# Informatique Embarquée

## Syllabus de 5ème INFO

python™

## & ordinogramme

**Edition : 2014** **Professeurs : D. Mus, J. Pochet**

# Table des matières

# 1 La variable

## 1.1 La variable

Une variable a un nom et est destinée à contenir une information.
Une variable est associée à un type.

## 1.2 La déclaration de variable

Une variable est déclarée lorsqu'on précise son nom et qu'on lui donne une valeur (on affecte cette variable de la valeur). Le type sera automatiquement déduit de la valeur.
En python les variables sont dynamiquement typées (le type peut changer pour une même variable), cependant il est préférable de manipuler une variable de manière à ce qu'elle corresponde toujours au même type.

## 1.3 Le type de variable

On considérera trois catégories de types simples : entier, réel et booléen

## 1.4 L'affectation de la variable

Affecter une variable signifie lui donner une valeur.
En python le symbole d'affectation est « = ».
Une affectation va toujours de la droite vers la gauche, on met dans la variable à gauche la valeur qui est à droite.

**Exemples de déclarations de variables**

En python une déclaration de variable est : son nom suivi de « = » suivi de la valeur.

| Exemples de déclarations de variables | Commentaires |
|---|---|
| a=1 | On déclare une variable nommée « a » qui reçoit la valeur 1<br>→ le type de a est déduit comme un entier |
| b=1.0 | La variable nommée « b » est affectée de 1.0 (type réel) |
| c=True | La variable nommée « c » est affectée de True (type booléen) |
| somme=0 | La variable « somme » a un nom évocateur |

Même si en python la déclaration de variable n'est pas obligatoire, il est conseillé de prendre l'habitude de déclarer ses variables au début. Il faut que le nom de variable soit évocateur et ne contienne que des caractères valides (pas d'espaces, accents,…).
De plus, une variable doit toujours commencer par une lettre.

### 1.5 Incrémenter ou décrémenter une variable

Incrémenter une variable sous-entend augmenter cette variable d'une unité ; ce qui signifie « ajouter 1 à cette variable ».
Décrémenter une variable sous-entend diminuer cette variable d'une unité ; ce qui signifie « enlever 1 à cette variable ».

Comme on veut modifier une variable, on passe par l'affectation : « = » en python.
L'élément qui doit être modifié à gauche du signe « = » et la nouvelle valeur à droite.
Ainsi pour incrémenter une variable X on veut que celle-ci reçoive son ancienne valeur plus une unité. On écrira : X = X + 1

| Exemple d'incrémentation et de décrémentation | Commentaires |
|---|---|
| nb1 = 8<br>nb2 = 5<br>nb1 = nb1 + 1<br>nb2 = nb2 - 1 | Déclaration de nb1<br>Déclaration de nb2<br>Incrémentation de nb1<br>Décrémentation de nb2 |

**Incrémentation et décrémentation d'un pas différent de 1**

Lorsqu'on ne précise pas le pas, par défaut il s'agit de la valeur 1.
On peut cependant incrémenter (ou décrémenter) une variable en spécifiant un pas.

| Exemple d'incrémentation et de décrémentation | Commentaires |
|---|---|
| nb1 = 8<br>nb2= 5<br>nb1= nb1 + 10<br>nb2 = nb2 - 2 | Déclaration de nb1<br>Déclaration de nb2<br>Incrémentation de nb1 par un pas de 10<br>Décrémentation de nb2 par un pas de 2 |

## 2 L'instruction Ecrire et la fonction « print »

L'instruction Ecrire signifie que le programme renvoie au monde extérieur une information.
Le monde extérieur sera souvent représenté par l'utilisateur.
L'information renvoyée lors de l'instruction Ecrire est en général le contenu d'une variable.
Le plus souvent l'instruction Ecrire se fera par l'intermédiaire de la fonction « print ».

### 2.1 Afficher le type d'une variable

La fonction type() est une fonction préexistante qui renvoie une chaîne de caractère indiquant le type de son argument (ce qu'il y a entre les parenthèses).
Pour afficher le type d'une variable il faut d'abord avoir déclaré cette variable.

| Exemple d'affichage du type d'une variable | Commentaires |
|---|---|
| cote_max=10<br>print type(cote_max) | Déclaration de l'entier « cote_max »<br>Affichage du type de « cote_max » |

### 2.2 Afficher du texte

Pour afficher du texte il suffit de mettre la chaîne de caractère entre " " ou entre ' ' et de le précéder du mot clé « print ».

| Exemples d'affichage de texte | Commentaires |
|---|---|
| print "bonjour "<br>print 'bonjour '<br>print "bonjour l'inraci " | On demande d'afficher la chaîne "bonjour " à l'écran<br>Effet identique à la ligne précédente<br>Une chaîne avec un ' sera plutôt entourée de " " |

Dans certains cas on voudrait pouvoir afficher un texte qui comprend à la fois des « ' » et des « " ». On peut alors utiliser l'opérateur de concaténation « + » qui a pour effet de former une chaîne de caractère à partir de deux chaînes de caractères en les mettant bout à bout.

| Exemples de concaténation | Commentaires |
|---|---|
| print "bonjour " + "l'inraci "<br>print ' je dis : "bonjour ' + "l'inraci" +' ! "' | La chaîne "bonjour l'inraci " est formée puis affichée<br>Sera affiché je dis : "bonjour l'inraci ! " |

### 2.3 Afficher une variable

Pour afficher une variable il ne faut pas mettre les guillemets. Il faut simplement préciser le nom de la variable.

| Exemple d'affichage de variable | Commentaires |
|---|---|
| cote_max=10<br>print cote_max | Déclaration d'un entier « cote_max » initialisé à 10<br>Affichage du contenu du moment de la variable « cote_max » |

### 2.4 Affichage mixte

Par affichage mixte on entend un affichage à la fois de variable(s) et de chaîne(s) de caractères. Une façon simple de faire est de séparer les éléments par une virgule au sein de la fonction « print ».
Le texte doit toujours être entre " " ou entre ' ' mais pas les variables.
Attention les variables doivent avoir été préalablement déclarées.
La virgule sépare les éléments à afficher.

| Exemple d'affichage mixte avec la virgule | Commentaires |
|---|---|
| cote_max=10<br>cote=9<br>print "Laurent a eu ", cote , "/ ", cote_max | Déclaration d'un entier « cote_max » initialisé à 10<br>Déclaration d'un entier « cote » initialisé à 9<br>Affichage mixte ( « *Laurent a eu 9 / 10* » ) |

Une autre façon d'avoir un affichage mixte est d'utiliser l'opérateur de concaténation « + ».
Pour ce faire il faut que les éléments à concaténer soient tous les deux des chaînes de caractères.
Pas question de faire : print "ma cote : " + cote
Car la variable « cote » n'est pas une chaîne de caractères !

Ce qu'on peut par contre faire c'est utiliser la fonction str() qui va transformer le contenu de la variable en une chaîne de caractères (un string) correspondant à la valeur du moment de la variable.

| Exemple d'affichage mixte avec la fonction str() | Commentaires |
|---|---|
| cote_max=10<br>cote=9<br>print "Laurent a eu " + str(cote) + " / " + str( cote_max) | Affichage mixte ( « *Laurent a eu 9 / 10* » ) |

Remarques : Le séparateur « , » va ajouter un espace à l'affichage.
Avec la concaténation « + », l'espace doit être ajouté par l'utilisateur.

### 2.5 Utilisation de commentaire

Un commentaire est une explication que le programmeur ajoute qui ne sera pas prise en compte par le compilateur.
Sur une ligne, un commentaire est du texte placé après le caractère « # ».

| Exemples de commentaires | Commentaires |
|---|---|
| #mini programme<br>nb1 = 8 #déclaration de nb1<br>nb1= nb1 + 1 #incrémentation de nb1 | Les commentaires rendent le code plus lisible.<br>On peut placer un commentaire sur une ligne seule ou placer un commentaire pour détailler une ligne |

## 3 L'instruction Lire

L'instruction Lire signifie que le programme, pendant son exécution, doit faire l'acquisition d'une donnée extérieure. En général la donnée extérieure sera fournie par l'utilisateur par l'intermédiaire du clavier.

### 3.1 Lire un nombre

| Exemple de lecture et d'écriture | Commentaires |
|---|---|
| #somme de deux nombres | Commente l'énoncé de l'exercice |
| nb1 = 0 #déclaration de nb1 | |
| nb2 = 0 #déclaration de nb2 | |
| somme = 0 #déclaration de somme | |
| nb1= input("introduisez nb1 SVP: ") #Lire nb1 | L'utilisateur voit le texte "introduisez nb1 SVP: " et le programme attend sa donnée pour l'affecter à nb1 |
| nb2= input("introduisez nb2 SVP: ") #Lire nb2 | L'utilisateur voit le texte "introduisez nb2 SVP: " et le programme attend sa donnée pour l'affecter à nb2 |
| somme = nb1+nb2 | La variable somme est affectée du résultat de l'opération |
| print "la somme de ",nb1, " et ",nb2, " = ",somme | L'affichage renvoie le résultat à l'utilisateur (Ecrire) |

Pour lire un nombre nous avons utilisé la fonction « input ».

Cette fonction a pour but d'afficher sur écran un message destiné à l'utilisateur (afin qu'il sache la donnée à introduire) et à renvoyer cette donnée pour l'affecter à la variable qui doit la recevoir.
La transmission de donnée se fait grâce à la fonction « input » mais aussi grâce à l'affectation (« = » en python).

### 3.2 Lire une chaîne de caractères alphanumériques

Pour lire un caractère alphanumérique, on utilisera la fonction « raw_input ».

| Exemple de lecture et d'écriture | Commentaires |
|---|---|
| #somme de deux nombres<br>nom=" "<br>#Lire nom<br>nom= raw_input("introduisez votre nom: ")<br><br><br>#ecrire le contenu de la variable nom<br>print nom | Commente l'énoncé de l'exercice<br><br><br><br>L'utilisateur voit le texte "introduisez votre nom " et le programme attend sa donnée pour l'affecter à nom<br><br>L'affichage renvoie le contenu de la variable nom à l'utilisateur (Ecrire) |

## 4 Les opérateurs et l'affectation

### 4.1 Introduction

Les opérations sont formées à partir d'opérateurs. Les opérateurs arithmétiques sont :
+, -, *, / et permettent de constituer des opérations comme par exemple :
(nb1+nb2), (nb1-1), (10*nb2), (nb1/nb2),…
Les opérations sont constituées d'opérateurs mais aussi de variables et/ou constantes.

| Exemple d'utilisation d'opérations | Commentaires |
|---|---|
| #tests sur les opérateurs arithmétiques<br>nb1 = 0<br>nb2 = 0<br>somme = 0<br>difference=0<br>produit =0<br>quotient=0<br>nb1= input("introduisez nb1 SVP: ") #Lire nb1<br>nb2= input("introduisez nb2 SVP: ") #Lire nb2<br>somme = nb1+nb2<br>difference = nb1-nb2<br>produit = nb1*nb2<br>quotient = nb1/nb2<br>print "les nombres sont ", nb1, " et ", nb2<br>print "somme : ", somme, " et difference : ", difference<br>print "produit : ", produit, " et quotient : ", quotient | Commente l'énoncé de l'exercice<br><br><br><br><br><br><br>Lire<br>Lire<br>Affectation d'une opération<br>Affectation d'une opération<br>Affectation d'une opération<br>Affectation du quotient (entier)<br>Réaffichage des nombres lus<br>Ecrire somme et différence<br>Ecrire produit et quotient |

La priorité des opérateurs est la priorité habituelle. On peut aussi ajouter des ( ).

Les opérations sont souvent liées à une affectation. On peut alors dire que le résultat de l'opération sera affecté à une variable.
Exemple : produit = nb1*nb2

Dans certains cas le résultat de l'opération n'est pas simplement affecté à une variable mais est utilisé par une fonction ; comme au sein de la fonction « print » par exemple.
Exemple : print "somme : ", nb1+nb2, " et difference : ", nb1-nb2

### 4.2 Particularité de l'opérateur « / »

L'opérateur « / » est particulier dans le sens où le résultat de l'opération n'est pas celui que l'on imagine naturellement.
Par exemple l'opération « 4 / 5 » fournit comme résultat « 0 » alors que si on divise 4 par 5 on doit trouver 0,8.
La raison de cette différence est que l'opérateur « / » va fournir un résultat de type entier si les tous les éléments intervenants au sein de l'opération sont de type entier.
Ici comme le résultat de la division est 0,8 et que la division entière ne tient pas compte de ce qu'il y a après la virgule, le résultat de la division « 4 / 5 » donne 0.

*Comment obtenir la valeur exacte ?*

Imaginons que l'on veuille calculer la moyenne de deux nombres entiers.
Comme le résultat est susceptible d'être une valeur réelle il faut trouver une astuce pour avoir le résultat exact. Une façon de faire est illustrée ci-dessous en multipliant un nombre par « 1.0 ». Cet artifice ne change rien à la valeur du nombre mais va indiquer à l'opérateur « / » qu'un élément est de type réel et donc que le résultat de l'opération devra être réel.

| Exemple du calcul de la moyenne | Commentaires |
|---|---|
| #somme de deux nombres | Commente l'énoncé de l'exercice |
| nb1 = 0 #déclaration de nb1 | |
| nb2 = 0 #déclaration de nb2 | |
| moyenne = 0.0 #déclaration de moyenne | |
| nb1= input("introduisez nb1 SVP: ") #Lire nb1 | L'utilisateur voit le texte "introduisez nb1 SVP: " et le programme attend sa donnée pour l'affecter à nb1 |
| nb2= input("introduisez nb2 SVP: ") #Lire nb2 | L'utilisateur voit le texte "introduisez nb2 SVP: " et le programme attend sa donnée pour l'affecter à nb2 |
| moyenne = (nb1 + nb2 * 1.0) / 2 | La moyenne sera le résultat d'une division réelle |
| print "la moyenne de ",nb1, " et ",nb2, " = ",moyenne | L'affichage renvoie le résultat à l'utilisateur (Ecrire) |

### 4.3 L'opérateur modulo « % »

L'opérateur modulo est utilisé pour connaître le reste de la division entière.
Nous avions vu que pour la division entière, l'opération « 4 / 5 » fournit comme résultat « 0 » car « 4 » rentre 0 fois de manière entière dans « 5 ».

L'opérateur modulo va alors fournir le nombre entier qui correspond à ce qu'il reste à diviser lorsqu'on soustrait, du numérateur, le dénominateur autant de fois que le résultat de la division entière.
Dans notre exemple « 4 / 5 » la division entière est 0 et ce qu'il reste à diviser est 4.
Ainsi 4 MODULO 5 (noté 4%5) fournit comme résultat 4.

Autre exemple : 38%5 fournira comme résultat 3
En effet, la division entière « 38/5 » donne comme résultat 7 car 38 rentre 7 fois en entier dans 38. Ainsi, si l'on soustrait à 38 sept fois le dénominateur (autrement dit 35) il reste 3.

### *4.4 Opérateur exposant « ** »*

L'opérateur exposant est composé des deux caractères « ** ».
Ces caractères ne peuvent être séparés d'un espace, ils doivent être collés.

Exemple : nb = 5
print nb ** 3 # 125 ( = $5^3$) sera affiché à l'écran

### *4.5 Attention aux types des opérandes*

Les opérandes sont les paramètres d'une opération.
Par exemple dans l'opération « a + 1 » :
« a » et « 1 » sont les opérandes
« + » est l'opérateur
« a + 1 » est l'opération
L'opération correspondra à un résultat après l'exécution par la machine.

Il faut savoir qu'il est important de bien prendre en compte les types des opérandes.
En effet, le fonctionnement de l'opérateur sera différent suivant le type des opérandes ; un exemple vu précédemment est la particularité de l'opérateur « / ».
Pour rappel, avec l'opérateur « / », si toutes les opérandes sont des entiers le résultat est entier alors que si toutes les opérandes sont réelles le résultat est réel.

On peut également se demander ce qu'il se passera si les opérandes ont des types différents.
On a vu avec l'opérateur « / » que si nous avons un opérande entier et un opérande réel le résultat de l'opération sera réel mais ce n'est pas toujours aussi simple.

**• Dans certains cas des opérandes différents provoquent une erreur.**

Exemple, « 5 + ″bonjour″ » :
Il n'est pas possible d'additionner un nombre entier avec une chaîne de caractères.

**• Dans certains cas des opérandes différents provoquent un comportement non désiré.**

Exemple 1, « ″en effet″ + ″bonjour″ » :
On pourrait croire que l'opérateur « + » va tenter d'effectuer l'addition.
En réalité, avec des opérandes qui sont des chaînes de caractères (STRING), l'opérateur « + » effectue une concaténation.
La concaténation consiste à créer une chaîne de caractère composée d'autres chaînes de caractères.
Dans notre cas le résultat de l'opération « ″en effet″ + ″bonjour″ » donnera la chaîne de caractères : « ″en effetbonjour″ »

Exemple 2, « 5 * ″bonjour″ » :
On pourrait croire que l'opérateur « * » va tenter d'effectuer la multiplication.
En réalité, lorsqu'une des opérandes est une chaîne de caractères (STRING) et l'autre est un entier, l'opérateur « * » effectue une succession de concaténations.
Dans notre cas le résultat de l'opération « 5 * ″bonjour″ » donnera la chaîne de caractères : « ″ bonjourbonjourbonjourbonjourbonjour ″ »

Remarque : Si on souhaite concaténer un string et une constante, on peut transformer la constante en string avec la fonction str().
Exemple : print ″bonjour ″+ ″la ″ + str(5) + ″eme elec″

### *4.6 L'affectation*

Des opérations telles que décrites ci-dessus ne sont jamais utilisées seules.
En effet une opération telle que « a + 1 » ne sera jamais utilisée seule car le résultat n'est pas exploité.
Une façon simple d'exploiter le résultat est de le transmettre à une variable par l'intermédiaire d'une affectation.
Le signe de l'affectation en python est le symbole « = ».
Une affectation consiste à placer ce qu'il y a à droite de son symbole au sein de l'élément qui se trouve à sa gauche.

```
Exemple :   a = a + 1      #on place dans la variable « a » le résultat de « a + 1 »
                           # la variable « a » voit alors sa valeur augmentée de 1
```

Il est alors classique d'avoir une ligne qui reprend à la fois une affectation et une ou plusieurs opérations.

Remarque : Le résultat d'une opération peut également être fournie à une fonction :
Exemples : res = math.fabs(a + b)
print cote1 + cote 2

# 5 La séquence

## 5.1 Les éléments de base dans une séquence

L'informatique permet de faire un traitement automatique des informations.
La séquence est une façon simple qui permet de montrer la solution à un problème informatique de manière séquentielle. L'ordre a donc de l'importance.

