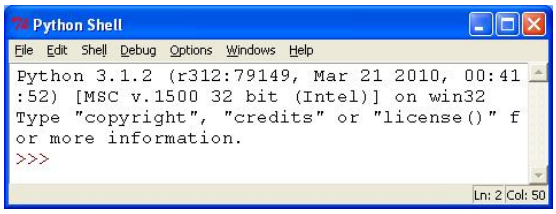
Python : les bases (corrigé)

1 Utilisation de Python

Créez un répertoire Python dans votre espace de travail. Lancez Python (sous Windows, menu Programmes/IDLE (Python GUI)

Une fenêtre s'ouvre :



La fenêtre Python Shell permet d'entrer des calculs ligne par ligne.

Une invite (*prompt* en anglais) formée de trois chevrons >>> autorise à entrer des expressions Python. Entrez par exemple, après l'invite, la ligne suivante :

```
>>> (2011*2012)//2 #la somme des entiers de 1 a 2011
2023066
```

La fin de ligne à partir du # est un commentaire, ignoré par Python. L'opérateur // est celui de la division euclidienne.

Dans cette fenêtre, seule la dernière ligne est éditable. On peut toutefois récupérer et modifier les lignes déjà entrées par les raccourcis <Alt>P (précédent) et <Alt>N (suivant) permettant de naviguer dans l'historique des commandes.

De plus, un nom spécial, `_` (tiret de souignement, *underscore*), permet de récupérer le résultat de la dernière commande.

Une aide très complète est fournie avec Python en passant par le menu Help/Python Docs, raccourci F1. En particulier, The Python Tutorial est une lecture indispensable.

2 Python, une calculatrice simple

Python permet de calculer avec trois sortes de nombres :

- les entiers `int`, comme `2`, `-723` ou `1234567890123` ;
- les décimaux `float`, comme `3.14159`, `2.011e3` ou `-3.5e-7` ;
- les complexes `complex`, comme `1.0+2.3j`.

Dans The Python Tutorial, rendez-vous au paragraphe

- 3. An Informal Introduction to Python
  - 3.1. Using Python as a Calculator
    - 3.1.1. Numbers

et consultez-le pour les exercices suivants.

Exercice 1.

à rendre **Écrivez en Python les nombres suivants :**

$1\,234$ $\quad -123\,456\,789$ $\quad 6,02 \times 10^{23}$
$-1,602\,176\,53 \times 10^{-19}$ $\quad 2+3\,\mathrm{i}$ $\quad 2-\mathrm{i}$

Corrigé

```
**************************************************************************
>>> 1234
1234
>>> -123456789
-123456789
>>> 6.02e23
6.02e+23
>>> -1.60217653e-23
-1.60217653e-23
>>> 2+3j
(2+3j)
>>> 2-1j
(2-1j)
**************************************************************************
```

Exercice 2.

à rendre **Quels sont les résultats (valeur et type) des expressions suivantes ?**

```
23+8                    23-8                    23*8
23/8                    23//8                   23%8
23.3+8.5                23.3-8.5                23.3*8.5
23.3/8.5                23.3//8.5               23.3%8.5
(3+2j)/(2-1j)           (2+4j)/(1+2j)           (1j)**2
(2.1+3j).real           (2.1+3j).imag           (-1)**(1/2)
```

Note : L'appel à la fonction `type(_)` retourne le type du dernier résultat.

