Formation à Python pour les Sciences

Release 0.3

December 15, 2012

CONTENTS

Contenu

Matériel pédagogique sur l'écosystème Python. Une rapide introduction sur les outils et techniques clés. Chaque chapitre correspond à un cours de 1 à 2 heures avec un degré croissant de complexité.

Emmanuelle Gouillard

Pierre Haessig

Bernard Uguen

CHAPTER

ONE

FORMATIONPYTHON

Préparation d'une formation à Python et Python Scientifique

make html pour construire de document html - index.html dans _build/html/index.html make latexpdf pour construire le pdf

- le répertoire *discussions* est destiné à échanger sur le contenu, la forme, le fond, le plan
- le répertoire *contenu* est destiné à recevoir le contenu des séances de formation
- le répertoire *document* contient les ressources pdf, en particulier le document en cour d'élaboration *PythonScientifique.pdf*
- le répertoire *notebook* contient les notebooks associés à la formation

CHAPTER

TWO

LICENSE

CHAPTER

THREE

PREMIÈRE JOURNÉE : LE LANGAGE PYTHON

3.1 Séance 1 : Un apercu des possibles utilisations de Python

- Objectif de la formation
- **Présentation générale du monde Python**
  - Historique
  - Le tao de python
  - Les différentes communautés Python
  - Modules et espace de nom
  - **Les bibliothèques - modules**
    - numpy scipy matplotlib
    - scikit-image scikits-learn
    - networkx, shapely
    - simpy, SymPy
- **les outils de travail avec Python**
  - éditeur de texte + IPython (commandes intéractives + *%run*)
  - environnement de développement intégré (ex. Spyder)
  - Notebook IPython (proche de Maple/Mathematica)

3.2 Séance 2 : Éléments de base du langage

- **structures de données**
  - (),[],{}
  - listes en compréhension
- **éléments du langage** boucles, conditions, fonctions, itérateur, map , enumerate
- Exemple en algorithmique de base

```
In [1]: def tri1():
```

3.3 Séance 3 : Chaînes et fichiers

- **traitement des chaines de caractères**
  - s.replace()
  - s1 + s2
- un exemple de regexp simple
- **type de fichiers**
  - mode d'accès
- glob.glob
- Sans doute ces points peuvent être intégrés dans la séance 2.

3.4 Séance 4 : Python Objet & Bases de données

- Classe
- Méthodes
- Surcharge d'opérateurs
- Construire un exemple de classe progressivement

  Idéalement un exemple avec l'utilisation d'une base de données MySQL utiliser *pymysql*

CHAPTER

FOUR

DEUXIÈME JOURNÉE : PYTHON SCIENTIFIQUE

4.1 Séance 5 : Calcul numérique avec Numpy

- Lecture fichiers (type structuré)
- Algèbre de base
- broadcasting
- stacking(hstack,vstack,dstack)
- boucle ou pas boucle einsum vs numba comme exemple

4.2 Séance 6 : Graphiques avec Matplotlib

- visite de la grande galerie
- construction d'un graphe simple en 2d en ajoutant des éléments graduellement pour enrichir le graphe (légendes, titre, ...)
- imshow , contouplot
- 3D ?

