ECG/ECT

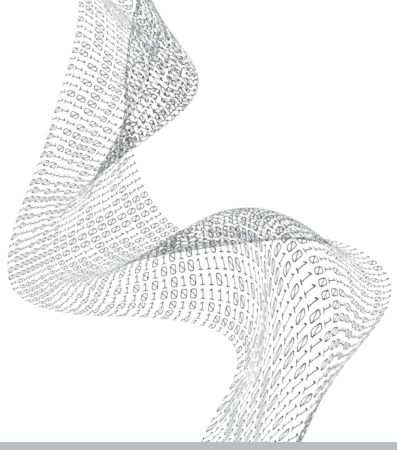
1re et 2e années

INFORMATIQUE

AVEC LE LANGAGE PYTHON

Arnaud Bégyn
François Kany
Florian Marty
Quentin Souillot

ellipses



Chapitre 1

PREMIÈRE ANNÉE, PARTIE 1

1.1 Les objets Python

Lorsqu'on tape une instruction dans la ligne de commande [1], celle ci est exécutée dès que la ligne est validée par la touche entrée $\boxed{\hookleftarrow}$.

La réponse apparaît à la ligne suivante.

La console fonctionne comme l'interface de calcul de votre calculatrice graphique et permet donc de réaliser des calculs simples.
Une instruction validée ne peut plus être modifiée.
Cependant on peut rappeler une instruction dans la ligne de commande en utilisant les touches $\boxed{\uparrow}$ ou $\boxed{\downarrow}$. **Essayez !**

Les nombres

Les nombres réels

Les nombres réels reconnus par Python, sont des nombres appelés *flottants* (de type `float`) composés de 16 *chiffres significatifs* qui ne sont pas tous affichés à l'écran. **Dans la suite**, on écrira en général *valeur de* x au lieu de *valeur approchée* de x à 10^{-16} près.

Un nombre réel se tape comme sur toute calculatrice en mode numérique (on utilise le point-décimal anglo-saxon et pas la virgule) :
`9.44` représente le nombre $9,44$

Dans la suite, comme la virgule sert de séparateur dans les suites de nombres, pour éviter les ambigüités de lecture, nous écrirons partout les nombres "à virgule" avec un point décimal.

Les nombres peuvent être écrits avec des puissances de 10 :

`96.57567e+3` représente 96.57567×10^3 ou encore 96575.67 ; noter qu'il n'y a pas

1. ligne généralement indiquée par "In :" ou ">>>" dans la console selon l'environnement choisi

d'espace entre `7` et `e`.
`3.14e-3` représente 3.14×10^{-3} ou encore 0.00314; noter qu'il n'y a pas d'espace entre `4` et `e`.
En pratique, `e` et `E` sont acceptés.

Pour avoir accès à certains nombres, on pourra avoir besoin de la bibliothèque `numpy`. Pour celà, on importe la bibliothèque c'est à dire un ensemble de variables prédéfinies (fonctions, flottants etc...)

Une possibilité est d'importer avec la commande suivante :

```
import numpy as np
```

puis d'accéder par exemple au nombre `e` avec la commande :

```
np.e
```

On peut obtenir π en changeant `e` par `pi` dans la ligne précédente, obtenir $\sqrt{2}$ en changeant `e` par `sqrt(2)`, etc.

Remarques

- Il est possible d'importer par exemple seulement le nombre π, qui sera utilisable sans préfixe, à l'aide de la commande :

```
from numpy import pi
```

- Il est possible d'importer toute la bibliothèque `numpy`, utilisable sans préfixe, à l'aide de la commande suivante : `from numpy import *`. On évitera en général d'utiliser cette commande en raison d'un risque de conflit entre plusieurs bibliothèques - par exemple le nom de variable `random` est utilisé dans les bibliothèques `numpy` et `numpy.random`.

Remarque
Le codage interne à la machine des nombres réels se fait sur un nombre fixe d'octets [2].
Cette limitation a plusieurs conséquences.

