TP Python 1 - Notions de base

Edupython

Edupython est un **environnement de développement intégré** (de l'anglais *integrated development environment*, en abrégé **IDE**), c'est-à-dire un logiciel qui permet entre autres d'écrire des programmes dans un éditeur de texte et de les exécuter.

L'interface d'Edupython est essentiellement divisée en deux parties :

– La console, en bas avec les chevrons (`>>>`), où on peut taper des instructions qui sont exécutées immédiatement.

– La zone de script, en haut, dans laquelle on tape des programmes qu'on peut exécuter en cliquant sur la flèche verte (raccourci Ctrl+F9). On peut n'exécuter qu'une partie du code en la sélectionnant et en faisant Ctrl+F7.

Dans la suite du TP les instructions précédées par des chevrons sont à tester dans la console, les autres dans la zone de script.

Il est recommandé d'enregistrer régulièrement son travail (raccourci Ctrl+S).

Hello world

La première fois qu'on utilise un langage de programmation, il est de tradition de commencer par faire afficher Hello world ! à l'ordinateur.

```
>>> print("Hello world!")
Hello world!
```

Exercice 1 Taper dans la zone de script un programme permettant d'afficher `Hello world!`, l'enregistrer dans votre espace personnel et l'exécuter.

Opérations

Opérations élémentaires : `+`, `-`, `*`, `/` (division réelle), `//` (division euclidienne), `%` (modulo), `**` (puissances, entières ou non).

```
>>> 7 / 3
2.3333333333333335 # nombre à virgule flottante (flottant)
>>> 7 // 3
2                  # quotient dans la division euclidienne 7 = 3 * 2 + 1
>>> 7 % 3
1                  # reste dans la division euclidienne 7 = 3 * 2 + 1
```

Tout ce qui suit `#` est un commentaire qui n'est pas pris en compte par l'interpréteur.

Priorité des opérations :

1. Parenthèses
2. Exposants
3. Multiplications, divisions, modulo
4. Addition, soustraction

```
>>> 1 + 2 * 3
7
>>> (1 + 2) * 3
9
>>> 3 + 2.5 * 2  # il y a un flottant dans le calcul :
8.0              # le résultat est un flottant
```

Certaines fonctions ne sont pas accessibles directement, il faut aller les chercher dans un module extérieur. Par exemple, la fonction racine carrée `sqrt` est dans le module `math` :

```
>>> from math import sqrt
>>> sqrt(2)         # on peut aussi écrire 2 ** 0.5
1.4142135623730951
```

Exercice 2 Calculer $\cfrac{1}{1+\cfrac{1}{2+\cfrac{1}{3+\cfrac{1}{4}}}}$. On trouvera `0.6976744186046512`.

Affectation

Noms de variables : séquences de lettres (minuscules ou majuscules) ou de chiffres (le premier caractère doit être une lettre). Le underscore _ est également autorisé. Certains noms sont réservés (`and`, `or`, `if`, `for`...). Python distingue les minuscules et les majuscules.

L'affectation se fait avec `=`.

```
>>> a = 1   # a est le nom de la variable, 1 est sa valeur
>>> a
1
>>> 3 = a
SyntaxError: can't assign to literal
>>> b = a
>>> b
1
```

Affectation multiple :

```
>>> a = b = 5
>>> x, y = 2, 3
>>> print(x, y, a, b) # la fonction print sert à afficher un résultat
2 3 5 5
>>> y, x = x, y       # échange des valeurs de x et y
>>> print(x, y)
3 2
```

Il est recommandé de bien choisir les noms des variables. Cela améliore grandement la lisibilité des programmes.

Exercice 3 Que font les programmes suivants (chercher sans les taper avant de vérifier) ?

1)
```
a = 1
b = 2
b = a
print(a, b)
```

2)
```
a = 1
b = 2
a = a + b
b = b + a
print(a, b)
```

3)
```
a = 1
b = 2
c = a
b = c
a = b
print(a, b)
```

4)
```
a = 1
b = 2
c = a
a = b
b = c
print(a, b)
```

Tests

Les booléens de Python sont `True` et `False` (attention aux majuscules). Les opérateurs de comparaison élémentaires sont `==` (égalité), `<`, `>`, `<=`, `>=` et `!=` (différent de). Les opérateurs logiques sont `and`, `or` et `not`.

```
>>> 1 + 1 == 2
True
>>> not(1 + 1 == 2)
False
>>> 1 + 1 == 3 or 2 * 2 <= 4
True
>>> 1 + 1 == 3 and 2 * 2 <= 4
False
```

Attention à ne pas confondre `=` (affectation) et `==` (test d'égalité).