Pour pouvoir traiter une information, il faut en disposer. C'est l'instruction Lire qui va permettre de stocker cette information dans une variable. Classiquement il faut avoir préalablement déclaré les variables avant de pouvoir y stocker une information.
La séquence commencera alors typiquement par :

- La déclaration des variables
- La lecture des informations à traiter

Une fois toutes les informations nécessaires acquises, il faut les traiter.
Un traitement simple consiste à effectuer un ensemble d'opérations et de fournir le résultat à une variable.
Le traitement sera typiquement :

- Des opérations
- Une affectation

Une fois le traitement terminé le programme se doit de fournir le résultat au monde extérieur. La sortie des données après le traitement se fera via :

- L'écriture

En résumé les éléments de base d'une séquence sont :

- La déclaration des variables
- La lecture des informations à traiter
- Les opérations
- L'affectation
- L'écriture

## 5.2 La séquence en python

| Exemple de séquence | Commentaires |
|---|---|
| moyenne = 0<br>nombre1 = 0<br>nombre2 = 0<br>nombre3 = 0 | #Déclaration des variables<br>#moyenne, nombre1, nombre2 et nombre3<br>#<br># |
| nombre1 = input("un nombre entier SVP : ")<br>nombre2 = input("un nombre entier SVP : ")<br>nombre3 = input("un nombre entier SVP : ") | #Lecture des trois nombres<br>#<br># |
| moyenne = (nombre1 + nombre2 + nombre3) / 3 | #traitement = opérations et affectation |
| print "la moyenne vaut : ", moyenne | # L'affichage renvoie le contenu de la variable moyenne à #l'utilisateur (Ecrire) |

Le résultat risque d'être approximatif parce que le programme est conçu pour ne manipuler que des entiers. Même si le but du programme est de calculer la moyenne de trois nombres entiers, il faudrait prévoir une variable « moyenne » de la catégorie des réels et s'arranger lors de l'affectation de cette variable pour lui affecter la valeur réelle de la moyenne.

***Exercice : Modifiez la séquence pour avoir un résultat réel pour la moyenne***

Même si en python la déclaration de variables n'est pas obligatoire, nous recommandons vivement de déclarer toutes les variables en début de programme.
Et cela pour plusieurs raisons :

- Nous pourrons facilement transposer ce programme vers un autre langage où la déclaration est obligatoire.
- Nous pouvons facilement faire le rapprochement avec l'algorithme
- Le fait de visualiser les noms de variables (évocateurs bien sûr) nous apporte le contexte du programme et augmente la lisibilité.

### *5.3 La séquence en LDA*

LDA signifie Langage de Description Algorithmique, il permet de mettre en évidence la structure du programme en restant indépendant du langage de programmation.
Le LDA est écrit en français, il utilise un certain nombre de mots clés qui doivent être soulignés. Il y a des mots clés tels que : Lire, Ecrire, Entiers, Réels… et des mots clés qui permettent de mettre en évidence les structures (voir plus tard).

• Pour déclarer une variable en LDA :

Il suffit d'écrire la catégorie de type suivi du nom de la variable.
Exemple de déclaration en LDA : Entier nombre1

Les trois catégories de types simples sont : Entier, Réel et Booléen

• Pour lire une donnée en LDA :

On écrit le mot clé « Lire » (en le soulignant) suivit de la (ou des) variable(s)
Exemple de lecture en LDA : Lire nombre1, nombre2, nombre3

Il est important que les variables aient été préalablement déclarées.

• Les opérations en LDA :

Les opérations sont constituées d'opérandes et d'opérateurs.
Les opérateurs les plus simples sont les opérateurs arithmétiques.
Les symboles en LDA sont simplement : + - * /
La division est considérée comme réelle.

• L'affectation en LDA :

L'affectation en LDA est symbolisée par une flèche dirigée vers la gauche. Ainsi on fait apparaître que c'est l'élément qui se trouve à gauche de la flèche qui se voit affectée du résultat de ce qui se trouve à droite.
Il est courant que l'affectation soit liée à une opération.

Exemple d'affectation en LDA : moyenne ← (nombre1 + nombre2 + nombre3) / 3

• Pour écrire une donnée en LDA :

On utilise le mot clé « Ecrire » (en le soulignant) suivit de la (ou des) variable(s)
Exemple d'écriture en LDA : Ecrire nombre1, nombre2, nombre3

| Exemple de séquence en LDA | Commentaires |
|---|---|
| Entiers nombre1, nombre2, nombre3, moyenne | #Déclaration des variables<br>#moyenne, nombre1, nombre2 et nombre3 |
| Lire nombre1, nombre2, nombre3 | #Lecture des trois nombres |
| moyenne ← (nombre1 + nombre2 + nombre3) / 3 | #affectation et opération |
| Ecrire moyenne | # L'affichage renvoie le contenu de la variable moyenne au<br># monde extérieur (Ecrire) |

Remarque : Le résultat risque d'être approximatif parce que le programme est conçu pour ne manipuler que des entiers.

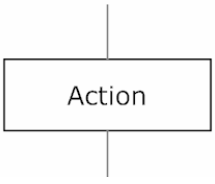
***Exercice :*** ***Modifiez la séquence en LDA pour avoir un résultat réel pour la moyenne***

### *5.4 La séquence en ordinogramme*

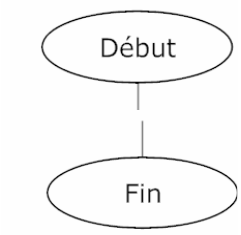
L'idée de l'ordinogramme est de faire apparaître, de manière visuelle, la structure du programme tout en restant indépendant du langage de programmation.

Pour une séquence pure (simple succession d'actions), l'ordinogramme ressemblera fort au LDA mais de manière visuelle.

Toutes les formes d'action sont représentées comme ici à droite par un rectangle.
Il peut s'agir d'une déclaration, d'une lecture, d'une écriture, d'une instruction ou d'un ensemble d'instructions.

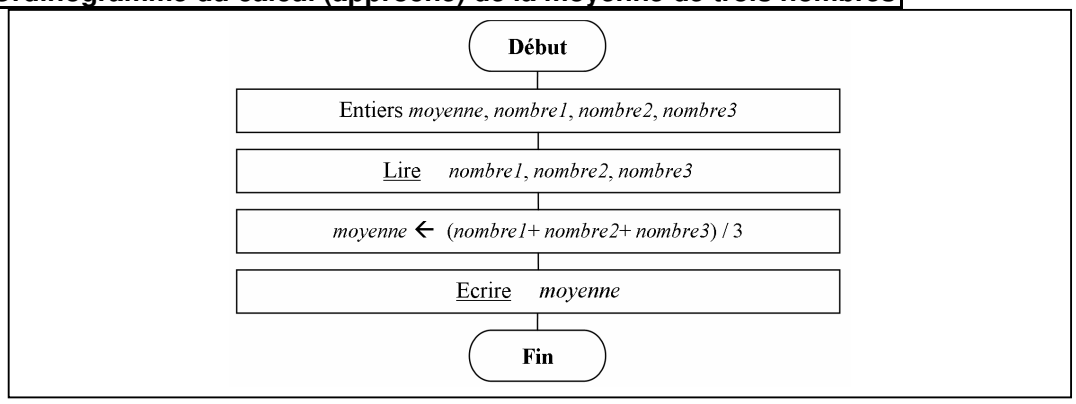


D'autre part, un ordinogramme commence toujours par l'entité




Et se termine toujours par

**Ordinogramme du calcul (approché) de la moyenne de trois nombres**



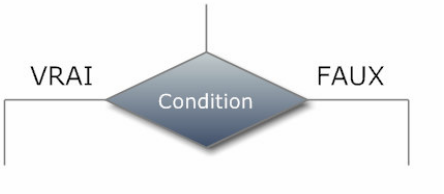

Remarque : Le résultat risque d'être approximatif parce que le programme est conçu pour ne manipuler que des entiers.

***Exercice :*** ***Modifiez la séquence en ordinogramme pour avoir un résultat réel pour la moyenne***

Pour une séquence pure comme celle ci-dessus, l'ordinogramme n'apporte rien de plus que le LDA. En effet, les différentes actions sont notées de la même manière.
De plus, le LDA est plus concis ce qui le rend préférable à l'ordinogramme.

Cependant, il y a une autre entité de l'ordinogramme qui permet d'examiner une condition et poursuivre la suite du programme suivant le résultat vrai ou faux de la condition. Cette entité, un losange (avec une entrée et deux sorties), apporte un aspect visuel intéressant à l'ordinogramme que l'on ne retrouve pas dans le LDA.

Tous les tests sont représentés comme ici par un losange à l'intérieur duquel se trouve le prédicat (la condition) dont le résultat sera soit vrai soit faux selon que la condition est vérifiée ou non. Suivant le cas, la suite du programme s'exécutera soit dans un sens soit dans l'autre.



### 5.5 Exemple de séquence simple avec l'utilisation de délais

**Exemple avec décompte**

On peut faire un décompte en utilisant des délais et une décrémentation.
Ici notre exemple fonctionne mais devra être amélioré à l'aide d'une structure répétitive.

| Exemple de décompte sans structure répétitive | Commentaires |
|---|---|
| import time<br>nb_sec_rest=3<br>print "fini dans ", nb_sec_rest, " s"<br>time.sleep(1)<br>nb_sec_rest= nb_sec_rest – 1<br>print "fini dans ", nb_sec_rest, " s"<br>time.sleep(1)<br>nb_sec_rest= nb_sec_rest - 1<br>print "fini dans ", nb_sec_rest, " s"<br>time.sleep(1)<br>nb_sec_rest= nb_sec_rest - 1<br>print "FINI" | On répète trois fois :<br><br>print "fini dans ", nb_sec_rest, " s"<br>time.sleep(1)<br>nb_sec_rest= nb_sec_rest – 1 |

En LDA ou en ordinogramme on se contentera d'écrire « Délai 1 seconde » pour symboliser l'action du délai d'une seconde.

## 6 La structure de choix simple

Lors de l'exécution du programme, il est possible d'examiner une condition afin d'exécuter un bloc d'actions que si cette condition est vraie.
Nous dirons qu'un bloc d'actions est simplement une suite successive de lignes.

**Constituer une condition**

Une condition est ce qui devra être examiné lors de l'exécution du programme afin de déterminer si elle est vraie ou fausse.
Après examen, une condition est donc un booléen (vrai ou faux).
Souvent une condition apparaît comme une opération dont le résultat est un booléen (vrai ou faux) ; Les opérateurs de comparaison le montrent bien.

Les opérateurs de comparaison en python sont :

< , <= , > , >= , == , !=

Exemples de conditions utilisant des opérateurs de comparaison :

(nb1<nb2), (nb1<=1), (10>nb2), (nb1>=nb2), (nb1==nb2), (nb1 !=nb2), …

Les conditions ci-dessus sont constituées d'opérateurs de comparaison mais aussi de variables et/ou de constantes. Les parenthèses ne sont en général pas nécessaires.
L'opérateur de comparaison « == » signifie «est la même chose que », à ne pas confondre avec l'opérateur d'affectation « = ».
L'opérateur de comparaison « != » signifie « est différent de ».

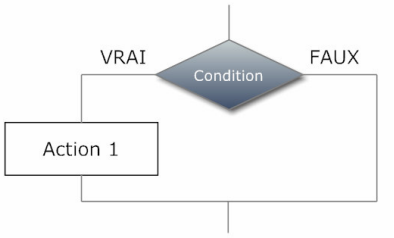
En LDA, tout comme pour les ordinogrammes, les opérateurs de comparaisons sont :
< , ≤, > , ≥ , = , ≠

### 6.1 La structure si-alors

Dans la mesure où il faut faire une action ou un bloc d'actions lorsqu'une condition est vérifiée, et qu'il ne faut rien faire lorsque cette même condition n'est pas vérifiée, on utilise la structure de choix simple de la forme « Si – Alors ».

**Principe général :**

Structure en ordinogramme :




En LDA

si condition alors
    Action 1

fsi

Syntaxe en python :

if condition :
    Action 1

Bien entendu le mot « condition » symbolise une condition (exemple : vitesse <30 ) et « Action 1 » symbolise une ou plusieurs lignes d'actions.

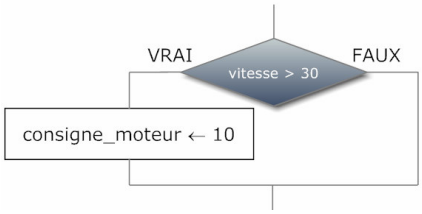
L'ordinogramme fait apparaître deux branches : une pour le vrai et une pour le faux.
Dans cette configuration « si-alors » on ne fait rien lorsque c'est faux.
La structure se termine après la recombinaison des deux branches.

Le LDA fait apparaître les mots clés « si », « alors » et « fsi ».
Ces mots clés doivent être soulignés ; le « fsi » indique la fin de la structure (la recombinaison des branches pour l'ordinogramme).

Le python utilise le « if », le « : » et l'indentation (décalage à droite).
Toutes les lignes d'actions liées à la branche « vrai » doivent être indentées (décalées vers la droite) par rapport à la position horizontale du « if ».
On considère que l'on est après la structure lorsque ce décalage disparaît.

**Exemple :**

Structure en ordinogramme :




En LDA

si vitesse > 30 alors
    consigne_moteur ← 10

fsi

Syntaxe en python :

if vitesse > 30 :
    consigne_moteur = 10

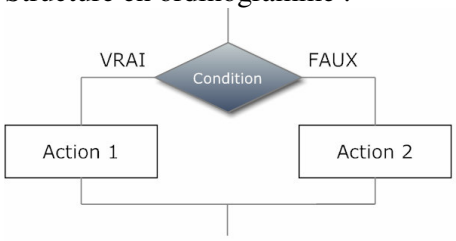
### 6.2 La structure si-alors-sinon

Nous avons vu que la structure de choix permet de préciser (en indentant) les lignes d'actions à exécuter lorsque la condition est examinée comme vraie au moment de l'exécution. Pour rappel la condition à examiner se trouve entre le « if » et le « : ».
Cependant il est également possible de définir un autre bloc d'actions à exécuter lorsque la condition est examinée comme fausse, pour ce faire on utilise le mot clé « else ».
Après le « else » on met « : » pour montrer que va commencer le bloc d'actions lié à une condition fausse ; toutes les lignes du bloc d'actions doivent être indentées.

| Exemple d'utilisation du « if » et du « else » | Commentaires |
|---|---|
| a = 55<br>b = input("un entier SVP:")<br>if b>a:<br>  print "1ere ligne du bloc d'actions du if"<br>  print b," est plus grand que ",a<br>  print "derniere ligne du bloc d'actions du if "<br>else:<br>  print "1ere ligne du bloc d'actions du else"<br>  print b," est plus petit ou égal à  ",a<br>  print "derniere ligne du bloc d'actions du else"<br>print "sans indentation, plus dans le else " | Lire b<br>Pendant l'exécution, la condition (b>a) est examinée<br><br>← Ce n'est que lorsque la condition est vraie<br>← que ce bloc d'actions (ici 3 lignes) est exécuté<br>Attention plus de condition après le mot else<br><br>← Ce n'est que lorsque la condition est fausse<br>← que ce bloc d'actions (ici 3 lignes) est exécuté<br>Ligne toujours affichée (peu importe la condition) |

**Principe général :**

Structure en ordinogramme :




En LDA

si condition alors
    Action 1
sinon
    Action 2
fsi

Syntaxe en python :

```
if condition :
    Action 1
else :
    Action 2
```

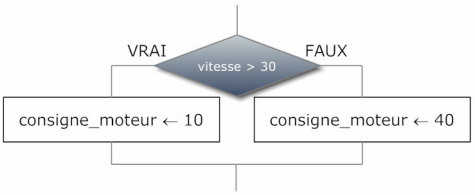
L'ordinogramme fait toujours apparaître deux branches, la structure se termine après leur recombinaison. Dans cette configuration « si-alors-sinon », la branche faux est liée à un bloc d'actions.

Le LDA fait apparaître le mot clé supplémentaire « sinon ».
Le « fsi » indique toujours la fin de la structure (la recombinaison des branches pour l'ordinogramme).

Le python fait apparaître le mot clé supplémentaire « else ».
Les différents blocs d'actions et la fin de la structure sont identifiés grâce à des indentations.

**Exemple :**

Structure en ordinogramme :




En LDA

```
si vitesse > 30 alors
   consigne_moteur ← 10
sinon
   consigne_moteur ← 40
fsi
```

Syntaxe en python :

```
if vitesse > 30 :
   consigne_moteur = 10
else :
   consigne_moteur = 40
```

**Exemple complet avec utilisation d'une condition au sein de la structure de choix**

Comme montré un peu plus tôt, il est apparu le besoin d'utiliser une condition avec la structure de choix. Nous verrons plus loin qu'il existe d'autres structures qui utilisent aussi une condition (comme la structure répétitive « tant que » par exemple).

| Exemples de conditions au sein de structures de choix | Commentaires |
|---|---|
| cote = input("votre cote sur 10 SVP:")<br>if cote==10:<br>  print "bravo !"<br>  print "vous avez la cote max !"<br>print "je suis après la structure " | Lire cote<br>Si cote est la même chose que 10<br>Alors afficher "bravo !"<br>  afficher "vous avez la cote max !"<br>afficher "je suis après la structure " |
| cote = input("votre cote sur 10 SVP:")<br>if cote !=10:<br>  print "snif !"<br>  print "vous n'avez pas la cote max !"<br>else :<br>  print "vous avez la cote max !"<br>print "je suis après la structure " | Lire cote<br>Si cote est différent de 10<br>Alors afficher "snif !"<br>  afficher "vous n'avez pas la cote max !"<br>sinon<br>  afficher "vous avez la cote max !"<br>afficher "je suis après la structure " |

### *6.3 Les structures imbriquées*

Nous pouvons résumer le principe de la structure de choix (avec un else) comme :

| Descriptif de la structure de choix (avec un else) | Commentaires |
|---|---|
| if condition :<br>  bloc_V<br>else:<br>  bloc_F<br>print "je suis après la structure " | Si condition est vraie Alors<br>  bloc_V<br>Sinon<br>  bloc_F<br>afficher "je suis après la structure " |

« bloc_V » correspond alors au bloc d'actions qui sera exécuté lorsque la condition est examinée comme vraie et « bloc_F » lorsqu'elle est examinée comme fausse.
Les deux blocs d'actions bloc_V et bloc_F représentent une suite successive de lignes. Il peut s'agir de 2 ,10 ou 1000 lignes. Un bloc d'actions peut même contenir lui-même une ou plusieurs autres structures. Il faut juste s'assurer que tout bloc d'actions contienne un nombre entier de structures (et pas de structures non achevées).

Lorsqu'on a une structure dans une structure on appelle ça des structures imbriquées.

| Descriptif de la structure de choix (avec un else) | Commentaires |
|---|---|
| if a>=0 :<br>  if  a==0 :<br>  print "a=0"<br>  else :<br>  print "a>0"<br>else :<br>  print "a<0" | Si a>=0 alors<br>  Si a est la même chose que 0 Alors<br>  afficher  "a=0<br>  Sinon<br>  afficher  "a>0"<br>Sinon<br>  afficher  "a<0" |

**Un if dans un else, une notation abrégée.**

Le bloc d'actions associé à un « else » est un ensemble de lignes successives.
Si ce bloc d'actions doit commencer par une structure de choix, au lieu d'écrire « else : if … » on peut utiliser une notation abrégée notée « elif… ».

| Un if dans un else, notation longue | Un if dans un else, notation abrégée |
|---|---|
| a = input("un entier SVP :")<br>if a>0 :<br>  print "positif"<br>else :<br>  if a==0 :<br>  print "nul"<br>  else :<br>  print "négatif" | a = input("un entier SVP :")<br>if a>0 :<br>  print "positif"<br>elif a==0 :<br>  print "nul"<br>else :<br>  print "négatif" |

Les deux exemples ci-dessus sont similaires (fonctionnement identique) cependant la version à droite est plus concise et n'exige pas deux niveaux d'indentation.

