Python

TP 0 - Introduction au langage python

Simon Dauguet
simon.dauguet@gmail.com

4 septembre 2025

Le but de ce premier TP introductif est de se familiariser avec l'environnement qui sera proposé cette année et de (ré)activer certains réflexes qui seront utiles tout au long de l'année. Ce TP est sensé être conçu comme une sorte de cours-activité-découverte-bibliothèque. On vous demande de taper des lignes de commandes et d'observer les réactions de l'ordinateur. De la place est laissée pour vos commentaires afin que vous puissiez vous y référer dans la suite de l'année pour savoir comment résoudre les problèmes que vous allez rencontrer. Il faut voir ce TP comme une sorte d'indexe des situations problématiques que vous pourriez rencontrer avec les solutions proposées. C'est un index fait de vous pour vous. Donc il faut être suffisamment explicite maintenant sur la nature du problème et sa solution pour que vous puissiez le réutiliser plus tard dans n'importe quel contexte.

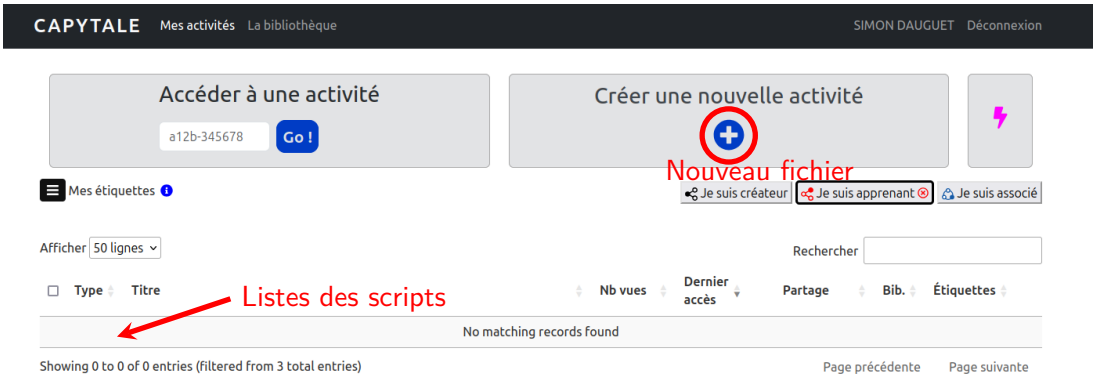
Table des matières

1 Prise en main

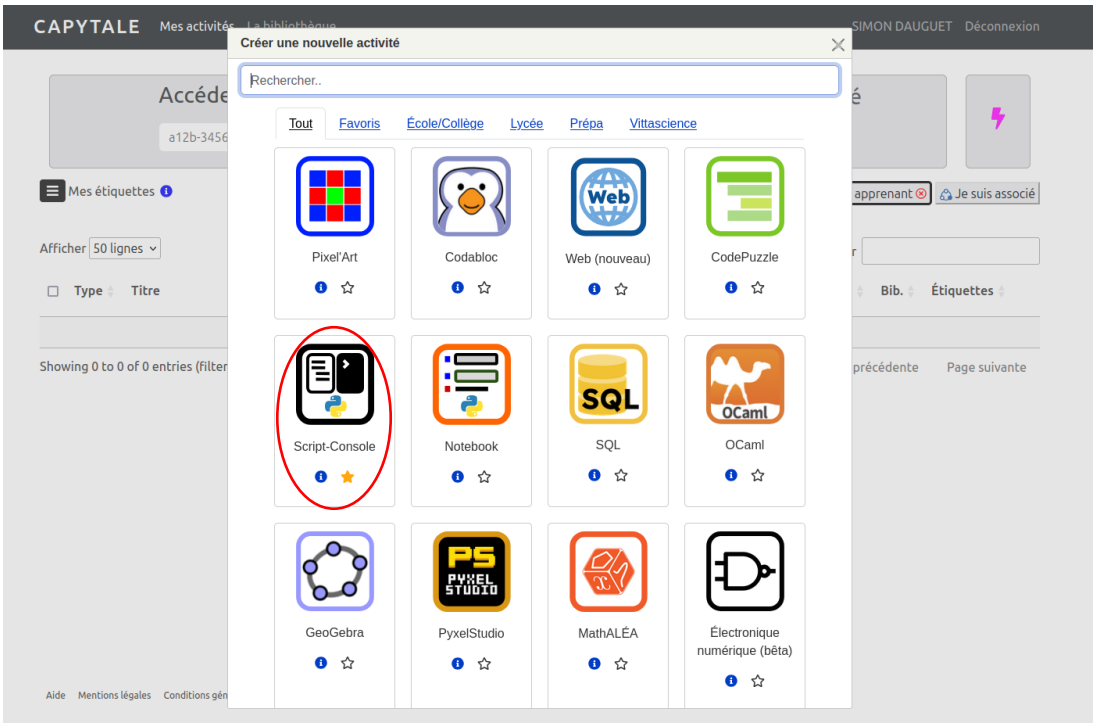
1.1 L'interpréteur Python

Il est recommandé d'utiliser l'application Capytale dans l'ENT pour pouvoir coder en python (aucune installation à faire et accès à tous les fichier depuis n'importe quel ordinateur connecté à internet).

Une fois connecté à l'ENT, il suffit de cliquer sur le bouton "Capytale V2".



Vous pouvez alors choisir l'activité `Script-Console`.



Et vous aurez accès à votre script :

Nous allons dans un premier temps manipuler l'interpréteur. Il s'agit d'une fenêtre avec une une invite de commandes symbolisé par les 3 chevrons `>>>` qui invite à rentrer une instruction. Cette fenêtre fonctionne un peu comme un chat. À chaque instruction donnée, Python répondra immédiatement dessous.

1.2 Type numérique et type chaîne de caractères

1.2.1 Les premiers types en python

Les objets informatiques sont classés en fonction de leur nature – de leur type –. Il en existe plusieurs. Certains types sont communs à tous les langages. Des langages informatiques utilisent des types particuliers qui leur sont propres. On va voir ici quelques types d'objets informatiques relatif à python. Nous en verront d'autres plus tard.

