Logiciels Scientifiques

Licence Mathématiques
Licence MIASHS
Mineur Mathématiques

Cours 01

Andrzej Stos

Organisation

- Rappel : Feuilles TP 1 + TP2 = trois séances.
  (finir en mode Devoir Maison si nécessaire)
- Puis 2 séances par feuille (3 séances pour la feuille 8)
- Travail en binôme :
  - En TP, utilisez en alternance les deux comptes
  - À la fin de chaque séance, partagez vos programmes (UCA Drive, email) et sauvegardez-les directement sur le `disque M` de chacun.
  - Pendant les épreuves, l'accès à l'ENT est interdit
  - Seulement un travail régulier en binôme donne le droit de passer le premier CC en binôme avec la même personne
  - Ne comptez pas sur les clés USB dans les salles de TP ! !

Machines virtuelles

Accéder aux serveurs de TP

Portail SCI : https://sciportail.dsi.uca.fr

Deux possibilités :

- Travailler directement dans votre navigateur
- Installer un logiciel ("Client VMWare") et se connecter avec celui-ci.

S'identifier (comme sur l'ENT)

Choisir la machine "Mathématiques" (pour Python / Pyzo)
↪ vous retrouvez votre disque M et l'environnement de salle de TP.

Rappel - Style de programmation !

"A quoi ça sert, les langages de programmation ?"
"A décrire et communiquer ses idées – et les destinataires, ce sont surtout des humains, et non des ordinateurs"

Oui	Non
`x = 10`	`x=10`
`y = x + 1`	`y=x+1`
`y = 2*x + 1`	`y=2*x+1`
`y = 2 * x + 1`	`y=2*x+1`
`x += 1`	`x+=1`
`if n < 10:`	`if n<10 :`
`f(x)`	`f (x)`

\+ du bon sens

Zen of Python : Readability counts. (-1p au contrôle...)

Remarques Pratiques Très Importantes

- Lisez l'énoncé **entier** (et réfléchissez) avant de coder
  - comprendre le contexte
  - prendre du recul par rapport à l'objectif
  - arriver aux **indications** et remarques ! elles sont là pour vous aider...
- **Testez** chaque morceau de code avant d'aller plus loin !
- Utilisez des noms de variables qui **expliquent** leur rôle.
- Une fois votre fonction mise au point et testée, ne la **modifiez jamais**. Pour réutiliser le code, créez et modifiez **une copie**.
- Pour afficher des multiples réponses, mettez une **courte description**. Exemples :

```
print("Question 3:", valeur_max)
print("Message crypté:", msg_crypte)
```

Remarques pratiques : Erreurs de syntaxe

En cas d'erreur :

- ne pas se faire intimider :)
- trouver **le numéro de la ligne** problématique
- lire le message (à la fin du texte en rouge)

Le problème le plus fréquent : SyntaxError : invalid syntax

Astuce à retenir :
La plus souvent ça vient d'un problème de parenthèses
une ligne avant de celle signalée dans le message d'erreur.

Compétence fondamentale

Savoir traduire un énoncé du langage naturel en Python.

Exemple : Suite de Syracuse.

Si un nombre est pair, on le divise par 2. S'il est impair, on le multiplie par 3, on lui ajoute 1 et on divise le résultat par 2. On répète ce procédé jusqu'à ce que $n = 1$. Écrire une fonction `syracuse(n)` qui, étant donné un entier `n`, retourne la longueur de la suite de Syracuse commençant par `n`.

Compétence fondamentale

Si un nombre est pair, on le divise par 2. S'il est impair, on le multiplie par 3, on lui ajoute 1 et on divise le résultat par 2.

```
if n % 2 == 0:
    n = n // 2
else:
    n = (3*n + 1) // 2
```

Compétence fondamentale

Si un nombre est pair, on le divise par 2. S'il est impair, on le multiplie par 3, on lui ajoute 1 et on divise le résultat par 2. On répète ce procédé jusqu'à ce que $n = 1$.

```
while n != 1:

    if n % 2 == 0:
        n = n // 2
    else:
        n = (3*n + 1) // 2
```

Compétence fondamentale

Si un nombre est pair, on le divise par 2. S'il est impair, on le multiplie par 3, on lui ajoute 1 et on divise le résultat par 2.
On répète ce procédé jusqu'à ce que $n = 1$. Écrire une fonction `syracuse(n)` qui, étant donné un entier `n`, retourne la longueur de la suite de Syracuse commençant par `n`.

```
def syracuse(n):
    long = 1
    while n != 1:
        long += 1
        if n % 2 == 0:
            n = n // 2
        else:
            n = (3*n + 1) // 2
    return long
```

Comprendre : variables locales et globales

Variable locale : une variable définie à l'intérieur d'une fonction.
Variable globale : une variable définie en dehors de toutes les fonctions (sans décalage dans un script).

