Université de Msila

Faculté des mathématiques et de l'informatique

Département d'informatique

2ème année Master IDO

Le langage de simulation Python

Objectif : découvrir des éléments de base du langage python au travers de la programmation d'une simulation de propagation de feux de forêt

Partie 1

*** L'environnement ***

Rapide historique

- "Inventeur" :Guido Van Rossumdans les années 90

Rapide historique

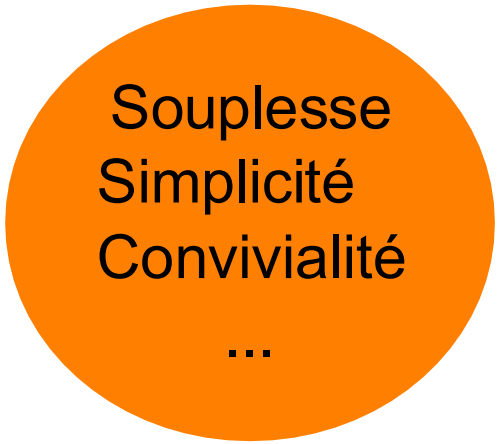
- "Inventeur" :Guido Van Rossumdans les années 90
- Succès du langage parmi des centaines de langages différents (voir Wikipedia)

Rapide historique

- "Inventeur" :Guido Van Rossumdans les années 90
- Succès du langage parmi des centaines de langages différents (voir Wikipedia)

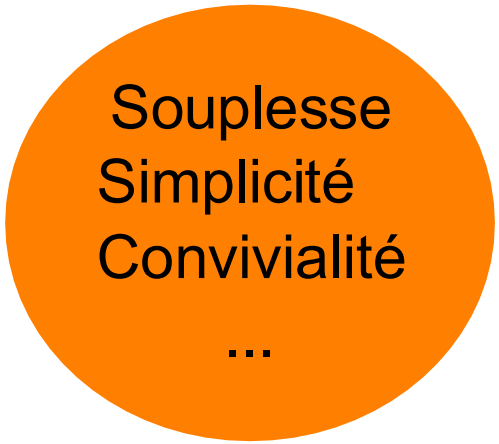
Rapide historique

- "Inventeur" : Guido Van Rossum dans les années 90
- Succès du langage parmi des centaines de langages différents (voir Wikipedia)



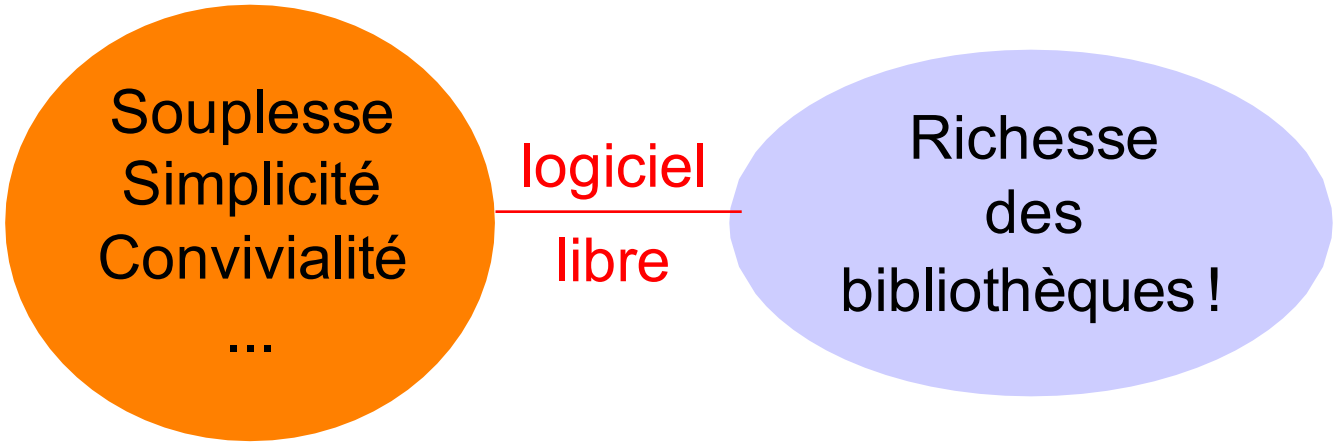
Rapide historique

- "Inventeur" : Guido Van Rossum dans les années 90
- Succès du langage parmi des centaines de langages différents (voir Wikipedia)



historique

- "Inventeur" :Guido Van Rossumdans les années 90
- Succès du langage parmi des centaines de langages différents



