Projet 4: introduction aux variables

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

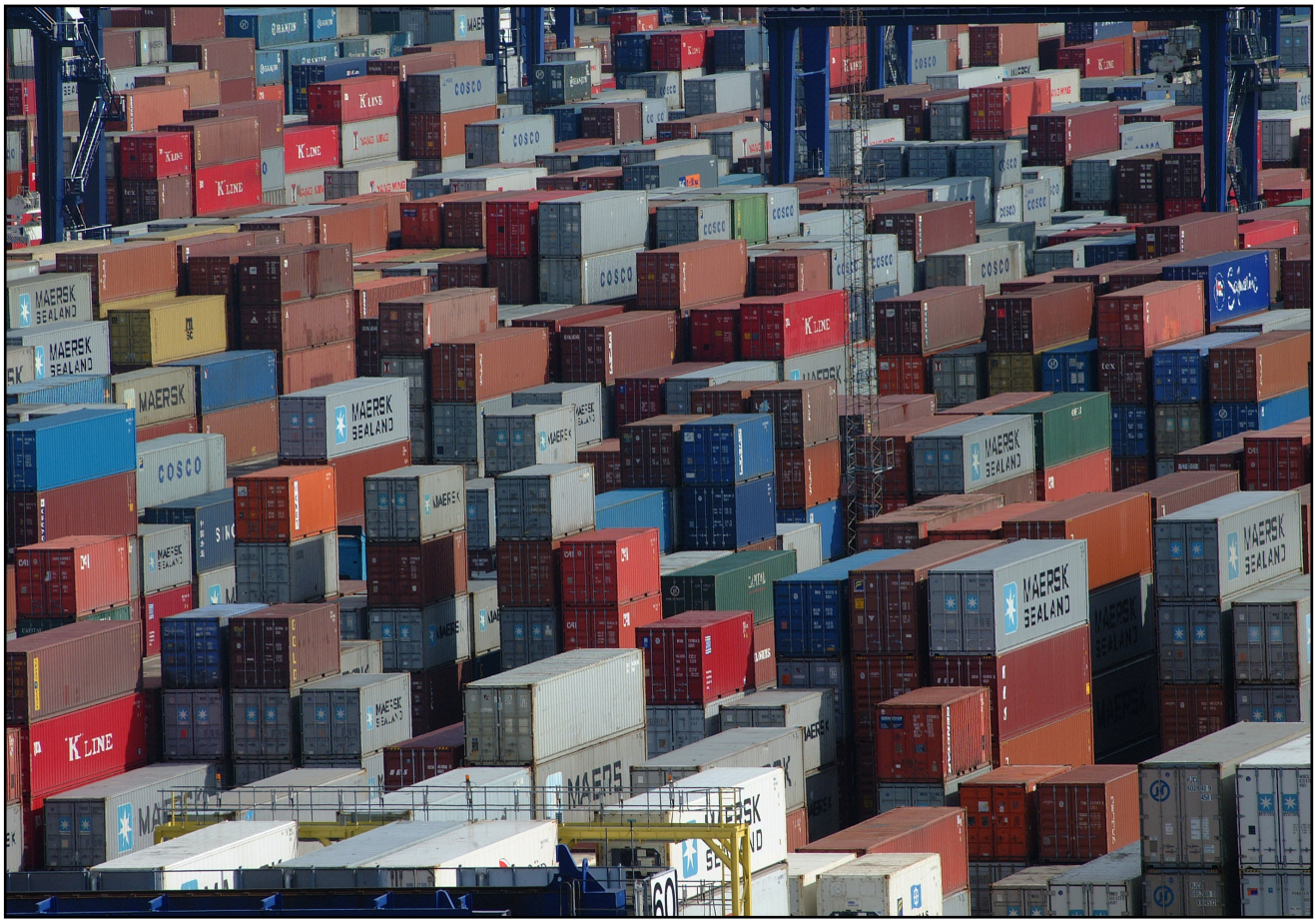
Imaginons un nombre dont nous devons nous souvenir. Ce nombre est stocké dans un espace de stockage de la mémoire vive (RAM) du microcontrôleur. Chaque espace de stockage est identifié de manière unique.

