Inspiré de « L'informatique pas à pas »
Frédéric Butin - *Ellipses*

I Algorithme

On définit couramment un algorithme comme un ensemble d'instructions permettant de résoudre, en un temps fini, un problème donné. Un tel algorithme est souvent exprimé avec une notation indépendante de tout langage de programmation.

La structure d'un algorithme est à peu près toujours la même, avec trois parties :
▷ Le nom, l'auteur et la date de l'algorithme (en commentaires) ;
▷ La déclaration des variables et des constantes (qui souvent n'est pas nécessaires en Python) ;
▷ Le traitement des actions, conditions ... regroupées dans des sections « début... fin », des accolades ou des indentations.

II Types d'objets

II.1 Entiers et flottants

Différents types d'objets existent en Python. La commande `type` donne le type d'un objet (vu dans le document 1).

On peut convertir l'entier -20 en un nombre flottant en écrivant `float(-20)` et réciproquement, peut convertir le flottant $-24,3$ en un entier en saisissant `int(-24.3)` (qui donnera -24). La partie entière est donnée par la commande `floor` de la librairie `math`. D'ailleurs, prendre garde à `floor(-24.3)` donne ....
La commande `round` permet d'obtenir un arrondi, et l'on peut préciser le nombre de décimales souhaité.

Exemple : `round(16.485632, 2)` donne ...

II.2 Variables

Une variable est un référence à une « case mémoire » dans laquelle on peut mettre un objet (cela peut être un nombre, une chaîne de caractères, une liste, une fonction, ...).
Par exemple, pour créer une variable a qui fait référence à l'objet 13 (on dit que l'on affecte 13 à la variable a), on écrit $a = 13$.

On peut créer plusieurs variables en même temps (vu dans le document 1)

L'instruction `b,c=c,b` échange le contenu des variables b et c ; cela évite le recours à une troisième variable.

Très important : On peut ajouter une valeur à une variable donnée (c'est ce que l'on appelle `l'incrémentation`). L'instruction `a=a+4` (qui peut s'écrire `a+=4`) remplacera le contenu de a par la valeur de a plus 4.

II.3 Booléens

Les constantes booléennes sont `True` et `False`. Par exemple, si l'on veut savoir si l'inégalité $4 < 13$ est vraie, on entre `4<13` et Python renvoie la valeur `True`.

On teste l'égalité et la non égalité de deux valeurs au moyen des instructions `==` (égalité) et `!=` (non égalité).

La conjonction (resp. la disjonction) est donnée par `and` (resp. `or`). Quant à la négation, elle se note `not`.

II.4 Chaînes de caractères

Une chaîne de caractères est une suite de caractères placée entre guillemets ou entre apostrophes.
Par exemple `x='vacances'`. On peut **concaténer** des chaînes de caractères grâce à l'opérateur +. On peut ainsi écrire `'vive les '+x` .

La commande `str` convertit un entier ou un flottant en chaîne de caractères.
`str(4)` donne le caractère '4'. On a également `int('24')` qui convertit la chaîne '44' en nombre entier 44.

II.5 Listes

Une liste en Python est une suite finie d'objets séparés par des virgules qui est placée entre crochets.

Un exemple de liste est `L=[2,-56.1,'bonjour',False]`.

C'est en fait un tableau au sens où l'on peut accéder facilement à n'importe quelle « case » indépendamment de la longueur de la liste , dynamique dans la mesure où l'on peut ajouter, retirer des éléments, modifier la valeur des éléments.

Attention, toute liste est indexée à partir de 0 (et non 1). La liste vide est notée `[]`. La commande `len` retourne le nombre d'éléments de la liste et `L[j]` retourne le $j-$éme élément de la liste (son index est $j-1$).

Opérations sur les listes :

▷ On peut ajouter un élément à la fin d'une liste grâce à l'extension `append`
Exemple : créer une liste et lui ajouter un élément.

▷ On peut concaténer deux listes avec l'opérateur +. On peut également construire des listes avec l'opérateur *.
Exemple : créer une liste de 50 zéros.

▷ La commande `del` supprime un élément de la liste. Exemple `del(L[2])`.

▷ L'extension `pop` supprime le dernier élément de la liste et le récupère. Exemple `z=L.pop()`.

▷ L'instruction `L.reverse()` (avec l'extension `reverse`) modifie la liste L en renversant l'ordre de ses éléments.

▷ L'instruction `L[i:j]` renvoie la sous-liste de L formée des éléments d'index compris dans l'intervalle $[\![i, j[\![$.
Si l'on ne met pas l'index i, Python le remplace automatiquement par 0. Exemple : `L[:3]`.
Si l'on ne met pas l'index j, Python le remplace automatiquement par la taille de la liste. Exemple : `L[1:]`.
On peut accéder aux éléments d'une liste en utilisant des indices négatifs. Exemple : `L[-1]` permet de renvoyer le dernier élément de la liste L.

.