4.3 Séance 7 : Calcul scientifique avec Scipy

- optimisation
- intégration, ode
- stats

4.4 Séance 8 : Cas pratique de mise en oeuvre

1. Récupérer des données physiques ouvertes sur le réseau (T°, ...)
2. Appliquer un traitement

3. Mettre en forme une représentation graphique des données

Informatique Voies : TSI, MP, PC, PT, TPC, PSI

CHAPTER

FIVE

OBJECTIFS DE FORMATION

5.1 Généralités

L'informatique, omniprésente dans les différentes sphères de l'entreprise, de la recherche, des services, de la culture et des loisirs, repose sur des mécanismes fondamentaux devant être maîtrisés par les futurs ingénieurs, enseignants et chercheurs qui auront à s'en servir pour agir en connaissance de cause dans leur vie professionnelle. La rapide évolution des outils informatiques et des sciences du numérique dans tous les secteurs de l'ingénierie (industrielle, logicielle et des services) et de la recherche rend indispensable un enseignement de l'informatique spécifiquement conçu pour l'étudiant de CPGE scientifiques. Celui-ci devra pouvoir dans sa vie professionnelle communiquer avec les informaticiens de son entreprise ou de son laboratoire, participer aux prises de décision en matière de systèmes d'information, posséder des connaissances de base nécessaires à la compréhension des défaillances et des risques informatiques, ainsi que des solutions permettant d'y remédier, et exploiter à bon escient les résultats de calculs numériques. Pour ce faire, il devra comprendre des concepts tels que la précision numérique, la faisabilité, l'efficacité, la qualité et les limites de solutions informatiques, ce qui requiert une certaine familiarité avec les architectures matérielles et logicielles, les systèmes d'exploitation, le stockage des données et les réseaux. Cette diversité d'exigences impose une formation à la fois fondamentale et appliquée. Au niveau fondamental, on se fixe pour objectif la maîtrise d'un certain nombre de concepts de base, et avant tout, la conception rigoureuse d'algorithmes et le choix de représentations appropriées des données.

Note: Ceci impose une expérience pratique de la programmation et de la manipulation informatique de données, notamment d'origine expérimentale ou industrielle, et parfois disponibles en ligne.

Au niveau des applications, la rapidité d'évolution des technologies logicielles et matérielles renforce l'intérêt de présenter des concepts fondamentaux pérennes sans s'attacher outre mesure à la description de technologies, protocoles ou normes actuels. En revanche, la formation s'attachera à contextualiser le plus souvent possible les activités pratiques en s'appuyant sur les autres disciplines scientifiques : chimie, physique, mathématiques, sciences technologiques et de l'ingénieur.

5.2 Compétences visées

Cet enseignement doit permettre de développer les compétences suivantes :

- Analyser et modéliser un problème, une situation ;
- Imaginer et concevoir une solution algorithmique modulaire, utilisant des méthodes de programmation, des structures de données appropriées pour le problème étudié ;
- Traduire un algorithme dans un langage de programmation moderne et généraliste ;
- Spécifier rigoureusement les modules ou fonctions ; Évaluer, contrôler, valider

des algorithmes et des programmes ;

- Communiquer à l'écrit ou à l'oral, une problématique, une solution ou un algorithme,

une documentation.

L'étude et la maîtrise de quelques algorithmes fondamentaux, l'utilisation de structures de données adaptées et l'apprentissage de la syntaxe du langage de programmation choisi permettent de développer des méthodes (ou paradigmes) de programmation appropriés, fiables et efficaces : programmation impérative, approche descendante, programmation structurée, utilisation de bibliothèques logicielles, notions élémentaires de complexité en temps ou en mémoire, documentation des programmes en vue de leur réutilisation et possibles modifications ultérieures.

La pratique régulière de la résolution de problèmes par une approche algorithmique et des activités de programmation qui en résultent constitue un aspect essentiel de l'apprentissage de l'informatique. Il est éminemment souhaitable que les exemples choisis ainsi que certains exercices d'application soient directement inspirés par les enseignements de physique et chimie, de mathématiques, et de sciences industrielles et de l'ingénieur.

Note: Enfin, les compétences acquises en informatique ont vocation à participer pleinement à l'élaboration des travaux d'initiative personnelle encadrée (T.I.P.E.) et à être réutilisées au sein des autres enseignements scientifiques.

CHAPTER

SIX

PROGRAMME

6.1 Introduction

Première partie du semestre 1.

