Université Paul Sabatier Octobre 2014

F.S.I **TP N°1** CB

Suivant votre environnement, les commandes suivantes pourront devoir être adaptées. Nous supposons ici que l'environnement est Linux.

Ouvrir un terminal et tapez : `python`. L'interpréteur affiche un message puis une invite (en anglais un "prompt") : `>>>`.

Il attend qu'on lui soumette des commandes (par exemple : `2+3`). Lorsqu'une commande est validée par l'appui sur entrée alors elle est évaluée et le résultat de cette évaluation est affiché (sauf s'il n'y a pas de résultat - valeur `None`).

Remarquez dès à présent que les commentaires sont introduits par le caractère `#` et s'étendent jusqu'à la fin de la ligne. Pour sortir de l'interpréteur, tapez `quit()`.

```
[christophe:Mountain]> python
Python 2.7.2 (default, Oct 11 2012, 20:14:37)
[GCC 4.2.1 Compatible Apple Clang 4.0 (tags/Apple/clang-418.0.60)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+3
5
>>> 4/3*3.141459*2**3
25.131672
>>> quit()
```

Exercice 1. *Quelques manipulations basiques*

Les nombres entiers

Le type entier, essayez :

```
>>> type(4)
```

Utilisez les quatre opérations de base ('+', '-', '*', '/') sur des entiers relatifs.

Utilisez la variable prédéfinie '_' pour rappeler le dernier résultat.

Essayez et commentez :

```
>>> 20 / 3
>>> 20 // 3
>>> 20 % 3
```

Hiérarchie des opérateurs. Essayez et commentez :

```
>>> 7 + 3 * 4
>>> (7 = 3) * 4
```

Essayez et commentez :

```
>>> 18 @ 3
>>> 15 / 0
```

Essayez de trouver la limite des nombres entiers en faisant :

```
>>> 1 * 1000
>>> _ * 1000 # Répéter cela plusieurs fois.
```

Les flottants Ils sont notés par la présence d'un point décimal «.» ou une notation exponentielle «e» ou «E» dans le nombre :

```
>>> 2.718
>>> 3e8
```

Le type flottant, essayez :

```
>>> type(4.56)
```

Les opérateurs arithmétiques sont les mêmes que ceux déjà vus, essayez-en quelques uns.

Autres opérateurs Expérimentez : que font ces opérateurs ? Fonctionnent-ils avec les autres types déjà vus ?

```
>>> 20.0 // 3
>>> 20.0 % 3
>>> 2.0 ** 10
>>> 5 ** 2 % 3 # Ici, quel est l'opérateur prioritaire ?
>>> 5 % 2 ** 3 # Et là ?
```

Le type complexe Les nombres complexes sont représentés en cartésien par deux flottants, partie réelle et partie imaginaire. L'imaginaire est suffixé par `j` (ou `J`). Testez les expressions suivantes (affichez les valeurs affectées aux variables) :

```
>>> z1 = 1j
>>> type(z1)
>>> z2 = 2 + 1j
>>> z3 = 2 + 1j * 3
>>> z4 = (2 + 1j) * 3
>>> z5 = 2 - 1j
>>> z5.imag
>>> z5.real
>>> z6 = z2 + z5
>>> z7 = z2 + z5
>>> z8 = 1.414 + 1.414j
>>> abs(z8)
```

Le type booléen Il n'accepte que deux valeurs prédéfinies : `True` et `False`. Essai des opérateurs booléens (affichez les valeurs affectées aux variables) :

```
>>> a = True
>>> type(a)
>>> b = not a
>>> a or b
>>> a and b
```

Expressions booléennes Les opérateurs de comparaison ($<$ $<=$ $>$ $>=$ $==$ $!=$), d'appartenance (`in`) et d'identité (`is`) s'évaluent comme expressions booléennes, vraies ou fausses. Essayez :

```
>>> 4 < 12.8
>>> 4 < "4"
>>> 7 <= 9 < 8
>>> v = None
>>> v is None
>>> 7 in [ 1, 3, 5, 7, 9,11 ]
>>> 4 < (6+4j)
```

Expliquer le résultat des différentes expressions boolénnes suivantes

```
C = 41
C == 40
C != 40 and C < 41
C!=40 or C<41
not C == 40
not C > 40
C <= 41
not False
True and False
False or True
False or False or False
True and True and False
False == 0
True == 0
True == 1
```

Les affichages et saisies Toutes les instructions que vous avez expérimentées en mode calculatrice sont valables dans un script (entre autre celles de formatage de chaînes de caractères), sauf l'affichage qui s'indique explicitement par la fonction `print`.

