Algorithmes au lycée : Python ?

Vincent Tolleron

Juin 2014

Table des matières

1 Introduction 3

2 Présentation rapide de python 3

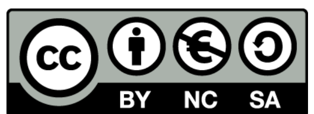
3 Un peu de mauvaise foi pour commencer 3

4 Niveau 1 : prise en main 5

5 Niveau 2 : maîtrisons la bête 9

6 Niveau 3 : soyons fous et allons encore plus loin ! 12

7 Tableaux de synthèse 25



Toutes les images sont la propriété de leurs auteurs respectifs. Cette œuvre est mise à disposition selon les termes de la Licence Creative Commons Attribution - Pas d'utilisation Commerciale - Partage à l'identique 3.0 non transposé.

© 2014 Vincent Tolleron

Vous êtes libre de reproduire, distribuer, communiquer et adapter l'œuvre selon les conditions suivantes :

- Vous n'avez pas le droit d'utiliser cette œuvre à des fins commerciales.
- Si vous modifiez, transformez ou adaptez cette œuvre, vous n'avez le droit de distribuer votre création que sous une licence identique ou similaire à celle-ci.

Ce document a été réalisé avec le système de composition LaTeX.

1 Introduction

Ce document se veut une introduction au langage de programmation *python*. Le propos n'est pas d'être exhaustif (la documentation officielle de Python, quantité d'ouvrages ou de documents sur le net le sont), mais de proposer une première approche — destinée aux enseignants et aux élèves — d'un langage de programmation moderne.

Les sections de ce document vont par niveau de « difficulté » croissante. Dans le cadre d'une utilisation avec les élèves (et notamment d'une alternative à `Algobox`) on peut se contenter du niveau 1.

Le contenu de ces page est en grande partie issu du cours dispensé aux terminales scientifiques suivant la spécialité ISN [1] du lycée Frantz Fanon, à Trinité (Martinique). Des erreurs peuvent subsister dans ce document, le lecteur voudra bien m'en excuser et me les signaler le cas échéant (`vincent.tolleron@me.com`). Je ne suis pas informaticien mais professeur de mathématiques, aussi les « professionnels de la profession » voudront bien me pardonner les quelques approximations que j'ai pu commettre.

2 Présentation rapide de python

La petite histoire

- Date de naissance : 1990
- Père : Guido Van Rossum (Pays-Bas)
- Pourquoi Python ? À cause de la série humoristique britannique *Monty Python*

Caractéristiques de Python [2]

- langage de **haut niveau**
- **portable** sur tous les systèmes d'exploitation
- **typage dynamique** : pas besoin de déclarer le type des variables !
- **extensible** (interfaçable avec d'autres bibliothèques et modules)
- sous licence **libre**
- syntaxe « très simple »
- **multiparadigme** : impératif et orienté-objet
- interprété et/ou pré-compilé puis interprété

3 Un peu de mauvaise foi pour commencer

Comparaison C/Java/Algobox/Python

Les quatre programmes suivants calculent $\sum_{n=1}^{1000} \frac{1}{n^2}$ et affichent le résultat.

1. Informatique et Sciences du Numérique
2. si vous ne comprenez pas tous les mots de ce petit paragraphe, ce n'est pas très grave

En C

```
 # include <stdio.h>
double Invsqr(double n)
{
  return 1/(n*n);
}
int main(int argc, char *argv[]){
  int i, start=1, end=1000;
  for(i=start ; i<= end; i++)
    sum +=Invsqr((double)i);
    printf("%16.14f\",sum);
    return 0;
}
```

En Java

```
 public class Sum{
  public statif double f(double x){
     return 1/(x*x);
  }
  public statif voir main(String[] args){
    double start=1;
    double end=1000;
     sum=0;
    for(double x=start ; x<=end , x++)
      sum+=f(x);
    System.out.println(sum);
  }
}
```

En Algobox

```
1   VARIABLES
2     i EST_DU_TYPE NOMBRE
3     sum EST_DU_TYPE NOMBRE
4   DEBUT_ALGORITHME
5     sum PREND_LA_VALEUR 0
6     POUR i ALLANT_DE 1 A 1000
7       DEBUT_POUR
8       sum PREND_LA_VALEUR sum+1/(i*i)
9       FIN_POUR
10    AFFICHER sum
11  FIN_ALGORITHME
```

En Python

```
 print (sum(1/(x*x) for x in range(1,1001)))
```

4 Niveau 1 : prise en main

4.1 Installation

- On utilisera la dernière version : 3.4.1
- On peut aussi travailler avec la 2.7.7 (quelques différences, mais davantage de modules disponibles).
  - Linux : c'est déjà là !
  - Mac Os : c'est déjà là, mais on peut installer une version plus récente, et surtout l'IDLE (voir plus loin).
  - Windows : faut installer, mais vous devriez en être capable.
- Pour tout cela, une seule adresse : `https://www.python.org/download/`

4.2 Python comme calculette

Python comme calculette : les opérations de base

On lance l'application IDLE (Integrated DeveLopment Environment [3])

Code
```
>>> 15+11
26
>>> 7-3*5
-8
>>> (2-3)*4+2
-2
```

Jusqu'ici, tout va bien . . .

Code
```
>>> 5,2-1
(5,1)
>>> 3,2-1,2
(3,1,2)
```

??????????

Never forget that most of the programming languages speak english !

Le nombre « 5 virgule 2 » s'écrira donc 5.2 en Python (et en C, Java, etc.)