6.1.1 Présentation du système informatique utilisé et éléments d'architecture des ordinateurs

Une ou deux séances introductives seront consacrées à présenter et à familiariser les étudiants

- aux principaux composants d'une machine numérique telle que l'ordinateur personnel,

une tablette, etc : sources d'énergie, mémoire vive, mémoire de masse, unité centrale, périphériques d'entrée-sortie, ports de communication avec d'autres composants numériques (aucune connaissance particulière des composants cités n'est cependant exigible), à la manipulation d'un système d'exploitation (gestion des ressources, essentiellement : organisation des fichiers, arborescence, droits d'accès, de modification, entrées/sorties), à la manipulation d'un environnement de développement.

La principale capacité développée dans cette partie de la formation est : manipuler en mode « utilisateur » les principales fonctions d'un système d'exploitation et d'un environnement de développement.

6.1.2 Représentation des nombres et conséquences

Il s'agit de familiariser les étudiants avec les problèmes liés à la représentation finie des nombres et à la discrétisation des modèles numériques. Les calculatrices peuvent servir de support d'étude de ces questions.

- Principe de la représentation des nombres entiers en mémoire. On introduit ou rappelle brièvement le principe de la représentation binaire ainsi que ses limites.
- Principe de la représentation des nombres réels en mémoire.

On se limite à la définition de l'écriture en virgule flottante normalisée et on explique le codage d'un nombre réel en les infinis général sans entrer dans les cas particuliers comme les nonnombres « not a number " et les infinis

Conséquences de la représentation limitée des nombres réels en machine.

On illustre, sur des exemples simples, pouvant être illustrés au moyen d'une calculatrice, les phénomènes de dépassement de capacité (ou « overflow ») de séquences de calculs conduisant à des résultats faux et erreurs d'arrondis. On illustre aussi le problème de la comparaison à zéro, par exemple dans une équation du second degré.

Les principales capacités développées dans cette partie de la formation sont :

- appréhender les limitations intrinsèques à la manipulation informatique des nombres,

- initier un sens critique au sujet de la qualité et de la précision des résultats de calculs numériques sur ordinateur.

CHAPTER

SEVEN

ALGORITHMIQUE ET PROGRAMMATION

Seconde partie du semestre 1.

7.1 Outils employés

Au premier semestre, l'enseignement se fonde sur un environnement de programmation (langage et bibliothèques) basé sur un langage interprété largement répandu et à source libre.

Note: Au moment de la conception de ce programme, l'environnement sélectionné est **Python**.

Les travaux pratiques conduiront à éditer et manipuler fréquemment des codes sources et des fichiers ; c'est pourquoi un environnement de développement efficace doit être choisi et utilisé.

Les étudiants doivent être familiarisés avec les tâches de création d'un fichier source, d'édition d'un programme, de gestion des fichiers, d'exécution et d'arrêt forcé d'un programme.

Avant la fin du premier trimestre, un environnement de calcul scientifique est présenté et utilisé en lien avec l'étude des problèmes de simulation. Afin d'en permettre rapidement une utilisation dans d'autres enseignements, une séance de présentation de cet environnement sera prévue en fin de premier trimestre.

Note: **Au moment de la conception de ce programme**, l'environnement sélectionné est Scilab.

L'étude approfondie de ces divers outils et environnements n'est pas une fin en soi et n'est pas un attendu du programme.

Des textes réglementaires ultérieurs pourront mettre à jour ces choix d'outils et d'environnements en fonction des évolutions et des besoins.

7.2 Algorithmique

Les compétences en matière d'algorithmique et de programmation étant profondément liées, il est souhaitable que ces deux sujets soient abordés de concert, même si pour des raisons de clarté d'exposition ils sont ici séparés.

L'introduction à l'algorithmique contribue à apprendre à l'étudiant à analyser, à spécifier et à modéliser de manière rigoureuse une situation ou un problème. Cette démarche algorithmique procède par décomposition en sous-problèmes et par affinements successifs. L'accent étant porté sur le développement raisonné d'algorithmes, leur implantation

dans un langage de programmation n'intervient qu'après une présentation organisée de la solution algorithmique, indépendante du langage choisi.