Pour exécuter des instructions si une certaine condition est vérifiée, on utilise `if`. La syntaxe est :

```
if conditions:
    instructions
```

Noter les `:` marquant le début du bloc d'instructions à exécuter et **l'indentation** (décalage) avant les instructions. En Python, le début et la fin des tests, des boucles et des fonctions est marqué par l'indentation des instructions.

Les programmes suivants sont à tester dans la zone de script. Noter que lorsqu'on appuie sur Entrée après un deux-points, Edupython crée automatiquement une indentation à la ligne suivante.

```
x = 1
if x == 1:            # ne pas oublier les :
    print('Bonjour')
```

Dans l'exemple suivant le bloc d'instructions à exécuter comporte deux lignes :

```
x = 1
if x == 1:
    print('Bonjour')
    print('Au revoir')
```

Il faut donc faire très attention aux indentations. Essayer de prévoir ce qu'affiche le programme suivant avant de l'exécuter :

```
x = 1
if x == 2:
    print('Bonjour')
    print('Au revoir')
if x == 2:
    print('Bonjour')
print('Au revoir')
```

Pour distinguer plusieurs cas, on peut utiliser `else` (sinon) et `elif` (sinon si). Tester le programme suivant avec différentes valeurs de `x` :

```
x = 2
if x == 1:
    print('x vaut 1')
elif x > 1:
    print('x est strictement plus grand que 1')
else:
    print('x est strictement plus petit que 1')
```

Exercice 4 Qu'affiche le programme suivant ? Et pour `x = 1`, pour `x = -1`, pour `x = 0.5` ?

```
x = 0
y = 1 - x**2
if x > 0 and y > 0:
    print('A')
elif x > 0 or y > 0:
    print('B')
else:
    print('C')
print('D')
```

Boucles

Il y a deux types de boucles en Python : `for` (pour) et `while` (tant que).

Boucle `for` : si `a` et `b` sont deux entiers,

```
for i in range(a, b):
```

permet de faire varier un entier `i` entre `a` et `b-1` (si `a` n'est pas précisé, on commence à 0). Ne pas oublier le deux-points.

Les programmes suivants peuvent être testés dans la console ou dans la zone de script.

```
>>> for i in range(2, 5):    # i prend successivement les valeurs 2, 3 et 4
...     print(i)             # attention à l'indentation
2
3
4
```

On peut utiliser des tests dans des boucles :

```
>>> for i in range(5):
...     if i % 2 == 0:            # pour tester si i est divisible par 2
...         print(i, 'est pair')
...     else:
...         print(i, 'est impair')
0 est pair
1 est impair
2 est pair
3 est impair
4 est pair
```

Calcul de la somme des entiers naturels inférieurs ou égaux à 1000 :

```
>>> s = 0
>>> for k in range(1001):
...     s = s + k
>>> s
500500
```

Exercice 5

1. Écrire un programme qui affiche les cubes des entiers compris entre 0 et 10.

2. Calculer 100!.

3. Entre 1 et 1000, combien y a-t-il d'entiers divisibles par 17 ?

Dans une boucle `while`, les instructions sont exécutées tant qu'une certaine condition est vérifiée.

```
>>> i = 0
>>> while i < 3:
...     print(i)
...     i = i + 1   # incrémentation de i (on peut aussi écrire i += 1)
0
1
2
```

Dans l'exemple suivant, on cherche le plus petit entier naturel n tel que $2^n > 10^9$.

```
>>> n = 0
>>> while 2**n <= 10**9:
...     n = n + 1
>>> n
30
```

Exercice 6

1. Écrire un programme qui affiche les cubes des entiers compris entre 0 et 10 en utilisant une boucle `while`.

2. Calculer la somme des entiers compris entre 1 et 1000 en utilisant une boucle `while`.

Conventions de rédaction du code

Pour une bonne lisibilité du code, on veillera à respecter les règles suivantes.