Parmi les types principaux de python, on peut citer les entiers (`int`, de *integer* en anglais), les flottants (`float`, de *float* en anglais) correspondant aux réels (avec des restrictions qui seront discutées plus tard) et des chaînes de caractères (`str`, de *string* en anglais).

`20`, `3`, `5.45`, `"bonjour"` sont des données. Chaque donnée a un type. Il s'agit de :

- du type *entier* ou *int* pour les données `20` et `3`
- du type *flottant* ou *float* pour la donnée `5.45`
- du type *chaîne de caractères* ou *string* pour la donnée `"bonjour"`

Il existe d'autres types de données que nous verrons par la suite.

Chaque type se manipule avec des opérateurs spécifiques et se définit à partir d'une syntaxe (d'une "orthographe") particulière.

Pour connaître le type d'une donnée, on peut utiliser la fonction `type`. Entrer les instructions suivantes et commenter les résultats obtenus :

Instructions	Résultats + Commentaires
`>>>type(3)`	
`>>>type(1.5)`	
`>>>type(3.0)`	
`>>>type("bonjour")`	
`>>>type("3")`	
`>>>type(3+1.5)`	

1.2.2 Les premières commandes Python - les opérateurs

Nous allons commencer à dialoguer avec l'interpréteur Python. Taper dans l'interpréteur les commandes de la colonne de gauche du tableau suivant et inscrire dans la colonne de droite la réponse de Python ainsi qu'un commentaire, si nécessaire (expliquer pourquoi la réponse obtenue est différente de celle attendue par exemple).

Attention à bien recopier *exactement* l'instruction.

Les opérateurs `+`, `*`, `/`, `//`, `%`, `**` sont des opérateurs Python. Ils permettent de réaliser des opérations sur les données.

Instructions	Résultat + Commentaires
`>>>20+1`	
`>>>20/3`	
`>>>20//3`	
`>>>20%3`	
`>>>5.45*100`	
`>>>2**4`	
`>>>(3+2)*5`	
`>>>3+2*5.0`	
`>>>"bonjour"`	
`>>>'bonjour'`	
`>>>"il fait "+"beau"`	
`>>>"bonjour"*5`	

Remarque :
Pour les chaînes de caractères, l'opérateur + s'appelle la concaténation (il fusionne bout à bout les chaînes de caractères).

Lorsque l'on donne une instruction erronée (avec une faute d'orthographe ou un opérateur non adéquat, par exemple), Python renvoie un message d'erreur. Il est impératif de bien savoir lire et interpréter les messages d'erreurs. Ils contiennent des indications sur la nature de l'erreur de l'instruction.

Taper les instructions suivantes, observer les erreurs et commenter.

