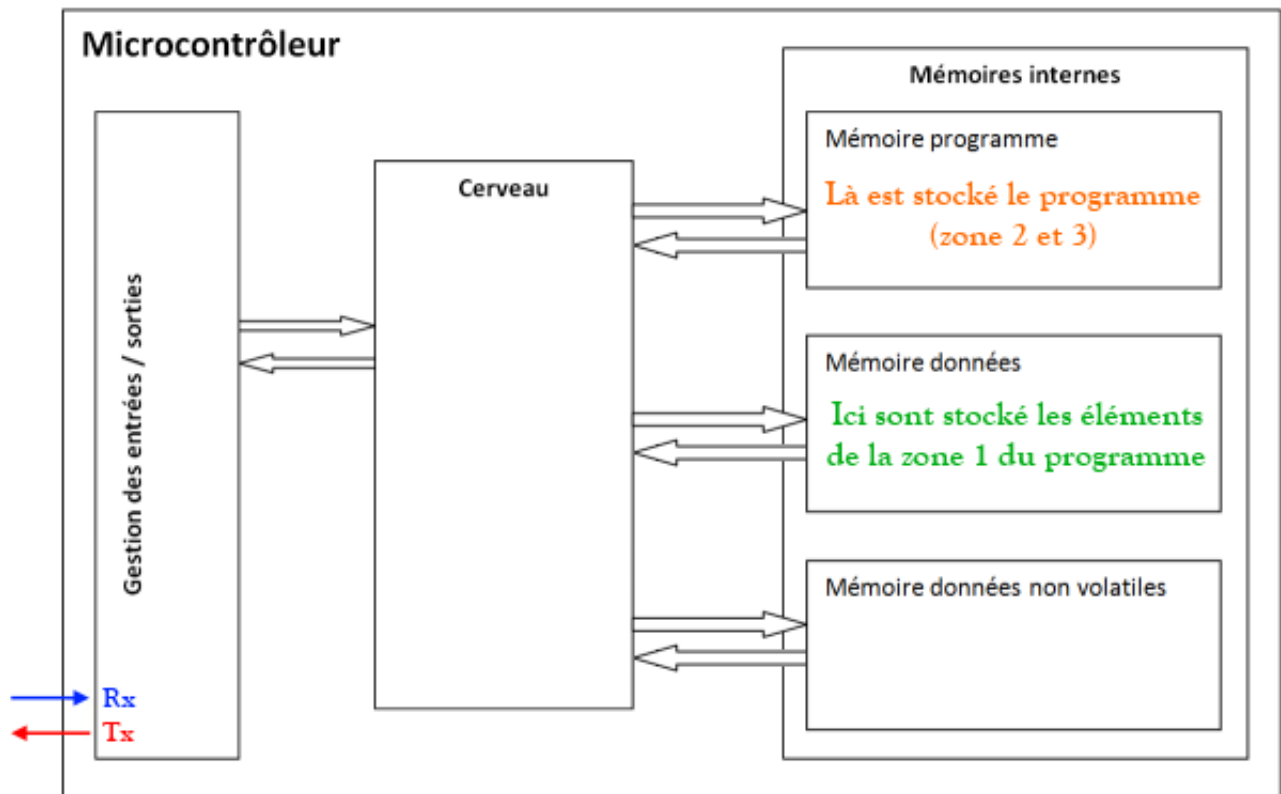
Le microcontrôleur reçoit le programme sous forme de signal électrique sur ses broches Tx et Rx, d'ailleurs disponible sur les broches de la carte (cf. image). Une fois qu'il est reçu, il est intégralement stocké dans une mémoire de type Flash que l'on appellera "la mémoire de programme". Ensuite, lorsque la carte démarre "normalement" (qu'aucun programme n'est en train d'être chargé), le cerveau va alors gérer les données et les répartir dans les différentes mémoires :

- La **mémoire programme** est celle qui va servir à savoir où l'on en est dans le programme, à quelle instruction on est rendu. C'est à dire, en quelque sorte, pointer sur des morceaux des zones 2 et 3 que l'on a vu dans le précédent exemple de programme.
- La **mémoire de données**, aussi appelé "RAM" (comme dans votre ordinateur) va



stocker les variables telles que le numéro de la broche sur laquelle est connectée une LED, ou bien une simple valeur comme un chiffre, un nombre, des caractères, etc.

Voici un petit synoptique qui vous montre un peu l'intérieur du microcontrôleur (c'est très simplifié !):



### Démarrage du microcontrôleur

Lorsque le microcontrôleur démarre, il va commencer par lancer un bout de code particulier : le bootloader. C'est ce dernier qui va surveiller si un nouveau programme arrive sur la voie USB et s'il faut donc changer l'ancien en mémoire par le nouveau. Si rien n'arrive, il donne la main à votre programme, celui que vous avez créé. Ce dernier va alors défiler, instruction par instruction. Chaque fois qu'une nouvelle variable sera nécessaire, elle sera mise en RAM pour qu'on ait une mémoire de cette dernière (et supprimer lorsqu'elle n'est plus nécessaire). Sinon, les instructions vont se suivre une par une, dans l'ordre que vous les avez écrites.

## [Arduino 105] Le langage Arduino (1/2)

A présent que vous avez une vision globale sur le fonctionnement de la carte Arduino, nous allons pouvoir apprendre à programmer avant de nous lancer dans la réalisation de programmes très simple pour débiter ! Pour pouvoir programmer notre carte, il nous faut trois choses :

- Un ordinateur
- Une carte Arduino
- Et connaître le langage Arduino



C'est ce dernier point qu'il nous faut acquérir. Le but même de ce chapitre est de vous apprendre à programmer avec le langage Arduino. Cependant, ce n'est qu'un support de cours que vous pourrez parcourir lorsque vous devrez programmer tout seul votre carte. En effet, c'est en manipulant que l'on apprend, ce qui implique que votre apprentissage en programmation sera plus conséquent dans les prochains chapitres que dans ce cours même.

Le langage Arduino est très proche du C et du C++. Pour ceux dont la connaissance de ces langages est fondée, ne vous sentez pas obligé de lire les deux chapitre sur le langage Arduino. Bien qu'il y ait des points quelques peu important.

---

## La syntaxe du langage

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

### Le code minimal

Avec Arduino, nous devons utiliser un *code minimal* lorsque l'on crée un programme. Ce code permet de diviser le programme que nous allons créer en deux grosses parties.

```
1 //fonction d'initialisation de la carte
2 void setup()
3 {
4     //contenu de l'initialisation
5 }
6
7 //fonction principale, elle se répète (s'exécute) à l'infini
8 void loop()
9 {
10     //contenu de votre programme
11 }
```

Vous avez donc devant vous le code minimal qu'il faut insérer dans votre programme. Mais que peut-il bien signifier pour quelqu'un qui n'a jamais programmé ?

### La fonction

Dans ce code se trouvent deux fonctions. Les fonctions sont en fait des portions de code.

```
1 //fonction d'initialisation de la carte
2 void setup()
3 {
4     //contenu de l'initialisation
5     //on écrit le code à l'intérieur
6 }
```

Cette fonction **setup()** est appelée une seule fois lorsque le programme commence. C'est pourquoi c'est dans cette fonction que l'on va écrire le code qui n'a besoin d'être exécuté une seule fois. On appelle cette fonction : "**fonction d'initialisation**". On y retrouvera la mise en place des différentes sorties et quelques autres réglages. C'est un peu le check-up de démarrage. Imaginez un pilote d'avion dans sa cabine qui fait l'inventaire 😊 : - *patte 2 en sortie, état haut ?* – OK – *timer 3 à 15 millisecondes ?* – OK ... Une fois que l'on a initialisé le programme il faut ensuite créer son "cœur", autrement dit le programme en lui même.

```
1 //fonction principale, elle se répète (s'exécute) à l'infini
2 void loop()
3 {
4     //contenu de votre programme
5 }
```

C'est donc dans cette fonction **loop()** où l'on va écrire le contenu du programme. Il faut savoir que cette fonction est appelée en permanence, c'est-à-dire qu'elle est exécutée une fois, puis lorsque son exécution est terminée, on la ré-exécute et encore et encore. On parle de **boucle infinie**.

