Chapitre 1 : Environnement de développement, présentation de Spyder et de Python.

I) Introduction

1) L'algorithmique

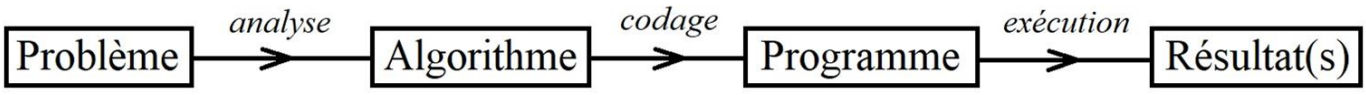
Def : Un algorithme est une suite d'actions qu'il faut appliquer à des données primaires (souvent entrées par l'utilisateur) pour résoudre un problème et aboutir au résultat.

Toute question que nous résoudrons à l'aide d'un algorithme, doit commencer par une étude de la problématique, et se poursuivre par la création d'un algorithme en langage « naturel » avant d'être programmée dans le langage de votre choix (Pascal, C, Java, Python …).

En effet, un algorithme est indépendant du langage de programmation à utiliser. Vu la vitesse à laquelle l'informatique se développe, il ne faut pas s'enfermer dans un seul langage de programmation (qui sera peut-être obsolète dans 10 ans). Il sera nécessaire de savoir bâtir un algorithme efficace en langage naturel pour l'écrire ensuite en n'importe quel langage de programmation.

Cette année nous utiliserons exclusivement le **langage Python**, qui est en libre accès, performant, et employé dans bon nombre d'écoles d'ingénieurs et d'entreprises.

Def : Un programme est un enchaînement d'instructions, écrit dans un langage de programmation, exécuté par un ordinateur via un environnement de développement (nous y reviendrons), permettant de traiter un problème et de renvoyer des résultats. Il représente la traduction d'un algorithme à l'aide d'un langage de programmation.



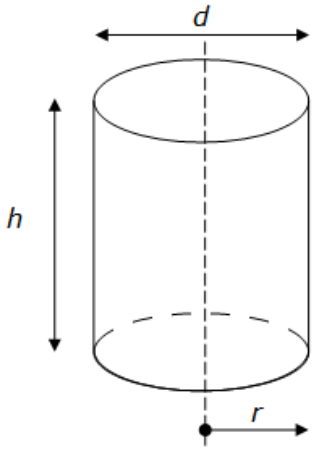
Pour résoudre un problème, l'utilisateur doit donc apprendre à communiquer avec la machine (*langage de programmation et codage*) mais surtout à établir des méthodes de résolution de problèmes (*analyse et algorithmique*). Ce sont deux objectifs du cours d'informatique.

2) Structure d'un algorithme

L'analyse d'un problème sert à extraire 3 composantes :

- Les données d'entrée : données indispensables pour résoudre le problème.
- Les données de sortie : résultat(s) recherché(s).
- Le traitement : L'ensemble des opérations à appliquer sur les données d'entré pour aboutir aux résultats.

Exemple : Problème à analyser : calculer la surface totale d'un cylindre :



