HAUTE ECOLE DE COMMERCE DE KINSHASA

B.P. KINSHASA X

SECTION INFORMATIQUE

Cours d'Algorithmiques et Structure des données 1

A l'usage des étudiants du Premier Licence LMD en Informatique

Titulaire du Cours

BUKANGA CHRISTIAN Parfum

Chef de Travaux

Ce support présente les notions de base de l'algorithme et donne les notions du langage Python, de la conception d'un programme simple à la manipulation des fichiers

ANNEE – ACADEMIQUE: 2023-2024

À l'origine, le présent support est conçu à l'intention des apprenants du premier graduat en informatique. Il nous a semblé par la suite que ce cours pouvait également convenir à toute personne n'ayant encore jamais programmé en python, mais souhaitant s'initier à cette discipline en autodidacte.

Nous y proposons une démarche d'apprentissage non linéaire qui est très certainement critiquable. Nous sommes conscients qu'elle apparaîtra un peu chaotique aux yeux de certains puristes, mais nous l'avons voulue ainsi parce que nous sommes convaincus qu'il existe de nombreuses manières d'apprendre (pas seulement la programmation, d'ailleurs), et qu'il faut accepter d'emblée ce fait établi que des individus différents n'assimilent pas les mêmes concepts dans le même ordre. Nous avons donc cherché avant tout à susciter l'intérêt et à ouvrir un maximum de portes, en nous efforçant tout de même de respecter les principes directeurs suivants :

- ✓ L'apprentissage que nous visons se veut généraliste : nous souhaitons mettre en évidence les invariants de la programmation et de l'algorithme, sans nous laisser entraîner vers une spécialisation quelconque, ni supposer que le lecteur dispose de capacités intellectuelles hors du commun.

- ✓ Les outils utilisés au cours de l'apprentissage doivent être modernes et performants, mais il faut aussi que le lecteur puisse se les procurer en toute légalité sur internet pour son usage personnel. Notre texte s'adresse en effet en priorité à des étudiants, et toute notre démarche d'apprentissage vise à leur donner la possibilité de mettre en chantier le plus tôt possible des réalisations personnelles qu'ils pourront développer et exploiter à leur guise.

BUKANGA CHRISTIAN Parfum

Introduction

0.1. Définition

L'utilisation d'un ordinateur pour la résolution d'une application (travail que l'on soumet à un ordinateur) nécessite tout un travail de préparation. N'ayant aucune capacité d'invention, l'ordinateur ne peut en effet qu'exécuter les ordres qui lui sont fournis par l'intermédiaire d'un programme. Ce dernier doit donc être établi de manière à envisager toutes les éventualités d'un traitement.

- ✓ L'algorithmique est un terme d'origine arabe, composé d'une suite d'instructions élémentaires, qui une fois exécutée correctement, conduit à un résultat donné.
- ✓ Un algorithme : est une suite finie d'instructions que l'on applique à un nombre fini de données dans un ordre précis pour arriver à un résultat.
- ✓ L'algorithmique désigne la discipline qui étudie les algorithmes et leurs applications en informatique

0.2. Propriétés et étape d'un algorithme

1. Propriétés d'un algorithme

a. Un algorithme doit :
   - Avoir un nombre fini d'étapes,
   - Avoir un nombre fini d'opérations par étape,
   - Se terminer après un nombre fini d'opérations,
   - Fournir un résultat.

b. Chaque opération doit être :
   - Définie rigoureusement et sans ambiguïté
   - Effective, c.-à-d. réalisable par une machine

c. Le comportement d'un algorithme est déterministe

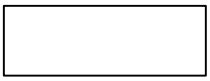
2. Etape d'un algorithme

L'algorithme comprend trois étapes : les entrées, le traitement et la sortie

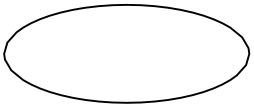
- *Les entrées* : Il s'agit de repérer les données nécessaires à la résolution du problème.
- *Le traitement* : Il s'agit de déterminer toutes les étapes des traitements à faire et donc des "instructions" à développer pour arriver aux résultats.
- *Les sorties* : les résultats obtenus peuvent être affichés sur l'écran, ou imprimés sur papier, ou bien encore conservés dans un fichier.

0.3. Représentation graphique d'un algorithme ou ordinogramme

La représentation graphique permet une lecture aisée des algorithmes mais présente toutefois l'inconvénient de consommer une place importante. Les opérations dans un organigramme sont représentées par les symboles dont les formes sont normalisées. Ces symboles sont reliés entre eux par des lignes fléchées qui indiquent le chemin. C'est ainsi qu'on a :

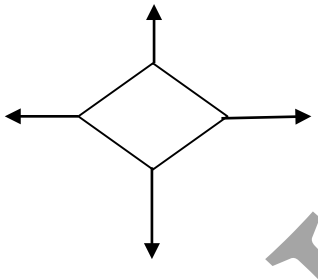


Est utilisé pour les calculs



Est utilisé pour le début et fin

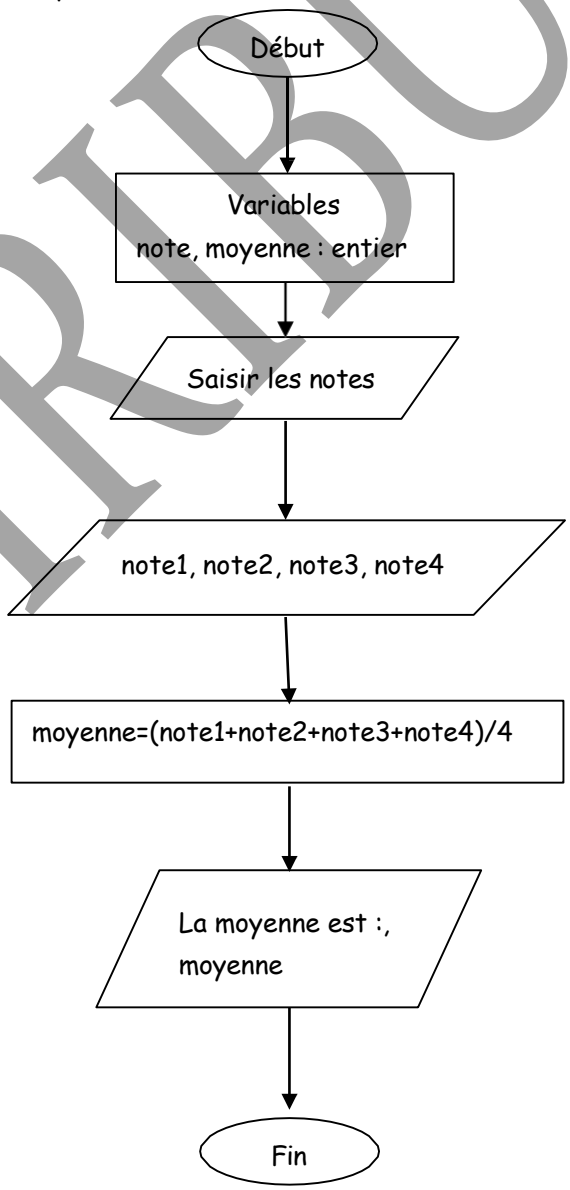
Est utilisé pour la lecture et l'affichage



Est utilisé pour représenter des tests

Exemple : créer un algorithme pour calculer la moyenne de 4 notes

Solution : algo moyenne
//déclaration des variables
Variables note, moyenne : entier
Début
// saisie des notes au clavier
Ecrire (entrer les notes)
Lire (note1, note2, note3)
//calcul ou traitement
Moyenne= (note1+note2+note3) /3
//affichage du résultat
Ecrire (la moyenne est :,moyenne)
Fin



0.4. Les étapes de résolution d'un problème

La résolution d'un problème comprend quatre étapes, il s'agit de :

1. Comprendre l'énoncé du problème

Il s'agit de déterminer toutes les informations disponibles et la forme des résultats désirés.

2. Décomposer le problème en sous-problèmes plus simple à résoudre

Il s'agit de décomposer les informations disponibles en sous-problèmes.

3. Associer à chaque sous problème, une spécification :

   - ✓ Les données nécessaires
   - ✓ Les données résultantes
   - ✓ La démarche à suivre pour arriver au résultat en partant d'un ensemble de données.

4. Elaboration d'un algorithme.

Une fois qu'on trouve le moyen de passer des données aux résultats, il faut être capable de rédiger une solution claire et non ambiguë. Comme il est impossible de le faire en langage naturel pour les raisons que nous donnerons par la suite, l'existence d'un langage algorithmique s'impose.

0.5. Langage de programmation

Il existe de nombreux langages de programmation : C, C++, Java, Basic, Pascal, Fortran, ... Le langage Python est utilisé dans ce cours en raison de son caractère pédagogique.

CHAP1 : NOTION DE VARIABLES ET D'AFFECTATION

1.1. Les variables

Une **variable** sert à stocker la valeur d'une donnée dans un langage de programmation autrement dit une variable désigne un emplacement mémoire dont le contenu peut changer au cours d'un programme (d'où le nom de variable)
Chaque emplacement mémoire a un numéro qui permet d'y faire référence de façon unique : c'est l'adresse mémoire de cette cellule.

Le **nom** de la variable (l'étiquette de la boîte) obéit à des impératifs changeant selon les langages. Toutefois, une règle absolue est qu'un nom de variable peut comporter des lettres et des chiffres, mais qu'il exclut la plupart des signes de ponctuation, en particulier les espaces. Un nom de variable correct commence également impérativement par une lettre. Quant au nombre maximal de signes pour un nom de variable, il dépend du langage utilisé.

En pseudo-code algorithmique, on est bien sûr libre du nombre de signes pour un nom de variable, même si pour des raisons purement pratiques, et au grand désespoir de Stéphane Bern, on évite généralement les noms à rallonge.

Lorsqu'on déclare une variable, il ne suffit pas de créer une boîte (réserver un emplacement mémoire) ; encore doit-on préciser ce que l'on voudra mettre dedans, car de cela dépendent la **taille** de la boîte (de l'emplacement mémoire) et le **type de codage** utilisé.

1.1.1. Déclaration d'une variable

La variable doit être **déclarée** avant d'être utilisée, elle doit être caractérisée par : un nom (**Identificateur**), un **type** qui indique l'ensemble des valeurs que peut prendre la variable (entier, réel, booléen, caractère, chaîne de caractères, …) et une valeur.

Le **nom** de la variable (l'étiquette de la boîte) obéit à des impératifs changeant selon les langages. Toutefois, une règle absolue est qu'un nom de variable peut comporter des lettres et des chiffres, mais qu'il exclut la plupart des signes de ponctuation, en particulier les espaces. Un nom de variable correct commence également impérativement par une lettre. Quant au nombre maximal de signes pour un nom de variable, il dépend du langage utilisé.

Le choix du nom d'une variable est soumis à quelques règles qui varient selon le langage, mais en général :

- ✓ Un nom doit commencer par une lettre alphabétique
  Exemple : E1 (1E n'est pas valide)
- ✓ Doit être constitué uniquement de lettres, de chiffres et du soulignement (« _ ») (Éviter les caractères de ponctuation et les espaces)
- ✓ Doit être différent des mots réservés du langage (par exemple en C:**int, float, double, switch, case, for, main, return**, …)
- ✓ La longueur du nom doit être inférieure à la taille maximale spécifiée par le langage utilisé

Remarque : pour une bonne lisibilité du code il faut choisir des noms significatifs qui décrivent les données manipulées.

Exemple : NoteEtudiant, Nombre_cours, Prix_HT

1.1.2. Type de variable

Le type d'une variable détermine l'ensemble des valeurs qu'elle peut prendre. Les types offerts par la plupart des langages sont : Type numérique, type alphanumérique, type booléen.

a) Type numérique

Une variable de type **numérique** peut prendre comme valeur l'ensemble des nombres entiers signés. Les opérations associées sont les opérations usuelles +, -, *, /.
Dans la gamme du type numérique nous avons :

- ✓ **Byte** (codé sur 1octet) : de [-27,27[ou [0, 28[
- ✓ **Entier court** (codé sur 2 octets) : [-215,215[
- ✓ **Entier long** (codé sur 4 octets): [-231,231[
- ✓ **Réel simple précision** (codé sur 4 octets) : précision d'ordre 10-7
- ✓ **Réel double précision** (codé sur 8 octets) : précision d'ordre 10-14

En pseudo-code, une déclaration de variables aura ainsi cette tête :

Variable Prixtot **en Numérique**
ou encore

Variables PrixUnit, TauxTVA, Qtecmd **en Numérique**

b) Type alphanumérique

Une variable de type **car** peut prendre comme valeur l'ensemble des caractères imprimables. On notera les valeurs entre guillemets. On considère souvent que les caractères sont ordonnés dans l'ordre alphabétique.
On dispose donc également du **type alphanumérique** (également appelé **type caractère**, **type chaîne** ou en anglais, le **type string**.

Dans une variable de ce type, on stocke des **caractères**, qu'il s'agisse de lettres, de signes de ponctuation, d'espaces, ou même de chiffres.

En pseudo-code, une déclaration de variables destinée à recevoir des caractères aura ainsi cette forme :

Variable Cours en Caractère

Remarque ; tout nombre entre guillemet s'interprète comme une chaine de caractère et non un entier. Exemple : "125 " est un string à l'inverse de 125 qui est un numérique.

c) Type booléen

Le dernier type de variables est le type **booléen** : on y stocke uniquement les valeurs logiques VRAI et FAUX.

On peut représenter ces notions abstraites de VRAI et de FAUX par tout ce qu'on veut : de l'anglais (TRUE et FALSE) ou des nombres (0 et 1). Peu importe. Ce qui compte, c'est de comprendre que le type booléen est très économique en termes de place mémoire occupée, puisque pour stocker une telle information binaire, un seul bit suffit.

1.1.3. Constance

Une constante est une variable dont la valeur ne change pas au cours de l'exécution du programme, elle peut être un nombre, un caractère, ou une chaine de caractères.

En pseudo-code, Constante identificateur = valeur : type, (par convention, les noms de constantes sont en majuscules)

Pour calculer la surface des cercles, la valeur de pi est une constante mais le rayon est une variable.

Constante PI=3.14 : réel,

Exemple d'une déclaration

```
Variables Nbrcours, PtOb : entiers
          Mention : chaines de caractère
          Pourcentage : réel
Constante PI=3.14
```

1.2. L'affectation

1.2.1. Syntaxe et signification d'affection

L'affectation consiste à attribuer une valeur à une variable (c'est-à-dire remplir ou modifier le contenu d'une zone mémoire).
Cette seule chose qu'on puisse faire avec une variable, c'est l'affecter, c'est-à-dire lui attribuer une valeur. Pour poursuivre la superbe métaphore filée déjà employée, on peut remplir la boîte.

En pseudo-code, l'affectation est notée par le signe ←

Var ←E : attribue la valeur de E à la variable Var

- ✓ E peut être une valeur, une autre variable ou une expression
- ✓ Var et e doivent être de même type ou de types compatibles
- ✓ L'affectation ne modifie que ce qui est à gauche de la flèche

Exemple : dorcas ←16

Ceci signifierait que la variable *dorcas* doit impérativement être du type numérique. Si *dorcas* a été défini dans un autre type, il faut bien comprendre que cette instruction provoquera une erreur.