A titre informatif, on n'est pas obligé d'écrire quelque chose dans ces deux fonctions. En revanche, il est **obligatoire** de les écrire, même si elles ne contiennent aucun code !

## Les instructions

Dans ces fonctions, on écrit quoi ?

C'est justement l'objet de ce paragraphe. Dans votre liste pour le dîner de ce soir, vous écrivez les tâches importantes qui vous attendent. Ce sont des **instructions**. Les instructions sont des lignes de code qui disent au programme : "fait ceci, fait cela, ..." C'est tout bête mais très puissant car c'est ce qui va orchestrer notre programme.

## Les points virgules

Les points virgules terminent les instructions. Si par exemple je dis dans mon programme : "appelle la fonction *couperDuSaucisson*" je dois mettre un point virgule après l'appel de cette fonction.

Les points virgules ( ; ) sont synonymes d'erreurs car il arrive très souvent de les oublier à la fin des instructions. Par conséquent le code ne marche pas et la recherche de l'erreur peut nous prendre un temps conséquent ! Donc faites bien attention.

## Les accolades

Les accolades sont les "conteneurs" du code du programme. Elles sont propres aux fonctions, aux conditions et aux boucles. Les instructions du programme sont écrites à l'intérieur de ces accolades. Parfois elles ne sont pas obligatoires dans les *conditions* (nous allons voir plus bas ce que c'est), mais je recommande de les **mettre tout le temps** ! Cela rendra plus lisible votre programme.

## Les commentaires

Pour finir, on va voir ce qu'est un commentaire. J'en ai déjà mis dans les exemples de codes. Ce sont des lignes de codes qui seront ignorées par le programme. Elles ne servent en rien lors de l'exécution du programme.

Mais alors c'est inutile ? o\_O

Non car cela va nous permettre à nous et aux programmeurs qui liront votre code (s'il y en a) de savoir ce que signifie la ligne de code que vous avez écrite. C'est très important de mettre des commentaires et cela permet aussi de reprendre un programme laissé dans l'oubli plus facilement ! Si par exemple vous connaissez mal une instruction que vous avez écrite dans votre programme, vous mettez une ligne de commentaire pour vous rappeler la prochaine fois que vous lirez votre programme ce que la ligne signifie. Ligne unique de commentaire :

```
1 //cette ligne est un commentaire sur UNE SEULE ligne
```

Ligne ou paragraphe sur plusieurs lignes :

```
1 /*cette ligne est un commentaire, sur PLUSIEURS lignes
2 qui sera ignoré par le programme, mais pas par celui qui lit le code */
```

## Les accents

Il est formellement interdit de mettre des accents en programmation. Sauf dans les commentaires.

---

## Les variables

Nous l'avons vu, dans un microcontrôleur, il y a plusieurs types de mémoire. Nous nous occuperons seulement de la mémoire "vive" (RAM) et de la mémoire "morte" (EEPROM). Je vais vous poser un problème. Imaginons que vous avez connecté un bouton poussoir sur une broche de votre carte Arduino. Comment allez-vous stocker l'état du bouton (appuyé ou éteint) ?

### Une variable, qu'est ce que c'est ?

Une **variable est un nombre**. Ce nombre est stocké dans un espace de la mémoire vive (RAM) du microcontrôleur. La manière qui permet de les stocker est semblable à celle utilisée pour ranger des chaussures : dans un casier numéroté.

Chaussures rangées dans des  
cases numérotées

```
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
```

41 42 43 44 45 46 47 48 49 50  
51 52 53 54 55 56 57 58 59 60

Une variable est un nombre, c'est tout ? o\_O

Ce nombre a la particularité de changer de valeur. Etrange n'est-ce pas ? Et bien pas tant que ça, car une variable est en fait le **conteneur** du nombre en question. Et ce conteneur va être stocké dans une case de la mémoire. Si on matérialise cette explication par un schéma, cela donnerait :

**nombre => variable => mémoire**

- le symbole “=>” signifiant : “est contenu dans...”

### *Le nom d'une variable*

Le nom de variable accepte quasiment tous les caractères sauf :

- . (le point)
- , (la virgule)
- é,à,ç,è (les accents)

Bon je vais pas tous les donner, il n'accepte que l'alphabet alphanumérique ([a-z], [A-Z], [0-9]) et \_ (underscore)

### *Définir une variable*

Si on donne un nombre à notre programme, il ne sait pas si c'est une variable ou pas. Il faut le lui indiquer. Pour cela, on donne un **type** aux variables. Oui, car il existe plusieurs types de variables ! Par exemple la variable “x” vaut 4 :

```
1 x = 4;
```

Et bien ce code ne fonctionnerait pas car il ne suffit pas ! En effet, il existe une multitude de nombres : les nombres entiers, les nombres décimaux, ... C'est pour cela qu'il faut assigner une variable à un type. Voilà les types de variables les plus répandus :

Type	Quel nombre il stocke ?	Valeurs maximales du nombre stocké	Nombre sur X bits	Nombre d'octets
<b>int</b>	entier	-32 768 à +32 767	16 bits	2 octets
<b>long</b>	entier	-2 147 483 648 à +2 147 483 647	32 bits	4 octets
<b>char</b>	entier	-128 à +127	8 bits	1 octets
<b>float</b>	décimale	$-3.4 \times 10^{38}$ à $+3.4 \times 10^{38}$	32 bits	4 octets
<b>double</b>	décimale	$-3.4 \times 10^{38}$ à $+3.4 \times 10^{38}$	32 bits	4 octets

Par exemple, si notre variable “x” ne prend que des valeurs entières, 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**.

```
1 char x = 0;
```

Si en revanche  $x = 260$ , alors on utilisera le type supérieur (qui accepte une plus grande quantité de nombre) à **char**, autrement dit **int** ou **long**.

Mais t'es pas malin, pour éviter les dépassements de valeur ont met tout dans des double ou long !

Oui, mais NON. Un microcontrôleur, ce n'est pas un ordinateur 2GHz multicore, 4Go de RAM ! Ici on parle d'un système qui fonctionne avec un CPU à 16MHz (soit 0,016 GHz) et 2 Ko de SRAM pour la mémoire vive. Donc deux raisons font qu'il faut choisir ses variables de manière judicieuse :

- La RAM n'est pas extensible, quand il y en a plus, y en a plus !
- Le processeur est de type 8 bits (sur Arduino UNO), donc il est optimisé pour faire des traitements sur des variables de taille 8 bits, un traitement sur une variable 32 bits prendra donc (beaucoup) plus de temps !

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. Voici le tableau des types non signés, on repère ces types par le mot **unsigned** (de l'anglais : non-signé) qui les précède :

Type	Quel nombre il stocke ?	Valeurs maximales du nombre stocké	Nombre sur X bits	Nombre d'octets
<b>unsigned char</b>	entier non négatif	0 à 255	8 bits	1 octets
<b>unsigned int</b>	entier non négatif	0 à 65 535	16 bits	2 octets
<b>unsigned long</b>	entier non négatif	0 à 4 294 967 295	32 bits	4 octets

Une des particularités du langage Arduino est qu'il accepte un nombre plus important de types de variables. Je vous les liste dans ce tableau :

Type	Quel nombre il stocke ?	Valeurs maximales du nombre stocké	Nombre sur X bits	Nombre d'octets
<b>byte</b>	entier non négatif	0 à 255	8 bits	1 octets
<b>word</b>	entier non négatif	0 à 65535	16 bits	2 octets
<b>boolean</b>	entier non négatif	0 à 1	1 bits	1 octets

Pour votre information, vous pouvez retrouver ces tableaux sur [cette page](#).