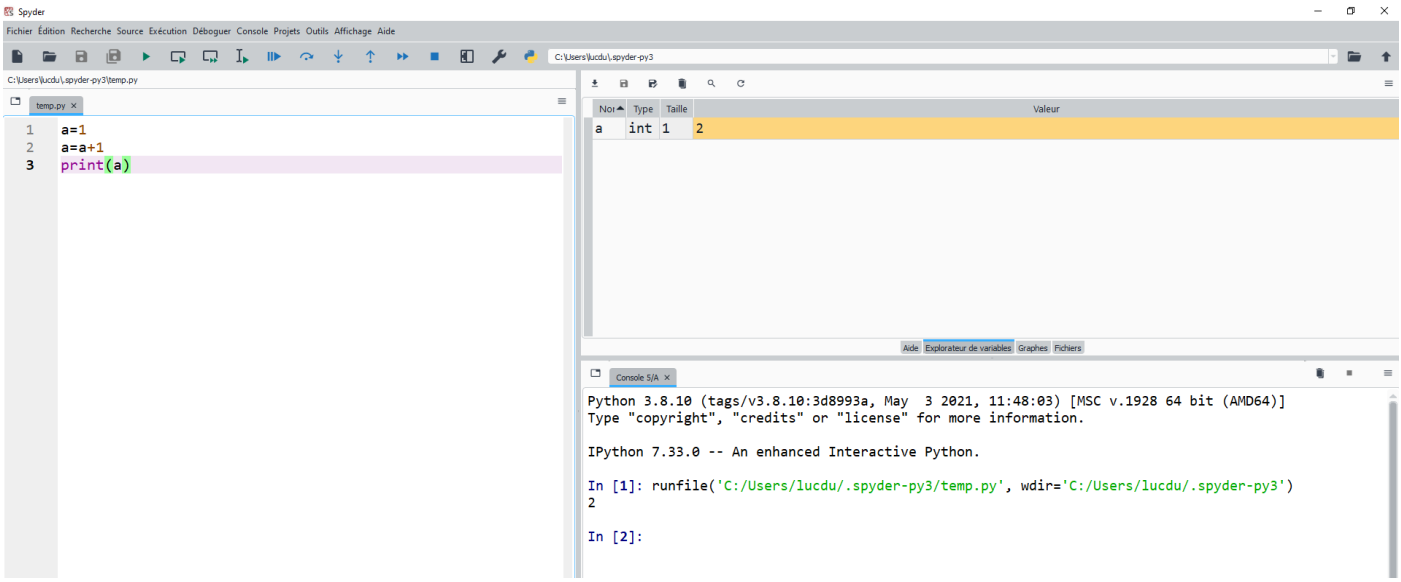
- ➢ Données d'entrée :
- ➢ Données de sortie :
- ➢ Traitement :

3) Environnement de développement interactif (IDE)

On appelle *environnement de développement interactif* un logiciel permettant :

- d'écrire des programmes dans un éditeur adapté au langage,
- d'exécuter les programmes que l'on a écrits,
- de corriger des erreurs (ou déboguer) dans ces programmes,
- de consulter de la documentation (aide).

Pour Python, le logiciel **Spyder** est un logiciel idéal pour les débutants comme pour les utilisateurs confirmés. Il est libre d'accès et téléchargeable facilement sur internet.



La fenêtre de Spyder est divisée en trois parties :

- l'*éditeur* (à gauche),
- l'*explorateur* en haut à droite,
- la *console* interactive en bas à droite.

a) Console

Au démarrage de Spyder, la console interactive affiche la version installée et quelques fonctions d'aide :

> Python 3.8.10 (tags/v3.8.10:3d8993a, May 3 2021, 11:48:03) [MSC v.1928 64 bit (AMD64)]
> Type "copyright", "credits" or "license" for more information.

La dernière ligne, qui commence par un **In** suivi d'un nombre entre crochets, attend que l'on tape une commande : c'est le mode interactif de Python.

In [1] : 5/2

Out [1] : 2.5

L'ensemble des instructions saisies dans cette fenêtre interactive s'appelle une *session de travail*.

Il est possible de créer des variables pour stocker des valeurs :

In [2] : a=2

In [3] : print(a)

2

In [4] : a+a

Out [4] : 4

A chaque nouvelle session, les valeurs des variables sont perdues : on ne peut pas écrire un programme complet de la sorte, mais on peut y tester rapidement une expression spécifique.

b) Éditeur

Pour garder la trace d'une suite d'instructions ou d'un programme complet, on utilise l'éditeur.

Par rapport à la console, l'éditeur met en forme automatiquement les lignes de code :

- les mots-clés du langage, les nombres ou les chaînes de caractère se colorent,

- une parenthèse ouvrante génère une parenthèse fermante.

```
1 print("hello, world")
2 année = 2017
3 print(année)
```

La commande d'exécution ( ou F5) permet de lire et d'effectuer les instructions. Le résultat, s'il y en a un, apparaît dans la fenêtre de console interactive.

c) Explorateur

L'explorateur possède plusieurs onglets :

- l'inspecteur d'objets fournit de l'aide sur un type ou sur une fonction,

- l'explorateur de variables indique la valeur de toutes les variables à tout moment. La fonction déboguer ( ou Ctrl+F5) et l'exécution pas à pas ( ou Ctrl+F10) d'un programme permet d'y suivre les affectations successives des différentes variables (données d'entrée ou de sortie d'un algorithme, ou données intermédiaires du programme) et d'ainsi cibler et corriger d'éventuelles erreurs de code.

- l'explorateur de fichiers permet de parcourir les fichiers Python présents dans le système de fichiers,

- les graphes (depuis la version 4 de Spyder).