Les variables locales ne sont pas accessibles en dehors de leur fonctions.
Les variables globales sont visibles dans les fonctions – mais non modifiables. Exemple :

```
def mafonc1(n):
    print("Dans mafonc1 x vaut ", x)    # (A)
    y = 15
    return y + n
x = 10
print(mafonc1(15))     # (B)
print("Hors mafonc1, y vaut", y)     # (C)
```

A : ↪ Dans mafonc1 x vaut 10
B : ↪ 30
C : ↪ Erreur !

Comprendre : variables locales et globales

Et si on essaye de modifier une variable globale dans une fonction ?

Autrement dit, si on utilise une variable locale qui porte un même nom qu'une variable globale ?

```
def mafonc2(n):
    print("Dans mafonc2 x vaut d'abord ", x)    # (A)
    x = 25      # modification!
    return x + n
x = 10
print(mafonc2(15))     # (B)
print("Apres mafonc2 x vaut", x)     # (C)
```

A : $\hookrightarrow$ Dans mafonc2 x vaut 10
B : $\hookrightarrow$ 40
C : $\hookrightarrow$ Apres mafonc2 x vaut 10

Récursivité (TP 3)

En mathématique on considère des suites récurrentes. Par exemple

$$x_0 = 0, \qquad x_n = 2x_{n-1} + 1, \quad n > 0.$$

En programmation, cela conduit naturellement à la notion d'une fonction *récursive*. Il s'agit d'une fonction qui fait appel à elle-même.

On ne tourne pas en rond grâce à un argument entier qui décroît à chaque appel. Exemple :

```
def suite(n):
    if n == 0:
        return 0
    return 2 * suite(n-1) + 1
```

Récursivité (TP 3)

Comment ça marche ?

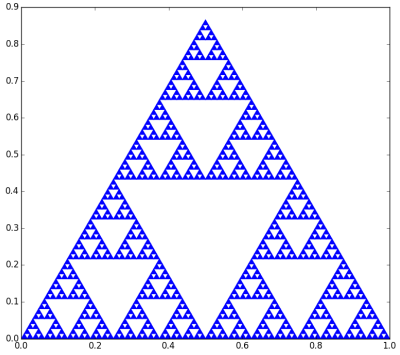
```
def suite(n):
    if n == 0:
        return 0
    return 2 * suite(n-1) + 1

suite(2)
    →suite(1)
        →suite(0)
            return 0          # suite(0) renvoie 0
        return 2 * 0 + 1      # suite(1) renvoie 1
    return 2 * 1 + 1          # suite(2) renvoie 3
```

Remarque : Ceci n'est qu'une version basique. En informatique, beaucoup de structures de données (arbres !) et toute une théorie de calculabilité reposent sur la notion de récursivité... Dans ce cours, on ne parlera que des rudiments.

Fractales (TP 3)

Un triangle de Sierpiński est la figure composée de 3 triangles de Sierpiński.



Ébauche d'une solution

```
TdS(niveau, position, taille) :
    si niveau 0 :
        dessine un triangle simple(position, taille)
    sinon :
        dessine 3 TdS de niveau-1
        chacun avec position ajustée et avec taille/2
```

Question : Comment dessiner un triangle ?

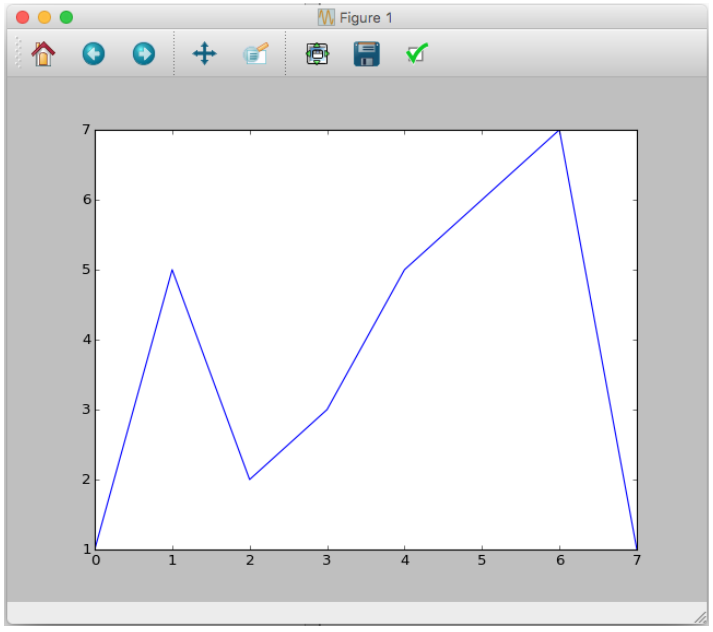
Graphiques

Modules à importer (une fois au début de votre fichier)

```
import numpy as np
import matplotlib.pyplot as plt
```

```
plt.plot([1,5,2,3,5,6,7,1])
plt.show()
```

