Boîte à outil Python

1 Typage, vocabulaire

1.1 Les types à connaître

Définition 1

- le type **entier** (integer) qui correspond aux entiers relatifs ;
- le type **flottant** (float), souvent représenté en notation scientifique : $M \times 10^e$ (une mantisse M, nombre décimal compris entre -10 et 10, et un exposant entier e) qui correspond aux nombres décimaux ;
- le type **chaîne de caractère** (string), qui permet d'implémenter des messages textuels ;
- le type **list** (liste) utile en statistique et pour réaliser des graphiques.

1.2 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 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 chaînes de caractères sont encodés en Unicode (quand Python est bien installé !), donc pas de soucis d'accents & co ;
- les listes sont délimitées par des crochets `[]` ;
- on peut connaître le type d'un objet en faisant `type(o)`.

1.3 Forcer un typage

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.

1.4 Quelques types « bonus »

- le type **complex** (en fait deux flottants), à définir par exemple en `z=1+2j` ;
- le type **tuple** (t-uplet), qui sert aussi à créer des listes, par exemple `t=(1,2.7,'box')` ;
- le type **dict** (dictionnaire), qui sert à créer des répertoires : `d={'seconde':17 ;'première es':3 ;'première es_ l':1}`.

N.B. : les tuple sont délimités par des parenthèses `()` et sont immuables (on ne peut pas y rajouter ni enlever d'éléments), *a contrario* des listes (list, dites mutables).

1.5 Vocabulaire

On utilisera essentiellement, en plus des variables

- des **objets intégrés** qui nécessitent de mettre des parenthèses `()` pour identifier l'objet ou les objets sur lequel (lesquels) elle s'applique : `abs(-4)`, `min(-4,7,9)` ;
- des **mots clefs** qui prennent en argument l'objet qui le suit directement après un espace : `import numpy`, `2 is 4.0/2` ;
- des méthodes qui se placent après un objet, juxtaposé avec un point : `liste.append(60)`, `liste.remove(50)`.

2 Affectation

Définition 3
On affecte à une variable une valeur (numérique ou non) grâce à `=`.

N.B. : Toute nouvelle affectation d'une variable efface la précédente affectation.
N.B. 2 :

```
a=2
b=7      est équivalent à      a,b,c=2,7,0
c=0
```

Définition 4
On peut accéder à la dernière valeur évaluée *via* le tiret bas (« barre du 8, underscore ») : `_`.

Définition 5
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 est le type voulu.

↪ Exemple 1

```
m=input() #sera traité en str#

n=int(input())
n+=3
print(n)

n,p=int(input()),int(input())
n*=5
p-=2
print(n,p)
```

3 Opérations

Définition 6

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 `%`.

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.

N.B. :

```
n+=1          est équivalent à   n=n+1
a-=3                             a=a-3
c*=0.9                           c=c*0.9
h/=2                             h=h/2
```

4 Instruction conditionnelle

4.1 Booléen

Définition 7

Le résultat d'un **test** est un **booléen** : `True, False`.
Python les traite en objets intégrés.

N.B. : Ces objets sont aussi « la base » des systèmes informatiques, car peuvent représenter les bits (circuit ouvert/fermé, aimant polarisé Sud/Nord, fibre optique allumée/éteinte, *etc.*).

4.2 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.

4.3 Tests composés

Définition 9

- Test1 ET Test2 : `b1 and b2;`
- Test1 OU Test2 : `b1 or b2;`
- Non(Test1) : `not(b1).`

N.B. : le test « OU BIEN » (XOR, OU exclusif) n'est pas implémenté mais on peut le programmer.

```
def xor(P,Q) :
    return (not(P) and Q) or (P and not(Q))
```

4.4 Mot clef if

Définition 10

Pour programmer une instruction conditionnelle, on procède ainsi

```
if condition :
    effet
```

4.5 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.
N.B.2 : l'instruction `pass` permet qu'une alternative ne soit pas examinée (utile quand on conçoit un programme, ou pour détecter les erreurs).

↪ Exemple 2

```
def premier_degré(a,b) :
    if a==0 and b==0 :
        print('Tout nombre réel est solution')
        return True
    elif a==0 :
        print('Aucun nombre réel n\'est solution')
        return False
    else :
        print('L\'unique solution est',-b/a)
        return -b/a
```

5 Boucle bornée

5.1 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 » $[\![0\,;b[\![$;
- `range(a,b)` fournit « l'intervalle semi-ouvert » $[\![a\,;b[\![$;
- `range(a,b,k)` fournit l'ensemble $\{a+\ell\times k \mid a+\ell\times k<b, \ell\in\mathbb{N}\}$.