## 7 Les opérateurs logiques

Les opérateurs logiques permettent de constituer des conditions plus complexes.
Les opérateurs logiques sont : and (ET) or (OU) not(NON)

En LDA et en ordinogramme nous utiliserons les mots en français (ET, OU, NON).
En python nous utiliserons les mots clés réservés (and, or, not).

Les opérateurs logiques permettent de constituer une condition générale à partir d'autres conditions.

Par exemple la condition (a<=b and a<=c) est constituée de :
la condition a<=b , l'opérateur logique « and » , la condition a<=c

L'opérateur logique « and » constitue une condition qui sera vraie si et seulement si les deux autres conditions qui s'y rapportent sont vraies toutes les deux.
L'opérateur logique « or » constitue une condition qui sera vraie dès qu'une des deux autres conditions qui s'y rapportent est vraie (ou même le deux).

L'opérateur logique « not » ne se rapporte qu'à une seule condition.
Si cette dernière est « True », « not True » renvoie False, sinon « not False » renvoie True.

Les conditions constituées à l'aide d'opérateurs logiques sont en général utilisées au sein d'une structure (comme la structure de choix, la structure répétitive tant que,…).

| Utilisations d'opérateurs logiques, exemple1 : | Commentaires |
|---|---|
| a = input("veuillez introduire le nombre a :")<br>b = input("veuillez introduire le nombre b :")<br>c = input("veuillez introduire le nombre c :")<br>if a<=b and a<=c:<br>  print a, " est le plus petit"<br>else:<br>  print a, " n'est pas le plus petit" | Lire a<br>Lire b<br>Lire c<br>Si a<=b et que a<=c:<br>Alors  afficher a, " est le plus petit"<br>Sinon<br>  afficher a, " n'est pas le plus petit" |
| Utilisations d'opérateurs logiques, exemple2 : | Commentaires |
| cote = input("votre cote sur 10 SVP:")<br>if not cote ==10:<br>  print "snif !"<br>  print "vous n'avez pas la cote max !"<br>else :<br>  print "vous avez la cote max !"<br>print "je suis après la structure " | Lire cote<br>Si on n'a pas cote qui est la même chose que 10<br>Alors  afficher "snif !"<br>  afficher "vous n'avez pas la cote max !"<br>sinon<br>  afficher "vous avez la cote max !"<br>afficher "je suis après la structure " |

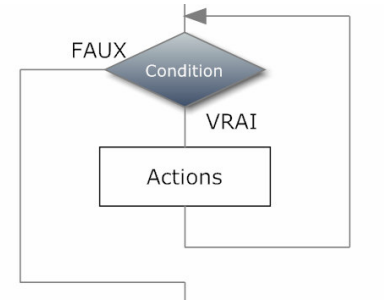
# 8 La tant que

## *8.1 Principe*

Ci-dessous une phrase qui décrit le principe général de la boucle « tant que » :

**Tant que condition répéter action.**

Cela signifie que tant qu'une condition est vérifiée, on répète un bloc d'actions.
Si nous représentons la structure d'une boucle « tant que » à l'aide d'un ordinogramme, nous obtenons la figure ci-dessous :



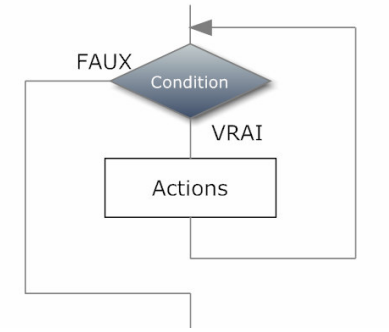

On remarque que la structure est en accord avec la description.

En effet, tant que la condition est vraie on répète le bloc d'actions.

On sort de la structure lorsque la condition est fausse.

**Représentation générale :**

Structure en ordinogramme :




En LDA

Tant que condition faire
    Actions
ftant

Syntaxe en python :

while condition :
    Actions

Comme pour la structure de choix, la structure « tant que » fonctionne avec une condition. La « tant que » est associée à un bloc d'actions qui sera exécuté tant que la condition est vraie. Il s'agit d'une structure répétitive car le bloc d'actions associé à la « tant que » est susceptible d'être répété si la condition continue à être vraie.

Contrairement à la structure de choix la condition peut être examinée plusieurs fois.
A la première exécution de la « tant que » la condition est examinée, si elle est fausse on sort de la structure et on passe à la suite (cas particulier où le bloc d'actions n'aura jamais été exécuté), si elle est vraie on effectue une fois le bloc d'actions et on réexamine la condition ; si au deuxième examen de la condition celle-ci est toujours vraie, on exécute pour la deuxième fois le bloc d'actions, on réexamine la condition et ainsi de suite. Tant que la condition sera vraie on va répéter le bloc d'actions, on ne sortira de la structure « tant que » que lorsque la condition sera examinée comme fausse (ou qu'une instruction break a eu lieu, voir plus tard).

| Descriptif de la structure répétitive « tant que » | Commentaires |
|---|---|
| while condition :<br>  bloc<br>print "je suis après la structure  " | Tant que condition est vraie, répéter<br>  bloc<br>afficher "je suis après la structure  " |
| Exemple d'utilisation de la "tant que" | Commentaires |
| #Attente de Lire la valeur 5<br>nb = input("Veuillez introduire un nombre entre 1 et 10")<br>while nb !=5 :<br>  print  nb, "n'est pas le nombre attendu, recommencez"<br>  nb = input("Veuillez introduire un nombre entre 1 et 10")<br>print "après la structure la condition est fausse"<br>print "votre nombre (",nb, ")vaut bien la valeur 5" | <br>Lire nb<br>tant qu'on n'a pas nb qui vaut 5 répéter<br>  afficher "…, recommencez"<br>  Lire nb<br>afficher "… la condition est fausse"<br>afficher "… vaut bien la valeur 5" |

### *8.2 La « tant que » pour contrôler le nombre de répétitions*

Il est possible d'utiliser la « tant que » pour répéter un bloc d'actions un nombre de fois donné. Nous avions vu un exemple de décompte qui montrait clairement que l'on répétait trois fois un bloc d'actions.
Pour rappel voici l'exemple à nouveau :

| Exemple de décompte sans structure répétitive | Commentaires |
|---|---|
| import time<br>nb_sec_rest=3<br><br>print "fini dans ", nb_sec_rest, " s"<br>time.sleep(1)<br>nb_sec_rest= nb_sec_rest – 1<br><br>print "fini dans ", nb_sec_rest, " s"<br>time.sleep(1)<br>nb_sec_rest= nb_sec_rest – 1<br><br>print "fini dans ", nb_sec_rest, " s"<br>time.sleep(1)<br>nb_sec_rest= nb_sec_rest – 1<br><br>print "FINI" | On répète trois fois :<br><br>print "fini dans ", nb_sec_rest, " s"<br>time.sleep(1)<br>nb_sec_rest= nb_sec_rest – 1 |

On voit bien ici le bloc d'actions qui est répété plusieurs fois.
Il vaut mieux dans ce cas utiliser une structure répétitive.
L'exemple ci-dessous utilise la variable de décompte au sein de la condition.

| Exemple de décompte avec structure répétitive | Commentaires |
|---|---|
| import time<br>nb_sec_rest=3<br>while nb_sec_rest>0:<br>  print "fini dans ", nb_sec_rest, " s"<br>  time.sleep(1)<br>  nb_sec_rest= nb_sec_rest – 1<br>print "FINI" | On répète trois fois :<br><br>print "fini dans ", nb_sec_rest, " s"<br>time.sleep(1)<br>nb_sec_rest= nb_sec_rest – 1 |

Une idée intéressante lorsqu'on veut répéter un bloc d'actions un nombre connu de fois est d'utiliser une variable dont le rôle est de compter le nombre de fois que le bloc d'actions a déjà été répété. On peut alors demander que « tant qu'on n'a pas atteint le nombre de répétitions souhaitées on répète le bloc d'actions », on utilise alors une structure répétitive « tant que » qui va utiliser une condition créée à partir de notre variable « compteur » et du nombre de répétitions souhaitées.

La condition sera par exemple (nb_repet_actu < nb_repet_a_faire) où :

| | |
|---|---|
| nb_repet_actu | est la variable qui indique le nombre de répétitions qui a déjà eu lieu actuellement |
| nb_repet_a_faire | est une variable ou une constante qui indique le nombre total de répétitions que l'on souhaite faire. |

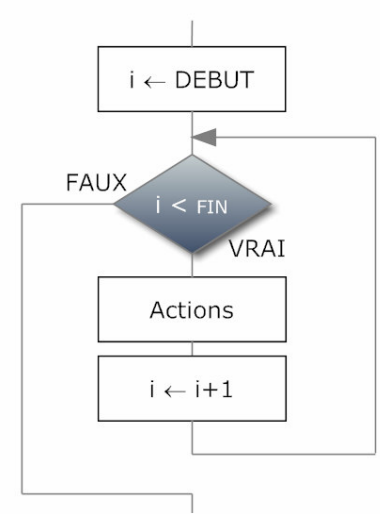
Pour que cette façon de faire fonctionne, il faut s'assurer que la variable *nb_repet_actu* soit initialisée à 0 et qu'elle soit incrémentée à chaque exécution du bloc d'actions.

La variable qui sert de compteur interne (nommée nb_repet_actu ci avant) est conventionnellement notée « i ».

Typiquement il y a une incrémentation de cette variable à la fin de chaque répétition du bloc d'actions. Il y a également un test propre à la boucle qui est effectué régulièrement de manière à déterminer s'il faut continuer à exécuter le bloc d'actions ou si le nombre de répétitions qui ont déjà été effectuées a atteint ce qui a été spécifié (dans ce cas on sort de la boucle).

**Répétition d'un bloc d'actions un nombre donné de fois avec une « tant que »**

**Ordinogramme**




**En python**

**Principe :**

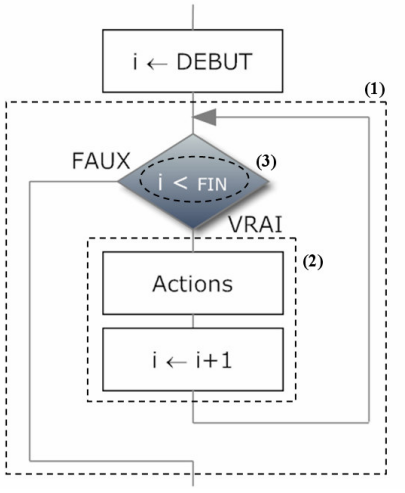
```
i =DEBUT
while i < FIN:
    Actions
    i = i– 1
#après la structure
```

**Exemple ( 10 affichages de « Bonjour » ) :**

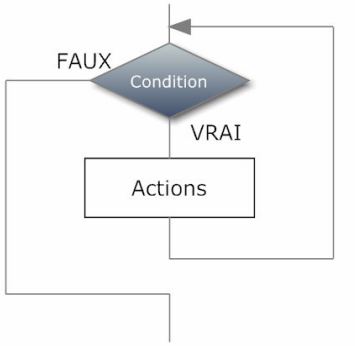
```
i =0
while i < 10:
    print ″bonjour″
    i = i– 1
#après la structure
```

On peut voir ci-dessous que La zone (1) fait bien apparaître une « tant que » qui englobe un bloc d'actions (2) et une condition (3).

**Gestion des répétitions**




**Structure d'une « tant que »**



| Exemples de répétitions connaissant le nombre d'itérations | Commentaires |
|---|---|
| #Afficher bonjour 10 fois de suite<br>nb_repet_actu =0<br>while nb_repet_actu < 10 :<br>  print "bonjour !"<br>  nb_repet_actu = nb_repet_actu + 1 | On initialise nb_repet_actu à 0<br>Tant qu'on n'a pas atteint 10 répétitions<br>  Afficher "bonjour !"<br>  « mettre à jour la variable nb_repet_actu » |
| #Afficher bonjour n fois de suite (n introduit par l'utilisateur)<br>nb_repet_a_faire = input("Combien de répétitions SVP ? ")<br>nb_repet_actu =0<br>while nb_repet_actu < nb_repet_a_faire :<br>  print "bonjour !"<br>  nb_repet_actu = nb_repet_actu + 1 | Lire *nb_repet_a_faire*<br>On initialise nb_repet_actu à 0<br>Tant qu'on n'a pas atteint *nb_repet_a_faire*<br>  Afficher "bonjour !"<br>  « mettre ajour la variable nb_repet_actu » |

Cependant il existe une structure répétitive plus adaptée que la « tant que » lorsqu'on connaît le nombre de répétitions, il s'agit de la boucle FOR. Cette boucle utilise d'ailleurs le principe d'une variable « compteur » initialisée au début, mise à jour à chaque tour de boucle et testée régulièrement pour assurer le nombre de répétitions souhaitées.

### *8.3 L'usage du break*

L'instruction break permet de sortir de la structure englobante la plus proche qui ne soit pas une structure de choix.
On peut très bien avoir une boucle « tant que » classique où tant que la condition est vraie on répète le bloc d'actions et dés que la condition devient fausse on sort de la boucle.
Cependant, on peut prévoir une façon de sortir de la boucle anticipativement grâce à l'instruction break.

| Exemple d'utilisation du break au sein d'une tant que | Commentaires |
|---|---|
| #Somme de 5 chiffres introduits par l'utilisateur<br>print "un nombre n'est pas toujours un chiffre"<br>chiffre=0<br>somme=0<br>nb_repet_actu =0<br>while nb_repet_actu < 5 :<br>  chiffre = input("un nombre positif à un seul chiffre SVP : ")<br>  if chiffre>9 :<br>  print "vous avez introduits plusieurs chiffres"<br>  break<br>  somme=somme + chiffre<br>  nb_repet_actu = nb_repet_actu + 1<br>print "je suis après la structure "<br>if nb_repet_actu == 5 :<br>  print "la somme des 5 chiffres vaut : ", somme | L'utilisateur doit introduire 5 fois de suite un nombre à un seul chiffre.<br>On utilise alors une structure tant que et la gestion de la variable *nb_repet_actu* pour assurer 5 répétitions.<br>Cependant l'exécution de la boucle peut s'interrompre anticipativement si l'utilisateur introduit un nombre à plusieurs chiffres.<br><br>Une fois que nous sommes sortis de la boucle, la variable *nb_repet_actu* permet de déterminer si nous en sommes sortis anticipativement ou non. |

# 9 L'algèbre de Boole

## 9.1 Généralités

L'algèbre de Boole est un outil mathématique qui permet de mettre en équation des fonctions combinatoires élémentaires (ET, OU,…) ou des fonctions combinatoires plus complexes (imbrication de fonctions combinatoires élémentaires).
Toute fonction combinatoire correspond à une expression mathématique appelée expression booléenne.

De la même manière qu'au cours de mathématique, les expressions booléennes sont constituées de variables, de constantes et d'opérateurs mathématiques. La grande différence réside dans le fait qu'avec l'algèbre de Boole les variables et les constantes ne peuvent prendre que 2 valeurs possibles : 0 et 1.

Souvent, pour représenter une fonction combinatoire, on utilise une équation qui contient la sortie dans le membre de gauche et l'expression booléenne correspondante dans le membre de droite. On trouvera par exemple : S = a + b

a, b et S sont des variables (ne peuvent donc prendre que les valeurs 0 et 1).
a et b sont les entrées.
S est la sortie, elle dépend de l'état des entrées et de l'expression booléenne.
a + b est l'expression booléenne de la fonction combinatoire de S.

## 9.2 Quelques particularités de l'algèbre de Boole

Non seulement les variables ne peuvent prendre que les valeurs 0 et 1 mais il en est de même pour les expressions. Ce qui signifie qu'une expression telle que (a+b) ne pourra contenir que les valeurs 0 et 1, même si a et b valent tous les deux 1.

$\boxed{1+1=1}$ Car le 2 n'existe pas et le résultat ne peut valoir que 0 ou 1.

$\boxed{a+a=a}$ Soit a vaut 0 et a + a = 0, soit a vaut 1 et a + a =1.

$\boxed{a+\overline{a}=1}$ $\boxed{a\times\overline{a}=0}$ Le trait horizontal au dessus signifie « complément de ».
On comprend alors que si $a=0$, $\overline{a}=1$ et si $a=1$, $\overline{a}=0$.

En examinant tous les cas, on peut montrer que les formules ci-dessus sont toujours vérifiées.

### 9.3 Les fonctions combinatoires élémentaires

Il y a trois fonctions combinatoires élémentaires : les fonctions ET, OU et NON.
Nous verrons que ces fonctions correspondent à des opérateurs particuliers au niveau de l'algèbre de Boole.

- La fonction ET correspond à une multiplication

L'opération $a \times b$ ne vaudra 1 que si $a = 1$ et $b = 1$
La table de vérité de la fonction ET est donnée ci-dessous :

| $a$ | $b$ | $a$ ET $b$ | $S = a \times b$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 |

Au point de vue schématique, on peut montrer que la fonction logique ET correspond à la mise en série.

- La fonction OU correspond à une addition

L'opération $a + b$ vaudra 1 dés que $a = 1$ ou $b = 1$ (ou non exclusif)
La table de vérité de la fonction OU est donnée ci-dessous :

| $a$ | $b$ | $a$ OU $b$ | $S = a + b$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 |

Au point de vue schématique, on peut montrer que la fonction logique OU correspond à la mise en parallèle.

- La fonction NON correspond au trait horizontal au dessus

L'opération $\overline{a}$ vaudra 1 dés que $a = 0$ et inversement
La table de vérité de la fonction NON est donnée ci-dessous :

| $a$ | NON $a$ | $S = \overline{a}$ |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 0 | 0 |

### 9.4 Les théorèmes de De Morgan

Le complément d'une somme est égal au produit des compléments → $\boxed{\overline{a+b} = \overline{a} \times \overline{b}}$

Le complément d'un produit est égal à la somme des compléments → $\boxed{\overline{a \times b} = \overline{a} + \overline{b}}$

Ces lois traduisent le fait que :
- Le résultat d'une somme vaudra 1 dés qu'un des éléments de la somme vaut 1
- Le résultat d'un produit vaudra 0 dés qu'un des éléments du produit vaut 0

### 9.5 Exemple d'application de De Morgan en programmation

Dans certains cas on veut rester dans une boucle tant qu'on n'a pas une condition qui soit vraie. Par exemple on voudrait que l'utilisateur introduise un nombre positif à un chiffre (donc compris entre 0 et 9).
On doit alors faire une boucle qui demande à l'utilisateur de recommencer tant qu'on n'a pas une introduction correcte.

