Boîte à outil Python

I. Typage, vocabulaire

I.1. Les types à connaître

Définition 1

- le type **entier** (integer) qui correspond aux entiers relatifs ;
- le type **flottant** (float) qui correspond aux nombres décimaux ;
- le type **chaîne de caractère** (string), qui permet d'implémenter des messages textuels ;
- le type **booléen** (boolean) qui regroupe les mots clefs `True` et `False`.
- le type **list** (liste) utile en statistique et pour réaliser des graphiques.

I.2. Pot-pourri de remarques

- les entiers (int) sont *a priori* illimités ;
- les flottants (float) sont compris entre $\pm 1{,}7 \times 10^{308}$ (32bits) ou $\pm 3{,}4 \times 10^{308}$ (64bits), avec une limite basse en valeur absolue à 10^{-324} ;
- les nombres flottants parfois représentés en notation scientifique : $M \times 10^e$ (une mantisse M, nombre décimal compris strictement entre -10 et 10, et un exposant entier e).
- les chaînes de caractère (string) sont délimitées à l'aide d'apostrophes ' ou de guillemets anglais " (on peut néanmoins afficher un de ces deux caractères grâce à un anti-slash \ (`Alt Gr + 8`) en faisant par exemple `'l\'apostrophe'`);
- on peut faire un retour charriot ↵ dans une chaîne de caractère avec `\n` ;
- les listes sont délimitées par des crochets `[]` ;
- on peut connaître le type d'un objet en faisant `type()` ;
- si une ligne est « trop longue » (typiquement, 79 caractères), on peut utiliser l'anti-slash \ puis un retour chariot pour changer de ligne sans exécution ;
- les commentaires sont possibles dans un script Python : ils commencent par un dièse #.

I.3. Conversion de types

Définition 2

Des variables peuvent être changées de type avec les commandes

- `int()` pour convertir en entier ;
- `float()` pour convertir en flottant ;
- `str()` pour convertir en chaîne de caractère ;
- `list()` pour convertir en liste.

II. Opérations

Définition 3

Les deux opérations « de base » et leurs opérations associées sont définies :

- l'addition `+` ;
- la soustraction `-` ;
- la multiplication `*` ;
- la division `/` ;
- la puissance `**` (et non pas $\wedge$) ou alors `pow(a,n)`.

D'autres opérations utiles à connaître :

- la division euclidienne `//` (donne le quotient) ;
- le reste de la division euclidienne `%` ;
- la valeur absolue d'un nombre est donné par l'objet intégré `abs(...)`.

Pour les chaînes de caractères :

- l'opération `+` avec des chaînes de caractère fournit la concaténation (on « colle » les deux chaînes) ;
- une multiplication entre un entier k et une chaîne de caractère renvoie la chaîne concaténée par elle-même k fois.

Exemple 1

- $\frac{4+9}{12}$: `(4+9)/12` ;
- $5^4 + 3^3$: `5**4+3**3` ;
- $\frac{25{,}4}{2{,}1\times10^{-6}}$: `25.4/2.1e-6`.

III. Affectation

Définition 4

On affecte à une variable une valeur (numérique ou non) grâce à `=`.

N.B. : Toute nouvelle affectation d'une variable efface la précédente affectation.

Exemple 2

```
a = 5 #on affecte la valeur 5 à la variable a#
a = a + 2 #nouvelle valeur de a : 5 (ancienne) + 2 elle vaut donc 7 #
a = a**2 #la nouvelle valeur de a est 7**2=49 #
u, v = 4, 5 #équivalent à u = 4 puis v = 5 #
u += 1 #équivalent à : u = u+1 #
v *= 2 #équivalent à : v = v*2 #
```

Définition 5

On peut accéder à la dernière valeur évaluée *via* le tiret bas (« barre du 8, underscore ») : `_`.

Définition 6

On peut faire rentrer à l'utilisateur une donnée *via* l'instruction `input()`.

N.B. : Sans indication, l'objet sera considéré comme une chaîne de caractères.
On ainsi utilise donc le forçage du typage pour qu'une entrée ait le type voulu.
Néanmoins, l'usage de cet objet n'est pas très convivial dans une console Python, donc est peu utilisé.