Instructions	Erreur observée + Commentaires
`>>>20/0`	
`>>>20@3`	
`>>>"bonjour"/3`	
`>>>"bonjour"+5`	
`>>>(3+2))*5`	
`>>>(3+2*5`	

1.2.3 Le transtypage

On peut parfois transformer une donnée d'un certain type en une donnée d'un autre type. Cela s'appelle le transtypage.

Données de départ	Types	Instructions	Types
`3.0`		`>>>int(3.0)`	
`3.5`		`>>>int(3.5)`	
`3`		`>>>float(3)`	
`4`		`>>>str(4)`	
`"3"`		`>>>int("3")`	
`"3.5"`		`>>>float("3.5")`	
`"3.5"`		`>>>int("3.5")`	
`"bonjour"`		`>>>int("bonjour")`	

Exercice 1 (Correction d'instructions erronées) :
Corriger les instructions suivantes pour qu'elles ne provoquent plus d'erreurs.

1. On veut obtenir la chaîne de caractères `"Vous etes 2 dans ce binome de TP"` :

```
>>>"Vous etes" + 2 + " dans ce binome de TP"
```

2. On veut afficher `"blablablablablablablablablabla"`, c'est-à-dire `"bla"` 10 fois de suite.

```
>>>"bla"*10.0
```

1.3 Les variables et l'affectation

1.3.1 L'affection

Pour conserver les données pour une utilisation ultérieure, on peut les mettre dans des contenants qu'on appelle des *variables*. Le rangement d'une donnée dans une variable s'appelle *l'affectation*. L'affection se fait par le biais de l'opérateur **=**, qui est l'opérateur d'affectation (⚠cet opérateur, en python, ne se lit que de gauche à droite !).

L'affectation de fait en deux temps :

- L'évaluation (c'est-à-dire calcul) de la partie droite de l'opérateur d'affectation. Le résultat de cette évaluation est la donnée qui va être conservée dans la variable.
- Le rangement de la donnée de l'évaluation dans la variable à gauche de l'opérateur d'affectation.

⚠ Ne pas confondre l'opérateur d'affection **=** avec l'opérateur mathématique $=$ d'égalité. L'affectation n'est pas une opération mathématique.

En particulier, l'affectation **n'est pas** symétrique. Elle ne se lie **que** de gauche à droite. On doit toujours avoir la variable à gauche, et la donnée à sauvegardée à droite.

Exercice 2 (Affectations) :
Taper les instructions suivantes dans l'interpréteur et commenter :

Instructions	Résultats + Commentaires
`>>>x=2`	
`>>>x`	
`>>>y=1.5`	
`>>>x+y`	
`>>>"x-y"`	
`>>>z=str(x-y)`	
`>>>z`	
`>>>z-0.5`	
`>>>str(float(z)-0.5)`	
`>>>a,b=2,3`	
`>>>a*b`	
`>>>type(z)`	

Remarque :
Le langage Python permet de faire plusieurs affectations à la fois. Il suffit de les séparer par des virgules, sur une seule ligne. On met les variables séparées par des virgules, l'opérateur d'affection et les valeurs respectives que doivent sauvegarder les variables. Par exemple

```
1 >>> a,b,c = 1,1.2,"Hello world"
```

permet d'affecter la valeur `1` à la variable `a`, la valeur `1.2` à la variable `b` et la valeur `"Hello world"` à la variable `z`.

1.3.2 Les noms de variables

Un nom de variable peut être n'importe quelle suite de caractère vérifiant les critères suivants :

1. Démarrer par une lettre sans accent (majuscule ou minuscule)
2. Contenir uniquement des lettres sans accents, des chiffres et le tirets de soulignement (l'"underscore" ou "tiret du bas" ou "tiret du 8")

Il est recommander d'utiliser des noms de variables qui décrivent le rôle de la variable afin de pouvoir simplifier la lecture du code et pouvoir s'y retrouver plus facilement.

Exemple 1.1 :

```
nom,prenom,age,etudiant="Dupont","Joe",20,True
```

1.4 Le mode éditeur

On va maintenant d'intéresser à l'éditeur (partie de gauche dans la fenêtre de Python). Il ne faut pas hésiter à utiliser l'interpréteur pour vérifier une commande, tester une instruction, tester une fonction ...

L'éditeur sert à écrire à un programme en entier que Python pourra lire afin de l'utiliser dans l'interpréteur plus tard.