*Exemple, demande d'introduction d'un nombre positif à un chiffre, version 1 :*

```
nb_introduit = -1       # valeur qui assurera de rentrer dans la boucle
while (nb_introduit < 0) or (nb_introduit > 9) :
        nb_introduit  = input("Veuillez introduire un chiffre positif ou nul, SVP : ")
print "merci, vous avez introduit : ", nb_introduit
```

L'exemple ci-dessus fonctionne très bien mais il arrive parfois de se tromper au niveau des conditions. Si on remplace le « or » par un « and » on a une condition toujours fausse.
Pour augmenter la lisibilité, on peut utiliser De Morgan pour faire apparaître une condition équivalente

*Exemple, demande d'introduction d'un nombre positif à un chiffre, version 2 :*

```
nb_introduit = -1       # valeur qui assurera de rentrer dans la boucle
while not ((nb_introduit >= 0) and (nb_introduit <= 9)) :
        nb_introduit  = input("Veuillez introduire un chiffre positif ou nul, SVP : ")
print "merci, vous avez introduit : ", nb_introduit
```

Cette façon de faire illustre bien De Morgan de la forme : $\boxed{\overline{a \times b} = \overline{a} + \overline{b}}$

| | | | |
|---|---|---|---|
| Avec | a | qui correspond à | (nb_introduit >= 0) |
| | b | qui correspond à | (nb_introduit <= 9) |

Ceci augmente la lisibilité car on peut lire « tant qu'on n'a pas nb_introduit >= 0 et nb_introduit <= 9 (autrement dit un chiffre positif ou nul) on reste dans la boucle.

# 10 Les fonctions

Une fonction permet de contenir un ensemble d'instructions à exécuter.
Une fonction a un nom et permet d'effectuer une tâche.

## *10.1 Utilisation d'un délai (fonction existante)*

**Utilisation d'un délai (exemple manipulant une fonction de « time »)**

Pour utiliser des utilitaires qui permettent d'attendre un délai, on écrira au début du programme « import time ».

| Exemple d'utilisation d'un délai | Commentaires |
|---|---|
| import time | On écrit « import time » au début |
| print "salut" | On affiche « salut » |
| time.sleep(1) | On attend une seconde |
| print "tout le monde" | Lorsque la seconde est passée on affiche « tout le monde » |

## *10.2 La définition et l'appel d'une fonction*

Une fonction peut être appelée régulièrement au sein de votre programme mais doit avoir été préalablement définie. La définition de la fonction indique ce qu'il faut faire lorsque la fonction est appelée afin d'être exécutée. La définition est aussi appelée l'implémentation, on la reconnaît grâce au mot clé « def ».

L'appel d'une fonction correspond à l'utilisation proprement dite. Une fonction peut être appelée au sein du programme principal ou même au sein d'une autre fonction.
Ci-dessous un exemple de fonction dont le rôle est d'afficher un message, qui est définie (une seule fois bien sûr), mais qui est utilisée plusieurs fois.

Exemple de mise en œuvre d'une fonction (la déclaration et deux appels de fonctions).

```
#On commence à définir la fonction
def msg_derivee_fct():
    print "la dérivée d'une grandeur par rapport au temps correspond à la vitesse de variation de cette grandeur."
#On n'est plus dans la définition de la fonction car la ligne n'est plus indentée
print "la dérivée de l'énergie par rapport au temps est la puissance."
print "pour rappel :"
msg_derivee_fct()                                   #On appelle la fonction
print "la dérivée de la vitesse par rapport au temps est l'accélération."
print "Nous avions vu l'interprétation de la dérivée :"
msg_derivee_fct()                                   #On appelle la fonction
```

## *10.3 Fonction sans paramètres et sans valeurs de retours*

L'exemple montré ci-dessus n'a pas de paramètre. En effet, tant au niveau de la définition de la fonction qu'au niveau de ses appels, il n'y a rien entre les parenthèses.

Les parenthèses permettent d'identifier une fonction mais aussi d'y placer des paramètres éventuels.
Des parenthèses vides décrivent alors des fonctions sans paramètre.

Les paramètres sont également appelés des arguments, ils permettent de transmettre des informations à la fonction pour que celle-ci puisse effectuer sa tâche.

Par ailleurs l'exemple précédent n'a pas non plus de valeur de retour, autrement dit l'appel de la fonction permet d'exécuter un certain nombre d'instructions mais la fonction ne renvoie pas un résultat qui pourrait être exploité.
Ci-dessous vous avez un autre exemple de mise en œuvre d'une fonction sans paramètre ni valeur de retour. La fonction se contente d'afficher la table de 5.

Deuxième exemple de mise en œuvre d'une fonction (la déclaration et deux appels de fonctions).

```
#On commence à définir la fonction
def affiche_table_5():
    facteur=1
    while facteur<=10:
        print facteur," fois 5 :",5*facteur
        facteur=facteur+1
print "voici la table de 5"
affiche_table_5()
print "et encore une fois :"
affiche_table_5()
```

Le nom de la fonction est « affiche_table_5 », elle est définie au début et est appelée ici deux fois dans le programme.

### *10.4 Fonction avec paramètre mais sans valeur de retour*

Une fonction avec un paramètre doit recevoir une valeur au moment de l'appel pour qu'elle puisse s'exécuter. Par exemple lorsqu'on veut utiliser une fonction pour générer un délai, il est commode de préciser le délai que l'on souhaite attendre lors de l'appel de fonction. On peut par exemple imaginer alors une fonction nommée attendre_en_sec() où il suffit d'indiquer le nombre de secondes à attendre entre les parenthèses. Ainsi,

pour générer un délai de 5 secondes on écrit : attendre_en_sec(5)
pour générer un délai de 3 secondes on écrit : attendre_en_sec(3)

On comprend que le nombre de secondes d'attente sera paramétrable, c'est pour ça que l'information transmise à la fonction au moment de l'appel est appelée paramètre.

Pour définir une fonction avec un paramètre, il faut placer une variable entre les parenthèses de la définition et utiliser la variable au sein de l'implémentation de la fonction. Cette variable peut avoir n'importe quel nom valable et sa valeur* représente l'information qui sera transmise à la fonction au moment de l'appel.

* Le passage de paramètre se fait par valeur ou par copie pour les types primitifs. Il est à noter qu'il existe également ce qu'on appelle le passage par référence.

La variable utilisé au sein de la définition de la fonction est une variable locale à la fonction (ainsi que toutes les autres variables déclarées de façon classique au sein de la fonction).

Ci-dessous nous allons définir une fonction dont le rôle est d'afficher la table d'un entier passé en paramètre :

Exemple de mise en œuvre d'une fonction avec un paramètre

```
#On commence à définir la fonction
def affiche_table(nb):
    facteur=1
    while facteur<=10:
        print facteur," fois ",nb," : ",nb*facteur
        facteur=facteur+1
print "voici la table de 5"
affiche_table(5)
print "et la table de 9 :"
affiche_table(9)
```

Pour comprendre l'exécution du programme, il faut commencer par se pencher sur les quatre lignes concernées, ce sont les quatre dernières :

| | | |
|---|---|---|
| print "voici la table de 5" | → | Du texte est affiché sur écran |
| affiche_table(5) | → | La fonction affiche_table() est appelée, la constante 5 est communiquée à la variable « nb » de la fonction |
| print "et la table de 9 :" | → | Du texte est affiché sur écran |
| affiche_table(9) | → | La fonction affiche_table() est appelée, la constante 9 est communiquée à la variable « nb » de la fonction |

**Une fonction peut recevoir une variable comme paramètre au moment de l'appel**

Il est possible de transmettre une variable à une fonction au moment de l'appel.
Ci-dessous on trouve un exemple presque identique au précédent sauf que l'appel fait apparaître une variable entre parenthèses.

| Exemple d'appel d'une fonction en utilisant une variable comme paramètre | Commentaires |
|---|---|
| def affiche_table(nb):<br>  facteur=1<br>  while facteur<=10:<br>  print facteur," fois ",nb," : ",nb*facteur<br>  facteur=facteur+1<br>nb_choisi=input("Veuillez donner un nb entier SVP")<br>print "voici la table de ",nb_choisi<br>affiche_table(nb_choisi) | Lorsque la dernière ligne est exécutée, c'est-à-dire :<br>affiche_table(nb_choisi)<br><br>la valeur de la variable *nb_choisi* sera communiquée à la variable *nb* de la définition de la fonction. |

Il est possible d'utiliser le même nom de variable comme paramètre de l'appel et au sein de la définition de la fonction. Cependant la variable dans la fonction (variable locale) ne représente pas celle dans le programme principal (variable globale).

### 10.5 Fonction avec valeur de retour

Normalement une « vraie » fonction renvoie un résultat et donc possède une valeur de retour. Les fonctions qui ne possèdent pas de valeur de retour sont parfois appelées procédures.

L'avantage d'une fonction qui renvoie un résultat (donc avec une valeur de retour) et que ce résultat peut être exploité en dehors de la fonction.

On peut par exemple affecter la valeur de retour de la fonction à une variable.

Imaginons que l'on veuille calculer la température en Celcius d'une température donnée en Fahrenheit. On peut alors créer une fonction nommée *conversion_fahrenheit_celcius* qui reçoit comme paramètre une température en fahrenheit et qui calcule la température Celcius équivalente avant de fournir le résultat comme valeur de retour.

Pour la conversion on peut se baser sur la formule suivante : T(°C) = (T(°F) - 32)/1,8

| Exemple d'utilisation d'une fonction avec une valeur de retour | Commentaires |
|---|---|
| def conversion_fahrenheit_celcius(temp_F):<br>  temp_C= (temp_F - 32)/9.0*5.0<br>  return temp_C | Le mot clé *return* termine la fonction et donne le résultat |
| print "un petit test ..." | |
| temp_test=conversion_fahrenheit_celcius(82) | La valeur de retour est affectée à *temp_test* |
| print "82 Fahrenheit correspondent à ",temp_test," Celcius" | |
| temp_F_choisie=input("Une température en Fahrenheit SVP ") | |
| temp_C_convertie=conversion_fahrenheit_celcius(temp_F_choisie) | Le résultat est affecté à *temp_C_convertie* |
| print temp_F_choisie," F --> ",temp_C_convertie," C" | |

### 10.6 Fonction avec plusieurs valeurs de retour

En python, contrairement à beaucoup de langage, il est possible d'avoir plusieurs valeurs de retour pour une fonction.

Cette fonctionnalité un peu particulière provient de la possibilité de faire des affectations multiples.
En python on peut par exemple écrire : a,b = 8, 20
Ça signifie que *a* est affecté de la valeur 8 et *b* de la valeur 20

On peut alors imaginer une fonction nommée *min_max* qui a plusieurs valeurs de retour et dont l'instruction *return* serait par exemple : return min, max
L'appel de la fonction sera alors lié à une affectation multiple comme par exemple :
nb_min, nb_max = *min_max(nb1, nb2, nb3, nb4, nb5, nb6)*

### 10.7 La fonction RANDOM (chiffre aléatoire)

| Exemple d'utilisation d'une fonction de la librairie random | Commentaires |
|---|---|
| `x=random.randint`$(a, b)$ | On affecte x d'un nombre entier aléatoire compris entre a et b (inclus). |

## 11 Les chaînes de caractères en python

Pour expliquer l'utilisation des chaînes de caractères en python, nous décrirons la notion de tableau, la notion d'objet et l'utilisation de la boucle FOR avec des chaînes de caractères.

### 11.1 Introduction sur les tableaux

Bien que le python n'utilise pas vraiment de tableaux (il peut néanmoins se servir d'une liste comme d'un tableau), il est utile de connaître le principe d'un tableau ainsi que son utilité et sa mise en œuvre. Nous expliquerons ici le tableau de manière très succincte sans aborder la partie allocation de la mémoire (on suppose une taille donnée et fixe).

Un tableau permet, à l'aide d'un seul identifiant (le nom du tableau), de contenir plusieurs données qui ont le même type. Chaque donnée sera localisée au sein du tableau à l'aide d'un indice. L'indice commence à 0 et terminera à un indice qui dépend du nombre de données contenues dans le tableau.
S'il y a N données à stocker dans le tableau, le dernier indice sera N-1.

Descriptif du tableau dans un des questionnaires des olympiades d'informatiques

*Pour manipuler plusieurs éléments avec une seule variable, on utilise un tableau.*
*Les éléments individuels d'un tableau sont indiqués par un index (que l'on écrit entre crochets après le nom du tableau).*
*Le premier élément d'un tableau* ***tab*** *est d'indice 0 et est noté tab[0].*
*Le second est celui d'indice 1 et le dernier est celui d'indice N - 1 si le tableau contient N éléments. Par exemple, si le tableau tab contient les 3 nombres 5, 9 et 12 (dans cet ordre), alors :*

*tab[0]= 5*
*tab[1]= 9*
*tab[2]= 12*

*La longueur est 3, mais l'index le plus élevé est 2.*

### 11.2 Introduction sur la boucle FOR

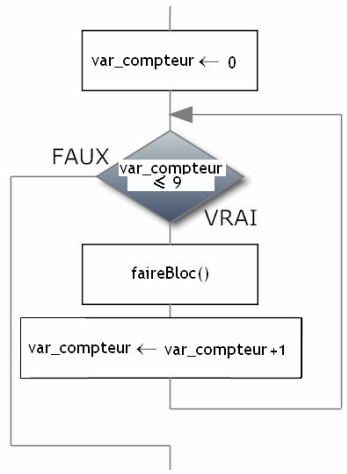
Il est intéressant de connaître le principe général de la boucle FOR avant d'étudier la mise en œuvre de celle-ci en python (qui est parfois un peu particulière).

La boucle FOR est une structure répétitive que l'on utilise en général lorsque le nombre de répétitions à faire est connu au préalable.

Le principe repose sur l'utilisation d'une variable qui servira de compteur afin d'assurer les répétitions souhaitées. Cette variable compteur est initialisée un première fois avant de démarrer, ensuite elle est testée régulièrement afin de déterminer s'il faut répéter le bloc d'actions ou si les répétitions sont terminées. Bien entendu la variable compteur est mise à jour (par pas de 1 en général) à chaque nouvelle exécution du bloc d'actions.

Ci-dessous un exemple pour 10 répétitions du bloc d'actions en commençant à 0, en vérifiant la condition *var_compteur* ≤ 9 et en incrémentant la variable à chaque tour.
Le bloc d'actions est symbolisé par la fonction faireBloc() :

Ordinogramme :




LDA : (anglais)

**for** (*var_compteur* ← 0 **to** 9 **step** 1) {
faireBloc()
}

LDA : (français)

pour *var_compteur* de 0 à 9 faire
faireBloc()
fpour

Descriptif de la « FOR » dans un des questionnaires des olympiades d'informatiques

*Pour répéter du code, par exemple pour parcourir les éléments d'un tableau, on peut utiliser une boucle* ***for****. La notation* ***for*** *(i←a* ***to*** *b* ***step*** *k) représente une boucle qui sera répétée tant que i ≤ b, dans laquelle i commence à la valeur a, et est augmentée de k à la fin de chaque étape.*
*L'exemple suivant calcule la somme des éléments du tableau tab en supposant que sa taille vaut N. La somme se trouve dans la variable sum à la fin de l'exécution de l'algorithme.*

```
sum ← 0
for (i← 0 to N - 1 step 1)
{
sum ← sum + tab[i]
}
```

### 11.3 Les chaînes de caractères en python et la notion d'objet

Les chaînes de caractères également nommées « strings » en anglais s'apparentent dans certains langages à un tableau de caractères avec quelques spécificités particulières ; chaque case du tableau contient alors une donnée du type 'caractère' et tous ces caractères sont placés dans un ordre bien précis pour reconstituer la chaîne de caractères. Il est également courant de voir apparaître un caractère particulier à la fin des « strings » nommé caractère de fin de chaîne.

Les « strings », les tableaux ainsi que les listes et les dictionnaires que nous verrons plus tard, font partie de ce qu'on peut appeler des structures de données.

Cette appellation provient du fait que plusieurs données sont contenues au sein d'un seul élément et que ces données sont structurées suivant la convention adoptée.

Les structures de données présentées ici sont relativement simples dans la mesure où il s'agit simplement de séquences contenant une collection d'éléments.

Python est un langage orienté objet ce qui implique qu'il manipule des variables classiques (associées à un type tels que « entier », « réel » ou « booléen ») mais aussi des objets.
La notion d'objet repose en partie sur le principe de la structure de données, ainsi en python les « strings », les listes et les dictionnaires sont considérés comme des objets.

Les objets sont rattachés à un modèle qui décrit la structure des différentes données associées à ces objets ainsi que les fonctionnalités disponibles. Certains objets sont prédéfinis au sein du langage (comme les «strings », les listes et les dictionnaires en python), d'autres sont définis par le programmeur pour répondre à un besoin.

Un objet étend la notion de variable :

| | | |
|---|---|---|
| Une variable contient une donnée | ↔ | Un objet contient un ensemble de données |
| Une variable est associée à un type | ↔ | Un objet est associé à un modèle |
| On peut faire des opérations sur des variables | ↔ | On peut faire des opérations sur des objets |
| | ↔ | On peut appliquer des méthodes à des objets |

Lorsqu'on crée une nouvelle donnée liée à une variable on dit qu'on déclare une variable.
La variable reçoit alors un nom au moment de sa création.
La variable est alors rattachée à un type (avec ou sans initialisation de donnée).

Lorsqu'on crée une nouvelle collection de données liées à un objet, on dit qu'on instancie un objet. L'instance de l'objet reçoit alors un nom au moment de sa création. L'instance de l'objet (également appelé objet) est alors rattachée à un modèle (avec ou sans initialisation de données).

Le modèle qui définit l'objet précise les attributs de l'objet (identifiants internes qui contiennent des données de l'objet) et les méthodes de l'objet (fonctionnalités internes à l'objet qui sont en quelques sorte des fonctions implémentées pour cet objet).

Un « string » est un objet immutable.
Ça signifie qu'il ne peut être modifié localement.

Autrement dit il faut passer par un affectation pour modifier un « string ».

En python il n'existe pas le type « caractère » mais il est possible de manipuler directement des « strings ».
Contrairement à d'autres langages il n'y a pas de caractère de fin de chaîne au sein des strings en python.

Une instance de l'objet string (que l'on appellera string) peut contenir une collection de caractères (0, 1 ou plus) qui sont ordonnés et accessibles individuellement de la même façon qu'un tableau.
Cependant il n'est pas possible de modifier un seul caractère d'un string individuellement car un string est immutable.

Exemple de mise en œuvre des « strings » (initialisation, concaténation et accès à un caractère)

```
my_str1= 'hello'                    #Lors de sa déclaration un string est initialisé
my_str2= "world"                    #entre " " ou entre ' '
my_str3= ""                         # un string peut-être vide
print  "mettons en oeuvre des strings"
print my_str1
print my_str1+my_str2               # l'opérateur + est la concaténation pour les strings
print my_str2[3]                    # le caractère d'indice 3 de my_str2 sera affiché
                                    # il s'agit du 4ème caractère donc le « l » car l'indice commence à 0
```

Comme indiqué ci-dessus un opérateur qui a un sens particulier peut avoir un autre sens pour des types ou objets différents. L'opérateur « + » qui signifie additionner pour des nombres sert à concaténer (joindre) des strings.

De la même manière l'opérateur « * » permet une répétition pour les strings (plusieurs concaténations successives).

Il est également possible d'accéder à une tranche d'un string ou d'utiliser une fonction de python qui indique la longueur du string :

Autre exemple sur les « strings » (l'opérateur « * », accès à une tranche et la fonction len())

```
my_str1= "hello world"
#my_str1[2]="r"                     #Provoquerait une erreur
my_str2= 'Ni'
my_str3= my_str2*3                  #my_str3 contient alors 'NiNiNi'
my_str2=my_str1[3:7]                #my_str2 contient la tranche de l'indice 3 à l'indice 7 non compris
print my_str1                       #affichera->hello world
print my_str2                       #affichera->lo w
print my_str3,"de longueur :",len(my_str3)   #affichera->NiNiNi de longueur : 6
```

Les tranches (slices) d'un string peuvent être présentés de différentes façons.
Un indice négatif par exemple fait référence à une position en partant de la fin.