## 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. Nous verrons pourquoi. Une variable booléenne peut être définie de plusieurs manières

:

```
1 //variable est fausse car elle vaut FALSE, du terme anglais "faux"
2 boolean variable = FALSE;
3 //variable est vraie car elle vaut TRUE, du terme anglais "vrai"
4 boolean variable = TRUE;
```

Quand une variable vaut "0", on peut considérer cette variable comme une variable booléenne, elle est donc fausse. En revanche, lorsqu'elle vaut "1" ou n'importe quelle valeurs différente de zéro, on peut aussi la considérer comme une variable booléenne, elle est donc vraie. Voilà un exemple :

```
1 //variable est fausse car elle vaut 0
2 int variable = 0;
3 //variable est vraie car elle vaut 1
4 int variable = 1;
5 //variable est vraie car sa valeur est différente de 0
6 int variable = 42;
```

Le langage Arduino accepte aussi une troisième forme d'écriture (qui lui sert pour utiliser les broches de sorties du microcontrôleur) :

```
1 //variable est à l'état logique bas (= traduction de "low"), donc 0
2 int variable = LOW;
3 //variable est à l'état logique haut (= traduction de "high"), donc 1
4 int variable = HIGH;
```

Nous nous servirons de cette troisième écriture pour allumer et éteindre des lumières...

## Les opérations "simples"

On va voir à présent les opérations qui sont possibles avec le langage Arduino (addition, multiplication, ...). Je vous vois tout de suite dire : "Mais pourquoi on fait ça, on l'a fait en primaire ! 😞" Et bien parce que c'est quelque chose d'essentiel, car on pourra ensuite faire des opérations avec des variables. Vous verrez, vous changerez d'avis après avoir lu la suite ! 😊

### L'addition

Vous savez ce que c'est, pas besoin d'explications. Voyons comment on fait cette opération avec le langage Arduino. Prenons la même variable que tout à l'heure :

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

Faisons maintenant une addition de variables :

```
1 //définition de la variable x et assignation à la valeur 38
2 int x = 38;
3 int y = 10;
4 int z = 0;
```



```
5 //faisons une addition
6 // on a donc z = 38 + 10 = 48
7 z = x + y;
```

## La soustraction

On peut reprendre les exemples précédents, en faisant une soustraction :

```
1 //définition de la variable x
2 int x = 0;
3
4 //on change la valeur de x par une opération simple
5 x = 12 - 3;
6 // x vaut maintenant 12 - 3 = 9
```

Soustraction de variables :

```
1 int x = 38; //définition de la variable x et assignation à la valeur 38
2 int y = 10;
3 int z = 0;
4
5 z = x - y; // on a donc z = 38 - 10 = 28
```

## La multiplication

```
1 int x = 0;
2 int y = 10;
3 int z = 0;
4
5 x = 12 * 3; // x vaut maintenant 12 * 3 = 36
6
7 z = x * y; // on a donc z = 36 * 10 = 360
8
9 // on peut aussi multiplier (ou toute autre opération) un nombre et une variable :
10
11 z = z * ( 1 / 10 ); //soit z = 360 * 0.1 = 36
```

## La division

```
1 float x = 0;
2 float y = 15;
3 float z = 0;
4
5 x = 12 / 2; // x vaut maintenant 12 / 2 = 6
6
7 z = y / x; // on a donc z = 15 / 6 = 2.5
```

Attention cependant, si vous essayer de stocker le résultat d'une division dans une variable de type char, int ou long, le résultat sera stocké sous la forme d'un entier arrondi au nombre inférieur. Par exemple dans le code précédent si on met z dans un int on aura :

```
1 float x = 0;
2 float y = 15;
3 int z = 0;
4
5 x = 12 / 2; // x vaut maintenant 12 / 2 = 6
```

```
6
7 z = y / x; // on a donc z = 15 / 6 = 2 !
```

## Le modulo

Après cette brève explication sur les opérations de base, passons à quelque chose de plus sérieux. Le modulo est une opération de base, certes moins connue que les autres. Cette opération permet d'obtenir le reste d'une division.

```
1 18 % 6 // le reste de l'opération est 0, car il y a 3*6 dans 18 donc 18 - 18 = 0
2 18 % 5 // le reste de l'opération est 3, car il y a 3*5 dans 18 donc 18 - 15 = 3
```

Le modulo est utilisé grâce au symbole %. C'est tout ce qu'il faut retenir. Autre exemple :

```
1 int x = 24;
2 int y = 6;
3 int z = 0;
4
5 z = x % y; // on a donc z = 24 % 6 = 0 (car 6 * 4 = 24)
```

Le modulo ne peut-être fait que sur des nombres entiers

## Quelques opérations bien pratiques

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

### L'incrémentation

Derrière ce nom barbare se cache une simple opération d'addition.

```
1 var = 0;
2 var++; //c'est cette ligne de code qui nous intéresse
```

“var++;” revient à écrire : “var = var + 1;” En fait, on ajoute le chiffre 1 à la valeur de *var*. Et si on répète le code un certain nombre de fois, par exemple 30, et bien on aura *var* = 30.

### La décrémentation

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

```
1 var = 30;
2 var--; //décrémentation de var
```

## Les opérations composées

Parfois il devient assez lassant de réécrire les mêmes chose et l'on sait que les programmeurs sont des gros fainéants ! 😊 Il existe des raccourcis lorsque l'on veut effectuer une opération sur une même variable :

```
1 int x, y;
```

```

2
3 x += y; // correspond à x = x + y;
4 x -= y; // correspond à x = x - y;
5 x *= y; // correspond à x = x * y;
6 x /= y; // correspond à x = x / y;

```

Avec un exemple, cela donnerait :

```

1 int var = 10;
2
3 //opération 1
4 var = var + 6;
5 var += 6; //var = 16
6
7 //opération 2
8 var = var - 6;
9 var -= 6; //var = 4
10
11 //opération 3
12 var = var * 6;
13 var *= 6; //var = 60
14
15 //opération 4
16 var = var / 5;
17 var /= 5; //var = 2

```

## L'opération de bascule (ou "inversion d'état")

Un jour, pour le projet du BAC, je devais (ou plutôt "je voulais") améliorer un code qui servait à programmer un module d'une centrale de gestion domestique. Mon but était d'afficher un choix à l'utilisateur sur un écran. Pour ce faire, il fallait que je réalise une **bascule programmée** (c'est comme ça que je la nomme maintenant). Et après maintes recherches et tests, j'ai réussi à trouver ! Et il s'avère que cette "opération", si l'on peut l'appeler ainsi, est très utile dans certains cas. Nous l'utiliserons notamment lorsque l'on voudra faire clignoter une lumière. Sans plus attendre, voilà cette astuce :

```

1 //on définit une variable x qui ne peut prendre que la valeur 0 ou 1 (vraie ou fausse)
2 boolean x = 0;
3
4 x = 1 - x; //c'est la toute l'astuce du programme !

```

Analysons cette instruction. A chaque exécution du programme (oui, j'ai omis de vous le dire, il se répète jusqu'à l'infini), la variable x va changer de valeur :

- 1<sup>er</sup> temps :  $x = 1 - x$  soit  $x = 1 - 0$  donc  $x = 1$
- 2<sup>e</sup> temps :  $x = 1 - x$  or x vaut maintenant 1 donc  $x = 1 - 1$  soit  $x = 0$
- 3<sup>e</sup> temps : x vaut 0 donc  $x = 1 - 0$  soit  $x = 1$

Ce code se répète donc et à chaque répétition, la variable x change de valeur et passe de 0 à 1, de 1 à 0, de 0 à 1, etc. Il agit bien comme une bascule qui change la valeur d'une variable booléenne. En mode console cela donnerait quelque chose du genre (n'essayez pas cela ne marchera pas, c'est un exemple) :

```

1 x = 0
2 x = 1

```

```
3 x = 0
4 x = 1
5 x = 0
6 ...
```

Mais il existe d'autres moyens d'arriver au même résultat. Par exemple, en utilisant l'opérateur '!' qui signifie "not" ("non"). Ainsi, avec le code suivant on aura le même fonctionnement :

```
1 x = !x;
```

Puisqu'à chaque passage x devient "pas x" donc si x vaut 1 son contraire sera 0 et s'il vaut 0, il deviendra 1.

## Les conditions

### Qu'est-ce qu'une condition

C'est un choix que l'on fait entre plusieurs propositions. En informatique, les conditions servent à tester des variables. Par exemple : *Vous faites une recherche sur un site spécialisé pour acheter une nouvelle voiture. Vous imposez le prix de la voiture qui doit être inférieur à 5000€ (c'est un petit budget 🤖). Le programme qui va gérer ça va faire appel à un **test conditionnel**. Il va éliminer tous les résultats de la recherche dont le prix est supérieur à 5000€.*

### Quelques symboles

Pour tester des variables, il faut connaître quelques symboles. Je vous ai fait un joli tableau pour que vous vous repérez bien :

Symbole	A quoi il sert	Signification
==	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...