Pour pouvoir s'y repérer facilement dans le fichier du programme, il est recommandé de choisir un en-tête qui permet d'ordonner les choses afin de pouvoir s'y retrouver facilement. On recommande le cartouche suivant :

```
#!/usr/bin/python3
#-*- codinf : utf8 -*-

"""Documentation du fichier"""

#fichier : nom_fichier.py
#auteur : prof
#date : 27 juillet 2018

#---------- Modules ----------


#---------- Programmes -----------
```

1.4.1 Le programmeur et l'utilisation du programme

Il faut distinguer le programmeur (ou développeur) avec l'utilisateur. Le programme fabrique une fonction, un algorithme dans l'éditeur que l'utilisateur pourra utiliser à loisirs dans l'interpréteur.

Seul le programmeur peut modifier le fichier du programme (fichier en .py) et l'exécuter (touche F5) pour qu'il soit disponible à l'utilisation dans l'interpréteur par l'utilisateur.

1.4.2 Les fonctions d'entrées / sorties

Afin que l'utilisateur puisse utiliser correctement le programme, le programmeur introduit dans l'algorithme des fonctions d'entrées et de sorties. Les entrées du programme sont les paramètres dont dépend le programme et que doit spécifier l'utilisateur pour que le programme puisse tourner.

La fonction `input()` est une fonction d'entrée que peut introduit le programme dans l'algorithme. La fonction `input()` prend en argument une chaîne de caractère. À l'appel de cette fonction, la chaîne de caractère dans l'argument est affichée et attend une réaction de l'utilisateur en réponse. L'utilisateur doit terminer par la touche "Entrée" pour que le programme puisse continuer son exécution.

La suite de caractère tapées par l'utilisateur doit être conservée dans une variable pour qu'elle puisse être utilisé par le programme.

La donnée récupérée par la fonction `input()` est une chaîne de caractères. Si l'on souhaite récupérer un nombre, il ne faudra pas oublier de transtyper.

Instructions	Commentaires
`>>>num=input("entrez un chiffre entre 1 et 5 : ")`	
`>>>type(num)`	
`>>>num=num+1`	
`>>>num=int(num)`	
`>>>type(num)`	
`>>>num=num+1`	

Exercice 3 (Afficher la somme de deux valeurs fournies en entrée par l'utilisateur) :
Nous allons créer notre premier programme.

1. Cliquer sur Fichier > Nouveau pour ouvrir un nouveau fichier vide. Le renommer TP_0_0.py et l'enregistrer dans votre dossier personnel.
2. Vérifier que l'en-tête est adéquat
3. Taper les lignes suivantes dans l'éditeur le fichier :

```
1 var1 = input("Entrez une première valeur ")
2 var2 = input("Entrez une deuxième valeur ")
3 var_som = var1 + var2
```

4. Exécuter le fichier. Qu'obtient-on dans l'interpréteur.
5. On souhaite que la variable **var_som** contienne la somme des deux valeurs entrées par l'utilisateur. Corriger le code pour qu'il fasse ce que l'on veut.

1.4.3 De l'importance de commentaires

Des commentaires doivent toujours figurer dans les codes des programmes : ils permettront aux autres programmeur de pouvoir lire et comprendre le code plus facilement et vous seront utiles pour reprendre le code à froid.

Les commentaires sont des lignes ignorées par Python lors de la lecture du code. Ils sont introduits en début de ligne par le caractère #. Tout ce qui suit ce symbole sur la ligne est considéré comme un commentaire.

On notera que tous les codes rendus devront impérativement contenir des commentaires. Par exemple, le code suivant n'est pas facile à lire et à comprendre. Quelques ligne de commentaires pour aider à la lecture ne sont pas superflues.

Essayer de comprendre ce que fait le code suivant, d'abord seulement en le lisant, puis en essayant de le tester.

```
def Fct(n) :
    F=[0,1]
    while F[-1]<=n :
        F=F+[F[-1]+F[-2]]
    F.remove(F[-1])
    if F[-1]==n :
        a=[len(F)-1]
        return(F[-1],[F[-1]])
    else :
        a=[len(F)-1]
        R=[F[-1]]
        r=n-F[-1]
        while r!=0 :
            k=0
            while F[k]<=r and k<=a[-1]-1 :
                k=k+1
            R=R+[F[k-1]]
            a=a+[k-1]
            r=r-F[k-1]
        return(sum(R), a)
```

Vous noterez qu'il peut être utile aussi de choisir judicieusement les noms des variables. Avoir un nom de variable clair permet de pouvoir s'y retrouver et d'aider à la lecture d'un code. C'est une sorte de commentaire intégré.