Les tranches de « strings »

| | | |
|---|---|---|
| S="spam" | #résultat de l'affichage | #commentaires |
| print S[0] | s | |
| print S[-2] | a | #2ème caractère à partir de la fin |
| print S[1:3] | pa | #de l'index 1 au 3 non compris |
| print S[1:] | pam | #de l'index 1 à la fin |
| print S[:-1] | spa | #du début au dernier caractère non compris |

Il existe des méthodes que l'on peut utiliser sur des objets strings. Une méthode est l'équivalent d'une fonction (avec ou sans paramètre) dédiée à un objet.

Pour appliquer une méthode sur une instance d'un objet, on précisera l'instance de l'objet suivi de « . » suivi du nom de la méthode (avec ou sans paramètre).

Ainsi une instance nommée « my_obj1 » pour laquelle on veut appliquer une méthode sans paramètre nommée « do_this() » sera notée : my_obj1.do_this()

Dans certains cas, la méthode peut modifier l'objet sur lequel elle est appliquée (ce qui ne sera pas le cas pour les strings car ils sont immutables).

Très souvent la méthode renvoie une valeur de retour exploitable (pour l'afficher, pour une affectation, au sein d'une condition,…).

Quelques méthodes de l'objet string

| | #les affichages | #commentaires |
|---|---|---|
| S1="spam55" | | #S1 est une instance de l'objet string |
| S2=S1.capitalize() | | #La méthode capitalize() appliquée sur S1 |
| | | #renvoie un string comme valeur de retour |
| S3="98" | | |
| print S1, S2 | spam55 Spam55 | #S2 comme S1 mais commence par une #majuscule |
| print S1.islower(), S2.islower() | True False | #La méthode islower() indique si tout en minuscule |
| print S1.isdigit(), S3.isdigit() | False True | #La méthode isdigit() indique si que des chiffres |
| print S1[4:].isdigit() | True | #Une méthode peut être appliquée à une tranche |
| S1=S1.replace("55","UR") | | #replace() renvoie le string après remplacement |
| print S1,S1[4:].isdigit() | spamUR False | #S1 a été modifié grâce à l'affectation |

### *11.4 La boucle FOR – utilisation avec des strings*

Python permet de facilement parcourir les différents caractères d'un string avec la boucle FOR. Au sein de la boucle For en python, on précise un identifiant (au choix) qui contiendra successivement les différents caractères du string en cours de parcours.

L'exemple ci après utilise l'identifiant *my_id* pour contenir successivement les différents caractères de S1, nous aurions pu choisir un autre identifiant.

L'identifiant est en fait un string qui ne contient chaque fois qu'un seul caractère.

Mise en œuvre de la For pour un string (comptage du nombre de chiffres)

| S1="spam55" | # affichage | #commentaires |
|---|---|---|
| nb_dig=0 #nombre de digits | | |
| for my_id in S1: | | #my_id contiendra successivement tous les |
| if my_id.isdigit(): | | #caractères de S1 |
| nb_dig=nb_dig+1 | | |
| print S1,"a", nb_dig," chiffres" | spam55 a 2 chiffres | |

# 12 Les listes

## 12.1 Principe des listes

Le principe des listes étend la notion de tableau ; fort connu dans un langage structuré comme le C. Un tableau permet de contenir un ensemble d'éléments de même type.

Chaque élément du tableau est identifiable grâce à un identifiant unique (le nom du tableau) et à sa position au sein du tableau (son indice).

En python, on utilise des listes lorsqu'on veut implémenter un tableau. Il faut cependant être très prudent car l'utilisation des listes en python est très souple, elle permet de faire certaines choses qui ne sont pas recommandées.
Les listes permettent par exemple de stocker des éléments de types différents au sein d'une même liste. Sauf exception, il faut éviter ça !!!

Une liste peut contenir tous types de variables ou d'objet. Une liste peut d'ailleurs elle-même contenir des listes ; ça ne pose pas de problème.

**Il est cependant recommandé de s'arranger pour que chaque liste contienne des éléments de même type.**
**Par exemple tous des entiers, tous des strings, toutes des listes,…**

**• La déclaration d'une liste**

Pour déclarer une liste on donne :
- son nom
- le signe « = »
- la valeur de départ de la liste entre [].
  On sépare les éléments par une virgule.

| #exemples de déclaration | #commentaires |
|---|---|
| my_list_1 = [ ] | #on déclare une liste vide |
| my_list_2 = [1, 2, 3 ] | #on déclare une liste de trois entiers |
| my_list_3 = [0.1, 'a', 3 ] | #on déclare en donnant des éléments de types différents → à éviter |
| my_list_4 = [ [ ], [ ] ] | #on déclare une liste qui contient deux listes vides |
| my_list_5= [ [8,9 ], [9,3 ] ] | #on déclare une liste qui contient deux listes d'entiers |

**• L'affectation d'une liste**

Une liste est un objet, une structure de données.
Lorsqu'on affecte une liste, en supposant que ce ne soit pas une déclaration, on fixe la nouvelle situation de la liste. La situation précédente est alors perdue.

| #exemples d'affectation | #commentaires |
|---|---|
| my_list_2 = [9, 2, 8 ] | #on déclare une liste de trois entiers |
| my_list_2 = [ ] | # les 3 entiers de la déclaration n'existent plus |

```
my_list_5= [ [8,9 ], [9,3 ] ]      #on déclare une liste qui contient deux listes d'entiers
my_list_5= [ 5.2, 6.6 ]            #la liste n'est plus une liste de deux listes d'entiers mais une liste de
                                   #deux nombres réels.
```

**• L'ajout d'un élément à la fin d'une liste**

Pour ajouter un élément à la fin d'une liste on utilise la méthode append().
On écrit le nom de la liste suivi de « . » suivi de « append() ».
On place alors de qu'on veut ajouter entre les parenthèses.

```
#exemples d'ajouts                 #commentaires

my_list_2 = [9, 2, 8 ]             #on déclare une liste de trois entiers
my_list_2.append(54)               # la liste contient alors  [9, 2, 8, 54 ]

my_list_3= []                      #on déclare une liste vide
my_list_3.append(2.2)              #la liste contient alors [2.2]

my_list_5= [ [8,9 ], [9,3 ] ]      #on déclare une liste qui contient deux listes d'entiers
my_list_5.append([0, 8])           #la liste contient alors [ [8,9 ], [9,3 ], [0, 8] ]
```

**• L'affichage d'une liste**

Pour afficher le contenu d'une liste, il suffit d'utiliser « print »

**• Le nombre d'éléments d'une liste**

Le nombre d'éléments d'une liste peut-être obtenu grâce à la fonction « len() »

```
#exemple illustrant len()          #commentaires

my_list_2 = [9, 2, 8 ]             #on déclare une liste de trois entiers
x = len(my_list_2)                 # x contient le nombre d'éléments de my_list_2

print x                            #sera affiché 3
print my_list_2                    #sera affiché [9, 2, 8 ]

my_list_2.append(4)                #La liste contient alors  [9, 2, 8, 4 ]
print len(my_list_2)               # sera affiché 4
```

### *12.2 L'affectation et la mutation d'un objet*

Lorsqu'on fait une affectation, on fait en réalité correspondre l'identifiant à un nouvel objet ; l'identifiant ne fait plus référence à l'ancien objet.

Illustration :

```
#affectation d'un objet            #commentaires

my_list = [9, 2, 8 ]               #L'objet [9, 2, 8 ] est créé, appelons-le OBJET1
```

| | |
|---|---|
| | #L'identifiant « my_list » fait référence à OBJET1 |
| my_list = [8 ,8, 5 ] | #L'objet [8,8,5] est créé, appelons-le OBJET2<br>#L'identifiant « my_list » fait référence à OBJET2<br>#OBJET1 n'est plus référencé par l'identifiant « my_list » |

Lorsqu'on fait une affectation d'un objet1 vers un autre objet2, on dit que l'objet2 fait référence à l'objet1 !

Illustration :

| #affectation d'un objet | #commentaires |
|---|---|
| my_list = [9, 2, 8 ] | #L'objet [9, 2, 8 ] est créé, appelons-le OBJET1<br>#L'identifiant « my_list » fait référence à OBJET1 |
| my_id = my_list | #L'identifiant « my_id » fait référence à l'objet référencé par « my_list »<br>#L'identifiant « my_id » fait référence à OBJET1 |
| my_list = [8 ,8, 5 ] | #L'objet [8,8,5] est créé, appelons-le OBJET2<br>#L'identifiant « my_list » fait référence à OBJET2<br>#OBJET1 n'est plus référencé par l'identifiant « my_list » |
| print my_list, my_id | # il sera affiché « [8 ,8, 5 ] [9, 2, 8 ] » |

Lorsqu'on effectue une mutation à un objet, on modifie des données de cet objet mais on considère que c'est toujours le même objet !

Illustration :

| #affectation d'un objet | #commentaires |
|---|---|
| my_list = [9, 2, 8 ] | #L'objet [9, 2, 8 ] est créé, appelons-le OBJET1<br>#L'identifiant « my_list » fait référence à OBJET1 |
| my_id = my_list | #L'identifiant « my_id » fait référence à l'objet référencé par « my_list »<br>#L'identifiant « my_id » fait référence à OBJET1 |
| my_list[0] = 100 | # OBJET1 [9, 2, 8 ] est muté et devient [100, 2, 8 ]<br>#L'identifiant « my_list » fait toujours référence à OBJET1 |
| print my_list, my_id | # il sera affiché « [100, 2, 8], [100, 2, 8 ]»<br>#la valeur affichée de « my_id » a changé car **fait référence** à un objet<br>#qui a simplement muté ! |

Une autre façon d'avoir un comportement similaire est lorsqu'on applique une méthode sur un objet.

Illustration (avec la méthode « append() »):

| #affectation d'un objet | #commentaires |
|---|---|
| my_list = [9, 2, 8 ] | #L'objet [9, 2, 8 ] est créé, appelons-le OBJET1<br>#L'identifiant « my_list » fait référence à OBJET1 |
| my_id = my_list | #L'identifiant « my_id » fait référence à l'objet référencé par « my_list » |

| | |
|---|---|
| | #L'identifiant « my_id » fait référence à OBJET1 |
| my_list.append(47) | # OBJET1 [9, 2, 8 ] est modifié et devient [9, 2, 8 , 47]<br>#L'identifiant « my_list » fait toujours référence à OBJET1 |
| print my_list, my_id | # il sera affiché « [9, 2, 8 , 47], [9, 2, 8 , 47]»<br>#la valeur affichée de « my_id » a changé car **fait référence** à un objet<br>#qui a simplement été modifié ! |

### *12.3 Exemples d'utilisation de listes en pratique*

• *Exemple 1, introduction de deux cotes dans une liste et affichage de la moyenne :*

```
liste_cotes = []
cote = input("une cote SVP : ")
liste_cotes.append(cote)
cote = input("une cote SVP : ")
liste_cotes.append(cote)
moyenne = (liste_cotes[0] + liste_cotes[1])/2.0
print "votre moyenne vaut : ", moyenne
```

On a commencé par créer une liste vide.
On a ensuite ajouté (avec « append ») deux cotes au sein de notre liste
On calcule la moyenne en utilisant les éléments de la liste grâce à leurs index.
On affiche la moyenne.

• *Exemple 2, moyenne de 5 cotes avec initialisation de la liste :*

```
liste_cotes = [0, 0, 0, 0, 0]
total_cotes = 0
i = 0

#insertion des cotes
indice_cote = 0
while indice_cote < 5 :
    cote = input("une cote SVP : ")
    liste_cotes[indice_cote] = cote
    indice_cote = indice_cote + 1

#calcul de la somme des cotes
i = 0
while i < 5 :
    total_cotes = total_cotes + liste_cotes[i]
    i = i + 1

moyenne = total_cotes /5.0
print "votre moyenne vaut : ", moyenne
```

On peut remarquer que nous avons initialisé la liste avec 5 valeurs.
Cette façon de faire nous a permis d'insérer nos 5 cotes sans utiliser la méthode « append » mais simplement en donnant la valeur à un élément de la liste : liste_cotes[indice_cote] = cote

Un risque de vouloir insérer directement une valeur à un élément de la liste est de tenter d'accéder un élément qui n'existe pas.
Si nous avions tapé « liste_cotes[5] = cote » il y aurait eu une erreur ; en effet, l'indice maximum dans notre exemple est 4.

En résumé, on peut donner une valeur à un élément d'une liste qui a déjà été initialisé mais il faut faire attention à ne pas tenter de faire référence à un élément qui n'est pas existant (dont l'indice est hors de portée).

D'autre part on peut remarquer que nous utilisons une while pour parcourir les 5 éléments de la liste. Nous verrons plus tard qu'une boucle « FOR » permet de simplifier l'itération.

• *Exemple 3, moyenne de n cotes sans initialisation de la liste :*

```
liste_cotes = []
total_cotes = 0
n = 0
i = 0

#lecture de la valeur de n
n = input("indroduisez le nombre de cotes N, SVP : ")

#insertion des cotes
i = 0
while i < n :
    cote = input("une cote SVP : ")
    liste_cotes.append(cote)
    i = i + 1

#calcul de la somme des cotes puis de la moyenne
i = 0
while i < len(liste_cotes) :
    total_cotes = total_cotes + liste_cotes[i]
    i = i + 1
moyenne = total_cotes / (1.0 * len(liste_cotes) )
print "votre moyenne vaut : ", moyenne
```

On demande à l'utilisateur d'introduire la valeur de « n ».
On utilise la valeur de « n » pour s'assurer de lire les cotes et de les placer dans la liste à l'aide de la méthode « append() ».
On parcourt toute la liste pour calculer la somme des cotes puis la moyenne.
On peut voir que l'on n'utilise plus « n » mais la longueur de la liste « len(liste_cotes) ». Cette façon de faire est recommandée car elle fonctionne même si « n » a changé et ne correspond plus au nombre d'éléments de la liste. Il faut juste s'assurer que lorsqu'on parcourt tous les éléments de la liste (dans la boucle), on ne modifie pas le nombre d'éléments de la liste.

• *Exemple 4, trier les éléments d'une liste :*

```
liste_cotes = []
n = 0
i = 0

#lecture de la valeur de n
n = input("indroduisez le nombre de cotes N, SVP : ")

#insertion des cotes
i = 0
while i < n :
   cote = input("une cote SVP : ")
   liste_cotes.append(cote)
   i = i + 1

#trier la liste
liste_cotes.sort()

#afficher la liste
print liste_cotes
```

Une fois les « n » cotes introduites, la méthode « sort() » trie la liste dans l'ordre croissant.

Une explication exhaustive des listes n'est pas l'objet de ce cours ; il y a de nombreuses fonctionnalités que nous ne détaillerons pas.
Cependant, voici un petit aperçu de fonctionnalités supplémentaires :

- Comme pour les « strings », les listes peuvent être manipulées par tranches
  Exemple : print liste_cotes[2 :]
- Deux listes peuvent être combinées en une seule avec l'opérateur « + »
- On peut avoir des listes à deux dimensions ou plus.
- Il existe des méthodes telles que extend(), remove(), reverse(), …
- Certaines fonctions de python acceptent une liste comme argument.
  Exemples : len(), sorted(), reversed(),…

### *12.4 La boucle FOR, utilisation avec des listes*

Il existe une fonction en python qui est capable de créer une liste ordonnée de nombres entiers, il s'agit de la fonction « range() ».
Par exemple lorsqu'on écrit : print range(5) # Il apparaît [0, 1, 2, 3, 4]

Cette fonction est couramment utilisée pour créer une boucle « FOR ».
Pour rappel, une boucle « FOR » est une boucle que l'on utilise lorsqu'on connaît à l'avance le nombre de répétitions.

Si on veut par exemple afficher 5 fois « bonjour », on écrira :

```
for i in range(5):
   print "bonjour"
```

• *Exemple, introduction de cinq cotes dans une liste et affichage de la moyenne :*

```
liste_cotes = []
cote = 0
total_cotes = 0
moyenne = 0
for i in range(5):
    cote = input("une cote SVP : ")
    liste_cotes.append(cote)
for i in range(len(liste_cotes)):
    total_cotes = total_cotes + liste_cotes[i]
moyenne =  1.0 * total_cotes / len(liste_cotes)
print "votre moyenne est de : ", moyenne
```

Dans certains cas on souhaite utiliser une liste ordonnée de nombres mais qui ne commencent pas forcément à 0.
Dans ce cas on peut donner deux paramètres à la fonction « range() », exemple :

```
liste_nb = range(1, 11)          # la liste vaut alors [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

• *Exemple, affichage de la table de multiplication (jusqu'à 10) d'un nombre :*

```
nombre = 0
print "table de multiplication "
nombre = input("veuillez introduire votre nombre : ")
for facteur in range(1, 11):
    print facteur, " * ", nombre, " = ", facteur * nombre
```

Remarque : Il est possible d'utiliser la fonction range() pour manipuler directement une liste sans la boucle « FOR ». Exemple : liste_nombres = range(100)

# 13 Les fichiers

Des fonctionnalités sont prévues en python pour créer, lire et écrire dans des fichiers.
Il existe en python des objets de type <file> qui vont permettre de manipuler directement des fichiers.
On créera alors un fichier en utilisant la fonction open().

## *13.1 Création d'un fichier*

On commence par ouvrir le fichier en python.

Pour ouvrir le fichier on utilise la fonction *open* et on donne le mode d'ouverture
'r' pour une ouverture en lecture
'w' pour une ouverture en écriture (le contenu du fichier est écrasé)
'a' pour une ouverture en mode ajout

| Exemple d'ouverture d'un fichier | Commentaires |
|---|---|
| my_file = open("data_base.txt","r") | On ouvre le fichier data_base.txt en mode lecture et on lui donne le nom my_file |

Le nom que l'on donne à l'objet qui représente le fichier (« my_file » dans l'exemple ci-dessus) va nous permettre de faire appel à des méthodes pour manipuler le fichier.

### *13.2 Ecriture dans un fichier, la méthode write()*

Pour pouvoir écrire dans un fichier on utilise la méthode write() sur un objet de type fichier. L'objet doit donc avoir été créé préalablement avec la fonction open().
Il est nécessaire d'avoir un fichier dont le mode d'ouverture est 'w' (écriture) ou 'a' (ajout) pour lui appliquer la méthode write().
Tenter d'appliquer la méthode write() à un fichier ouvert en mode lecture provoquera une erreur !

Pour écrire dans un fichier :

| Exemple d'écriture dans un fichier | Commentaires |
|---|---|
| my_file = open("data_base.txt","w")<br>my_file .write("texte à mettre dans le fichier")<br><br><br>my_file.close() | On ouvre le fichier data_base.txt en mode écriture<br>On écrit « texte à mettre dans le fichier » dans le fichier data_base.txt (en écrasant le contenu précédant)<br><br>On ferme le fichier |

Pour ajouter une information dans un fichier :

| Exemple d'ajout dans un fichier | Commentaires |
|---|---|
| my_file = open("data_base.txt","a")<br>my_file .write("texte à mettre dans le fichier")<br><br><br>my_file.close() | On ouvre le fichier data_base.txt en mode ajout<br>On écrit « texte à mettre dans le fichier » dans le fichier data_base.txt (à la suite du contenu précédant)<br><br>On ferme le fichier |

Dans les deux exemples donnés ci-dessus, si le fichier n'existe pas dans le dossier courant (celui qui contient votre fichier python) il sera créé automatiquement.

### *13.3 Fermeture d'un fichier, la méthode close()*

Les deux exemples du point précédent montraient bien qu'on terminait la manipulation du fichier à l'aide de la méthode close().
Ce qui permet à python de libérer son emprise sur le fichier.

### 13.4 La gestion des répertoires des fichiers

Il faut savoir que la gestion des répertoires (les dossiers) se fait par défaut à partir du dossier courant (celui qui contient votre fichier python).
Si on demande de créer un dossier, celui-ci sera donc créé par défaut au sein du dossier qui contient votre programme python.

• Création d'un dossier

Pour créer un nouveau dossier, on utilisera les fonctionnalités de l' « operating system ». On importera la librairie python « os » afin de pouvoir utiliser la méthode mkdir() qui permettra de créer un nouveau répertoire (mkdir ↔ « make directory »).

On écrira par exemple :

