Mémento langage Arduino C++

```
1  // Commentaire sur une ligne
2
3  void setup() {
4  // Initialisation : instructions exécutées une seule fois
5  Instruction1;
6  Instruction2;
7  }
8
9  void loop() {
10 // Instructions répétées en boucle infinie
11 Instruction3;
12 }
```

void : en C++, le mot-clé **void** indique qu'une fonction ne renvoie pas de valeur (cf. autres exemples de fonctions page 11).

Les **délimiteurs** sont les **accolades** (analogue à l'indentation en python).
⚠ Erreur : Compilation error: expected '}' at end of input (par exemple).

Chaque instruction doit se terminer par un point-virgule (en python, uniquement pour séparer 2 instructions sur une même ligne).
⚠ Erreur : Compilation error: expected ';' before ….

Commentaires sur une seule ligne : **//** (# en python) ; multilignes **/* … */** (""" … """ en python).

Noms de variables

Caractères utilisables : _, 0, 1, 2, ..., 9, A, B, ..., Z, a, b, ..., z (commence toujours par une lettre ou un tiret bas « _ »).

Types

Entiers :	**short**	(2 octets $-2^{-15} \le n \le 2^{15}-1$, 16 bits)	$2^{15} = 32\,768$
	unsigned short	(2 octets $0 \le n \le 2^{16}-1$)	
	char	(1 octet $-2^{-7} \le n \le 2^{7}-1$, 8 bits) ('A' et 65 sont équivalents, codage ASCII)	
	byte	(1 octet $0 \le n \le 2^{8}-1$)	
	int, unsigned int	dépend du microcontrôleur (16 bits sur Arduino Uno)	
	long	(4 octets $-2^{-31} \le n \le 2^{32}-1$, 32 bits)	
	unsigned long	(4 octets $0 \le n \le 2^{32}-1$)	
	bool ou **boolean**	deux valeurs **true** ou **false** (sans majuscule au contraire de python)	
Flottants :	float	(9,4039548.10^{-38} à 3,4028235.10^{+38}, 32 bits)	Ex : 1.5e-2
Chaînes :	https://docs.arduino.cc/language-reference/en/variables/data-types/string/		

Affectation

Comme en python, le symbole d'affectation est le signe égal : « **=** ».

Déclaration des variables et types

Dans le langage C le type et les variables doivent être déclarés.
⚠ Erreur : *'nom_variable'* was not declared in this scope.

```
int i;                  // Déclaration sans affectation (l'affectation aura lieu plus loin dans le programme)
float pi = 3.14;        // Déclaration et affectation simultanées
char c = 'a';
```

Opérateurs

+, −, *, /, % (reste division entière)
== (test d'égalité), != (test de différence), <, >, <=, >= (tests de comparaison)
! (négation), || (ou), && (et)

Raccourcis

```
i=i+1;    ⇔    i++;        i=i-1;    ⇔    i--;
a=a+b;    ⇔    a+=b;       a=a-b;    ⇔    a-=b;
```

Instruction conditionnelle

```
if (conditions) {instructions}
else if (conditions) {instructions}          (instruction else if facultative, peut être répétée)
else (conditions) {instructions}             (instruction else facultative)
```

Rq : il existe une instruction conditionnelle multiple **switch…case**

Boucles

```
while (conditions) {instructions}
for (i=0;i<100;i=i+1) {instructions}         (par exemple)
```

Rq : il existe une boucle **do…while**

Tableaux

La taille des tableaux doit être une constante.

```
type nom_tableau[dimension]
```

Exemples :

```
int liste[10];
float L[3] = {1.1, 2.2, 3.3};
```

Accès via l'indice/index (la numérotation commence à 0 comme en python) : `nom_tableau[indice]`

Directives de compilation

Une directive de compilation indique au compilateur de procéder à des opérations préalables au début de la compilation.
Ces directives se situent en tout début du programme source.

```
#include <fichier>      // Inclure des librairies (analogue à import en python)
#include "fichier.h"    // Autre syntaxe
#define alias valeur    // Remplace alias par valeur
```

Exemple : `#define capteur1 A0` analogue à `const int capteur1 = A0` mais gain de mémoire avec define.

Const

Ce mot clé permet de définir une variable qui, une fois initialisée, ne pourra plus être modifiée.
Exemple :

```
Const float pi = 3.1415 ;
x = 2 * pi ;
```

Aide en ligne	https://docs.arduino.cc/language-reference/
Fonctions	https://docs.arduino.cc/language-reference/#functions
Variables	https://docs.arduino.cc/language-reference/#variables
Structure	https://docs.arduino.cc/language-reference/#structure

Transfert des données - Annexes techniques

Port série – Cf. « Découvrir en expérimentant » et tutoriels détaillés.

Un ***port série***, également appelé port COM, est un type d'interface informatique qui permet la communication entre un ordinateur et des périphériques externes. Il s'agit d'un port physique sur un ordinateur ou un appareil qui permet d'envoyer et de recevoir des données bit par bit et séquentiellement sur un seul fil.

Ouverture et écriture dans le port série

`Serial.begin(v)`	Ouvrir le port série et fixer la vitesse *v* de transmission (valeurs prédéfinies, cf. exemple 1).
`Serial.print(s)`	Ecrire la chaîne *s* sur la ligne courante (à la suite de la dernière chaîne écrite si elle existe) ou sur une nouvelle ligne sinon sans retour à la ligne.
`Serial.println(s)`	Ecrire la chaîne *s* sur la ligne courante (à la suite de la dernière chaîne écrite si elle existe) ou sur une nouvelle ligne sinon puis retour à la ligne.

Time

Il est parfois nécessaire d'indiquer au microcontrôleur un délai d'attente (entre une mesure et son traitement par exemple).

`delay(n)`	Attendre *n* ms (millisecondes).
`millis()`	Renvoie le nombre de millisecondes écoulées depuis que le sketch a été téléversé et exécuté.

Fichiers et caractères de codage « invisibles » – « Découvrir en expérimentant » et tutoriels détaillés.

Les fichiers comportent des caractères « invisibles » (on peut les visualiser dans un éditeur de texte tel que Notepad++).
Il s'agit par exemple des caractères qui provoquent un retour à la ligne, en python : \n (système Linux), \r\n (Windows), \r (Mac).
La bibliothèque python ***serial*** (https://pyserial.readthedocs.io/) permet de lire les lignes écrites dans le port série :
ligne = serial.Serial(*port_serie*, *vitesse*). readline() (lecture et stockage du résultat dans une variable nommée *ligne*).
Les données lues dans le port série sont au ***format binaire***, *ligne*.decode("utf-8") permet de décoder ce format.

Ces informations sont utiles lorsqu'il s'agit de sauvegarder des données lues sur le port série dans un fichier texte.