Travaux Pratiques n°1
Introduction IC et instructions de base Python

I. L'environnement

A. Le système d'exploitation

Peu importe le système d'exploitation utilisé, il est important de bien organiser vos enregistrements et stockages de TP d'informatique pendant l'année. Nous vous conseillons de créer un répertoire `Informatique` dans vos documents dans lequel vous enregistrerez l'ensemble de votre travail.

Créer dans son répertoire `Informatique` un répertoire pour cette première séance, `TP01` par exemple. Durant l'année un répertoire similaire sera créé pour chaque TP.

B. Utilisation du clavier

Pour bien utiliser son clavier, il est recommandé de placer les 2 pouces sur la barre d'espace et les petits doigts sur les touches des bords droit et gauche du clavier, comme la touche `Return` (retour) ou les touches `Shift`. Les plus utiles de ces touches sont :

- `CTRL`, `ALT`, `ALT GR` : servent pour les raccourcis clavier, voir ci-après ;
- `Shift` : la flèche montante de chaque côté des lettres sur le clavier, au-dessus des `CTRL` : si elle est utilisée en combinaison avec une lettre, celle-ci est donnée en majuscule ;
- `Caps Lock` permet de bloquer le `Shift`, donc toutes les lettres tapées seront obtenues en majuscule. Cette touche est déconseillée, car on tape rarement un mot entièrement en majuscule, on se contente généralement du `Shift`.
- `Tab` : touche de tabulation. Très pratique, elle permet de faire un saut de plusieurs colonnes. Nous l'utiliserons en `Python` pour les indentations. Elle permet aussi, lorsque plusieurs champs sont à remplir dans une page web, de passer d'un champ au suivant. Utilisée en combinaison avec `Shift`, elle effectue l'opération inverse, c'est-à-dire qu'elle retourne au champ précédent.
- `Echap, esc` : touche escape, permet parfois de sortir d'un programme, ou de retourner à la situation antérieure.
- Les touches `Suppr` effacent le caractère à droite du curseur, alors que la touche au-dessus de la touche `Entrée` efface le caractère à gauche du curseur.
- Pour les touches contenant plusieurs caractères, c'est le caractère du bas qui est obtenu, sauf si on les utilise en combinaison avec la touche `Shift`, auquel cas on obtient le caractère du haut, ou avec la touche `ALT GR`, auquel cas on obtient le caractère à droite de la touche.

Les systèmes d'exploitation proposent chacun une série de raccourcis clavier. Il s'agit de combinaisons de touches qui permettent d'accéder rapidement à certaines commandes. Ces commandes sont également accessibles grâce à la souris (avec un clic droit par exemple, ou en passant par la barre de menu en haut de la fenêtre), mais lorsqu'on travaille sur ordinateur on a souvent les mains sur le clavier, donc il est intéressant de les connaître. Voici la liste des plus utiles :

- `CTRL-C` pour copier : grâce à la souris on sélectionne une zone de texte, ou un ou plusieurs fichiers. Ils sont alors surlignés (grisés). L'utilisation de `CTRL-C` enregistre cette zone ou ces fichiers dans une mémoire volatile appelée *presse-papiers*.
- `CTRL-X` pour couper : cette combinaison supprime la zone sélectionnée et l'enregistre, dans le presse-papiers. On peut alors la coller ailleurs.
- `CTRL-V` pour coller : cette combinaison recopie à la place du curseur le contenu du presse-papiers, c'est-à-dire la zone de texte qui a été copiée ou coupée.
- `CTRL-S` pour sauvegarder le fichier sur lequel on travaille. Il faut prendre l'habitude d'utiliser cette combinaison très souvent.
- `CTRL-A` sélectionne tout.
- `CTRL-Z` annule la dernière opération, par exemple la saisie de la dernière phrase.

C. Environnement Python

Pour écrire des programmes en `Python` et les exécuter on utilise un *environnement de développement intégré* (EDI). Pour les différentes séances de l'année, vous avez le choix (Spyder, Pyzo...). Lancer votre interface à partir d'un raccourci créé sur le bureau par exemple (cf. tuto fourni par Mr Michel).