↪ finalise la construction et provoque l'affichage



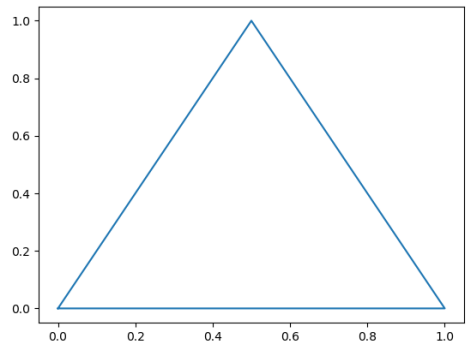
Attention :

la fenêtre graphique peut se cacher derrière celle de Pyzo !

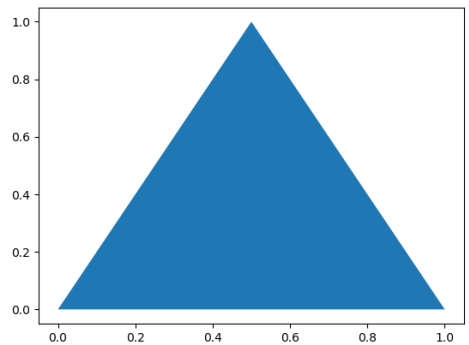
Graphiques

Dessiner une ligne brisée : deux listes, la première contient les abscises, la deuxième les ordonnées des points (sommets).

```
plt.plot([0, 1, 0.5, 0], [0, 0, 1, 0])
```




```
plt.fill([0, 1, 0.5, 0], [0, 0, 1, 0])
```



↪ à utiliser pour le triangle de Sierpiński

Graphiques

Courbe représentative d'une fonction f en mathématiques :

$$\{(x, f(x)) \in \mathbb{R}^2 : x \in D_f\}$$

Courbe représentative de la fonction `np.sin` sur $[0, \pi]$ en Python

```
x = np.linspace(0, np.pi, 500)
```

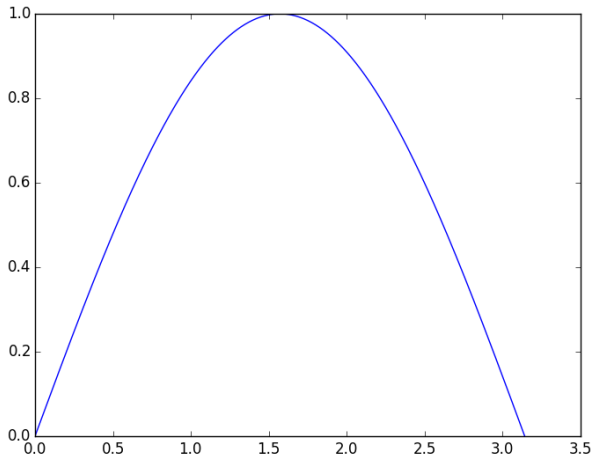
↪ création d'un **tableau** de 500 points entre 0 et π (inclus)

```
plt.plot(x, np.sin(x))
```

↪ application de la fonction `sin` **à tout élément du tableau** `x` (fonctions de `numpy` sont intelligentes !)

```
plt.show()
```

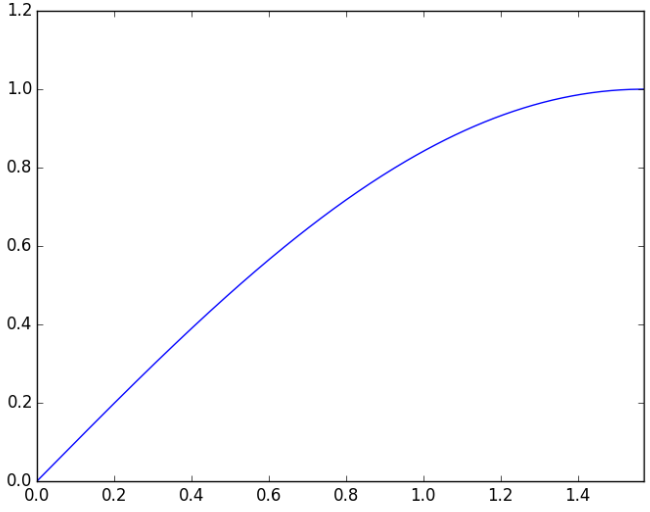
↪ provoque l'affichage



Graphiques

Contrôle manuel des axes (fenêtre d'affichage)

```
x = np.linspace(0, np.pi, 500)
plt.plot(x, np.sin(x))
plt.xlim(0, np.pi/2)
plt.ylim(0, 1.2)
plt.show()
```