IV. Instruction conditionnelle

IV.1. Booléen

Définition 7

Le résultat d'un **test** est un **booléen** : `True`, `False`.

IV.2. Tests simples

Définition 8

- Test d'égalité : `a == b ;`
- Test de non-égalité : `a != b ;`
- Test inférieur ou égal : `a <= b ;`
- Test strictement inférieur : `a < b ;`
- Test strictement supérieur : `a > b ;`
- Test supérieur ou égal : `a >= b ;`
- Test d'identité : `a is b ;`
- Test d'appartenance : `a in B .`

N.B. : pour $\leqslant$ et $\geqslant$, noter que c'est le symbole d'ordre puis d'égalité.
N.B. 2 : on peut réaliser un test d'inégalité double directement : `1<2<5`.

IV.3. Tests composés

Définition 9

- Test1 ET Test2 : `b1 and b2` : renvoie `True` si les tests `b1` et `b2` sont tous les deux vrais ;
- Test1 OU Test2 : `b1 and b2` : renvoie `True` si au moins un des deux tests `b1` et `b2`est vrai ;
- Non(Test1) : `not(b1)` : renvoie `True` si le test `b1` est faux.

IV.4. Mot clef if

Définition 10

Pour programmer une instruction conditionnelle, on procède ainsi

```
if condition:
    effet
```

IV.5. Alternatives

Définition 11

- Le mot clef `elif` permet de proposer une alternative de test si le test de `if` renvoie `False`.
- Le mot clef `else` permet de donner une instruction pour le cas où le test de `if` (et le cas échéant, ceux de `elif`) renvoie(nt) `False`.

```
if condition:
    effet
elif condition:
    effet
else:
    effet
```

N.B. : `elif` et `else` sont facultatifs ; on peut mettre autant de `elif` que nécessaire.

Exemple 3

```
def signe(x):
    if x > 0:
        return 1
    elif x == 0:
        return 0
    else:
        return -1
```

```
def signe(x):
    if x > 0:
        return 1
    if x == 0:
        return 0
    return -1
```

Pour une fonction, on peut se contenter de ne mettre que des `if`.

V. Boucle bornée

V.1. Itérable

N.B. : $[\![a\,;b]\!] = [a\,;b] \cap \mathbb{Z}$ est « l'intervalle de nombres entiers » compris entre a et b.

Définition 12

L'**itérable** est un ensemble (liste ou tuple).
Le plus classique est fourni par la commande `range`.

- `range(b)` fournit « l'intervalle semi-ouvert » $[\![0\,;b[\![$, soit les nombres entiers compris entre 0 et `b-1` ;
- `range(a, b)` fournit « l'intervalle semi-ouvert » $[\![a\,;b[\![$, soit les nombres entiers compris entre `a` et `b-1` ;
- `range(a, b, k)` fournit l'ensemble $\{a+\ell\times k \mid a+\ell\times k<b, \ell\in\mathbb{N}\}$.

V.2. Mot clef for

Définition 13

Pour programmer une boucle bornée, on procède ainsi

```
for variable in itérable:
    effet
```

La variable `variable` prendra alors successivement toutes les valeurs de l'ensemble `itérable`.
L'indentation est nécessaire pour le programme interprète bien l'espace `effet`.

Exemple 4

```
u = 3
for rang in range(1, 4):
    u = 2*u + 1
print(u)
```

Ce script permet de calculer le terme d'indice 4 de la suite définie par $\begin{cases} u_0 = 3 \\ u_{n+1} = 2u_n + 1 \end{cases}$.

VI. Boucle non bornée

Définition 14

Pour programmer une boucle non bornée, on procède ainsi

```
while condition:
    effet
```

Exemple 5

```
n = 0
while 10**(-n) != 0
    n += 1
print(n)
```

Ce script renvoie le premier entier n tel que $10^{-n} = 0$ pour le programme Python ; il renvoie $n = 324$.

VII. Fonction

Définition 15

Une fonction informatique est créée par

```
def nomfonction(entrée):
    effet
    return sortie
```

Exemple 6

```
def f(x):
    return x**3 - x**2 - x - 1
```

correspond à la fonction d'expression $f(x) = x^3 - x^2 - x - 1$.

