Fiche 1 - Objets manipulés, opérations élémentaires

Il existe plusieurs implémentations du langage PYTHON. Nous utilisons l'implémentation Pyzo (gratuit) :

http://www.pyzo.org/downloads.html

Elle peut être installée sur clé usb (2Go).
Pour dépanner, il existe des versions online : basthon

I. VARIABLE ET AFFECTATION

Une **variable** *informatique* se définit par

- un **nom**,
- un **type** (nature de l'objet, ensemble des opérations qui lui correspondent, représentation machine)
- et une **valeur** (son contenu).

L'**affectation** est l'opération qui permet de modifier la valeur d'une variable, notée = :

```
>>> a=9+3.1
>>> a
12.1
```

La variable de nom "a" prend la valeur 12,1 (et le type associé, ici `float`). Ce qui était dans la variable "a" a été effacé et remplacé par 12,1.

Attention ! Les nombres à virgule se notent avec un point (notation anlgosaxone).
Remarque : Des affectations peuvent être simultanées.

```
>>> a,b=1,2
>>> a
1
>>> b
2
>>> c
NameError: name 'c' is not defined
```

La console PYTHON permet d'effectuer des calculs comme une calculatrice : on indique les opérations à effectuer, puis on appuie sur la touche entrée pour les exécuter.

```
>>> 3+4*5
23
```

Remarque : Les instructions séparées par des point-virgules sont exécutées les une après les autres, dans l'ordre d'écriture.

```
>>> a=2+2;b=4;a=6
>>> a;b;
6
4
```

Attention ! L'utilisation d'une virgule est associée à une structure de tableau. Elle ne peut être utilisée pour séparer les instructions :

```
>>> a=7,b=8
SyntaxError: can't assign to literal
```

Les différents types étudiés sont : entier, nombre réel, nombre complexe, tableau, chaîne de caractères et booléen.
Remarque : Le même nom peut successivement être attribué à des variables de types différentes : c'est le ***typage dynamique***. Lors de l'affectation, les anciennes caractéristiques d'une variable sont modifiées en fonction de la nouvelle expression.

```
>>> a=2;type(a)
<class 'int'>
>>> a=2==3;type(a)
<class 'bool'>
```

➜ Règles pour le nom d'une variable :

- commencer pour une lettre minuscule, une lettre majuscule ou le caractère _ ;
- compléter avec des chiffres, des lettres ou _
- ne pas être un mot du langage comme : `global`, `in`, ...

II. OBJETS ÉLÉMENTAIRES

II.1. NOMBRES ENTIERS, RÉELS OU COMPLEXES

Les nombres manipulés par PYTHON sont typés : nombres entiers `int` (pour "integer"), nombres réels `float` (pour nombre à virgule) et les nombres complexes `complex`. Par défaut, il n'y a pas de type dédié aux fraction (il existe un module spécifique).

```
>>> a=3/3;b=1;c=1+0j;a;b;c;
1.0
1
(1+0j)
>>> type(a);
<class 'float'>
>>> type(b)
<class 'int'>
>>> type(c)
<class 'complex'>
```

Voici quelques *constantes* prédéfinies :

Code PYTHON	Constante
1j	i avec $1j * 1j = -1 + 0j$
pi	$\pi = 3.1415927\ldots$
e	$e = 2.7182818\ldots$
inf	infini ($\pm\infty$)
nan	indéterminé (not a number)

Lycée Saint-Exupery – maffre.saintexupery@gmail.com – MPSI

```
>>> a=inf-inf;a;
nan
>>> nan*inf;
nan
>>> 1/inf;
0.0
```

Les opérations usuelles sont :

Notation	Opération
+	Addition
-	Soustraction
*	Multiplication
/	Division
10**(-3)	Puissance (ici 10^{-3})
1.5e-1 ou 1.5E-1	Notation scientifique

```
>>> 2+1/3
2.3333333333333335
>>> 4/2*2
4.0
>>> 10E-5
0.0001
```

➜ L'exemple suivant montre comment incrémenter de 1 la valeur d'une variable ou encore la doubler :

```
>>> a=3;a
3
>>> a=a+1;a
4
>>> a=a*2;a
8
```

Remarque : Deux objets de types différents ne peuvent pas, en général, être associés dans une même opération. Par exemple la somme d'une variable de type *nombre* et d'une variable de type *fonction* donne un message d'erreur. Cependant, deux situations sont possibles :

➢ Lorsqu'il y a ***héritage de type*** : un type A définit un *sous-ensemble* d'un type B, plus général ; alors les opérations ou fonctions valident sur le type B s'appliquent aussi, *par héritage*, aux variables du type A. Par exemple, les fonctions s'appliquant à un nombre réel (`float`) s'appliquent aussi à un nombre entier (`int`). Les fonctions s'appliquant à un nombre complexe, s'appliquent aussi à un nombre réel. Le contraire peut être faux.
De même, il est possible de faire la somme d'un entier et d'un réel ; le résultat est un réel.

