Tle

Spécialité

Sciences de l'Ingénieur par la pratique

- Résumé de cours
- Travaux pratiques
- Exercices corrigés

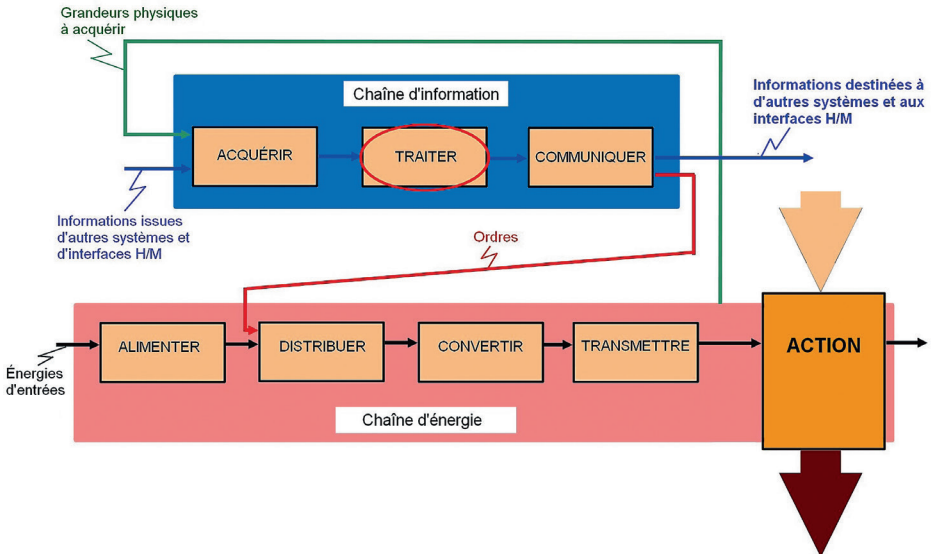
ellipses

La programmation

Sommaire

La programmation 10
Arduino (C++) 10
Python 12
Les différentes structures 13
TP 1 : Niveau de charge d'une batterie 18
Exercices 23
Corrigés des exercices 31
TP 2 : Serre autonome 38

Il est possible de considérer ce chapitre comme une suite du chapitre **Traiter l'information** du livre sur le programme de première. Cette partie a pour but de vous familiariser avec les langages de programmation comme le pseudo-code, python ou encore le C++.



La programmation

Arduino (C++)

L'environnement Arduino est composé de deux fonctions clés. Ces fonctions sont utilisées pour configurer et exécuter un programme sur le microcontrôleur de l'Arduino. Les fonctions sont « `void setup()` » et « `void loop()` ».

La fonction « `void setup()` » est appelée au début du programme pour initialiser les variables et les broches d'entrée/sortie. Cette fonction permet la configuration du microcontrôleur. Dans cette fonction, il est possible de déclarer les broches utilisées en entrée ou sortie, d'initialiser les bibliothèques, de définir la vitesse de transmission…

La fonction « `void loop()` » est la fonction principale qui s'exécute en boucle une fois que la fonction « `void setup()` » a été exécutée. Cette fonction contient le code qui sera exécuté en continu.

Pour programmer en Arduino, il est nécessaire de connaître certaines fonctions, variables, opérateurs, etc. :

Entrée/Sortie Numériques	
`digitalRead()`	Permet de lire la valeur d'une broche numérique
`digitalWrite()`	Permet d'écrire une valeur sur une broche numérique

`pinMode()`	Configure une broche en tant qu'entrée ou sortie
Entrée/Sortie Analogiques	
`analogRead()`	Permet de lire la valeur d'une broche analogique
`analogWrite()`	Permet d'écrire une valeur analogique sur une broche
`analogReference()`	Configure une tension de référence pour une entrée analogique
Communication série	
`Serial.begin()`	Définit le débit de données en bit par seconde (baud)
`Serial.print()`	Affiche les données
`Serial.println()`	Affiche les données sur une nouvelle ligne
Variables	
`int`	Entier
`float`	Nombre à virgule sur 32 bits
`double`	Nombre à virgule sur 64 bits
`bool`	Booléen (vrai ou faux)
`char`	Caractère unique
`long`	Entier sur 32 bits
`short`	Entier sur 16 bits
`string`	Chaîne de caractères
`array`	Collection fixe d'éléments du même type
`vector`	Collection dynamique d'éléments (similaire à une liste en Python)
Opérateurs	
=	Affectation à une variable
+	Addition
–	Soustraction
*	Multiplication
/	Division
%	Modulo (reste d'une division)
==	Égal
!=	Différent
<	Inférieur
<=	Inférieur ou égal
>	Supérieur
>=	Supérieur ou égal
&	ET
|	OU

^	OU-Exclusif
~	NON
++	Incrémentation
–-	Décrémentation
+=	Addition composée
&=	ET composé

Python

La structure d'un programme Python est un peu différente du C++. En Python, les limites des instructions et des blocs sont définies par la mise en page.

Le point-virgule (;) et les accolades ({}) utilisés en C++ sont remplacés par les niveaux d'indentations et par (:) pour les conditions.

Le langage Python oblige donc à écrire du code lisible et à prendre de bonnes habitudes. Ce qui en fait un langage idéal pour apprendre la programmation.