Code
```
>>> 14/3
4.666666666666667
```

Il s'agit de la division décimale. Si on veut le quotient dans la division euclidienne de 14 par 3 :

Code
```
>>> 14//3
4
```

Et si on veut le reste dans cette division ?
Facile ! Il suffit d'utiliser l'opérateur *modulo* : %

3. ou un hommage à Eric Idle, un des co-fondateurs des Monthy Python

Code
```
>>> 14 % 3
2
```
Pour obtenir des puissances :

Code
```
>>> 2**3
8
>>> 1.05**12.5
1.8402051355485856
```
Python manipule les nombres complexes :

Code
```
>>> (1+1j)**2
2j
>>> 1/(1+1j)
(0.5-0.5j)
```

On veut plus de maths !

Pas de panique, tout est prévu, il suffit d'importer au début de la session (ou du programme) le module math :

Code
```
>>> from math import *
```
Toutes les fonctions, constantes, etc. du module math seront alors disponibles :

Code
```
>>> sqrt(4)
2.0
>>> cos(pi)
-1.0
>>> exp(1)
2.718281828459045
```
Attention le logarithme népérien se note log :

Code
```
>>> log(2)
0.6931471805599453
```
Pour le logarithme décimal :

Code
```
>>> log(2,10)
0.30102999566398114
```

4.3 Python pour « faire tourner » des algorithmes

Dans IDLE, on va quitter le mode « shell » (interprété) pour le mode « compilé » :
- Menu File → New Window
- On tape ensuite son programme en entier.
- On sauvegarde.
- On compile (Menu Run → Run Module ou F5)

- Toute ligne de code commençant par # est un commentaire, ignoré lors de l'exécution du programme.
- Un bon programme doit être abondamment commenté !

Un premier exemple qui parle de lui même

Exercice. Calculer et afficher les 20 premiers termes de la suite (u_n) définie pour tout $n \in \mathbb{N}$ par $u_0 = 0$ et $u_{n+1} = \sqrt{1+u_n}$.

Code

```
from math import *
u=0
for i in range(20):
    print(u)
    u=sqrt(1+u)
```

Analyse de l'exemple

- ligne 1 : importation du module math
- ligne 2 : on affecte à une variable u la valeur 0
- ligne 3 : début de la boucle « pour »
- ligne 4 : affichage du terme courant
- ligne 5 : calcul du terme suivant, affectation au terme courant, et fin de la boucle

Premières remarques

- les variables utilisées (ici u et i) ne sont pas à déclarer
- l'affectation d'une valeur à une variable se fait au moyen du symbole =
- range(20) correspond à la liste des entiers naturels de 0 jusqu'à 19 (20 est donc exclu)
- pas de ; à la fin des lignes
- pas de « end » à la fin de la boucle

Le code obtenu est ainsi *plus concis* que dans la plupart des autres langages de programmation.
Parfait, mais ...

- s'il y a une deuxième boucle après ?
- s'il y a une boucle dans une boucle ?
- comment fait python pour s'y retrouver ?

L'indentation

- En Python, les blocs d'instructions ne sont pas délimités par des mots (endIf, enfFor) ni des symboles (; en C), mais par des lignes indentées (décalées) d'un nombre fixe de caractères (4 espaces ou une tabulation en général).
- On peut indenter « à la main » en appuyant 4 fois sur la barre espace, ou une fois sur la touche de tabulation.
- Les bons éditeurs de code feront cela automatiquement.
- Une mauvaise indentation va provoquer des erreurs.

Un petit exercice

Qu'affichera le programme suivant (on ne triche pas ...) ?

Code
```
for i in range(3):
    print(i)
    for j in range(2):
        print(i+j)
```

Un petit exercice

Déterminer le plus petit entier n à partir duquel $1{,}05^n > 100000$

Code solution
```
n=0
while 1.05**n<= 100000:
    n=n+1
print("le plus petit entier cherché est: ",n)
```

- <= signifie $\leqslant$
- dans une boucle `while`, l'incrémentation n'est pas automatique

Un test

Code
```
for a in range(20):
    if a%3 == 0:
        print(a,'est multiple de 3')
    elif a%3==1:
        print(a,'est congru à 1 modulo 3')
    else :
        print(a,'est congru à 2 modulo 3')
```

- un test d'égalité entre deux valeurs se fait au moyen de ==
- on peut afficher plusieurs éléments (nombres, texte) avec la commande print
- elif est la contraction de `else if`

Saisie par l'utilisateur

Voici un programme qui calcule la somme $1+2+\ldots+n$, où n est un nombre entier saisi par l'utilisateur.

Code
```
n=eval(input('Entrez un nombre entier naturel: '))
S=0
for i in range(n+1):
    S=S+i
print(S)
```

- la commande input récupère la saisie de l'utilisateur sous forme d'une chaîne de caractères
- la commande eval évalue la chaîne précédente, et retourne donc ici un entier
- range(n+1) car on veut sommer jusqu'à n inclus

On aurait pu remplacer les 4 dernières lignes par : `sum(range(n+1))`, mais ce n'est pas transposable à d'autres langages informatiques. À éviter donc en présence d'élèves.

Lançons un dé 50 fois

Code
```
from random import *
for lancer in range(50):
    print(" lancer n°",lancer," : ",randint(1,6))
```

Utilisation d'une fonction

Code
```
def f(x):
    return x**3-3*x+1
for x in range(3,11):
    print("f(",x,")=",f(x))
```

- une fonction (python) débute par le mot-clé def et retourne une valeur (tout type possible)
- la commande range(3,11) donne la liste des entiers de 3 à 10 inclus.