Les invariants de boucles sont introduits pour s'assurer de la correction des segments itératifs. Une attention particulière doit être apportée au choix de structures de données appropriées.

La notion de complexité d'algorithmes (en distinguant la complexité en mémoire, la complexité en temps dans le meilleur et dans le pire des cas) est introduite sur des exemples simples.

Pour faire mieux comprendre la notion d'algorithme et sa portée universelle, on s'appuie sur un petit nombre d'algorithmes simples, classiques et d'usage universel, que les étudiants doivent savoir expliquer et programmer, voire modifier selon les besoins et contraintes des problèmes étudiés.

Contenus	Précisions et commentaires
Recherche dans une liste, recherche du maximum dans une liste de nombres, calcul de la moyenne et de la variance.	
Recherche d'un mot dans une chaîne de caractères.	On se limite ici à l'algorithme "naïf", en estimant sa complexité.

On veillera à illustrer les concepts étudiés en les contextualisant au moyen d'exemples et de données notamment expérimentales issus des différentes disciplines scientifiques et technologiques.

Les principales capacités développées dans cette partie de la formation sont :

- comprendre un algorithme et expliquer ce qu'il fait,
- modifier un algorithme existant pour obtenir un résultat différent,
- concevoir un algorithme répondant à un problème précisément posé,
- expliquer le fonctionnement d'un algorithme,
- écrire des instructions conditionnelles avec alternatives, éventuellement imbriquées,
- justifier qu'une itération (ou boucle) produit l'effet attendu au moyen d'un invariant,
- démontrer qu'une boucle se termine effectivement,
- s'interroger sur l'efficacité algorithmique temporelle d'un algorithme.

Les étudiants devront être capables de programmer dans le langage de programmation indiqué ci-dessus les différents algorithmes étudiés.

7.3 Programmation

On insistera sur une organisation modulaire des programmes ainsi que sur la nécessité d'une programmation structurée et parfaitement documentée.

- **Variables** : notion de type et de valeur d'une variable, types simples.

  Les types simples présentés sont les entiers, flottants, booléens et chaînes de caractères.

- **Expressions et instructions simples** :

  affectation, opérateurs usuels, distinction entre expression et instruction Les expressions considérées sont à valeurs numériques, booléennes ou de type chaîne de caractères.

- **Instructions conditionnelles**:

expressions booléennes et opérateurs logiques simples, structurer et comprendre plusieurs instruction if. Variantes avec alternative (else).

Les étudiants devront être capables de niveaux d'alternatives implantées par des instructions conditionnelles imbriquées.

- **Instructions itératives** : boucles for, boucles conditionnelles while.

Les sorties de boucle (instruction break) peuvent être présentées et se justifient uniquement lorsqu'elles contribuent à simplifier notablement la programmation sans réelle perte de lisibilité des conditions d'arrêt.

- **Fonctions** : notion de fonction (au sens informatique), définition dans le langage utilisé, paramètres (ou arguments) et résultats, portée des variables. On distingue les variables locales des variables globales et on décourage l'utilisation des variables globales autant que possible.

  La récursivité sera présentée plus tard.

- **Manipulation de quelques structures de données**:

chaînes de caractères (création, accès à un caractère, concaténation), listes (création, ajout d'un élément, suppression d'un élément, accès à un élément, extraction d'une partie de liste), tableaux à une ou plusieurs dimensions.

On met en évidence le fait que certaines opérations d'apparence simple cachent un important travail pour le processeur. On met à profit la structure de tableau d'entiers à deux dimensions pour introduire la notion d'image ponctuelle (« bitmap »). Les algorithmes de traitement d'image seront abordés plus tard.

- **Fichiers** : notion de chemin d'accès, lecture et écriture de données numériques ou de type chaîne de caractères depuis ou vers un fichier.

On encourage l'utilisation de fichiers en tant que supports de données ou de résultats avant divers traitements, par exemple graphiques.