On peut en revanche sans aucun problème attribuer à une variable la valeur d'une autre variable, telle quelle ou modifiée. Par exemple :

Pamphile←dorcas // Signifie que la valeur de *dorcas* est maintenant celle de *Pamphile*.

Notons que cette instruction n'a en rien modifié la valeur de Pamphile : une instruction d'affectation ne modifie que ce qui est situé à gauche de la flèche.

Pamphile ←dorcas + 4// Si dorcas contenait 24, Pamphile vaut maintenant 28. De même que précédemment, dorcas vaut toujours 24.

Pamphile= Pamphile+1 // Si Pamphile valait 6, il vaut maintenant 7. La valeur de Pamphile est modifiée, puisque Pamphile est la variable située à gauche de la flèche.

Pour revenir à présent sur le rôle des guillemets dans les chaînes de caractères et sur la confusion numéro 2 signalée plus haut, comparons maintenant deux algorithmes suivants :

Exemple n°1

```
Début
        Uka←"Informatique"
        Faculte←"Uka"
Fin
```

Exemple n°2

```
Début
        Uka←"Informatique"
        Faculte←Uka
Fin
```

La seule différence entre les deux algorithmes consiste dans la présence ou dans l'absence des guillemets lors de la seconde affectation. Et l'on voit que cela change tout !

Dans l'exemple n°1, ce que l'on affecte à la variable Faculte, c'est la suite de caractères U– k– a. Et à la fin de l'algorithme, le contenu de la variable Faculte est donc « Uka ».

Dans l'exemple n°2, en revanche, Faculte étant dépourvu de guillemets, n'est pas considéré comme une suite de caractères, mais comme un nom de variable. Le sens de la ligne devient donc : « affecte à la variable Faculte le contenu de la variable Uka ». A la fin de l'algorithme n°2, la valeur de la variable Faculte est donc « Informatique ». Ici, l'oubli des guillemets conduit certes à un résultat, mais à un résultat différent.

A noter, car c'est un cas très fréquent, que généralement, lorsqu'on oublie les guillemets lors d'une affectation de chaîne, ce qui se trouve à droite du signe d'affectation ne correspond à aucune variable précédemment déclarée et affectée. Dans ce cas, l'oubli des guillemets se solde immédiatement par une erreur d'exécution. Ceci est une simple illustration. Mais elle résume l'ensemble des problèmes qui surviennent lorsqu'on oublie la règle des guillemets aux chaînes de caractères.

1.2.2. Ordre des instructions

Il va de soi que l'ordre dans lequel les instructions sont écrites va jouer un rôle essentiel dans le résultat final. Considérons les deux algorithmes suivants :

Exemple 1

```
Variable A en Numérique
Début
A ←34
A ←12
Fin
```

Exemple 2
Variable A en Numérique
Début
A ←12
A ←34
Fin

Il est clair que dans le premier cas la valeur finale de A est 12, dans l'autre elle est 34 .

Il est tout aussi clair que ceci ne doit pas nous étonner. Lorsqu'on indique le chemin à quelqu'un, dire « prenez tout droit sur 1km, puis à droite » n'envoie pas les gens au même endroit que si l'on dit « prenez à droite puis tout droit pendant 1 km ».

Énoncé des Exercices

Exercice 1.1

Quelles seront les valeurs des variables A et B après exécution des instructions suivantes ?

Variables A, B en Entier

Début
A ←1
B ←A + 3
A ←3
Fin

Exercice 1.2

Quelles seront les valeurs des variables A, B et C après exécution des instructions suivantes ?

Variables A, B, C en Entier

Début
A ←5
B ←3
C ←A + B
A ←2
C ←B – A
Fin

Exercice 1.3

Quelles seront les valeurs des variables A et B après exécution des instructions suivantes ?

Variables A, B en Entier
Début
A ←5
B ←A + 4
A ←A + 1
B ←A – 4
Fin

Exercice 1.4

Quelles seront les valeurs des variables A, B et C après exécution des instructions suivantes ?

Variables A, B, C en Entier
Début
A ←3
B ←10
C ←A + B
B ←A + B
A ←C
Fin

Exercice 1.5

Quelles seront les valeurs des variables A et B après exécution des instructions suivantes ?

Variables A, B en Entier
Début
A ←5
B ←2
A ←B
B ←A
Fin

Moralité : les deux dernières instructions permettent-elles d'échanger les deux valeurs de B et A ? Si l'on inverse les deux dernières instructions, cela change-t-il quelque chose ?

Exercice 1.6

Plus difficile, mais c'est un classique absolu, qu'il faut absolument maîtriser : écrire un algorithme permettant d'échanger les valeurs de deux variables A et B, et ce quel que soit leur contenu préalable.

Exercice 1.7

Une variante du précédent : on dispose de trois variables A, B et C. Ecrivez un algorithme transférant à B la valeur de A, à C la valeur de B et à A la valeur de C (toujours quels que soient les contenus préalables de ces variables).

Corrigés des Exercices

Exercice 1.1

Après La valeur des variables est :
A ← 1 A = 1 B = ?
B ← A + 3 A = 1 B = 4
A ← 3 **A = 3 B = 4**

Exercice 1.2

Après La valeur des variables est :
A ← 5 A = 5 B = ? C = ?
B ← 3 A = 5 B = 3 C = ?

C ← A + B A = 5 B = 3 C = 8
A ← 2 A = 2 B = 3 C = 8
C ← B – A **A = 2 B = 3 C = 1**

Exercice 1.3

Après La valeur des variables est :
A ← 5 A = 5 B = ?
B ← A + 4 A = 5 B = 9
A ← A + 1 A = 6 B = 9
B ← A – 4 **A = 6 B = 2**

Exercice 1.4

Après La valeur des variables est :
A ← 3 A = 3 B = ? C = ?
B ← 10 A = 3 B = 10 C = ?
C ← A + B A = 3 B = 10 C = 13
B ← A + B A = 3 B = 13 C = 13
A ← C **A = 13 B = 13 C = 13**

Exercice 1.5

Après La valeur des variables est :
A ← 5 A = 5 B = ?
B ← 2 A = 5 B = 2
A ← B A = 2 B = 2
B ← A **A = 2 B = 2**

Les deux dernières instructions ne permettent donc pas d'échanger les deux valeurs de B et A, puisque l'une des deux valeurs (celle de A) est ici écrasée.

Si l'on inverse les deux dernières instructions, cela ne changera rien du tout, hormis le fait que cette fois c'est la valeur de B qui sera écrasée.

Exercice 1.6

Début
…
C ← A
A ← B
B ← C
Fin

On est obligé de passer par une variable dite temporaire (la variable C).

Exercice 1.7

Début
…
D ← C
C ← B
B ← A
A ← D

Fin

En fait, quel que soit le nombre de variables, une seule variable temporaire suffit…

1.2.3. Expressions et Opérateurs

Si on fait le point, on s'aperçoit que dans une instruction d'affectation, on trouve :

- ✓ À gauche de la flèche, un nom de variable, et uniquement cela. En ce monde rempli de doutes qu'est celui de l'algorithmique, c'est une des rares règles d'or qui marche à tous les coups : si on voit à gauche d'une flèche d'affectation autre chose qu'un nom de variable, on peut être certain à 100% qu'il s'agit d'une erreur.
- ✓ À droite de la flèche, ce qu'on appelle une **expression**. Voilà encore un mot qui est trompeur ; en effet, ce mot existe dans le langage courant, où il revêt bien des significations. Mais en informatique, le terme d'**expression** ne désigne qu'une seule chose, et qui plus est une chose très précise :

Une expression est un ensemble de valeurs, reliées par des opérateurs, et équivalent à une seule valeur

Cette définition vous paraît peut-être obscure. Mais réfléchissez-y quelques minutes, et vous verrez qu'elle recouvre quelque chose d'assez simple sur le fond. Par exemple, voyons quelques expressions de type **numérique**. Ainsi :

7
5+4
123-45+844
Pamphile-12+5-Dorcas

…sont toutes des expressions valides, pour peu que Pamphile et Dorcas soient bien des nombres. Car dans le cas contraire, la quatrième expression n'a pas de sens. En l'occurrence, les opérateurs que j'ai employés sont l'addition (+) et la soustraction (-).
Revenons pour le moment sur l'affectation. Une condition supplémentaire (en plus des deux précédentes) de validité d'une instruction d'affectation est que :

- ✓ L'expression située à droite de la flèche soit du même type que la variable située à gauche. C'est très logique : on ne peut pas ranger convenablement des outils dans un sac à provision, ni des légumes dans une trousse à outils… sauf à provoquer un résultat catastrophique.

Si l'un des trois points énumérés ci-dessus n'est pas respecté, la machine sera incapable d'exécuter l'affectation, et déclenchera une erreur (est-il besoin de dire que si aucun de ces points n'est respecté, il y aura aussi erreur !)

On va maintenant détailler ce que l'on entend par le terme d'opérateur.

Un opérateur est un signe qui relie deux valeurs, pour produire un résultat.

Les opérateurs possibles dépendent du type des valeurs qui sont en jeu.

a) Opérateur numérique

Ce sont les quatre opérations arithmétiques tout ce qu'il y a de classique.

+ : addition

- : soustraction

* : multiplication

/ : division

Mentionnons également le ^ qui signifie « puissance ». 45 au carré s'écrira donc 45 ^ 2.

Enfin, on a le droit d'utiliser les parenthèses, avec les mêmes règles qu'en mathématiques. La multiplication et la division ont « naturellement » priorité sur l'addition et la soustraction. Les parenthèses ne sont ainsi utiles que pour modifier cette priorité naturelle.

Cela signifie qu'en informatique, 12 * 3 + 5 et (12 * 3) + 5 valent strictement la même chose, à savoir 41. Pourquoi dès lors se fatiguer à mettre des parenthèses inutiles ?

En revanche, 12 * (3 + 5) vaut 12 * 8 soit 96. Rien de difficile là-dedans, que du normal.

b) Opérateur alphanumérique &

Cet opérateur permet de **concaténer**, autrement dit d'agglomérer, deux chaînes de caractères. Par exemple :

Variables A, B, C en Caractère

Début
A ←"Infor"
B ←"matique"
C ←A & B
Fin

La valeur de C à la fin de l'algorithme est "Informatique"

c) Opérateur logique(booléen)

Il s'agit du ET, du OU, du NON et du mystérieux (mais rarissime XOR). Nous les laisserons de côté… provisoirement, soyez-en sûrs.

Énoncé des Exercices

Exercice 1.8

Que produit l'algorithme suivant ?

Variables A, B, C **en Caractères**
Début

A ←"423"
B ←"12"
C ←A + B
Fin

Exercice 1.9

Que produit l'algorithme suivant ?

Variables A, B, C **en Caractères**
Début
A ←"423"
B ←"12"
C ←A & B
Fin

Corrigés des Exercices

Exercice 1.8

Il ne peut produire qu'une erreur d'exécution, puisqu'on ne peut pas additionner des caractères.

Exercice 1.9

…En revanche, on peut les concaténer. A la fin de l'algorithme, C vaudra donc "42312".

1.3. La lecture et l'écriture

1.3.1. Introduction

Les instructions de lecture et d'écriture permettent à la machine de communiquer avec l'utilisateur.

Dans un sens, ces instructions permettent à l'utilisateur de rentrer des valeurs au clavier pour qu'elles soient utilisées par le programme. Cette opération est la **lecture***(saisie)*.

Dans l'autre sens, d'autres instructions permettent au programme de communiquer des valeurs à l'utilisateur en les affichant à l'écran. Cette opération est l'**écriture(***affichage***)**.

Le programme s'arrête lorsqu'il rencontre une instruction Lire et ne se poursuit qu'après la saisie de l'entrée attendue par le clavier et de la touche Entrée (cette touche signale la fin de l'entrée)

Avant de lire une variable, il est fortement conseillé d'écrire des messages à l'écran, afin de prévenir l'utilisateur de ce qu'il doit frapper

1.3.2. Les instructions de la saisie et d'affichage

En pseudo-code, on note : écrire (liste d'expressions) la machine affiche les valeurs des expressions décrite dans la liste.

Ces instructions peuvent être des variables ayant des valeurs, des nombres ou des commentaires sous forme de chaines de caractères.

Écrire (a, b+2, "Message")

La lecture permet d'entrer des donnés à partir du clavier (la saisie). **En pseudo-code, on note : lire (var) ;** Lorsque le programme rencontre une instruction de lecture (lire(var)), il passe la main à l'utilisateur pour la saisie de la valeur de var. Ce dernier doit appuyer sur la touche Entrée pour valider cette entrée.

Lire NomFamille

Énoncé des Exercices

Exercice 2.1

Quel résultat produit le programme suivant ?

Variables val, double numériques
Début
Val ←231
Double ←Val * 2
Ecrire Val
Ecrire Double
Fin

Exercice 2.2

Ecrire un programme qui demande un nombre à l'utilisateur, puis qui calcule et affiche le carré de ce nombre.

Exercice 2.3

Ecrire un programme qui lit le prix HT d'un article, le nombre d'articles et le taux de TVA, et qui fournit le prix total TTC correspondant. Faire en sorte que des libellés apparaissent clairement.

Exercice 2.4

Ecrire un algorithme utilisant des variables de type chaîne de caractères, et affichant quatre variantes possibles de la célèbre « belle marquise, vos beaux yeux me font mourir d'amour ». On ne se soucie pas de la ponctuation, ni des majuscules.

Corrigés des Exercices

Exercice 2.1

On verra apparaître à l'écran 231, puis 462 (qui vaut 231 * 2)

Exercice 2.2

Variables nb, carr **en Entier**
Début

```
Ecrire "Entrez un nombre :"
Lire nb
carr←nb * nb
Ecrire "Son carré est : ", carr
Fin
```

En fait, on pourrait tout aussi bien économiser la variable carr en remplaçant les deux avant-dernières lignes par :

```
Ecrire "Son carré est : ", nb*nb
```

C'est une question de style ; dans un cas, on privilégie la lisibilité de l'algorithme, dans l'autre, on privilégie l'économie d'une variable

Exercice 2.3

```
Variables nb, pht, ttva, pttcen Numérique
Début
Ecrire "Entrez le prix hors taxes :"
Lire pht
Ecrire "Entrez le nombre d'articles :"
Lire nb
Ecrire "Entrez le taux de TVA :"
Lire ttva
        pttc←nb * pht * (1 + ttva)
Ecrire "Le prix toutes taxes est : ", pttc
Fin
```

Là aussi, on pourrait squeezer une variable et une ligne en écrivant directement. :

```
Ecrire "Le prix toutes taxes est : ", nb * pht * (1 + ttva)
```

C'est plus rapide, plus léger en mémoire, mais un peu plus difficile à relire (et à écrire !)

Exercice 2.4

```
Variables t1, t2, t3, t4 en Caractère
Début
t1 ←"belle Marquise"
t2 ←"vos beaux yeux"
t3 ←"me font mourir"
t4 ←"d'amour"
Ecrire t1 & " " & t2 & " " & t3 & " " & t4
Ecrire t3 & " " & t2 & " " & t4 & " " & t1
Ecrire t2 & " " & t3 & " " & t1 & " " & t4
Ecrire t4 & " " & t1 & " " & t2 & " " & t3
Fin
```

CHAP2 : LES INSTRUCTIONS CONDITIONNELLES ET LES BOUCLES

2.1. Les instructions conditionnelles

2.1.1. Structure d'un test

Il n'y a que deux formes possibles pour un test ; la première est la plus simple, la seconde la plus complexe.

