Chapitre 9

Python

L'objectif de ce chapitre est de consolider des notions de variable, d'instruction conditionnelle et de boucle ainsi que l'utilisation des fonctions. La seule notion nouvelle est celle de liste qui trouve naturellement sa place dans de nombreuses parties du programme et aide à la compréhension de notions mathématiques telles que les suites numériques, les tableaux de valeurs, les séries statistiques ...

1. Algorithme

Un algorithme est une suite d'instructions élémentaires s'appliquant dans un ordre déterminé à des données et fournissant en un nombre fini d'étapes des résultats.
Nous utilisons le langage de programmation Python pour traduire un algorithme en programme exploitable par la calculatrice ou l'ordinateur.

Algorithme

```
Saisir le rang initial
Saisir le terme initial
Saisir le rang du terme désiré
tant que rang < rang désiré faire
    u ← formule récurrence
    Incrémenter rang de 1
fin
Afficher u
```

Programme en Python

```
init=int(input("Rang initial = "))
u=float(input("Terme initial = "))
n=int(input("Rang àcalculer = "))
while init<n:
    u=2*u+2
    init=init+1
print("u",init,"=",u)
```

2. Variables

a) Types de variables

Dans un programme, une variable est repérée par un nom et possède une valeur qui évolue au cours de l'exécution du programme. On peut la schématiser par une boîte (un emplacement de la mémoire d'un ordinateur) qui porte une étiquette et son contenu. On utilise différents types de variables :

- nombre entier (integer → **int**) ;
- nombre flottant (nombre à virgule, floating point number → **float**) ;
- chaîne de caractères (character string → **str**) ;
- liste (collection d'éléments écrits entre [ ] ;
- booléen (boolean → **bool** prend la valeur *True* ou *False*).

b) Opérations sur les variables

Affectation

Lorsque l'on affecte (donne) une valeur, par exemple 37, à une variable par exemple a :

- dans un algorithme, on écrit l'instruction : $a \leftarrow 37$
- dans un programme, on écrit l'instruction : $a = 37$
- on lit : « a reçoit 37 » ou « a prend la valeur 37 » ou encore « la variable a est affectée de la valeur 37 ».

La nouvelle valeur remplace la valeur précédente.

```
a ← 37
```

```
a=37
```

Addition, multiplication, . . ., de variables

Additionner b à a	$a+b$	Soustraire b à a	$a-b$
Multiplier a par b	$a*b$	Diviser a par b	a/b
Calculer le quotient de la division euclidienne de a par b	$a//b$	Calculer le reste de la division euclidienne de a par b	$a\%b$
Élever a à la puissance n	$a**n$		

c) Instructions d'entrées et sorties

Elles permettent de saisir en entrée et d'afficher en sortie les valeurs des variables.

Saisir a
Afficher a

```
a=float(input())
print(a)
```

Remarques :

- ces instructions permettent également d'afficher un message :
  - . n = int(input("Nombre d'essais = ")) ;
  - . print("La surface obtenue est : ",S).
- en Python, l'instruction d'entrée précise le type de la variable : int pour les nombres entiers, float pour les nombres à virgules et si rien n'est précisé la variable sera considérée comme une chaîne de caractères.

Saisir n
Afficher S

```
n=int(input("Nombre d'essais = "))
print("La surface obtenue est : ",S)
```

3. Instructions conditionnelles

Dans un algorithme, on est parfois amené à exécuter une ou plusieurs instructions uniquement si une certaine condition est vérifiée : ce sont des instructions conditionnelles. Si la condition n'est pas vérifiée, on peut exécuter un autre bloc d'instruction ou ne rien faire. Dans tous les cas, ensuite, on exécute la suite de l'algorithme

Opérateurs de tests								
Égalité	Non égalité	$<$	$>$	$\leqslant$	$\geqslant$	ET	OU	NON
==	!=	<	>	<=	>=	and	or	no

