

University of Msila
FACULTY OF MATHEMATICS AND
INFORMATICS
DEPARTMENT OF COMPUTER
SCIENCE

Python :simulation language

Objective: discover the basics of the Python language by programming a simulation of an example

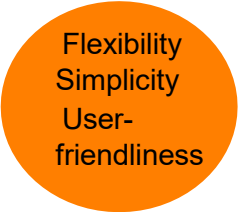
Part 1
***** The environment *****

A brief history

- "Inventor: Guido Van Rossum in the 90s years
Language success among hundreds of different languages

Rapide historique

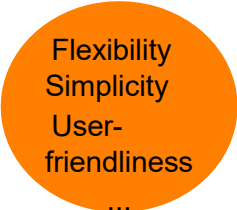
- "Inventor: Guido Van Rossum in the 90s years
- Language success among hundreds of different languages



Flexibility
Simplicity
User-
friendliness

Rapide historique

- "Inventor: Guido Van Rossum in the 90s years
- Language success among hundreds of different languages



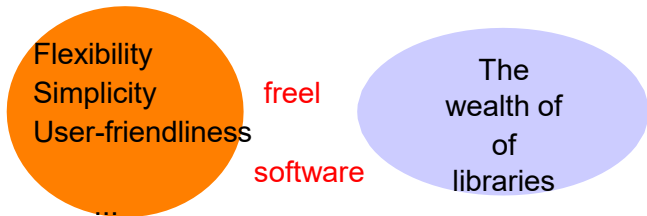
Flexibility
Simplicity
User-
friendliness



The
wealth of
of
libraries

historique

- "Inventor: Guido Van Rossum in the 90s years
- Language success among hundreds of different languages



Development environment

Development environments

(IDE= Integrated Development Environment). Here are three of the countless solutions available:



[IDLE](#) : this is the original, written in Python by Guido himself



[PyZo](#) : multi-platform, science-oriented...



[EduPython](#) : windows, oriented for high school (math/ISN)...

The principle

Two input modes :

- an interactive mode, called shell...
- a script window...

Note:

the language is self-documenting (known as introspection). For example, you can type `help(print)` in the interactive window to get help on the `print...` function.

Part 2
*** Basic syntax ***

Variables, basic types

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

Langage algorithmique	Traduction Python 3.x
<i>a</i> reçoit 3	<code>a = 3</code>
afficher la valeur de <i>a</i>	<code>print("la valeur de a est :",a)</code>

Python connaît les types de base :

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

qui correspondent respectivement à :

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

Instructions conditionnelles (tests)

Langage algorithmique	Traduction Python 3.x
Si <i>condition</i> alors <i>traitement</i> ₁ sinon <i>traitement</i> ₂	if <i>condition</i> : <i>traitement</i> ₁ else : <i>traitement</i> ₂

Instructions conditionnelles (tests)

Langage algorithmique	Traduction Python 3.x
Si <i>condition</i> alors <i>traitement</i> ₁ sinon <i>traitement</i> ₂	if <i>condition</i> : <i>traitement</i> ₁ else : <i>traitement</i> 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, 'est un nombre pair')
else :
    print(n, 'est un nombre impair')
```

La boucle WHILE

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

Langage algorithmique	Traduction Python 3.x
Tant que <i>condition</i> faire <i>traitement</i> Fin de tant que	<i>while condition :</i> <i>traitement</i>

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

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

La boucle FOR

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

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

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

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

Conteneurs

- En python il existe des objets appelés **conteneurs**. Parmi eux on trouve les **listes**.
- Une liste pour python est une collection ordonnée et modifiable d'éléments de types éventuellement hétérogènes!
- C'est un objet très souple et confortable à utiliser.
- La syntaxe : éléments séparés 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 in range(11):  
    if i // 2 == 0:  
        pair.append(i)
```

Conteneurs

- Par intension (comme en math !) :

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

ou encore

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

Exemple

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

on peut écrire de façon naturelle :

```
s = 0
for n in range ( 1 , 1000 ) : #de 1 a 1000 exclu
    s += 1 / n
print (s)
```

ou encore de façon plus "pythonique" :

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

Accès aux éléments d'une liste

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

```
lst = ['a','b','c','d']  
lst[2] -> 'c'
```

- on enlève et on récupère le dernier élément de la liste avec la méthode **pop**:

```
lst = ['a','b','c','d']  
lst.pop() -> 'd'  
lst -> ['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 échantillons de taille 100

on peut coder ainsi

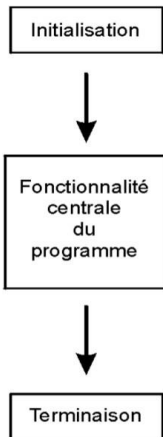
```
import random
def echant ( taille ):
    s = 0
    for i in range ( taille ):
        if random . randrange ( 6 ) + 1 == 6 :
            s += 1
    return s / taille