"Et si on s'occupait des conditions ? Ou bien sinon on va tranquillement aller boire un bon café ?" Comment décortiquer cette phrase ? Mmm... 🤖 Ha ! Je sais ! Cette phrase implique un choix : le premier choix est de s'occuper des conditions. Si l'interlocuteur dit oui, alors il s'occupe des conditions. Mais s'il dit non, alors il va boire un bon café. Il a donc l'obligation d'effectuer une action sur les deux proposées. En informatique, on parle de **condition**.

“si la condition est vraie”, on fait une action. En revanche “si la condition est fausse”, on exécute une autre action.

## If...else

La première condition que nous verrons est la condition if...else. Voyons un peu le fonctionnement.

### if

On veut tester la valeur d’une variable. Prenons le même exemple que tout à l’heure. Je veux tester si la voiture est inférieure à 5000€.

```
1 int prix_voiture = 4800; //variable : prix de la voiture définit à 4800€
```

D’abord on définit la variable “prix\_voiture”. Sa valeur est de 4800€. Ensuite, on doit tester cette valeur. Pour tester une condition, on emploie le terme *if* (de l’anglais “si”). Ce terme doit être suivi de parenthèses dans lesquelles se trouveront les variables à tester. Donc entre ces parenthèses, nous devons tester la variable prix\_voiture afin de savoir si elle est inférieure à 5000€.

```
1 if(prix_voiture < 5000)
2 {
3     //la condition est vraie, donc j'achète la voiture
4 }
```

On peut lire cette ligne de code comme ceci : “**si** la variable *prix\_voiture* est inférieure à 5000, on exécute le code qui se trouve entre les accolades.

Les instructions qui sont entre les accolades ne seront exécutées que si la condition testée est vraie !

Le “schéma” à suivre pour tester une condition est donc le suivant :

```
1 if(/* contenu de la condition à tester */)
2 {
3     //instructions à exécuter si la condition est vraie
4 }
```

### else

On a pour l’instant testé que si la condition est vraie. Maintenant, nous allons voir comment faire pour que d’autres instructions soient exécutées si la condition est fausse. Le terme *else* de l’anglais “sinon” implique notre deuxième choix si la condition est fausse. *Par exemple, si le prix de la voiture est inférieur à 5000€, alors je l’achète. Sinon, je ne l’achète pas.* Pour traduire cette phrase en ligne de code, c’est plus simple qu’avec un if, il n’y a pas de parenthèses à remplir :

```
1 int prix_voiture = 5500;
2
3 if(prix_voiture < 5000)
4 {
5     //la condition est vraie, donc j'achète la voiture
```

```

6 }
7 else
8 {
9     //la condition est fausse, donc je n'achète pas la voiture
10 }

```

Le *else* est généralement utilisé pour les **conditions dites de défaut**. C'est lui qui a le pouvoir sur toutes les conditions, c'est-à-dire que si aucune condition n'est vraie, on exécute les instructions qu'il contient.

Le *else* n'est pas obligatoire, on peut très bien mettre plusieurs *if* à la suite.

Le "schéma" de principe à retenir est le suivant :

```

1 else // si toutes les conditions précédentes sont fausses...
2 {
3     //...on exécute les instructions entre ces accolades
4 }

```

### *else if*

A ce que je vois, on a pas trop le choix : soit la condition est vraie, soit elle est fausse. Il n'y a pas d'autres possibilités ? o\_O

Bien sur que l'on peut tester d'autres conditions ! Pour cela, on emploie le terme *else if* qui signifie "sinon si..." *Par exemple, SI le prix de la voiture est inférieur à 5000€ je l'achète; SINON SI elle est égale à 5500€ mais qu'elle a l'option GPS en plus, alors je l'achète ; SINON je ne l'achète pas.* Le *sinon si* s'emploie comme le *if* :

```

1 int prix_voiture = 5500;
2
3 if(prix_voiture < 5000)
4 {
5     //la condition est vraie, donc j'achète la voiture
6 }
7 else if(prix_voiture == 5500)
8 {
9     //la condition est vraie, donc j'achète la voiture
10 }
11 else
12 {
13     //la condition est fausse, donc je n'achète pas la voiture
14 }

```

A retenir donc, si la première condition est fausse, on teste la deuxième, si la deuxième est fausse, on teste la troisième, etc. "Schéma" de principe du *else*, idem au *if* :

```

1 else if(/* test de la condition */) //si elle est vraie...
2 {
3     //...on exécute les instructions entre ces accolades
4 }

```

Le "else if" ne peut pas être utilisée toute seule, il faut obligatoirement qu'il y ait un "if" avant !

# Les opérateurs logiques

Et si je vous posais un autre problème ? Comment faire pour savoir si la voiture est inférieure à 5000€ ET si elle est grise ? 🤔

C'est vrai ça, si je veux que la voiture soit grise en plus d'être inférieure à 5000€, comment je fais ?

Il existe des opérateurs qui vont nous permettre de tester cette condition ! Voyons quels sont ses opérateurs puis testons-les !

## Opérateur Signification

&& ... ET ...

|| ... OU ...

! NON

**ET**

Reprenons ce que nous avons testé dans le *else if* : *Si la voiture vaut 5500€ ET qu'elle a l'option GPS en plus, ALORS je l'achète*. On va utiliser un *if* et un opérateur logique qui sera le **ET** :

```
1 int prix_voiture = 5500;
2 int option_GPS = TRUE;
3
4 /* l'opérateur && lie les deux conditions qui doivent être
5 vraies ensemble pour que la condition soit remplie */
6 if(prix_voiture == 5500 && option_GPS)
7 {
8     //j'achète la voiture si la condition précédente est vraie
9 }
```

**OU**

On peut reprendre la condition précédente et la première en les assemblant pour rendre le code beaucoup moins long.

Et oui, les programmeurs sont des flemmards ! 🤪

Rappelons quelles sont ces conditions :

```
1 int prix_voiture = 5500;
2 int option_GPS = TRUE;
3
4 if(prix_voiture < 5000)
5 {
6     //la condition est vraie, donc j'achète la voiture
7 }
8 else if(prix_voiture == 5500 && option_GPS)
9 {
10    //la condition est vraie, donc j'achète la voiture
11 }
12 else
13 {
```



```
14 //la condition est fausse, donc je n'achète pas la voiture
15 }
```

Vous voyez bien que l'instruction dans le *if* et le *else if* est la même. Avec un opérateur logique, qui est le OU, on peut rassembler ces conditions :

```
1 int prix_voiture = 5500;
2 int option_GPS = TRUE;
3
4 if((prix_voiture < 5000) || (prix_voiture == 5500 && option_GPS))
5 {
6     //la condition est vraie, donc j'achète la voiture
7 }
8 else
9 {
10    //la condition est fausse, donc je n'achète pas la voiture
11 }
```

Lisons la condition testée dans le if : “Si le prix de la voiture est inférieur à 5000€ OU Si le prix de la voiture est égal à 5500€ ET la voiture à l’option GPS en plus, ALORS j’achète la voiture”.

Attention aux parenthèses qui sont à bien placer dans les conditions, ici elles n’étaient pas nécessaires, mais elles aident à mieux lire le code. 😊

## NON

Moi j’aimerais tester “si la condition est fausse j’achète la voiture”. Comment faire ?

~~Toi t'as un souci~~ Il existe un dernier opérateur logique qui se prénomme NON. Il permet en effet de tester si la condition est fausse :

```
1 int prix_voiture = 5500;
2
3 if(!(prix_voiture < 5000))
4 {
5     //la condition est vraie, donc j'achète la voiture
6 }
```

Se lit : “Si le prix de la voiture N’EST PAS inférieur à 5000€, alors j’achète la voiture”. On s’en sert avec le caractère ! (point d’exclamation), généralement pour tester des variables booléennes. On verra dans les boucles que ça peut grandement simplifier le code.

