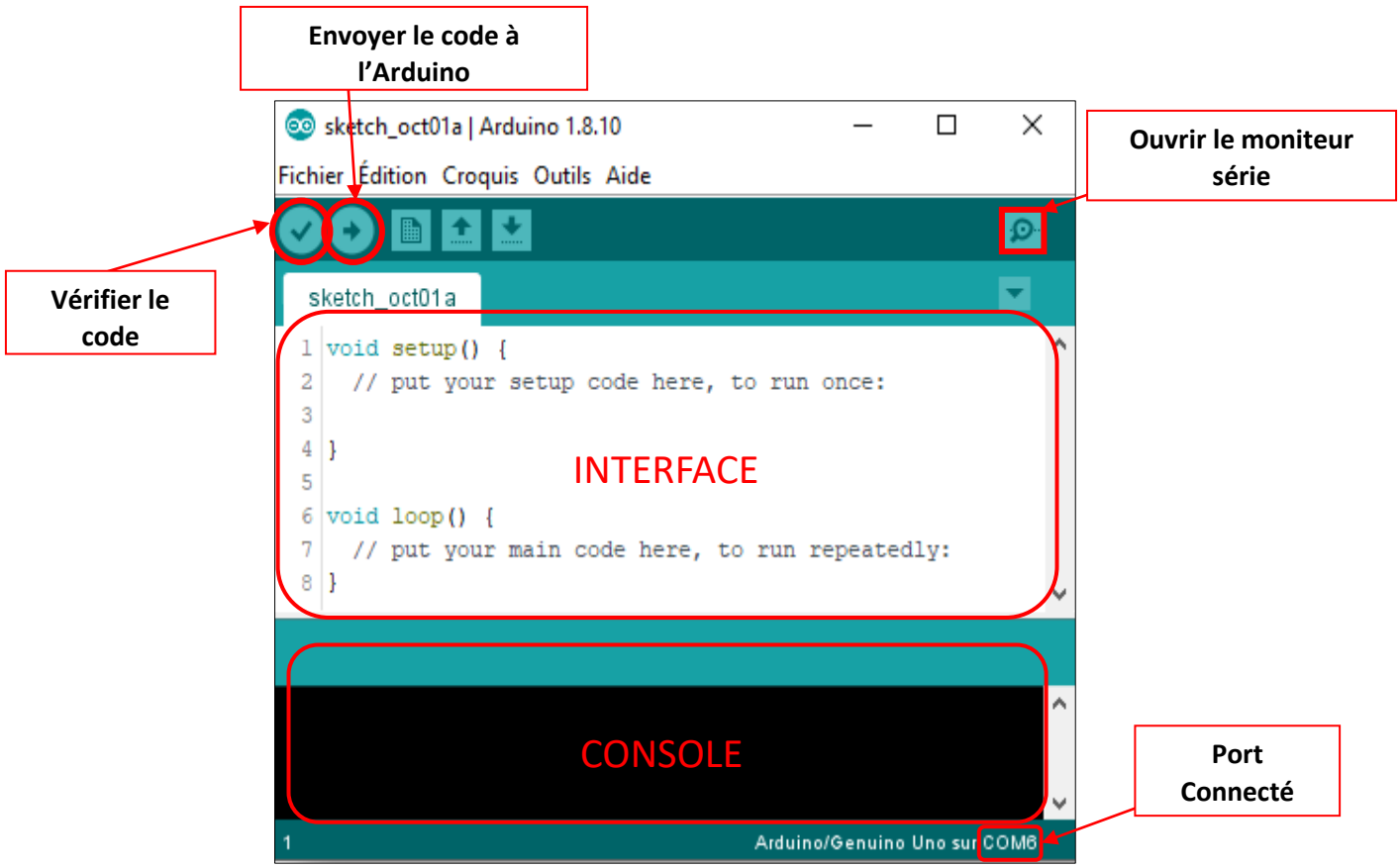
Étape 3a : Programmation - Interface

Objectifs:

- ❑ Comprendre le fonctionnement de l'interface Arduino;

Comprendre le fonctionnement de l'interface Arduino.