Environnement de développement

Environnements de développement
(IDE= Integrated Development Environment).
Parmi les inombrables solutions en voici trois :

Environnement de développement

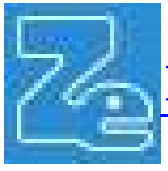
Environnements de développement
(IDE= Integrated Development Environment).
Parmi les inombrables solutions en voici trois :


IDLE : c'est l'original, écrit en python par Guido lui-même

Environnement de développement

Environnements de développement
(IDE= Integrated Development Environment).
Parmi les inombrables solutions en voici trois :


IDLE : c'est l'original, écrit en python par Guido lui-même


PyZo : multi-plateforme, orienté pour faire des sciences...

Environnement de développement

Environnements de développement
(IDE= Integrated Development Environment).
Parmi les inombrables solutions en voici trois :


IDLE : c'est l'original, écrit en python par Guido lui-même


PyZo : multi-plateforme, orienté pour faire des sciences...


EduPython : windows, orienté pour le lycée (math/ISN)...

Le principe

Deux modes de saisie :

- un mode interactif, appelé `shell`...

Le principe

Deux modes de saisie :

- un mode interactif, appelé `shell`...
- une fenêtre de `scripts`...

Le principe

Deux modes de saisie :

- un mode interactif, appelé `shell`...
- une fenêtre de `scripts`...

Le principe

Deux modes de saisie :

- un mode interactif, appelé `shell`...
- une fenêtre de `scripts`...

Remarque :
le langage est autodocumenté (on parle d'**introspection**).
Par exemple, on peut taper `help(print)` dans la fenêtre interactive pour avoir de l'aide sur la fonction `print`...

Partie 2

*** La syntaxe de base ***

Variables, types de base

L'affectation en python se fait grâce au symbole"=".

Langage algorithmique	Traduction Python 3.x
a reçoit 3	a = 3
afficher la valeur de *a*	print("la valeur de a est :",a)

Python connait les type de base :

`bool`, `int`, `float`, `complex`, `str`

qui corresdpondent respectivement à :

booléens, entiers (illimtés), flottants, complexes, chaîne de caractères

Instructions conditionnelles (tests)

Langage algorithmique	**Traduction Python 3.x**
Si *condition* alors *traitement*$_1$ sinon *traitement*$_2$	if *condition* : *traitement*$_1$ else : *traitement*$_2$

Instructions conditionnelles (tests)

Langage algorithmique	Traduction Python 3.x
Si *condition* alors *traitement*$_1$ sinon *traitement*$_2$	if *condition* : *traitement*$_1$ else : *traitement*$_2$

Remarque :
Une seule contrainte en python c'est l'indentation. En effet c'est elle qui détermine la taille des blocs d'insctructions...

Un exemple de code python :

```
import random
n = random.randrange(100)
if n%2 == 0 :
        print(n,'estunnombrepair')
else :
        print(n,'estunnombreimpair')
```

La boucle WHILE

En python la boucle générique est la boucle**while**:

Langage algorithmique	**Traduction Python 3.x**
Tant que *condition* faire *traitement* Fin de tant que	while *condition* : *traitement*

Exemple

On souhaite déterminer après combien de temps une somme de 1000 euros placée à un taux de rémunération annuel de 2,5% aura doublé...

Exemple

On souhaite déterminer après combien de temps une somme de 1000 euros placée à un taux de rémunération annuel de 2,5% aura doublé...

Le code ci-dessous réponds à la question

```
capitl    , annees = 1000 , 0
while capital < 2000 :
    capital *= 1.025
    annees += 1
print('Capital double apres:',annees,'ans')
```

La boucle FOR

Souvent on connait à l'avance le nombre de répétitions que l'on souahite faire. Pour cela, la plupart des langages de programmation fournissent une syntaxe plus concise que le recourt au while...