III Algorithmique

III.1 Gestion des entrées : saisie

En Python, on utilise la commande `input` ; mais par défaut, la valeur saisie et affectée à la variable est de type `string` (chaîne de caractères). Si l'on souhaite recueillir une valeur numérique de type entière ou flottante, il faut la convertir. On procèdera de la façon suivante :

```
a=input()
a=int(a) # pour une variable de type entier ou directement a=int(input())
```

III.2 Fonctions

Lu dans "Wikibooks"

« *La programmation est l'art d'apprendre à un ordinateur comment accomplir des tâches qu'il n'était pas capable de réaliser auparavant. L'une des méthodes les plus intéressantes pour y arriver consiste à ajouter de nouvelles instructions au langage de programmation que vous utilisez, sous la forme de fonctions originales. De plus, créer ses propres fonctions permet de factoriser le code redondant en pouvant être appelées depuis plusieurs endroits.* »

Un premier exemple de fonction (à écrire dans un fichier : `New File`

```
def salutation(nom):
    print('Bonjour '+nom)
```

On exécute cette fonction en écrivant par exemple `salutation('Michel')` (on dit que l'on « appelle » la fonction). La variable `nom` sera remplacée par la chaîne de caractères `'Michel'` et la commande `print` permettra l'affichage du message `'Bonjour Michel'`.

Remarque 1 Le décalage de la deuxième ligne de la fonction s'appelle **l'indentation**. Elle est obligatoire et remplace en Python, les commandes « begin » et « end » présentent dans d'autres langages.

Important : Pour qu'une fonction renvoie un résultat (qui peut être stocké dans une variable), il faut utiliser la commande `return`.

Un exemple de fonction :

```
def carre(n):
    return(n*n)
```

`x=carre(12)` permet d'affecter à la variable x, la valeur renvoyée par la fonction `carre`.

III.3 Variables locales et variables globales

Par défaut, les variables utilisées dans une fonction sont locales, c'est à dire qu'elles ne sont pas définies en dehors de la fonction.

Ainsi, si dans une fonction, on définit la variable locale x à laquelle on donne la valeur 4, et si l'on demande ensuite, en dehors de la fonction, ce que vaut x, un message d'erreur sera renvoyé précisant que x n'est pas affectée.

III.4 Tests

La syntaxe d'un test est la suivante :

```
if condition1:
   bloc instructions 1
else:
   bloc instructions 2
```

```
if condition1:
   bloc instructions 1
elif condition2:
   bloc instructions 2
......
else:
   bloc instructions 3
```

Voici un exemple de test dans une fonction :

```
def parite(n):
    if n%2==0:
        print('Le nombre ',n,' est pair')
    else:
        print('Le nombre ',n,' est impair')
```

L'appel de `parite(157)` déclenchera l'affichage de la phrase :

```
'Le nombre 157 est impair'
```

III.5 Boucles

On distingue deux types de boucles.

→ Les **boucles inconditionnelles** (« boucles for »), dans lesquelles un bloc d'instructions est effectué pour chaque valeur prise par un compteur parcourant un ensemble de valeurs.

→ Les **boucles conditionnelles** dans lesquelles un bloc d'instructions est effectué tant qu'une condition est vraie (de valeur `true`).

La syntaxe des deux types de boucles est la suivante :

```
for j in liste:
    bloc instructions
```

```
while condition:
      bloc instructions
```

Mise en œuvre :

1. Soit n un entier naturel non nul. On note S_n la somme des n premiers entiers naturels,

$$S_n = 1 + 2 + 3 + \ldots + (n-1) + n$$

Écrire S_{10} et la calculer.
On souhaite écrire un algorithme qui calcule la somme S_n lorsque l'on saisit au clavier le nombre n. Voici une proposition pour calculer S_4.

```
Déclaration des variables
    S est un entier
Début
    S⟵0
    S⟵S+1
    S⟵S+2
    S⟵S+3
    S⟵S+4
    Afficher S
Fin
```

Quelle est la valeur de S à la sortie de cet algorithme ?

Quel inconvénient voyez-vous à cet algorithme ?

Une nouvelle structure algorithmique :

Modifier l'algorithme précédent de sorte qu'il puisse calculer S_n lorsque l'on saisit n au clavier.

EXERCICE 1 Écrire un algorithme qui demande un nombre de départ, et qui ensuite affiche les 10 nombres suivants. Par exemple, si l'utilisateur entre le nombre 20, la programme affichera les nombres de 21 à 30.

EXERCICE 2 Écrire un algorithme qui demande un nombre de départ, et qui ensuite écrit la table de multiplication de ce nombre, présentée comme suit (l'utilisateur saisit 4) :

Table de 4 :
$4 \times 1 = 4$
$4 \times 2 = 8$
$4 \times 3 = 12$
...
$4 \times 10 = 40$

2. Une personne souhaite disposer d'une somme de 1000€ pour un achat. Elle peut mettre 70€ de côté tous les mois. Utiliser une boucle conditionnelle pour déterminer le nombre de mois nécessaires à l'obtention de cette somme.