Une fenêtre s'ouvre alors et vous avez accès à une fenêtre `Python Shell`. C'est *l'interpréteur* (shell en anglais), on peut y taper des instructions courtes en `Python`.

Il y a également une autre fenêtre qu'on appelle *l'éditeur*, dans laquelle il est possible d'écrire du code, avant de l'exécuter puis d'observer les résultats éventuels dans le Shell.

II. Présentation du Shell et de l'éditeur

Le shell

La fenêtre de droite (ou en haut à droite) contient le *shell*, qui est l'interpréteur. Il est possible d'ouvrir plusieurs shells, accessibles par différents onglets.

Le raccourci CTRL-L efface le shell sur Pyzo.

▷ Calculer $7 \times 11 \times 13$ dans le shell. Utiliser la flèche montante pour calculer ensuite $7 \times 11 \times 13 \times 17$: il suffit de rappeler l'entrée précédente et d'ajouter la fin de la ligne.

On peut dans le shell demander la documentation sur une instruction ou une fonction :

```
>>> ? print
```

ou

```
>>> help(print)
```

L'éditeur

La fenêtre en bas à gauche (ou à gauche) contient l'éditeur. Là encore il est possible de travailler sur plusieurs fichiers à la fois, accessibles par des onglets.

Il est fortement recommandé de sauvegarder (CTRL-S) *l'éditeur avant même le début de l'écriture du programme.*

▷ **Exercice 1.**

▷ Taper et exécuter dans le Shell (en pressant la touche `Entrée` entre chaque expression) et répondre aux questions ci-dessous.

```
2+5 2*5 2**5 5/2 5.0/2 1/10000 2e5 1+3*10 (1+3)*10 3*10**2
```

La fenêtre doit ressembler à ceci :

```
>>> 2+5
7
>>> 2*5
10
...
```

▷ Que représente l'opérateur ** ? Attention le symbole ∧ ne permet pas d'obtenir la puissance avec Python !

▷ Quel symbole représente le séparateur décimal ?

▷ Comment est notée l'écriture scientifique d'un réel $a * 10^n$?

▷ Les priorités usuelles des opérateurs (*,+,**) sont-elles respectées ?

▷ Taper maintenant les instructions suivantes : `23/5 23//5 23%5`

▷ Que représentent les opérateurs // et % ?

L'interpréteur permet de faire des petits calculs. Pour des tâches plus complexes, on écrit un programme dans l'éditeur. C'est dans l'éditeur que l'on écrit les programmes, qui sont exécutés ensuite dans l'interpréteur.

▷ **Exercice 2.**

▷ Ouvrir une nouvelle fenêtre. Sauvegarder immédiatement le fichier dans un dossier correct et sous un nom correct. Taper les lignes suivantes :

```
1 print("Bonjour")
2 2*7
3 print(3*7)
4 4*7
5 a=5*7
6 print(a)
```

▷ Sauvegarder le fichier, l'exécuter et constater le résultat dans l'interpréteur.

▷ Pourquoi 14 et 28 ne sont-ils pas affichés ?

▷ **Exercice 3.**

▷ Ouvrir une nouvelle fenêtre, la sauvegarder sous le nom `Ex03.py`. Saisir dans l'éditeur :

```
1 for i in range(1,7):
2         print(i**2)
3 print("La valeur finale de i est",i)
```

Attention à respecter l'alignement : normalement, après les deux points, le curseur est automatiquement placé à la cinquième colonne. On dit que la ligne est *indentée*.

▷ Que se passe-t-il si la troisième ligne est également indentée ?

III. Variables

Une variable est une zone de la mémoire de l'ordinateur qui :

- porte un nom
- contient de l'information (une valeur).

Le nom est une chaîne de caractère arbitrairement longue, éventuellement avec des chiffres (mais pas en première position). Attention : les minuscules et les majuscules sont différenciées : `x1` est différent de `X1`. La valeur peut être un entier, un réel (nous dirons un *flottant*), un caractère ('a' ou "a" par exemple), une chaîne de caractères ('Bonjour, ça va ?', "Oui, et toi ?"), ou encore d'autres choses. Ce sont les types de variables.