- Il existe un plus grand nombre reconnu par le logiciel. Ce nombre est de l'ordre de 10^{308}, (à comparer par exemple avec la dette privée US qui n'est (été 2013) que de l'ordre de 10^{14}\$).
- Il existe un plus petit nombre strictement positif reconnu par le logiciel. Ce nombre est de l'ordre de 10^{-308}.
- Si a est un nombre réel non nul et que x est un réel tel que $-2.10^{-16} < \dfrac{x}{a} < 2.10^{-16}$ alors, pour Python, le résultat de l'addition $a + x$ est égal à a, ce qui revient à dire que x est négligé devant a.

2. Un octet est une suite de 8 bits (***bi**nary digit*).
Un bit est un des deux symboles 0 ou 1. Il y a donc $2^8 = 256$ octets différents.

Travaux pratiques 1
Tapez dans la ligne de commande :

```
1+10**(-16)-1
```

Que constate-t-on ?

Tapez dans la ligne de commande :

```
1+2.10**(-16)-1
```

Que constate-t-on ?

Les nombres entiers

Un autre type de nombre reconnu par Python est `int` (de l'anglais *integer* pour entier). Ces nombres peuvent être positifs ou négatifs, et ne possèdent pas de virgule.

Ainsi, 37 est de type `int` alors que 37.0 est de type `float` (on pourra le vérifier à l'aide de l'instruction `type(37.0)`).

Remarque. Contrairement au type `float`, il n'y a pas de maximum (ni de minimum) pour le type `int`.

Les opérations sur les nombres

Voici comment faire les calculs classiques à l'aide de Python :

Opérateur	Utilité
+	Addition
−	Soustraction
*	Multiplication
/	Division
//	Quotient dans la division euclidienne
%	Reste dans la division euclidienne
**	Puissance

Elles fonctionnent avec l'ordre de priorité habituel, ce qui nécessite l'usage convenable des **parenthèses**.

Remarques :

$\dfrac{1}{2\times 3}$ se tape `1/(2*3)` ou `1/2/3`

$\dfrac{1}{2}\times 3$ peut se taper `1/2*3`.

Travaux pratiques 2
On effectue les calculs suivants en Python. Donner le type de chaque résultat.

$$2.0\star 4 \quad ; \quad -1.7+0.7 \quad ; \quad 48//10 \quad ; \quad 12.5//12.5$$

Les fonctions de la variable réelle

Pour les fonctions classiques, il convient d'importer ici la bibliothèque `numpy` (voir la partie précédente). De la même façon que pour les nombres, il est aussi possible d'importer une seule fonction, par exemple pour le logarithme :

```
from numpy import log
```

Supposons que la bibliothèque `numpy` ait été importée avec le raccourci `np` (c'est à dire via l'instruction `import numpy as np`). Les fonctions d'une variable réelle au programme sont :

le logarithme népérien : `np.log(10)` calcule $\ln(10)$;

l'exponentielle : `np.exp(-1)` calcule e^{-1} ;

le sinus (♣) : `np.sin(np.pi/4)` calcule $\sin\left(\frac{\pi}{4}\right)$, c'est-à-dire $\frac{\sqrt{2}}{2}$;

le cosinus (♣) : `np.cos(0.01)` calcule $\cos(0.01)$;

la valeur absolue : si x est un nombre réel, `np.abs(x)` donne la valeur absolue de x ; Dans ce cas particulier `abs(x)` fonctionne car cette fonction est aussi définie de base dans Python.

la partie entière : si x est un nombre réel, `np.floor(x)` donne la partie entière de x, c'est-à-dire le plus grand entier inférieur ou égal à x ;

la racine carrée : `np.sqrt(421)` calcule $\sqrt{421}$.

Remarque
En cas de division par zéro, ou si le résultat dépasse le plus grand nombre reconnu par Python, la réponse est un message d'erreur. Selon le cas, le message d'erreur prend une des formes suivantes :
`division by zero`, `math domain error`, etc ...