Une seule condition **if . . .**

Pour éviter une division par 0

```
if b!=0:
  div=a/b
```

Deux conditions **if . . . else**

Expression en fonction d'intervalles

```
if x<=2:
  y=x**2
else:
  y=2*x+1
```

Trois conditions et plus **if . . . elif . . . else**

Étude de signes

```
if x<0:
  print("négatif")
elif x==0:
  print("nul")
else:
  print("positif")
```

4. Boucles bornées ou non

Une boucle permet de répéter plusieurs fois de suite un bloc d'instructions. Lorsque le nombre n d'itérations est connu à l'avance, on définit une boucle **bornée** appelée également boucle Pour (For en anglais). Notamment elle permet de parcourir tous les éléments d'une liste ou d'une chaîne.
Certaines fois, le nombre n d'itérations n'est pas connu à l'avance, il peut dépendre d'une condition ; on définit alors une boucle **non bornée** appelée également boucle Tant que (While en anglais). Le bloc d'instructions est répété tant que la condition est vraie ; quand celle-ci devient fausse, alors on sort de la boucle et on applique la suite du programme. Notamment elle permet de rechercher le rang des termes d'une suite à partir du quel on atteint un seuil.
Pour obtenir l'affichage du carré des entiers de 0 à 4, on peut programmer les boucles suivantes :

Boucle bornée

```
for i in range(5):
  print(i**2)
```

Boucle non bornée

```
i=0
while i<5:
  print(i**2)
  i=i+1
```

On effectue la somme des n premiers nombres impairs ; le nombre de répétitions est connu donc on utilise une boucle bornée.

Somme des n premiers impairs

```
n=int(input("Nb d'impairs ="))
s=0
for i in range(1,2*n,2):
  s=s+i
print(s)
```

On effectue la somme des nombres impairs tant que le total n'a pas atteint 50 ; le nombre de répétitions est inconnu donc on utilise une boucle non bornée.

Sommes des impairs jusqu'à un total minimal de 50

```
s=0
i=1
while s<50:
  s=s+i
  i=i+2
print(s)
```

5. Fonctions

Une fonction réalise un bloc d'instructions et renvoie un résultat ; elle ne sera exécutée que si elle est appelée dans le programme et ceux-ci éventuellement plusieurs fois. Elle possède généralement des paramètres qui prendront pour valeur les arguments donnés à la fonction.

Exemple : on considère les algorithmes suivants calculant le volume de pyramides à base carrée. Soient c, h et v des variables réelles.

Algorithme sans fonction

$c \leftarrow 2$
$h \leftarrow 5$
Afficher $1/3 \times h \times c^2$
$c \leftarrow .5$
$h \leftarrow 10$
Afficher $1/3 \times h \times c^2$

Algorithme avec fonction

Définir fonction
$volpyr \leftarrow 1/3 \times h \times c^2$

Afficher $volpyr(2, 5)$
Afficher $volpyr(.5, 10)$

Ce qui se traduit par les programmes suivants :

sans fonction

```
c=2
h=5
print(1/3*h*c**2)
c=.5
h=10
print(1/3*h*c**2)
```

avec fonction

```
def volpyr(c,h):
  return 1/3*h*c**2

print(volpyr(2,5))
print(volpyr(.5,10))
```

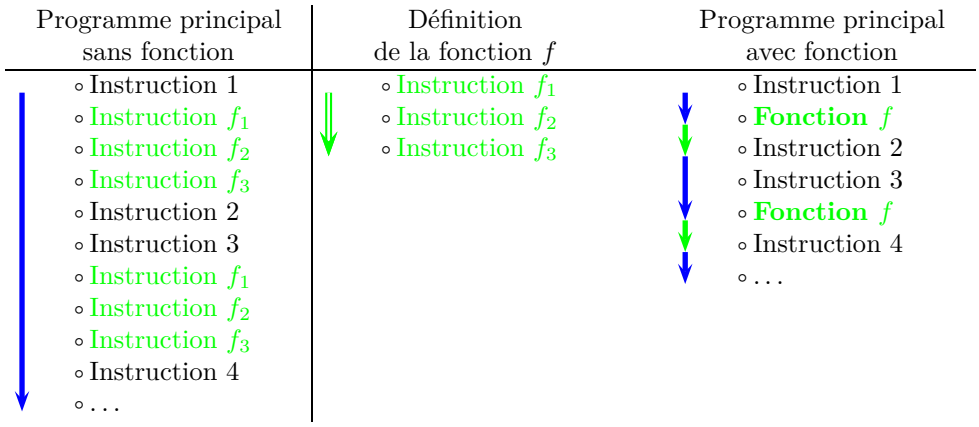
La fonction volpyr prend en paramètres les valeurs du côté de la base et la hauteur de la pyramide et renvoie le volume de la pyramide.