On affecte la variable en attribuant une valeur à son nom. En Python on utilise l'instruction `x=5`, qui se lit *x reçoit 5*. En algorithmique on écrit plutôt x ← 5. On peut ensuite utiliser cette variable, par exemple l'instruction `x+3` renverra `8`. On peut réaffecter la variable à tout moment : `x=14`. La valeur précédente sera oubliée. On peut aussi *incrémenter* une variable grâce à l'instruction `x=x+1`.

▷ **Exercice 4.**

Dans l'interpréteur taper les instructions suivantes (en pressant la touche entrée après chacune d'entre elles) :

```
>>> x=10
>>> x
>>> y=x
>>> x=15
```

Anticiper les valeurs de x et y puis le vérifier. Taper ensuite :

```
>>> x,y=8,5
>>> x
>>> y
>>> x*y
```

Il s'agit d'une affectation multiple.

▷ **Exercice 5.**

Créer une nouvelle fenêtre de l'éditeur, taper les instructions suivantes sans exécuter le programme :

```
1 x=10
2 y=15
3 z=x+y
4 x=y
5 y=z
6 print(x+y+z)
```

Que va t-il se passer à l'exécution ? Le vérifier.

▷ **Exercice 6.** Inversion des valeurs de deux variables

Après chacune des trois séquences ci-dessous, quelles valeurs ont les variables x et y ? Le deviner puis vérifier.

▷ x=42, y=10, x=y, y=x

▷ x=42, y=10, z=x, x=y, y=z

▷ x=42, y=10, x,y=y,x

▷ **Exercice 7.** Le type tableau ou liste (list)

Il s'agit de variables contenant plusieurs données ordonnées, chacune portant un numéro à partir de 0. Exécuter dans l'interpréteur et comprendre les commandes suivantes :

```
>>> A=[6,4,1.2,213,0,2/3,1,'Ab']
>>> A
>>> A[1]
>>> A[0]
>>> A[4]
>>> A[-1]
>>> len(A)
>>> A[1]=418
>>> A
```

Que renvoie A[k] ? A[-1] ? len(A) ? Nous aurons l'occasion lors du TP suivant de revenir plus en profondeur sur cette notion de liste.

▷ **Exercice 8.** Le type chaîne de caractères (str)

Exécuter maintenant :

```
>>> x='Bonjour'
>>> y='Adieu'
>>> x[1]
>>> y[2]
>>> x[7]
>>> y[-2]
```

Puis ;

```
>>> z=x+y
>>> z
>>> t=x+' et '+y
>>> t
>>> print(t)
>>> print(4*x)
```

L'opérateur + sur les chaînes de caractère réalise la concaténation.

IV. Boucles, tests

A. Observation

▷ **Exercice 9.** Boucles for et while

Dans l'éditeur, taper et exécuter les lignes suivantes :

```
1 for i in range(10,15):
2         print("Bonjour",i)
```

Quelles valeurs prend successivement i ? Saisir puis exécuter maintenant (while signifie "tant que") :

```
1 i=10
2 while i<15 :
3         print("Bonjour",i)
4         i=i+1
```

Le résultat est-il identique ?

▷ **Exercice 10.** Instruction conditionnelle if

Dans l'éditeur, taper les deux programmes suivants :

```
1 if 10**2<99 :
2         print('Paf')
3 if 5**2>20 :
4         print('Pif')
5 if 10**2<99 or 5**2>20 :
6         print('Pof')
```

```
1 if 10**2<99 :
2         print('Paf')
3         if 5**2>20 :
4                 print('Pif')
5 if 10**2<99 or 5**2>20 :
6         print('Pof')
```

Exécuter ces programmes et expliquer la différence.

▷ **Exercice 11.** Variables d'accumulation

Dans l'éditeur, taper puis exécuter les lignes suivantes :

```
1 somme=0
2 for i in range(1,5):
3         somme=somme+i
4         print(somme)
```

Après exécution, que valent i et somme ? (Vérifier dans l'interpréteur)

▷ **Exercice 12.**

Dans l'éditeur, taper puis exécuter les lignes suivantes :

```
somme=0
for i in range(10,18):
        if i%3==0:
                somme=somme+i
```

Quelle est la valeur de somme après exécution ? Vérifier.

