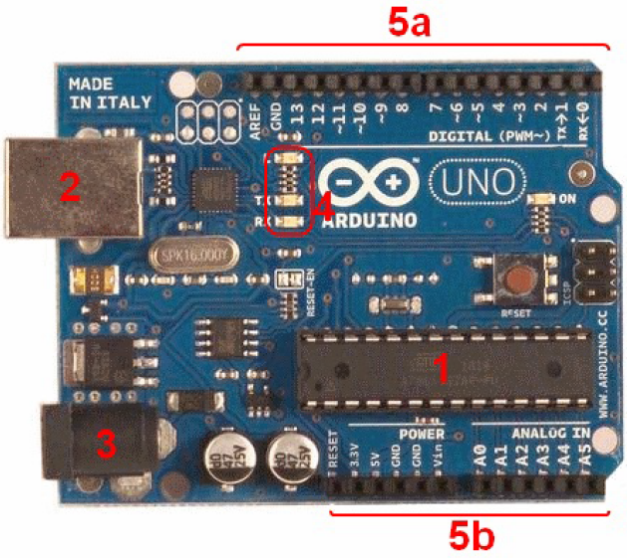
Séquentiel et Algorithmie
Carte ARDUINO - « SOS »

1. Présentation de la carte Arduino UNO :

1.1. La carte Arduino Uno :



1) Le cœur de la carte, un µcontrôleur Atmel ATMEGA328P.

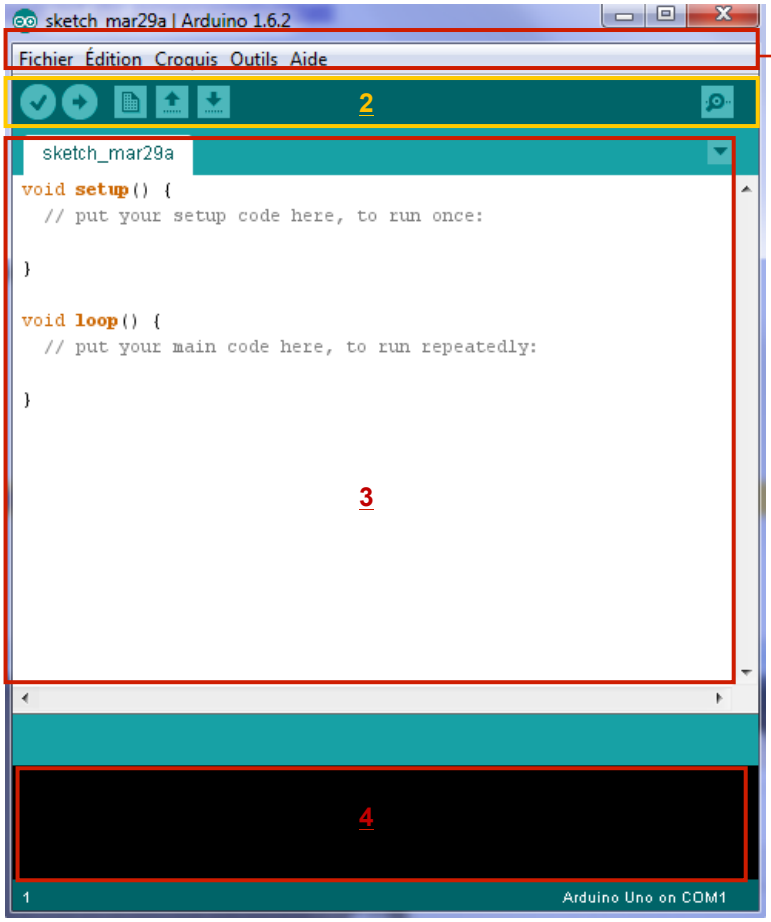
2) Le port USB, qui permet entre autre d'alimenter la carte en +5V.

3) Prise d'alimentation externe. L'alimentation doit être de type continue et avec une valeur comprise entre +7V et +15V. Il est parfois nécessaire d'utiliser cette alimentation lorsque l'on connecte une charge "importante" ou si l'on souhaite l'autonomie de la carte arduino.