4.4 Des exercices !

Des exercices !

Exercice. Écrire un programme qui donne le discriminant et les racines d'un trinôme du second degré, les coefficients étant saisis par l'utilisateur.

Exercice. Soit f la fonction définie pour tout réel x par $f(x) = \begin{cases} -x^2+x+2 & \text{si } x \leqslant 1 \\ \ln(x)+2 & \text{sinon} \end{cases}$. Écrire un programme qui permet donne toutes les valeurs de $f(x)$ pour x allant de -4 à 4 avec un pas de 0,5.

Exercice. Écrire un programme qui permet de déterminer, par dichotomie, la racine cubique d'un nombre a à une précision ε. Les nombres a et ε étant saisis par l'utilisateur.

5 Niveau 2 : maîtrisons la bête

5.1 Les listes

- Structure importante en Python.
- Une liste est une collection ordonnée d'objets (pas forcément de même nature).
- Une liste est délimitée par des crochets.

Code
```
>>> maListe=[1,3,"a",3.1416,"lol"]
>>> maListe[1]
3
>>> maListe[-1]
'lol '
>>> len(maListe)
5
```

- Attention : l'indice du premier élément est 0.
- On peut compter à partir de la fin avec des indices négatifs

Quelques méthodes sur les listes, en vrac :

- Ajout d'un élément en queue de liste :

Code ajout
```
>>> liste1=[2,3,4]
>>> liste1.append(6)
>>> print(liste1)
[2,3,4,6]
```

- Ajout d'un élément en une position quelconque :

Code (insertion)
```
>>> liste1.insert(2,'coucou')
>>> print(liste1)
[2,3,'coucou',4,6]
```

- Recherche d'un élément :

Code recherche
```
>>> liste1.index(4)
3
>>> liste1.index(5)
```

- Tester si un élément appartient à une liste :

Code appartenance
```
>>> 5 in liste1
False
```

- Enlever un élément :

Code (suppression)
```
>>> liste1.remove(4)
>>> print(liste1)
[2,3,'coucou',6]
```

(seule la première occurrence est recherchée ou enlevée)

- Concaténation de listes :

Code concaténation
```
>>> liste1=[2,3,4]
>>> liste2=[4,5,12]
>>> liste3=liste1+liste2
>>> print(liste3)
[2,3,4,4,5,12]
>>> print(liste1*3)
[2,3,4,2,3,4,2,3,4]
```

- Attention à ça :

Code
```
>>> l=[1,2,3]
>>> m=l
>>> l.append(4)
>>> print(l)
[1,2,3,4]
>>> print(m)
[1,2,3,4]
```

Les listes `l` et `m` pointent vers le même objet, modifier l'une c'est aussi modifier l'autre ! Ce n'est pas le cas avec les variables de type nombre ou chaîne. Pour copier une liste vers une autre afin que les

deux objets soient différents :

Code
```
>>> l=[1,2,3]
>>> m=l[:]
>>> l.append(2014)
>>> print(l,m)
[1,2,3,2014] [1,2,3]
```

5.2 Les chaînes de caractères

On les déclare à l'intérieur d'apostrophes simples, doubles ou triples (*simple quotes, double quotes, triple quotes*)

Code
```
>>> a='Ceci est un texte'
>>> print(a)
Ceci est un texte
>>> b="éèàçù§"
>>> print(b)
éèàçù§
```

- python 3 supporte l'encodage utf-8 et donc les caractères accentués

Opérations de base sur les chaînes de caractères

Code
```
>>> maChaine="Je suis heureux d'apprendre Python et je veux progresser"
>>> len(maChaine)
56
```

La fonction `len()` retourne la longueur de la chaine (espaces comprises).

Code
```
>>> maChaine.upper()
"JE SUIS HEUREUX D'APPRENDRE PYTHON ET JE VEUX PROGRESSER"
```

De même, la méthode `.lower()` met toute la chaîne en minuscules.

Code
```
>>> maChaine.count("eu")
3
```

La séquence de lettres eu apparaît trois fois dans la chaîne.

Code
```
>>> maChaine.find("Python")
28
```

La chaine "Python" apparaît à l'indice 28 dans la chaîne (les indices démarrent à zéro).

Code
```
>>> maChaine.replace("Python","les maths")
"Je suis heureux d'apprendre les maths et je veux progresser"
```

Code
```
>>> maChaine[3]
's'
```

On a obtenu le caractère d'indice 3 dans la chaîne (attention, les indices démarrent à zéro).
Pour obtenir la sous-chaîne allant de l'indice 3 (inclus) à l'indice 15 (exclus) :

Code
```
>>> maChaine[3:15]
'suis heureux'
```

Un truc joli :

Code
```
>>> 'Python' in maChaine
True
```

Concaténation de chaînes :

Code
```
>>> tex1='belle marquise '
>>> tex2='vos beaux yeux '
>>> tex3=" me font mourir d'amour "
>>> monTexte=tex1+tex2+tex3
>>> print(monTexte)
belle marquise vos beaux yeux me font mourir d'amour
>>> print(monTexte*4)
```

5.3 Encore des exercices !

Exercice. Écrire un programme qui donne la liste des diviseurs d'un entier naturel saisi par l'utilisateur.

Exercice. Écrire un programme qui recueille des nombres au clavier et en calcule la moyenne, la médiane, l'étendue, les premiers et troisièmes quartiles (au sens du programme de seconde).

Exercice. Écrire un programme qui inverse une chaîne de caractères saisie par l'utilisateur (par exemple *bonjour* devient *ruojnob*).