```
import os
os.mkdir("mon_dossier1")
```

Il apparaîtra alors un nouveau dossier nommé « mon_dossier1 » au sein du dossier courant. Il est important de savoir que si ce dossier existait déjà, les lignes ci-dessus provoqueraient une erreur !

Une façon de s'assurer de ne pas avoir une erreur lors de la création d'un dossier est de vérifier si un dossier spécifique existe. Pour ce faire on utilise la méthode « isdir() » sur un objet qui représente le dossier courant.

On peut tester par exemple :

```
import os
if os.path.isdir("mon_dossier1"):
    print "dossier existe"
else:
    print "dossier n'existe pas"
```

On remarquera alors un affichage qui indique si le dossier spécifié existe.

Pour créer un dossier sans risque d'erreur :

```
import os
if not os.path.isdir("mon_rep"):
        os.mkdir("mon_rep")
```

On a alors créé le dossier « mon_rep » que dans le cas où celui-ci n'existe pas.

Il est également possible de donner comme paramètre aux méthodes isdir() et mkdir() non pas le nom d'un répertoire mais tout un chemin.

On peut par exemple écrire :

```
import os
os.mkdir("dos_glob/sous_dos1/repertoire5")
```

Lors de l'exécution, dans la mesure où « dos_glob » existe dans le dossier courant et qu'il contient un dossier nommé « sous_dos1 », il sera créé un nouveau dossier « repertoire5 » au sein de « sous_dos1 ».

Dans les autres cas, l'exécution provoquera une erreur.

On peut aussi écrire :

```
import os
if not os.path.isdir("dos_glob/sous_dos1/repertoire5") :
   print  "le chemin existant n'est pas complet"
   print  "au minimum un dossier n'existe pas"
else :
   print  "le chemin existant est complet"
```

### *13.5 Changement du répertoire courant*

• Chemins relatifs et absolus

Les exemples précédents montraient comment créer des dossiers vis-à-vis du répertoire courant (celui qui contient votre fichier python).

D'autre part, lorsqu'on donnait un chemin qui reflète l'organisation interne des dossiers on précisait un chemin relatif ; ce qui veut dire que l'on précisait un chemin relatif vis-à-vis du répertoire courant.

Un gros avantage de travailler avec un chemin relatif est d'assurer une portabilité du code. Ainsi, même si on déplace le répertoire courant, le chemin relatif reste valable.

Il est également possible de manipuler ce qu'on appelle des chemins absolus.
Voici un exemple de chemin absolu : "C:\Documents and Settings\All Users\Bureau"
C'est un chemin qui commence à partir de la racine.

• La méthode chdir()

La méthode chdir() signifie « change directory » et permet de changer le répertoire considéré comme le répertoire courant.
Au départ le répertoire courant est celui qui contient votre fichier python.

Lorsqu'on va préciser un chemin comme paramètre de chdir(), le programme considèrera le répertoire mentionné comme le dossier courant.
Il faut bien sûr que le chemin donné corresponde à un chemin déjà existant.

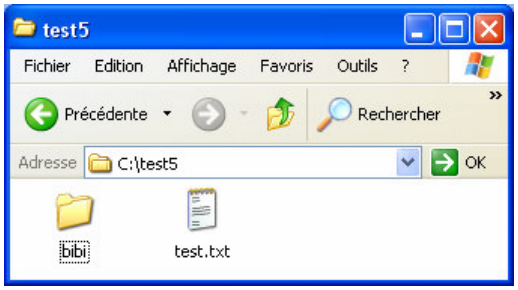
Ensuite, tous les dossiers et fichiers créés, s'ils utilisent des chemins relatifs, seront créés à un emplacement relatif au dernier dossier courant spécifié par chdir().

Exemple :

```
import os
os.mkdir("C:/test5")
os.chdir("C:/test5")
os.mkdir("bibi")
my_file = open("test.txt","w")
my_file .write("texte pour voir")
my_file.close()
```

Dans cet exemple, un nouveau dossier « test5 » sera créé au sein de « C : » (on suppose qu'il n'existait pas).
Ce dossier sert ensuite de dossier courant.
On crée alors un directory « bibi » relatif au dossier courant.
On crée ensuite un fichier « test.txt » au sein du dossier courant.

La capture d'écran ci-dessous montre alors le résultat de l'exécution.




### *13.6 Lecture d'un fichier*

Pour lire un fichier, il faut l'ouvrir en mode lecture et accéder au contenu à l'aide de la méthode read(). On peut ensuite affecter à une variable la valeur de retour de la méthode read().

Exemple de lecture et d'affichage d'un fichier :