L'utilisation de bases de données sera étudiée plus tard.

Les exemples de programmation ne se limitent pas à la traduction des algorithmes introduits en partie 2-b.

Les principales capacités développées dans cette partie sont les suivantes :

- choisir un type de données en fonction d'un problème à résoudre,
- concevoir l'en-tête (ou la spécification) d'une fonction, puis la fonction elle-même,
- traduire un algorithme dans un langage de programmation,
- gérer efficacement un ensemble de fichiers correspondant à des versions successives

d'un fichier source, + rechercher une information au sein d'une documentation en ligne, analyser des exemples fournis dans cette documentation, + documenter une fonction, un programme plus complexe.

CHAPTER

EIGHT

INGÉNIERIE NUMÉRIQUE ET SIMULATION

Première partie du semestre 2.

8.1 Objectifs et organisation de cet enseignement

Dans cette partie de programme, on étudie le développement d'algorithmes numériques sur des problèmes scientifiques étudiés et mis en équation dans les autres disciplines. La pédagogie par projets est encouragée.

8.2 Outils employés

L'objectif est de familiariser les étudiants avec un environnement de simulation numérique. Cet environnement doit permettre d'utiliser des bibliothèques de calcul numérique et leur documentation pour développer et exécuter des programmes numériques. On veillera à faire aussi programmer par les étudiants les algorithmes étudiés.

Aucune connaissance des fonctions des bibliothèques n'est exigible des étudiants.

Note: **Au moment de l'élaboration de ces programmes d'enseignement**, l'atelier logiciel Scilab ou le langage de programmation **Python**, avec les bibliothèques **Numpy/Scipy**, sont les environnements choisis.

8.3 Simulation numérique

Il s'agit d'apprendre aux étudiants à utiliser des algorithmes numériques simples et/ou à utiliser des bibliothèques pour résoudre des problèmes étudiés et mis en équation dans les autres disciplines. Le problème d'origine doit être exposé mais la modélisation (et la mise en équations) n'est pas un objectif de ce programme. Dans cette partie, on n'aborde pas les aspects théoriques des algorithmes étudiés (qui peuvent être traités dans d'autres disciplines). Seules la mise en œuvre constructive des algorithmes et l'analyse empirique des résultats sont concernées. On s'attache à comparer la solution numérique à une solution analytique quand elle existe, à des résultats expérimentaux, aux solutions obtenues en utilisant les fonctions de la bibliothèque de l'environnement de travail choisi. On illustre ainsi les performances de différents algorithmes pour la résolution des problèmes. On met l'accent sur les aspects pratiques comme l'impact des erreurs d'arrondi sur les résultats, les conditions d'arrêt, la complexité en temps de calcul ou le stockage en mémoire

Contenus Bibliothèques logicielles : utilisation de quelques fonctions d'une bibliothèque et de leur documentation en ligne.

Précisions et commentaires On met en évidence l'intérêt de faire appel aux bibliothèques, évitant de devoir réinventer des solutions à des problèmes bien connus. La recherche des spécifications des bibliothèques joue un rôle essentiel pour le développement de solutions fiables aux problèmes posés.

Problème stationnaire à une dimension, linéaire ou non conduisant à la résolution approchée d'une équation algébrique ou transcendante. Méthode de dichotomie, méthode de Newton.

On souligne les différences du comportement informatique des deux algorithmes en termes de rapidité. On illustre à nouveau le problème du test d'arrêt (inadéquation de la comparaison à zéro).

Problème dynamique à une dimension, linéaire ou non, conduisant à la résolution approchée d'une équation différentielle ordinaire par la méthode d'Euler.

On compare les résultats obtenus avec les fonctions de résolution approchée fournies par une bibliothèque numérique. On met en évidence l'impact du pas de discrétisation et du nombre d'itérations sur la qualité des résultats et sur le temps de calcul.