Comme pour Arduino, en Python, il est nécessaire de connaître certains éléments de programmation :

Entrée/Sortie	
`print(a)`	Affiche la valeur de a
`print("a")`	Affiche la lettre « a »
`print("La valeur de a est :", a)`	Affiche le texte et la valeur
`nom = input("Quel est votre nom ?")`	Demander du texte
`nombre = int(input("Quel est votre nombre ?"))`	Demander un nombre entier
`nombre = float(input("Quel est votre nombre ?"))`	Demander un nombre à virgule
Variables	
`int`	Entier
`float`	Nombre à virgule
`bool`	Booléen (vrai ou faux)
`str`	Chaîne de caractères
`list`	Collection ordonnée d'éléments pouvant être de types différents
`tuple`	Collection ordonnée d'éléments immuables
`dict`	Collection d'éléments sous forme de paires clé-valeur
`set`	Collection non ordonnée d'éléments uniques
`None`	Représente l'absence de valeur

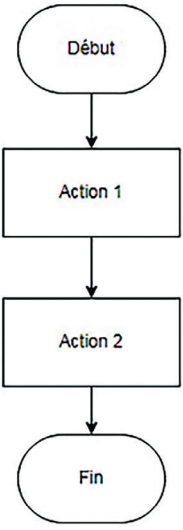
Opérateurs	
//	Division entière
**	Puissance
+=	Incrémentation (++ n'existe pas en Python)

Comme vous avez pu le constater en Python, les types de données sont plus généralement inclusifs et dynamiques, ce qui signifie que vous pouvez stocker différents types de données dans une même structure de données (comme une liste). En C++, les types de données sont souvent plus explicites et statiquement typés, nécessitant des déclarations de types claires lors de la déclaration des variables.

Les opérateurs quand à eux sont très similaires en Python et C++, seuls quelques-uns sont différents.

Les différentes structures

Structure linéaire



Pseudo-code

```
DÉBUT
    Action 1
    Action 2
FIN
```

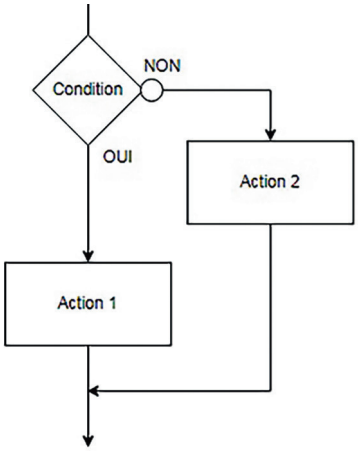
Python

```
Action 1
Action 2
```

Arduino (C++)

```
{
Action 1 ;
Action 2 ;
}
```

Structure alternative



Pseudo-code

```
SI Condition
    ALORS Action 1
SINON Action 2
FIN SI
```

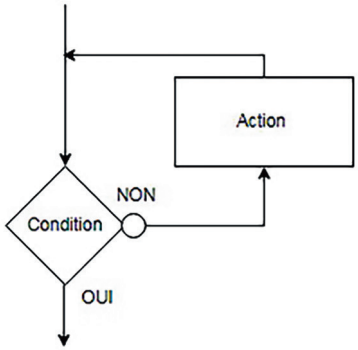
Python

```
if Condition :
    Action 1
else :
    Action 2
```

Arduino (C++)

```
if (Condition) {
    Action 1 ;
}
else {
    Action 2 ;
}
```

Structure itérative (Tant que)



Pseudo-code

```
TANT QUE Condition vraie
    FAIRE Action
FIN TANT QUE
```

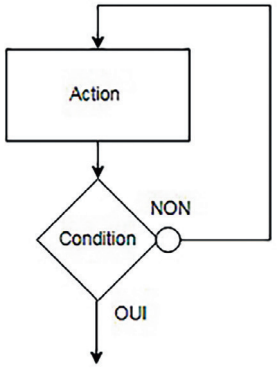
Python

```
while Condition vraie :
    Action
```

Arduino (C++)

```
while (Condition vraie){
    Action ;
}
```

Structure itérative (Faire tant que ou Répéter jusqu'à)



Pseudo-code

```
Action
FAIRE Action
TANT QUE Condition vraie
```

Python

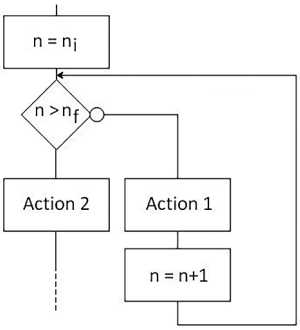
Cette structure n'existe pas directement en python.

```
while True :
    Action
if not Condition :
    break
```

Arduino (C++)

```
do{
    Action ;
}
while (Condition vraie) ;
```

Structure itérative (Pour)



n : variable ; n_i : valeur début
et n_f : valeur finale

Pseudo-code

```
POUR n ALLANT DE ni À nf NON INCLUS
    Action 1
FIN POUR
    Action 2
```

Python

```
for n in range(ni, nf) :
    Action 1
Action 2
```

Arduino (C++)

```
for (ni ; n>nf ; n++){
    Action 1 ;
}
{
    Action2 ;
}
```