Interface : L'interface est l'endroit où l'on écrit toutes les instructions (le code) que l'Arduino devra exécuter.

Console : La console est votre meilleur ami. C'est l'endroit où les erreurs apparaissent lorsqu'on demande de vérifier le code. Il indique quel type d'erreur et à quelle ligne elle s'est produite, ce qui facilite grandement la correction du code.

Moniteur Série : Il est possible que l'on demande au code d'afficher du texte ou des nombres. Ces valeurs s'afficheront dans le Moniteur Séries.

Étape 3b : Programmation - Variables

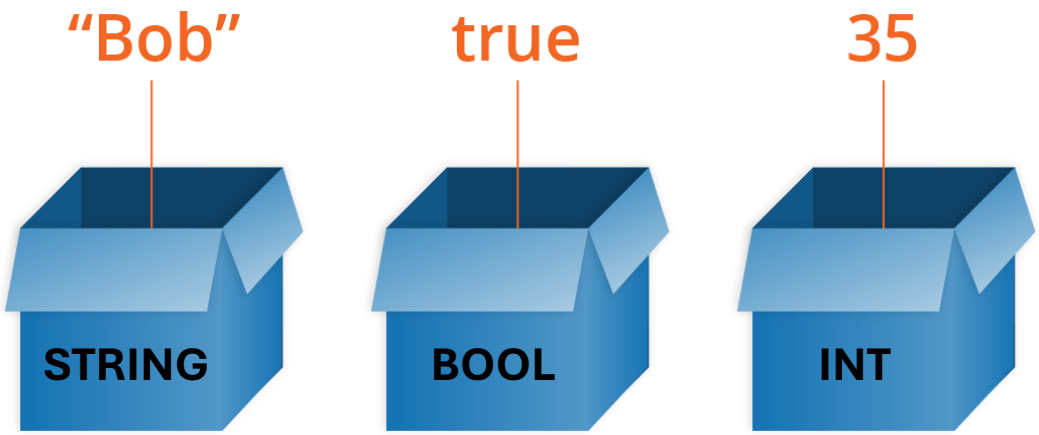
Objectifs:

- ❑ Comprendre le fonctionnement d'une variable;

Comprendre le fonctionnement d'une variable.

Une variable est une boîte dans laquelle on peut ranger des valeurs, textes, matrices, fonctions etc… Il en existe plusieurs types, voici celles que vous allez utiliser le plus souvent :

- **int** : Les variables int contiennent des nombres entiers. Ex : 0 , 1, 465, -34 etc…
- **float** : Les variables float contiennent des nombres à virgule. Ex : 0.0; 2.34; 435.65; 23.0; -32.3, etc…
- **string** : Les variables string contiennent des chaînes de caractères (des lettres, ponctuations, des chiffres etc…). Elles sont notées entre guillemets. Ex : "mot", "56" etc…
- **bool** : Les variables bool prennent deux valeurs : Vrai ou Faux
- **Tableau** : Les tableaux sont une liste de variables. Elles permettent de stocker plusieurs paramètres dans une seul variable. Ex : [6, "salut", true, 33, "bonjour"]



N'hésitez pas à utiliser l'ordinateur pour comprendre cette leçon

Afin d'utiliser une variable, il faut d'abord la déclarer (On dit au programme qu'il existe une boîte avec le nom de notre variable). Pour cela, au début de notre code, on écrit:

[Type de Variable] Nom_de_la_variable = Valeur_de_la_valeur_initiale;

Exemples

```
1. int C = 3;   // On crée une variable entière "C" à
   laquelle on associe la valeur 3
2. int F;       // On ne met rien dedans. La boîte est
   alors vide jusqu'à ce qu'on y mette quelque chose
3. float Nombre = 2.64;   // On crée une variable
   "Nombre" à laquelle on associe 2,64
4. bool verif = true;
```

? Pour mieux comprendre la déclaration d'une variable, déclarez une variable avec comme nom « Prénom » et comme valeur votre Prénom. Laquelle est correct entre « int N = "32"» et « int N = 32» ?