Corrigé

```
**************************************************************************
23+8 31 (int)                   23-8 15 (int)
23*8 184 (int)                  23/8 2.875 (float)
23//8 2 (int)                   23%8 7 (int)

23.3+8.5 31.8 (float)           23.3-8.5 14.8 (float)
23.3*8.5 198.05 (float)         23.3/8.5 2.74117647059 (float)
23.3//8.5 2.0 (float)           23.3%8.5 6.3 (float)

(3+2j)/(2-1j) (0.8+1.4j) (complex)
(2+4j)/(1+2j) (2+0j) (complex)
(1j)**2 (-1+0j) (complex)
(2.1+3j).real 2.1 (float)
(2.1+3j).imag 3.0 (float)
(-1)**(1/2) (6.123233995736766e-17+1j) (complex)


23+8 ⇒ 31 (int)                   23-8 ⇒ 15 (int)
23*8 ⇒ 184 (int)                  23/8 ⇒ 2.875 (float)
23//8 ⇒ 2 (int)                   23%8 ⇒ 7 (int)
23.3+8.5 ⇒ 31.8 (float)           23.3-8.5 ⇒ 14.8 (float)
23.3*8.5 ⇒ 198.05 (float)         23.3/8.5 ⇒ 2.7411764705882353 (float)
23.3//8.5 ⇒ 2.0 (float)           23.3%8.5 ⇒ 6.300000000000001 (float)
(3+2j)/(2-1j) ⇒ (0.8+1.4j) (complex)
```

```
(2+4j)/(1+2j) ⇒ (2+0j) (complex)
(1j)**2 ⇒ (-1+0j) (complex)
(2.1+3j).real ⇒ 2.1 (float)
(2.1+3j).imag ⇒ 3.0 (float)
(-1)**(1/2) ⇒ (6.123233995736766e-17+1j) (complex)
```

****

Exercice 3.

Les calculs sur les nombres décimaux sont arrondis et l'ordre des calculs a un effet :

à rendre **Que valent les expression suivantes ?**

$0,3+0,2+0,1$ $\quad 0,1+0,2+0,3$ $\quad 0,3\times 0,2\times 0,1$

$0,1\times 0,2\times 0,3$ $\quad (1,2+1,3)^2$ $\quad 1,2^2+2\times 1,2\times 1,3+1,3^2$

$$16+19\,\frac{3}{\dfrac{1}{1+\dfrac{1}{1+\dfrac{1}{2}}}-\dfrac{1}{1+\dfrac{1}{1+\dfrac{1}{3}}}}$$

Corrigé

****

```
>>> 0.3+0.2+0.1
0.6
>>> 0.1+0.2+0.3
0.6000000000000001
>>> 0.3*0.2*0.1
0.006
>>> 0.1*0.2*0.3
0.006000000000000001
>>> (1.2+1.3)**2
6.25
>>> 1.2**2 + 2*1.2*1.3 + 1.3**2
6.250000000000001
>>> 16 + 19 * 3 / (1/(1+1/(1+1/2)) - 1/(1+1/(1+1/3)))
2010.9999999999916 # ça devrait être 2011
```

****

Exercice 4. Le *Talkhys*

Le *Talkhys* est un traité d'arithmétique pratique d'Ibn Albanna, mathématicien et astronome marocain de la première moitié du XIII$^{\text{e}}$ siècle.

à rendre **Vérifiez quelques unes des égalités suivantes extraites du *Talkhys*.**

1	×	1	=	1
11	×	11	=	121
111	×	111	=	12321
1111	×	1111	=	1234321
11111	×	11111	=	123454321
111111	×	111111	=	12345654321
1111111	×	1111111	=	1234567654321

9	×	1	+	2	=	11
9	×	12	+	3	=	111
9	×	123	+	4	=	1111
9	×	1234	+	5	=	11111
9	×	12345	+	6	=	111111
9	×	123456	+	7	=	1111111
9	×	1234567	+	8	=	11111111
9	×	12345678	+	9	=	111111111
9	×	123456789	+	10	=	1111111111

9	×	9	+	7	=	88
9	×	98	+	6	=	888
9	×	987	+	5	=	8888
9	×	9876	+	4	=	88888
9	×	98765	+	3	=	888888
9	×	987654	+	2	=	8888888
9	×	9876543	+	1	=	88888888
9	×	98765432	+	0	=	888888888