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

Part 3
*** graphical Interface (GUI) ***

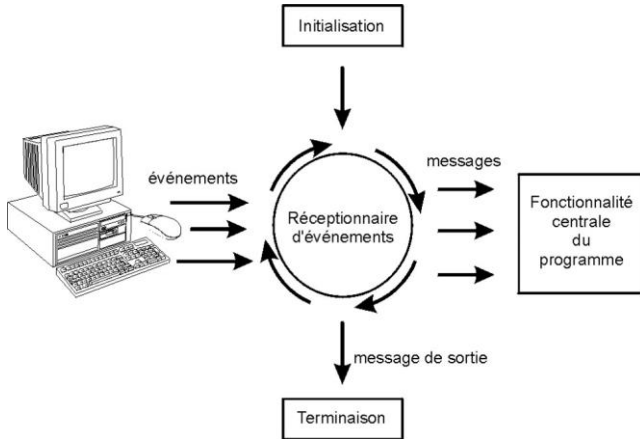
A little computer theory

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



A little computer theory

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**).

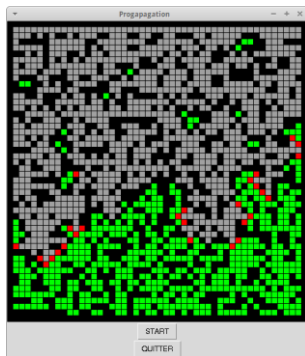
Nous utiliserons ici uniquement deux objets :

- 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...

part 4
*** Example of Modeling***

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

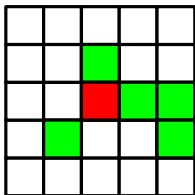


On modélise une forêt par une grande matrice, et chaque case peut se trouver dans l'un des états suivants :

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

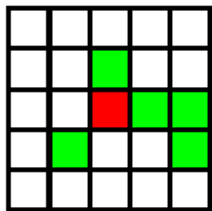
- **état blanc** : zone incombustible (rocher, rivière, ...)
- **état vert** : zone combustible (arbres, végétaux, ...)
- **état rouge** : zone en feu
- **état gris** : zone en cendre (donc incombustible !)

Simulation de feux de forêt

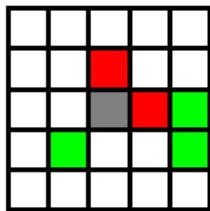


Etape 0

Simulation de feux de forêt



Etape 0



Etape 1

Les règles de propagations

BLANC → BLANC

GRIS → GRIS

ROUGE → GRIS

VERT → $\begin{cases} \text{ROUGE} & \text{si au moins un voisin ROUGE} \\ \text{VERT} & \text{sinon} \end{cases}$

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 case en feu faire :

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 préférera utiliser la bibliothèque 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.

=> voir le code sur PyZo... et le résultat!

Rem 1 : on peut améliorer le résultat visuel par exemple en augmentant le nombre d'états entre le feu et les cendres.

Rem 2 : on peut aussi modifier les règles de propagation pour tenir compte du vent.

Part 5

*** Analysis of the threshold effect ***

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 "basculé"...

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 forêt 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

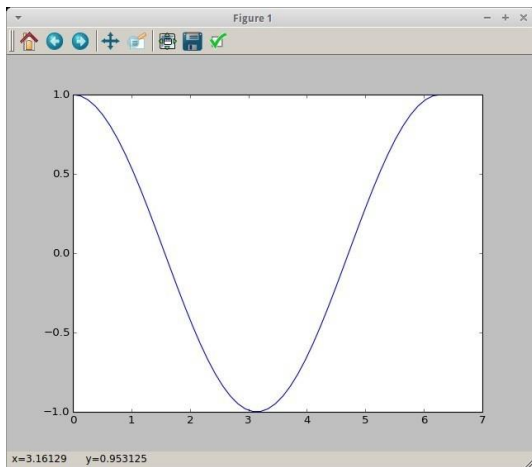
La bibliothèque **matplotlib** offre de grandes facilités pour tracer la représentation graphique de données.

Voyons son fonctionnement sur un exemple :

```
import matplotlib.pyplot as mpl
import numpy as np #discretisation
ndel'intervalle[0,2 *pi]:
x = np.linspace(0, 2*np.pi)
#calcul des images des elements de x:
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 in range (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)