Après avoir déclaré la variable, on n'a plus besoin de rappeler son type lorsque l'on veut changer sa valeur, l'utiliser dans des opérations, l'utiliser dans des fonctions etc… Il suffit juste d'utiliser son nom.

Exemples

```
1. int C = 3;
2. F =3*C;
```

? Pour mieux comprendre l'opération de variables, déclarez deux variables que vous additionnerez ensuite dans une troisième variable.
? Que se passe-t-il quand on additionne 2 variable de type string?

Rappel: On peut vérifier la valeur de notre variable en utilisant la fonction Serial.println(Nom_Variable), puis en ouvrant le moniteur série.

Etape 3c : Programmation et variables

Objectif:

- ❑ Comprendre la structure d'un code Arduino
- ❑ Être capable d'utiliser les fonctions propres d'Arduino

La structure:

Arduino est un dérivé de la programmation en C+, elle est faite pour la robotique et donc possède une structure bien particulière. Elle se décompose en 2 parties principales et d'autres parties annexes.

Attention, une des particularités d'Arduino est qu'à chaque fin de ligne de code il faut mettre « ; ».
Il faut aussi faire attention au «{» lorsqu'une est ouverte, elle doit être refermée à un moment.

Les deux parties principales:

Le void setup:
C'est une partie du programme qui ne s'exécute qu'une seule fois

- Permet de définir nos variables en entré (INPUT), si c'est un capteur (exemple: le bouton) ou sinon en sortie (OUTPUT) (exemple: LED).
- Permet aussi de déclarer le moniteur série, qui permet de communiquer avec la carte.

Le void loop:
C'est ici que ce situe le corps du programme. Il tourne en boucle.

```
1
2
3
4   void setup() {
5     // put your setup code here, to run once:
6
7   }
8
9   void loop() {
10    // put your main code here, to run repeatedly
11
12  }
```

Parties secondaires:

Avant le void setup:
- On peut y définir les variables (voir fiches sur les variables).

Void fonction annexe:
C'est un sous-programme qui s'exécute une fois lorsque il est appelé (pour rendre l'ensemble plus lisible).

Les fonctions propres:
Arduino possède son propre langage, ce qui inclut une syntaxe particulière, et des fonctions de base très utile .

Dans le void setup:

pinMode():
Permet de dire si la broche est considérée comme une entrée ou une sortie.

Serial.begin(9600):
Permet de lancer le moniteur série.

EXAMPLE

```
1. const int LED = 5;     //Je dis que la LED est reliée à la broche 5
2.
3. void setup() {
4.   pinMode(LED, OUTPUT);    // Je déclare le LED en tant que sortie
5.   Serial.begin(9600);      // Je démarre le moniteur série
6. }
```

Dans le void loop:

digitalWrite:
Permet d'envoyer du courant dans une broche en sortie.
Même principe que précédemment pour 1 et 0.

EXAMPLE

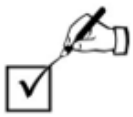
```
12   void loop() {
13   digitalWrite(LED,HIGH);        //allumer
14   digitalWrite(LED,LOW);         //éteindre
15   }
```

Les notations 0 ou LOW et 1 ou HIGH sont équivalentes, autrement dit pour la carte HIGH =1 et LOW =0

Delay(t):
Permet de mettre le programme en pause (le temps t est en ms)

Serial.println(...):
Permet d'écrire dans le moniteur série.

A toi de programmer! Tu peux essayer d'écrire sur le moniteur série « hello », ou essayer de faire clignoter une LED



Est-ce-que tu as bien compris ?