On prendra garde aussi à donner un nom qui fait sens à une fonction. Toutes les appeler **`fct_k(n)`** ne va pas vraiment aider à pouvoir les différencier. Surtout s'il y en a un grand nombre.

1.5 Le type booléen

Le typer booléen est un type de données particulier. Il ne peut prendre que deux valeurs : `True` (vrai) ou `False` (faux). Les opérateurs qui s'appliquent aux booléen sont des connecteurs logiques. Ce sont **`or`**, **`and`**, **`not`**.

!!! ATTENTION !!!

⚠ Les majuscules sont très importantes : **`true`** et **`false`** vont être traités comme des noms de variables par Python.

Certains opérateurs qui s'appliquent sur les autres types de données produisent des booléen ; ce sont les opérateurs de comparaison :

Opérateurs	Sens
`==`	"est égal à"
`!=`	"est différent de"
`<,<=,>,>=`	"est inférieur, inférieur ou égal, supérieur, supérieur ou égal"

Ces opérateurs permettent de faire des tests. Le résultat de l'opération est `True` ou `False` selon si la comparaison est vraie ou fausse.

Exercice 4 (Opérateurs booléens) :
Taper les instructions suivantes dans l'interpréteur et commenter

Instructions	Résultats + Commentaires
`>>>1==1`	
`>>>1==3`	
`>>>1!=3`	
`>>>1==1.0`	
`>>>3+7==5*2`	
`>>>(1==1) and (2<2)`	
`>>>(1==1) and (2<=2)`	
`>>>(2==15) or (3>2)`	
`>>>1<4<15`	
`>>>val=(1==1)`	
`>>>type(val)`	
`>>>not val`	
`>>>True and False`	
`>>>True or False`	

Les opérateurs `or` et `and` sont les mêmes que les opérateurs logiques mathématiques. On peut écrire leur tables de vérité.

1.6 Importer des modules et utiliser l'aide

On peut utiliser des variables ou des fonctions des variables ou des programmes déjà prédéfinis qui sont rangés dans des modules. Les modules peuvent être chargé en utilisant le mot clé `import`, suivi du nom de ce module.

On peut afficher une aide à l'aide de la fonction `help()` en indiquant en argument la fonction, la variable ou le module dont on veut une description.

Exercice 5 (Afficher l'aide d'un module) :

1. Dans l'interpréteur de Python, taper les lignes suivantes :

```
>>> import numpy as np
>>> help(np)
```

   Que se passe-t-il ? Commenter le début de l'affichage.
2. Vérifier que `numpy` contient les fonctions `cos`, `arccos`, `tan`, `arctan`, `sqrt`, `pi`, `exp`, `log`, `log10`. Répondre alors aux questions suivantes en s'aidant de l'interpréteur :
   (a) Chercher l'aide de la fonction `arccos`. Qu'en pensez-vous ? Que dit le premier paragraphe ? S'en servir pour donner le sens de la fonction `arccos` et `arctan`.
   (b) À quoi correspondent les fonctions `cosh`, `sinh` ?
   (c) À quoi sert la fonction `sqrt` ?
   (d) Quelle est la fonction qui donne le logarithme népérien ?
   (e) Quelle est la différence entre `e` et `exp` ?

2 Exercices

Exercice 6 (Échanger la valeur de deux variables) :

1. Taper les lignes suivantes dans l'interpréteur et commenter :

```
>>>a,b=2,3
>>>b=a
>>>a=b
>>>a,b
```

2. Proposer un moyen d'échanger les valeurs de deux variables `a` et `b` à l'aide d'une tierce variable `c`.
3. Tester le code suivant et commenter :

```
>>>a,b=2,3
>>>b,a=a,b
>>>a,b
```

Exercice 7 (Volume d'un cône) :
Dans un nouveau fichier TP_0_2.py, écrire un code permettant de demander à l'utilisateur un rayon et une hauteur et qui renvoie le volume du cône correspondant.

Exercice 8 (Conversion Celsius / Fahrenheit) :
En supposant une loi affine et en sachant que $32°\mathrm{F} = 0°\mathrm{C}$ et que $212°\mathrm{F} = 100°\mathrm{C}$, écrire un code dans un fichier TP_0_3.py qui demande à l'utilisateur une température en Celsius et renvoie la température correspondante en Fahrenheit.

Exercice 9 (Division euclidienne) :
Dans un nouveau fichier TP_0_4.py, écrire un code qui effectue les opérations suivantes :

- Demander à l'utilisateur de saisir deux entiers naturels non nuls.
- Calculer le quotient et le reste de la division euclidienne du premier nombre par le second
- Afficher la division euclidienne sous la forme d'un couple (quotient,reste).