4) DELS de visualisation. L'une (orange) permet de tester la carte, les 2 autres (vertes) permettent de visualiser la liaison série en émission Tx et en réception Rx.

5) Connecteur permettant une liaison filaire avec des composants extérieurs (capteurs, boutons poussoir, Dels, transistor...)

1.2. Le logiciel de programmation :



1 L'ensemble de ces onglets permettent principalement de choisir ou de définir les options de configuration du logiciel.

2 Les icônes permettent de contrôler les fichiers programmes, de télécharger les programmes.

3 La partie ou l'on écrit le programme.

4 L'ecran du débugger (indication d'erreurs de compilation)

2. Premier programme :

2.1. Le code minimal :

```
/*
Commentaire sur plusieurs lignes, permettant généralement de décrire
la fonctionalité du programme, les auteurs, la carte, la date...
 */

void setup()      //fonction d'initialisation de la carte
{
                  // contenu de l'initialisation

}
void loop()       // fonction principale qui s'exécute à l'infini
{
                  // contenu du programme
}
```

Les commentaires :

- En grisé,
- doivent être contenus par **/*commentaire*/** s'ils s'écrivent sur plusieurs lignes.
- ou après **//** **commentaire** s'ils peuvent tenir sur une seule ligne
- Les commentaires sont essentiels pour une bonne compréhension du programme.

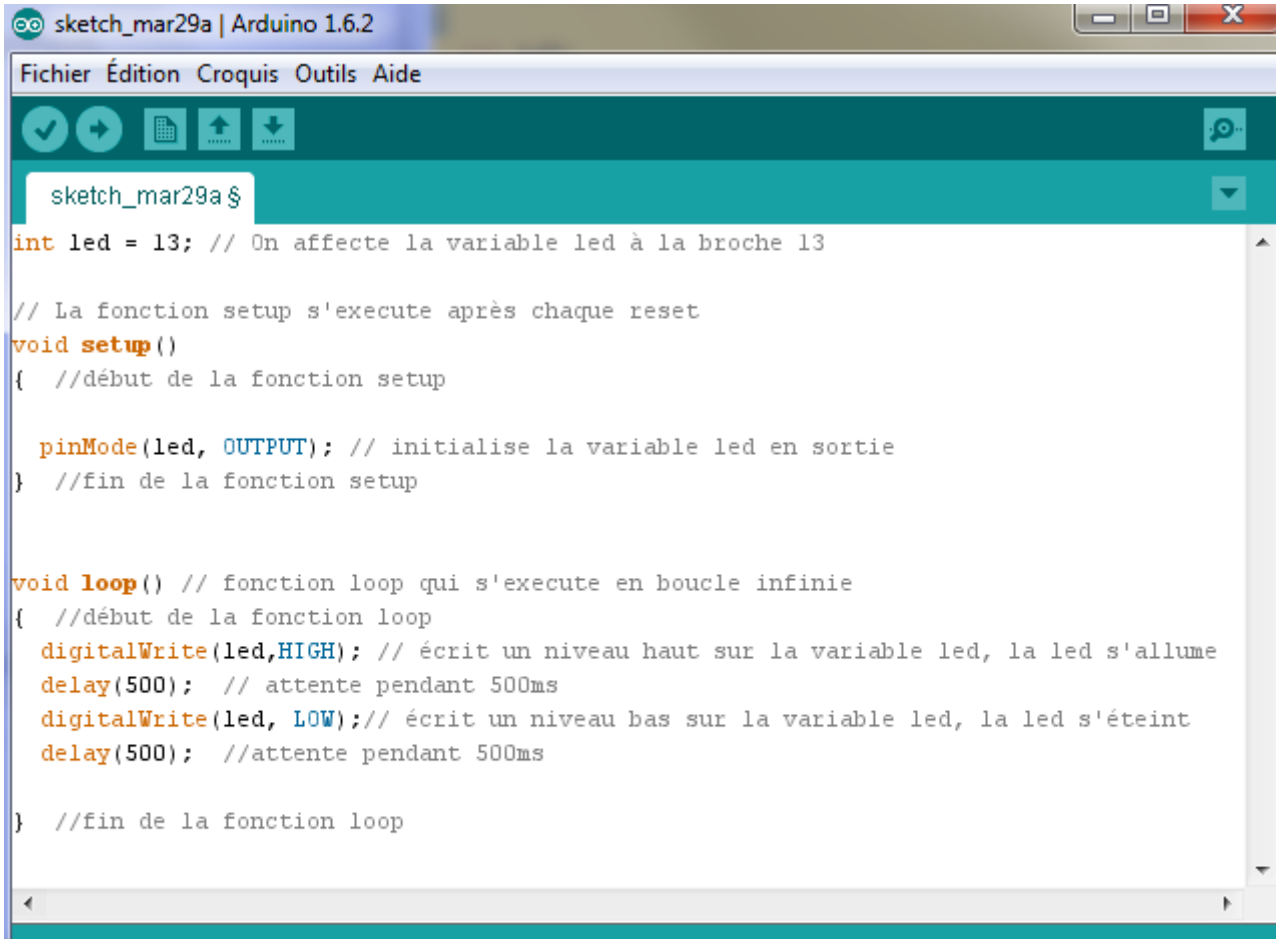
2.1.1. La fonction setup() :