II) Variables

L'essentiel du travail effectué par un programme informatique consiste à manipuler des données. Ces données peuvent être très diverses, mais dans la mémoire de l'ordinateur elles se ramènent toujours en définitive à une suite finie de nombres binaires. Ces données sont nommées *variables*.

1) Généralités

Def: Une variable est caractérisée par :

- un *identificateur* ou *nom*, composé de lettres ou de chiffres (ne peut pas commencer par un chiffre)
- un *type*, qui est une information pour sa manipulation par Python et son stockage,
- une *valeur* : une fois créée, une valeur est assignée à la variable, modifiée ou non par la suite,
- une *adresse* ; emplacement dans la mémoire de l'ordinateur.

Rq : En langage naturel, l'affectation d'une valeur à une variable s'écrit par le symbole « ← ».

Exemple : Soit l'algorithme suivant, écrit en langage naturel :

```
0   Algorithme Calcul
1   Variables A, B, C, D : entier
2   Debut
3       A ← 10
4       B ← 30
5       C ← A+B
6       D ← C*A
7       C ← A+C
8       D ← C*A
9   Fin
```

Remplir le tableau suivant des valeurs des différentes variables. On notera « X » si la variable n'est pas encore définie.

n° de ligne	A	B	C	D
3				
4				
5				
6				
7				
8				

Def : - L'*affectation* d'une valeur à une variable (ou *déclaration*) se fait par le simple symbole « = » :

```
1   a=10
2   a=a+1
3   b=2*a
4   print(a,b) #a->11 et b->22
```

- Le symbole « **#** » marque le début d'un commentaire, qui n'est pas exécuté par Python.

(ici ont été données les réponses affichées dans la console)

Rq : - On s'efforcera d'utiliser des identificateurs de variables compréhensibles (autre chose que **a**, **b** ou **x**, mais plutôt par exemple **Surface_cylindre**) ou de donner des détails sur la variable en commentaire.

- Il est très important de se souvenir que le code peut s'étoffer, être lu, utilisé, corrigé par quelqu'un d'autre : utiliser des commentaires pour aider le lecteur et structurer son code est un bon réflexe !

2) Types

Def : La fonction **type(***nom_variable***)** permet d'obtenir le type de variable si nécessaire.

Prop : Dès la déclaration d'une variable, Python lui affecte automatiquement un type. Python manipule chaque type différemment et leur affecte également une taille de mémoire différente.

Les principaux types en Python sont :

- les booléens (*bool*) : **True** et **False**
- les entiers (*int* pour *integer*),
- les réels (*float*),
- les caractères et chaînes de caractères (*str* pour *string*),
- les listes (*list*) et tableaux (*array*),

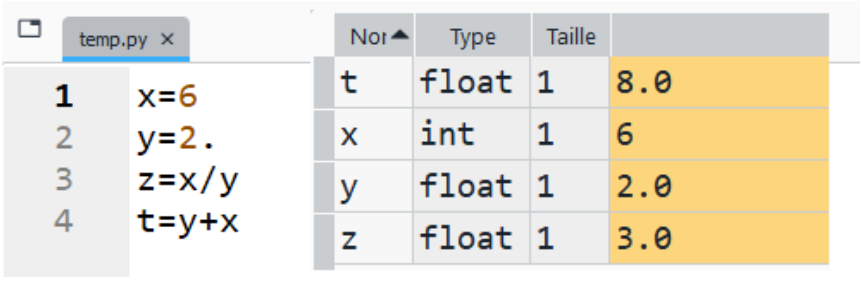
Rq : - Le type de variable est implicite et dynamique en Python (contrairement à de nombreux autres langages).

- Pour des grandeurs physiques, on peut mettre l'unité utilisée en commentaire :

```
D = 1 #m
t = 2 #s
v = D/t #m/s
```

- De nombreuses erreurs de code proviennent d'opérations hasardeuses ou interdites sur le type d'une variable...
- En général, Python donne la priorité au type *float* (plus « flexible ») par rapport au type *int* dans les calculs.

Def : En affectant à une variable une valeur numérique, Python différentie un entier (type int) d'un entier suivi d'un « . » (type float). Lors des calculs, les types peuvent également être modifiés.