```
import os
my_file = open("test.txt","r")
contenu = my_file.read()
print contenu
my_file.close()
```

La variable « contenu » de notre exemple sera en fait une variable de type « string » qui contient sous forme d'une chaîne de caractère l'ensemble du fichier.

# Exercices

## 1 Prise en main

Les notions associées sont :
- La variable, sa déclaration et son affectation
- L'affichage de texte, d'une variable d'un type
- L'utilisation de délais
- Opérateurs, incrémentation, décrémentation
- La lecture d'une donnée avec input

### *1.1 Hello World*

| | |
|---|---|
| Affichez "Hello World" sur écran | Difficulté : 1 |

### *1.2 Affichage d'une variable initialisée*

| | |
|---|---|
| Commencez par déclarer une variable nommée *ma_variable* et initialisez sa valeur à 20<br>Affichez ensuite la valeur de *ma_variable* | Difficulté : 1 |

### *1.3 Affichage mixte et initialisation d'une variable*

| | |
|---|---|
| Commencez par déclarer une variable nommée *ma_variable* et initialisez sa valeur à 100<br>Affichez ensuite "La valeur est : " suivi de la valeur de *ma_variable* | Difficulté : 1 |

### *1.4 Affichage mixte et plusieurs variables*

| | |
|---|---|
| Commencez par déclarer trois variables nommées *nb1*, *nb2* et *somme* et initialisez respectivement leurs valeurs à 10, 20 et 0.<br>Utilisez ensuite l'opérateur + pour affecter à la variable *somme* la somme de *nb1* et *nb2*.<br>Affichez ensuite "Si on ajoute "<br>suivi de la valeur de *nb1*<br>suivi de " et "<br>suivi de la valeur de *nb2*<br>suivi de " on obtient "<br>suivi de la valeur de *somme* | Difficulté : 1 |

### *1.5 Somme de deux nombres introduits par l'utilisateur*

| | |
|---|---|
| Commencez par déclarer trois variables nommées *nb1*, *nb2* et *somme* et initialisez les à 0.<br>Utilisez la fonction input pour demander à l'utilisateur d'introduire la valeur de *nb1*.<br>Utilisez la fonction input pour demander à l'utilisateur d'introduire la valeur de *nb2*.<br>Affectez la variable *somme* de la somme des deux nombres introduits par l'utilisateur.<br>Affichez un message convivial qui informe l'utilisateur de la valeur des deux nombres introduits ainsi que leur somme. | Difficulté : 2 |

### 1.6 Affichage mixte et affichage de type

| | |
|---|---|
| Commencez par déclarer trois variables nommées *var_entier*, *var_reel* et *var_bool*.<br>Affectez les respectivement de 5 , 5.0 et True<br>Affichez ensuite la valeur de *var_entier* et son type<br>Affichez ensuite la valeur de *var_reel* et son type<br>Affichez ensuite la valeur de *var_bool* et son type<br>Qu'observez-vous ? | Difficulté : 1 |

### 1.7 Décompte 5 secondes

| | |
|---|---|
| Commencez par déclarer une variable *sec* et initialisez la à 5.<br>Affichez "Bonjour, décompte dans 3 secondes"<br>Attendez 3 secondes<br>Répétez ensuite 5 fois de suite (simple copier/coller des lignes) les actions suivantes :<br>Afficher "il reste " suivi de la valeur de *sec* suivi de "s. "<br>Attendre une seconde<br>Affichez "Décompte terminé" | Difficulté : 1 |

### 1.8 Différence de deux nombres introduits par l'utilisateur

| | |
|---|---|
| Commencez par déclarer trois variables nommées *nb1*, *nb2* et *diff* et initialisez les à 0.<br>Utilisez la fonction input pour demander à l'utilisateur d'introduire la valeur de *nb1*.<br>Utilisez la fonction input pour demander à l'utilisateur d'introduire la valeur de *nb2*.<br>Affectez la variable *diff* de la différence des deux nombres introduits par l'utilisateur.<br>Affichez un message convivial qui informe l'utilisateur de la valeur des deux nombres introduits ainsi que leur différence | Difficulté : 2 |

### 1.9 Table d'un nombre introduit par l'utilisateur

| | |
|---|---|
| Le programme consiste à demander à l'utilisateur d'introduire un nombre entre 1 et 9 et de fournir la table de multiplication de ce nombre.<br>Imaginons par exemple que l'utilisateur introduise le nombre 5, il sera affiché :<br>1 fois 5 = 5<br>2 fois 5 = 10<br>…<br>10 fois 5 = 50<br>On remarque que 10 lignes sont affichées les 10 premiers multiples de 5.<br>Commencez par déclarer trois variables nommées *multiple*, *nb* et *result* .<br>Initialisez *multiple* à 1 et les autres variables à 0.<br>Demandez à l'utilisateur un nombre entre 1 et 9 (sans vérifier ici si c'est correct).<br>Tentez de trouvez les lignes d'actions qui seront répétées 10 fois de manière à afficher les 10 premiers multiples du nombre introduit par l'utilisateur.<br>Il faut que l'ensemble de ces lignes soit identique pour constituer un bloc d'actions que nous pourrons plus tard lier à une structure répétitive. | Difficulté : 2 |

## 2 La structure de choix

Les notions associées sont :
- Les notions vue en 1.) Prise en main
- La structure de choix simple
- L' opérateur '/' sur entiers

### 2.1 Minimum de deux nombres

| | |
|---|---|
| Commencez par déclarer trois variables : une variable pour le 1er nombre, une variable pour le 2ème nombre et une variable qui contiendra le plus petit des deux nombres.<br>Demandez à l'utilisateur d'introduire la valeur du 1er nombre.<br>Demandez à l'utilisateur d'introduire la valeur du 2ème nombre.<br>Affectez à votre autre variable le plus petit des deux nombres introduits par l'utilisateur.<br>Affichez la valeur des 2 nombres et un message informant la valeur du plus petit nombre. | Difficulté : 2 |

### 2.2 Addition ou soustraction de deux nombres

| | |
|---|---|
| Commencez par déclarer cinq variables : une variable pour le 1er nombre, une variable pour le 2ème nombre et une variable pour la somme, une pour la différence et une pour le choix.<br>Demandez à l'utilisateur d'introduire la valeur du 1er nombre.<br>Demandez à l'utilisateur d'introduire la valeur du 2ème nombre.<br>Affichez ensuite "Tapez 1 pour + ou n'importe quoi pour -" en mémorisant le choix.<br>Affichez ensuite avec un message convivial fournissant :<br>la somme des deux nombres si « 1 » a été introduit<br>la différence des deux nombres si « 1 » n'a pas été introduit | Difficulté : 2 |

### 2.3 Produit de deux nombres différents

| | |
|---|---|
| Demandez à l'utilisateur d'introduire la valeur du 1er nombre.<br>Demandez à l'utilisateur d'introduire la valeur du 2ème nombre en précisant qu'il doit être différent du 1er nombre.<br>Si les deux nombres sont différents, affichez un message convivial indiquant le produit.<br>Si les deux nombres sont identiques, dites poliment à l'utilisateur qu'il n'a pas compris. | Difficulté : 2 |

### 2.4 Cinq fois « Bonjour » mais trois vitesses possibles

| | |
|---|---|
| L'idée est qu'il apparaisse "Bonjour" cinq fois de suite mais avec un délai fixe entre les affichages successifs. Cependant ce délai doit être un choix de l'utilisateur qui commence par voir sur écran :<br>Veuillez entrer la vitesse : normal (tapez 1) – lent (tapez 2) – très lent (tapez 3)<br>Le choix de l'utilisateur doit bien sûr être mémorisé, s'il s'agit de :<br>1 c'est qu'il faut un délai de 1 seconde entre chaque "Bonjour"<br>2 c'est qu'il faut un délai de 2 secondes entre chaque "Bonjour"<br>3 c'est qu'il faut un délai de 3 secondes entre chaque "Bonjour"<br>Une autre valeur, dites poliment à l'utilisateur qu'il n'a pas compris. | Difficulté : 3 |

### 2.5 Tapez 1 pour l'addition et 2 pour la soustraction

| | |
|---|---|
| L'idée est que l'utilisateur introduise deux nombres puis choisisse l'opération + ou -<br>Demandez à l'utilisateur d'introduire la valeur du 1er nombre.<br>Demandez à l'utilisateur d'introduire la valeur du 2ème nombre.<br>L'utilisateur doit ensuite voir apparaître :<br>Veuillez entrer votre choix : addition (tapez 1) – soustraction (tapez 2)<br>Le choix de l'utilisateur doit bien sûr être mémorisé, s'il s'agit de :<br>1 c'est qu'il faut lui afficher le résultat de l'addition<br>2 c'est qu'il faut lui afficher le résultat de la soustraction<br>Une autre valeur, dites poliment à l'utilisateur qu'il n'a pas compris. | Difficulté : 3 |

### 2.6 Maximum de deux nombres

| | |
|---|---|
| Demandez à l'utilisateur d'introduire deux nombres.<br>Affichez la valeur des 2 nombres et un message informant la valeur du plus grand nombre. | Difficulté : 2 |

### 2.7 Minimum ou maximum de deux nombres

| | |
|---|---|
| Demandez à l'utilisateur d'introduire deux nombres.<br>Affichez ensuite "Tapez 1 pour MIN ou n'importe quoi pour MAX" en mémorisant le choix.<br>Affichez ensuite avec un message convivial fournissant :<br>Le minimum des deux nombres si « 1 » a été introduit<br>Le maximum des deux nombres si « 1 » n'a pas été introduit | Difficulté : 2 |

### 2.8 Tapez 1 pour le minimum et 2 pour le maximum

| | |
|---|---|
| L'idée est que l'utilisateur introduise deux nombres puis choisisse MIN ou MAX<br>Demandez à l'utilisateur d'introduire deux nombres.<br>L'utilisateur doit ensuite voir apparaître :<br>Veuillez entrer votre choix : MIN (tapez 1) – MAX (tapez 2)<br>Le choix de l'utilisateur doit bien sûr être mémorisé, s'il s'agit de :<br>1 c'est qu'il faut lui afficher le plus petit des deux nombres<br>2 c'est qu'il faut lui afficher le plus grand des deux nombres<br>Une autre valeur, dites poliment à l'utilisateur qu'il n'a pas compris. | Difficulté : 3 |

### 2.9 Table d'un nombre (à un chiffre) introduit par l'utilisateur

| | |
|---|---|
| L'idée est que l'utilisateur introduise un nombre compris entre 1 et 9 afin que le programme puisse afficher la table de ce nombre (les 10 premiers multiples).<br>Cependant on souhaite vérifier que le nombre introduit est bien compris entre 1 et 9.<br>Vérifiez que ce nombre est bien compris entre 1 et 9 sans utiliser d'opérateur logique.<br>L'utilisateur doit voir apparaître :<br>Veuillez introduire un nombre entre 1 et 9 :<br>Le choix de l'utilisateur doit bien sûr être mémorisé, s'il s'agit de :<br>Un nombre entre 1 et 9, affichez la table de ce nombre.<br>Une autre valeur, dites poliment à l'utilisateur qu'il n'a pas compris. | Difficulté : 3 |

### 2.10 Différence positive de deux nombres

| | |
|---|---|
| Demandez à l'utilisateur d'introduire deux nombres.<br>Affichez ensuite la différence de deux nombres mais en vous assurant de faire le plus grand moins le plus petit (pour que le résultat soit positif ou nul). | Difficulté : 2 |

### 2.11 Maximum de trois nombres

| | |
|---|---|
| Demandez à l'utilisateur d'introduire trois nombres.<br>Affichez ensuite, sans utiliser d'opérateurs logiques, le plus grand parmi ces trois nombres. | Difficulté : 3 |

### 2.12 Parité d'un nombre

| | |
|---|---|
| En exploitant le fait que l'opérateur '/' donne le quotient de la division (et non le résultat exact) lorsque les nombres sont entiers, essayez d'afficher la parité d'un nombre entier introduit par l'utilisateur.<br>Astuce : Diviser un nombre pair par 2 tombe juste (le résultat fois 2 redonne le nombre). | Difficulté : 2 |

### 2.13 Trier du plus petit au plus grand

| | |
|---|---|
| Demandez à l'utilisateur d'introduire trois nombres.<br>Affichez ensuite, sans utiliser d'opérateurs logiques, le trois nombres triés du plus petit au plus grand. | Difficulté : 3 |

## 3 Les opérateurs logiques

Les notions associées sont :
- Les notions vue en 1.) Prise en main
- Les notions vue en 2.) La structure de choix
- Les opérateurs logiques

### 3.1 Trier du plus petit au plus grand

| | |
|---|---|
| Demandez à l'utilisateur d'introduire trois nombres.<br>Affichez ensuite, en exploitant les opérateurs logiques, le trois nombres triés du plus petit au plus grand. | Difficulté : 3 |

### 3.2 Maximum de trois nombres

| | |
|---|---|
| Demandez à l'utilisateur d'introduire trois nombres.<br>Affichez ensuite, en exploitant les opérateurs logiques, le plus grand parmi ces trois nombres. | Difficulté : 3 |

### 3.3 Table d'un nombre (à un chiffre) introduit par l'utilisateur

| | |
|---|---|
| L'idée est que l'utilisateur introduise un nombre compris entre 1 et 9 afin que le programme puisse afficher la table de ce nombre (les 10 premiers multiples).<br>Cependant on souhaite vérifier que le nombre introduit est bien compris entre 1 et 9.<br>Vérifiez que ce nombre est bien compris entre 1 et 9 en exploitant les opérateurs logiques.<br>L'utilisateur doit voir apparaître :<br>Veuillez introduire un nombre entre 1 et 9 :<br>Le choix de l'utilisateur doit bien sûr être mémorisé, s'il s'agit de :<br>Un nombre entre 1 et 9, affichez la table de ce nombre.<br>Une autre valeur, dites poliment à l'utilisateur qu'il n'a pas compris. | Difficulté : 3 |

### 3.4 Produit de deux nombres différents (utilisation de « not »)

| | |
|---|---|
| Demandez à l'utilisateur d'introduire la valeur du 1er nombre.<br>Demandez à l'utilisateur d'introduire la valeur du 2ème nombre en précisant qu'il doit être différent du 1er nombre.<br>Si les deux nombres sont différents, affichez un message convivial indiquant le produit.<br>Si les deux nombres sont identiques, dites poliment à l'utilisateur qu'il n'a pas compris.<br>Une contrainte supplémentaire, vous devez utiliser l'opérateur « not » au sein de votre condition. | Difficulté : 2 |

### 3.5 Prix du billet suivant l'age de l'utilisateur

| | |
|---|---|
| Demandez à l'utilisateur d'introduire son age et d'introduire le prix du billet.<br>Si son age est inférieur à 24 ans ou que son age est supérieur à 65 ans il bénéficie d'une réduction de 20%.<br>Affichez le prix qu'il devra effectivement payer. | Difficulté : 2 |

## 4 La structure répétitive « tant que »

Les notions associées sont :
- Les notions vue en 1.) Prise en main
- Les notions vue en 2.) La structure de choix
- Les notions vue en 3.) Les opérateurs logiques
- La structure répétitive « tant que »

### 4.1 Cinq fois « Bonjour »

| | |
|---|---|
| L'idée est qu'il apparaisse "Bonjour" cinq fois de suite mais avec un délai de 1 seconde entre les affichages successifs. Il faut éviter de faire des copier/coller de lignes équivalentes et utiliser la structure répétitive « tant que » pour gérer les répétitions souhaitées | Difficulté : 3 |

### 4.2 Tapez 1 pour l'addition et 2 pour la soustraction

| | |
|---|---|
| L'idée est que l'utilisateur introduise deux nombres puis choisisse l'opération + ou -<br>Demandez à l'utilisateur d'introduire la valeur du 1er nombre.<br>Demandez à l'utilisateur d'introduire la valeur du 2ème nombre.<br>L'utilisateur doit ensuite voir apparaître :<br>Veuillez entrer votre choix : addition (tapez 1) – soustraction (tapez 2)<br>Le choix de l'utilisateur doit bien sûr être mémorisé, s'il s'agit de :<br>1 c'est qu'il faut lui afficher le résultat de l'addition<br>2 c'est qu'il faut lui afficher le résultat de la soustraction<br>Une autre valeur, dites poliment à l'utilisateur qu'il n'a pas compris et demandez à nouveau d'introduire un choix valide (et ce tant que nécessaire). | Difficulté : 3 |

### 4.3 Cinq fois « Bonjour » mais trois vitesses possibles

| | |
|---|---|
| L'idée est qu'il apparaisse "Bonjour" cinq fois de suite mais avec un délai fixe entre les affichages successifs. Cependant ce délai doit être un choix de l'utilisateur qui commence par voir sur écran :<br>Veuillez entrer la vitesse : normal (tapez 1) – lent (tapez 2) – très lent (tapez 3)<br>Le choix de l'utilisateur doit bien sûr être mémorisé, s'il s'agit de :<br>1 c'est qu'il faut un délai de 1 seconde entre chaque "Bonjour"<br>2 c'est qu'il faut un délai de 2 secondes entre chaque "Bonjour"<br>3 c'est qu'il faut un délai de 3 secondes entre chaque "Bonjour"<br>Une autre valeur, dites poliment à l'utilisateur qu'il n'a pas compris et demandez à nouveau d'introduire une vitesse valide (et ce tant que nécessaire).<br>Attention, il faut éviter de faire des copier/coller de lignes équivalentes et utiliser la structure répétitive « tant que » pour gérer les répétitions souhaitées. | Difficulté : 4 |

### 4.4 Table d'un nombre (à un chiffre) introduit par l'utilisateur

| | |
|---|---|
| L'utilisateur doit voir apparaître :<br>Veuillez introduire un nombre entre 1 et 9 :<br>Le choix de l'utilisateur doit bien sûr être mémorisé, s'il s'agit de :<br>Un nombre entre 1 et 9, affichez la table de ce nombre.<br>Une autre valeur, dites poliment à l'utilisateur qu'il n'a pas compris et demandez à nouveau d'introduire un choix valide (et ce tant que nécessaire).<br>Attention, il faut éviter de faire des copier/coller de lignes équivalentes et utiliser la structure répétitive « tant que » pour gérer la table complète. | Difficulté : 4 |

### 4.5 Décompte du nombre de secondes introduit par l'utilisateur

| | |
|---|---|
| Demandez à l'utilisateur d'introduire un nombre qui correspondra au nombre de secondes souhaitées pour le décompte.<br>Démarrez ensuite un décompte d'autant de secondes que le nombre qu'a introduit l'utilisateur. | Difficulté : 3 |

### 4.6 Décompte considérant les minutes et les secondes

| | |
|---|---|
| Demandez à l'utilisateur d'introduire une valeur de départ pour les minutes et les secondes.<br>Démarrez ensuite un décompte démarrant de cette valeur.<br>Pour éviter d'attendre trop longtemps pour les tests, vous pouvez par exemple, temporairement, remplacer les délais de 1s par des délais de 0,1 s. | Difficulté : 4 |

### 4.7 Produit de deux nombres différents

| | |
|---|---|
| Demandez à l'utilisateur d'introduire la valeur du 1er nombre.<br>Demandez à l'utilisateur d'introduire la valeur du 2ème nombre en précisant qu'il doit être différent du 1er nombre.<br>Si le 2ème nombre est identique au 1er, dites poliment à l'utilisateur qu'il n'a pas compris et demandez à nouveau d'introduire un nombre différent du 1er (et ce tant que nécessaire). | Difficulté : 3 |

### 4.8 Attente d'un nombre pair

| | |
|---|---|
| Demandez à l'utilisateur d'introduire un nombre pair.<br>Si le nombre est pair affichez cette information sur écran.<br>Si le nombre, n'est pas pair, dites poliment à l'utilisateur qu'il n'a pas compris et demandez à nouveau d'introduire un nombre pair (et ce tant que nécessaire). | Difficulté : 3 |

### 4.9 Ping-Pong

| | |
|---|---|
| L'idée est que l'équipe 1 va jouer contre l'équipe 2 au tennis de table.<br>L'utilisateur doit voir apparaître :<br>Si l'équipe 1 marque tapez 1, si l'équipe 2 marque tapez 2<br>Au fur et à mesure de la partie le score (X-Y) doit se mettre à jour (X représente les points de l'équipe 1 et Y les points de l'équipe 2) et ce tant que la partie n'est pas terminée.<br>Si l'utilisateur n'a pas enfoncé un choix correct (1 ou 2), il faut lui demander de recommencer son choix.<br>Les points s'incrémentent chaque fois d'une unité jusqu'à ce que la partie soit terminée.<br>La partie se termine dés qu'une équipe atteint 11 points avec au moins deux points d'écarts sur l'autre équipe.<br>Une fois la partie terminée, vous devez afficher sur écran le nom du vainqueur. | Difficulté : 4 |

### 4.10 Ping-Pong et 1, 2 ou 3 sets gagnants

| | |
|---|---|
| Le principe est le même que l'exercice précédent mais l'utilisateur doit commencer par préciser le nombre de sets gagnants souhaités (1, 2 ou 3). Si l'utilisateur n'a pas enfoncé un choix correct (1, 2 ou 3), il faut lui demander de recommencer son choix.<br>Une fois le nombre de sets gagnants introduits, il faudra mettre à jour le score actuel en précisant le nombre de sets gagnés pour chaque équipe et le score du set en cours.<br>A la fin de chaque set il faudra indiquer le vainqueur du set et mettre à jour les sets gagnés de chaque équipe.<br>Une fois qu'une équipe a atteint le nombre de sets à gagner, il faudra indiquer l'équipe qui a gagné le match. | Difficulté : 5 |

### 4.11 Décomposition d'un nombre en produit de nombres premiers

| | |
|---|---|
| Demandez à l'utilisateur d'introduire un nombre compris entre 1 et 30.<br>Si le nombre introduit n'est pas un entier entre 1 et 30, dites poliment à l'utilisateur qu'il n'a pas compris et demandez à nouveau d'introduire le nombre (et ce tant que nécessaire).<br>Une fois qu'un nombre valide a été introduit, tentez de décomposez ce nombre en produit de nombres premiers sachant que les nombres premiers compris entre 1 et 30 sont :<br>2, 3, 5, 7, 11, 13, 17, 19, 23, 29. Essayez de résoudre cet exercice avec les outils actuels. | Difficulté : 5 |

### 4.12 Somme de n nombres

| | |
|---|---|
| Demandez à l'utilisateur d'introduire le nombre de nombres dont il vaut calculer la somme.<br>Stockez ce nombre dans une variable n.<br>Demandez ensuite à l'utilisateur d'introduire les n nombres dont on doit calculer la somme.<br>Affichez les n nombres et leur somme. | Difficulté : 3 |

### 4.13 Devinez le nombre introduit par l'utilisateur précédent

| | |
|---|---|
| Imaginons un programme destiné à un jeu « devinez le nombre que l'utilisateur précédent a introduit». Le programme se déroule en deux étapes :<br>- la 1ère étape le 1er utilisateur introduit un nombre entre 1 et 100<br>- La 2ème étape le 2ème utilisateur tente de trouver ce nombre avec le minimum d'essais. A chaque essai le programme renvoie « trop petit » ou « trop grand » selon le cas. Dés que le nombre est correct il est indiqué en combien d'essais il a été trouvé. | Difficulté : 4 |

## 5 Fonction sans paramètre

Les notions associées sont : - Les notions vue en 1.), 2.), 3.), et 4.)
- Les notions de base sur les fonctions sans paramètre

### 5.1 Une fonction pour Hello World

| | |
|---|---|
| Définissez une fonction nommée fct_HW dont le rôle est d'afficher "Hello World" sur écran<br>Appelez ensuite trois fois de suite la fonction au sein du programme principal. | Difficulté : 2 |

### 5.2 Une fonction decompte_5s

| | |
|---|---|
| Définissez une fonction nommée decompte_5s dont le rôle est de faire apparaître un décompte de 5 à 0 par intervalles de 1seconde.<br>Affichez ensuite « ça va commencer … » puis appeler la fonction.<br>Affichez encore« et encore un décompte… » et appeler à nouveau la fonction. | Difficulté : 3 |

### 5.3 Une fonction affiche_20_facteurs_table7

| | |
|---|---|
| Définissez une fonction nommée affiche_20_facteurs_table7 dont le rôle est de faire apparaître la table de multiplication de 7 allant de 1 à 20.<br>Utilisez ensuite cette fonction au sein de votre programme principal pour s'assurer de son bon fonctionnement. | Difficulté : 3 |

### 5.4 Trois fonctions bonjour_5

| | |
|---|---|
| Définissez trois fonctions proches mais pourtant différentes. Les trois fonctions doivent afficher « Bonjour » 5 fois de suite mais l'intervalle de temps entre chaque affichage n'est pas identique.<br>Les trois fonctions sont donc :<br>bonjour5_norm() → Affichage des « Bonjour » à vitesse normale (délais de 1s.)<br>bonjour5_lent() → Affichage des « Bonjour » à vitesse lente (délais de 2s.)<br>bonjour5_tres_lent() → Affichage des « Bonjour » à vitesse très lente (délais de 3s.)<br>Ensuite l'utilisateur doit voir apparaître sur écran :<br>Veuillez entrer la vitesse : normal (tapez 1) – lent (tapez 2) – très lent (tapez 3)<br>Le choix de l'utilisateur doit bien sûr être mémorisé, s'il s'agit de :<br>1 c'est qu'il faut un délai de 1 seconde entre chaque "Bonjour"<br>2 c'est qu'il faut un délai de 2 secondes entre chaque "Bonjour"<br>3 c'est qu'il faut un délai de 3 secondes entre chaque "Bonjour"<br>Une autre valeur, dites poliment à l'utilisateur qu'il n'a pas compris.<br>Gérez ce cas en utilisant les fonctions définies avant. | Difficulté : 3 |

### 5.5 Ping-Pong en utilisant une fonction et des variables globales

| | |
|---|---|
| L'idée est que l'équipe 1 va jouer contre l'équipe 2 au tennis de table.<br>L'utilisateur doit voir apparaître :<br>Si l'équipe 1 marque tapez 1, si l'équipe 2 marque tapez 2<br>Au fur et à mesure de la partie le score (X-Y) doit se mettre à jour (X représente les points de l'équipe 1 et Y les points de l'équipe 2) et ce tant que la partie n'est pas terminée.<br>Si l'utilisateur n'a pas enfoncé un choix correct (1 ou 2), il faut lui demander de recommencer son choix.<br>Les points s'incrémentent chaque fois d'une unité jusqu'à ce que la partie soit terminée.<br>La partie se termine dés qu'une équipe atteint 11 points avec au moins 2 points d'écarts sur l'autre équipe (dans certains cas il faudra plus de 11 points pour avoir 2 points d'écarts).<br>Une fois la partie terminée, vous devez afficher sur écran le nom du vainqueur.<br>Attention, l'exercice ici consiste également à mettre en œuvre une fonction, cette fonction est nommée « affiche_score() » qui ne reçoit pas de paramètre mais qui pourra afficher le score parce que les variables liées au score sont des variables globales.<br>Essayer de voir les avantages et inconvénients de cette méthode. | Difficulté : 4 |