L'instruction `print("a :")` produira à l'écran `a :`.

Il est possible de lui donner plusieurs valeurs, et de différents types. Essayez en mode calculatrice :

```
>>> print("pi =",3.14159)
```

De plus, pour saisir une valeur (opération «inverse» de l'affichage, c'est une affectation programmée), on emploie la fonction `raw_input` :

```
a = raw_input("Donnez votre valeur : ")
```

L'ordinateur affiche alors le message indiqué et attend la saisie d'une valeur au clavier (jusqu'à l'appui sur la touche entrée). Le texte qui a été saisi est retourné par la fonction `raw_input` et affecté à la variable `a`.

Essayez le petit script :

```
n = raw_input("Votre nom : ")
print (n)
```

La fonction `raw_input` ne permettant de saisir que du texte, si l'on veut récupérer des valeurs numériques, il faut demander explicitement une conversion soit en nombre en nombre entier avec `int`, soit en nombre flottant avec `float`. Par exemple :

```
a = int (raw_input ("Votre age : "))
b = float (raw_input ("Moyenne des TPs : "))
```

Exercice 2.

— Afficher le résultat de l'opération `1+1` directement dans l'interpréteur Python

— Faire la même chose en saisissant les commandes dans un fichier que l'on nommera `1plus1.py`. À titre d'exemple,

```
#!/usr/bin/env python
#
# This program says "hello" to the world

print "Hello World!"
```

On peut exécuter ce programme dans le shell soit avec la commande `python nom_fichier.py`, soit, si on a pris soin de rendre le fichier executable (`chmod +x nom_fichier.py`), en tapant directement `nom_fichier.py`.

Exercice 3. Est-ce qu'un nouveau né peut espérer vivre plus de un milliard (10^9) secondes ? Pour cela, créer un programme Python qui convertit les secondes en années.

Exercice 4. Écrire un programme où on donne une longeur en mètre et qui calcule et affiche la longueur correspondante en pouces (inches), en pieds (feet), en yards et en miles britannique. On rappelle qu'un pouce vaut 2.54cm, qu'un pied vaut 12 pouces, qu'un yard vaut 3 pieds et qu'un mile britanique vaut 1760 yards. Nommer le programme `conversion_longueur.py`. À titre de vérification, une longueur de 640 mètres corresponds à 25196.85 pouces, 2099.74 pieds, 699.91 yards ou 0.3977 miles.

Exercice 5. La densité d'une substance est définie comme $\rho = m/V$, où m est la masse de volume V. Calculer et afficher la masse d'un litre de chacune des sustances suivantes dont on donne la densité en g/cm^3.

air	0.0012
gasoil	0.67
glace	0.9
eau douce	1.0
eau de mer	1.025
corps humain	1.03
granite	2.7
fer	7.8
argent	10.5

Exercice 6. Soit p un taux d'intérêt bancaire en pourcent par an. Un montant initial A croit alors de

$$A\left(1+\frac{p}{100}\right)^n$$

après n années. Faire un programme pour calculer combien on rapporter 1000 euros après trois ans si le taux d'intérêt est de 5%. Nommer le programme `taux_interet.py`.

Exercice 7. Supposons que quelqu'un a écrit un programme constitué d'une ligne qui calcule sin(1) :

```
x=1; print 'sin(%g%=%g'% (x, sin(x))
```

Taper le programme et essayer de le lancer. Quel est le problème ?