Exercice. Écrire un programme qui simule le temps d'arrêt nécessaire pour l'obtention du six quand on lance un dé (on utilisera la fonction `randint(1,6)` du module `random`).

Exercice. Écrire un programme qui simule 1000 lancers de deux dés, et dresse la table des fréquences de la somme de ces deux dés.

6 Niveau 3 : soyons fous et allons encore plus loin !

6.1 Affectations multiples, affectations parallèles

Affectations multiples

Il est possible d'assigner une même valeur à plusieurs variables en une seule commande :

Code
```
>>> x=y=z=3
>>> print(x,y,z)
3 3 3
```

Affectations parallèles

Il est possible d'affecter plusieurs valeurs à plusieurs variables en une seule ligne de commande :

Code
```
>>> anne,mois,jour,nomJour=2014,'juin',12,'jeudi'
>>> print(nomJour,jour,mois,annee)
jeudi 12 juin 2014
```

Les affectations parallèles sont bien pratiques pour échanger le contenu de deux variables :

Code
```
>>> u,v=1,3
>>> print(u,v)
1  3
>>> u,v=v,u
>>> print(u,v)
3  1
```

Il n'est donc pas nécessaire, en python, de faire appel à une variable auxiliaire pour échanger le contenu de deux variables.

6.2 Variables et typage

Variables

- Les variables permettent de stocker une information.
- Les variables sont caractérisées par :
  - un identificateur (le « nom » donné à la variable)
  - un type (entier, flottant, etc.)
  - une référence (une adresse dans la mémoire de l'ordinateur)
  - une valeur (le contenu de la case mémoire)

Code
```
>>> annee=2014
>>> Annee="Deux mille quatorze"
>>> type(annee)
<type 'int'>
>>> type(Annee)
<type 'str'>
```

La variable `annee` est de type `int` (c'est-à-dire entier), la variable `Annee` est de type `str` (c'est-à-dire chaîne de caractères).

Code
```
>>> id(annee)
4341319664
```

La variable `annee` occupe l'adresse `4341319664` en mémoire [4].

4. le résultat varie bien sûr selon la machine utilisée

Identificateurs (noms des variables)

- Séquence de lettres (A à Z, a à z), de chiffres (0 à 9), avec éventuellement le caractère _
- Commence par une lettre ou par _
- Autres caractères ($, #, accents, etc) interdits
- Ne doit pas faire partie de la liste des mots réservés (voir plus loin)
- Attention à la **casse** (par exemple `annee`, `Annee` et `ANNEE` sont différents)

Conseil. En général, on choisira un identificateur assez court mais explicite (surtout si le programme est long).

Les mots réservés

and	as	assert	break	class	continue
def	del	elif	else	except	False
finally	for	from	global	if	import
in	is	lambda	None	nonlocal	not
or	pass	print	raise	return	True
try	while	with	yield		

Types

En Python, le typage est **dynamique** : il est inutile de déclarer le type de la variable, la valeur que l'on attribue à la variable suffit à Python pour le définir.

Code
```
>>> x=500
>>> type(x)
<type 'int'>
>>> x=x+0.12
>>> type(x)
<type 'float'>
>>> nomLycee="Frantz Fanon"
<type 'str'>
>>> affixe=1+3j
<type 'complex'>
```

Cela peut perturber les habitués d'autres langages :

Code
```
>>> y=10
>>> type(y)
<type 'int'>
>>> z=10.0
>>> type(z)
<type 'float'>
>>> y==z
True
>>> y is z
False
```

Remarques.

- Bien que de types différents, les variables y et z sont déclarées égales par Python (ou plutôt leurs *valeurs* le sont) ;
- toutefois, les « objets » y et z sont distincts (je sais, c'est subtil) :
  - la commande `y==z` est un **test** qui retourne `True` lorsque les valeurs sont égles, et `False` sinon ;

  - la commande is est un test qui retourne `True` lorsque les « objets » sont identiques.
- On n'est pas obligé de raconter tout ça aux élèves !
- Le typage étant dynamique, on ne retiendra finalement, comme sur Algobox, que les types :
  - nombres (sans rentrer dans les détails) ;
  - chaînes de caractères ;
  - listes.

6.3 Fonctions et récursivité

On a vu plus haut quelques exemples de fonctions, voici quelques éléments supplémentaires à leur propos.
- Ce sont des « bouts » de programme réutilisables, accomplissant une tâche précise
- La syntaxe générale est :

Code
```
def maFonction(paramètres):
    bloc d'instructions
```

- Il peut ne pas y avoir de paramètre ; il peut y avoir des paramètres, prédéfinis ou pas
- Une fonction peut faire appel à elle-même : on parle alors de récursivité
- La définition d'une fonction doit intervenir *avant* son appel

Code (fonction sans paramètre)
```
def table11():
    for i in range(20):
        print(i,"*11 = ",i*11)
```

Code (fonction avec paramètres)
```
def table11bis(n):
    for i in range(n):
        print(i,"*11 = ",i*11)
```

Code (fonction avec paramètres prédéfinis)
```
def table11ter(n=10):
    for i in range(n):
        print(i,"*11 = ",i*11)
```

Tester ensuite ces fonctions en les appelant (dans le même script) :

Code (appel des fonctions précédentes)
```
table11()
table11bis(21)
table11bis(30)
table11ter()
table11ter(30)
```

Exemple de récursivité

La suite de Fibonacci est définie par ses deux premiers termes : 0 et 1. Chaque terme est ensuite la somme des deux précédents. Cette suite commence donc par :

$$0 \quad 1 \quad 1 \quad 2 \quad 3 \quad 5 \quad 8 \quad 13 \quad 21\ldots$$

Code (fonction récursive)

```
def fibonacci(n):
    if n==0:
        return 0
    elif n==1:
        return 1
    else :
        return fibonacci(n-1)+fibonacci(n-2)
```

Documentation d'une fonction

Une chaîne de caractères située juste après la définition d'une fonction permet de la documenter.

