Chap.3 : Introduction à l'algorithmique

1 Introduction

Un ordinateur est une machine électronique programmable servant au traitement de l'information codée sous forme binaire, c'est-à-dire sous forme de tout ou rien (soit le courant passe, soit il ne passe pas). Contrairement à la vision des films de science-fiction, un ordinateur est une machine totalement dénuée d'intelligence. Il n'est capable de traiter qu'un nombre limité d'instructions. Donc il ne faut en aucun cas être intimidé par les ordinateurs : ils sont infiniment plus bêtes que vous. Ce n'est que lorsqu'on réalise vraiment la stupidité des ordinateurs qu'on commence à progresser, car il faut s'abaisser à son niveau : il faut tout lui dire, car il fait tout au pied de la lettre, sans réfléchir.
Pourtant, contrairement aux autres machines qui sont dédiées à un nombre limité de tâches, l'ordinateur est potentiellement capable d'effectuer une infinité de tâches concernant le traitement rationnel de l'information. On dit que c'est une machine universelle.

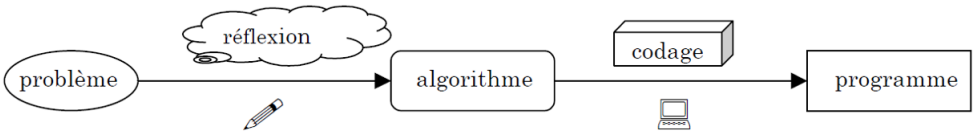
Alors comment une machine « stupide » peut traiter autant de problèmes différents ?

C'est que, grâce aux actions de base qu'elle sait réaliser, il est possible en les assemblant de façon pertinente, de résoudre la plupart des problèmes concernant le traitement de l'information. Il suffit de lui indiquer l'ordre dans lequel il faut qu'il effectue ces actions basiques et avec quelles données. Ces ordres élémentaires sont appelés instructions et sont rassemblés au sein d'un programme. Comme l'ordinateur a l'avantage d'exécuter très rapidement et sans erreurs les ordres qu'on lui donne (les instructions), il exécute beaucoup de traitements complexes plus vite et plus sûrement qu'un homme.
Pour donner des ordres à l'ordinateur, il est nécessaire de pouvoir communiquer avec lui. Cette communication passe par un langage de programmation, dans lequel est écrit le programme.

Un programme est un assemblage et un enchaînement d'instructions élémentaires écrit dans un langage de programmation, et exécuté par un ordinateur afin de traiter les données d'un problème et renvoyer un ou plusieurs résultats.

Un algorithme représente l'enchaînement des actions (instructions) nécessaires pour faire exécuter une tâche à un ordinateur (résoudre un problème). Un algorithme s'écrit le plus souvent en pseudo-langage de programmation(appelé langage algorithmique).

Un algorithme n'est donc exécutable directement par aucune machine. Mais il a l'avantage d'être traduit facilement dans tous les langages de programmation.
L'algorithmique, l'art d'écrire des algorithmes, permet de se focaliser sur la procédure de résolution du problème sans avoir à se soucier des spécificités d'un langage particulier. Pour résoudre un problème, il est vivement conseillé de réfléchir d'abord à l'algorithme avant de programmer proprement dit, c'est à dire d'écrire le programme en langage de programmation.



2 Présentation d'un langage d'écriture des algorithmes

2.1 Les données

Les algorithmes sont amenés à manipuler des données. Ces données seront repérées par des noms. En informatique le nom d'une donnée est appelé identificateur. La règle communément adoptée pour écrire un identificateur est une suite de lettres ou de chiffres commençant par une lettre, le caractère espace étant interdit, pour faciliter la lecture des noms composés le caractère "_" (appelé *"tiret bas"*, ou encore *"underscore"*) est considéré comme une lettre. Une donnée possède toujours un type.

Types élémentaires :

- caractère : lettres minuscules, lettres majuscules, chiffres, symboles comme +, *, /, -, @, <,. . . .
- texte : un texte, appelé en informatique chaîne de caractères (string en anglais) est une suite finie de caractères.
- Booléen : une donnée de type booléen ne peut avoir que deux valeurs possibles vrai ou faux (true ou false).
- Nombre entier
- Flottant : nombres réels (attention, en informatique que des approximations des nombres réels).

Deux types composés : les types tableau (array) et liste (list) représentent une séquence linéaire d'éléments. A savoir qu'il y a un premier élément, un dernier élément et que l'élément de rang i est précédé (s'il n'est pas le premier) de l'élément de rang $i-1$ et suivi (s'il n'est pas le dernier) de l'élément de rang $i+1$.