Exercice 8. *Les modules d'extension*

Jusqu'à maintenant nous n'avons utilisé que des fonctions arithmétiques élémentaires. Bien sûr, Python fournit plus que cela ! Comme vous pouvez le voir dans la documentation intégrée, il existe beaucoup de modules permettant des tâches très diverses.

Par exemple le module mathématique donne accès aux fonctions mathématiques classiques. Regardez sa documentation et essayez :

```
>>> from math import pi, sin # importation depuis un module
>>> pi
>>> a = float(raw_input("angle : ")) # attention à l'unité...
>>> sin(a)
```

Utilisez les différentes fonctions du module `math`. Faites de même avec le module `cmath`.

Taper les trois programmes courts suivant et les exécuter. Quand ils ne fonctionnent pas, identifier et corriger les erreurs.

1. Est-ce que $\sin^2(x) + cos^2(x) = 1$?

```
from math import sin, cos
x = pi/4
1_val = sin^2(x) + cos^2(x)
print 1_VAL
```

2. Travail avec les équations du mouvement avec accéleration constante

```
v0 = 3 m/s
t=1 s
a = 2 m/s**2
s = v0*t + 1/2 a*t**2
print s
```

3. Vérification des équations

$$(a+b)^2 = a^2 + 2ab + b^2$$
$$(a-b)^2 = a^2 - 2ab + b^2$$

Taper programme suivant et corriger si nécessaire.

```
a = 3,3    b=5,3
a2 = a**2
b2 = b**2

eq1_sum = a2 + 2ab + b2
eq2_sum = a2 - 2ab + b2

eq1_pow = (a + b)**2
eq2_pow = (a - b)**2

print 'First equation:  %g = %g', % (eq1_sum, eq1_pow)
print 'Second equation: %h = %h', % (eq2_pow, eq2_pow)
```

Exercice 9. Évaluation d'une fonction gaussienne.

On considère la fonction gaussienne

$$f(x) = \frac{1}{\sqrt{2\pi}s} \exp\left[-\frac{1}{2}\left(\frac{x-m}{s}\right)^2\right].$$

Les paramètres s et m sont des nombres réels, avec s plus grand que zéro. Faire un programme qui évalue cette fonction avec $m = 0$, $s = 2$ et $x = 1$.

Exercice 10. *Calcul de la résistance de l'air sur un ballon*

La force de trainée, due à la resistance de l'air, sur un objet peut être exprimée par

$$F_d = \frac{1}{2} C_D \rho A V^2,$$

où ρ est la densité de l'air, V est la vitesse de l'objet, A est la surface en coupe transversale (normale à la direction de la vitesse), et C_D est le coefficient de trainée, qui dépend fortement de la forme de l'objet et de la rugosité de la surface.

La force de gravité sur l'objet de masse m est $F_g = mg$ où $g = 9.81(m\ s)^{-2}$.

On peut utiliser les formules pour F_d et F_g pour étudier l'importance de la résistance de l'air par rapport à la gravité quand on frappe dans une ballon. La densité de l'air est $\rho = 1.2\text{kg m}^{-3}$. On a $A = \pi a^2$ pour n'importe quelle balle de rayon a. Pour un ballon de football, on prend $a = 11\text{cm}$. La masse d'un ballon de football est de 0.43kg, C_D peut être pris à 0.2.

Faire un programme qui calcule la force de trainée et la force de gravité pour un ballon de football. Ecrire le résultat des forces avec une décimale dans les unités Newtoniennes (N = kg m / s^2). Afficher également le rapport des forces de trainée et de gravité. Définir C_D, ρ, A, V, m, g, F_d et F_g commme des variables, et mettre en commentaires l'unité correspondante. Utiliser le programme pour calculer les forces qui s'exercent sur la balle si on frappe fort (V = 120 km/h) et si on frappe doucement (V = 10 km/h). (Attention, l'unité de vitesse à utiliser est m/s). Nommer le programme `frappe.py`.