Expliquez précisément ce que fait ce programme.

```
1. const int LED_verte = 5;
2. const int LED_rouge = 6;
3.
4. void setup() {
5.   pinMode(LED_verte, OUTPUT);
6.   pinMode(LED_rouge, OUTPUT);
7.   Serial.begin(9600);
8. }
9.
10.void loop() {
11.  digitalWrite(LED_verte,HIGH);
12.  Serial.println("la LED verte est allumée")
13.  delay(1000)
14.  digitalWrite(LED_verte,LOW);
15.  digitalWrite(LED_rouge,HIGH);
16.  Serial.println("la LED verte est éteinte et
   la LED rouge est allumée");
17.  delay(1000);
18.}
```

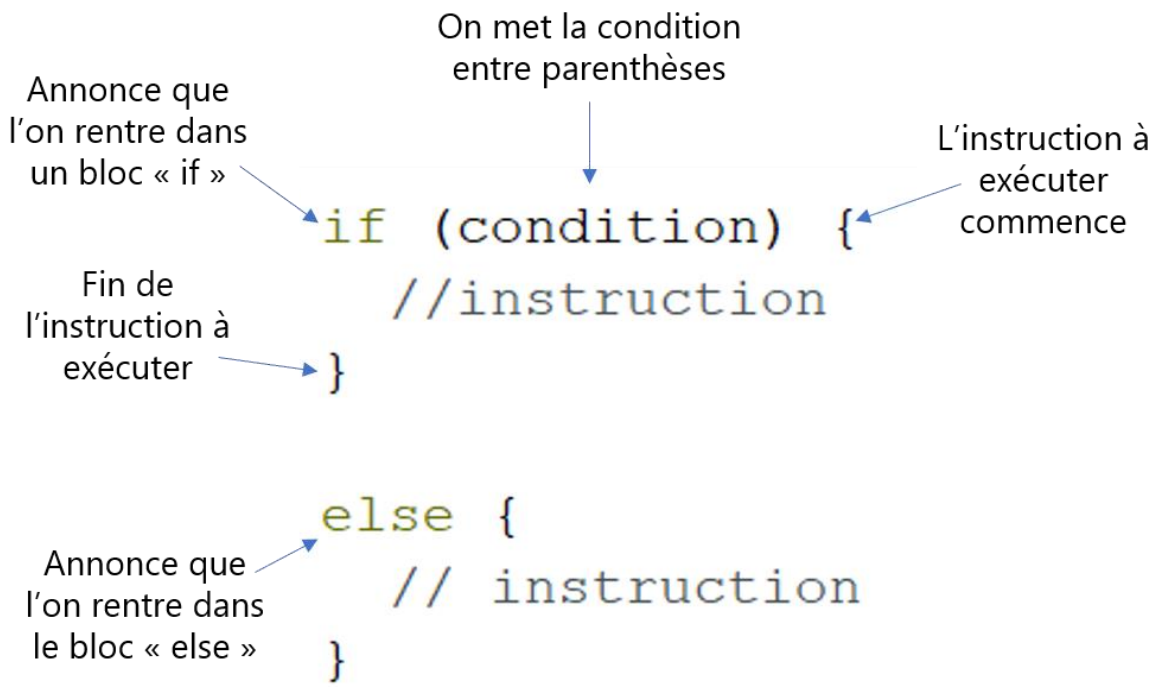
Étape 3d : Programmation - Blocs

Objectifs:

- ❑ Comprendre le fonctionnement d'un bloc **<< if/else >>;**
- ❑ Comprendre le fontionnement d'un bloc << **while >>;**
- ❑ Comprendre le fonctionemment d'un bloc << **for >>;**
- ❑ Comprendre les conditions et opérations logiques.

Comprendre le fonctionnement d'un bloc **<< if/else >>**

Un bloc conditionnel est un moyen d'exécuter des instructions seulement si une condition est remplie. Si la condition est vérifiée, l'instruction du bloc « if » est exécuté. Sinon, c'est celle du bloc « else ». Voilà comment faire :



Exemple simple:

```
if (il fait nuit){
  J'allume la lumière
}
else {
  J'éteins la lumière
}
```

Les **conditions** sont normalement liées aux états des **entrées**, telles que les boutons. Les **instructions** sont associées aux **sorties**, comme allumer des LED ou produire des sons avec un buzzer.