Souvent on connait à l'avance le nombre de répétitions que l'on souahite faire. Pour cela, la plupart des langages de programmation fournissent une syntaxe plus concise que le recourt au while... c'est la boucle**for**:

Langage algorithmique	**Traduction Python 3.x**
Pour *i* allant de 0 à $n-1$ faire : *traitement*	for i in range(n) : *traitement*

Remarque : la fonction `range(n)` renvoie une sorte de liste (en fait un itérateur) composée des nombres de 0 à n exclu.

Exemple

On souhaite calculer la somme des 5000 premiers entiers...

le code suivant répond à la question

```
s = 0
for n inrange ( 5000 ):
s += n
print ("0+1+2+...+4999=",s)
```

Conteneurs

- En python il existe des objets appelés conteneurs. Parmi eux on trouve les listes.
- Un liste pour python est un collection ordonnée et modifiable d'éléments de types eventuellement hétérogènes !

- C'est un objet très souple et confortable à utiliser.
- La syntaxe : éléments séparé par des virgules et entourés de crochets.
- Voyons par exemples différentes façons de définir la liste des nombres pairs inférieurs ou égaux à 10 :

  - Par extension :

```
pair = [0,2,4,6,8,10]
```

  - Progressive (méthode `append`) :

```
pair=[]
for i inrange (11):
    if i//2 == 0 :
        pair.append(i)
```

Conteneurs

- Par intension (comme en math !) :

```
pair = [ 2*i for i inrange (5)]
```

ou encore

```
pair= [ i for i inrange (11) if i%2==0]
```

Exemple

Pour déterminer la valeur de $1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{999}$

on peut écrire de façon naturelle :

```
s = 0
for n inrange ( 1 , 1000 ): #de1a1000exclu
        s += 1 / n
print(s)
```

ou encore de façon plus "pythonique" :

```
print(sum([1/n for n inrange (1,1000)]))
```

Accès aux éléments d'une liste

- on accède à un élément d'une liste avec son indice (qui commence à 0) :

```
l s t = ['a','b','c','d']
l s t [ 2 ] –>'c'
```

- on enlève et on récupère le dernier élément de la liste avec la méthodepop:

```
l s t = ['a','b','c','d']
l s t . pop ( ) –>'d'
l s t –> ['a','b','c']
```

Les fonctions

la syntaxe usuelle pour définir une fonction est la suivante :

```
def nomFonction(parametres eventuels) :
        traitements
        return valeur(s)
```

On remarquera encore l'indentation pour délimiter le corps de la fonction...

Exemple

Pour obtenir la liste des fréquences de sortie de la face 6 sur 50 echantillons de taille 100

on peut coder ainsi

```
import random
def echant(taille):
    s = 0
    for i inrange (taille):
        if random.randrange(6)+1==6:
        s += 1
    return s/taille

lst = [ echant(100) for i inrange (50)]
```