Vérifiez le en tapant par exemple `np.sqrt(-1)`.

Plus généralement, **nous invitons les étudiants à ne jamais utiliser, sans invitation expresse de l'énoncé, une expression qui n'est pas définie dans leur cours de mathématique :** introduire une expression non définie dans le cours de mathématiques comme `(-1)**.5` même si Python lui donne un sens, ne leur rapporterait que les pires ennuis aux concours.

Les chaînes de caractères

En Python, le type `str` (pour *string*) permet de manipuler des chaînes de caractères. Celles-ci doivent être entrées à l'aide d'apostrophes au début ou à la fin ; les guillemets peuvent aussi être utilisés. Par exemple, on entrera indifféremment

```
"bonjour"  ou  'bonjour'
```

Pour concaténer (juxtaposer) des chaînes de caractères, on utilise `+`, par exemple on peut obtenir `'bonjour'` avec l'instruction :

```
'bon' + 'jour'
```

Il est aussi possible d'utiliser `*` lorsque l'on souhaite répéter la même chaîne de caractères un nombre (entier) de fois. Par exemple, on peut obtenir `'bonbonbon'` avec l'instruction :

```
'bon' * 3
```

Remarque.
Si l'on veut insérer une apostrophe dans une chaîne de caractère, il convient alors d'utiliser les guillemets, par exemple on écrira en Python : `"aujourd'hui"`.

Remarque.
Lorsqu'une chaîne de caractères contient un nombre, il est possible de la "convertir" en un objet de type `int` ou `float` ; par exemple `int('8')` est un objet de type `int`. Ceci est souvent utile lorsque l'on demande à l'utilisateur d'entrer des paramètres pour des programmes ; nous en reparlerons lorsque les variables seront abordées.

Les booléens

En Python, les objets de type `bool`, appelés des booléens, ne peuvent prendre que deux valeurs : `True` et `False`. Par exemple, l'instruction suivante donne `False` :

```
5 < 0
```

Les booléens apparaissent souvent comme des résultats de tests, effectués à l'aide des opérateurs de comparaison. Nous reparlerons de tout ceci plus tard.

Les listes ♠

En Python, une liste (de type `list`) est une série d'éléments, rangés dans un certain ordre, mais pas nécessairement de mêmes types. Une possibilité pour définir une liste est de donner tous ses éléments entre crochets, en les séparant par des virgules. Par exemple, l'instruction suivante définit une liste ayant 4 éléments :

```
['banane',2,True,7.98]
```

Une section entière sera dédiée aux listes.

Les variables

Qu'est qu'une variable ?

Intuitivement, une variable peut être considérée comme une fiche composée de deux parties :

- **son nom**, qui est une suite formée de lettres, de chiffres, et du caractère "souligné", commençant par une lettre.
- **sa valeur** (ou affectation)

Comment créer une variable ?

Pour créer une nouvelle variable numérique, on écrit son nom à gauche de `=`, puis on écrit sa valeur ou le procédé de calcul de sa valeur à droite.

Voici quelques exemples :

```
a = 20.6
b = ['banane',2,True,7.98]
c = -8.5 > -10
```

La première instruction crée la variable `a`, c'est-à-dire la variable dont le nom est `a`, et lui affecte le nombre 20.6 ; cette instruction s'appelle l'**affectation**. Le type de la variable a est `float`.

Attention !
L'opérateur "=" ne doit pas être confondu avec l'égalité mathématique. En mathématiques, le symbole "=" est utilisé pour indiquer que deux objets sont identiques, et cette égalité est symétrique, autrement dit elle se lit dans les deux sens. Ce n'est pas le cas de l'opérateur "=" en Python qui ne se lit que de gauche à droite.

Remarque
Python est sensible à la casse, c'est-à-dire qu'il distingue les majuscules des minuscules.

Remarque
Taper l'instruction `del a` dans la console supprime la variable `a`.