## Switch

Il existe un dernier test conditionnel que nous n’avons pas encore abordé, c’est le *switch*. Voilà un exemple :

```
1 int options_voiture = 0;
2
3 if(options_voiture == 0)
4 {
5     //il n'y a pas d'options dans la voiture
6 }
```

```

7  if(options_voiture == 1)
8  {
9      //la voiture a l'option GPS
10 }
11 if(options_voiture == 2)
12 {
13     //la voiture a l'option climatisation
14 }
15 if(options_voiture == 3)
16 {
17     //la voiture a l'option vitre automatique
18 }
19 if(options_voiture == 4)
20 {
21     //la voiture a l'option barres de toit
22 }
23 if(options_voiture == 5)
24 {
25     //la voiture a l'option  siège éjectable
26 }
27 else
28 {
29     //retente ta chance ;-)
30 }

```

Ce code est indigérable ! C'est infâme ! Grotesque ! Pas beau ! En clair, il faut trouver une solution pour changer cela. Cette solution existe, c'est le *switch*. Le *switch*, comme son nom l'indique, va tester la variable jusqu'à la fin des valeurs qu'on lui aura données. Voici comment cela se présente :

```

1  int options_voiture = 0;
2
3  switch (options_voiture)
4  {
5      case 0:
6          //il n'y a pas d'options dans la voiture
7          break;
8      case 1:
9          //la voiture a l'option GPS
10         break;
11     case 2:
12         //la voiture a l'option climatisation
13         break;
14     case 3:
15         //la voiture a l'option vitre automatique
16         break;
17     case 4:
18         //la voiture a l'option barres de toit
19         break;
20     case 5:
21         //la voiture a l'option siège éjectable
22         break;
23     default:
24         //retente ta chance ;-)
25         break;
26 }

```

Si on testait ce code, en réalité cela ne fonctionnerait pas car il n'y a pas d'instruction pour afficher à l'écran, mais nous aurions quelque chose du genre :

```
1 il n'y a pas d'options dans la voiture
```

Si option\_voiture vaut maintenant 5 :

```
1 la voiture a l'option siège éjectable
```

L'instruction **break** est hyper importante, car si vous ne la mettez pas, l'ordinateur, ou plutôt la carte Arduino, va exécuter toutes les instructions. Pour éviter cela, on met cette instruction break, qui vient de l'anglais "casser/arrêter" pour dire à la carte Arduino qu'il faut arrêter de tester les conditions car on a trouvé la valeur correspondante.

## La condition ternaire ou condensée

Cette condition est en fait une simplification d'un test if...else. Il n'y a pas grand-chose à dire dessus, par conséquent un exemple suffira : Ce code :

```
1 int prix_voiture = 5000;
2 int achat_voiture = FALSE;
3
4 if(prix_voiture == 5000) //si c'est vrai
5 {
6     achat_voiture = TRUE; //on achète la voiture
7 }
8 else //sinon
9 {
10     achat_voiture = FALSE; //on n'achète pas la voiture
11 }
```

Est équivalent à celui-ci :

```
1 int prix_voiture = 5000;
2 int achat_voiture = FALSE;
3
4 achat_voiture= (prix_voiture == 5000) ? TRUE : FALSE;
```

Cette ligne :

```
1 achat_voiture= (prix_voiture == 5000) ? TRUE : FALSE;
```

Se lit comme ceci : "Est-ce que le prix de la voiture est égal à 5000€ ? Si oui, alors j'achète la voiture SINON je n'achète pas la voiture"

Bon, vous n'êtes pas obligé d'utiliser cette condition ternaire, c'est ~~vraiment pour les gros flemmards~~ juste pour simplifier le code, mais pas forcément la lecture de ce dernier.

Nous n'avons pas encore fini avec le langage Arduino. Je vous invite donc à passer à la partie suivante pour poursuivre l'apprentissage de ce langage.

## [Arduino 106] Le langage Arduino (2/2)

J'ai une question. Si je veux faire que le code que j'ai écrit se répète, je suis obligé

de le recopier autant de fois que je veux ? Ou bien il existe une solution ? o\_O

Voilà une excellente question qui introduit le chapitre que vous allez commencer à lire car c'est justement l'objet de ce chapitre. Nous allons voir comment faire pour qu'un bout de code se répète. Puis nous verrons, ensuite, comment organiser notre code pour que celui-ci devienne plus lisible et facile à déboguer. Enfin, nous apprendrons à utiliser les tableaux qui nous seront très utiles. Voilà le programme qui vous attend ! 😊

## Les boucles

### Qu'est-ce qu'une boucle ?

En programmation, une **boucle** est une instruction qui permet de répéter un bout de code. Cela va nous permettre de faire se répéter un bout de programme ou un programme entier. 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**.* Pour résoudre le problème posé, il va falloir que l'on utilise une boucle.

```
1  /* ICI, un bout de programme permet de faire les choses suivantes :
2  _ un capteur détecte la tombée de la nuit et la levée du jour
3  o Si c'est la nuit, alors on doit fermer le volet
4  o Sinon, si c'est le jour, on doit ouvrir le volet
5
6  _ le programme lit l'état du capteur qui indique si le volet est ouvert ou fermé
7
8  _ enregistrement de cet état dans la variable de type String : position_volet
9  o Si le volet est ouvert, alors : position_volet = "ouvert";
10 o Sinon, si le volet est fermé : position_volet = "ferme";
11 */
12
13 while(position_volet == "ouvert")
14 {
15     //instructions qui font descendre le volet
16 }
```

### Comment lire ce code ?

En anglais, le mot *while* signifie “tant que”. Donc si on lit la ligne :

```
1 while(position_volet == "ouvert") { /* instructions */ }
```

Il faut la lire : “TANT QUE la position du volet est *ouvert*”, on boucle/répète les instructions de la boucle (entre les accolades).

## Construction d'une boucle while

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

```
1 while(/* condition à tester */)
2 {
3     //les instructions entre ces accolades sont répétées tant que la condition est vraie
4 }
```

## Un exemple

Prenons un exemple simple, réalisons un compteur !

```
1 //variable compteur qui va stocker le nombre de fois que la boucle
2 int compteur = 0;
3 //aura été exécutée
4
5 while(compteur != 5) //tant que compteur est différent de 5, on boucle
6 {
7     compteur++; //on incrémente la variable compteur à chaque tour de boucle
8 }
```

Si on teste ce code (dans la réalité rien ne s'affiche, c'est juste un exemple pour vous montrer), cela donne :

```
1 compteur = 0
2 compteur = 1
3 compteur = 2
4 compteur = 3
5 compteur = 4
6 compteur = 5
```

Donc au départ, la variable *compteur* vaut 0, on exécute la boucle et on incrémente *compteur*. Mais *compteur* ne vaut pour l'instant que 1, donc on ré-exécute la boucle. Maintenant *compteur* vaut 2. On répète la boucle, ... jusqu'à 5. Si *compteur* vaut 5, la boucle n'est pas ré-exécutée et on continue le programme. Dans notre cas, le programme se termine.

## La boucle do...while

Cette boucle est similaire à la précédente. Mais il y a une différence qui a son importance ! En effet, si on prête attention à la place la condition dans la boucle *while*, on s'aperçoit qu'elle est testée avant de rentrer dans la boucle. Tandis que dans une boucle *do...while*, la condition est testée seulement lorsque le programme est rentré dans la boucle :

```
1 do
2 {
3     //les instructions entre ces accolades sont répétées TANT QUE la condition est vraie
4 }while(/* condition à tester */);
```

Le mot *do* vient de l'anglais et se traduit par *faire*. Donc la boucle *do...while* signifie "faire les instructions, tant que la condition testée est fausse". Tandis que dans une boucle *while* on pourrait dire : "tant que la condition est fausse, fais ce qui suit".

Qu'est-ce que ça change ?