Cette fonction setup() est appelée une seule fois lorsque le programme commence. On y écrit le code qui n'a besoin d'être exécuté qu'une seule fois. On appelle cette fonction : "fonction d'initialisation". On y retrouvera la configuration des principales fonctions que l'on souhaite utiliser, la configuration des broches en entrée ou en sortie...

2.1.2. La fonction loop() :

C'est dans cette fonction loop() que l'on écrit le contenu du programme. Cette fonction s'exécute en boucle infinie, lorsque la dernière ligne de programme est exécutée, le programme reprend de la première ligne.

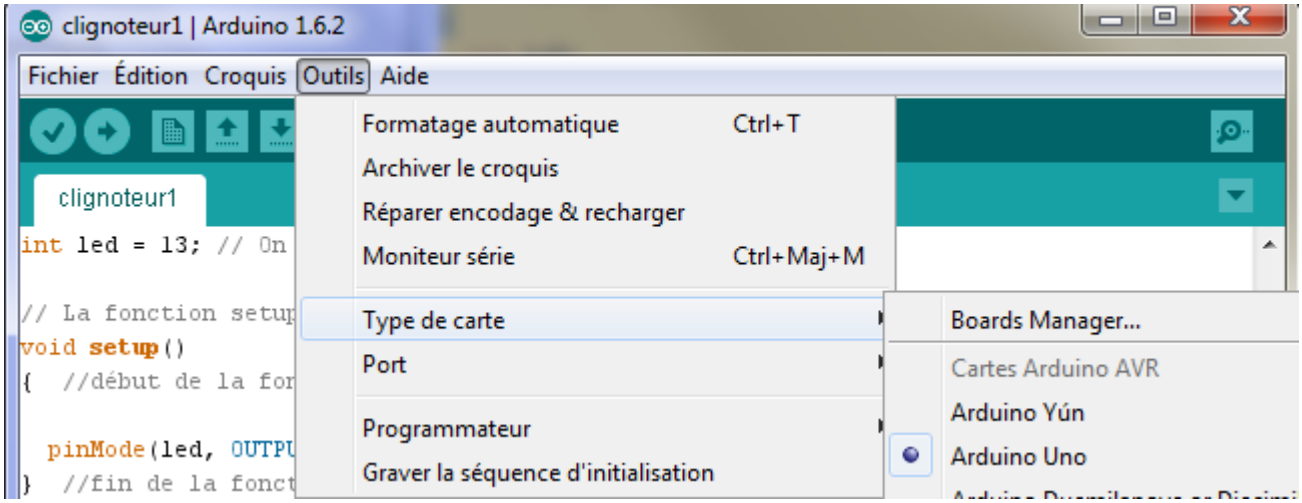
2.2. Exemple :



