

TP Informatique n° 1

Tests, boucles et types de données usuels

Méthodes numériques pour les suites et les séries

Conformément au programme, nous aborderons l'apprentissage de l'informatique avec plusieurs objectifs :

1. la compréhension et la maîtrise de différents algorithmes (recherche dans une liste, tri d'un tableau de nombres ...)
2. l'utilisation autonome de méthodes numériques (calcul approché d'une intégrale, simulation d'une variable aléatoire, résolution d'un système linéaire ...) en lien avec le cours de mathématiques
3. la connaissance d'un langage de programmation et de son environnement (Python et Spyder)
4. la réalisation d'un projet présenté à l'oral


1 Echauffement

Cette section constitue un rappel des connaissances de première année, et une mise à niveau pour les cubes.

1.1 L'environnement de travail

L'environnement de développement Spyder est choisi pour sa simplicité de mise en oeuvre. Il est constitué

1. d'un éditeur qui permet de saisir le programme (les mots-clé du langage sont colorés, ce qui permet une relecture facile, et l'indentation est automatique)
2. d'une console, qui permet d'exécuter des instructions en ligne de commande (en tapant à la suite de `>>>`) et de suivre le déroulement de son programme
3. d'un explorateur de variables, qui permet de connaître les valeurs contenues dans les variables en cours d'utilisation

 On fera attention à bien enregistrer son travail sous la forme de fichiers `.py` de façon régulière, dans un répertoire créé à cet effet (par exemple `/TP1`) et on exécutera le programme en cours à l'aide de la commande « exécution » ou de la touche F5.

Remarque 1 En Python, toute ligne commençant par un `#` est un commentaire : son contenu sera ignoré lors de l'exécution du programme. Ainsi, il est essentiel d'écrire dans un script des commentaires précisant en toutes lettres ce que le programme fait (ou est censé faire !) afin d'en faciliter la relecture.

De plus, l'indentation (décalage du début de ligne pour aligner verticalement les instructions) est non seulement essentiel pour relire son programme (quelles sont les instructions exécutées après un `if` ? dans une boucle `for` ?) mais aussi obligatoire pour le bon fonctionnement du programme !

1.2 Aide-mémoire des types de données usuels

En Python, une expression est une suite de caractères définissant une valeur, qui possède un type. Saisir par exemple :

```
>>> type(42)
```

```
>>> type(2.5)
```

```
>>> type('Bonjour')
```

Les types usuels sont les suivants :

Type	Exemple	Opérations
Entier <code>int</code>	<code>>>> 42</code>	<code>+</code> , <code>-</code> , <code>*</code> , <code>**</code> , <code>//</code> , <code>%</code>
Flottant <code>float</code>	<code>>>> 0.3</code>	<code>+</code> , <code>-</code> , <code>*</code> , <code>**</code> , <code>/</code>
Booléen <code>bool</code>	<code>>>> 1==2</code>	<code>and</code> , <code>or</code> , <code>not</code>
Chaîne de caractères <code>str</code>	<code>"blablabla"</code>	<code>+</code> , <code>len</code> , <code>in</code>
n-uplets <code>tuple</code>	<code>(0,1,2)</code>	<code>+</code> , <code>len</code> , <code>in</code>

On verra ultérieurement des structures de données plus complexes (liste, tableau multidimensionnel).

Remarque 2 Il est possible de convertir une expression de type entier en flottant (de la même façon qu'un mathématicien, un nombre entier est aussi un nombre réel).

```
>>> a=1
```

```
>>> type(a)
```

```
>>> float(a)
```

```
>>> a=float(a)
```

```
>>> type(a)
```

Ainsi, on constate que la variable `a` possède le type de son contenu. (Noter que l'affectation d'une variable se fait avec `=`.)

■ **Exercice 1** Ecrire un programme qui demande à l'utilisateur deux nombres entiers `a` et `b` (stockés dans les variables `a` et `b`), affiche le contenu de `a` et `b`, puis échange leur contenu et l'affiche à nouveau. On pourra utiliser les instructions `input` et `print`. ■

1.3 Les tests logiques

On considère le programme suivant :

Listing 1 – Test avec alternative

```

1 x = int(input("entrer x: "))
2
3 if x %2 ==1:
4     x = x+1
5 else:
6     x = x//2
7
8 print(x)
```

Que fait ce programme? Que font les instructions `%` , `//` et `==`? Quel est le type de l'expression `x % 2 == 1`?