Et bien, dans une *while*, si la condition est vraie dès le départ, on entrera jamais dans cette boucle. A l'inverse, avec une boucle *do...while*, on entre dans la boucle puis on test la condition. Reprenons notre compteur :

```
1 //variable compteur = 5
2 int compteur = 5;
3
4 do
5 {
6     compteur++; //on incrémente la variable compteur à chaque tour de boucle
7 }while(compteur < 5); //tant que compteur est inférieur à 5, on boucle
```

Dans ce code, on définit dès le départ la valeur de *compteur* à 5. Or, le programme va rentrer dans la boucle alors que la condition est fausse. Donc **la boucle est au moins exécutée une fois** ! Et ce quelle que soit la véracité de la condition. En test cela donne :

```
1 compteur = 6
```

## Concaténation

Une boucle est une instruction qui a été répartie sur plusieurs lignes. Mais on peut l'écrire sur une seule ligne :

```
1 //variable compteur = 5
2 int compteur = 5;
3
4 do{compteur++;}while(compteur < 5);
```

C'est pourquoi il ne faut pas oublier le point virgule à la fin (après le *while*). Alors que dans une simple boucle *while* le point virgule **ne doit pas** être mis !

## La boucle *for*

Voilà une boucle bien particulière. Ce qu'elle va nous permettre de faire est assez simple. Cette boucle est exécutée X fois. Contrairement aux deux boucles précédentes, on doit lui donner trois paramètres.

```
1 for(int compteur = 0; compteur < 5; compteur++)
2 {
3     //code à exécuter
4 }
```

## Fonctionnement

```
1 for(int compteur = 0; compteur < 5; compteur++)
```

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
- suivit de la définition de la condition à tester
- suivit de l'instruction à exécuter

Donc, si on lit cette ligne : “POUR compteur = 0 et compteur inférieur à 5, on incrémente compteur”. De façon plus concise, la boucle est exécutée autant de fois qu'il sera nécessaire à *compteur* pour arriver à 5. Donc ici, le code qui se trouve à l'intérieur de la boucle sera exécuté 5 fois.

## A retenir

La structure de la boucle :

```
1 for(/*initialisation de la variable*/ ; /*condition à laquelle la boucle s'arrête*/ ;
```

## La boucle infinie

La boucle infinie est très simple à réaliser, d'autant plus qu'elle est parfois très utile. Il suffit simplement d'utiliser une *while* et de lui assigner comme condition une valeur qui ne change jamais. En l'occurrence, on met souvent le chiffre 1.

```
1 while(1)
2 {
3     //instructions à répéter jusqu'à l'infinie
4 }
```

On peut lire : “TANT QUE la condition est égale à 1, on exécute la boucle”. Et cette condition sera toujours remplie puisque “1” n'est pas une variable mais bien un chiffre. Également, il est possible de mettre tout autre chiffre entier, ou bien le booléen “TRUE” :

```
1 while(TRUE)
2 {
3     //instructions à répéter jusqu'à l'infinie
4 }
```

Cela ne fonctionnera pas avec la valeur 0. En effet, 0 signifie “condition fausse” donc la boucle s'arrêtera aussitôt...

La fonction `loop()` se comporte comme une boucle infinie, puisqu'elle se répète après avoir fini d'exécuter ses tâches.

---

# Les fonctions

Dans un programme, les lignes sont souvent très nombreuses. Il devient alors impératif de séparer le programme en petits bouts afin d'améliorer la lisibilité de celui-ci, en plus



d'améliorer le fonctionnement et de faciliter le débogage. Nous allons voir ensemble ce qu'est une fonction, puis nous apprendrons à les créer et les appeler.

## Qu'est-ce qu'une fonction ?

Une **fonction** est un “conteneur” mais différent des variables. En effet, une variable ne peut contenir qu'un nombre, tandis qu'une fonction peut contenir un programme entier ! Par exemple ce code est une fonction :

```
1 void setup()  
2 {  
3     //instructions  
4 }
```

En fait, lorsque l'on va programmer notre carte Arduino, on va écrire notre programme dans des fonctions. Pour l'instant nous n'en connaissons que 2 : **setup()** et **loop()**. Dans l'exemple précédent, à la place du commentaire, on peut mettre des instructions (conditions, boucles, variables, ...). C'est ces instructions qui vont constituer le programme en lui même. Pour être plus concret, une fonction est un bout de programme qui permet de réaliser une tâche bien précise. Par exemple, pour mettre en forme un texte, on peut colorier un **mot** en bleu, mettre le **mot** en gras ou encore grossir ce **mot**. A chaque fois, on a utilisé une fonction :

- *gras*, pour mettre le mot en gras
- *colorier*, pour mettre le mot en bleu
- *grossir*, pour augmenter la taille du mot

En programmation, on va utiliser des fonctions. Alors ces fonctions sont “réparties dans deux grandes familles”. Ce que j'entends par là, c'est qu'il existe des fonctions toutes prêtes dans le langage Arduino et d'autres que l'on va devoir créer nous même. C'est ce dernier point qui va nous intéresser.

On ne peut pas écrire un programme sans mettre de fonctions à l'intérieur ! On est obligé d'utiliser la fonction *setup()* et *loop()* (même si on ne met rien dedans). Si vous écrivez des instructions en dehors d'une fonction, le logiciel Arduino refusera systématiquement de compiler votre programme. Il n'y a que les variables globales que vous pourrez déclarer en dehors des fonctions.

J'ai pas trop compris à quoi ça sert ? o\_O

L'utilité d'une fonction réside dans sa capacité à simplifier le code et à le séparer en “petits bouts” que l'on assemblera ensemble pour créer le programme final. Si vous voulez, c'est un peu comme les jeux de construction en plastique : chaque pièce à son propre mécanisme et réalise une fonction. Par exemple une roue permet de rouler ; un bloc permet de réunir plusieurs autres blocs entre eux ; un moteur va faire avancer l'objet créé... Et bien tous ces éléments seront assemblés entre eux pour former un objet (voiture, maison, ...). Tout comme, les fonctions seront assemblées entre elles pour former un programme. On aura par exemple la fonction : “mettre au carré un nombre” ; la fonction : “additionner a + b” ; etc. Qui au final donnera le résultat souhaité.

## Fabriquer une fonction

Pour fabriquer une fonction, nous avons besoin de savoir trois choses :

- Quel est le **type** de la fonction que je souhaite créer ?
- Quel sera son **nom** ?
- Quel(s) **paramètre(s)** prendra-t-elle ?

## Nom de la fonction

Pour commencer, nous allons, en premier lieu, choisir le nom de la fonction. Par exemple, si votre fonction doit récupérer la température d'une pièce fournie par un capteur de température : vous appellerez la fonction *lireTemperaturePiece*, ou bien *lire\_temperature\_piece*, ou encore *lecture\_temp\_piece*. Bon, des noms on peut lui en donner plein, mais soyez logique quant au choix de ce dernier. Ce sera plus facile pour comprendre le code que si vous l'appellez *tmp* (pour température 😊).

Un nom de fonction explicite garantit une lecture rapide et une compréhension aisée du code. Un lecteur doit savoir ce que fait la fonction juste grâce à son nom, sans lire le contenu !

## Les types et les paramètres

Les fonctions ont pour but de découper votre programme en différentes unités logiques. Idéalement, le programme principal ne devrait utiliser que des appels de fonctions, en faisant un minimum de traitement. Afin de pouvoir fonctionner, elles utilisent, la plupart du temps, des “choses” en *entrées* et renvoient “quelque chose” en *sortie*. Les entrées seront appelées des **paramètres de la fonction** et la sortie sera appelée **valeur de retour**.

Notez qu'une fonction ne peut renvoyer qu'un seul résultat à la fois. Notez également qu'une fonction ne renvoie pas obligatoirement un résultat. Elle n'est pas non plus obligée d'utiliser des paramètres.