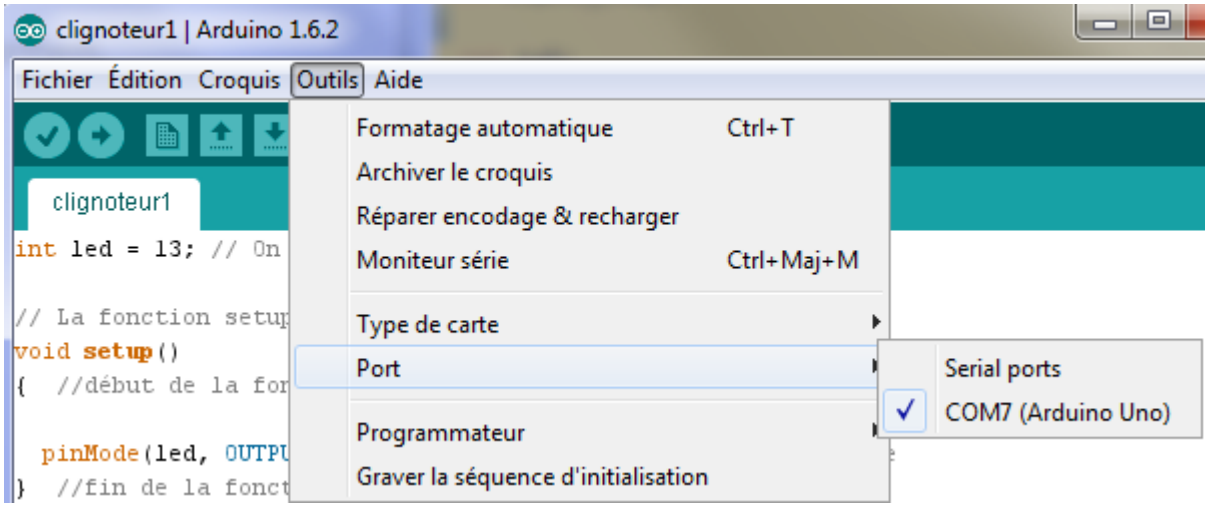
Q1. Ecrire le programme ci-dessus.

Q2. Transférer  *le programme vers la carte Arduino Uno.*

Dans le cas ou un message d'erreur apparaît on vérifiera :



et :



La valeur de COM peut naturellement différée de COM7

Q3. Décrire l'action du programme.

3. Génération d'un signal S.O.S :

3.1. Cahier des charges :

Nous allons créer pour l'Arduino un programme permettant de générer un signal **MORSE lumineux d'un SOS**.

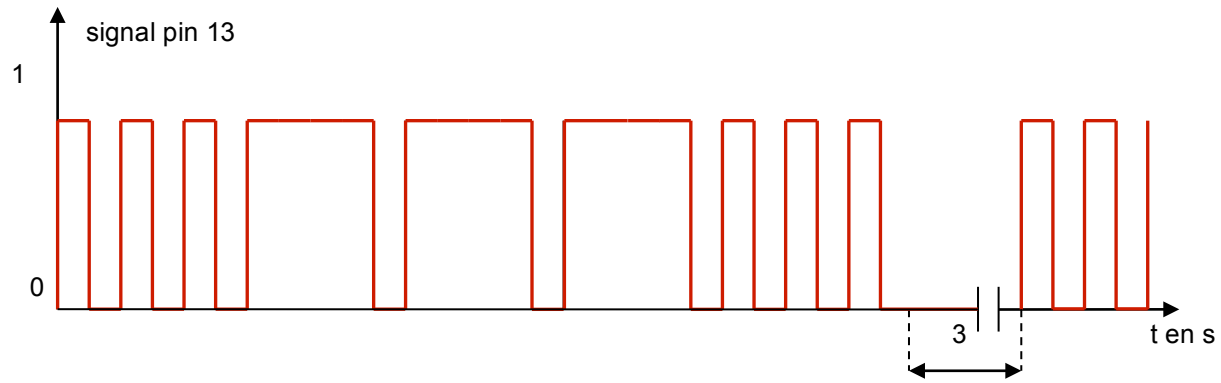
Rappelons que le SOS correspond à 3 signaux courts suivi de 3 longs puis à nouveau de 3 courts.