8	×	1	+	1	=	9
8	×	12	+	2	=	98
8	×	123	+	3	=	987
8	×	1234	+	4	=	9876
8	×	12345	+	5	=	98765
8	×	123456	+	6	=	987654
8	×	1234567	+	7	=	9876543
8	×	12345678	+	8	=	98765432
8	×	123456789	+	9	=	987654321

Corrigé

****

Par exemple :

```
>>> 8 * 1234567 + 7
9876543
```

****

3 Python, une calculatrice à mémoire de données

Lorsque les enchaînements de calcul sont trop nombreux ou trop longs, il est possible de mémoriser des données, en utilisant des noms prermettant de s'y référer.

Il faut utiliser des noms parlants, pemettant au lecteur de se passer de commentaires. Par exemple, si on entre :

```
>>> rayon = 3.0
```

puis, les lignes suivantes :

```
>>> périmètre = 2 * 3.1416 * rayon
>>> aire = 3.1416 * rayon**2
>>> print("Le périmètre d'un cercle de rayon",rayon,"vaut",périmètre,"et ./.
./.son aire",aire)
```

La lecture des instructions est beaucoup plus facile que s'il on avait utilisé des noms comme `r, p, s` ou pire `a, b, c`. L'affichage est alors

```
Le périmètre d'un cercle de rayon 3.0 vaut 18.8496 et son aire 28.2744
```

Exercice 5. Les grains de blé sur l'échiquier

L'écrivain arabe Asaphad *rapporte, en effet, que* Sessa, *fils de* Daher, *imagina le jeu des échecs, où le roi, quoique la pièce la plus importante, ne peut faire un pas sans le secours de ses sujets les pions, dans le but de rappeler au monarque indien* Scheran *les principes de justice et d'équité avec lesquels il devait gouverner.* Scheran, *enchanté d'une leçon donnée d'une manière si ingénieuse, promit à*

> *l'inventeur de lui donner tout ce qu'il voudrait pour sa récompense. Celui-ci répondit : « Que Votre Majesté daigne me donner un grain de blé pour la première case de l'échiquier, deux pour la seconde, quatre pour la troisième, et ainsi de suite, en doublant jusqu'à la soixante-quatrième case. »*
>
> Édouard LUCAS, *L'arithmétique amusante*, Blanchard 1974

On estime qu'un grain de blé pèse 0,035 g et que la production annuelle mondiale de blé est de 600 millions de tonnes.

à rendre **Écrivez la suite d'instructions permettant de calculer le nombre d'années de production qu'il faudrait pour garnir l'échiquier ?**

Note : La solution :

```
>>> int((2**64-1)*0.035/1000000/600000000)
1076
```

donne le résultat, mais n'est pas assez claire. Utilisez des variables avec des noms parlants.

Corrigé

****

```
>>> #Les grains de blé sur l'échiquier
>>> nbTotalDeGrains=2**64-1 # 2^0+2^1+2^2+...2^(n-1)=2^n-1
>>> nbGrammesParTonne=10**6 #g/t
>>> nbTonnesParAn=600*10**6 #t/an
>>> masseUnGrain=0.035 #g
>>> nbAnnées = nbTotalDeGrains * masseUnGrain / nbGrammesParTonne / ./.
./. nbTonnesParAn
>>> print(int(nbAnnées)) #troncature entiere du resultat
1076
```

****

4 Python, une calculatrice programmable

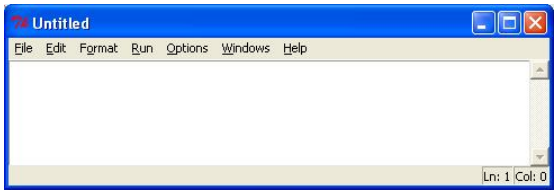
Jusqu'à présent, les instructions Python s'exécutent en séquence (l'une après l'autre). Python permet de contrôler le déroulement des instructions, par exemple en les répétant automatiquement un certain nombre de fois (`for i in range(10) :` , `while (n>0) :`) ou en ne les exécutant qu'à certaines conditions (`if .. elif .. else`).