Code (exemple de documentation)

```
def maFonction():
    "Cette fonction affiche un joli message"
     print("Hello World !")

maFonction()
print(maFonction.__doc__)
```

6.4 Utilisation de modules

- on peut importer dans un script un ou plusieurs modules
- chaque module contient un certain nombre de fonctions ou d'objets prédéfinis
- pour de longs projets, il sera avantageux de découper son programme en plusieurs modules

On veut par exemple utiliser : $\sqrt{\ }$, cosinus et le nombre π.

Code

```
from math import sqrt, pi, cos
```

Si on veut importer *toutes* les fonctions du module `math` :

Code

```
from math import *
```

On pourra ensuite utiliser ces fonctions dans notre programme :

Code Appel des fonctions

```
print(sqrt(2))
print(cos(pi/3))
```

On peut importer plus simplement, mais l'appel sera différent :

Code

```
import math
print(math.sqrt(2))
```

Il faut faire attention

Voici un exemple qui se passe mal :

Code
```
>>> from math import *
>>> print(e)
2.718281828459045
>>> a=3
>>> c=4
>>> e=a+c
>>> print(e)
7
```
La valeur de la constante e a été perdue !

Une solution :

Code
```
>>> import math
>>> print(math.e)
2.718281828459045
>>> a=3
>>> c=4
>>> e=a+c
>>> print(e)
7
>>> print(math.e)
2.718281828459045
```

Autre solution :

Code
```
>>> from math import e as E
>>> e=3
>>> print(e)
3
>>> print(E)
2.718281828459045
```

Création d'un module

Il est très facile de créer un module :

Code
```
from math import log as ln
from math import factorial as factorielle
from math import *
def comb(n,k):
    return factorielle(n)/(factorelle(k)*factorielle(n-k))
def binomiale(n,p,k):
     comb(n,k)*p**k*(1-p)**(n-k)
```
- On a « francisé » les noms des fonctions logarithme népérien et factorielle (lignes 1 et 2)
- On a importé tout le module math (au cas où, ligne 3)
- On a défini deux fonctions : `comb(n,k)` qui retourne $\binom{n}{k}$ et `binomiale(n,p,k)` qui retourne $p(X = k)$ lorsque $X \hookrightarrow \mathscr{B}(n, p)$.
- On sauvegarde ensuite ce fichier, par exemple sous le nom `monModule.py`

On peut alors utiliser ce module depuis un autre fichier :

Code
```
from monModule import *
a=binomiale(10,0.5,3)
print(a)
b=ln(1)
c=ln(e)
print(b,c)
```
On obtiendra, après exécution :

Code
```
>>>
0.1171875
0.0
1.0
```
On peut ainsi, à l'échelle d'un lycée, d'une académie (cf. Amiens), créer un module spécifique répondant à des besoins particuliers.

Des exemples de modules qu'on pourra employer

- `math` : permet d'utiliser les fonctions et constantes mathématiques
- `random` : génération de nombres de façon aléatoire
- `matplotlib` : graphiques mathématiques
- `time` et `datetime` : gestion du temps, des dates, conversions, etc.
- `turtle` : pour faire des petits dessins en mode « tortue »
- `scipy,numpy` : pour faire du calcul scientifique, des graphiques, etc.

Il existe des centaines de modules, tous ne sont pas installés par défaut !

Amusons nous avec la tortue

Essayez de comprendre ce que fait ce programme :

Code
```
from turtle import *
color('red')
forward(100)
left(90)
color('purple')
forward(300)
left(90)
color('orange')
forward(100)
left(90)
color('green')
forward(300)
left(90)
input()
```

Quelques fonctions utiles du module `turtle`

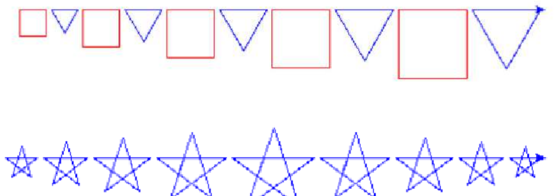
En vrac :

`reset()`	on efface tout et on recommence
`goto(`x, y`)`	la tortue se déplace au point de coordonnées (x, y)
`forward(`d`)`	la tortue avance de la distance d en coloriant sur son passage
`backward(`d`)`	la tortue recule de la distance d en coloriant sur son passage
`up()`	on relève le crayon (par exemple pour se déplacer sans dessiner)
`down()`	on abaisse le crayon (pour recommenced à designer)
`color(maCouleur)`	on choisir la couleur du crayon (`'red'`, etc.)
`left(angle)`	la tortue tourne à gauche d'un angle donné (en degrés)
`right(angle)`	la tortue tourne à droite d'un angle donné (en degrés)
`width(épaisseur)`	on choisit l'épaisseur du crayon (en pixels)
`circle(`r`)`	trace un cercle de rayon r (le centre est situé r pixels à gauche de la tortue)

Exercice. En utilisant le module `turtle` dessiner :