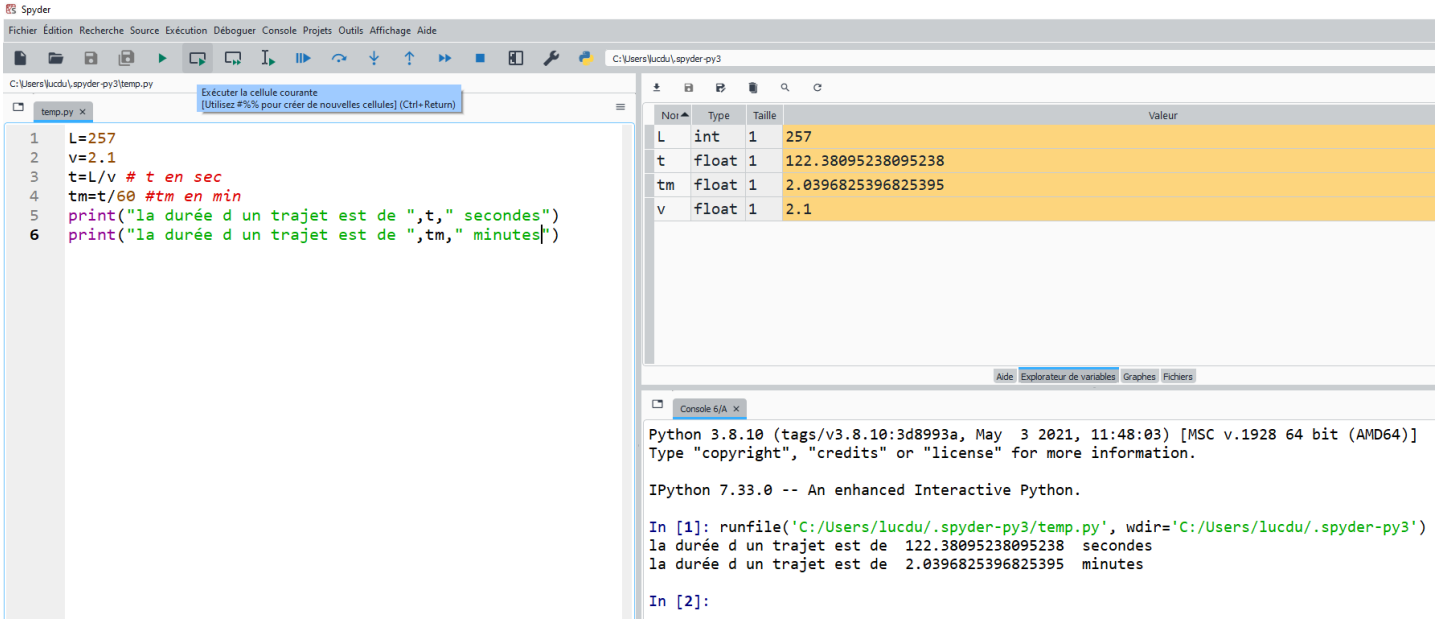


3) Affichage d'une variable en console

Def : La fonction **print()** permet d'évaluer une expression et d'en afficher le résultat, elle permet d'afficher des phrases, des valeurs numériques ou des composés …

```
print("bonjour") # affichage : bonjour
x = 5
a = 'x est égal à'
print(a,x) # affichage : x est égal à 5
```

Exemple : L'ascenseur de la Tour Eiffel effectue un aller-retour de 257 m avec une vitesse moyenne de 2,1 m/s. Affecter les valeurs indiquées aux variables L et v et afficher la durée d du trajet en secondes puis en minutes.



4) Déclaration d'une variable par la console

Def : Il est possible de demander à l'utilisateur d'affecter lui-même une variable dans la console.

Pour cela, on utilise la fonction **input()** :

```
age=input("Quel est votre âge ?")
```

Rq: une variable déclarée par ce moyen est de type **str**.

On peut cependant contraindre Python à en changer le type pour pouvoir manipuler comme on le souhaite cette variable dans la suite du programme.

Il existe pour cela des fonctions qui modifient le type et même la forme d'une variable (quand c'est possible).

- fonction **int()** : transforme en entier une chaîne de caractères composée de chiffres, ou arrondit un réel en entier :

```
age2=int(input("Quel est votre âge ?"))
```

- fonction **float()** : transforme en réel une chaîne de caractères composée de chiffres, ou convertit un entier en réel,

- fonction **str()** : convertit en chaînes de caractères un entier, un réel, une liste,

- fonction **list()** : convertit en liste d'éléments distincts une chaînes de caractères.