Exercice 11. *Comment cuire un oeuf de manière parfaite*

Quand un oeuf cuit, les protéines commencent par se dénaturer et coagulent. Quand la température dépasse un niveau critique, des réactions débutent et s'accelèrent avec l'accroissement de la température. Dans le blanc d'oeuf, les protéines commencent à coaguler à des températures supérieures à 63^oC, tandis que dans le jaune, elles démarrent leur coagulation à partir d'environ 70^oC. Pour un oeuf à la coque, le blanc a besoin d'avoir été cuit assez longtemps pour coaguler à une température supérieure à 63^oC, mais le jaune ne doit pas être exposé à des températures supérieures à 70^oC. Pour un oeuf dur, le centre du jaune peut dépasser les 70^oC.

La formule suivante exprime le temps t qu'il faut (en seconde) pour que le centre du jaune atteigne la température T_y (en degrés Celcius)

$$t = \frac{M^{2/3} c \rho^{1/3}}{K\pi^2(4\pi/3)^{2/3}} \ln\left[0.76\frac{T_o - T_w}{T_y - T_w}\right].$$

Ici, M, ρ, c and K représentent les propriétés de l'oeuf : M est la masse, ρ est la densité, c est la capacité calorifique spécifique, et K est la conductivité thermique. Des valeurs réalistest sont $M = 47$ g pour une petit oeuf et $M = 67$ g pour un gros oeuf, $\rho = 1.038\ \text{g cm}^{-3}$, $c = 3.7\ \text{J g}^{-1}\ \text{K}^{-1}$, et $K = 5.4.10^{-3}\ \text{W cm}^{-1}\ \text{K}^{-1}$. En outre, T_w est la température (en degrés Celcius) de l'eau bouillante, et T_o est la température de départ (en degrés Celcius) de l'oeuf avant d'être plongé dans l'eau.

Implémenter la formule dans un programme, avec $T_w = 100$ C et $T_y = 70$ C, et calculer t pour un gros oeuf sorti du refrégérateur ($T_o = 4$ C) et à température ambiante ($T_o = 20$ C). Nommer le programme `oeuf.py`.

Exercice 12. *Les chaînes de caractères*

Essayer et interpréter

```
>>> 'abc'+'efg'
>>> len('toto')
>>> len('abcdefgh')
>>> 'abcdefgh'[0:2]
>>> 'abcdefgh'[0:7:2]
>>> 'abcdefgh'[-1]
>>> str.capitalize('toto')
>>> str.upper('abGc')
>>> dir(str)
>>> print "this   is\na\ttest"
```

```
>>> str.split("this   is\na\ttest")  # split without any arguments splits on
                                     # whitespace. So three spaces, a carriage
                                     # return, and a tab character are all the same.
['this', 'is', 'a', 'test']
>>> print " ".join(str.split("this   is\na\ttest"))
this is a test
```

Exercice 13. *Trouver des erreurs dans des expressions Python*

Essayer les expressions suivantes dans une session interactive Python. Expliquer pourquoi certaines d'entres elles sont incorrectes et corriger.

```
1a = 2
a1 = b
x=2
y=X+4 #isit6? from Math import tan print tan(pi)
pi = "3.14159'
print tan(pi)
c = 4**3**2**3
_ = ((c-78564)/c + 32))
discount = 12%
AMOUNT = 120.-
amount = 120$
address = hpl@simula.no
and = duck
class = 'INF1100, gr 2"
continue_ = x > 0
rev = fox = True
Norwegian = ['a human language']
true = fox is rev in Norwegian
```

Exercice 14. *Les fonctions en ligne*

L'expression `lambda x1,....,xn: ....` définit une fonction anonyme prenant n paramètres. Elle peut-être appliquée à n arguments.

```
>>> (lambda x,y: x+y) (2,3)
5
>>> (lambda f,x: f(x+2)) (lambda x:2*x, 5)
14
```

Nous donnons ici trois fonctions de Python qui peuvent intéragir avec des fonctions.