- un carré rouge
- un triangle équilatéral vert
- un triangle rectangle isocèle jaune
- un triangle isocèle bleu
- un cercle violet
- un escalier avec 10 marches
- une étoile à 5 branches dont chaque trait change de couleur
- un polygone régulier à 9 côtés

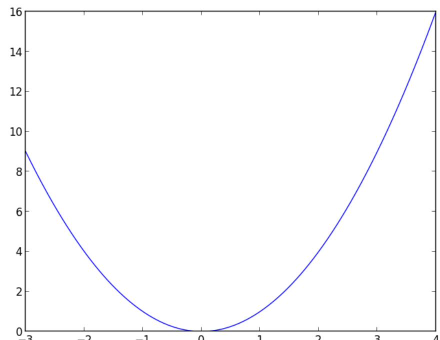
Exercice. Réaliser les figures suivantes :



Amusons nous avec Matplotlib

Code

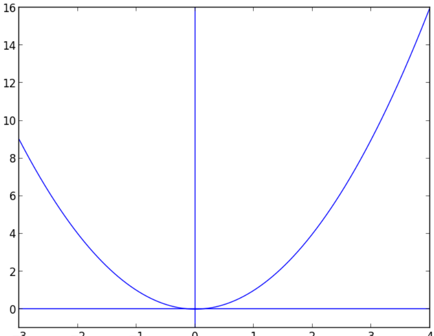
```
from pylab import *
x=arange(-3,4,0.01)
plot(x,x**2)
show()
```



On peut raffiner :

Code

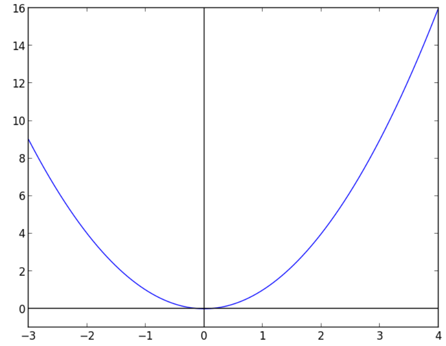
```
from pylab import *
x=arange(-3,4,0.01)
plot(x,x**2)
axis([-3,4,-1,16])
axhline()
axvline()
show()
```

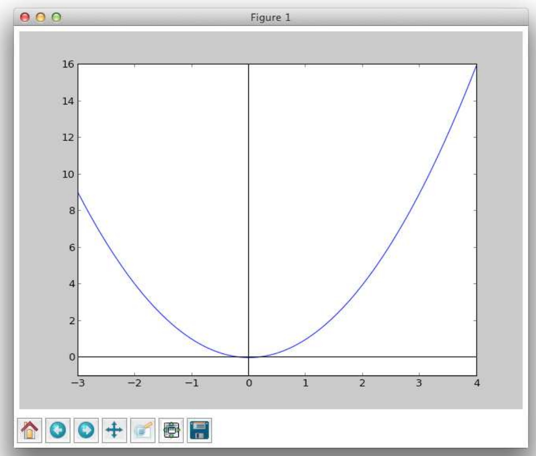


Encore mieux :

Code

```
from pylab import *
x=arange(-3,4,0.01)
plot(x,x**2)
axis([-3,4,-1,16])
axhline(color="k")
axvline(color="k")
show()
```

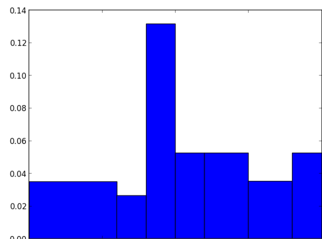




Les boutons servent à naviguer dans la fenêtre graphique, à zoomer, à sauvegarder la figure, etc.
Création d'un histogramme à pas inégaux :

Code

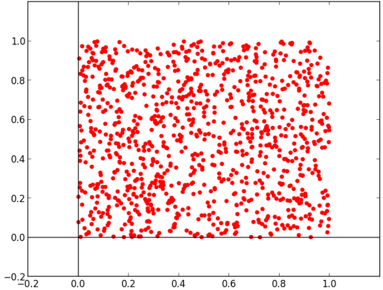
```
from pylab import *
valeurs=[2,5,5,5,7,8,8,9,9,9,10,10, 12,13,13,15,17,18,19]
bornes=[0,6,8,10,12,15,18,20]
hist(valeurs,bornes,normed=True)
show()
```



Création d'un nuage de 1000 points aléatoires dans $[0,1] \times [0,1]$ (avec la loi uniforme) :

Code

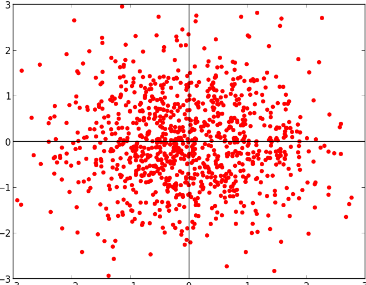
```
from pylab import *
X=np.random.uniform(0,1,1000)
Y=np.random.uniform(0,1,1000)
scatter(X,Y,color='r')
axis([-0.2,1.2,-0.2,1.2])
axhline(color='k')
axvline(color='k')
show()
```



Le même avec la loi normale $\mathcal{N}(0;1)$:

Code

```
from pylab import *
X=np.random.normal(0,1,1000)
Y=np.random.normal(0,1,1000)
scatter(X,Y,color='r')
axis([-0.2,1.2,-0.2,1.2])
axhline(color='k')
axvline(color='k')
show()
```



Exercice

Approximation de π par une méthode de Monte-Carlo

- on choisit des points au hasard (uniformément) dans le carré $[0,1]\times[0,1]$;
- on colorie en bleu ceux qui sont à l'intérieur du quart de disque de centre $(0,0)$ et de rayon 1 ;
- on colorie en rouge ceux qui sont au dessus ;
- pour un grand nombre de points, le quotient $\frac{\text{nb. de points bleus}}{\text{nb. total de points}}$ donne une approximation de $\frac{\pi}{4}$, et donc de π

Les possibilités de `matplotlib` sont innombrables :

- tous type de graphiques statistiques ;
- graphiques en 3D ;
- utilsiation de curseurs, de boutons ;
- etc.