Comprendre le fonctionnement d'un bloc **<< while >>**

Un bloc "while" est également une structure conditionnelle, mais sa structure diffère légèrement. Il fonctionne comme une boucle, où les instructions à l'intérieur se répètent tant que la condition est vraie. Lorsque la condition n'est plus satisfaite, il sort du bloc "while" et poursuit l'exécution du code. Voilà comment faire :

```
while (condition) {
  // instruction
}
```

? Pour mieux comprendre le fonctionnement de la boucle while, remplace le bloc « if » ci-dessus par une boucle « while » pour allumer la lumière tant qu'il fait nuit.

Attention: pour écrire des conditions, on a besoin de comprendre les possibles tests qu'on peut faire avec nos variables.

Conditions

Pour les conditions, nous avons plusieurs options :

1- Égalité : Pour vérifier si une variable est égale à une valeur, nous utilisons l'opérateur "==" (variable == valeur). Fais attention à utiliser deux signes égaux pour l'opération d'égalité.
2 - Inégalité : Pour vérifier si une variable est différente d'une valeur, nous utilisons l'opérateur "!=" (variable != valeur).
3 - Comparaison : Pour effectuer des comparaisons, nous utilisons les opérateurs de comparaison tels que "<=" (inférieur ou égal à) et ">=" (supérieur ou égal à) (variable <= valeur et variable >= valeur).

```
1. int A = 3;
2. if (A==3) {
3.   Serial.println("A vaut 3")
4.   }
```

Exemple simple

Opérations Logiques

Il est également possible d'avoir plus d'une condition et d'utiliser des symboles pour identifier les opérations "ET" et "OU" :

1 - "ET" logique : Pour vérifier si deux conditions sont toutes les deux vraies, nous utilisons l'opérateur "&&" (par exemple : (condition1) && (condition2)). Cette condition sera vraie uniquement si les deux conditions sont satisfaites.
2 - "OU" logique : Pour vérifier si au moins l'une des deux conditions est vraie, nous utilisons l'opérateur "||" (par exemple : condition1 || condition2). Cette condition sera vraie si l'une des deux conditions est satisfaite.

```
1. int A = 3;
2. string B = "salut"
3. if (A==3 && B=="salut") {
4.   Serial.println("A vaut 3 et B vaut 'salut'")
5.   }
```

Exemple simple

Comprendre le fonctionnement d'un bloc << **for** >>

Le bloc "for" est une structure de programmation qui permet de répéter un ensemble d'instructions un certain nombre de fois.
Pour mieux comprendre: Imagine que tu as une boîte de bonbons et que tu veux en manger un chaque jour pendant une semaine. Tu pourrais utiliser une boucle "for" pour t'aider. Voici un exemple simple de boucle "for" pour manger un bonbon chaque jour pendant une semaine :


```
for (int jour = 1; jour <= 7; jour++) {
  mangerBonbon();
}
```

Dans cet exemple, le bloc « for » commence en créant une variable appelée "jour" et l'initialise à 1. Ensuite, il spécifie que tant que "jour" est inférieur ou égal à 7, les instructions à l'intérieur du bloc "for" sont répétées. Finalement, "jour++" signifie que, à chaque tour de boucle, on incrémente de 1 la valeur de jour. À chaque itération de la boucle, nous appelons la fonction "mangerBonbon()" pour manger un bonbon. La boucle continue jusqu'à ce que "jour" atteigne la valeur 8, puis elle se termine.

C'est un moyen pratique d'effectuer une action répétée un nombre spécifique de fois. Voici la structure en résumé:

```
for (initialization; condition; increment) {
  //instruction
}
```

Est-ce-que tu as bien compris ?

1) Expliquez pourquoi les codes ne sont pas corrects.

```
for (i=1; i++; i<4) {
  //instruction
}
```

```
if(state = 1){
  //instruction
}
```

```
while (state != 1):
  //instruction
```

2) Pourquoi dans la boucle **for** on écrit « variable = valeur » et dans le bloc **if** on écrit « variable == valeur » ?