Partie 3

*** Interface graphique (GUI) ***

Un peu de théorie informatique

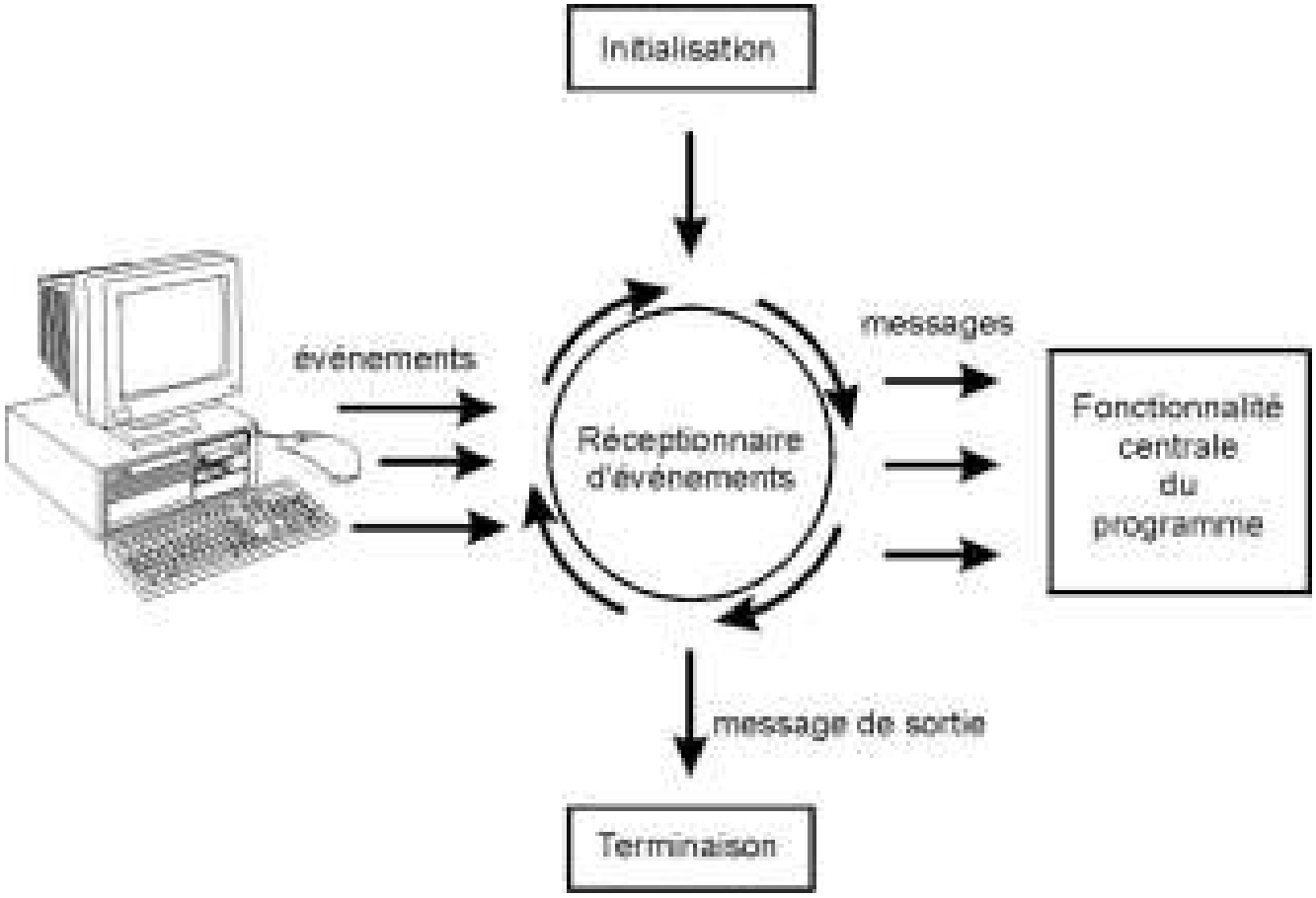
voir [Gérard Swinnen] chp 8 (http://inforef.be/swi/python.htm)

La nature d'un programme utilisant une « interface graphique » diffère fortement de celle d'un programme en « mode texte » :

Schéma du mode texte :

Un peu de théorie informatique

Schéma du mode GUI :



Le choix pour gérer des interfaces graphiques avec python est plétorique !
On utilisera ici celle issue de la bibliothèque standard : `tkinter`.
Une fenêtre graphique est composée d'objets (appelé widgets).
Nous utiliserons ici uniquement deux objets :

- objet Button(...) : des boutons qui réagissent au clic
- objet Canvas(...) : un canevas sur lequel on pourra dessiner

Le choix pour gérer des interfaces graphiques avec python est plétorique !
On utilisera ici celle issue de la bibliothèque standard : `tkinter`.
Une fenêtre graphique est composée d'objets (appelé widgets).
Nous utiliserons ici uniquement deux objets :

- objet Button(...) : des boutons qui réagissent au clic
- objet Canvas(...) : un canevas sur lequel on pourra dessiner

Le choix pour gérer des interfaces graphiques avec python est plétorique !
On utilisera ici celle issue de la bibliothèque standard : `tkinter`.
Une fenêtre graphique est composée d'objets (appelé*widgets*).
Nous utiliserons ici uniquement deux objets :