Première forme

Si booléen **Alors**
    Instructions
Finsi

Deuxième forme

Si booléen **Alors**
        Instructions 1
    Sinon Instructions 2
Finsi

Ceci appelle quelques explications.

Un **booléen** est une **expression** dont la valeur est VRAI ou FAUX. Cela peut donc être (il n'y a que deux possibilités) :

- une **variable** (ou une expression) de type booléen
- une **condition**

Nous reviendrons dans quelques instants sur ce qu'est une **condition** en informatique.

Toujours est-il que la structure d'un test est relativement claire. Dans la forme la plus simple, arrivé à la première ligne (Si… Alors) la machine examine la valeur du booléen. Si ce booléen a pour valeur VRAI, elle exécute la série d'instructions. Cette série d'instructions peut être très brève comme très longue, cela n'a aucune importance. En revanche, dans le cas où le booléen est faux, l'ordinateur saute directement aux instructions situées après le FinSi.

Dans le cas de la structure complète, c'est à peine plus compliqué. Dans le cas où le booléen est VRAI, et après avoir exécuté la série d'instructions 1, au moment où elle arrive au mot « Sinon », la machine saute directement à la première instruction située après le « Finsi ». De même, au cas où le booléen a comme valeur « Faux », la machine saute directement à la première ligne située après le « Sinon » et exécute l'ensemble des « instructions 2 ». Dans tous les cas, les instructions situées juste après le FinSi seront exécutées normalement.

En fait, la forme simplifiée correspond au cas où l'une des deux « branches » du Si est vide. Dès lors, plutôt qu'écrire « sinon ne rien faire du tout », il est plus simple de ne rien écrire. Et laisser un Si... complet, avec une des deux branches vides, est considéré comme une très grosse maladresse pour un programmeur, même si cela ne constitue pas à proprement parler une faute.

Exprimé sous forme de pseudo-code, la programmation de notre touriste de tout à l'heure donnerait donc quelque chose du genre :

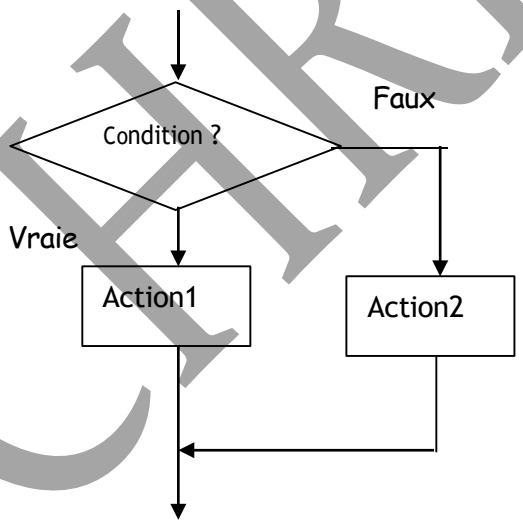
Allez tout droit jusqu'au prochain carrefour
Si la rue à droite est autorisée à la circulation **Alors**
    Tournez à droite
    Avancez
    Prenez la deuxième à gauche
Sinon
    Continuez jusqu'à la prochaine rue à droite
    Prenez cette rue
    Prenez la première à droite
Finsi

2.1.2. **Définition d'une condition**

Une condition est une expression écrite entre parenthèse à valeur booléenne
Cette définition est essentielle ! Elle signifie qu'une condition est composée de trois éléments : une valeur, un **opérateur de comparaison** et une autre valeur

Les valeurs peuvent être a priori de n'importe quel type (numériques, caractères…). Mais si l'on veut que la comparaison ait un sens, il faut que les deux valeurs de la comparaison soient du même type.

Les instructions conditionnelles servent à n'exécuter une instruction ou une séquence d'instructions que si une condition est vérifiée.



Énoncé des Exercices

Exercice 3.1

Ecrire un algorithme qui demande un nombre à l'utilisateur, et l'informe ensuite si ce nombre est positif ou négatif (on laisse de côté le cas où le nombre vaut zéro).

Corrigés des Exercices

Exercice 3.1

```
Variable n en Entier
Début
Ecrire "Entrez un nombre : "
Lire n
Si n > 0 Alors
        Ecrire "Ce nombre est positif"
 Sinon
        Ecrire "Ce nombre est négatif
Finsi
Fin
```

2.1.2. **Conditions composées**

Certains problèmes exigent parfois de formuler des conditions qui ne peuvent pas être exprimées sous la forme simple exposée ci-dessus. Reprenons le cas « Toto est inclus entre 5 et 8 ». En fait cette phrase cache non une, mais **deux** conditions. Car elle revient à dire que « Toto est supérieur à 5 et Toto est inférieur à 8 ». Il y a donc bien là deux conditions, reliées par ce qu'on appelle un **opérateur logique**, le mot ET.

Comme on l'a évoqué plus haut, l'informatique met à notre disposition quatre opérateurs logiques : ET, OU, NON, et XOR.

Le ET a le même sens en informatique que dans le langage courant. Pour que "Condition1 ET Condition2" soit VRAI, il faut impérativement que Condition1 soit VRAI et que Condition2 soit VRAI. Dans tous les autres cas, "Condition 1 et Condition2" sera faux.

Il faut se méfier un peu plus du OU. Pour que "Condition1 OU Condition2" soit VRAI, il suffit que Condition1 soit VRAIE ou que Condition2 soit VRAIE. Le point important est que si Condition1 est VRAIE et que Condition2 est VRAIE aussi, Condition1 OU Condition2 reste VRAIE. Le OU informatique ne veut donc pas dire « ou bien »

Le XOR (ou OU exclusif) fonctionne de la manière suivante. Pour que "Condition1 XOR Condition2" soit VRAI, il faut que soit Condition1 soit VRAI, soit que Condition2 soit VRAI. Si toutes les deux sont fausses, ou que toutes les deux sont VRAI, alors le résultat global est considéré comme FAUX. Le XOR est donc l'équivalent du "ou bien" du langage courant.

J'insiste toutefois sur le fait que le XOR est une rareté, dont il n'est pas strictement indispensable de s'encombrer en programmation.

Enfin, le NON inverse une condition : NON(Condition1)est VRAI si Condition1 est FAUX, et il sera FAUX si Condition1 est VRAI. C'est l'équivalent pour les booléens du signe "moins" que l'on place devant les nombres.

Alors, vous vous demandez peut-être à quoi sert ce NON. Après tout, plutôt qu'écrire NON(Prix > 20), il serait plus simple d'écrire tout bonnement Prix=<20. Dans ce cas précis, c'est évident qu'on se complique inutilement la vie avec le NON. Mais si le NON n'est jamais indispensable, il y a tout de même des situations dans lesquelles il s'avère bien utile.

Énoncé des Exercices

Exercice 3.2

Ecrire un algorithme qui demande deux nombres à l'utilisateur et l'informe ensuite si leur produit est négatif ou positif (on laisse de côté le cas où le produit est nul). Attention toutefois : on ne doit **pas** calculer le produit des deux nombres.

Exercice 3.3

Ecrire un algorithme qui demande trois noms à l'utilisateur et l'informe ensuite s'ils sont rangés ou non dans l'ordre alphabétique.

Corrigés des Exercices

Exercice 3.2

```
Variables m, n en Entier
Début
Ecrire "Entrez deux nombres : "
Lire m, n
Si (m > 0 ET n > 0) OU (m < 0 ET n < 0) Alors
        Ecrire "Leur produit est positif"
    Sinon
        Ecrire "Leur produit est négatif"
Finsi

Fin
```

Exercice 3.3

```
Variables a, b, c en Caractère
Début
Ecrire "Entrez successivement trois noms : "
Lire a, b, c
Si a < b ET b < c Alors
        Ecrire "Ces noms sont classés alphabétiquement"
  Sinon
        Ecrire "Ces noms ne sont pas classés"
Finsi
Fin
```

2.1.3. Test imbriqué

Graphiquement, on peut très facilement représenter un SI comme un aiguillage de chemin de fer (ou un aiguillage de train électrique, c'est moins lourd à porter). Un SI ouvre donc deux voies, correspondant à deux traitements différents. Mais il y a des tas de situations où deux voies ne suffisent pas. Par exemple, un programme devant donner l'état de l'eau selon sa température doit pouvoir choisir entre trois réponses possibles (solide, liquide ou gazeuse).

Une première solution serait la suivante :

```
Variable Temp en Entier
Début
Ecrire "Entrez la température de l'eau :"
Lire Temp
Si Temp =< 0 Alors
        Ecrire "C'est de la glace"
FinSi
Si Temp > 0 Et Temp < 100 Alors
        Ecrire "C'est du liquide"
Finsi
Si Temp > 100 Alors
        Ecrire "C'est de la vapeur"
Finsi
Fin
```

Une autre façon d'écrire ce programme serait d'imbriquer le SI de la manière suivante

```
Variable Temp en Entier
Début
Ecrire "Entrez la température de l'eau :"
Lire Temp
Si Temp =< 0 Alors
        Ecrire "C'est de la glace"
  Sinon
Si Temp < 100 Alors
        Ecrire "C'est du liquide"
Sinon
        Ecrire "C'est de la vapeur"
Finsi
```

Énoncé des Exercices

Exercice 3.4

Ecrire un algorithme qui demande un nombre à l'utilisateur, et l'informe ensuite si ce nombre est positif ou négatif (on inclut cette fois le traitement du cas où le nombre vaut zéro).

Exercice 3.5

Ecrire un algorithme qui demande deux nombres à l'utilisateur et l'informe ensuite si le produit est négatif ou positif (on inclut cette fois le traitement du cas où le produit peut être nul). Attention toutefois, on ne doit pas calculer le produit !

Exercice 3.6

Ecrire un algorithme qui demande l'âge d'un enfant à l'utilisateur. Ensuite, il l'informe de sa catégorie :

- "Poussin" de 6 à 7 ans
- "Pupille" de 8 à 9 ans
- "Minime" de 10 à 11 ans
- "Cadet" après 12 ans

Peut-on concevoir plusieurs algorithmes équivalents menant à ce résultat ?

Corrigés des Exercices

Exercice 3.4

```
Variable n en Entier
Début
Ecrire "Entrez un nombre : "
Lire n
Si n < 0 Alors
        Ecrire "Ce nombre est négatif"
 Sinon Si n = 0 Alors
          Ecrire "Ce nombre est nul"
  Sinon Ecrire "Ce nombre est positif"
Finsi
Fin
```

Exercice 3.5

```
Variables m, n en Entier
Début
Ecrire "Entrez deux nombres : "
Lire m, n
Si m = 0 OU n = 0 Alors
        Ecrire "Le produit est nul"
 Sinon Si (m < 0 ET n < 0) OU (m > 0 ET n > 0) Alors
          Ecrire "Le produit est positif"
  Sinon
      Ecrire "Le produit est négatif"
Finsi
Fin
```

2.1.4. L'aiguillage et les variables booléennes

L'aiguillage

Dans un programme, une structure SI peut être facilement comparée à un aiguillage de train. La voie principale se sépare en deux, le train devant rouler ou sur l'une, ou sur l'autre, et les deux voies se rejoignant tôt ou tard pour ne plus en former qu'une seule, lors du FinSi.

Mais dans certains cas, ce ne sont pas deux voies qu'il nous faut, mais trois, ou même plus. Dans le cas de l'état de l'eau, il nous faut trois voies pour notre « train », puisque l'eau peut être solide, liquide ou gazeuse. Alors, nous n'avons pas eu le choix : pour deux voies, il nous fallait un aiguillage, pour trois voies il nous en faut deux, imbriqués l'un dans l'autre.

Ainsi, il est possible (mais non obligatoire, que l'algorithme initial :

```
Variable Temp en Entier
Début
0.0Ecrire "Entrez la température de l'eau :"
Lire Temp
Si Temp =< 0 Alors
        Ecrire "C'est de la glace"
   Sinon Si Temp < 100 Alors
          Ecrire "C'est du liquide"
     Sinon
                Ecrire "C'est de la vapeur"
   Finsi
Finsi
Fin
```

Devienne :

```
Variable Temp en Entier
Début
Ecrire "Entrez la température de l'eau :"
Lire Temp
Si Temp =< 0 Alors
Ecrire "C'est de la glace"
        Sinon Si Temp < 100 Alors
                Ecrire "C'est du liquide"
        Sinon
                Ecrire "C'est de la vapeur"
Finsi
Fin
```

Dans le cas de tests imbriqués, le Sinon et le Si peuvent être fusionnés en un Sinon Si. On considère alors qu'il s'agit d'un seul bloc de test, conclu par un seul FinSi

Les variables booléennes

Nous nous sommes servi des conditions pour réaliser nos tests, il existe d'autres variables du type booléen qui peuvent être susceptibles de stocker les valeurs VRAI ou FAUX. En fait, on peut donc entrer des conditions dans ces variables, et tester ensuite la valeur de ces variables.

Reprenons l'exemple de l'eau. On pourrait le réécrire ainsi :

```
Variable Temp en Entier
Variables A, B en Booléen
Début
```

Ecrire "Entrez la température de l'eau :"

```
Lire Temp
        A ←Temp =< 0
        B ←Temp < 100
Si A Alors
                Ecrire "C'est de la glace"
        Sinon Si B Alors
                Ecrire "C'est du liquide"
         Sinon
                Ecrire "C'est de la vapeur"
Finsi
Fin
```

Dans la programmation les parenthèses jouent un rôle important à savoir dans une condition composée employant à la fois des opérateurs ET et des opérateurs OU, la présence de parenthèses possède une influence sur le résultat, tout comme dans le cas d'une expression numérique comportant des multiplications et des additions.

Quand faut-il ouvrir la fenêtre de la salle ? Uniquement si les conditions l'imposent, à savoir :

```
Siil fait trop chaud ET il ne pleut pas Alors
        Ouvrir la fenêtre
Sinon
        Fermer la fenêtre
Finsi
```

Cette petite règle pourrait tout aussi bien être formulée comme suit :

```
Si il ne fait pas trop chaud OU il pleut Alors
        Fermer la fenêtre
Sinon
        Ouvrir la fenêtre
Finsi
```

Ces deux formulations sont strictement équivalentes, par exemple :

```
Si A ET B Alors
Instructions 1
Sinon
Instructions 2
Finsi
```

Équivaut à :

```
Si NON A OU NON B Alors
Instructions 2
Sinon
Instructions 1
Finsi
```

Énoncé des Exercices

Exercice 4.1

Formulez un algorithme équivalent à l'algorithme suivant :

Si Tutu > Toto + 4 OU Tata = "OK" **Alors**
Tutu ←Tutu + 1
Sinon Tutu ←Tutu – 1
Finsi

Exercice 4.2

Cet algorithme est destiné à prédire l'avenir, et il doit être infaillible !

Il lira au clavier l'heure et les minutes, et il affichera l'heure qu'il sera une minute plus tard. Par exemple, si l'utilisateur tape 21 puis 32, l'algorithme doit répondre :

"Dans une minute, il sera 21 heures(s) 33".