Ces instructions se terminent par : (*deux points*) et les instructions répétées ou conditionnées sont *indentées* du même nombre d'espaces.

Pour pouvoir saisir un programme (plusieurs lignes devant s'exécuter en séquence) et le sauvegarder, on dispose d'un éditeur de texte écrit lui même en Python : Idle.

Dans la fenêtre Python Shell, choissez le File/New window.

La fenêtre suivante aparaît :



Cette fenêtre, titrée pour l'instant Untitled, va nous permettre de saisir notre premier programme. Entrez les lignes suivantes.

```
# fichier : devine.py
# auteur : Jean-Claude GEORGES
# date : 19/05/2010
# màj  : 16/10/2012
# exemple simple de programme python : le nombre à deviner

import random # pour des fonctions de génération de nombres aléatoires

MINI = 1
MAXI = 999

print ( """Voici le classique jeu du nombre à deviner (niveau Adibou CE1).
L'ordinateur va choisir un nombre au pseudo-hasard entre """
→→+ str( MINI ) + ' et ' + str ( MAXI ) +
"\net vous allez essayer de le deviner.\n" )
#initialisation du programme
nb_à_deviner = random.randrange(MINI , MAXI +1) # nombre à deviner

## *************** testeurs ****************************
## décommentez la ligne suivante pour "tester" en un coup
# print(nb_à_deviner)
## ******************************************************

nb_essais = 0 # nombre d'essais
trouvé = False # type booleen
message = 'Proposez un nombre : '
#corps du programme
while not trouvé : # tant qu'on n'a pas trouve
→x = int (input(message)) # affichage message, puis demande entrée
                          #  clavier et conversion en int
→## *************** testeurs + cheat code ***************
→## décommentez les trois ligne suivantes si vous voulez tester
→## que tentative reste au singulier dans le dernier message
→## en cas de découverte du nombre en un coup.
→## ******************************************************
```

```
#⟶if x == 2011 :
#⟶⟶⟶print(nb_à_deviner)
#⟶⟶⟶continue
⟶⟶nb_essais += 1
⟶⟶if x < nb_à_deviner :
⟶⟶⟶⟶print ("C'est plus !")
⟶⟶elif nb_à_deviner < x :
⟶⟶⟶⟶print ("C'est moins !")
⟶⟶else : # x == nb_à_deviner
⟶⟶⟶⟶trouvé = True
⟶⟶⟶⟶#terminaison du programme
⟶⟶⟶⟶print ("\nBravo. Vous avez deviné le nombre en ",nb_essais,
      " tentative", ('s' if nb_essais > 1 else''),' !', sep = '')
                    # on ajoute un s au pluriel
```

Sauvegardez le programme avec le menu File/Save, raccourci <Ctrl>S (devine.py par exemple, le suffixe .py étant utilisé par Idle pour mettre en couleur les différents éléments du programme).

La fenêtre Idle se renomme alors devine.py. Pour recharger un programme sauvegardé, c'est le menu File/Open... , raccourci <Ctrl>O qu'il faut utiliser.

Exécutez le programme (menu Run/Run Module , raccourci <F5>). S'il y a des erreurs, corrigez le programme, sinon le programme s'exécute dans la fenêtre Python Shell.

Dans The Python Tutorial, rendez-vous aux paragraphes

- 4. More Control Flow Tools
  - 4.1. if Statements
  - 4.2. for Statements
  - 4.3. The range() Function
  - 4.4. break and continue Statements, and else Clauses on Loops

et consultez-les pour les exercices suivants.