Remarque (hors programme)
Pour demander un nombre à l'utilisateur, il convient d'utiliser la fonction `input`, avec entre parenthèses la chaîne de caratères à afficher à l'écran. Ce qui est entré par l'utilisateur est du type `str` ; une commande qui nous sera souvent utile est donc la suivante :

```
a = float(input("Entrer un nombre"))
```

Cette commande demande un nombre à l'utilisateur, convertit le résultat en un objet de type `float`, et le stocke dans la variable a. On pourra éventuellement remplacer `float` par `int`, si l'on attend que l'utilisateur entre un nombre entier.

Comment modifier la valeur d'une variable ?

Pour modifier la valeur d'une variable numérique qui existe déjà, on réalise une nouvelle affectation. Voici un exemple :

```
a = 20.6
a = 'modif'
# a contient maintenant la valeur 'modif' de type str
```

Remarque.
A chaque affectation, le type d'une variable est celui de la valeur qui lui est affectée. Dans l'exemple ci-dessus, à la fin de la première instruction, la variable `a` est de type `float`, et à la fin de la deuxième instruction, elle est de type `str`.

Calculs avec les variables

Pour utiliser (la valeur d') une variable dans un calcul, on fait comme en mathématiques, on écrit son nom. Considérons l'exemple suivant :

```
a = 2.18
b = a+2
b = b**3
```

La seconde instruction crée la variable `b`, et lui affecte la valeur de `a` augmentée de 2, c'est-à-dire 4.18. La troisième instruction prend la valeur de la variable `b` (ici 4.18), calcule b^3, puis affecte cette valeur à `b`. L'ancienne valeur de `b` est perdue.

Pour les calculs sur une seule variable, on peut raccourcir certaines instructions. Par exemple, on peut utiliser l'opérateur `+=` pour ajouter un nombre à une variable (ou pour concaténer deux chaînes de caractères). Par exemple, à la fin des instructions suivantes, la variable `r` vaut 12 :

```
r = 10
r += 2
```

Avec les variables numériques, on peut également utiliser les opérateurs suivants :

```
-=  ;  *=  ;  /=  ;  //=  ;  %=  ;  **=
```

Il est à noter que l'opérateur `*=` fonctionne aussi avec les chaînes de caractères.

L'affectation parallèle

Il est possible en Python d'effectuer une affection parallèle, c'est à dire d'affecter simultanément des valeurs à des variables. Voici un exemple :

```
a,b,c = -7,['hello',1.4],2*5.3
```

Après cette instruction, la variable `a` contient -7, la variable `b` contient la liste `['hello',1.4]`, et la variable `c` contient 10.6.

Travaux pratiques 3
Quelle est la valeur de `b` après les instructions suivantes ? Commenter.

```
a = 12
b = 4*a
a = 7
```

Corrigés des TP

Corrigé du TP 1
Lorsque l'on tape `1+10**(-16)-1`, la console affiche `0.0`. On constate que 10^{-16} est négligé devant 1.

Lorsque l'on tape `1+2.10**(-16)-1`, on constate que le résultat n'est pas exactement égal à 2.10^{-16}. Ceci est dû aux erreurs d'arrondi dans le calcul.

Corrigé du TP 2
Le type de `2.0*4` est `float` ; la console renvoie `8.0`.
Le type de `-1.7+0.7` est `float` ; la console renvoie `-1.0`.
Le type de `48//10` est `int` ; la console renvoie `4`.
Le type de `12.5//12.5` est `float` ; la console renvoie `1.0`.

Corrigé du TP 3
Après les instructions, la valeur de `b` est 48. On constate que la valeur de `b` n'a pas changé après la nouvelle affectation de `a`.

Exercices

Exercice 1
• Écrire une suite de deux instructions qui affecte à la variable `a` le nombre 10, calcule le nombre $\left(1+\frac{1}{a}\right)^a$ et fait afficher cette valeur.

• En rappelant les instructions précédentes dans la ligne de commande, faire afficher le nombre $\left(1+\frac{1}{100}\right)^{100}$, puis le nombre $\left(1+\frac{1}{1000}\right)^{1000}$.