- **Tableau** : dans ce type de données il est possible d'accéder à un élément par son indice.
  On peut ainsi le consulter, le modifier, insérer avant ou après un autre élément. On utilise la notation $t[i]$ pour désigner la composante de rang i du tableau t. En informatique, lorsqu'on programme, ce qui caractérise un tableau est que ses éléments, tous du même type, sont stockés séquentiellement, c'est-à-dire que les éléments se trouvent à des adresses consécutives en mémoire.
- **Liste** : comme pour une donnée de type tableau il est possible d'accéder à un élément par son numéro pour le consulter, le modifier. En Python, $l[i]$ désigne la composante de rang i+1 de la liste l.
  On désigne la **liste vide** par []. En informatique, ce qui caractérise une "vraie" liste est que les éléments ne sont plus situés à des emplacements consécutifs de la mémoire. Les éléments d'une liste, contrairement aux éléments d'un tableau, peuvent être de différents types.

  Exemple sous Python :

```
>>> l=[1,'aaa',2==5]
>>> l[0]
1
>>> l[2]
False
>>> id(l[1])
>>> id(l[1])
90704352
>>> id(l[2])
505435356
>>> type(l[1])
<class 'str'>
>>>
```

2.2 Les variables

Dans la description d'un algorithme on est amené à donner des noms aux données manipulées par celui-ci. Ainsi à chaque donnée est associé un nom et un type. Le nom permet de désigner la donnée et le type permet de préciser la nature de cette donnée, ce qui détermine les opérations que l'on peut lui appliquer. Certains langages (Pascal, C, ...) obligent le programmeur à préciser le type de chaque variable avant son utilisation, d'autres ont fait le choix inverse (Caml,Python). Dans ce dernier cas c'est la façon de les manipuler (initialisations, opérateurs utilisés, ...) qui permet de leur

attribuer un type. Dans tous les cas, à une variable est toujours associé un type.

Sous Python, pour afficher le type d'une expression, on utilise l'instruction « type » :

```
>>> type(42)
<class 'int'>
>>> type(2.3)
<class 'float'>
>>> type(4)
<class 'int'>
>>> type('tsi1')
<class 'str'>
>>> type(['maths',3,'physique',4])
<class 'list'>
```

```
>>> type(2==3)
<class 'bool'>
```

2.3 Les instructions

2.3.1 L'instruction d'affectation

Cette instruction permet de modifier la valeur d'une variable. Une syntaxe pourrait être :

"expression partie gauche" $\leftarrow$ **"expression partie droite"**

L'expression en partie gauche (très souvent une variable) doit impérativement désigner une adresse. Cette instruction a pour but de ranger la valeur de l'expression partie droite à l'adresse indiquée par la partie gauche.

Exemple : $k \longleftarrow k+1$, la variable k reçoit la valeur de l'expression $k+1$. On dit aussi que l'on affecte à k la valeur de l'expression $k+1$. Le symbole varie suivant les langages.
En Python, on notera « $k = k+1$ » ou $k+=1$.

```
>>> k=3
>>> k=k+1
>>> print(k)
4
```

```
>>> x=3
>>> y=2
>>> x=x+y
>>> print(x)
5
```

2.3.2 Les instructions conditionnelles

- L'instruction **"si ... alors"** : elle s'écrit :

  si "expression booléenne" alors "instruction".

L'instruction qui suit le "alors" n'est exécutée que si l'expression booléenne est vraie. Dans le cas contraire, c'est l'instruction suivante (celle qui suit le "si alors") qui est exécutée.
Dans le cas où l'on souhaiterait exécuter après le alors non pas une, mais plusieurs instructions, on écrira ces instructions sur plusieurs lignes (une instruction par ligne) précédées par un trait vertical qui englobe toutes ces instructions.

si "expression booléenne" **alors**
| *instruction* 1
| *instruction* 2
| ⋮
| *instruction n*

- L'instruction **"si... alors... sinon"** Elle s'écrit : si "expression booléenne" alors "instruction 1" sinon "instruction 2" Si l'évaluation de l'expression donne la valeur vraie, c'est l'instruction 1 qui suit immédiatement le "alors" qui est exécutée, sinon c'est l"instruction 2 qui suit le sinon qui est exécutée.