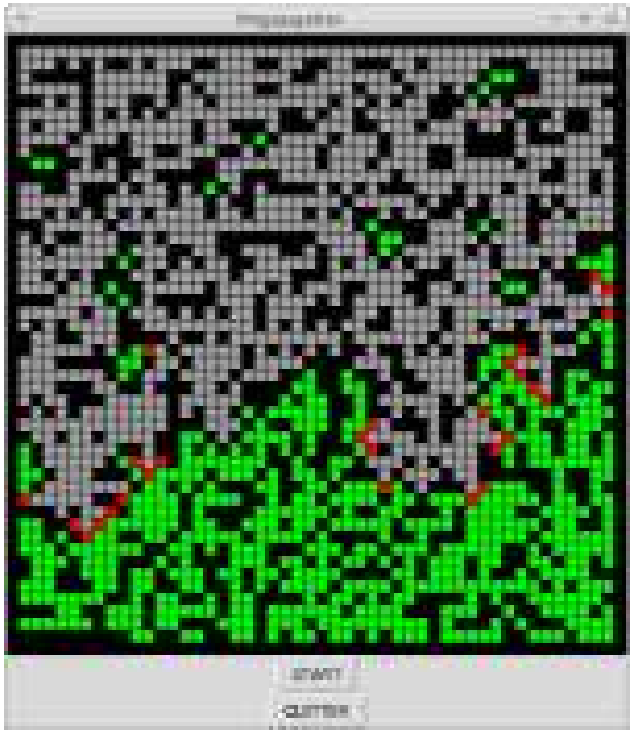
- objet`Button(...)`: des boutons qui réagissent au clic
- objet`Canvas(...)`: un canevas sur lequel on pourra dessiner

Voyons avec PyZo comment construire pas à pas une fenêtre graphique...

Partie 4

*** Exemple de Modélisation ***

L’utilisation des **automates cellulaires** permet de simuler la propagation de feux de forêt.

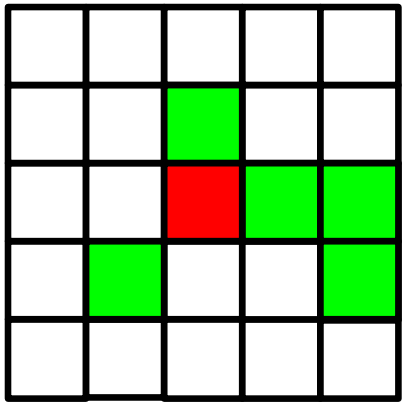


On modélise une forêt par une grande matrice, et chaque case peut se trouver dans l'un des états suivants :

On modélise une forêt par une grande matrice, et chaque case peut se trouver dans l'un des états suivants :

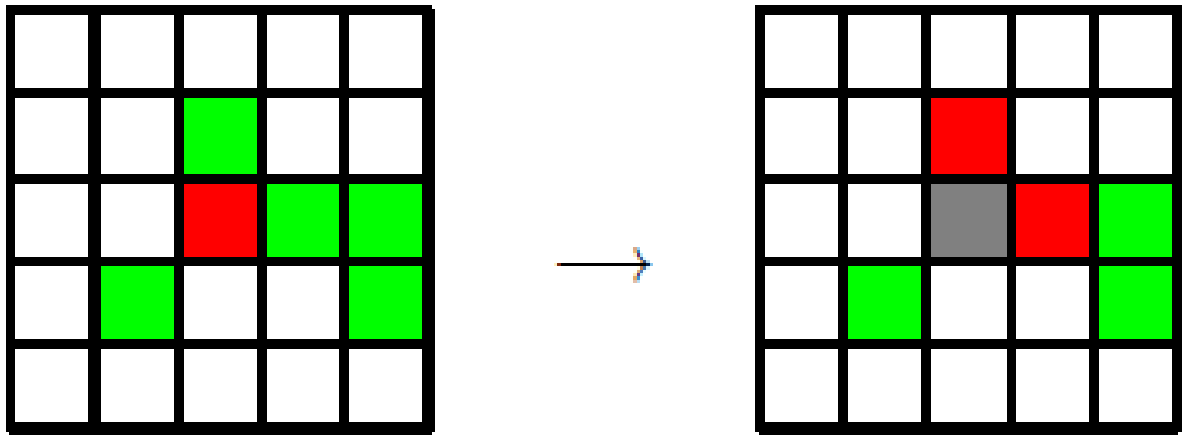
- **état blanc :** zone incombustible (rocher, rivière, ...)
- **état vert :** zone combustible (arbres, végétaux, ...)
- **état rouge :** zone en feu
- **état gris :** zone en cendre (donc incombustible !)