➢ Lorsque PYTHON effectue une **interprétation** dans des situations particulières. Par exemple, la somme d'une nombre et d'une matrice donne une matrice où la somme a été effectuée sur chaque coefficient.

➜ Il existe un raccourci pour les affections *élémentaires*. Les écritures suivantes sont équivalentes :

Affectation	Raccourci
a=a+b	a+=b
a=a*b	a*=b
a=a**b	a**=b

```
>>> a=1;a+=1;a;
2
>>> a*=3;a;
6
>>> a**=2;a;
36
>>> a/=2;a;
18.0
```

II.2. VALEURS LOGIQUES VRAI (True) ET FAUX (False)

Une variable est de type booléen, notée `bool`, lorsque son contenu est soit `True`, soit `False`.

On utilise les valeurs logiques dans les structures conditionnelles (branchement `if`, boucle `while`) où certaines instructions sont exécutées suivant qu'une condition est réalisée ou non. Elles sont le résultat d'opération logique, comme les opérations de comparaison ou les opérations logiques élémentaires (négation, et, ou).

Notation	Opération
==	Égal
!=	Différent
<	Inférieur strict
<=	Inférieur ou égal
>	Supérieur strict
>=	Supérieur ou égal

```
>>> 2!=3
True
>>> a=12
>>> a>8;a==10;a+2==14;
True
False
True
>>> b=1==1+0j;c=1==1.0;b;c;
True
True
>>> type(b)
<class 'bool'>
```

Attention ! Ne pas confondre :

- l'**affectation** : a=2, modifie la valeur de la variable a, elle devient 2 ;
- le **test d'égalité** : a==2, retourne `True` si la valeur de a est 2, et `False` sinon.

Notation	Opération
not	Négation logique
and	*et* logique
or	*ou* logique

```
>>> a=12;a<8;
False
>>> a>4 and a<20
True
>>> a>4 or a<10
True
>>> a>4 and a<10
False
>>> not(True or False)
False
>>> not 1==0 or 1==1
True
>>> not (1==0 or 1==1)
False
```

Remarque : a or b est vrai lorsque a est vrai, ou b est vrai ou a et b sont simultanément vrais (c'est le "ou" inclusif).

Attention ! Le test d'égalité fonctionne sur les entiers mais pas sur les nombres réels (de type `float`).
Pour tester l'égalité de deux nombres réels, il est préférable de majorer la différence absolue par la précision d'écriture des nombres réels sous PYTHON: 10^{-15}.

```
>>> 3*0.1==0.3
False
>>> abs(3*0.1-0.3)<1e-15
True
```

II.3. CHAÎNE DE CARACTÈRES

Les chaînes de caractères, de type `str`, sont délimitées par des apostrophes ou des guillemets.

Notation	Explication
`'...'+'...'`	Concaténation de deux chaînes
`str(x)`	Convertit la valeur de x en chaîne de caractères

```
>>> x='coucou'
>>> type(x)
<class 'str'>
>>> x[0];x[1];
'c'
'o'
>>> 'abc'+'def'
'abcdef'
>>> x=12;x;str(x);
12
'12'
>>> a=3.14;'pi = '+str(a);
'pi = 3.14'
```

Remarque : Il est parfois utile de définir une chaine de caractère tenant compte de la valeur d'une variable. Par exemple, lorsque que l'on sauvegarde un graphe (en .pdf ou autre format) et que ce dernier dépend d'un ou plusieurs paramètres.

```
>>> n=12
>>> nom='figure pour n = '+str(n)+'.pdf';nom;
'figure pour n = 12.pdf'
```

Attention ! Lorsqu'une chaîne de caractère contient une apostrophe, il est préférable d'utiliser les guillemets.

```
>>> "aujourd'hui"
"aujourd'hui"
>>> 'aujourd\'hui'
"aujourd'hui"
```

Attention ! Une chaîne de caractères est une structure indicée immuable, c'est à dire, non modifiable : on ne peut ni modifier, ni ajouter, ni enlever un caractère.