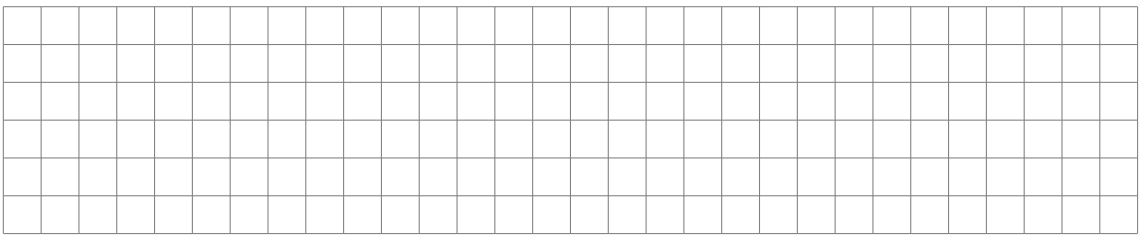
En Python, la syntaxe est la suivante :

```
k=6
if k==3:
    print('k est égal à 3')
else:
    if k<3:
        print('k est plus petit que 3')
    else:
        print('k est plus grand que 3')
```

Exemple 2.1. *Que fait cet algorithme ?*

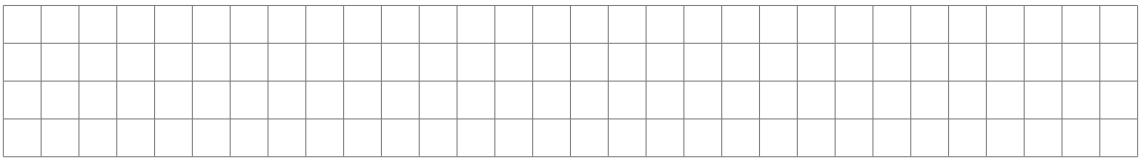
```
nb1=float(input('saisir un premier nombre'))
nb2=float(input('saisir un autre nombre'))
l=str(input('saisir le premiere lettre de l operation a effectuer'))
if l=='a':
    print(nb1,'+',nb2,'=',nb1+nb2)
elif l=='m':
    print(nb1,'*',nb2,'=',nb1*nb2)
else :
    print('operation non prise en charge')
```

Exemple 2.2. *Supposons que nous voulions afficher la valeur absolue de la différence entre deux nombres entiers. Ces nombres entiers seront notés x et y. Écrire un algorithme.*



Exemple 2.3. *Que fait cet algorithme ?*

```
nb=float(input('saisir un prix'))
if nb<100:
    print('prix a payer =',nb)
elif nb>=100 and nb<300:
    print('prix a payer =',nb*0.9)
elif nb>=300 and nb<500:
    print('prix a payer =',nb*0.8)
else :
    print('prix a payer =',nb*0.7)
```



2.3.3 Les instructions itératives

- **La boucle "tant que"** : elle s'écrit :

  tant que "expression booléenne " faire " instruction".

  Si l'expression booléenne est vraie on exécute l'instruction qui suit faire, puis l'expression" booléenne est réévaluée à nouveau et ainsi de suite.
  La boucle s'arrête lorsque l'évaluation de l'expression booléenne donne la valeur faux. On passe alors à l'instruction suivante.
  Dans le cas où l'on souhaiterait exécuter non pas une, mais plusieurs instructions, on écrira ces instructions sur plusieurs lignes (une instruction par ligne) précédées par un trait vertical :

tant que "expression booléenne " **faire**
    *instruction*1
    *instruction*2
    ⋮
    instruction n

En Python :

```
p=1
c=4
while c>0:
    p=p*2
    c=c-1
print(p)
```

Exemple 2.4. *Que fait ce programme ?*

```
x=int(input('saisir un nombre entier'))
while x!=0:
    print('Le cube de la valeur saisie est',x**3)
    x=int(input('saisir un nombre entier'))
```

Exemple 2.5. *Que fait ce programme ?*

```
n=int(input('Saisir un entier naturel'))
somme=0
i=0
while i<=n:
    somme=somme+i
    i=i+1
print(somme)
```

- **La boucle "pour"** :

pour "variable" **variant de** "expr. init" **jusqu'à** "expr. fin" **faire**
    *instruction*1
    *instruction*2
    ⋮
    instruction n

ATTENTION : on s'interdira de modifier la variable de contrôle à l'intérieur de la boucle et de l'utiliser en dehors de celle-ci.

Exemple 2.6. *Que fait ce programme ?*

```
p=3
n=int(input('saisir un entier n'))
for i in range(n):
    if n%2==0:
        p=p*4
    else :
        p=p*5
print(p)
```

Exemple 2.7. *Écrire un algorithme qui calcule et affiche la valeur de* $n! = 1 \times 2 \times ... \times n$ *où* n *est un nombre entier saisi par l'utilisateur.*

Exemple 2.8. *Écrire l'algorithme qui permet de calculer un nombre à un exposant donné. Le nombre* x *et l'exposant* n *sont saisis.*

Exemple 2.9. *Écrire un programme sous Python qui :*

- *demande à l'utilisateur de saisir 10 nombres*
- *stocke ces 10 nombres dans une listez*
- *affiche le plus petit nombre*