Simulation de feux de forêt



Etape 0

Simulation de feux de forêt



Les règles de propagations

$$
\begin{aligned}
\text{BLANC} &\rightarrow \text{BLANC} \\
\text{GRIS} &\rightarrow \text{GRIS} \\
\text{ROUGE} &\rightarrow \text{GRIS} \\
\text{VERT} &\rightarrow \begin{cases} \text{ROUGE} & \text{si au moins un voisin ROUGE} \\ \text{VERT} & \text{sinon} \end{cases}
\end{aligned}
$$

Un premier jet avec un langage "évolué" pour modéliser notre simulation :

Créer et initialiser un tableau de 50x50
Mettre certaines cases en ROUGE
Afficher le tableau
Appeler la fonction evolution...

Choix 1, on peut envisager le cheminement suivant :

Evolution1 :
S'il y a au moins une case en feu faire :
    Pour chaque case faire :
        Si ROUGE alors NOIR
        Si VERT et "au moins un voisin ROUGE" alors ROUGE
    Afficher le tableau
    Recommencer

Un premier jet avec un langage "évolué" pour modéliser notre simulation :

Créer et initialiser un tableau de 50x50
Mettre certaines cases en ROUGE
Afficher le tableau
Appeler la fonction evolution...

Choix 2, on peut aussi faire le raisonnement suivant :

Evolution2 :
S'il y a au moins une case en feu faire :
    Pour chaque caseen feufaire :
        Mettre la case en GRIS
        Mettre tous les "voisins" VERTS en ROUGE
    Afficher le tableau
    Recommencer

Un premier jet avec un langage "évolué" pour modéliser notre simulation :

Créer et initialiser un tableau de 50x50
Mettre certaines cases en ROUGE
Afficher le tableau
Appeler la fonction evolution...

On retiendra le deuxième choix, car il est moins gourmand en nombres d'opérations (on n'a pas besoin d'explorer toute la grille à chaque étape). De plus il s'avèrera plus pratique pour la deuxième partie du problème !

Passons au code, en précisant les choix "techniques" :

- **Stockage de la grille :** on a besoin d'un tableau à double entrée (une matrice !).
  - On peut simuler cela avec des listes de listes, mais ce n'est pas très pratique.
  - On préfèrera utiliser la biblitohèque scientifique **numpy** qui est prévue pour gérer de façon efficace ces objets.
- **Stockage des cases en feu :** le choix d'une liste qui contiendra les coordonnées des cases en feu est ici bien adapté.

$\Longrightarrow$ voir le code sur PyZo... et le résultat !

Rem 1 : on peu améliorer le résultat visuel par exemple en augmentant le nombre d'états entre le feu et les cendres.
Rem 2 : on peu aussi modifier les règles de propagation pour tenir compte du vent.

Partie 5

*** Analyse de l'effet de seuil ***

Après divers essais avec des densités de végétations différentes, il semble que l'on assiste à deux type de situations :

- Soit la forêt brûle presque entièrement

Après divers essais avec des densités de végétations différentes, il semble que l'on assiste à deux type de situations :

- Soit la forêt brûle presque entièrement
- Soit le feu s'arrête très rapidement

La situation intermédiaire ne se produit pas. L'objectif est alors de visualiser cet effet de seuil puis de trouver la densité pour laquelle s'opère la "bascule"...

Objectif :

On souhaite tracer la représentation graphique de la fonction qui donne la probabilité que la forêt brûle entièrement en fonction de la densité de végétation initiale.

Cette probabilité n'étant pas facile à calculer on se contentera d'une approche statistique en répétant l'expérience sur des échantillons de taille suffisante.

Remarque : dans notre modèle, on considèrera que la fôret a (presque) entièrement brûlée lorsque le feu a traversé l'écran (c'est à dire que la dernière ligne contient au moins une case grise...).