N.B. : On peut aussi créer une fonction avec le mot clef `lambda`.

```
nomfonction = lambda x: f(x)
```

, ce qui donne par exemple pour la fonction d'expression $g(x) = x^2 - 3$.

```
g = lambda x: x**2 - 3
```

Définition 16

On peut créer une aide contextuelle (« docstring ») pour expliciter le fonctionnement d'une fonction. L'espace est délimité par trois apostrophes et se place après la ligne avec le mot clef `def`.

Exemple 7

```
def distance_freinage(v, r):
    '''cette fonction renvoie la distance de freinage
    v contient la vitesse en m/s
    r est "s" pour route sèche ou "m" pour route mouillée'''
    if r == 's'
        return (1/202*v + 1/7) * 2*x
    return (1/404*v + 1/7) * 2*x
```

VIII. Fonctions utiles du type liste

VIII.1. Mots intégrés et méthodes

Définition 17

- Pour une liste `Liste` comportant n éléments, on accède à son k-ème élément *via* `Liste[k-1]`, les éléments étant numérotés de 0 à $n-1$.
- `len(Liste)` renvoie la longueur de la liste, *i.e.* son nombre d'éléments.
- `sorted(Liste)` crée une liste dans laquelle les éléments de `Liste` sont rangés par ordre croissant.
- On peut rajouter un objet `a` à une liste *via* `Liste.append(a)`.
  Attention, on écrit juste cette méthode et non `L = L.append(a)`.

Exemple 8

```
def echant(n):
    '''crée une liste de taille n de nombres entiers aléatoires
    compris entre 1 et 27'''
    L =[]
    for exp in range(n):
        L.append(rd.randint(1,27))
    return L
```

VIII.2. Liste par compréhension

Définition 18

On peut créer une liste selon la procédure suivante

```
Liste = [f(x) for x in range(a, b)]
```

Exemple 9

```
Liste1 = [x**2 for x in range(0, 11)]
```

crée la liste $\{0\,;1\,;4\,;9\,;16\,;25\,;36\,;49\,;64\,;81\,;100\}$.

VIII.3. Somme d'une liste

Définition 19

La commande suivante est l'équivalent de notre $\sum_k$:

```
import math
math.fsum(terme(k) for k in itérable)
```

Exemple 10

```
import math
S = math.fsum(1/k**2 for k in range(1, 101))
```

calcule la somme $\frac{1}{1^2}+\frac{1}{2^2}+\cdots+\frac{1}{100^2}$.

N.B. : Python possède de base un objet intégré `sum` mais `math.fsum` évite certaines erreurs de calcul dues à l'usage des flottants.

IX. Bibliothèques

IX.1. Chargement d'une bibliothèque

De base, Python ne possède pas un certain nombre d'outils dont on a besoin pour faire des mathématiques : fonctions numériques, simulateurs de nombre pseudo-aléatoire, outils pour générer des graphiques.
Plutôt que de tout recréer soi-même, des bibliothèques sont téléchargeables afin de palier à ces lacunes.

Définition 20

Si la bibliothèque est correctement installée, il suffit de rentrer

```
import bibliothèque
```

La bibliothèque est alors chargée et on peut utiliser toute fonction la composant *via*

IX.2. Bibliothèque math

Elle possède toutes les fonctions numériques : `sqrt, cos, sin, tan`.

Définition 21

- `math.sqrt(x)` donne une valeur approchée de $\sqrt{x}$.
- `math.cos(x)` donne une valeur approchée de $\cos(x)$.
- `math.sin(x)` donne une valeur approchée de $\sin(x)$.
- `math.tan(x)` donne une valeur approchée de $\tan(x)$.

IX.3. Bibliothèque random

Elle possède des fonctions permettant de générer des nombres pseudo-aléatoires (un ordinateur ne fait pas de choix, ces nombres sont obtenus *via* des algorithmes garantissant qu'ils possèdent des propriétés attendues du hasard).

On l'importe souvent *via* `import random as rd`.

Définition 22

- `rd.random()` simule un nombre aléatoire sur $[0\,;1]$.
- `rd.randint(a, b)` donne un nombre entier aléatoire sur $[\![a\,;b]\!]$.