Exercice 6. La suite de Fibonacci

> *Supposez qu'on place un couple de lapins adultes dans un enclos. Supposez que deux lapereaux — un mâle et une femelle — naissent deux mois après la naissance des parents, puis à nouveau que deux lapereaux naissent des mêmes parents un mois plus tard et ainsi de suite : chaque mois, chaque couple, à partir de l'âge de deux mois, engendre deux lapins. Si aucun lapin ne meurt, combien y aura-t-il de couples à la fin de l'année dans l'enclos ?*
>
> FIBONACCI, *Liber Abaci*, v. 1202

Soit la suite définie par :

$$\left|\begin{array}{lcll} u_0 & = & 0 & \\ u_1 & = & 1 & \\ u_n & = & u_{n-1}+u_{n-2} & \text{pour } n \geq 2 \end{array}\right.$$

à rendre **Que vaut u_{100} ?**

Corrigé

****

```
>>> avant_dernier, dernier = 0, 1
>>> for i in range(2,101):
⟶avant_dernier, dernier = dernier, avant_dernier + dernier
>>> print('Fib(',i,") = ",dernier,sep='')
Fib(100) = 354224848179261915075
```

****

5 Python, un langage de programmation

Pour que des traitements puissent être réutilisés, Python permet de leur attribuer des noms.

Dans une nouvelle fenêtre IDLE (File/New window), entrez les lignes suivantes :

```
# exemple de programme definissant des fonctions

import math

# périmètre d'un cercle
def périmètre (r) :
⟶""" calcule et retourne le périmètre d'un cercle de rayon r
La formule utilisee est p = 2 Pi r"""
⟶return 2 * math.pi *r

# aire d'un cercle
def aire (r) :
⟶""" retourne l'aire d'un cercle de rayon r
La formule utilisee est a = Pi r*r"""
⟶return math.pi * r**2
```

Sauvegardez sous le nom `cercle.py`, puis exécutez le programme . S'il n'y a pas d'erreurs, dans la fenêtre Python Shell, vous pouvez maintenant utiliser vos deux fonctions `périmètre` et `aire` :

```
>>> périmètre(0.5)
3.14159265358979
>>> aire(2.0)
12.566370614359172
```

De plus, la première chaîne de caractères entre `"""triples double quotes"""` est interprétée par Python comme une *docstring* (chaîne de documentation), et peut être affichée de manière naturelle par la commande :

```
>>> help(aire)
Help on function aire in module __main__:

aire(r)
    retourne l'aire d'un cercle de rayon r
    La formule utilisee est a = Pi r*r
```

Exercice 7. Retour sur le *Talkhys*

Comment écrire une fonction qui affiche le premier tableau du *Talkhys* ?

$$
\begin{array}{rcrcr}
1 & \times & 1 & = & 1 \\
11 & \times & 11 & = & 121 \\
111 & \times & 111 & = & 12321 \\
1111 & \times & 1111 & = & 1234321 \\
11111 & \times & 11111 & = & 123454321 \\
111111 & \times & 111111 & = & 12345654321 \\
1111111 & \times & 1111111 & = & 1234567654321
\end{array}
$$

Il faut toujours commencer par le cœur du programme.

- programmer l'affichage d'une ligne

```
>>> a= 11
>>> print(a,'*',a,'=',a*a)
11 * 11 = 121
```

- puis regarder comment on passe d'une ligne à l'autre. Ici, on peut remarquer que le premier nombre de chaque ligne est égal à 10 fois le nombre de la ligne précédente plus 1.

```
>>> a = 10*a + 1
>>> print(a,'*',a,'=',a*a)
111 * 111 = 12321
```

on dirait que ça marche !