5) Opérations

Def : étant donnés deux nombres réels a et b et un entier $n \geq 0$. On réalise les calculs suivants en python à l'aide des instructions indiquées :

$a + b$	$a + b$
$a - b$	$a - b$
$a \times b$	$a * b$
a^n	$a ** n$
$\frac{a}{b}$	a/b
$a \ mod \ b$ (reste dans la division euclidienne de a par b lorsque a et b sont entiers)	$a\%b$
Le quotient de la division euclidienne de a par b (lorsque a et b sont entiers)	$a \ // \ b$

Exemple :

```
1  a=30
2  b=4
3  print("reste ",a%b,"quotient ", a//b)
```

```
In [1]: runfile('C:/Users/lucdu/.spyder-py3/temp.py', wdir='C:/Users/lucdu/.spyder-py3')
reste  2 quotient  7
```

III) Chaînes de caractères et listes

Les chaînes de caractères et les listes ont des manipulations voisines, malgré quelques différences.

1) Chaîne de caractères

Déf : Caractère :

Un caractère est un symbole graphique ; élément de base d'un langage écrit.

Par exemple :

- les lettres de l'alphabet latin (majuscules ou minuscules) : A, a, B, b, etc.
- les lettres accentuées : é, è, à, ï, (souvent propres à une langue, d'où certains problèmes de compatibilité au niveau international),
- les chiffres : 0, 1, 2, etc.
- les lettres de l'alphabet grec : α, β, etc.
- des caractères mathématiques : +, -, >, etc.
- des caractères de ponctuation : !, ?, etc.
- des caractères non imprimables : saut à la ligne, espace, tabulation, etc.

Rq : Ces caractères sont stockés en mémoire selon des standards internationaux : ASCII, Unicode, Utf-8, etc. A chaque caractère est associé un nombre binaire unique.

Déf : chaîne de caractères

Une chaîne de caractères (ou *string*) est une suite de caractères, stockée dans une seule variable.

- Pour affecter une chaîne de caractères à une variable, on utilise le symbole ' … ' ou " … ".

```
chaine1 ='bonjour tout le monde'
chaine2 = '256.89'
```

- Un nombre déclaré comme une chaîne de caractères ne sera pas traité comme un nombre par Python.
- Pour les caractères usuels, la taille d'une chaîne de n caractères en mémoire est de n octets.
- La fonction **len()** permet de connaître le nombre de caractères d'une chaîne de caractères.
- La numérotation (ou **indexation**) des caractères d'une chaîne se fait dans l'ordre de lecture, en commençant à 0. On écrit **nom_chaine[index]** pour appeler un caractère dans une chaîne.

Exemple :

```
a="bonjour"
print(a[0])
print(a[1])
print(a[6])
print(a[7])
```

```
In [3]: runfile('C:/Users/lucdu/.spyder-py3/temp.py', wdir='C:/Users/lucdu/.spyder-py3')
b
o
r
Traceback (most recent call last):

  File "C:\Users\lucdu\AppData\Local\Programs\Spyder\pkgs\spyder_kernels\py3compat.py", line 356,
in compat_exec
    exec(code, globals, locals)

  File "c:\users\lucdu\.spyder-py3\temp.py", line 5, in <module>
    print(a[7])

IndexError: string index out of range
```

Rq : - On retrouve bien l'affichage des lettres situées au rang 0 ;1 et 6, à savoir « b » , « o » et « r » mais il n'y a aucune lettre située au rang 7. Le message d'erreur ci-dessus nous informe qu'il s'agit d'un problème d'indexation, et que l'indice 7 n'est pas une valeur possible.

- Il s'agit d'un principe courant en Python : le premier terme est toujours indexé par 0, le dernier terme d'une chaîne de longueur n est donc indexé par $n-1$.

2) Liste

Def : On appelle liste une suite ordonnée valeurs (pas forcément numériques, on peut faire des listes de lettres par exemple). L'indice de la première valeur est 0.

Exemple : $L=[1,3,5,7,9]$ est une liste et $L[0]=1$; $L[1]=3$ etc…

Def : L'instruction len renvoie le nombre d'éléments d'une liste. Une liste vide se note [].

Exemple : Si $L = [1,3,5,7,9]$ alors len$(L) = 5$

Def : l'instruction insert permet d'insérer un terme à un rang donné d'une liste L.

Exemple : Si $L = [1,3,5,7,9]$ alors l'instruction L.insert(2,5) insert l'élément 5 à l'indice 2 de la liste.
L devient alors : $L = [1,3,5,5,7,9]$

Def : pour insérer un élément x à la fin d'une liste L, on peut aussi écrire $L = L + [x]$. On dit que l'on concatène la liste L avec la liste $[x]$.