a) Intérêts

— Lisibilité : les fonctions rendent le programme principal plus facile à lire.
— Répétabilité : les fonctions permettent d'éviter la redondance du programme. Il est assez fréquent d'avoir à répéter des blocs d'instructions, on les regroupes alors dans une fonction.
— Flexibilité : il est plus rapide de modifier une fonction que de modifier l'ensemble des parties du programme où elle est utilisée.

b) Structure

Les schémas ci-dessous montrent la différence entre l'exécution d'un programme sans fonction, à gauche, et d'un programme avec fonction, à droite ; où les instructions f_1, f_2, f_3 sont regroupées dans une fonction f.



c) Les fonctions existantes

Il existe déjà un très grand nombre de fonctions stockées dans des bibliothèques ou modules que l'on importe en début de programme. Voir le chapitre Bibliothèques.

d) Mes fonctions

Le programmeur peut créer ses propres fonctions et enrichir une ou des bibliothèques personnelles. Les lignes de création d'une fonction sont les suivantes :

- la première ligne du bloc d'instructions commence par le mot clé *def*, suivi du nom de la fonction avec entre parenthèses les éventuels paramètres et terminée par : ;
- on place une indentation pour les instructions ;
- la dernière ligne du bloc d'instructions commence par le mot clé *return*, suivi entre parenthèses du(des) résultat(s) renvoyé(s) par la fonction.

Exemple :

Fonction calculant le volume d'un pavé

Définir fonction
volpav $\leftarrow L \times l \times h$
Renvoyer le volume

```
def volpav(a,b,c):
  return a*b*c
```

Le mot clé *def* permet la construction d'une fonction.
Le mot clé *return* permet à la fonction de renvoyer un résultat.
Les paramètres sont des variables nécessaires au bon fonctionnement de la fonction ; certaines d'entre-elles n'admettent aucun paramètre.
Remarques :

- on choisit le nom de la fonction en rapport avec ce qu'elle réalise pour rendre le programme principal plus lisible, plus intelligible ;
- on retrouve le double-point : et l'indentation pour indiquer le début et la fin de la fonction ;
- de nombreuses bibliothèques de fonctions sont disponibles ; il faut au préalable les importer avec le mot clé *import*.

6. Bibliothèques ou modules

Les bibliothèques sont des collections de fonctions prédéfinies que l'on appelle si besoin en début de programme avec les instructions **from** et **import**, à noter que import* permet d'importer toutes les fonctions de la bibliothèque et non une liste.
Les principales Bibliothèques que vous utiliserez sont :

- math : qui définit des fonctions mathématiques comme la racine carrée (sqrt) ou les fonctions trigonométriques (cos, sin, ...) mais aussi certaines constantes comme le nombre π (pi) ;
- random : nécessaire pour modéliser le hasard ;
- matplotlib : nécessaire pour représenter graphiquement une fonction, une suite de points, ...

a) Constitution d'une bibliothèque

Aire d'un carré

L'aire d'un carré est obtenue en élevant au carré la longueur d'un côté donc on peut traduire cela avec une fonction telle que :

```
def Scarre(c):
  return c**2
```

Bibliothèque de fonctions calculant des aires

On crée un fichier *Surfaces.py* dans lequel on stocke les fonctions calculant les aires. On importe le module **math** pour avoir le nombre π.

```
from math import*

def Scarre(c):
  return c**2

def Sdisque(r):
  return pi*r**2

def Srectangle(l,h):
  return l*h

def Striangle(b,h):
  return b*h/2

def Strapeze(b,B,h):
  return (b+B)*h/2
```

Utilisation de la bibliothèque *Surfaces*

En début de programme j'importe la bibliothèque désirée puis je peux utiliser les fonctions qui y sont définies.