## Les paramètres

Les paramètres servent à nourrir votre fonction. Ils servent à donner des informations au traitement qu'elle doit effectuer. Prenons un exemple concret. Pour changer l'état d'une sortie du microcontrôleur, Arduino nous propose la fonction suivante: [digitalWrite\(pin, value\)](#). Ainsi, la référence nous explique que la fonction a les caractéristiques suivantes:

- - paramètre *pin*: le numéro de la broche à changer
- - paramètre *value*: l'état dans lequel mettre la broche (HIGH, (haut, +5V) ou LOW (bas, masse))
- - retour: pas de retour de résultat

Comme vous pouvez le constater, l'exemple est explicite sans lire le code de la fonction. Son nom, *digitalWrite* (“écriture digitale” pour les anglophobes), signifie qu'on va changer l'état d'une broche *numérique* (donc pas analogique). Ses paramètres ont eux aussi des noms explicites, *pin* pour la broche à changer et *value* pour l'état à lui donner. Lorsque vous allez créer des fonctions, c'est à vous de voir si elles ont besoin

de paramètres ou non. Par exemple, vous voulez faire une fonction qui met en pause votre programme, vous pouvez faire une fonction `Pause()` et déterminera la durée pendant laquelle le programme sera en pause. On obtiendra donc, par exemple, la syntaxe suivante : `void Pause(char duree)`. Pour résumer un peu, on a le choix de créer des **fonctions vides**, donc sans paramètres, ou bien des **fonctions “typées”** qui acceptent un ou plusieurs paramètres.

Mais c’est quoi ça “void” ?

J’y arrive ! Souvenez vous, un peu plus haut je vous expliquais qu’une fonction pouvait retourner une valeur, la fameuse valeur de sortie, je vais maintenant vous expliquer son fonctionnement.

## Le type void

On vient de voir qu’une fonction pouvait accepter des paramètres et éventuellement renvoyer quelque chose. Mais ce n’est pas obligatoire. En effet, si l’on reprend notre fonction “Pause”, elle ne renvoie rien car ce n’est pas nécessaire de signaler quoi que ce soit. Dans ce cas, on préfixera le nom de notre fonction avec le mot-clé “void”. La syntaxe utilisée est la suivante :

```
1 void nom_de_la_fonction()  
2 {  
3     //instructions  
4 }
```

On utilise donc le type `void` pour dire que la fonction n’aura pas de retour. Une fonction de type void ne peut donc pas retourner de valeur. Par exemple :

```
1 void fonction()  
2 {  
3     int var = 24;  
4     return var; //ne fonctionnera pas car la fonction est de type void  
5 }
```

Ce code ne fonctionnera pas, parce que la fonction `int`. Ce qui est impossible ! Le compilateur le refusera et votre code final ne sera pas généré. Vous connaissez d’ailleurs déjà au moins deux fonctions qui n’ont pas de retour... Et oui, la fonction “set” et la fonction “loop” 😊 . Il n’y en a pas plus à savoir. 😊

## Les fonctions “typées”

Là, cela devient légèrement plus intéressant. En effet, si on veut créer une fonction qui calcule le résultat d’une addition de deux nombres (ou un calcul plus complexe), il serait bien de pouvoir renvoyer directement le résultat plutôt que de le stocker dans une variable qui a une portée globale et d’accéder à cette variable dans une autre fonction. En clair, l’appel de la fonction nous donne directement le résultat. On peut alors faire “ce que l’on veut” avec ce résultat (le stocker dans une variable, l’utiliser dans une fonction, lui faire subir une opération, ...)

**Comment créer une fonction typée ?**

En soit, cela n'a rien de compliqué, il faut simplement remplacer `long`, ...) Voilà un exemple :

```
1 int maFonction()
2 {
3     int resultat = 44; //déclaration de ma variable resultat
4     return resultat;
5 }
```

Notez que je n'ai pas mis les deux fonctions principales, à savoir `loop()`, mais elles sont obligatoires ! Lorsqu'elle sera appelée, la fonction `resultat`. Voyez cet exemple :

```
1 int calcul = 0;
2
3 void loop()
4 {
5     calcul = 10 * maFonction();
6 }
7
8 int maFonction()
9 {
10    int resultat = 44; //déclaration de ma variable resultat
11    return resultat;
12 }
```

Dans la fonction `calcul = 10 * 44`; Ce qui nous donne : `calcul = 440`. Bon ce n'est qu'un exemple très simple pour vous montrer un peu comment cela fonctionne. Plus tard, lorsque vous serez au point, vous utiliserez certainement cette combinaison de façon plus complexe. 😊

Comme cet exemple est très simple, je n'ai pas inscrit la valeur retournée par la fonction `maFonction()` dans une variable, mais il est préférable de le faire. Du moins, lorsque c'est utile, ce qui n'est pas le cas ici.

## Les fonctions avec paramètres

C'est bien gentil tout ça, mais maintenant vous allez voir quelque chose de bien plus intéressant. Voilà un code, nous verrons ce qu'il fait après :

```
1 int x = 64;
2 int y = 192;
3
4 void loop()
5 {
6     maFonction(x, y);
7 }
8
9 int maFonction(int param1, int param2)
10 {
11     int somme = 0;
12     somme = param1 + param2;
13     //somme = 64 + 192 = 255
14
15     return somme;
16 }
```

## Que se passe-t-il ?

J'ai défini trois variables : `maFonction()` est "typée" et accepte des **paramètres**. Lisons le code du début :

- On déclare nos variables
- La fonction `maFonction()` que l'on a créée

C'est sur ce dernier point que l'on va se pencher. En effet, on a donné à la fonction des paramètres. Ces paramètres servent à "nourrir" la fonction. Pour faire simple, on dit à la fonction : *"Voilà deux paramètres, je veux que tu t'en serves pour faire le calcul que je veux"* Ensuite arrive la signature de la fonction.

## La signature... de quoi tu parles ?

La signature c'est le "titre complet" de la fonction. Grâce à elle on connaît le **nom** de la fonction, le **type** de la valeur retournée, et le type des différents **paramètres**.

```
1 int maFonction(int param1, int param2)
```

La fonction récupère dans des variables les paramètres que l'on lui a envoyés. Autrement dit, dans la variable `y`. Soit : `param2 = y = 192`. Pour finir, on utilise ces deux variables créées "à la volée" dans la signature de la fonction pour réaliser le calcul souhaité (une somme dans notre cas).

A quoi ça sert de faire tout ça ? Pourquoi on utilise pas simplement les variables `x` et `y` dans la fonction ?

Cela va nous servir à simplifier notre code. Mais pas seulement ! Par exemple, vous voulez faire plusieurs opérations différentes (addition, soustraction, etc.) et bien au lieu de créer plusieurs fonctions, on ne va en créer qu'une qui les fait toutes ! Mais, afin de lui dire quelle opération faire, vous lui donnerez un paramètre lui disant : *"Multiplie ces deux nombres"* ou bien *"additionne ces deux nombres"*. Ce que cela donnerait :

```
1 unsigned char operation = 0;
2 int x = 5;
3 int y = 10;
4
5 void loop()
6 {
7     //le paramètre "opération" donne le type d'opération à faire
8     maFonction(x, y, operation);
9 }
10
11 int maFonction(int param1, int param2, int param3)
12 {
13     int resultat = 0;
14     switch(param3)
15     {
16         case 0 : //addition, resultat = 15
17             resultat = param1 + param2;
18             break;
19         case 1 : //soustraction, resultat = -5
20             resultat = param1 - param2;
```

```

21     break;
22     case 2 : //multiplication, resultat = 50
23         resultat = param1 * param2;
24         break;
25     case 3 : //division, resultat = 0 (car nombre entier)
26         resultat = param1 / param2;
27         break;
28     default :
29         resultat = 0;
30         break;
31 }
32
33 return resultat;
34 }

```

Donc si la variable `x`. Simple à comprendre, n'est-ce pas ? 😊

## Les tableaux

Comme son nom l'indique, cette partie va parler des tableaux.

Quel est l'intérêt de parler de cette surface ennuyeuse qu'utilisent nos chers enseignants ?

Eh bien détrompez-vous, en informatique un tableau ça n'a rien à voir ! Si on devait (beaucoup) résumer, un tableau est une grosse variable. Son but est de **stocker des éléments de mêmes types en les mettant dans des cases**. Par exemple, un prof qui stocke les notes de ses élèves. Il utilisera un tableau de *float* (nombre à virgule), avec une case par élèves. Nous allons utiliser cet exemple tout au long de cette partie. Voici quelques précisions pour bien tout comprendre :

- chaque élève sera identifié par un numéro allant de 0 (le premier élève) à 19 (le vingtième élève de la classe)
- on part de 0 car en informatique la première valeur dans un tableau est 0 !