- mettre tout ça dans une boucle et initialiser a :

```
>>> a = 1
>>> for i in range(5) :
⟶print(a,'*',a,'=',a*a)
⟶a = 10*a + 1

1 * 1 = 1
11 * 11 = 121
111 * 111 = 12321
1111 * 1111 = 1234321
11111 * 11111 = 123454321
```

- mettre tout ça dans une fonction, sans oublier la *docstring* :

```
def Talkhys1(n) :
⟶""" affiche les n premieres lignes du tableau :
1 * 1 = 1
11 * 11 = 121
111 * 111 = 12321
1111 * 1111 = 1234321
11111 * 11111 = 123454321    """
⟶a = 1
⟶for i in range(n) :
⟶⟶print(a,'*',a,'=',a*a)
⟶⟶a = 10*a + 1
```

- et enfin l'exécution :

```
>>> Talkhys1(9)
1 * 1 = 1
11 * 11 = 121
111 * 111 = 12321
1111 * 1111 = 1234321
11111 * 11111 = 123454321
111111 * 111111 = 12345654321
1111111 * 1111111 = 1234567654321
11111111 * 11111111 = 123456787654321
111111111 * 111111111 = 12345678987654321
```

à rendre **Écrire les trois fonctions affichant les n premières lignes de chacun des trois autres tableaux du *Talkhys* (page 4 de l'énoncé)**

Corrigé

```
**********************************************************************
# Talkhys : exemples de boucles
```

```
def Talkhys1(nb_lignes) : #autre sol que l'énoncé
    """ affiche les n premieres lignes du tableau :
1 * 1 = 1
11 * 11 = 121
111 * 111 = 12321
1111 * 1111 = 1234321
11111 * 11111 = 123454321    """
    n=0
    for i in range(nb_lignes) :
        n = n*10 +1
        print(n,"*",n,"=",n*n)

def Talkhys2(nb_lignes) :
    """ affiche les n premieres lignes du tableau :
9 * 1 + 2 = 11
9 * 12 + 3 = 111
9 * 123 + 4 = 1111
9 * 1234 + 5 = 11111
9 * 12345 + 6 = 111111"""
    a=9
    b=1
    c=2
    for i in range(nb_lignes) :
        print(a,"*",b,"+",c,"=",a*b+c)
        b=10*b+c
        c+=1

def Talkhys3(nb_lignes) :
    """ affiche les n premieres lignes du tableau :
9 * 9 + 7 = 88
9 * 98 + 6 = 888
9 * 987 + 5 = 8888
9 * 9876 + 4 = 88888
9 * 98765 + 3 = 888888"""
    a=9
    b=9
    c=nb_lignes-1
    while c>=0 :
        print(a,"*",b,"+",c,"=",a*b+c)
        b=10*b+c+1
        c-=1

def Talkhys4(nb_lignes) :
    """ affiche les n premieres lignes du tableau :
8 * 1 + 1 = 9
8 * 12 + 2 = 98
8 * 123 + 3 = 987
8 * 1234 + 4 = 9876
8 * 12345 + 5 = 98765"""
    a=8
    b=1
    c=1
    for i in range(nb_lignes) :
        print(a,"*",b,"+",c,"=",a*b+c)
        c+=1
        b=10*b+c
```

En exécutant :

```
>>> Talkhys1(6)
1 * 1 = 1
```

```
11 * 11 = 121
111 * 111 = 12321
1111 * 1111 = 1234321
11111 * 11111 = 123454321
111111 * 111111 = 12345654321
>>> Talkhys2(6)
9 * 1 + 2 = 11
9 * 12 + 3 = 111
9 * 123 + 4 = 1111
9 * 1234 + 5 = 11111
9 * 12345 + 6 = 111111
9 * 123456 + 7 = 1111111
>>> Talkhys3(6)
9 * 9 + 5 = 86
9 * 96 + 4 = 868
9 * 965 + 3 = 8688
9 * 9654 + 2 = 86888
9 * 96543 + 1 = 868888
9 * 965432 + 0 = 8688888
>>> Talkhys4(6)
8 * 1 + 1 = 9
8 * 12 + 2 = 98
8 * 123 + 3 = 987
8 * 1234 + 4 = 9876
8 * 12345 + 5 = 98765
8 * 123456 + 6 = 987654
```

****