NB : on suppose que l'utilisateur entre une heure valide. Pas besoin donc de la vérifier.

Exercice 4.3

De même que le précédent, cet algorithme doit demander une heure et en afficher une autre. Mais cette fois, il doit gérer également les secondes, et afficher l'heure qu'il sera une seconde plus tard.

Par exemple, si l'utilisateur tape 21, puis 32, puis 8, l'algorithme doit répondre : "Dans une seconde, il sera 21 heures(s), 32 minutes(s) et 9 secondes(s)".

NB : là encore, on suppose que l'utilisateur entre une date valide.

Exercice 4.4

Un magasin de reprographie facture 0,10 E les dix premières photocopies, 0,09 E les vingt suivantes et 0,08 E au-delà. Ecrivez un algorithme qui demande à l'utilisateur le nombre de photocopies effectuées et qui affiche la facture correspondante.

Exercice 4.5

Les habitants de Zorglub paient l'impôt selon les règles suivantes :

- les hommes de plus de 20 ans paient l'impôt
- les femmes paient l'impôt si elles ont entre 18 et 35 ans
- les autres ne paient pas d'impôt

Exercice 4.6

Le programme demandera donc l'âge et le sexe du Zorglubien, et se prononcera donc ensuite sur le fait que l'habitant est imposable.

Les élections législatives, en Guignolerie Septentrionale, obéissent à la règle suivante :

- lorsque l'un des candidats obtient plus de 50% des suffrages, il est élu dès le premier tour.

- en cas de deuxième tour, peuvent participer uniquement les candidats ayant obtenu au moins 12,5% des voix au premier tour.

Vous devez écrire un algorithme qui permette la saisie des scores de quatre candidats au premier tour. Cet algorithme traitera ensuite le candidat numéro 1 (et **uniquement** lui) : il dira s'il est élu, battu, s'il se trouve en ballottage favorable (il participe au second tour en étant arrivé en tête à l'issue du premier tour) ou défavorable (il participe au second tour sans avoir été en tête au premier tour).

Exercice 4.7

Une compagnie d'assurance automobile propose à ses clients quatre familles de tarifs identifiables par une couleur, du moins au plus onéreux : tarifs bleu, vert, orange et rouge. Le tarif dépend de la situation du conducteur :

- un conducteur de moins de 25 ans et titulaire du permis depuis moins de deux ans, se voit attribuer le tarif rouge, si toutefois il n'a jamais été responsable d'accident. Sinon, la compagnie refuse de l'assurer.

- un conducteur de moins de 25 ans et titulaire du permis depuis plus de deux ans, ou de plus de 25 ans mais titulaire du permis depuis moins de deux ans a le droit au tarif orange s'il n'a jamais provoqué d'accident, au tarif rouge pour un accident, sinon il est refusé.

- un conducteur de plus de 25 ans titulaire du permis depuis plus de deux ans bénéficie du tarif vert s'il n'est à l'origine d'aucun accident et du tarif orange pour un accident, du tarif rouge pour deux accidents, et refusé au-delà

- De plus, pour encourager la fidélité des clients acceptés, la compagnie propose un contrat de la couleur immédiatement la plus avantageuse s'il est entré dans la maison depuis plus d'un an

Ecrire l'algorithme permettant de saisir les données nécessaires (sans contrôle de saisie) et de traiter ce problème. Avant de se lancer à corps perdu dans cet exercice, on pourra réfléchir un peu et s'apercevoir qu'il est plus simple qu'il n'en a l'air (cela s'appelle faire une analyse !)

Exercice 4.8

Ecrivez un algorithme qui après avoir demandé un numéro de jour, de mois et d'année à l'utilisateur, renvoie s'il s'agit ou non d'une date valide.

Cet exercice est certes d'un manque d'originalité affligeant, mais après tout, en algorithmique comme ailleurs, il faut connaître ses classiques ! Et quand on a fait cela une fois dans sa vie, on apprécie pleinement l'existence d'un type numérique « date » dans certains langages…).

Il n'est sans doute pas inutile de rappeler rapidement que le mois de février compte 28 jours, sauf si l'année est bissextile, auquel cas il en compte 29. L'année est bissextile si elle est divisible par quatre. Toutefois, les années divisibles par 100 ne sont pas bissextiles, mais les années divisibles par 400 le sont. Ouf !

Un dernier petit détail : vous ne savez pas, pour l'instant, exprimer correctement en pseudo-code l'idée qu'un nombre A est divisible par un nombre B. Aussi, vous vous contenterez d'écrire en bons télégraphistes que A divisible par B se dit « A dp B ».

Corrigés des Exercices

Exercice 4.1

Aucune difficulté, il suffit d'appliquer la règle de la transformation du OU en ET vue en cours (loi de Morgan). Attention toutefois à la rigueur dans la transformation des conditions en leur contraire...

```
Si Tutu <= Toto + 4 ET Tata <> "OK" Alors
Tutu ←Tutu – 1
Sinon
Tutu ←Tutu + 1
Finsi
```

Exercice 4.2

```
Variables h, m en Numérique

Début
Ecrire "Entrez les heures, puis les minutes : "
Lire h, m
m←m + 1
        Si m = 60 Alors
                m←0
                h←h + 1
        FinSi
        Si h >= 24 Alors
                h←0
        FinSi
Ecrire "Dans une minute il sera ", h, "heure(s) ", m, "minute(s)"
Fin
```

```
Variables h, m, s en Numérique
Début
Ecrire "Entrez les heures, puis les minutes, puis les secondes : "
Lire h, m, s
s←s + 1
Si s = 60 Alors
        s←0
        m←m + 1
FinSi
Si m = 60 Alors
        m←0
        h←h + 1
FinSi
Si h = 24 Alors
        h←0
FinSi
Ecrire "Dans une seconde il sera ", h, "h", m, "m et ", s, "s"
Fin
```

Exercice 4.4

```
Variables n, p en Numérique
Début
Ecrire "Nombre de photocopies : "
Lire n
Si n <= 10 Alors
        p←n * 0,1
Sinon Si n<= 30 Alors
        p←10 * 0,1 + (n – 10) * 0,09
Sinon
        p←10 * 0,1 + 20 * 0,09 + (n – 30) * 0,08
FinSi

Ecrire "Le prix total est: ", p
Fin
```

Exercice 4.5

```
Variable sex en Caractère
Variable age en Numérique
Variables C1, C2 en Booléen
Début
Ecrire "Entrez le sexe (M/F) : "
Lire sex
Ecrire "Entrez l'âge: "
Lire age
C1 ←sex = "M" ET age> 20
C2 ←sex = "F" ET (age> 18 ET age< 35)
Si C1 ou C2 Alors
Ecrire "Imposable"
Sinon Ecrire "Non Imposable"
FinSi
```

Exercice 4.6

Cet exercice, du pur point de vue algorithmique, n'est pas très méchant. En revanche, il représente dignement la catégorie des énoncés piégés.

En effet, rien de plus facile que d'écrire : si le candidat a plus de 50%, il est élu, sinon s'il a plus de 12,5 %, il est au deuxième tour, sinon il est éliminé. Hé héhé... mais il ne faut pas oublier que le candidat peut très bien avoir eu 20 % mais être tout de même éliminé, tout simplement parce que l'un des autres a fait plus de 50 % et donc qu'il n'y a pas de deuxième tour !...

Moralité : ne jamais se jeter sur la programmation avant d'avoir soigneusement mené l'analyse du problème à traiter.

```
Variables A, B, C, D en Numérique
Début
Ecrire "Entrez les scores des quatre prétendants :"
Lire A, B, C, D
C1 ←A > 50
```

C2 ←B > 50 ou C > 50 ou D > 50

```
C3 ←A >= B et A >= C et A >= D
C4 ←A >= 12,5
Si C1 Alors
        Ecrire "Elu au premier tour"
  SinonsiC2 ou Non(C4) Alors
        Ecrire "Battu, éliminé, sorti !!!"
   Sinon Si C3 Alors
        Ecrire "Ballotage favorable"
    Sinon Ecrire "Ballotage défavorable"
FinSi
Fin
```

Exercice 4.7

Là encore, on illustre l'utilité d'une bonne analyse. Je propose deux corrigés différents. Le premier suit l'énoncé pas à pas. C'est juste, mais c'est vraiment lourd. La deuxième version s'appuie sur une vraie compréhension d'une situation pas si embrouillée qu'elle n'en a l'air.

Dans les deux cas, un recours aux variables booléennes aère sérieusement l'écriture.

Donc, premier corrigé, on suit le texte de l'énoncé pas à pas :

```
Variables age, perm, acc, assur en Numérique
Variables C1, C2, C3 en Booléen
Variable situ en Caractère
Début
Ecrire "Entrez l'âge: "
Lire age
Ecrire "Entrez le nombre d'années de permis: "
Lire perm
Ecrire "Entrez le nombre d'accidents : "
Lire acc
Ecrire "Entrez le nombre d'années d'assurance : "
Lire assur
C1 ← age>= 25
C2 ← perm >= 2
C3 ← assur> 1
Si Non(C1) et Non(C2) Alors
        Si acc = 0 Alors
                situ← "Rouge"
Sinon
                situ← "Refusé"
FinSi

Sinon si ((Non(C1) et C2) ou (C1 et Non(C2)) Alors
        Si acc = 0 Alors
                situ← "Orange"
         Sinon Si acc = 1 Alors
                situ← "Rouge"
         FinSi   Sinon
```

situ← "Refusé"

```
Sinon
Si acc = 0 Alors

                situ← "Vert"
Sinon Si acc = 1 Alors
                situ← "Orange"
Sinon Si acc = 2 Alors
                situ← "Rouge"
Sinon
                situ← "Refusé"
FinSi
FinSi
Si C3 Alors

 Si situ = "Rouge" Alors
        situ← "Orange"
Sinon Si situ = "Orange" Alors
        situ← "Orange"
Sinon Si situ = "Vert" Alors
        situ← "Bleu"
FinSi
FinSi
Ecrire "Votre situation : ", situ
Fin
```

Vous trouvez cela compliqué ? Oh, certes oui, ça l'est ! Et d'autant plus qu'en lisant entre les lignes, on pouvait s'apercevoir que ce galimatias de tarifs recouvre en fait une logique très simple : un système à points. Et il suffit de comptabiliser les points pour que tout s'éclaire... Reprenons juste après l'affectation des trois variables booléennes C1, C2, et C3. On écrit :

```
P ←0
Si Non(C1) Alors
P ←P + 1
FinSi
Si Non(C2) Alors
P ←P + 1
FinSi
P ←P + acc
Si P < 3 et C3 Alors
P ←P – 1
FinSi
Si P = -1 Alors
situ←"Bleu"
SinonSiP = 0 Alors

situ←"Vert"
SinonSiP = 1 Alors
situ←"Orange"
SinonSiP = 2 Alors
```

situ←"Rouge"

Sinon
situ←"Refusé"
FinSi
Ecrire "Votre situation : ", situ
Fin

Exercice 4.8

En ce qui concerne le début de cet algorithme, il n'y a aucune difficulté. C'est de la saisie bête et même pas méchante:

Variables J, M, A, JMax**en Numérique**
Variables VJ, VM, B **en Booleen**
Début
Ecrire "Entrez le numéro du jour"
Lire J
Ecrire "Entrez le numéro du mois"
Lire M
Ecrire "Entrez l'année"
Lire A

C'est évidemment ensuite que les ennuis commencent… La première manière d'aborder la chose consiste à se dire que fondamentalement, la structure logique de ce problème est très simple. Si nous créons deux variables booléennes VJ et VM, représentant respectivement la validité du jour et du mois entrés, la fin de l'algorithme sera d'une simplicité biblique (l'année est valide par définition, si on évacue le débat byzantin concernant l'existence de l'année zéro) :

Si VJ et VM **alors**
Ecrire "La date est valide"
Sinon
Ecrire "La date n'est pas valide"
FinSi

Toute la difficulté consiste à affecter correctement les variables VJ et VM, selon les valeurs des variables J, M et A. Dans l'absolu, VJ et VM pourraient être les objets d'une affectation monstrueuse, avec des conditions atrocement composées. Mais franchement, écrire ces conditions en une seule fois est un travail de bénédictin sans grand intérêt. Pour éviter d'en arriver à une telle extrémité, on peut sérier la difficulté en créant deux variables supplémentaires :

B : variable booléenne qui indique s'il s'agit d'une année bissextile

JMax: variable numérique qui indiquera le dernier jour valable pour le mois entré.

Avec tout cela, on peut y aller et en ressortir vivant.

On commence par initialiser nos variables booléennes, puis on traite les années, puis les mois, puis les jours.

On note "dp" la condition "divisible par" :

B ← A dp 400 ou (non(A dp 100) et A dp 4)
Jmax← 0
VM ← M >= 1 et M =< 12

```
Si VM Alors
Si M = 2 et B Alors
JMax← 29
SinonSiM = 2 Alors
JMax← 28
SinonSiM = 4 ou M = 6 ou M = 9 ou M = 11 Alors
        JMax← 30
Sinon
        JMax← 31
FinSi

VJ ← J >= 1 et J =<Jmax
FinSi
```

Cette solution a le mérite de ne pas trop compliquer la structure des tests, et notamment de ne pas répéter l'écriture finale à l'écran. Les variables booléennes intermédiaires nous épargnent des conditions composées trop lourdes, mais celles-ci restent néanmoins sérieuses.

Une approche différente consisterait à limiter les conditions composées, quitte à le payer par une structure beaucoup plus exigeante de tests imbriqués. Là encore, on évite de jouer les extrémistes et l'on s'autorise quelques conditions composées lorsque cela nous simplifie l'existence. On pourrait aussi dire que la solution précédente "part de la fin" du problème (la date est elle valide ou non ?), alors que celle qui suit "part du début" (quelles sont les données entrées au clavier ?) :

```
Si M < 1 ou M > 12 Alors
Ecrire "Date Invalide"
SinonSiM = 2 Alors
Si A dp 400 Alors
Si J < 1 ou J > 29 Alors
Ecrire "Date Invalide"
Sinon
Ecrire "Date Valide"
FinSi
SinonSiA dp 100 Alors
Si J < 1 ou J > 28 Alors
Ecrire "Date Invalide"

Sinon
Ecrire "Date Valide"
FinSi
SinonSiAdp 4 Alors
Si J < 1 ou J > 28 Alors
Ecrire "Date Invalide"
Sinon
Ecrire "Date Valide"
FinSi
Sinon
Si J < 1 ou J > 28 Alors
Ecrire "Date Invalide"
Sinon
```

Ecrire "Date Valide"
FinSi
FinSi
SinonSiM = 4 ou M = 6 ou M = 9 ou M = 11 **Alors**
Si J < 1 ou J > 30 **Alors**
Ecrire "Date Invalide"
Sinon
Ecrire "Date Valide"
FinSi
Sinon
Si J < 1 ou J > 31 **Alors**
Ecrire "Date Invalide"
Sinon
Ecrire "Date Valide"
FinSi
FinSi

B ← (A dp 4 et Non(A dp 100)) ou A dp 400
K1 ← (m=1 ou m=3 ou m=5 ou m=7 ou m=8 ou m=10 ou m=12) et (J>=1 et J=<31)
K2 ← (m=4 ou m=6 ou m=9 ou m=11) et (J>=1 et J=<30)
K3 ← m=2 et B et J>=1 et J=<29
K4 ← m=2 et J>=1 et J=<28
Si K1 ou K2 ou K3 ou K4 **Alors**
Ecrire "Date valide"
Sinon
Ecrire "Date non valide"
FinSi
Fin

2.2. Les boucles

Une boucle permet de parcourir une partie d'un programme un certain nombre de fois. Une itération est la répétition d'un même traitement plusieurs fois. Un indice de boucle varie alors de la valeur minimum (initiale) jusqu'à la valeur maximum (finale).