Remarque 3 S'il y a plus que deux cas nécessitant un traitement différencié, on peut utiliser l'instruction `elif` (contraction de `else` et `if`). L'instruction `else` finale sera traitée seulement si les cas précédents n'ont pas été rencontrés.

1.4 Les boucles inconditionnelles

Ce sont les boucles dont on connaît à l'avance le nombre d'étapes nécessaires pour arriver au résultat. Voici l'exemple du calcul de $n!$:


Listing 2 – Factorielle

```

1 n=int(input("entrer n: "))
2
3 fact=1
4
5 for k in range(n):
6     fact=(k+1)*fact
7
8 print(fact)
```

En particulier, l'instruction `range(n)` produit une liste d'entiers consécutifs entre 0 et $n - 1$:

```
>>> list(range(5))
```

 Faire attention au décalage!

Remarque 4 Quelle est la valeur contenue dans la variable `k` après l'exécution du programme? Que fait l'instruction `range(1,6)`? Réécrire le programme précédent sans décalage.

1.5 Les boucles conditionnelles

Listing 3 – Division euclidienne pour les entiers naturels

```

1 a=int(input("entrer a: "))
2 b=int(input("entrer b: "))
3 q=0
4 r=a
5 while r>=b:
6     q=q+1
7     r=r-b
8 print("q=",q)
9 print("r=",r)
```

2 Méthodes numériques pour les suites

2.1 Calcul de u_n

■ **Exercice 2** Ecrire un programme demandant un entier n à l'utilisateur et affichant les termes successifs u_0, \dots, u_n de la suite de Fibonacci. Que doit-on retenir ? ■

2.2 Calcul d'une valeur approchée de la limite

■ **Exercice 3** On considère la suite arithmético-géométrique définie par $u_0 = 4$ et

$$\forall n \geq 0, u_{n+1} = 2 - u_n/2$$

1. Donner l'expression de u_n en fonction de n .
2. Ecrire un programme qui demande à l'utilisateur de rentrer n et qui calcule u_n .
3. Quelle est la limite ℓ de la suite (u_n) ? Ecrire un programme qui demande à l'utilisateur de rentrer $\varepsilon > 0$ et qui affiche le premier entier n tel que $|u_n - \ell| \leq \varepsilon$.

2.3 Résolution d'équation par dichotomie

■ **Exercice 4** On souhaite résoudre numériquement l'équation $x^3 + x - 1 = 0$.

1. Montrer que cette équation admet une solution unique sur \mathbb{R} et que cette solution notée α est dans $[0; 1]$.
2. Ecrire un script qui demande à l'utilisateur de rentrer un réel $\varepsilon > 0$ et qui affiche une valeur approchée de α à une précision ε près.

3 Méthodes numériques pour les séries

3.1 Calcul de S_n

■ **Exercice 5** Ecrire un programme demandant à l'utilisateur un réel x et un entier n , et calculant $S_n = \sum_{k=0}^n \frac{x^k}{k!}$. ■

3.2 Calcul d'une valeur approchée de la somme

■ **Exercice 6** On considère la série de terme général $\frac{1}{n^2}$ pour $n \geq 1$.

1. Justifier la convergence de cette série.
2. Donner une majoration du reste R_n en fonction de n .
3. On note S la somme de cette série. Ecrire un programme demandant à l'utilisateur un réel $\varepsilon > 0$ et affichant une valeur approchée de S à ε près.

■

3.3 Vitesse de convergence

■ **Exercice 7** On considère la série de terme général $\frac{1}{3^n}$.

1. Justifier sa convergence et rappeler la valeur de $S = \sum_{k=0}^{+\infty} \frac{1}{3^k}$.
2. Ecrire un programme qui demande à l'utilisateur un réel $\varepsilon > 0$ et qui affiche le premier entier n tel que $|S - \sum_{k=0}^n \frac{1}{3^k}| \leq \varepsilon$.
3. Retrouver la valeur de n par le calcul.

■