Le nombre stocké a la particularité de *changer de valeur*. En effet, la variable n'est que le conteneur. Son contenu va donc pouvoir être modifié. 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..."

Imaginons que nous stockons le nombre dans un container (la variable). Chaque container est lui, déposé dans un espace bien précis, afin de le retrouver. Chaque container (variable) est aussi identifié par un nom unique.



Le nom d'une variable

Le nom de variable n'accepte que l'alphabet alphanumérique ([a-z], [A-Z], [0-9]) et _ (underscore). Il est unique; il ne peut donc pas y avoir deux variables portant le même nom.

Définir une variable

Imaginons que nous voulons stocker le nombre 4 dans une variable. Il tiendrait dans un petit carton. Mais on pourrait le stocker dans un grand container. Oui... mais non! Un microcontrôleur, ce n'est pas un ordinateur 3GHz multicore, 8Go de RAM ! Ici, il s'agit d'un système qui fonctionne avec un CPU à 16MHz (soit 0,016 GHz) et 2 Ko de SRAM pour la mémoire vive. Il y a donc au moins deux raisons pour choisir ses variables de manière judicieuse :

1. La RAM n'est pas extensible, quand il y en a plus, y en a plus! Dans un même volume, on peut stocker bien plus de petits cartons de que gros containers. Il faut donc optimiser la place.
2. Le processeur est de type 8 bits (sur un 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 les variables de la taille d'un container sont sur 32 bits, autant prendre un carton qui n'occupe que 8 bits quand la variable tient dedans!

Type de variable	Type de nombre stocké	Valeurs maximales du nombre stocké	Nombre sur X bits	Nombre d'octets
int	entier	-32'768 à +32'767	16 bits	2 octets
long	entier	-2'147'483'648 à +2'147'483'647	32 bits	4 octets
char	entier	-128 à +127	8 bits	1 octet
float	décimale	-3.4×10^{38} à $+3.4 \times 10^{38}$	32 bits	4 octets
double	décimale	-3.4×10^{38} à $+3.4 \times 10^{38}$	32 bits	4 octets

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

```
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*.

```
int x = 0;
```

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 de variable	Type de nombre stocké	Valeurs maximales du nombre stocké	Nombre sur X bits	Nombre d'octets
unsigned char	entier non négatif	0 à 255	8 bits	1 octet
unsigned float	entier non négatif	0 à 65'535	16 bits	2 octets
unsigned double	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, listées dans ce tableau:

Type de variable	Type de nombre stocké	Valeurs maximales du nombre stocké	Nombre sur X bits	Nombre d'octets
byte	entier non négatif	0 à 255	8 bits	1 octet
word	entier non négatif	0 à 65'535	16 bits	2 octets
boolean	entier non négatif	0 à 1	1 bits	1 octet

.

Définir les broches du microcontrôleur

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

Premièrement, définissons la broche utilisée du microcontrôleur en tant que variable.

```
const int led1 = 13;
```

Le terme *const* signifie que l'on définit la variable comme étant constante. Par conséquent, on change la nature de la variable qui devient alors constante.
Le terme *int* correspond à un type de variable. Dans une variable de ce type, on peut stocker un nombre allant de -2147483648 à +2147483647, ce qui sera suffisant! Ainsi, la broche *13* s'appellera *led1*.

Nous sommes donc en présence d'une variable, nommée *led1*, qui est en fait une constante, qui peut prendre une valeur allant de -2147483648 à +2147483647. Dans notre cas, cette constante est assignée au nombre 13.
Concrètement, qu'est-ce que cela signifie? Observons la différence entre les deux codes.

Broche non-définie

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

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

Broche définie

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

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

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

On peut trouver que de définir les broches allonge le code. Mais quand nous aurons de nombreuses broches en fonction, cela nous permettra de les identifier plus facilement. Ainsi, si nous avons plusieurs LED, nous pouvons les appeler *Led1*, *Led2*, *Led3*,... et si nous utilisons des LED de plusieurs couleurs, nous pourrons les appeler *rouge*, *vert*, *bleu*,...
Enfin (et surtout!), si on veut changer la broche utilisée, il suffit de corriger la variable au départ, sans devoir corriger tout le code.
Comme pour les variables, nous pouvons donner n'importe quel nom aux broches.