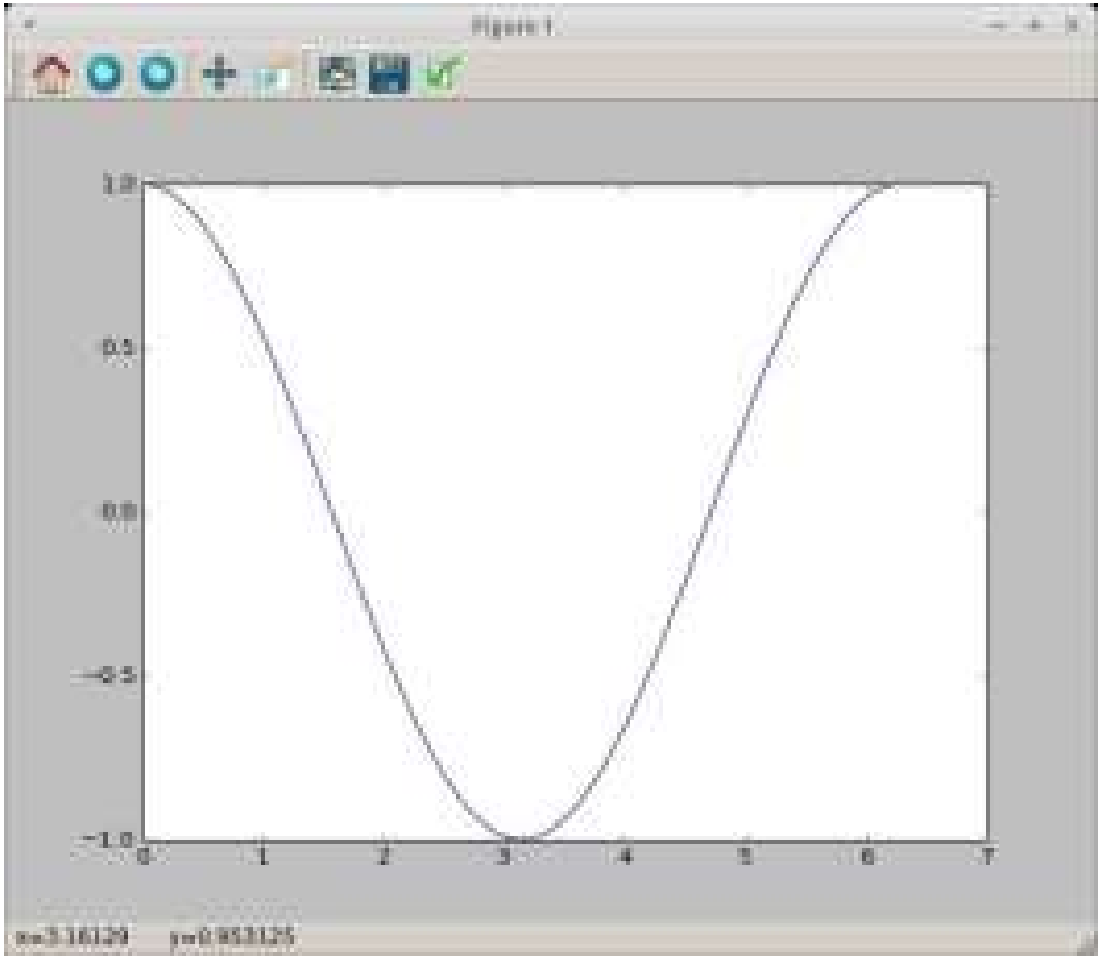
Bibliothèque matplotlib

La bibliothèque matplotlib offre de grandes facilités pour tracer la représentation graphique de données.
Voyons sont fonctionnement sur un exemple :

```
import matplotlib.pyplot as mpl
import numpy as np #discretisation de l'intarvalle[0,2*pi]:
x = np.linspace(0,2*np.pi)
#calcul des images des elements dex:
y = np.cos(x)

mpl.plot(x,y)
mpl.show()
```

On obtient



Mise en evidence du seuil

On modifie alors un peu notre programme :

- la fonction `evolution` renvoie maintenant `True` si le feu a traversé, `False` sinon,
- on crée la fonction `proba` ci-dessous :

```
def proba(t_ech,prop):
    s = 0
    for i inrange (t_ech):
        initTableau(prop)
        if evolution() :
            s += 1
    return s/t_ech
```

- On discrétise l'intervalle de densité [0, 1] on calcule les probas correspondantes et on trace la courbe grâce au module `matplotlib`...

=⇒ voir le code sur PyZo...

cela donne cette représentation (discrétisation avec un pas de 1/100e, et des tailles d'échantillons à 200)