Exemple : Si $L = [1,3,5,7,9]$ alors l'instruction L=L+[11] insert l'élément 11 à la fin de la liste.
L devient alors : $L = [1,3,5,7,9,11]$

Rq : On peut concaténer deux listes de n'importe quelle taille :

```
1  a=[14,25,87,64]
2  b=["lundi","mercredi","dimanche"]
3  print(a+b)
```

Def : pour supprimer un élément x d'une liste L, on utilise l'instruction « del », ou l'instruction « remove »

Exemple : Si $L = [1,3,5,7,9]$ alors l'instruction del $L[2]$ supprime l'élément d'indice 2 de la liste.
L devient alors : $L = [1,3,7,9\ ]$
L'instruction L.remove(5) enlèvera la première occurrence de « 5 » dans la liste L.

```
1 semaine=['lundi','mardi','mercredi','jeudi','vendredi','samedi','dimanche']
2 x=len(semaine) # x sera égal à 7
3 y=semaine[0] # permet d'accéder au 1er terme de la liste semaine
4 semaine.remove("mardi") #enlève la première occurence de 'mardi' dans la liste
5 semaine = semaine + ['dimanche2'] #concaténation de 2 listes
```

Rq :

- Une liste se déclare entre crochets,
- La numérotation des éléments d'une liste se fait dans l'ordre de lecture, en commençant à 0,
- On peut modifier un élément d'une liste, le supprimer, en ajouter un (ce qui n'est pas le cas avec une chaîne de caractères).

Exemple : Déclarer une liste L composée des éléments 'a', 'b', 'c' et 'd'.

i) Modifier le 'c' en 3.

ii) Ajouter la liste ['e','f'] à la liste L (à droite).

iii) Ajouter '0' à gauche.

```
1  L=['a','b','c','d']
2  L[2]=3
3  L=L+['e','f']
4  L=[0]+L
5  print(L)
```

IV) Fonctions mathématiques et bibliothèques usuelles

Au démarrage, Python ne connaît pas les fonctions usuelles (ln, exp, cos …) ou le nombre π, il faut **importer** la **bibliothèque math** en écrivant simplement : « **from math import *** » ou « **from math import log,exp** » (si on ne souhaite pas importer toute la bibliothèque).

On retiendra les noms des fonctions suivantes :

- racine carrée : **sqrt**
- partie entière : **floor**
- logarithme : **log** pour le logarithme népérien, **log10** pour le logarithme décimal
- fonctions trigonométriques : **cos**, **sin** ou **tan**
- valeur absolue : **abs**
- pi : **pi**

Rq : Pour accéder à l'aide sur une fonction, on peut taper **help(math.log)** ou utiliser directement l'inspecteur d'objet (panneau en haut à droite dans Spyder).

Il existe de nombreuses bibliothèques pré-installées avec Spyder (mais pas importées au démarrage), qui peuvent s'avérer très utiles. Citons quelques exemples :

- **numpy** : pour le calcul numérique et la manipulation de listes et tableaux,
- **pylab** : pour le calcul numérique,
- **matplotlib** et sa sous-bibliothèque **pyplot** : pour la création de graphiques,
- **scipy** : pour le calcul numérique,
- **csv**, **winsound**, **PIL**, etc. : pour la manipulation de fichiers externes,
- **random** : pour les fonctions de tirages aléatoires,
- **time** : pour des informations sur les instants et durées,
- **tkinter** : pour la création d'animations,
- etc.

Il existe également certaines bibliothèques téléchargeables sur des serveurs.

De nombreuses documentations en ligne existent pour aider les utilisateurs, le plus souvent en anglais et illustrées d'exemples : le programmeur ne connait pas toutes les fonctions de toutes les bibliothèques, mais doit savoir trouver et utiliser les informations !

Exercices :

1) Donner les valeurs successives affichées par le programme suivant.

```
1  L=[1,3]
2  L.insert(1,2)
3  print(L)
4  L=L+[1]
5  print(L)
6  L.remove(1)
7  print(L)
8  del(L[0])
9  print(L)
```

2) Pourquoi le code suivant contient une erreur ?

```
1  a=10
2  b=15
3  c=a-d
4  d=2
5  print(a,b,c,d)
```

3) Que permet de faire le programme suivant ?

```
1  mot="bonjour les TSI 1"
2  print(len(mot))
3  print(mot[len(mot)-1])
```

Que se passerait-il sans le « -1 » ?