### Un tableau en programmation

Un tableau, tout comme sous Excel, c'est un ensemble constitué de cases, lesquels vont contenir des informations. En programmation, ces informations seront des **nombres**. Chaque case d'un tableau contiendra une valeur. En reprenant l'exemple des notes des élèves, le tableau répertoriant les notes de chaque élève ressemblerait à ceci :

élève 0	élève 1	élève 2	[...]	élève n-1	élève n
10	15,5	8	[...]	18	7

### A quoi ça sert ?

On va principalement utiliser des tableaux lorsque l'on aura besoin de stocker des informations sans pour autant créer une variable pour chaque information. Toujours avec le même exemple, au lieu de créer une variable `eleve2` et ainsi de suite pour chaque élève, on inscrit les notes des élèves dans un tableau.

Mais, concrètement c'est quoi un tableau : une variable ? une fonction ?

Ni l'un, ni l'autre. En fait, on pourrait comparer cela avec un index qui pointe vers les valeurs de variables qui sont contenus dans chaque case du tableau. Un petit schéma pour simplifier :

#### élève 0

variable dont on ne connaît pas le nom mais qui stocke une valeur

#### élève 1

variable différente de la case précédente

Par exemple, cela donnerait :

#### élève 0

variable `note_eleve0`

#### élève 1

variable `note_eleve1`

Avec notre exemple :

#### élève 0 élève 1

10      15,5

Soit, lorsque l'on demandera la valeur de la case 1 (correspondant à la note de l'élève 1), le tableau nous renverra le nombre : 15,5. Alors, dans un premier temps, on va voir comment déclarer un tableau et l'initialiser. Vous verrez qu'il y a différentes manières de procéder. Après, on finira par apprendre comment utiliser un tableau et aller chercher des valeurs dans celui-ci. Et pour finir, on terminera ce chapitre par un exemple. Y'a encore du boulot ! 😊

## Déclarer un tableau

Comme expliqué plus tôt, un tableau contient des éléments de même type. On le déclare donc avec un type semblable, et une taille représentant le nombre d'éléments qu'il contiendra. Par exemple, pour notre classe de 20 étudiants :

```
1 float notes[20];
```

On peut également créer un tableau vide, la syntaxe est légèrement différente :

```
1 float notes[] = {};
```

On veut stocker des notes, donc des valeurs décimales entre 0 et 20. On va donc créer un tableau de float (car c'est le type de variable qui accepte les nombres à virgule, souvenez-vous ! 😊). Dans cette classe, il y a 20 élèves (de 0 à 19) donc le tableau contiendra 20 éléments. Si on voulait faire un tableau de 100 étudiants dans lesquels on recense leurs nombres d'absence, on ferait le tableau suivant:

```
1 char absentisme[100];
```

## Accéder et modifier une case du tableau



Pour accéder à une case d'un tableau, il suffit de connaître l'**indice** de la case à laquelle on veut accéder. L'indice c'est le numéro de la case qu'on veut lire/écrire. Par exemple, pour lire la valeur de la case 10 (donc indice 9 car on commence à 0):

```
1 float notes[20]; //notre tableau
2 float valeur; //une variable qui contiendra une note
3
4 //valeur contient désormais la note du dixième élève
5 valeur = notes[9];
```

Ce code se traduit par l'enregistrement de la valeur contenue dans la dixième case du tableau, dans une variable nommée **valeur**. A présent, si on veut aller modifier cette même valeur, on fait comme avec une variable normale, il suffit d'utiliser l'opérateur ' = ' :

```
1 notes[9] = 10,5; //on change la note du dixième élève
```

En fait, on procède de la même manière que pour changer la valeur d'une variable, car, je vous l'ai dit, chaque case d'un tableau est une variable qui contient une valeur ou non.

Faites attention aux indices utilisés. Si vous essayez de lire/écrire dans une case de tableau trop loin (indice trop grand, par exemple 987362598412 😊), le comportement pourrait devenir imprévisible. Car en pratique vous modifierez des valeurs qui seront peut-être utilisées par le système pour autre chose. Ce qui pourrait avoir de graves conséquences !

Vous avez sûrement rencontré des crashes de programme sur votre ordinateur, ils sont souvent dû à la modification de variable qui n'appartiennent pas au programme, donc l'OS "tue" ce programme qui essayait de manipuler des trucs qui ne lui appartiennent pas.

## Initialiser un tableau

Au départ, notre tableau était vide :

```
1 float notes[20]; //on créer un tableau dont le contenu est vide, on sait simplement q
```

Ce que l'on va faire, c'est **initialiser** notre tableau. On a la possibilité de remplir chaque case une par une ou bien utiliser une boucle qui remplira le tableau à notre place. Dans le premier cas, on peut mettre la valeur que l'on veut dans chaque case du tableau, tandis qu'avec la deuxième solution, on remplira les cases du tableau avec la même valeur, bien que l'on puisse le remplir avec des valeurs différentes mais c'est un peu plus compliqué. Dans notre exemple des notes, on part du principe que l'examen n'est pas passé, donc tout le monde à 0. 😊 Pour cela, on parcourt toutes les cases en leur mettant la valeur 0 :

```
1 char i=0; //une variable que l'on va incrémenter
2 float notes[20]; //notre tableau
3
4 void setup()
5 {
```

```

6 //boucle for qui remplira le tableau pour nous
7 for(i = 0; i < 20; i++)
8 {
9     notes[i] = 0; //chaque case du tableau vaudra 0
10 }
11 }

```

L'initialisation d'un tableau peut se faire directement lors de sa création, comme ceci :

```

1 float note[] = {0,0,0,0 /*, etc.*/ };

```

Ou bien même, comme cela :

```

1 float note[] = {};
2
3 void setup()
4 {
5     note[0] = 0;
6     note[1] = 0;
7     note[2] = 0;
8     note[3] = 0;
9     //...
10 }

```

## Exemple de traitement

Bon c'est bien beau tout ça, on a des notes coincées dans un tableau, on en fait quoi ? 🤔

Excellente question, et ça dépendra de l'usage que vous en aurez 😊 ! Voyons des cas d'utilisations pour notre tableau de notes (en utilisant des fonctions 😊).

### La note maximale

Comme le titre l'indique, on va rechercher la note maximale (le meilleur élève de la classe). La fonction recevra en paramètre le tableau de float, le nombre d'éléments dans ce tableau et renverra la meilleure note.

```

1 float meilleurNote(float tableau[], int nombreEleve)
2 {
3     int i = 0;
4     int max = 0; //variables contenant la future meilleure note
5
6     for(i=0; i<nombreEleve, i++)
7     {
8         if(tableau[i] > max) //si la note lue est meilleure que la meilleure actuelle
9         {
10             max = tableau[i]; //alors on l'enregistre
11         }
12     }
13     return max; //on retourne la meilleure note
14 }

```

Ce que l'on fait, pour lire un tableau, est exactement la même chose que lorsqu'on

l'initialise avec une boucle `for`.

Il est tout à fait possible de mettre la valeur de la case recherché dans une variable :

```
1 int valeur = tableau[5]; //on enregistre la valeur de la case 6 du tableau dans une v
```

Voilà, ce n'était pas si dur, vous pouvez faire pareil pour chercher la valeur minimale afin vous entraîner !

## Calcul de moyenne

Ici, on va chercher la moyenne des notes. La signature de la fonction sera exactement la même que celle de la fonction précédente, à la différence du nom ! Je vous laisse réfléchir, voici la signature de la fonction, le code est plus bas mais essayez de le trouver vous-même avant :

```
1 float moyenneNote(float tableau[], int nombreEleve)
```

### Une solution :

```
1 float moyenneNote(float tableau[], int nombreEleve)
2 {
3     int i = 0;
4     double total = 0; //addition de toutes les notes
5     float moyenne = 0; //moyenne des notes
6     for(i = 0; i < nombreEleve; i++)
7     {
8         total = total + tableau[i];
9     }
10    moyenne = total / nombreEleve;
11    return moyenne;
12 }
```

On en termine avec les tableaux, on verra peut être plus de choses en pratique. 😊

Maintenant vous pouvez pleurer, de joie bien sûr, car vous venez de terminer la première partie ! A présent, faisons place à la pratique...