II.4. TABLEAUX

Il y a plusieurs types de tableaux :

⇛ Les listes de type `list` : les éléments peuvent être de types différents.

Type `list`

```
>>> a=[1,2,'abc'];type(a)
<class 'list'>
>>> list() # liste vide
[]
>>> a[0];a[1];
1
2
>>> len(a);
3
```

⇛ Les liste de type `tuple` (les uplets) :

Type `tuple`

```
>>> a=(1,2,'ert');type(a)
<class 'tuple'>
>>> a[2];
'ert'
```

Remarque : Une différence fondamentale entre les deux est qu'il est possible de modifier un élément d'une liste au type `list`, mais pas d'une liste de type `tuple`.

⇛ Les intervalles d'entiers de type `range`

Notation	Explication
`range(a)`	$[\![0, a-1]\!]$
`range(a,b)`	$[\![a, b-1]\!]$
`range(a,b,c)`	$\{a, a+c, a+2c, \ldots, a+nc\}$ avec $a+cn < b \leq a+(c+1)n$ où c est le pas de variation

Type range

```
>>> a=range(1,4);a;
range(1, 4)
>>> type(a)
<class 'range'>
>>> list(range(1,4))
[1, 2, 3]
>>> list(range(5,1,-1))
[5, 4, 3, 2]
>>> list(range(1,13,2))
[1, 3, 5, 7, 9, 11]
```

⇒ Les tableaux de type `numpy.ndarray` : les éléments sont de même type. La structure est une *liste de listes*.

Avec array

```
>>> a=[1,2,3];a=array(a);a;type(a);
array([1, 2, 3])
<class 'numpy.ndarray'>
>>> t=array([[1,2],[3,4],[5,6]]);t;
array([[1, 2],
       [3, 4],
       [5, 6]])
```

Remarque : Une objet est dit ***iterable*** s'il peut se décomposer en éléments qui peuvent être listés : le premier est identifiable, puis le second ... puis le suivant ...
La position de ces éléments est numérotée en partant de 0. La taille de l'objet est donnée par la fonction `len()`.
Exemples d'objets itérables : chaîne de caractères, listes, uplets, intervalles d'entiers.
En particulier, l'argument de la fonction de conversion `list()` doit être une objet itérable.

```
 >>> list('python')
['p', 'y', 't', 'h', 'o', 'n']
 >>> list((1,2,'ert'))
[1, 2, 'ert']
```

III. Les nombres

III.1. Les nombres complexes : complex

Notation	Opération
`complex(a,b)`	`a+b*1j` le 2ème argument est optionnel
`c.real`	Partie réelle de c
`c.imag`	Partie imaginaire de c
`c.conjugate()`	Conjugué de c
`abs(c)`	Module de c

```
>>> a=complex(2,3);a
(2+3j)
>>> complex(2)
(2+0j)
>>> complex(1+1j,1+1j)
2j
>>> a.real;a.imag
2.0
3.0
>>> a.conjugate()
(2-3j)
>>> abs(a)
3.6055512754639896
```

Remarque : Certaines fonctions spécifiquement liées au type d'une variable sont accessibles via un point placé à la suite du nom de la variable.
Ces fonctions sont appelées *méthodes*, ***attributs***.
L'attribut peut nécessiter des arguments ou non.
Par exemple, la partie réelle d'un nombre complexe `c` est obtenue par l'attribut (sans argument) `real` : `c.real`.

III.2. Les entiers : int

Notation	Explication
`int(x)`	Troncature entière du réel x
`a//b`	Quotient de la division euclidienne de a par b
`a%b`	Reste de la division euclidienne de a par b voir aussi `mod(a,b)`

```
>>> int(2.1);int(-2.1)
2
-2
>>> 19%4;19//4;
3
4
```

Attention ! La définition Python de la division euclidienne diffère de celle du cours de mathématiques :
Soit $a, b \in \mathbb{Z}$, il existe un unique coupe $(q, r) \in \mathbb{Z}^2$ tel que
$a = qb + r$ et r compris entre 0 inclus et b exclu

```
>>> 19//(-4);19%(-4)
-5
-1   # reste negatif en python si b<0
```

III.3. Les nombres réels : float

Les fonctions suivantes s'appliquent à un nombre réel ou aux coefficients d'un tableau de nombres réels : (module `numpy`)

Notation	Explication
`floor`	Partie entière
`abs`	Valeur absolue
`sin`	Sinus
`cos`	Cosinus
`exp`	Exponentielle
`log`	Logarithme népérien
`sqrt`	Racine carrée