▷ **Exercice 13.** Instruction conditionnelle if... else...

```
for n in range(20):
        if n%2==0:
                print(n,"est pair")
        else:
                print(n,"est impair")
```

Que fait le programme ci-dessus ? Modifier ce programme pour afficher la liste des multiples de 7 compris entre 1 et 1000.

B. A vous de jouer !

▷ **Exercice 14.** Calcul de la factorielle

Ecrire une boucle permettant de calculer 10 !, puis 100 !, puis 1000 !

▷ **Exercice 15.** Calcul de la somme des carrés

Ecrire une boucle permettant de calculer la somme des carrés de 10 à 100.

▷ **Exercice 16.**

La liste des nombres strictement inférieurs à 10 qui sont multiples de 3 ou de 5 est 3, 5, 6 et 9. Leur somme est égale à 23. Quelle est la somme des multiples de 3 ou de 5 strictement inférieurs à 1000 ?

▷ **Exercice 17.** Tables de multiplication

Ecrire un programme qui demande à l'utilisateur d'entrer un entier n grâce à l'instruction

```
n=int(input("Valeur de n : "))
```

et qui affiche la table de multiplication de n jusqu'à 10 :

```
Valeur de n : 7
1 * 7 = 7
2 * 7 = 14
3 * 7 = 21
...
```

On pourrait supprimer les espaces en ajoutant l'option **sep=”** à l'instruction **print**

```
print('Anti', 'constitutionnellement',sep='')
```

▷ **Exercice 18.** Nombres premiers (pour les plus aguerris, ne pas y passer plus de 30 min)

Tous les programmes suivants doivent être testés.

▷ Ecrire un programme qui demande un entier n à l'utilisateur, puis qui affiche ses diviseurs stricts (c'est-à-dire différents de 1 et lui-même). Par exemple, si n=20, le programme doit afficher 2, 4, 5 et 10.

▷ Modifier le programme précédent de façon à utiliser une boucle **while** et non une boucle **for**.

▷ Modifier le programme précédent de façon à ce qu'il renvoie le plus petit diviseur de n autre que 1. (Changer la condition)

▷ Modifier le programme précédent de façon à ce qu'il renvoie **True** si n est premier et **False** sinon.

▷ Afficher les nombres premiers compris entre 1 et 100 (on doit en obtenir 25).

▷ Les entiers 2005001, 2005007 et 2000009000009 sont-ils premiers ? Chercher leur décomposition en facteurs premiers.

V. Instructions conditionnelles

Les exercices suivants nécessitent l'utilisation des instructions **if**, **elif** et **else**.

▷ **Exercice 19.**

▷ À l'aide des instructions **input** et **print**, écrire un programme qui demande son nom à l'utilisateur et lui dit bonjour, en l'appelant par son nom.

▷ Poursuivre son programme en demandant à l'utilisateur son année de naissance, et si son anniversaire a déjà eu lieu cette année.

▷ Afficher ensuite la phrase “Tu as donc ... ans.”

Rappel

Ne pas oublier de commenter ses programmes grâce au symbole # !

Ceci permet d'en dégager les grandes lignes, de séparer les étapes, d'expliquer le rôle d'une variable ou d'une instruction, etc... Ainsi vous le comprendrez si vous le reprenez plus tard, et de même pour un autre programmeur qui voudrait le prolonger.

Avec Pyzo par exemple il est possible de sélectionner toute une zone à la souris et de la commenter en pressant CTRL-R ou de la décommenter en pressant CTRL-T.

VI. Instructions itératives

Les exercices suivants nécessitent l'utilisation des instructions de boucles **for** et **while**.

▷ **Exercice 20.**

▷ Ecrire un programme qui affiche la liste de toutes les factorielles de 0 à n!

▷ Tester avec $n = 20$, $n = 100$ puis $n = 1000$.

▷ **Exercice 21.** Pour tout $n \in \mathbb{N}^*$ on pose : $S_n = \frac{4}{n}\sum_{k=0}^{n-1}\sqrt{1-\left(\frac{k}{n}\right)^2}$

a. Calculer à la main : $S_1 =$ $S_2 =$

b. Écrire un programme qui demande la valeur de n à l'utilisateur, puis calcule et affiche la valeur de S_n. Pour faire la racine carrée, on pourra utiliser la puissance 0.5.