### 5.6 Ping-Pong et 1, 2 ou 3 sets gagnants utilisant une fonction pour le score

| | |
|---|---|
| Le principe est le même que l'exercice précédent mais l'utilisateur doit commencer par préciser le nombre de sets gagnants souhaités (1, 2 ou 3). Si l'utilisateur n'a pas enfoncé un choix correct (1, 2 ou 3), il faut lui demander de recommencer son choix.<br>Une fois le nombre de sets gagnants introduits, il faudra mettre à jour le score actuel en précisant le nombre de sets gagnés pour chaque équipe et le score du set en cours.<br>A la fin de chaque set il faudra indiquer le vainqueur du set et mettre à jour les sets gagnés de chaque équipe.<br>Une fois qu'une équipe a atteint le nombre de sets à gagner, il faudra indiquer l'équipe qui a gagné le match.<br>Attention, tout comme l'exercice précédent, l'exercice ici consiste à mettre en œuvre une fonction, cette fonction est nommée « affiche_score() » qui ne reçoit pas de paramètre mais qui pourra afficher le score parce que les variables liées au score sont des variables globales. Le score est maintenant le score du match en cours, à savoir, les sets gagnés pour chaque équipe et les points du set en cours.<br>Essayer de voir les avantages et inconvénients de cette méthode. | Difficulté : 5 |

## 6 Fonction avec un paramètre, sans valeur de retour

Les notions associées sont : - Les notions vue de 1.) → 5.)
- Les fonctions avec un paramètre

### 6.1 Affiche le carré d'un nombre

| | |
|---|---|
| Définissez une fonction (choisissez un nom évocateur) qui reçoit un nombre comme paramètre et qui affiche un message informant de la valeur du carré de ce nombre.<br>Au sein du programme principal qui sera exécuté, faites un test de fonctionnement de la fonction en affichant le carré d'une constante suivi de l'affichage d'un nombre introduit par l'utilisateur. | Difficulté : 2 |

### 6.2 Affiche la parité d'un nombre

| | |
|---|---|
| Définissez une fonction (choisissez un nom évocateur) qui reçoit un nombre comme paramètre et qui affiche un message informant de la parité de ce nombre.<br>Au sein du programme principal qui sera exécuté, faites un test de fonctionnement de la fonction en affichant la parité d'une constante paire (au choix), puis la parité d'une constante impaire (au choix) et enfin la parité d'un nombre introduit par l'utilisateur. | Difficulté : 3 |

### 6.3 Affiche un nombre et son opposé

| | |
|---|---|
| Définissez une fonction (choisissez un nom évocateur) qui reçoit un nombre comme paramètre et qui affiche un message reprenant la valeur de ce nombre suivi de la valeur de son opposé.<br>Au sein du programme principal qui sera exécuté, faites un test de fonctionnement de la fonction en demandant à l'utilisateur d'introduire un nombre puis en faisant appel à votre fonction pour afficher la valeur de ce nombre et de son opposé. | Difficulté : 2 |

### 6.4 Affiche un nombre et son inverse

| | |
|---|---|
| Définissez une fonction (choisissez un nom évocateur) qui reçoit un nombre comme paramètre et qui affiche un message reprenant la valeur de ce nombre suivi de la valeur de son inverse (l'inverse d'un nombre est 1 divisé par ce nombre).<br>Au sein du programme principal qui sera exécuté, faites un test de fonctionnement de la fonction en demandant à l'utilisateur d'introduire un nombre puis en faisant appel à votre fonction pour afficher la valeur de ce nombre et de son inverse.<br>Attention l'opérateur « / » sur entier donne le quotient et non le résultat exact de la division. D'ailleurs la valeur « 1 » est considérée comme entier mais « 1.0 » comme réel. | Difficulté : 3 |

### 6.5 Affiche n fois bonjour où n est le paramètre

| | |
|---|---|
| Définissez une fonction (choisissez un nom évocateur) qui reçoit un nombre comme paramètre et qui affiche « Bonjour » autant de fois que la valeur du paramètre reçu.<br>Au sein du programme principal qui sera exécuté, faites un test de fonctionnement de la fonction en demandant à l'utilisateur d'introduire un nombre puis en faisant appel à votre fonction pour afficher « Bonjour » autant de fois que le nombre introduit. | Difficulté : 3 |

### 6.6 Trois fonctions bonjour avec paramètre

| | |
|---|---|
| Définissez trois fonctions proches mais pourtant différentes. Les trois fonctions doivent afficher « Bonjour » autant de fois de suite que le paramètre introduit mais l'intervalle de temps entre chaque affichage n'est pas identique. Les trois fonctions sont donc :<br>bonjour_norm(…) → Affichage des « Bonjour » à vitesse normale (délais de 1s.)<br>bonjour_lent(…) → Affichage des « Bonjour » à vitesse lente (délais de 2s.)<br>bonjour_tres_lent(…) → Affichage des « Bonjour » à vitesse très lente (délais de 3s.)<br>Les « … » des fonctions montrent l'utilisation d'un paramètre (le nombre de répétions).<br>Ensuite l'utilisateur doit voir apparaître sur écran :<br>Veuillez entrer la vitesse : normal (tapez 1) – lent (tapez 2) – très lent (tapez 3)<br>Le choix de l'utilisateur doit bien sûr être mémorisé, s'il s'agit de :<br>1 c'est qu'il faut un délai de 1 seconde entre chaque "Bonjour"<br>2 c'est qu'il faut un délai de 2 secondes entre chaque "Bonjour"<br>3 c'est qu'il faut un délai de 3 secondes entre chaque "Bonjour"<br>Une autre valeur, dites poliment à l'utilisateur qu'il n'a pas compris.<br>Gérez ce cas en utilisant les fonctions définies avant. | Difficulté : 4 |

### 6.7 Fonction décompte avec paramètre

| | |
|---|---|
| Définissez une fonction (choisissez un nom évocateur) qui provoque un décompte (décrémentation à chaque seconde) qui commence à partir du nombre passé en paramètre.<br>Testez ensuite votre programme en commençant par demander à l'utilisateur d'introduire un nombre et en démarrant le décompte à partir de ce nombre. | Difficulté : 3 |

### 6.8 Affiche la table du paramètre

| | |
|---|---|
| Définissez une fonction dont le rôle est d'afficher la table (jusqu'à 10) du nombre passé en paramètre. Testez ensuite votre programme en demandant à l'utilisateur d'introduire un nombre et en faisant appel à votre fonction avec ce nombre comme paramètre. | Difficulté : 3 |

### 6.9 Affiche la table de 3 jusqu'au paramètre

| | |
|---|---|
| Définissez une fonction dont le rôle est d'afficher la table de 3 mais plus forcément jusqu'à 10, le nombre de facteurs peut être différent. La fonction doit alors avoir un paramètre qui est le nombre de facteurs qu'il faudra afficher pour la table de 3.<br>Testez ensuite votre programme en demandant à l'utilisateur d'introduire un nombre (le nombre de facteurs) et en faisant appel à votre fonction avec ce nombre comme paramètre. | Difficulté : 3 |

### 6.10 Décompte sous forme « mm :ss » suivant le nombre de secondes reçues

| | |
|---|---|
| L'idée est d'avoir une fonction qui est prête à accepter un nombre supérieur à 60 comme paramètre et de réaliser un décompte indiquant les minutes et les secondes.<br>Le décompte devra être mis sous la forme « mm :ss » où « mm » représente les deux chiffres possibles des minutes en cours et « ss » les deux chiffres possibles des secondes en cours.<br>Testez ensuite votre programme en demandant à l'utilisateur d'introduire un nombre (le nombre de secondes) et en faisant appel à votre fonction pour tester le décompte.<br>Remarque : Un test peut s'avérer être très long, il alors est intéressant, juste pour le test, de par exemple modifier le délai de 1seconde en un délai de 0,1 seconde.<br>On écrira alors time.sleep(0.1) pour gagner du temps de test (on remet « 1 » une fois testé). | Difficulté : 4 |

### 6.11 Décomposition d'un nombre en produit de nombres premiers

| | |
|---|---|
| Définissez et utilisez une fonction qui affiche la décomposition d'un nombre en un produit de nombres premiers. On suppose que le nombre à décomposer est compris entre 1 et 30.<br>La fonction reçoit le nombre à décomposer comme paramètre.<br>Demandez à l'utilisateur d'introduire un nombre compris entre 1 et 30.<br>Si le nombre introduit n'est pas un entier entre 1 et 30, dites poliment à l'utilisateur qu'il n'a pas compris et demandez à nouveau d'introduire le nombre (et ce tant que nécessaire).<br>Une fois qu'un nombre valide a été introduit, tentez de décomposez ce nombre en produit de nombres premiers sachant que les nombres premiers compris entre 1 et 30 sont :<br>2, 3, 5, 7, 11, 13, 17, 19, 23, 29. Essayez de résoudre cet exercice avec les outils actuels. | Difficulté : 5 |

## 7 Fonction avec plusieurs paramètres

Les notions associées sont : - Les notions vue de 1.) → 6.)
- Les fonctions avec plusieurs paramètres

### 7.1 Affiche le minimum de deux paramètres

| | |
|---|---|
| La fonction doit recevoir deux paramètres et afficher le plus petit des deux paramètres.<br>Définissez la fonction et utilisez la au sein du programme en demandant à l'utilisateur d'introduire les deux nombres dont on voudrait déterminer le minimum. | Difficulté : 3 |

### 7.2 Affiche les trois paramètres par ordre croissant

| | |
|---|---|
| La fonction doit recevoir trois paramètres et afficher ces paramètres par ordre croissant.<br>Définissez la fonction et utilisez la au sein du programme en demandant à l'utilisateur d'introduire les trois nombres qu'on voudrait afficher par ordre croissant. | Difficulté : 3 |

### 7.3 Affiche des bonjour mais deux paramètres

| | |
|---|---|
| L'idée est de créer une fonction (choisissez un nom évocateur) qui a deux paramètres :<br>Le 1er indique le nombre de répétitions de « bonjour »<br>Le 2ème indique le délai en secondes entre les différentes répétitions.<br>Définissez cette fonction et mettez la en œuvre pour que l'utilisateur puisse choisir le nombre de répétitions ainsi que le délai souhaité entre les répétitions.<br>Testez ensuite des possibilités comme (3,4) et (10,1) comme combinaisons de paramètres. | Difficulté : 3 |

### 7.4 Affiche la table en utilisant deux paramètres

| | |
|---|---|
| L'idée est de créer une fonction qui, en quelque sorte, généralise les exercices 6.8 et 6.9.<br>La fonction contient deux paramètres :<br>Le 1er indique le nombre dont on veut afficher la table<br>Le 2ème indique le nombre de facteurs souhaités (jusqu'où on affiche la table)<br>Définissez cette fonction et mettez la en œuvre pour que l'utilisateur puisse choisir quel nombre utiliser pour la table ainsi que le nombre de facteurs souhaités.<br>Testez ensuite des possibilités comme (3,10), (3,20), (3,8) et (8,3) comme combinaisons de paramètres. | Difficulté : 3 |

### 7.5 Ping-Pong en utilisant une fonction sans manipuler de variable globale

| | |
|---|---|
| L'idée est que l'équipe 1 va jouer contre l'équipe 2 au tennis de table.<br>L'utilisateur doit voir apparaître :<br>Si l'équipe 1 marque tapez 1, si l'équipe 2 marque tapez 2<br>Au fur et à mesure de la partie le score (X-Y) doit se mettre à jour (X représente les points de l'équipe 1 et Y les points de l'équipe 2) et ce tant que la partie n'est pas terminée.<br>Si l'utilisateur n'a pas enfoncé un choix correct (1 ou 2), il faut lui demander de recommencer son choix.<br>Les points s'incrémentent chaque fois d'une unité jusqu'à ce que la partie soit terminée.<br>La partie se termine dés qu'une équipe atteint 11 points avec au moins 2 points d'écarts sur l'autre équipe (dans certains cas il faudra plus de 11 points pour avoir 2 points d'écarts).<br>Une fois la partie terminée, vous devez afficher sur écran le nom du vainqueur.<br>Attention, l'exercice ici consiste également à mettre en œuvre une fonction qui reçoit comme paramètres les points des équipes 1 et 2 et qui affiche sur écran le score en cours.<br>Essayer de voir les avantages et inconvénients de cette méthode. | Difficulté : 4 |

### 7.6 Ping-Pong et 1, 2 ou 3 sets gagnants utilisant une fonction pour le score

| | |
|---|---|
| Le principe est le même que l'exercice précédent mais l'utilisateur doit commencer par préciser le nombre de sets gagnants souhaités (1, 2 ou 3). Si l'utilisateur n'a pas enfoncé un choix correct (1, 2 ou 3), il faut lui demander de recommencer son choix.<br>Une fois le nombre de sets gagnants introduits, il faudra mettre à jour le score actuel en précisant le nombre de sets gagnés pour chaque équipe et le score du set en cours.<br>A la fin de chaque set il faudra indiquer le vainqueur du set et mettre à jour les sets gagnés de chaque équipe.<br>Une fois qu'une équipe a atteint le nombre de sets à gagner, il faudra indiquer l'équipe qui a gagné le match.<br>Attention, tout comme l'exercice précédent, il faut utiliser une fonction qui reçoit suffisamment de paramètres pour afficher tous les détails du match en cours. | Difficulté : 5 |

### 7.7 Affiche le max des trois paramètres

| | |
|---|---|
| Définissez une fonction qui affiche le maximum parmi trois nombres passés comme paramètre. Imaginez ensuite, au sein de votre programme principal, une demande d'introduction de ces trois nombres auprès de l'utilisateur afin de pouvoir tester le bon fonctionnement de votre fonction. | Difficulté : 3 |

### 7.8 Affiche le décompte recevant les minutes et les secondes

| | |
|---|---|
| Définissez une fonction qui a deux paramètres :<br>Le 1er indique les minutes<br>Le 2ème indique les secondes<br>Et qui gère un compteur sous la forme « mm :ss »,<br>« mm » représente les deux chiffres possibles des minutes en cours<br>« ss » les deux chiffres possibles des secondes en cours.<br>Votre programme principal doit alors demander à l'utilisateur d'introduire les minutes et les secondes pour que puisse démarrer le compteur après un appel de votre fonction.<br>Remarque : Un test peut s'avérer être très long, il alors est intéressant, juste pour le test, de par exemple modifier le délai de 1seconde en un délai de 0,1 seconde.<br>On écrira alors time.sleep(0.1) pour gagner du temps de test (on remet « 1 » une fois testé). | Difficulté : 4 |

### 7.9 Affiche la moyenne de quatre paramètres

| | |
|---|---|
| Définissez une fonction qui affiche la moyenne des quatre nombres passés en paramètre. Imaginez ensuite, au sein de votre programme principal, une demande d'introduction de ces nombres auprès de l'utilisateur afin de pouvoir tester le bon fonctionnement de votre fonction.<br>Attention, le résultat de « a/b » risque de ne pas tomber juste si « a » est un entier.<br>Par contre le fait de multiplier un entier par un réel permettra d'interpréter le produit comme un réel. | Difficulté : 3 |

### 7.10 Affiche un produit sans l'opérateur « * »

| | |
|---|---|
| Définissez une fonction qui affiche le produit de deux nombres entiers passés en paramètres. Attention, vous n'avez pas le droit d'utiliser l'opérateur « * ».<br>Rem : Une multiplication peut être exprimée sous la forme d'une succession d'additions. | Difficulté : 3 |

### 7.11 Affiche une puissance sans l'opérateur « ** »

| | |
|---|---|
| Définissez une fonction qui affiche une puissance créée à partir des deux nombres entiers passés en paramètres. Le premier nombre est la base et le deuxième est l'exposant.<br>Attention, vous n'avez pas le droit d'utiliser l'opérateur « ** ».<br>Rem : Une puissance peut être exprimée sous la forme d'une succession de multiplications. | Difficulté : 3 |

## 8 Fonction avec paramètre(s) et une valeur de retour

Les notions associées sont : - Les notions vue de 1.) → 7.)
- Les fonctions avec une valeur de retour

### 8.1 Renvoyer le minimum des deux paramètres

| | |
|---|---|
| La fonction doit recevoir deux paramètres et renvoyer le plus petit des deux paramètres. Définissez la fonction et utilisez la au sein du programme en demandant à l'utilisateur d'introduire les deux nombres dont on voudrait déterminer le minimum. | Difficulté : 3 |

### 8.2 Renvoyer le maximum des trois paramètres

| | |
|---|---|
| La fonction doit recevoir trois paramètres et renvoyer le plus grand des trois paramètres.<br>Définissez la fonction et utilisez la au sein du programme en demandant à l'utilisateur d'introduire les trois nombres dont on voudrait déterminer le maximum. | Difficulté : 3 |

### 8.3 Renvoyer la moyenne des quatre paramètres

| | |
|---|---|
| Définissez et mettez en œuvre une fonction qui renvoie la moyenne des quatre nombres passés en paramètre.<br>Attention avec l'opérateur « / », un résultat tel que « a/b » risque de ne pas tomber juste si « a » est un entier.<br>Par contre multiplier un entier par un réel fera interpréter le produit comme un réel. | Difficulté : 3 |

### 8.4 Renvoyer le produit sans l'opérateur « * »

| | |
|---|---|
| Définissez une fonction qui renvoie le produit de deux nombres entiers passés en paramètres.<br>Attention, vous n'avez pas le droit d'utiliser l'opérateur « * ».<br>Rem : Une multiplication peut être exprimée sous la forme d'une succession d'additions. | Difficulté : 3 |

### 8.5 Renvoyer une puissance sans l'opérateur « ** »

| | |
|---|---|
| Définissez une fonction qui renvoie une puissance créée à partir des deux nombres entiers passés en paramètres. Le premier nombre est la base et le deuxième est l'exposant.<br>Attention, vous n'avez pas le droit d'utiliser l'opérateur « ** ».<br>Rem : Une puissance peut être exprimée sous la forme d'une succession de multiplications. | Difficulté : 3 |

### 8.6 Renvoyer un booléen pour la parité

| | |
|---|---|
| Définissez une fonction qui renvoie un booléen pour indiquer si son paramètre est pair ou impair. Le booléen devra renvoyer la valeur « True » si le paramètre est pair et la valeur « False » s'il ne l'est pas.<br>Mettez ensuite en œuvre cette fonction pour afficher un message indiquant à l'utilisateur si le nombre qu'il vient d'introduire est pair ou impair. | Difficulté : 4 |

### 8.7 Renvoyer le résultat d'une opération parmi quatre

| | |
|---|---|
| Définissez une fonction qui a trois paramètres ;<br>Le 1er indique l'opération souhaitée<br>Le 2ème indique le 1er nombre à prendre en compte pour l'opération<br>Le 3ème indique le 2ème nombre à prendre en compte pour l'opération<br>Le 1er paramètre a quatre valeurs différentes possibles :<br>« 1 » pour une addition, on fait le 2ème paramètre + le 3ème paramètre<br>« 2 » pour une soustraction, on fait le 2ème paramètre - le 3ème paramètre<br>« 3 » pour une multiplication, on fait le 2ème paramètre * le 3ème paramètre<br>« 4 » pour une multiplication, on fait le 2ème paramètre ** le 3ème paramètre<br>Tout autre valeur doit engendrer un message d'erreur à l'utilisateur et demander de réintroduire un choix correct.<br>Trouvez un moyen de mettre en œuvre cette fonction de manière à susciter l'intervention de l'utilisateur. | Difficulté : 4 |

### 8.8 Renvoyer le résultat d'une fonction logique donnée

| | |
|---|---|
| Un circuit combinatoire est l'équivalent de l'expression booléenne suivante :<br><br>$S = \overline{a} \cdot b + \overline{b \cdot c} + \overline{a} \cdot \overline{b} \cdot \overline{c}$<br><br>Définissez une fonction qui renvoie S en tenant compte des trois paramètres a, b et c.<br>Remarques :<br>a, b, c et S sont des variables booléennes<br>un trait au dessus correspond à l'opérateur logique « NOT »<br>une multiplication est l'équivalent de l'opérateur logique « and »<br>une addition est l'équivalent de l'opérateur logique « or »<br>Trouvez ensuite une manière conviviale de tester votre fonction. | Difficulté : 3 |

### 8.9 Poules et moutons

| | |
|---|---|
| Un fermier a une drôle de façon de compter ses poules.<br>Le fermier possède des poules et des moutons et pour déterminer le nombre de poules il compte le nombre total de têtes (poule comme mouton) ainsi que le nombre total de pattes (poule comme mouton également).<br>Ensuite il note sur un morceau de papier les deux nombres :<br>Le 1er nombre est le nombre de têtes<br>Le 2ème nombre est le nombre de pattes<br>Il détermine enfin par raisonnement le nombre de poules présentes à partir de son morceau de papier. Cependant sa femme, informaticienne, souhaite définir une fonction qui renvoie le nombre de poules présentes à partir des deux nombres passés en paramètres.<br>Définissez et mettez en œuvre une telle fonction sachant que le 1er paramètre est le nombre de têtes et le 2ème le nombre de pattes. | Difficulté : 3 |

## 9 Fonction avec paramètre et plusieurs valeurs de retour

Les notions associées sont : - Les notions vue de 1.) → 8.)
- Les fonctions avec plusieurs valeurs de retour

Il faut savoir qu'il n'est pas habituel en programmation d'avoir plusieurs valeurs de retour. Le langage python permet cet artifice grâce au principe d'affectation multiple.

### 9.1 Fonction swap

| | |
|---|---|
| Définissez une fonction qui reçoit deux paramètres et qui renvoie ces deux mêmes paramètres mais dans l'ordre inverse. Une affectation multiple combinée à un appel de fonction permet alors de faire ce qu'on appelle un « swap », c'est-à-dire de commuter les valeurs des deux variables. | Difficulté : 3 |

### 9.2 Renvoyer les trois paramètres ordonnés

| | |
|---|---|
| Définissez une fonction qui reçoit trois paramètres et qui renvoie ces trois mêmes paramètres ordonnés du plus petit au plus grand.<br>Trouvez ensuite une manière conviviale de tester votre fonction. | Difficulté : 3 |

### 9.3 Renvoyer les trois paramètres ordonnés

| | |
|---|---|
| Définissez une fonction qui reçoit un entier représentant le nombre total de secondes et qui a deux valeurs de retour :<br>La 1ère représente les minutes<br>La 2ème représente les secondes<br>L'idée est de s'aider de cette fonction pour réaliser un décompte sous la forme « mm : ss ».<br>« mm » représente les deux chiffres possibles des minutes en cours<br>« ss » les deux chiffres possibles des secondes en cours.<br>Trouvez ensuite une manière conviviale de tester votre fonction au sein d'un décompte. | Difficulté : 4 |

### 9.4 Renvoyer le résultat d'une opération parmi quatre avec une variable d'état

| | |
|---|---|
| Définissez une fonction qui a deux valeurs de retour, la 1ère est un booléen indiquant si la fonction s'est déroulée sans erreur et la 2ème est le résultat de l'opération.<br>L'idée du paramètre booléen ('True' s'il n'y a pas eu d'erreurs, 'False' autrement) est de pouvoir gérer, en dehors de la fonction, la demande d'introduction des valeurs tant que celle-ci n'est pas correcte.<br>Sachant également que la fonction a trois paramètres ;<br>Le 1er indique l'opération souhaitée<br>Le 2ème indique le 1er nombre à prendre en compte pour l'opération<br>Le 3ème indique le 2ème nombre à prendre en compte pour l'opération<br>Que le 1er paramètre a quatre valeurs différentes possibles :<br>« 1 » pour une addition, on fait le 2ème paramètre + le 3ème paramètre<br>« 2 » pour une soustraction, on fait le 2ème paramètre - le 3ème paramètre<br>« 3 » pour une multiplication, on fait le 2ème paramètre * le 3ème paramètre<br>« 4 » pour une multiplication, on fait le 2ème paramètre ** le 3ème paramètre<br>Tout autre valeur doit engendrer une valeur de retour 'False' pour le 1er paramètre de retour.<br>Trouvez un moyen de gérer une introduction correcte des données de l'utilisateur en mettant en œuvre cette fonction, affichez ensuite le résultat de l'opération. | Difficulté : 4 |

### 9.5 Poules et moutons avec une variable d'état

| | |
|---|---|
| Le problème est le même que l'exercice 8.9 (poules et moutons) sauf qu'ici on souhaite avoir deux valeurs de retour :<br>La 1ère est un booléen indiquant si le résultat est possible ('True' si c'est possible)<br>La 2ème représente le nombre de poules<br>Dans certains cas, les valeurs des paramètres introduits ne sont pas possibles (par exemple 5 têtes et 2 pattes), il faut alors mettre la 1ère valeur de retour à 'False' pour permettre à l'utilisateur de corriger les données.<br>Trouvez une manière conviviale de tester votre fonction en assurant une introduction de données correctes par l'utilisateur. | Difficulté : 4 |