N.B. :

```
def frange(start,stop,step) :
    i=start
    while i <stop :
        yield i
        i += step
```

fournit un itérable avec des flottants, tout comme `arange` de la bibliothèque `numpy`.

5.2 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 3

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

5.3 « Casser » une boucle

Définition 14

Certains cas peuvent amener à prendre le cas de ne pas énumérer une boucle jusqu'à son terme défini au départ :

- l'instruction `break` met fin au programme ;
- l'instruction `continue` met fin à la boucle, mais le programme reprend juste après la boucle.

Ces fonctions sont à combiner avec une instruction conditionnelle pour être pertinente.

↪ Exemple 4

```
def lt(n) :
    t=0
    for essai in range(n) :
        if randint(n)==n :
            print('Le lièvre a gagné !')
            return 0
            break
        else :
            t=t+1
            if t==n :
                print('La tortue a gagné !')
                return 1
```

6 Boucle non bornée

Définition 15

Pour programmer une boucle non bornée, on procède ainsi

```
while condition :
    effet
```

↪ Exemple 5

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

7 Fonction

Définition 16

Une fonction informatique est créée par

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

↪ Exemple 6

```
def f(x) :
    return 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

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

8 Fonctions utiles du type liste

8.1 Mots intégrés et méthodes

Définition 17

- Pour une liste `Liste` comportant n éléments, on accède à son k-ième 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 7

```
def médiane(série) :
    L=sorted(série)
    N=len(série)
    if N%2==0 :
        return (L[N//2-1]+L[N//2])/2
    else :
        return L[(N+1)//2]
```

8.2 Liste par compréhension

Définition 18

On peut créer une liste selon la procédure suivante

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

Exemple 8

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

Liste2=[2*x-7 for x in range(-5,6)]
print(Liste2)
```

8.3 Objet intégré sum

Définition 19

La commande suivante

```
sum( terme(k) for k in itérable )
```

est l'équivalent de notre $\sum_{k}$.

Exemple 9

```
S=sum(1/k**2 for k in range(1,10001))
print(S)
```

9 Bibliothèques

9.1 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*

```
import bibliothèque
bibliothèque.fonction
```

N.B. : On peut alléger un peu l'appel de fonction d'une bibliothèque.

1. Raccourcir le nom de la bibliothèque.

   ```
   import bibliothèque as bb

   bb.fonction
   ```

2. Quand on a une seule fonction de cette bibliothèque que l'on va utiliser.

   ```
   from bibliothèque import fonction
   fonction
   ```

3. Méthode bourrin (tend à ralentir le système).

   ```
   import bibliothèque as *
   fonction
   ```

9.2 Bibliothèque numpy

Elle possède toutes les fonctions numériques : `sqrt, cos, sin, tan, log, exp`.
On l'importe souvent *via* `import numpy as np`.

Définition 21

- `np.cos(x)` donne une valeur approchée de $\cos(x)$.
- `np.sin(x)` donne une valeur approchée de $\sin(x)$.
- `np.tan(x)` donne une valeur approchée de $\tan(x)$.
- `np.exp(x)` donne une valeur approchée de $\exp(x)$.
- `np.log(x)` donne une valeur approchée de $\ln(x)$.
- `np.arccos(x)` donne une valeur approchée de $\arccos(x)$.

Définition 22

- `np.rint(a)` donne l'arrondi à l'entier près de `a` ;
- `np.floor(a)` donne la partie entière (arrondi à l'entier près par défaut) de `a` ;
- `np.round(a,n)` donne l'arrondi à 10^{-n} près de `a`.

9.3 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 23

- `rd.random()` simule un nombre aléatoire sur $[0;1]$.
- `rd.randrange(a,b)` donne un nombre entier aléatoire sur $[\![$`a` ; `b-1`$]\!]$.
- `rd.randint(a,b)` donne un nombre entier aléatoire sur $[\![$`a` ; `b`$]\!]$.
- `rd.uniform(a,b)` donne un nombre aléatoire sur [`a` ; `b`].
- `rd.gauss(m,s)` donne un nombre aléatoire par simulation d'une loi normale de paramètres `m` et `s`.

N.B. : `-np.log( rd.random() )/a` donne ainsi une nombre par simulation d'une loi exponentielle de paramètre `a` (attention, `lambda` est réservé sur Python).
N.B. 2 : On peut réinitialiser les générateurs grâce à la commande `seed()` (qui utilise l'horloge interne de l'ordinateur).