2.2.1. La boucle Pour

Cette structure est une BOUCLE ITERATIVE ; elle consiste à répéter un certain traitement un nombre de fois fixé à l'avance
Cette structure utilise une variable (indice) de contrôle d'itérations caractérisée par :

- ✓ Sa valeur initiale,
- ✓ Sa valeur finale,
- ✓ Son pas de variation.

La syntaxe de la boucle pour est comme suit :

Pour I de 1 jusqu'à N pas 1
    Instructions

Fin Pour

I : variable
1 : valeur initiale
N : valeur finale
Le pas de variation égale à 1

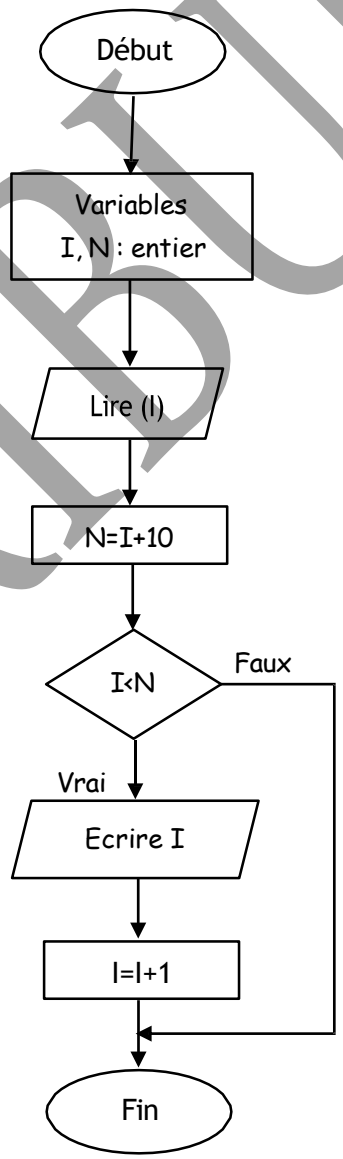
Exemple :

Ecrire un algorithme qui demande un nombre de départ, et qui ensuite affiche les dix nombres suivants. Par exemple, si l'utilisateur entre le nombre 17, le programme affichera les nombres de 18 à 27.

Solution :

Algorithme :

```
Algorithme nombres
Début
Variables I, N : entier
Écrire ("entrer la valeur de I ")
Lire (I)
N=I+10
        Pour I de 17 jusqu'à N pas 1
                Ecrire (I)
        Fin Pour
Fin
```



3.2.2. La boucle tant que

Une action ou un groupe d'actions est exécuté répétitivement tout le temps où une condition est vraie.

Syntaxe

Tant Que (Condition) Faire
Instructions
Fin Tant que

Remarque : la vérification de la condition s'effectue avant les actions. Celles-ci peuvent donc ne jamais être exécutées.

Exemple :
On veut écrire un algorithme qui calcul la somme des entiers positifs inférieurs ou égaux à N.

```
Algorithme somme
Début
Variables I, N, S : entier
Écrire ("entrer la valeur de N ")
Lire (N)
I=0
S=0
Tant Que (I ≤ N) alors
        S=S+I
        I=I+1
Fin Tant Que
Ecrire ('la Somme est', S)
Fin
```

2.2.3. La boucle REPETER ... JUSQUA ...

Une action ou un groupe d'actions est exécuté répétitivement jusqu'à ce qu'une condition soit vérifiée.

Syntaxe
Répéter
Instructions
Jusqu'à (Condition)

Remarque : la vérification de la condition s'effectue après les actions. Celles-ci sont donc exécutées au moins une fois.

Exemple :
Ecrire un algorithme qui demande successivement 10 nombres à l'utilisateur, et qui lui dise ensuite quel était le plus grand parmi ces 10 nombres :
Entrez le nombre numéro 1 : 7
Entrez le nombre numéro 2 : 24
.....etc.
Entrez le nombre numéro 10 : 13
Le plus grand de ces nombres est : 24

Solution :
Algorithme :

```
Algorithme plus_grand
Début
Variables I, N, PG : entier
```

```
I=1
Écrire ("entrer la valeur de N ")
Lire (N)
PG = N
Répéter
        Lire (N)
Si (PG ≤ N) alors
        PG = N
Fin Si
I=I+1
Jusqu'à (I = 10)
        Ecrire ('La valeur la plus grand est', PG)
Fin
```

Énoncé des Exercices

Exercice 5.4

Ecrire un algorithme qui demande un nombre de départ, et qui ensuite écrit la table de multiplication de ce nombre, présentée comme suit (cas où l'utilisateur entre le nombre 7) :

Table de 7 :
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
…
7 x 10 = 70

Exercice 5.5

Ecrire un algorithme qui demande un nombre de départ, et qui calcule la somme des entiers jusqu'à ce nombre. Par exemple, si l'on entre 5, le programme doit calculer :

1 + 2 + 3 + 4 + 5 = 15

NB : on souhaite afficher uniquement le résultat, pas la décomposition du calcul.

Exercice 5.6

Ecrire un algorithme qui demande un nombre de départ, et qui calcule sa factorielle.

NB : la factorielle de 8, notée 8 !, vaut

1 x 2 x 3 x 4 x 5 x 6 x 7 x 8

Exercice 5.7

Ecrire un algorithme qui demande successivement 20 nombres à l'utilisateur, et qui lui dise ensuite quel était le plus grand parmi ces 20 nombres :

Entrez le nombre numéro 1 : 12
Entrez le nombre numéro 2 : 14

etc.

Entrez le nombre numéro 20 : 6
Le plus grand de ces nombres est : 14

Modifiez ensuite l'algorithme pour que le programme affiche de surcroît en quelle position avait été saisie ce nombre :

C'était le nombre numéro 2

Exercice 5.8

Réécrire l'algorithme précédent, mais cette fois-ci on ne connaît pas d'avance combien l'utilisateur souhaite saisir de nombres. La saisie des nombres s'arrête lorsque l'utilisateur entre un zéro.

Exercice 5.9

Lire la suite des prix (en euros entiers et terminée par zéro) des achats d'un client. Calculer la somme qu'il doit, lire la somme qu'il paye, et simuler la remise de la monnaie en affichant les textes "10 Euros", "5 Euros" et "1 Euro" autant de fois qu'il y a de coupures de chaque sorte à rendre.

Exercice 5.10

Écrire un algorithme qui permette de connaître ses chances de gagner au tiercé, quarté, quinté et autres impôts volontaires.

On demande à l'utilisateur le nombre de chevaux partants, et le nombre de chevaux joués. Les deux messages affichés devront être :

Dans l'ordre : une chance sur X de gagner

Dans le désordre : une chance sur Y de gagner

X et Y nous sont donnés par la formule suivante, si n est le nombre de chevaux partants et p le nombre de chevaux joués (on rappelle que le signe ! signifie "factorielle", comme dans l'exercice 5.6 ci-dessus) :

X = n ! / (n - p) !
Y = n ! / (p ! * (n – p) !)

NB : cet algorithme peut être écrit d'une manière simple, mais relativement peu performante. Ses performances peuvent être singulièrement augmentées par une petite astuce. Vous commencerez par écrire la manière la plus simple, puis vous identifierez le problème, et écrirez une deuxième version permettant de le résoudre.

Corrigés des Exercices

Exercice 5.4

Variables N, i **en Entier**
Debut
Ecrire "Entrez un nombre : "
Lire N
Ecrire "La table de multiplication de ce nombre est : "
Pour i ← 1 à 10
Ecrire N, " x ", i, " = ", n*i
i**Suivant**

Fin

Exercice 5.5

```
Variables N, i, Som en Entier
Debut
Ecrire "Entrez un nombre : "
Lire   N
Som← 0
Pour i ← 1 à N
Som← Som + i
iSuivant
Ecrire "La somme est : ", Som
Fin
```

Exercice 5.6

```
Variables N, i, F en Entier
Debut
Ecrire "Entrez un nombre : "
Lire N
F ← 1
Pour i ← 2 à N
F ← F * i
iSuivant
Ecrire "La factorielle est : ", F
Fin
```

Exercice 5.7

```
Variables N, i, PG en Entier
Debut
PG ← 0
Pour i ← 1 à 20
Ecrire "Entrez un nombre : "
Lire N
Si i = 1 ou N > PG Alors
PG ← N
FinSi
iSuivant
Ecrire "Le nombre le plus grand était : ", PG
Fin
```

En ligne 3, on peut mettre n'importe quoi dans PG, il suffit que cette variable soit affectée pour que le premier passage en ligne 7 ne provoque pas d'erreur.
Pour la version améliorée, cela donne :

```
Variables N, i, PG, IPG en Entier
Debut
PG ←0
Pour i ←1 à 20
Ecrire "Entrez un nombre : "
```

```
Lire N
Si i = 1 ou N > PG Alors
PG ←N
IPG ←i
FinSi
iSuivant
Ecrire "Le nombre le plus grand était : ", PG
Ecrire "Il a été saisi en position numéro ", IPG
Fin
```

Exercice 5.8

```
Variables N, i, PG, IPG en Entier
Debut
N ←1
i←0
PG ←0
TantQueN <> 0
Ecrire "Entrez un nombre : "
Lire N
i←i + 1
Si i = 1 ou N > PG Alors
PG ←N
IPG ←i
FinSi
FinTantQue
Ecrire "Le nombre le plus grand était : ", PG
Ecrire "Il a été saisi en position numéro ", IPG
Fin
```

Exercice 5.9

```
Variables FF, somdue, M, IPG, Reste, Nb10F, Nb5F En Entier
Debut
E ←1
somdue←0
TantQueE <> 0
Ecrire "Entrez le montant : "
Lire E
somdue←somdue + E
FinTantQue
Ecrire "Vous devez :", E, " euros"
Ecrire "Montant versé :"
Lire M
Reste ←M – E
Nb10E ←0
TantQueReste >= 10
Nb10E ←Nb10E + 1
Reste ←Reste – 10
FinTantQue
Nb5E ←0
Si Reste >= 5
```

Nb5E ←1
Reste ←Reste – 5
FinSi
Ecrire "Rendu de la monnaie :"
Ecrire "Billets de 10 E : ", Nb10E
Ecrire "Billets de 5 E : ", Nb5E
Ecrire "Pièces de 1 E : ", reste
Fin

Exercice 5.10

Spontanément, on est tenté d'écrire l'algorithme suivant :

Variables N, P, i, Numé, Déno1, Déno2 en Entier
Debut **Ecrire** "Entrez le nombre de chevaux partants : "
Lire N
Ecrire "Entrez le nombre de chevaux joués : "
Lire P
Numé←1
Pour i ←2 à N
Numé←Numé * i
i**Suivant**
Déno1 ←1
Pour i ←2 à N-P
Déno1 ←Déno1 * i
i**Suivant**
Déno2 ←1
Pour i ←2 à P
Déno2 ←Déno2 * i
i**Suivant**
Ecrire "Dans l'ordre, une chance sur ", Numé / Déno1
Ecrire "Dans le désordre, une sur ", Numé / (Déno1 * Déno2)
Fin

Cette version, formellement juste, comporte tout de même deux faiblesses.

La première, et la plus grave, concerne la manière dont elle calcule le résultat final. Celui-ci est le quotient d'un nombre par un autre ; or, ces nombres auront rapidement tendance à être très grands. En calculant, comme on le fait ici, d'abord le numérateur, puis ensuite le dénominateur, on prend le risque de demander à la machine de stocker des nombres trop grands pour qu'elle soit capable de les coder (cf. le préambule). C'est d'autant plus bête que rien ne nous oblige à procéder ainsi : on n'est pas obligé de passer par la division de deux très grands nombres pour obtenir le résultat voulu.

La deuxième remarque est qu'on a programmé ici trois boucles successives. Or, en y regardant bien, on peut voir qu'après simplification de la formule, ces trois boucles comportent le même nombre de tours ! (Si vous ne me croyez pas, écrivez un exemple de calcul et biffez les nombres identiques au numérateur et au dénominateur). Ce triple calcul (ces trois boucles) peut donc être ramené(es) à un(e) seul(e).

Et voilà le travail, qui est non seulement bien plus court, mais aussi plus performant :

Variables N, P, i, O, F **en Entier**

```
Debut
Ecrire "Entrez le nombre de chevaux partants : "
Lire N
Ecrire "Entrez le nombre de chevaux joués : "
Lire P
A ←1
B ←1
Pour i ←1 à P
A ←A * (i + N - P)
B ←B * i
iSuivant
Ecrire "Dans l'ordre, une chance sur ", A
Ecrire "Dans le désordre, une chance sur ", A / B
Fin
```

CHAP3 : LES TABLEAUX

3.1. Utilité des tableaux

Imaginons que dans un programme, nous ayons besoin simultanément de 12 valeurs (par exemple, des notes pour calculer une moyenne). Evidemment, la seule solution dont nous disposons à l'heure actuelle consiste à déclarer douze variables, appelées par exemple Notea, Noteb, Notec, etc. Bien sûr, on peut opter pour une notation un peu simplifiée, par exemple N1, N2, N3, etc. Mais cela ne change pas fondamentalement notre problème, car arrivé au calcul, et après une succession de douze instructions « Lire » distinctes, cela donnera obligatoirement une atrocité du genre :

Moy← (N1+N2+N3+N4+N5+N6+N7+N8+N9+N10+N11+N12) /12

Ouf ! C'est tout de même bigrement laborieux. Et pour un peu que nous soyons dans un programme de gestion avec quelques centaines ou quelques milliers de valeurs à traiter, alors là c'est le suicide direct.

Cerise sur le gâteau, si en plus on est dans une situation on l'on ne peut pas savoir d'avance combien il y aura de valeurs à traiter, là on est carrément cuits.

C'est pourquoi la programmation nous permet de rassembler toutes ces variables en une seule, au sein de laquelle chaque valeur sera désignée par un numéro. En bon français, cela donnerait donc quelque chose du genre « la note numéro 1 », « la note numéro 2 », « la note numéro 8 ». C'est largement plus pratique, vous vous en doutez.

3.2. Notation et utilisation algorithmique

Dans notre exemple, nous créerons donc un tableau appelé Note. Chaque note individuelle (chaque élément du tableau Note) sera donc désignée Note (0), Note (1), etc. Eh oui, attention, les indices des tableaux commencent généralement à 0, et non à 1.

Un tableau doit être déclaré comme tel, en précisant le nombre et le type de valeurs qu'il contiendra (la déclaration des tableaux est susceptible de varier d'un langage à l'autre. Certains langages réclament le nombre d'éléments, d'autre le plus grand indice... C'est donc une affaire de conventions).