Vérifier ce programme pour $n = 1$ et $n = 2$.

Tester ensuite ce programme pour de grandes valeurs de n. Avez-vous une conjecture sur la limite de S_n ?

Oui. Je pense que

▷ **Exercice 22.**

L'ordinateur choisit aléatoirement un nombre compris entre 1 et 100. Vous devez le deviner en lui proposant des nombres. Il répond à chaque essai «trop grand» ou «trop petit».

a. Écrire un programme permettant de jouer à ce jeu. Le tester.

- Pour obtenir un entier aléatoire compris entre 1 et 100 on importe au début du programme la fonction `randint` du module `random` :

```
from random import randint
...
a=randint(1,100)
```

- On pourra commencer par écrire juste un tour : le programme demande un nombre à l'utilisateur, il le compare à son nombre secret et affiche «trop grand», «trop petit» ou «bravo vous avez trouvé».

b. Ajouter à la fin l'affichage du nombre d'essais qui vous ont été nécessaires pour trouver le nombre.

▷ **Exercice 23.** La *Suite de Syracuse* est définie de la façon suivante :

- Le premier terme est un entier strictement positif u_0.
- Les termes suivants sont définis par la relation de récurrence :

$$\forall n \in \mathbb{N} \qquad u_{n+1} = \begin{cases} u_n/2 & \text{si } u_n \text{ est pair} \\ 3u_n + 1 & \text{si } u_n \text{ est impair} \end{cases}$$

a. Vérifier à la main que pour $u_0 = 1, 2, 3, \ldots 10$, la suite finit toujours par arriver à 1.

1
2 →
3 → → → → → → →
4 →
5 →
6 →
7 →
8 →
9 →
10 →

b. Vérifier que si l'un des termes de la suite est égal à 1 alors la suite ne prend plus que 3 valeurs.

1 → → →

c. Écrire un programme demandant la valeur du premier terme, et affichant tous les termes suivants jusqu'à 1. Quel type de boucle utilisera-t-on ?

Attention à la division

On rappelle que `u/2` et `u//2` n'ont pas le même résultat, même si u est un entier pair.

d. Raffiner le programme précédent pour afficher à la fin du calcul le nombre de termes qu'il aura fallu calculer pour arriver à 1. Ce nombre de termes est appelé *longueur* de la suite.

e. Modifier le programme pour obtenir l'affichage suivant :

```
Si u0 = 1 alors la longueur de la suite est 1
Si u0 = 2 alors la longueur de la suite est 2
Si u0 = 3 alors la longueur de la suite est 8
...
Si u0 = n alors la longueur de la suite est ...
```

f. La conjecture de Syracuse prédit que pour tout nombre entier u_0 la suite finit par arriver à 1.

Vérifier expérimentalement cette conjecture jusqu'à 10^5.

On veillera à supprimer tout affichage superflu.

▷ **Exercice 24.** Soit (a_n) la suite définie par $a_0 = 1$ et pour tout $n \in \mathbb{N}$, $a_{n+1} = 1 + \frac{1}{a_n}$.

a. Écrire un programme qui demande à l'utilisateur un entier n puis qui calcule et affiche la valeur de a_n.

   On remarquera qu'il suffit d'une unique variable a (de type flottant) pour stocker les termes successifs de la suite.

b. Modifier votre programme pour qu'il affiche tous les termes intermédiaires, et utiliser ce programme pour établir une conjecture sur la limite de cette suite.

Je pense que

c. On définit maintenant la *suite de Fibonacci* par $u_0 = u_1 = 1$, et pour tout $n \in \mathbb{N}$: $u_{n+2} = u_{n+1} + u_n$.

   Écrire un programme qui affiche les n premiers termes de cette suite.

   On remarquera qu'il faut maintenant utiliser deux variables pour stocker les termes de la suite : une pour u_n et une pour u_{n-1}.

d. Déterminer expérimentalement la limite de $\frac{u_{n+1}}{u_n}$.

Je pense que