Ce programme demande à l'utilisateur de renseigner une longueur, une base et une hauteur puis affiche l'aire d'un rectangle de longueur l et de hauteur h, l'aire d'un triangle de base b et de hauteur h, l'aire d'un carré de côté l, l'aire d'un disque de rayon b et enfin l'aire d'un carré de côté 5.

```
from Surfaces import*

l=float(input("longueur ="))
b=float(input("base ="))
h=float(input("hauteur = "))

print(Srectangle(l,h))
print(Striangle(b,h))
print(Scarre(l))
print(Sdisque(b))
print(Scarre(5))
```

7. Simulations du hasard

La bibliothèque random propose des fonctions qui simplifient la simulation d'une expérience aléatoire ; par exemple :

- random() : renvoie un décimal appartenant à l'intervalle [0 ; 1[
- randint(1,6) : renvoie un entier naturel appartenant à l'intervalle [1 ; 6]
- uniform(1.5,3) : renvoie un décimal appartenant à l'intervalle [1.5 ; 3]

Si l'on souhaite simuler n lancers d'un dé à 6 faces, on peut définir une fonction qui prend le nombre de lancers en paramètres et ensuite on exécute une boucle bornée. Soit on affiche les résultats au fur et à mesure , soit on les stocke dans une liste. Dans l'exemple ci-dessous, $n = 15$.

Affichage de tous les résultats

```
n=int(input("Nb de lancers = "))
for i in range(1,n+1):
  print("Lancer ",i," =",randint(1,6))
```

Stockage des résultats

```
def lance_de(n):
  L=[randint(1,6) for i in range(n)]
  return L

lance_de(15)
```

8. Représentations graphiques

La bibliothèque matplotlib propose des fonctions qui permettent de représenter des courbes, des graphes, ...

- plt.plot() : trace des nuages de points ou des courbes représentatives ;
- plt.grid() : affiche le quadrillage ;
- plt.show() : en fin de script affiche la fenêtre graphique ;
- ...

Affichage par défaut

```
import matplotlib.pyplot as plt

abscis=[-2+i*.04 for i in range(101)]
ordon=[x**2 for x in abscis]
plt.plot(abscis,ordon)
plt.show()
```

Courbe en pointillés rouges et grille

```
import matplotlib.pyplot as plt

abscis=[-2+i*.04 for i in range(101)]
ordon=[x**2 for x in abscis]
plt.plot(abscis,ordon,'r--')
plt.grid()
plt.show()
```

9. Listes

Une liste est une collection ordonnée d'objets (nombres, caractères, ...). Une liste est codifiée en Python par une paire de crochets, [ ]. Les éléments la composant sont saisis entre les crochets et sont séparés par une virgule. Chaque élément de la liste est repéré, par son indice, sachant que l'indice du premier élément est 0, du deuxième 1, etc.

Exemples de définition de listes :

- *liste contenant les premiers nombres pairs : P=[0,2,4,6,8] ;*
- *liste contenant les mois du premier trimestre : M=["janvier","février","mars"] ;*
- *liste contenant des nombres et des caractères : L=[1,"a",3,"b",2,"c"] ;*
- *V=[ ] est une liste vide, elle ne contient pas d'objets.*

Les premiers objets sont repérés par P[0], M[0], L[0] et ils sont égaux respectivement à 0, janvier, 1. P[2] donne 4 ou L[5] donne c.

La longueur d'une liste ou le nombre d'objets composant une liste est obtenue par la commande **len**.

Exemples : len(P) renvoie 5, len(M) renvoie 3 et len(L) renvoie 6.

a) Créations de listes

Plusieurs possibilités existent pour générer une liste.