En nous calquant sur les choix les plus fréquents dans les langages de programmations, nous déciderons ici arbitrairement et une bonne fois pour toutes que :

- Les "cases" sont numérotées à partir de zéro, autrement dit que le plus petit indice est zéro.
- Lors de la déclaration d'un tableau, on précise la plus grande valeur de l'indice (différente, donc, du nombre de cases du tableau, puisque si on veut 12 emplacements, le plus grand indice sera 11). Au début, ça déroute, mais vous verrez, avec le temps, on se fait à tout, même au pire.

Tableau Note (11) en Entier

On peut créer des tableaux contenant des variables de tous types : tableaux de numériques, bien sûr, mais aussi tableaux de caractères, tableaux de booléens, tableaux de tout ce qui existe dans un langage donné comme type de variables. Par contre, hormis dans quelques rares langages, on ne peut pas faire un mixage de types différents de valeurs au sein d'un même tableau

L'énorme avantage des tableaux, c'est qu'on va pouvoir les traiter en faisant des boucles. Par exemple, pour effectuer notre calcul de moyenne, cela donnera par exemple :

```
Tableau Note(11) en Numérique
Variables Moy, Somen Numérique
Début
Pour i ← 0 à 11
Ecrire "Entrez la note n°", i
Lire Note(i)
iSuivant
        Som← 0
Pour i ← 0 à 11
Som← Som + Note(i)
iSuivant
Moy← Som / 12
Fin
```

NB : On a fait deux boucles successives pour plus de lisibilité, mais on aurait tout aussi bien pu n'en écrire qu'une seule dans laquelle on aurait tout fait d'un seul coup.

Remarques

L'indice qui sert à désigner les éléments d'un tableau peut être exprimé directement comme un nombre en clair, mais il peut être aussi une variable, ou une expression calculée.

Dans un tableau, la valeur d'un indice doit toujours :

- **être égale au moins à 0** (dans quelques rares langages, le premier élément d'un tableau porte l'indice 1). Mais comme je l'ai déjà écrit plus haut, nous avons choisi ici de commencer la numérotation des indices à zéro, comme c'est le cas en langage C et en Visual Basic. Donc attention, Truc (6) est le septième élément du tableau Truc !
- **être un nombre entier** Quel que soit le langage, l'élément Truc (3,1416) n'existe jamais.
- **être inférieure ou égale au nombre d'éléments du tableau** (moins 1, si l'on commence la numérotation à zéro). Si le tableau Bidule a été déclaré comme ayant 25 éléments, la présence dans une ligne, sous une forme ou sous une autre, de Bidule (32) déclenchera automatiquement une erreur.

Je le re-re-répète, si l'on est dans un langage où les indices commencent à zéro, il faut en tenir compte à la déclaration :

Tableau Note (13) **en Numérique**

Énoncé des Exercices

Exercice 6.1

Ecrire un algorithme qui déclare et remplisse un tableau de 7 valeurs numériques en les mettant toutes à zéro.

Exercice 6.2

Ecrire un algorithme qui déclare et remplisse un tableau contenant les six voyelles de l'alphabet latin.

Exercice 6.3

Ecrire un algorithme qui déclare un tableau de 9 notes, dont on fait ensuite saisir les valeurs par l'utilisateur.

Exercice 6.4

Que produit l'algorithme suivant ?

```
Tableau Nb(5) en Entier
Variable i en Entier
Début
Pour i ← 0 à 5
        Nb(i) ← i * i
isuivant
Pour i ← 0 à 5
        Ecrire Nb(i)
isuivant
Fin
```

Peut-on simplifier cet algorithme avec le même résultat ?

Exercice 6.5

Que produit l'algorithme suivant ?

```
Tableau N(6) en Entier
Variables i, k en Entier
Début
N(0) ←1
Pour k ←1 à 6
        N(k) ←N(k-1) + 2
kSuivant
Pour i ←0 à 6
        Ecrire N(i)
isuivant
Fin
```

Peut-on simplifier cet algorithme avec le même résultat ?

Exercice 6.6

Que produit l'algorithme suivant ?

```
Tableau Suite(7) en Entier
Variable i en Entier
Début
        Suite(0) ←1
        Suite(1) ←1
        Pour i ←2 à 7
        Suite(i) ←Suite(i-1) + Suite(i-2)
isuivant
Pour i ←0 à 7
Ecrire Suite(i)
isuivant
Fin
```

Corrigés des Exercices

Exercice 6.1

```
Tableau Truc(6) en Numérique
Variable i en Numérique
Debut
        Pour i ←0 à 6
        Truc(i) ←0
iSuivant
Fin
```

Exercice 6.2

```
Tableau Truc(5) en Caractère
Debut
Truc(0) ←"a"
Truc(1) ←"e"
Truc(2) ←"i"
Truc(3) ←"o"
Truc(4) ←"u"
Truc(5) ←"y"
Fin
```

Exercice 6.3

```
Tableau Notes(8) en Numérique
Variable i en Numérique
Pour i ←0 à 8
Ecrire "Entrez la note numéro ", i + 1
Lire Notes(i)
iSuivant
Fin
```

Exercice 6.4

Cet algorithme remplit un tableau avec six valeurs : 0, 1, 4, 9, 16, 25.

Il les écrit ensuite à l'écran. Simplification :

```
Tableau Nb(5) en Numérique
Variable i en Numérique
Début
Pour i ←0 à 5
Nb(i) ←i * i
Ecrire Nb(i)
iSuivant
Fin
```

Exercice 6.5

Cet algorithme remplit un tableau avec les sept valeurs : 1, 3, 5, 7, 9, 11, 13.

Il les écrit ensuite à l'écran. Simplification :

```
Tableau N (6) en Numérique
Variables i, k en Numérique
Début
N (0) ←1
Ecrire N(0)
Pour k ←1 à 6
N(k) ←N(k-1) + 2
Ecrire N(k)
kSuivant
Fin
```

Exercice 6.6

Cet algorithme remplit un tableau de 8 valeurs : 1, 1, 2, 3, 5, 8, 13, 21

Exercice 6.7

```
Variable S en Numérique
Tableau Notes(8) en Numérique
Debut
s←0
Pour i ←0 à 8
Ecrire "Entrez la note n° ", i + 1
Lire Notes(i)
s←s + Notes(i)
iSuivant
Ecrire "Moyenne :", s/9
Fin
```

3.3. Tableau dynamique

Il arrive fréquemment que l'on ne connaisse pas à l'avance le nombre d'éléments que devra comporter un tableau. Bien sûr, une solution consisterait à déclarer un tableau gigantesque (10 000 éléments, pourquoi pas, au diable les varices) pour être sûr que « ça rentre ». Mais d'une part, on n'en sera jamais parfaitement sûr, d'autre part, en raison de l'immensité de la place mémoire

réservée – et la plupart du temps non utilisée, c'est un gâchis préjudiciable à la rapidité, voire à la viabilité, de notre algorithme.

Aussi, pour parer à ce genre de situation, a-t-on la possibilité de déclarer le tableau sans préciser au départ son nombre d'éléments. Ce n'est que dans un second temps, au cours du programme, que l'on va fixer ce nombre via une instruction de redimensionnement : **Redim**.

Notez que **tant qu'on n'a pas précisé le nombre d'éléments d'un tableau, d'une manière ou d'une autre, ce tableau est inutilisable**.

Exemple : on veut faire saisir des notes pour un calcul de moyenne, mais on ne sait pas combien il y aura de notes à saisir. Le début de l'algorithme sera quelque chose du genre :

```
Tableau Notes ()en Numérique
Variable nb en Numérique
Début
Ecrire "Combien y a-t-il de notes à saisir ?"
Lire nb
RedimNotes(nb-1)
…
```

Cette technique n'a rien de sorcier, mais elle fait partie de l'arsenal de base de la programmation en gestion.

Énoncé des Exercices

Exercice 6.8

Ecrivez un algorithme permettant à l'utilisateur de saisir un nombre quelconque de valeurs, qui devront être stockées dans un tableau. L'utilisateur doit donc commencer par entrer le nombre de valeurs qu'il compte saisir. Il effectuera ensuite cette saisie. Enfin, une fois la saisie terminée, le programme affichera le nombre de valeurs négatives et le nombre de valeurs positives.

Exercice 6.9

Ecrivez un algorithme calculant la somme des valeurs d'un tableau (on suppose que le tableau a été préalablement saisi).

Corrigés des Exercices

Exercice 6.8

```
Variables Nb, Nbpos, Nbneg en Numérique
Tableau T() en Numérique
Debut
Ecrire "Entrez le nombre de valeurs :"
Lire Nb
Redim T(Nb-1)
Nbpos ← 0
Nbneg ← 0
Pour i ← 0 à Nb – 1
Ecrire "Entrez le nombre n° ", i + 1
Lire T(i)
```

```
Si T(i) > 0 alors
Nbpos ← Nbpos + 1
Sinon
Nbneg ← Nbneg + 1
Finsi
i Suivant
Ecrire "Nombre de valeurs positives : ", Nbpos
Ecrire "Nombre de valeurs négatives : ", Nbneg
Fin
```

Exercice 6.9

```
Variables i, Som, N en Numérique
Tableau T() en Numérique
Debut
… (on ne programme pas la saisie du tableau, dont on suppose qu'il compte N éléments)
Redim T(N-1)
…
Som ← 0
Pour i ← 0 à N – 1
Som ← Som + T(i)
i Suivant
Ecrire "Somme des éléments du tableau : ", Som
Fin
```

1. Tri dans un tableau

Il existe plusieurs stratégies possibles pour trier les éléments d'un tableau ; nous en verrons deux : le tri par sélection, et le tri à bulles.

- *Tri par sélection.*

Admettons que le but de la manœuvre soit de trier un tableau de 12 éléments dans l'ordre croissant. La technique du tri par sélection est la suivante : on met en bonne position l'élément numéro 1, c'est-à-dire le plus petit. Puis en met en bonne position l'élément suivant. Et ainsi de suite jusqu'au dernier. Par exemple, si l'on part de :

45	***122***	***12***	***3***	***21***	***78***	***64***	***53***	***89***	***28***	***84***	***46***

On commence par rechercher, parmi les 12 valeurs, quel est le plus petit élément , et où il se trouve. On l'identifie en quatrième position (c'est le nombre 3), et on l'échange alors avec le premier élément (le nombre 45). Le tableau devient ainsi :

3	***122***	***12***	***45***	***21***	***78***	***64***	***53***	***89***	***28***	***84***	***46***

On recommence à chercher le plus petit élément, mais cette fois, **seulement à partir du deuxième** (puisque le premier est maintenant correct, on n'y touche plus). On le trouve en troisième position (c'est le nombre 12). On échange donc le deuxième avec le troisième :

3	***12***	***122***	***45***	***21***	***78***	***64***	***53***	***89***	***28***	***84***	***46***

On recommence à chercher le plus petit élément à partir du troisième (puisque les deux premiers sont maintenant bien placés), et on le place correctement, en l'échangeant, ce qui donnera in fine:

3	***12***	***21***	***45***	***122***	***78***	***64***	***53***	***89***	***28***	***84***	***46***

Le processus de la manière suivante :

- Boucle principale : prenons comme point de départ le premier élément, puis le second, etc, jusqu'à l'avant dernier.
- Boucle secondaire : à partir de ce point de départ mouvant, recherchons jusqu'à la fin du tableau quel et le plus petit élément. Une fois que nous l'avons trouvé, nous l'échangeons avec le point de départ.

Cela s'écrit :

boucle principale : le point de départ se décale à chaque tour
Pour i ← 0 à 10
    on considère provisoirement que t(i) est le plus petit élément
posmini ← i
    on examine tous les éléments suivants
Pour j ← i + 1 à 11
    Si t(j) < t(posmini) **Alors**
        posmini ← j
Finsi
j **suivant**
A cet endroit, on sait maintenant où est le plus petit élément. Il ne reste plus qu'à effectuer la permutation.
temp ← t(posmini)
t(posmini) ← t(i)
t(i) ← temp
    On a placé correctement l'élément numéro i, on passe à présent au suivant.
i suivant

2. *Recherche dans un Tableau*

a. Recherche linéaire

Soit un tableau comportant, disons, 20 valeurs. L'on doit écrire un algorithme saisissant un nombre au clavier, et qui informe l'utilisateur de la présence ou de l'absence de la valeur saisie dans le tableau.

```
Tableau Tab(19) en Numerique
Variable N en Numerique
Debut
Ecrire "Entrez la valeur a rechercher"
Lire N
Trouve ← Faux
Pour i ← 0 a 19
Si N = Tab(i) Alors
Trouve ← Vrai
FinSi
i suivant
Si Trouve Alors
Ecrire "N fait partie du tableau"
Sinon
Ecrire "N ne fait pas partie du tableau"
FinSi
Fin
```

La difficulté est de comprendre que dans une recherche, le problème ne se formule pas de la même manière selon qu'on le prend par un bout ou par un autre. On peut résumer l'affaire ainsi : il suffit que N soit égal à une seule valeur de Tab pour qu'elle fasse partie du tableau. En revanche, il faut qu'elle soit différente de toutes les valeurs de Tab pour qu'elle n'en fasse pas partie.
Voilà la raison qui nous oblige à passer par une variable booléenne. Et cette technique de flag (que nous pourrions élégamment surnommer « gestion asymétrique de variable booléenne ») doit être mise en œuvre chaque fois que l'on se trouve devant pareille situation.
Autrement dit, connaître ce type de raisonnement est indispensable, et savoir le reproduire à bon escient ne l'est pas moins.

b. Recherche de l'élément maximum et de l'élément minimum dans un tableau linéaire

Soit un tableau Temp contenant 12 températures dans un tableau de single. Lorsqu'on veut chercher la température maximale et minimale dans ce tableau de 12 températures, on procédera comme suit :

- **Recherche de l'élément maximal**

On doit parcourir le tableau et de conserver la valeur maximale dans une variable dite temporaire « TempMax ».
Au départ, on suppose que la température maximale est la première du tableau.

```
Var TempMax : réel
TempMax = Temp (1)
Pour i ← 2 à 12
Si Temp (i) > TempMax alors TempMax ← Temp (i)
Fin Pour i
```

A la fin du processus, la variable TEMPMAX contiendra la valeur recherchée.

- **Recherche de l'élément minimal**

Mêmement qu'à la recherche de l'élément maximal, mais ici, il suffit de conserver la valeur minimale dans une variable dite temporaire « TempMin ».

Au départ, on suppose que la température maximale est la première du tableau.

```
Var TempMin : réel
TempMin = Temp (1)
Pour i ← 2 à 12
Si Temp (i) < TempMin alors TempMin ← Temp (i)
Fin Pour i
```

A la fin du processus, la variable TEMPMin contiendra la valeur recherchée.

c. Recherche dichotomique

Lorsqu'on veut trier un élément dans l'ordre croissant et retourner l'indice d'une occurrence de l'élément cherché, on procède à la recherche dichotomique. Une telle recherche peut être réalisée de manière séquentielle ou dichotomique. On développe ici la version dichotomique qui est la plus efficace en temps d'exécution.

Principe : On effectue la comparaison de l'élément cherché par rapport à celui qui se trouve au milieu du tableau. Si l'élément cherché est plus petit, on continue la recherche dans la première moitié du tableau sinon dans la seconde. On recommence ce processus sur la moitié. On s'arrête lorsqu'on a trouvé ou lorsque l'intervalle de recherche est nul.