Exercice 10 (Extraction et manipulation d'une chaîne de caractères) :
Dans un nouveau fichier TP_0_5.py, écrire un code qui demande à l'utilisateur de rentrer une date sous la forme 5 septembre 2018 et qui :

- Affiche le mois de l'année
- Donne le nombre de lettre du mois
- Renvoie `True` si le mois est un mois en "bre" et `False` dans le cas contraire.

Le code devra afficher toutes ces informations en affichant la phrase :

> Nous sommes au moins de **`nom_mois`**, ce mot contient **`nb_lettres`** lettres et c'est un mois en bre : `True`/`False`.

3 Pour les plus avancés

Exercice 11 (Décomposition d'entiers (*)) :
Quel est le plus grand entier naturel non nul inférieur à 1000 que l'on ne peut écrire sous la forme $13a+17b+19c$ avec $a, b, c \in \mathbb{N}$?

Exercice 12 (Somme de cubes (*)) :
Écrire une fonction qui renvoie le premier entier naturel inférieur à 10000 qui peut s'écrire de deux façons différentes comme la somme de deux cubes.

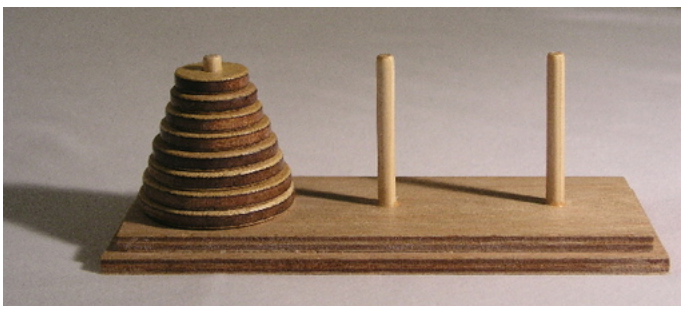
Exercice 13 (Théorème d'Erdòs-Suranyi (***)) :
Le théorème d'Erdòs-Suranyi affirme :

$$\forall n \in \mathbb{N},\ \exists p \in \mathbb{N},\ n = \sum_{k=1}^{p} \varepsilon_k k^2$$

où $\forall k \in \mathbb{N}$, $\varepsilon_k \in \{-1, 1\}$.

Écrire une fonction `Erdos(n)` qui renvoie la liste des entiers qui décompose n avec les signes. Par exemple, `Erdos(6)` devra renvoyer `[[1,1],[-1,2],[1,3]]` car $6 = 1^2 - 2^2 + 3^2$.

Exercice 14 (Tour de Hanoï (***)) :
Le jeu des tour de Hanoï consiste à déplacer une tour de palets de diamètre croissants sur une autre pile en passant par une étape intermédiaire.



On déplace les palets selon les règles suivantes :

- On ne peut déplacer que le palet supérieur d'une pile.
- On ne peut déplacer qu'un palet à la fois.
- On ne peut mettre un palet que sur un palet de diamètre plus grand (ou directement sur le sol).

Écrire une fonction **Hanoi(n)** qui joue à ce jeu avec un jeu de n palets. On affichera toutes les étapes jusqu'à déplacement complet de la tour initiale. On pourra utiliser et modifier la sous-fonction suivante qui permet de décrire les mouvements à faire :

```
def hanoi(Td, Ti, Tf, n):
    L=[]
    if(n > 0):
        # On déplace n-1 palets de Td vers Ti en passant par Tf
        L=L+hanoi(Td, Tf, Ti, n-1)
        # On déplace le dernier palet de Td vers Tf
        L=L+[[Td,Tf]]
        # On déplace n-1 palets de Ti vers Tf en passant par Td
        L=L+hanoi(Ti, Td, Tf, n-1)
    return(L)
```

Le but est d'afficher "graphiquement" toutes les étapes. Par exemple, dans le cas $n = 3$, on doit voir afficher la marche à suivre dont les premières étapes doivent ressembler à :

```
Situation de départ :
.  |    |
.. |    |
...|    |
-----------

Etape numero 1 :
   |    |
.. |    |
...|    |.
-----------

Etape numero 2 :
   |    |
   |    |
...|.. |.
-----------

Etape numero 3 :
   |    |
   |.   |
...|.. |
-----------

Etape numero 4 :
```

Évidemment, on est libre sur les choix esthétiques, tant que la marche à suivre est explicité correctement avec des dessins.