En extension	Par ajouts successifs	En compréhension
On saisit tous les objets de la liste pour la définir. *Exemple : J=["lundi","mardi", "mercredi","jeudi","vendredi","samedi", "dimanche"]* On peut automatiser la construction par l'usage de la commande **list(range())**. *Exemple : L=list(range(1,15,2)) créera la liste L composée des objets 1, 3, 5, 7, 9, 11 et 13.*	Aux objets présents dans la liste on vient en ajouter de nouveaux par la commande **append()**. *Exemple : J=["lundi","mardi", "mercredi","jeudi","vendredi","samedi"] avec J.append("dimanche") on obtiendra : J=["lundi","mardi","mercredi","jeudi", "vendredi","samedi","dimanche"]* *["a","c"]+["e","b"] permettra d'obtenir ["a","c","e","b"]*	La liste est définie par une propriété caractéristique ou un test. *Exemple : L=[x**2 for x in range(4)] permettra d'obtenir [0,1,4,9]* *Soit la liste I=[1,3,5,7,9], Carre=[x**2 for x in I if x<7] donnera le carré des nombres 1, 3 et 5 soit Carre=[1,9,25]*

b) Opérations sur des éléments d'une liste

On vient de voir comment ajouter un ou des objets en fin de liste, mais certaines fois on souhaite insérer un objet à l'intérieur d'une liste. Ceci sera réalisable par la commande **insert()** qui insère un objet à une position donnée.

Exemple : soit la liste lp=[1,3,5,3], lp.insert(2,4) permettra d'obtenir lp=[1,3,4,5,3]. Par la suite, lp.insert(3,1) permettra d'obtenir lp=[1,3,4,1,5,3]

Supprimer un objet d'une liste n'est pas plus compliqué, plusieurs commandes existent **remove()** pour supprimer la première occurrence d'une valeur ou **del** pour supprimer un élément connaissant son indice.

Exemple : soit la liste lp=[1,3,4,1,5,3], lp.remove(1) permettra d'obtenir lp=[3,4,1,5,3] et del lp[4] permettra d'obtenir lp=[3,4,1,5]

c) Utilisations des listes

On peut parcourir une liste de gauche à droite ou de droite à gauche, par convention de gauche à droite on utilise des indices positifs (0 inclus) et de droite à gauche des indices négatifs.

Exemple : soit la liste lp=[3,4,1,5], lp[1] renverra 4 alors que lp[-1] renverra 5 ; ici lp[2] et lp[-2] renverront 1
On peut itérer (répéter une action) sur les éléments d'une liste, on obtiendra tous les éléments de la liste à l'aide de l'instruction x for x in *liste*.

*Exemple : si l'on souhaite stocker l'image des nombres de la liste lp par la fonction $f(x) = 2x - 3$ dans la liste result, on écrira result=[2*x-3 for x in lp]*
Il n'existe pas une solution unique pour créer une liste répondant à une question. Malgré tout, il faut rechercher la simplicité dans la construction des listes pour qu'elles restent compréhensibles par les différents intervenants sur un programme.

Exemple : on souhaite créer la liste des multiples de 13 inférieurs à 100.

En extension

On saisit la liste de tous les multiples de 13 correspondants.
L=[0,13,26,39,52,65,78,91]

ou en encore avec la commande range, au préalable on doit créer une liste contenant 8 objets ici par défaut on choisit 0 donc initialement nous avons L=[0,0,0,0,0,0,0,0]

```
L=[0]*8
for i in range(8):
    L[i]=13*i
```

Par ajouts successifs

Avec une boucle bornée

```
L=[]
for i in range(8):
    L.append(13*i)
```

ou en encore

```
L=[]
for n in range(0,101,13):
    L.append(n)
```

Avec une boucle non bornée

```
L=[]
i=0
while 13*i<=100:
    L.append(13*i)
    i=i+1
```

ou en encore

```
L=[]
i=0
while n<=100:
    L.append(n)
    n=n+13
```

Par compréhension

Sans condition
*L=[13*i for i in range(8)]*
ou
L=[n for n in range(0,101,13)]

Avec condition
L=[n for n in range(101) if n%13==0]

10. Vers d'autres horizons -> à rectifier pour la leçon

Menu Suites

Numworks
Créer une suite

Sommer des termes

Calculer des termes
Y. Monka

Nombre d'or
M. Launay