Exemple

Recherche dans le tableau d'entiers suivant défini sur l'intervalle [1..8] :

T	5	13	18	23	46	53	89	97

Recherche de 46 :

Etape 1 : comparaison de 46 avec t(4) (4=(8+1)÷2), t(4)<46 => recherche dans [5..8] Etape 2 : comparaison de 46 avec t(6), (6=(8+5)÷2), t(6)>46 => recherche dans [5..5] Etape 3 : comparaison de 46 avec t(5), t(5)=46 => élément cherché est trouvé à l'indice 5

Recherche de 10 :

Etape 1 : comparaison de 10 avec t(4), t(4)>10 => recherche dans [1..3]
Etape 2 : comparaison de 10 avec t(2), t(2)>10 => recherche dans [1..1]
Etape 3 : comparaison de 10 avec t(1), t(1)<10 => recherche dans [2..1], Borne inférieure devient supérieure à la borne supérieure, donc on met fin à l'algorithme et l'élément cherché n'a pas été trouvé !

Nous avons l'algorithme suivant :

```
Début
Var e, n : entiers
Var Tableau t(n) : entier
    D ← 1
    F ← n
    trouve ←  faux
```

```
tant que (D <= F) et (trouve=faux) faire
   i ← (D + F)÷2
   Si t(i) = e alors
      trouve ← vrai
      sinon si t(i) > e alors
         F ← i -1
          Sinon
 D ←  i +1
      Fin si
   Fin si
Fin faire
Si trouve=vrai alors
   indice ←  i
   afficher " L'élément se trouve à la position" ; indice
sinon
indice ← -1
  afficher " L'élément n'existe pas dans le tableau"
Fin si
Fin
```

Légende :

- e désigne l'élément recherché
- n : taille du tableau
- t : tableau trié par ordre croissant
- D : début de la zone de recherche
- F : fin de la zone de recherche
- trouve : booléen, faux tant que l'élément cherché n'est pas trouvé
- i : indice de la case du milieu de la zone de recherche
- indice : indice de l'élément recherché ou -1 s'il n'est pas trouvé.

3.4. Tableaux multidimensionnels

Un tableau multidimensionnel est un tableau dont les éléments sont respectivement repérés par deux ou trois indices. C'est-à-dire un tableau à plusieurs dimensions et il utilise souvent des boucles imbriquées lors de sa mise en place.

Un tableau multidimensionnel doit être déclaré comme tel également avant son utilisation, en précisant la taille (intervalle de définition selon chaque dimension) et le type de valeurs qu'il contiendra. La syntaxe retenue est :

Var Tableau NomTableau (taille du tableau) : type d'éléments

La programmation autorise à déclarer des tableaux de dimensions multiples. Par exemple, l'instruction suivante déclare un tableau bidimensionnel de 10 sur 10 à l'intérieur d'une procédure :

A(9, 9) : Réels

Vous pouvez déclarer l'une et/ou l'autre de ces deux dimensions avec des limites inférieures explicites :

A(1 à 10, 1 à 10) : Réels

Vous pouvez étendre cela à plus de deux dimensions. Par exemple :

Declarer MultiD(3, 1 à 10, 1 à 15)

Cette déclaration crée un tableau en trois dimensions de 4 sur 10 sur 15. Le nombre total d'éléments est le produit de ces trois dimensions, soit 600.

Manipulation de tableaux à l'aide de boucles

Vous pouvez traiter efficacement un tableau multidimensionnel à l'aide de boucles Pour imbriquées. Les instructions suivantes, par exemple, initialisent chaque élément de A à l'aide d'une valeur basée sur sa position dans le tableau :

```
Declarer I, J: Entiers
A(1 à 10, 1 à 10) : Réels
Pour I = 1 à 10
  Pour J = 1 à 10
    A(I, J) = I * 10 + J
  Fin pour J
FinPour I
```

Tableaux à deux dimensions

Un tableau A à deux dimensions m x n est un ensemble de m.n éléments d'information tels que chacun d'entre eux est spécifié par une paire d'entiers (tels que J,K), appelés indices et possédant la propriété $1 \leq J \leq m$ et $1 \leq K \leq m$. L'élément de A dont le premier indice est j et le second, k sera noté $A_{j,k}$ ou A[J,K].

Les tableaux à deux dimensions sont appelés matrices et sont faciles à se représenter comme une grille ayant un certain nombre de lignes (première dimension) et un certain nombre de colonnes (seconde dimension).

Considérons une matrice (tableau) M avec 6 colonnes et 4 lignes dont :

56	54	1	- 56	20	22
72	8	54	34	43	2
70	5	16	78	90	4
56	23	- 47	0	5	12

Pour accéder à un élément du tableau, il suffit de préciser entre parenthèses l'indice de la case contenant cet élément, et ce pour chacune des dimensions. Par exemple, pour accéder à l'élément 23 du tableau d'entiers ci-dessus, on écrit : M (4,2). L'instruction suivante affecte à la variable x la valeur du premier élément du tableau, c'est à dire 56.

x ← M (1,1)

L'élément désigné du tableau peut alors être utilisé comme n'importe quelle variable :

M (2,1) ← - 56

Cette instruction modifie le contenu de la case (2,1) du tableau M. (72 en - 56)

Exemple : l'algorithme qui calcule la somme des éléments dans ce tableau

Début

Var Somme : réel

Var Tableau m() : réels

Var li, co : entiers

Afficher "Combien y a-t-il de lignes à saisir ?"

Lire li

Afficher "Combien y a-t-il de colonnes à saisir ?"

Lire co

Redim m(li, co)

Pour i de 1 à li : Pour j de 1 à co : Lire m(i,j) : Fin pour j : Fin pour i

Somme ← 0

Pour i de 1 à li : Pour j de 1 à co : Somme ← Somme + m(i,j) : Fin pour j
Fin pour i
Afficher Somme
Fin

CHAP4 : INTRODUCTION AU PYTHON

0. Introduction

Le langage de programmation Python a été créé en 1989 par Guido van Rossum, aux Pays-Bas. Le nom Python vient d'un hommage à la série télévisée Monty Python's Flying Circus dont G. van Rossum est fan. La première version publique de c en langage a été publiée en 1991.
La dernière version de Python est la version 3. Plus précisément, la version 3.7 a été publiée en juin 2018. La version 2 de Python est désormais obsolète et cessera d'être maintenue après le 1er janvier 2020. Dans la mesure du possible évitez de l'utiliser.

0.1. Installer et Configurer Python

Pour apprendre la programmation Python, il va falloir que vous pratiquiez et pour cela il est préférable que Python soit installé sur votre ordinateur. La bonne nouvelle est que vous pouvez installer gratuitement Python sur votre machine, que ce soit sous Windows, Mac OS X ou Linux. Pour installer Python vous pouvez vous référer, par exemple, au site suivant : http://matthieu-moy.fr/cours/infocpp-S2/install-python.html

0.2. Editeur de texte

L'apprentissage d'un langage informatique comme Python va nécessiter d'écrire des lignes de codes à l'aide d'un éditeur de texte. Si vous êtes débutants, on vous conseille d'utiliser notepad++ sous Windows, BBEdit ou CotEditor sous Mac OS X et gedit sous Linux. La configuration de ces éditeurs de texte est détaillée dans la rubrique Installation de Python disponible en ligne. Bien sûr, si vous préférez d'autres éditeurs comme Atom, Visual Studio Code, Sublime Text, emacs, vim, geany. . . utilisez-les !
À toute fin utile, on rappelle que les logiciels Microsoft Word, WordPad et LibreOffice Writer ne sont pas des éditeurs de texte, ce sont des traitements de texte qui ne peuvent pas et ne doivent pas être utilisés pour écrire du code informatique.

0.3. Caractéristiques du langage Python

Détaillons un peu les principales caractéristiques de Python, plus précisément, du langage et de ses deux implantations actuelles :

- ✓ Python est portable, non seulement sur les différentes variantes d'Unix, mais aussi sur les OS propriétaires : Mac OS, BeOS, NeXTStep, MS-DOS et les différentes variantes de Windows. Un nouveau compilateur, baptisé JPython, est écrit en Java et génère du bytecode Java.
- ✓ Python est gratuit, mais on peut l'utiliser sans restriction dans des projets commerciaux.
- ✓ Python convient aussi bien à des scripts d'une dizaine de lignes qu'à des projets complexes de plusieurs dizaines de milliers de lignes.
- ✓ La syntaxe de Python est très simple et, combinée à des types de données évolués (listes, dictionnaires...), conduit à des programmes à la fois très compacts et très lisibles.

0.4. Commentaires

Dans un script, tout ce qui suit le caractère # est ignoré par Python jusqu'à la fin de la ligne et est considéré comme un commentaire.
Les commentaires doivent expliquer votre code dans un langage humain. L'utilisation des commentaires est rediscutée dans le chapitre 15 Bonnes pratiques en programmation Python.

Voici un exemple :

```
1 # Votre premier commentaire en Python.
2 print (' Hello world !')
3# D'autres commandes plus utiles pourraient suivre.
```

0.5. Notions d'erreurs

La programmation est une démarche très complexe, et comme c'est le cas dans toute activité humaine, on y commet de nombreuses erreurs. Pour des raisons anecdotiques, les erreurs de programmation s'appellent des « *bugs* » (ou « bogues », en Français)4, et l'ensemble des techniques que l'on met en œuvre pour les détecter et les corriger s'appelle « *debug* » (ou « débogage »). Nous distinguons trois types d'erreurs :

- ✓ Erreur sémantique

L'erreur sémantique ou erreur de logique. S'il existe une erreur de ce type dans un de vos programmes, celui-ci s'exécute parfaitement, en ce sens que vous n'obtenez aucun message d'erreur, mais le résultat n'est pas celui que vous attendiez : vous obtenez autre chose.
En réalité, le programme fait exactement ce que vous lui avez dit de faire. Le problème est que ce que vous lui avez dit de faire ne correspond pas à ce que vous vouliez qu'il fasse.

- ✓ Erreur de syntaxe

Python ne peut exécuter un programme que si sa syntaxe est parfaitement correcte. Dans le cas contraire, le processus s'arrête et vous obtenez un message d'erreur. Le terme syntaxe se réfère aux règles que les auteurs du langage ont établies pour la structure du programme.

- ✓ Erreur à l'exécution

L'erreur en cours d'exécution (*Run-time error*), qui apparait seulement lorsque votre programme fonctionne déjà, mais que des circonstances particulières se présentent (par exemple, votre programme essaie de lire un fichier qui n'existe plus). Ces erreurs sont également appelées des *exceptions*, parce qu'elles indiquent en général que quelque chose d'exceptionnel (et de malencontreux) s'est produit. Vous rencontrerez ce type d'erreurs lorsque vous programmerez des projets de plus en plus volumineux,

4.1. Type de base

Une variable est une zone de la mémoire de l'ordinateur dans laquelle une valeur est stockée. Aux yeux du programmeur, cette variable est définie par un nom, alors que pour l'ordinateur, il s'agit en fait d'une adresse, c'est-à-dire d'une zone particulière de la mémoire.
En Python, la déclaration d'une variable et son initialisation (c'est-à-dire la première valeur que l'on va stocker dedans) se font en même temps.

4.1.1. Type des variables

Le type d'une variable correspond à la nature de celle-ci. Les trois principaux types dont nous aurons besoin dans un premier temps sont les entiers (integer ou int), les nombres décimaux que nous appellerons floats et les chaînes de caractères (string ou str). Bien sûr, il existe de nombreux autres types (par exemple, les booléens, les nombres complexes, etc.). Si vous n'êtes pas effrayés, vous pouvez vous en rendre compte ici

```
1 >>> y = 3.14
2 >>> y
3 3.14
4 >>> a = " bonjour "
5 >>> a
6 'bonjour '
7 >>> b = 'salut '
8 >>> b
9 'salut '
10 >>> c = """ girafe """
11 >>> c
```

Remarque :

Python reconnaît certains types de variable automatiquement (entier, float). Par contre, pour une chaîne de caractères, il faut l'entourer de guillemets (doubles, simples, voire trois guillemets successifs doubles ou simples) afin d'indiquer à Python le début et la fin de la chaîne de caractères. Dans l'interpréteur, l'affichage direct du contenu d'une chaîne de caractères se fait avec des guillemets simples, quel que soit le type de guillemets utilisé pour définir la chaîne de caractères.
En Python, comme dans la plupart des langages de programmation, c'est le point qui est utilisé comme séparateur décimal. Ainsi, 3.14 est un nombre reconnu comme un float en Python alors que ce n'est pas le cas de 3,14.

4.1.2. Nom des variables et nom réservé

Le nom des variables en Python peut être constitué de lettres minuscules (a à z), de lettres majuscules (A à Z), de nombres (0 à 9) ou du caractère souligné (_). Vous ne pouvez pas utiliser d'espace dans un nom de variable.
Sous Python, les noms de variables doivent en outre obéir à quelques règles simples :

- ✓ Un nom de variable est une séquence de lettres (a→z, A→Z) et de chiffres (0→9), qui doit toujours commencer par une lettre.

- ✓ Seules les lettres ordinaires sont autorisées. Les lettres accentuées, les cédilles, les espaces, les caractères spéciaux tels que $, #, @, etc. sont interdits, à l'exception du caractère _ (souligné).
- ✓ La casse est significative (les caractères majuscules et minuscules sont distingués). Attention : Info, info, INFO sont donc des variables différentes. Soyez attentifs !

En plus de ces règles, il faut encore ajouter que vous ne pouvez pas utiliser comme nom de variables les 33 « mots réservés » ci-dessous (ils sont utilisés par le langage lui-même) :

and	class	def	elif	except
break	continue	del	else	
False	if	None	raise	with
finally	import	nonlocal	return	yield
for	in	not	True	as
from	is	or	try	assert
global	lambda	pass	while	

4.1.3. Affectation

a. Affection simple

Nous savons désormais comment choisir judicieusement un nom de variable. Voyons à présent comment nous pouvons définir une variable et lui affecter une valeur. Les termes « affecter une valeur » ou « assigner une valeur » à une variable sont équivalents. Ils désignent l'opération par laquelle on établit un lien entre le nom de la variable et sa valeur (son contenu). En Python comme dans de nombreux autres langages, l'opération d'affectation est représentée par le signe égale.

```
>>> n = 7                    # définir la variable n et lui donner la valeur 7
>>> msg = "Quoi de neuf ?"   # affecter la valeur "Quoi de neuf ?" à msg
>>> pi = 3.14159             # assigner sa valeur à la variable pi
```

Les exemples ci-dessus illustrent des instructions d'affectation Python tout à fait classiques. Après qu'on les ait exécutées, il existe dans la mémoire de l'ordinateur, à des endroits différents :

- ✓ Trois noms de variables, à savoir n, msg et pi ;
- ✓ Trois séquences d'octets, où sont encodées le nombre entier 7, la chaîne de caractères Quoi de neuf ? et le nombre réel 3,14159.