Problème discret multidimensionnel, linéaire, conduisant à la résolution d'un système linéaire inversible (ou de Cramer) par la méthode de Gauss avec recherche partielle du pivot.

Il ne s'agit pas de présenter cet algorithme (qui fait partie du cours de mathématiques) mais de l'exécuter pour étudier sa mise en œuvre et les problèmes que pose cette démarche. On souligne la complexité de l'algorithme en fonction de la taille des matrices et son impact sur le temps de calcul.

Les principales capacités développées dans cette partie de la formation sont : réaliser un programme complet structuré allant de la prise en compte de données expérimentales à la mise en forme des résultats permettant de résoudre un problème scientifique donné,

utiliser les bibliothèques de calcul standard pour résoudre un problème scientifique mis en équation lors des enseignements de chimie, physique, mathématiques, sciences industrielles et de l'ingénieur, tenir compte des aspects pratiques comme l'impact des erreurs d'arrondi sur les résultats, le temps de calcul ou le stockage en mémoire.

CHAPTER

NINE

INITIATION AUX BASES DE DONNÉES

Seconde moitié du semestre 2.

9.1 Objectifs de l'enseignement

L'objectif de cette partie de la formation vise à développer les savoir-faire suivants : + recourir aux concepts des bases de données relationnelles ; + traduire les questions posées dans un langage de requête en respectant sa syntaxe ; + prototyper et créer une base de données simple, à l'aide d'un outil interactif ; + consulter une base de données à travers des requêtes de type SQL ; + comprendre et décrire les rôles des différents éléments d'une architecture trois-tiers.

La formation doit mettre en évidence la nécessité d'un niveau d'abstraction suffisant dans la conception d'outils permettant la gestion de bases de données de taille importante, là où des algorithmes de recherche simples sur des structures « plates », orientées tableaux, deviennent inopérants : les schémas relationnels sont une réponse à ce problème.

9.2 Contenu

Contenus

Précisions et commentaires

Vocabulaire des bases de données : relation, Ces concepts sont présentés dans une attribut, domaine, schéma de relation ; notion de perspective applicative, à partir d'exemples. clé primaire. Opérateurs usuels sur les ensembles dans un contexte de bases de données : union, intersection, différence. Opérateurs spécifiques de l'algèbre relationnelle : projection, sélection (ou restriction), renommage, jointure, produit et division cartésiennes ; fonctions d'agrégation : min, max, somme, moyenne, comptage.

Ces concepts sont présentés dans une perspective applicative. Les seules jointures présentées seront les jointures symétriques, simples (utilisant JOIN . . . ON . . . =...).

Concept de client-serveur. Brève extension au cas de l'architecture trois-tiers.

On se limite à présenter ce concept dans la perspective applicative d'utilisation de bases de données.

La liste suivante énumère un choix non exhaustif d'exercices pratiques. Les bases de données utilisées à des fins d'illustration concerneront de préférence des questions choisies au sein des autres disciplines scientifiques et technologiques.

utiliser une application de création et de manipulation de données, offrant une interface graphique, notamment pour créer une base de données simple, ne comportant pas plus de trois tables ayant chacune un nombre limité de colonnes. L'installation et l'exploitation d'un serveur SQL ne fait pas partie des attendus. lancer des requêtes sur une base de données de taille plus importante, comportant plusieurs tables, que les étudiants n'auront pas eu à construire, à l'aide

d'une application offrant une interface graphique ; enchaîner une requête sur une base de données et un traitement des réponses enregistrées dans un fichier.

Les principales capacités développées dans cette partie de la formation sont :

- utiliser une application offrant une interface graphique pour créer une base de données

et l'alimenter, + utiliser une application offrant une interface graphique pour lancer des requêtes sur une base de données, + distinguer les rôles respectifs des machines client, serveur, et éventuellement serveur de données, + traduire dans le langage de l'algèbre relationnelle des requêtes écrites en langage courant, + concevoir une base constituée de plusieurs tables, et utiliser les jointures symétriques pour effectuer des requêtes croisées.