Nous choisissons donc une durée :

- d'allumage et d'extinction de 250ms pour les signaux courts,
- d'allumage de 1000ms et d'extinction de 250ms pour les signaux longs.

De plus :
• Les appels de détresse seront espacés de 3 secondes.
• La DEL sera affectée à la broche 13 de la carte (la broche 13 comporte une DEL).

Chronogramme du signal sur la broche 13 :



3.2. Ecriture du programme :

3.2.1. Ecrire un programme très "mauvais".

*Q4. En reprenant la structure du clignoteur précédent, écrire le programme en utilisant la fonction **delay**, et faites valider son fonctionnement.*

3.2.2. Ecrire un programme amélioré par rapport à la version précédente :

Afin d'améliorer le programme, il est possible ***d'insérer des boucles*** qui nous permettrons de compter 3 allumages courts, puis 3 allumages longs, puis à nouveau 3 allumages courts. Pour cela nous devons créer une variable de comptage noter "i".

Les différents types de variables susceptibles d'être utilisées sont les suivantes :

Type de variable	Type de nombre	Valeur du nombre	Nombre de bits	Nombre d'octet
int	Entier	De -32768 à + 32767	16	2
long	Entier	De - 2147483648 à - 2147483647	32	4
char	Entier	De -128 à 127	8	1
float	Décimal	De $-3{,}4028235\times 10^{+38}$ à $3.4028235\times 10^{+38}$	32	4
double	Décimal	De $-3{,}4028235\times 10^{+38}$ à $3.4028235\times 10^{+38}$	32 (sur uno)	4
unsigned char	Entier non négatif	De 0 à 255	8	1
unsigned long	Entier non négatif	0 à 4 294 967 295	32	4
unsigned int	Entier non négatif	0 à 65535	16	2
byte	Entier non négatif	0 à 255	8	1
word	Entier non négatif	0 à 65535	16	2
boolean	Entier non négatif	0 à 1	1	1

Q5. Choisir le type de la variable "i" la plus appropriée à notre application (on rappelle que i ne dépassera pas la valeur 3)

La variable "i" doit permettre de réaliser une boucle du type :

Algorigramme	*Syntaxe ARDUINO (et langage C)*
TANT QUE i < 3 FAIRE "clignotement rapide"...	*While (condition sur "i") { instructions;}*

Le tableau ci-dessous liste les opérateurs communs :

Opérateur	Dénomination	effet	Syntaxe
==	égalité	si variable1 = variable2 résultat vrai sinon résultat faux	**((variable1)==(variable2))**
!=	différent de	si variable 1 ≠ variable 2 résultat vrai sinon résultat faux	**((variable1)!=(variable2))**
<	plus petit que	si variable 1 < variable 2 résultat vrai sinon résultat faux	**((variable1)<(variable2))**
<=	plus petit ou égal	si variable 1 ≤ variable 2 résultat vrai sinon résultat faux	**((variable1)<=(variable2))**
>	Plus grand que	si variable 1 > variable 2 résultat vrai sinon résultat faux	**((variable1)>(variable2))**
>=	Plus grand que ou égal	si variable 1 ≥ variable 2 résultat vrai sinon résultat faux	**((variable1)>=(variable2))**

Pour incrémenter la variable i il suffit d'écrire l'instruction :

`i=i+1;` **ou :** `i++;`

Q6. Modifier le programme en insérant les boucles de type While pour les différents types de clignotement de la LED.