Les trois instructions d'affectation ci-dessus ont eu pour effet chacune de réaliser plusieurs opérations dans la mémoire de l'ordinateur :

- ✓ Créer et mémoriser un nom de variable ; • lui attribuer un type bien déterminé (ce point sera explicité à la page suivante) ;
- ✓ Créer et mémoriser une valeur particulière ;

Établir un lien (par un système interne de pointeurs) entre le nom de la variable et l'emplacement mémoire de la valeur correspondante.

b. Affectation Multiple

Sous Python, on peut assigner une valeur à plusieurs variables simultanément. Exemple :

```
>>> x = y = 7
>>> x
7
>>> y
7
```

On peut aussi effectuer des affectations parallèles à l'aide d'un seul opérateur :

```
>>> a, b = 4, 8.33
>>> a
4
>>> b
8.33
```

Dans cet exemple, les variables a et b prennent simultanément les nouvelles valeurs 4 et 8,33.

4.1.4. Opérateurs et Expressions

On manipule les valeurs et les variables qui les référencent en les combinant avec des opérateurs pour former des expressions. Exemple :

```
>>> a, b = 7.3, 12
>>> y = 3*a + b/5
```

Dans cet exemple, nous commençons par affecter aux variables a et b les valeurs 7,3 et 12. Comme déjà expliqué précédemment, Python assigne automatiquement le type « réel » à la variable a, et le type « entier » à la variable b.

Python évalue chaque expression qu'on lui soumet, aussi compliquée soit-elle, et le résultat de cette évaluation est toujours lui-même une valeur. À cette valeur, il attribue automatiquement un type, lequel dépend de ce qu'il y a dans l'expression. Dans l'exemple ci-dessus, y sera du type réel, parce que l'expression évaluée pour déterminer sa valeur contient elle-même au moins un réel.

Les opérateurs Python ne sont pas seulement les quatre opérateurs mathématiques de base. Nous avons déjà signalé l'existence de l'opérateur de division entière //. Il faut encore ajouter l'opérateur ** pour l'exponentiation, ainsi qu'un certain nombre d'opérateurs logiques, des opérateurs agissants sur les chaînes de caractères, des opérateurs effectuant des tests d'identité ou d'appartenance, etc.

Signalons au passage la disponibilité de l'opérateur modulo, représenté par le caractère typographique %. Cet opérateur fournit le reste de la division entière d'un nombre par un autre

4.1.5. Priorité des Opérateurs

Lorsqu'il y a plus d'un opérateur dans une expression, l'ordre dans lequel les opérations doivent être effectuées dépend de règles de priorité. Sous Python, les règles de priorité sont les mêmes que celles qui vous ont été enseignées au cours de mathématique. Vous pouvez les mémoriser aisément à l'aide de l'acronyme PEMDAS :

- ✓ P pour parenthèses. Ce sont elles qui ont la plus haute priorité. Elles vous permettent donc de « forcer » l'évaluation d'une expression dans l'ordre que vous voulez.
  Ainsi 2*(3-1) = 4, et (1+1) **(5-2) = 8.
- ✓ E pour exposants. Les exposants sont évalués ensuite, avant les autres opérations.
  Ainsi 2**1+1 = 3 (et non 4), et 3*1**10 = 3 (et non 59049 !).
- ✓ M et D pour multiplication et division, qui ont la même priorité. Elles sont évaluées avant l'addition A et la soustraction S, lesquelles sont donc effectuées en dernier lieu.
  Ainsi 2*3-1 = 5 (plutôt que 4), et 2/3-1 = -0.3333... (plutôt que 1.0).
- ✓ Si deux opérateurs ont la même priorité, l'évaluation est effectuée de gauche à droite. Ainsi dans l'expression 59*100//60, la multiplication est effectuée en premier, et la machine doit donc ensuite effectuer 5900//60, ce qui donne 98. Si la division était effectuée en premier, le résultat serait 59 (rappelez-vous ici que l'opérateur // effectue une division entière, et vérifiez en effectuant 59*(100//60)).

4.2. Les fonctions

Les fonctions permettent en effet de décomposer un programme complexe en une série de sous-programmes plus simples, lesquels peuvent à leur tour être décomposés en fragments plus petits, et ainsi de suite. Nous allons commencer par décrire ici la définition de fonctions sous Python.
Les objets et les classes seront examinés plus loin. La syntaxe Python pour la définition d'une fonction est la suivante

```
def nomDeLaFonction(liste de paramètres):
    ...
    bloc d'instructions
    ...
```

- ✓ La fonction print

Nous avons bien évidemment déjà rencontré cette fonction. Précisons simplement ici qu'elle permet d'afficher n'importe quel nombre de valeurs fournies en arguments (c'est-à-dire entre les parenthèses). Par défaut, ces valeurs seront séparées les unes des autres par un espace, et le tout se terminera par un saut à la ligne.

Vous pouvez remplacer le séparateur par défaut (l'espace) par un autre caractère quelconque (ou même par aucun caractère), grâce à l'argument sep. Exemple :

```
>>> print ("Bonjour", "à", "tous", sep ="*")
Bonjour à tous
>>> print ("Bonjour", "à", "tous", sep ="")
Bonjour à tous
```

De même, vous pouvez remplacer le saut à la ligne par l'argument end :

```
>>> n =0
>>> while n<6 :
... print ("zut", end ="")
... n = n+1
...
zutzutzutzutzut
```

4.3. Contrôle de flux

Pour résoudre un problème informatique, il faut toujours effectuer une série d'actions dans un certain ordre. La description structurée de ces actions et de l'ordre dans lequel il convient de les effectuer s'appelle un algorithme. Les tests sont un élément essentiel à tout langage informatique si on veut lui donner un peu de complexité car ils permettent à l'ordinateur de prendre des décisions, Pour cela, Python utilise l'instruction if

4.3.1. Test

Pour aiguiller notre programme dans les différentes directions, nous avons besoin des techniques permettant de le faire, ainsi en fonction des circonstances rencontrées. Pour ce faire, nous devons disposer d'instructions capables de tester une certaine condition et de modifier le comportement du programme en conséquence. La plus simple de ces instructions conditionnelles est l'instruction if.

Exemple : Ecrire un algorithme qui demande un nombre à l'utilisateur, et l'informe ensuite si ce nombre est positif

```
>>> a=input("saisissez le nombre:")
>>> b=int(a)
>>> if (b>0) :
        print( "le nombre",a,"est positif")
```

- ✓ La première ligne nous demandons à l'utilisateur de saisir le nombre c'est une fonction que nous allons expliquer plupart
- ✓ La deuxième ligne nous avons converti le type
- ✓ La troisième ligne nous avons utilisé l'instruction if

1. Parfois, il est pratique de tester si la condition est vraie ou si elle est fausse dans une même instruction if. Plutôt que d'utiliser deux instructions if, on peut se servir des instructions if et else

Exemple : Ecrire un algorithme qui demande un nombre à l'utilisateur, et l'informe ensuite si ce nombre est positif ou négatif

```
>>> a=input ("saisissez le nombre :")
>>> b=int(a)
>>> if (b>0) :
>>>       print ("le nombre",a,"est positif")
>>> else:
>>>      print("le nombre",a,"est négatif")
```

2. On peut faire mieux encore en utilisant aussi l'instruction elif (contraction de "else if")

Exemple : Ecrire un algorithme qui demande un nombre à l'utilisateur, et l'informe ensuite si ce nombre est positif ou négatif (on inclut cette fois le traitement du cas où le nombre vaut zéro).

```
>>> a=input ("saisissez le nombre :")
>>> b=int(a)
>>> if (b>0):
>>>                print( "le nombre",a,"est positif")
>>> elif b== 0:
>>>                print("le nombre",a,"est nul")
>>> else:
>>>       print("le nombre",a,"est négatif")
```

4.3.2. Opérateur de comparaison

La condition évaluée après l'instruction if peut contenir les opérateurs de comparaison suivants (à connaitre ! ! !) :

X == y # x est égal à y
X ! = y # x est différent de y
X > y # x est plus grand que y

X < y # x est plus petit que y
X >= y # x est plus grand que, ou égal à y
X <= y # x est plus petit que, ou égal à y

Remarque :

Pour mieux programmer en python il faut faire attention à l'indentation Vous pouvez aussi indenter à l'aide de tabulations, mais alors vous devrez faire très attention à ne pas utiliser tantôt des espaces, tantôt des tabulations pour indenter les lignes d'un même bloc. En effet, même si le résultat paraît identique à l'écran, espaces et tabulations sont des codes binaires distincts : Python considérera donc que ces lignes indentées différemment font partie de blocs différents. Il peut en résulter des erreurs difficiles à déboguer.

En résumé

- ✓ L'exécution conditionnelle d'instructions est réalisée en utilisant le if ; ou le if …. else ; ou le if …. elif ….. else.
- ✓ Les blocs d'instructions sont toujours associés à une ligne d'en-tête contenant une instruction bien spécifique (if, elif, else, while, def, etc.) se terminant par un double point :
- ✓ Les blocs sont délimités par l'indentation : toutes les lignes d'un même bloc doivent être indentées exactement de la même manière.

4.3.3. Instruction Répétitive

L'une des tâches que les machines font le mieux est la répétition sans erreur de tâches identiques. Il existe bien des méthodes pour programmer ces tâches répétitives. En programmation, on appelle boucle un système d'instructions qui permet de répéter un certain nombre de fois (voire indéfiniment) toute une série d'opérations. Python propose deux instructions particulières pour construire des boucles : l'instruction for … in … et l'instruction while

4.3.3.1. La boucle while

Le mot while signifie « tant que » en anglais. Cette instruction utilisée à la seconde ligne indique à Python qu'il lui faut répéter continuellement le bloc d'instructions qui suit, tant que le contenu de la variable X reste inférieur à y.

Exemple :

```
>>> a = 0
>>> while (a < 7) :
        a = a + 1
        print(a)
```

Voici comment cela fonctionne :

- ✓ Avec l'instruction while, Python commence par évaluer la validité de la condition fournie entre parenthèses (celles-ci sont optionnelles, cad a est inférieur à 7)

- ✓ Si la condition se révèle fausse, alors tout le bloc qui suit est ignoré et l'exécution du programme se termine.
- ✓ Si la condition est vraie, alors Python exécute tout le bloc d'instructions constituant le corps de la boucle, c'est-à-dire :

– l'instruction a = a + 1 qui incrémente d'une unité le contenu de la variable a (ce qui signifie que l'on affecte à la variable a une nouvelle valeur, qui est égale à la valeur précédente augmentée d'une unité).

– l'appel de la fonction print () pour afficher la valeur courante de la variable a.

• lorsque ces deux instructions ont été exécutées, nous avons assisté à une première itération, et le programme boucle, c'est-à-dire que l'exécution reprend à la ligne contenant l'instruction while.

La condition qui s'y trouve est à nouveau évaluée, et ainsi de suite.

Dans notre exemple, si la condition a < 7 est encore vraie, le corps de la boucle est exécuté une nouvelle fois et le bouclage se poursuit.

4.3.3.2. La boucle for

En programmation, on est souvent amené à répéter plusieurs fois une instruction. Incontournables à tout langage de programmation, les boucles vont nous aider à réaliser cette tâche de manière compacte et efficace.

Imaginez par exemple que vous souhaitiez afficher les éléments d'une liste les uns après les autres. Dans l'état actuel de vos connaissances, il faudrait taper quelque chose du style :

```
animaux = ['girafe ', 'tigre ', 'singe ', 'souris ']

2 print ( animaux [0])
3 print ( animaux [1])
4 print ( animaux [2])
5 print ( animaux [3])
```

La forme générale d'une boucle inconditionnelle est

```
for < cible > in < objet > :
        < blocs d'instructions 1 >
        if < test1 > : break
        if < test2 > : continue
else :
        < blocs d'instructions 2 >
```

4.4. Les fonctions

1. La fonction print( )

Nous avons bien évidemment déjà rencontré cette fonction. Précisons simplement ici qu'elle permet d'afficher n'importe quel nombre de valeurs fournies en arguments (c'est-à-dire entre les parenthèses). Par défaut, ces valeurs seront séparées les unes des autres par un espace, et le tout se terminera par un saut à la ligne.

2. La fonction input ( )

La plupart des scripts élabores nécessitent a un moment ou l'autre une intervention de l'utilisateur (entrée d'un paramètre, clic de souris sur un bouton, etc.). Dans un script en mode texte (comme ceux que nous avons créés jusqu'à présent), la méthode la plus simple consiste a employer la fonction intégrée **input ( )**. Cette fonction provoque une interruption dans le programme courant. L'utilisateur est invite a entrer des caractères au clavier et a terminer avec ‹*Enter*›.

Pour pouvoir utiliser ces fonctions, il vous suffit d'incorporer la ligne suivante au début de votre script :

from math import *

Cette ligne indique a Python qu'il lui faut inclure dans le programme courant *toutes* les fonctions

- ✓ **sqrt(nombre)** pour assigner a la variable **racine** la racine carree de **nombre**,
- ✓ **sin(angle)** pour assigner a la variable **sinusx** le sinus de **angle** (en radians !),
- ✓ reset() On efface tout et on recommence
- ✓ goto(x, y) Aller à l'endroit de coordonnées x, y
- ✓ forward(distance) Avancer d'une distance donnée
- ✓ backward(distance) Reculer
- ✓ up() Relever le crayon (pour pouvoir avancer sans dessiner)
- ✓ down() Abaisser le crayon (pour recommencer à dessiner)
- ✓ color(couleur) couleur peut être une chaîne prédéfinie ('red', 'blue', etc.)
- ✓ left(angle) Tourner à gauche d'un angle donné (exprimé en degrés)
- ✓ right(angle) Tourner à droite
- ✓ width(épaisseur) Choisir l'épaisseur du tracé
- ✓ fill(1) Remplir un contour fermé à l'aide de la couleur sélectionnée
- ✓ write(texte) texte doit être une chaîne de caractères

Exercices

1. Affectez les variables temps et distance par les valeurs 6.892 et 19.7. Calculez et affichez la valeur de la vitesse. Améliorez l'affichage en imposant un chiffre après le point décimal.

2. Saisir un nom et un âge en utilisant l'instruction input(). Les afficher. Refaire la saisie du nom, mais avec l'instruction raw_input(). L'afficher.

3. Saisissez un flottant. S'il est positif ou nul, affichez sa racine, sinon affichez un message d'erreur.

4. L'ordre lexicographique est celui du dictionnaire. Saisir deux mots, comparez-les pour trouver le « plus petit » et affichez le résultat.

5. On désire sécuriser une enceinte pressurisée.

On se fixe une pression seuil et un volume seuil : pSeuil = 2.3, vSeuil = 7.41.

On demande de saisir la pression et le volume courant de l'enceinte et d'écrire un script qui simule le comportement suivant :

- ✓ Si le volume et la pression sont supérieurs aux seuils : arrêt immédiat ;
- ✓ Si seule la pression est supérieure à la pression seuil : demander d'augmenter le volume de l'enceinte ;
- ✓ Si seul le volume est supérieur au volume seuil : demander de diminuer le volume de l'enceinte ;
- ✓ Sinon déclarer que « tout va bien ».

Ce comportement sera implémenté par une alternative multiple.

6. Initialisez deux entiers : a = 0 et b = 10. Écrire une boucle affichant et incrémentant la valeur de a tant qu'elle reste inférieure à celle de b.
   Écrire une autre boucle décrémentant la valeur de b et affichant sa valeur si elle est impaire. Boucler tant que b n'est pas nul.