9.3 Algorithmique et programmation II

Seconde année.

9.4 Objectifs de l'enseignement

Le but de cette partie de la formation est de dépasser la vision des algorithmes qui a été introduite en semestre 1 et de donner accès à un petit nombre d'autres méthodes et structures, permettant d'envisager des applications à des domaines très variés. En combinaison avec les apports du semestre 2, les compétences acquises dans cette partie seront immédiatement utiles pour le développement des T.I.P.E. que les étudiants auront à réaliser.

9.5 Contenu

9.5.1 Piles

Algorithmes de manipulation : fonctions 'push' et 'pop'. On utilise des listes (ou tableaux à 1 dimension) pour leur implantation.

9.5.2 Récursivité

On en présente les avantages et les inconvénients.

Tris d'un tableau à une dimension de valeurs numériques : tri par insertion, tri rapide (ou 'quicksort'), tri par fusion. Application à la recherche de la médiane d'une liste de nombres.

On étudie et on compare ces algorithmes de tri du point de vue des complexités temporelles dans le meilleur et dans le pire cas.

Note: Les compétences en algorithmique et en programmation s'acquièrent par la pratique. Afin de développer des capacités opérationnelles en matière d'algorithmique et de programmation, il est nécessaire que la formation comprenne un volet de mise en pratique sur une variété de problèmes.

La liste suivante énumère un choix non exhaustif d'exercices pratiques dont un sous-ensemble pourra être étudié. Par la présentation succincte de leurs contextes, ces exercices seront aussi l'occasion d'introduire très brièvement et d'illustrer différents champs de l'informatique auprès des étudiants. Aucune connaissance de ces champs ni des algorithmes ci-dessous n'est cependant exigible.

- **Traitement des images. Représentation des couleurs par une liste de trois valeurs, d'une** image en couleurs par une matrice de pixels. Exemples de traitements d'images : augmentation du contraste, floutage, changement de résolution, recherche de contours.
- **Les images pourront être chargées en mémoire à partir de fichiers au moyen des** fonctions de bibliothèque. Aucune connaissance sur les différents formats de fichier d'image n'est exigible.
- **Codages, algorithmes de chiffrage et de cryptographie élémentaires. Algorithmes** élémentaires comme par exemple l'algorithme de Vigenère.
- **Transmission fiable de données. Sommes de contrôle (« checksum »)** [exemples] simples. Codes correcteurs : par exemple, le code de Hamming [7,4].
- **Ces questions permettent de faire le lien avec le codage binaire des nombres entiers.** Éléments de base de l'algorithmique des graphes pour la recherche opérationnelle et les réseaux (on représente les graphes pondérés par des matrices d'adjacence).
- Algorithme de Dijkstra de recherche du plus court chemin dans un graphe pondéré à poids positifs.
- **Programmation orientée objet et interfaces graphiques. Découverte de la programmation** orientée objet au travers de l'observation de l'implantation d'interfaces graphiques existants.

Les principales capacités développées dans cette partie de la formation sont :

- comprendre un algorithme et expliquer ce qu'il fait,
- programmer un algorithme dans un langage de programmation moderne et général,
- modifier un algorithme existant pour obtenir un résultat différent,
- concevoir un algorithme répondant à un problème précisément posé,
- expliquer le fonctionnement d'un algorithme,
- comprendre le fonctionnement d'un algorithme récursif et l'utilisation de la mémoire lors

de son exécution, + comprendre les avantages et défauts respectifs des approches récursive et itérative, s'interroger sur l'efficacité algorithmique temporelle d'un algorithme, distinguer par leurs complexités deux algorithmes résolvant un même problème.