Map

`map(function, sequence)` appelle la `function(item)` pour chacun des membres de `sequence` et renvoie une liste des valeurs de retours. Par exemple, pour calculer des cubes

```
>>> map(lambda x: x*x*x, range(1, 11))
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
>>>
```

Filter

`filter(function, sequence)` renvoie une sequence (du même type, si possible) consistant en les éléments de la sequence pour lesquels `function(item)` est vrai. Par exemple, pour calculer quelques nombres premiers

```
>>> filter(lambda x: x%2 != 0 and x%3 != 0, range(2, 25))
[5, 7, 11, 13, 17, 19, 23]
>>>
```

Reduce `reduce(function, iterable[, initializer])`

Applique `function`, fonction qui doit prendre deux arguments en paramètre, cumulativement aux éléments de =verb+iterable+, de gauche à droite, de manière à réduire l'iterable à une seule valeur. Par exemple,

```
>>> reduce(lambda x, y: x + y, [1, 2, 3, 4, 5])
15
```

L'argument de gauche, x est la valeur accumulée et l'argument de droite, y est la valeur mise à jour de l'itérable. Le comportement peut être assimilé à `((((1 2) 3)4) 5)`. La fonction décrite ci-dessus fait la somme des éléments de la liste.

Si l'argument optionnel est présent, il est placé avant les éléments de l'iterable dans le calcul, et sert de défaut lorsque l'iterable est vide. Si `initializer` n'est pas donné et `iterable` ne contient qu'un seul élément, le premier élément est retourné.

Créer la fonction `g` telle que

$$g : \begin{array}{l} \mathbb{R} \to \mathbb{R} \\ x \mapsto x^2 \end{array} .$$

De même, créer la fonction `h` telle que

$$h : \begin{array}{l} \mathbb{C} \to \mathbb{C} \\ z \mapsto z^2 + 4i \end{array} .$$

Soit la liste `foo = [2, 18, 9, 22, 17, 24, 8, 12, 27]`. En utilisant les fonctions `filter`, `map` et `reduce`, pour renvoyer les listes contenant

- les valeurs de `foo` telles que le reste de la division entière par 3 soit nul,
- $x \in$ foo, alors $2x + 10$,
- la somme des éléments de `foo`.

Que fait la fonction suivante

```
>>> nums = range(2, 50)
>>> for i in range(2, 8):
...     nums = filter(lambda x: x == i or x % i, nums)
...
>>> print nums
```

Que fait la fonction suivante

```
>>> sentence = 'It is raining cats and dogs'
>>> words = sentence.split()
>>> print words
['It', 'is', 'raining', 'cats', 'and', 'dogs']
>>>
>>> lengths = map(lambda word: len(word), words)
>>> print lengths
```

Ecrire ces suites d'instructions en une seule instruction.

Que fait la commande suivante

```
reduce(lambda x,y: x+y, [x for x in range(1,1000) if x % 3 == 0 or x % 5 == 0])
```

Interpréter

```
print reduce(lambda x,  y: x*y,  [2, 3, 4])
```

Quel est le résultat de

```
print reduce(lambda x,  y: x*int(y),  ['2',  '3',  '4'])
```

Pourquoi les résultats sont-ils différents ? Comment pourrait-on corriger la commande précédente pour trouver le résultat attendu ?

Créer une fonction qui transforme une liste de températures saisies en degrés celcius en degré Fahrenheit. La fonction mathématique qui réalise la transformation est $t \mapsto \frac{9}{5}t + 32$. Créer la fonction inverse et vérifier vos résultats.

Interpréter la commande

```
reduce(set.intersection,map(set,[[1,2,3,4,5],[2,3,4,5,6],[3,4,5,6,7]]))
```

Exercice 15.

Considérer les deux assignements de fonctions.

```
f = lambda x,g: x ** g / g
g = lambda f: f % 3
```

Sans exécuter le code, quel sera le résultat de `g(f(g(2),4))` ?

1. 0.25
2. 0.0 (float)
3. 0 (int)
4. 1

Considérer le code suivant

```
from math import sin,pi
h = lambda x,y,z: x**y*sin(z)
print(map(lambda x: h(x,2,pi/x), range(2,5)))
```

Quel sera le premier élément de l'affichage ?

1. 7.794228634059947
2. 0.0
3. pi
4. 4.0