Pour en savoir plus, voir : `http://matplotlib.org`

6.5 Autres types de données : tuples, dictionnaires

Les tuples

- comme des listes, mais non modifiables
- délimités par des parenthèses
- moins « gourmand » en ressources qu'une liste (donc plus rapides)
- permet de « protéger en écriture » certaines données

Code Exemple de tuple

```
joursSemaine=("Lundi","Mardi","Mercredi","Jeudi",
  "Vendredi","Samedi","Dimanche")
print(joursSemaine[3])
```

- les tuples n'étant pas modifiables, les méthodes `append()`, `remove()` ne sont pas utilisables ;
- on peut convertir un tuple en liste et *vice-versa* :

Code Conversion d'une liste en tuple

```
maListe=[1,2,10,'omg']
monTuple=tuple(maListe)
print(maListe,monTuple)
```

Voici l'opération inverse :

Code Conversion d'un tuple en liste

```
t1=(2,50,'Gollum')
l1=list(t1)
print(t1,l1)
```

Les dictionnaires

- Au lieu de repérer un élément par son indice (liste, tuple) on le repère par une clé
- Les dictionnaires sont modifiables
- Ils sont délimités par des accolades, la syntaxe générale d'un dictionnaire est :

{ clé1 : valeur1, clé2 : valeur2, etc.}

Le stock d'une marchande de glaces (en litres) est résumé dans le dictionnaire `stock` :

Code

```
stock={'vanille': 5,'coco':3.5, 'chocolat':3}
print(stock)
```

Testons si la marchande possède de la glace au chocolat, de la glace à la fraise :

Code

```
print('chocolat' in stock)
print('fraise ' in stock)
```

La marchande ajoute 5 litres de glace « passion » à son stock :

Code

```
stock['passion']=5
```

Pour obtenir la liste des parfums (clés) :

Code

```
print(stock.keys())
```

Pour obtenir la liste des quantités (valeurs) :

Code

```
print(stock.values())
```

On peut utiliser un boucle `for` pour parcourir un dictionnaire (au moyen des clés).
Par exemple notre marchande souhaite ajouter 3 litres à chacun de ses parfums en stock.

Code
```
for parfum in stock:
    stock[parfum]+=3

print(stock)
```

Pour enlever des éléments d'un dictionnaire (par exemple la glace passion) :

Code Suppression d'un élément
```
del stock['passion']
print(stock)
```

Pour supprimer **tous les éléments** :

Code On efface tout
```
stock.clear()
print(stock)
```

6.6 Manipulation de fichiers

On peut, avec Python, écrire des données dans un fichier, pour les utiliser par la suite :

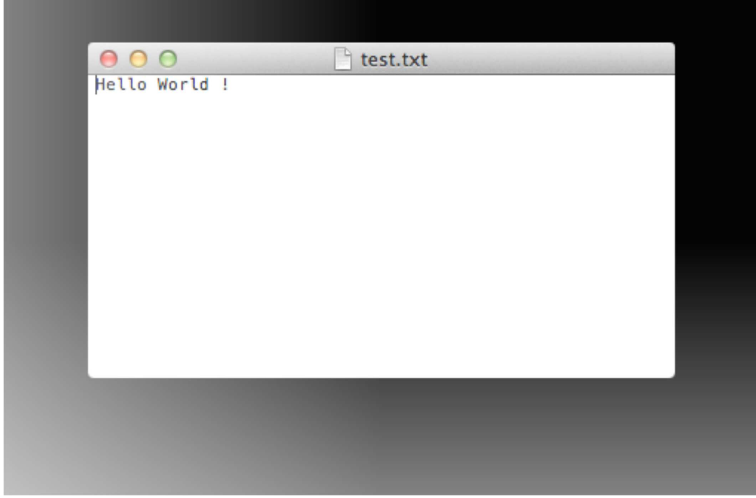
Code ouverture/création d'un fichier
```
monFichier=open('test.txt','w')
```

Explications :
- un objet python (monFichier) est créé ;
- la méthode `open` ouvre en écriture (mode `write`) un fichier texte dans le répertoire courant, nommé `test.txt` ;
- si un fichier portant le même nom existe déjà, il est écrasé.