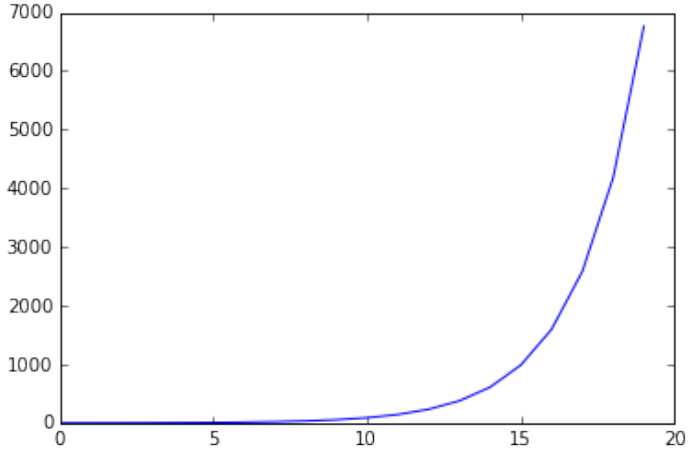


Graphiques : Croissance de suite de Fibonacci

0, 1, 1, 2, 3, 5, 8, 13, 21, ... (cf. TP 2)

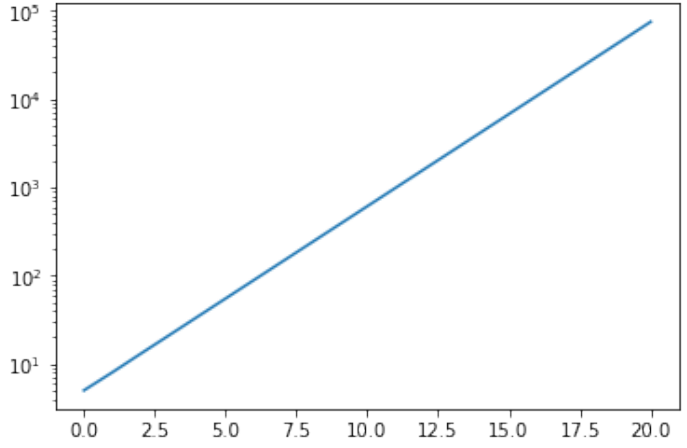
Échelle linéaire :

```
L = fibo_liste(25)
plt.plot(L[5:])
```



Échelle logarithmique :

```
plt.yscale('log')
plt.plot(L[5:])
```



Conclusion : croissance exponentielle !

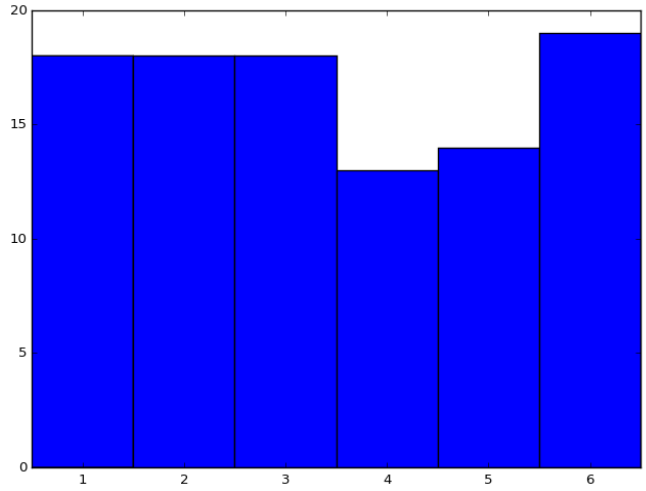
La pente : `L[21] / L[20]` $\hookrightarrow$ `1.61803399852`

Donc $F_n \sim 1.61803399852^n$

Graphiques : Aléas et histogrammes

Simulation de 100 lancers d'un dé

```
import numpy as np
import matplotlib.pyplot as plt
from numpy.random import randint
de = randint(1, 7, 100) # 100 entiers entre 1 et 6
# histogramme avec des boites spécifiques:
plt.hist(de, [0.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5])
plt.show()
```



$\hookrightarrow$ fondamental en analyse de données, probabilités, simulations. (TP6, TP7...)