1. Utiliser quatre espaces pour l'indentation.

2. Pas de lignes de plus de 80 caractères.

3. Laisser un espace avant et après un opérateur (`=`, `+`, `==`, etc.) et après une virgule (mais pas avant). Pas d'espace après `(` ni avant `)`. Pas d'espace avant un deux-points.

4. Bien choisir le nom des variables.

5. Commenter le code lorsque c'est utile.

Exercices

Exercice 7 Que font les programmes suivants (chercher sans les taper avant de vérifier) ?

1)
```
for k in range(5):
    print(k)
    print("Bonjour")
```

2)
```
for k in range(5):
        print(k)
    print("Bonjour")
```

3)
```
for k in range(5):
        print("Bonjour")
    print(k)
```

4)
```
k = 0
while k < 10:
    print("Bonjour")
```

5)
```
s = 0
for i in range(4):
    for j in range(i):
        s = s + i*j
print(s)
```

Exercice 8 - Suites définies par une relation de récurrence

1) Soit la suite $(u_n)_{n\in\mathbb{N}}$ telle que $u_0 = 0$ et, pour tout entier naturel n, $u_{n+1} = 2u_n + 1$. Considérons le programme suivant :

```
u = 0
for i in range(5):
    u = 2*u + 1
print(u)
```

a. Vérifier à la main que l'on obtient bien u_5.

b. Modifier le programme précédent pour qu'il affiche u_n pour tout $n \in \{1, \ldots, 5\}$ et pas seulement u_5.

c. Écrire un programme permettant de calculer $u_0 + u_1 + \ldots + u_{50}$ (on trouvera 2251799813685196).

2) On veut calculer les premiers termes des suites $(u_n)_{n\in\mathbb{N}}$ et $(v_n)_{n\in\mathbb{N}}$ définies par $u_0 = 1$, $v_0 = 2$, et, pour tout entier naturel n, $u_{n+1} = u_n + v_n$ et $v_{n+1} = u_n - v_n$.

a. Calculer à la main u_1, v_1, u_2, v_2, u_3 et v_3.

Considérons les programmes suivants :

```
u = 1
v = 2
for i in range(3):
    u = u + v
    v = u - v
print(u, v)
```

```
u = 1
v = 2
for i in range(3):
    w = u
    u = u + v
    v = w - v
print(u, v)
```

b. Tester ces programmes et comparer avec les résultats obtenus à la main. Afficher les calculs intermédiaires. Identifier le problème dans le premier programme.

c. En Python il existe une autre manière de régler le problème : l'affectation multiple. Écrire le programme correspondant.

3) Soit la suite de Fibonacci $(u_n)_{n\in\mathbb{N}}$ définie par $u_0 = 1$, $u_1 = 1$, et, pour tout entier naturel n, $u_{n+2} = u_{n+1} + u_n$. Calculer u_{20} (on trouvera 10946). Indication : on pourra représenter le terme u_n par la variable u et le terme u_{n+1} par la variable v.

4) Soit la suite (u_n) définie par $u_0 = 1$ et, pour tout $n \in \mathbb{N}$, $u_{n+1} = \dfrac{u_n}{2} - 1$ si $u_n > 0$ et $u_{n+1} = u_n + 1$ si $u_n \leqslant 0$.

a. Calculer à la main les premiers termes de cette suite.

b. Calculer u_{1000} (on doit trouver $-0{,}9990234375$). Que penser de ce résultat ?

Exercice 9 Trouver tous les couples (x, y) d'entiers naturels tels que $x^2 + y^2 = 1000$.

Exercice 10 Calculer la somme de tous les multiples de 3 ou de 5 inférieurs ou égaux à 1000 (on trouvera 234168).

Exercice 11 Trouver le plus petit entier naturel n tel que $\displaystyle\sum_{k=1}^{n} \frac{1}{k} > 10$ (réponse : 12367).

Exercice 12 Soient a, b, c des entiers tels que $1 \leqslant a \leqslant b \leqslant c$. On dit que (a, b, c) est un **triplet pythagoricien** si $a^2 + b^2 = c^2$. Par exemple, $(3, 4, 5)$ est un triplet pythagoricien.

1) Trouver tous les triplets pythagoriciens formés d'entiers inférieurs ou égaux à 100. Combien y en a-t-il ?

2) Trouver l'unique triplet pythagoricien (a, b, c) tel que $a + b + c = 1000$.