Le fichier étant ouvert en écriture, on peut maintenant y saisir des données, puis le fermer :

Code écriture et fermeture du fichier
```
monFichier.write('Hello World !')
monFichier.close()
```

Dans le répertoire courant figure maintenant un fichier texte, contenant le texte saisi :



Le fichier étant créé, on peut lui ajouter d'autres textes :

Code Ouverture d'un fichier en mode ajout

```
monFichier=open('test.txt','a')
monFichier.write('\nVoici une deuxième ligne')
monFichier.write('\nEt puis encore une autre\n')
monFichier.close()
```

Le symbole `\n` signifie « passage à la ligne »
On peut également ouvrir en mode lecture (`r`) un fichier pour ... le lire :

Code Ouverture d'un fichier en mode lecture

```
monFichier=open('test.txt','r')
leTexteduFichier=monFichier.read()
print(leTexteduFichier)
```

À retenir ...
Les trois principaux modes d'ouverture d'un fichier, avec la méthode `open` :

`w`	write	(création/écrasement)
`a`	append	(ajout)
`r`	read	(lecture)

7 Tableaux de synthèse

Les types de données Python

Type	Dénomination Python	Exemple
Booléen	`bool`	True False
Entier	`int`	2013
Entier long	`long`	2**31
Flottant	`float`	3.14159
Complexe	`complex`	2+3j
Chaînes de caractères	`string`	"bonjour"
Listes (ou tableaux)	`list`	[2,3,'a',-2.4]
tuples	`tuple`	(2,3,'b')
dictionnaires	`dict`	{ "pointsVie" :300,"nom" :"zorglub"}

On peut toutefois se passer très largement au niveau lycée des types *tuples* et *dictionnaires*.

Opérations de base sur les nombres

Soit x et y des nombres.

Opération	Syntaxe
Addition, soustraction, multiplication	`x+y   x-y   x*y`
Quotient entier	`x/y`
Reste	`x % y`
Valeur absolue	`abs(x)`
Puissance	`x**y` ou `pow(x,y)`
Conjugué	`x.conjugate()`

D'autres opérations/fonctions sont disponibles, notamment via l'utilisation du module `math`.

Opérations de base sur les chaînes de caractères

Soit `str`, `str1`, `str2` des chaînes de caractères.

Fonction ou méthode	Description
`len(str)`	nombre de caractères (espaces comprises) dans la chaîne `str`
`str.upper()`	met toute la chaine `str` en majuscules
`str.lower()`	met toute la chaine `str` en minuscules
`str.count(str1)`	compte les occurrences de `str1` dans `str`
`str.find(str1)`	retourne l'indice où `str1` apparaît pour la première fois dans `str`
`str.replace(str1,str2)`	remplace dans `str` toutes les sous-chaînes `str1` par `str2`
`str[i]`	retourne le caractère d'indice `i` dans la chaîne `str`
`str[i:j]`	extrait de la chaîne `str` allant de l'indice `i` (inclus) à l'indice `j` (exclus)
`str1+str2`	concatène les chaînes `str1` et `str2`
`str1in str`	retourne `True` si `str1` est incluse dans `str`, et `False` sinon

Opérations de base sur les listes

Soit `liste` une liste, k un entier naturel et `ob` un objet quelconque (nombre, chaîne ou liste)

Fonction ou méthode	Description
`len(liste)`	nombre d'objets constituant la liste `liste`
`liste.count(ob)`	compte les occurrences de l'objet `ob` dans `liste`
`liste.append(ob)`	ajoute l'objet `ob` à la fin de `liste`
`liste.insert(i,ob)`	insère l'objet `ob` dans `liste` à l'indice `i`
`liste.remove(ob)`	enlève la première occurrence de `ob` dans `liste`
`del liste[k]`	efface l'élément d'indice `k` de `liste`
`liste1+liste2`	concatène les listes `liste1` et `liste2`
`obin liste`	retourne `True` si `ob` est incluse dans `liste`, et `False` sinon

Opérations de base sur les dictionnaires

Soit `dico` un dictionnaire, `k` une clé, `x` une valeur

Fonction ou méthode	Description
`len(dico)`	nombre de couples clé/valeur dans le dictionnaire `dico`
`dico.keys()`	retourne la liste des clés du dictionnaire `dico`
`dico.values()`	retourne la liste des valeurs du dictionnaire `dico`
`del dico[k]`	efface l'élément clé/valeur référencé par la la clé `k`
`dico.clear()`	efface tout le contenu du dictionnaire `dico`

Opérateurs de comparaison

Opérateur	Signification
`x == y`	Est-ce que x est égal à y ?
`x != y`	Est-ce que x est différent de y ?
`x > y`	Est-ce que x est strictement supérieur à y ?
`x < y`	Est-ce que x est strictement inférieur à y ?
`x >= y`	Est-ce que x est supérieur ou égal à y ?
`x <= y`	Est-ce que x est inférieur ou égal à y ?
`x is y`	Est-ce que x et y représentent le même objet ?

Opérateurs logiques

Opérateur	Signification
`x or y`	retourne `True` ssi l'un des deux booléens (au moins) est vrai
`x and y`	retourne `True` ssi les des deux booléens sont vrais
`not x`	retourne la valeur booléenne contraire de `x`

Zen of python

import this

1. Préfèrer le beau au laid,
2. l'explicite à l'implicite,
3. le simple au complexe,
4. le complexe au compliqué,
5. le déroulé à l'imbriqué,
6. l'aéré au compact.
7. La lisibilité compte.
8. Les cas particuliers ne le sont jamais assez pour violer les règles,
9. même s'il faut privilégier l'aspect pratique à la pureté.
10. Ne jamais passer les erreurs sous silence,
11. ou les faire taire explicitement.
12. Face à l'ambiguïté, ne pas se laisser tenter à deviner.

13. Il doit y avoir une – et si possible une seule – façon évidente de procéder,
14. même si cette façon n'est pas évidente à première vue, à moins d'être Hollandais.
15. Mieux vaut maintenant que jamais,
16. même si jamais est souvent mieux qu'immédiatement.
17. Si l'implémentation s'explique difficilement, c'est une mauvaise idée.
18. Si l'implémentation s'explique facilement, c'est peut-être une bonne idée.
19. Les espaces de nommage sont une sacrée bonne idée, utilisons-les plus souvent !