Q7. Faire valider votre programme.

3.3. Utilisation de fonctions :

Notre programme, même amélioré n'est pas encore un programme propre au sens de la programmation. Pour qu'il le devienne, il faut réaliser des fonctions (des sous programmes) réutilisables par le programme principal.

Il existe deux types de fonctions, ***les fonctions de type "void"*** qui ne renvoient pas de valeur au programme principal après leurs exécutions et ***les fonctions "typées"*** qui elles, retournent une valeur vers le programme principal après leurs exécutions.

Dans notre application, nous n'utiliserons que des fonctions de type "**void**".

3.3.1. Création de la fonction "S" :

Q8. Juste après la fonction setup(), créer la fonction "S" telle que:

```
void S( ) {
  byte i=0;
  while (i<3){
    digitalWrite(led,HIGH); // écrit un niveau haut sur la variable LED, la LED s'allume
    delay(250); // attente pendant 250ms
    digitalWrite(led, LOW); // écrit un niveau bas sur la variable led, la led s'éteint
    delay(250); //attente pendant 250ms
    i = i+1;
  }
}
```

3.3.2. Création de la fonction "O".

Q9. De la même manière que la fonction "S", créer la fonction "O"

3.3.3. Création de la fonction "DETRESSE".

Q10. Créer la fonction "DETRESSE" de telle manière que le programme principal ne contienne que les lignes suivantes :

```
void loop( ) {
  DETRESSE( );
  delay(3000);
}
```

Votre programme principal ne contient plus que 2 lignes, c'est une programmation propre.
Félicitations!!!!!

3.4. Création et analyse d'une fonction typée :

Q11. Créer le code ci-dessous en utilisant au maximum le code précédemment écrit.

Remarque : le nombre issu de l'exécution de la fonction soustraction est un entier.

```
int led = 13; // On affecte la variable led à la broche 13

// La fonction setup s'execute après chaque reset

void setup()
{
  pinMode(led, OUTPUT); // initialise la variable led en sortie
}

void S()
{
byte i=0;
  while (i<3)
  {
    digitalWrite(led,HIGH); // écrit un niveau haut sur la variable led, la led s'allume
    delay(250);  // attente pendant 250ms
    digitalWrite(led, LOW);// écrit un niveau bas sur la variable led, la led s'éteint
    delay(250);  //attente pendant 250ms
    i = i+1;
  }
}
void O()
{
  byte i=0;
  while (i<3)
  {
    digitalWrite(led,HIGH); // écrit un niveau haut sur la variable led, la led s'allume
    delay(1000);  // attente pendant 500ms
    digitalWrite(led, LOW);// écrit un niveau bas sur la variable led, la led s'éteint
    delay(250);  //attente pendant 500ms
    i = i+1;
  }
}

int soustraction (int nombre1,int nombre2)
{
  int total=0;
  total=nombre1-nombre2;
  return total;
}
```

```
int x = 10;
int y = 3;
int resultat = 0;

void loop()
{
  resultat = soustraction(x,y);
  if (resultat >0) { S() ; delay (2000);}
  else if (resultat < 0) { O() ; delay (2000);}
  else {digitalWrite (led,HIGH);delay (3000);digitalWrite (led,LOW);delay(1000);}
  y=y+1;
}
```

Q12. La fonction typée "soustraction" renvoie une valeur, De quel type est-elle ?

Q13. Que se passe t-il tant que x > y ?

Q14. Que se passe t-il lorsque x = y ?

Q15. Que se passe t-il tant que x < y ?

Q16. Représenter le programme principal sous forme d'algorigramme.

Q17. Inverser x et y afin d'obtenir soustraction(y,x) à la place de soustraction(x,y). Compiler et tester. Que constater vous par rapport au fonctionnement initial ?

Q18. Remplacer nombre1 par x et nombre2 par y, le programme fonctionne t-il toujours ? Les variables x et y sont elles globales, ou locales, c'est à dire utilisable partout ou seulement dans le programme principal?