CHAPTER

TEN

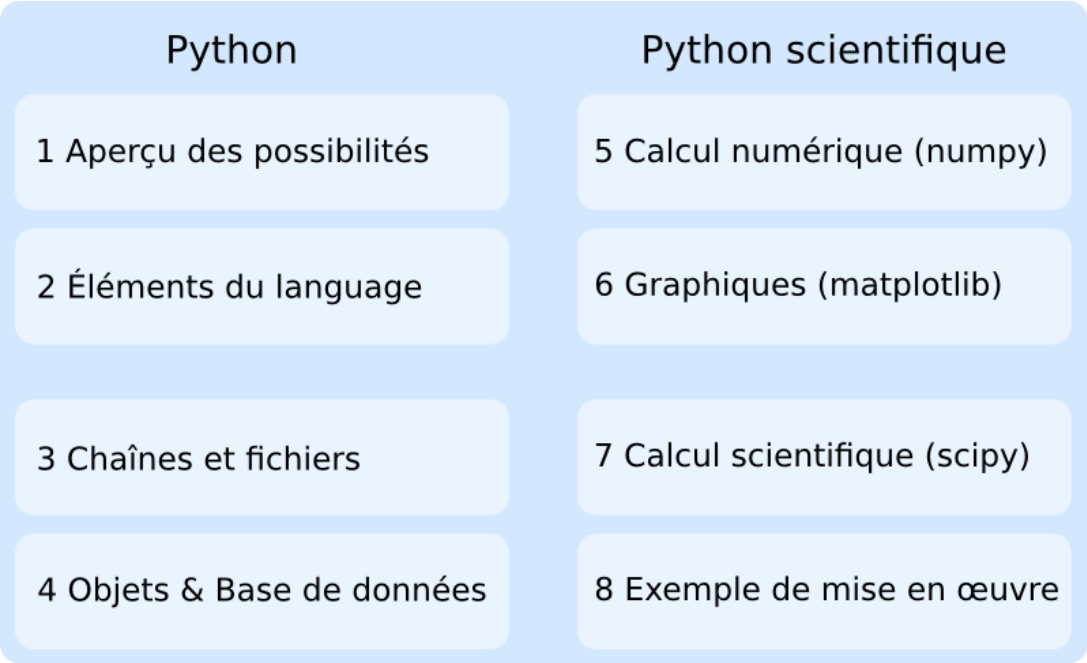
INTRODUCTION

Ce document est destiné à élaborer collaborativement une formation à Python et Python scientifique (Numpy, Scipy, matplotlib, ...) à destination des enseignants des classes préparatoires aux grandes écoles.

La formation, prévue sur *deux jours* comporte 2x4 séances de 1h30 (12h au total), avec

- un premier jour centré sur Python "en général"
- et le 2ème sur Python scientifique

Organisation éventuellement à revoir avec une possibilité de démarrer la présentation des aspects scientiques plus tôt



Pour rendre un peu plus concrets les concepts présentés, chaque séance s'appuira sur un ou des exemples.

CHAPTER

ELEVEN

SÉANCES

11.1 Généralités sur Python et l'intérêt de son utilisation

Python est un language interprété. L'utilisateur voit immédiatement l'effet de sa commande. Ce mécanisme s'avère être très utile en sciences où beaucoup de traitements complexes sont menées pas à pas, en vérifiant et en testant à chaque pas la validité des résultats.

Le langage Python rencontre aujourd'hui un engouement très grand dans de nombreuses communautés scientifiques qui ont la nécessité de "manipuler" des données et de les faire parler. Ce mouvement est profond et potentiellement extrèmement fécond car il génère des liens inter-disciplinaires autour de besoins communs : la maîtrise du calcul, la maîtrise des données, la communication des résultats.

Pour accompagner ce mouvement et pour l'accentuer, il importe que de plus en plus de scientifiques s'emparent de ces outils et apprennent à en maîtriser la puissance et à en apprécier l'élégance.

11.1.1 Premier exemple

Voici un exemple d'une session très simple, exécutée dans l'environnement interactif Ipython (développé à l'Université de Berkeley par Fernando Perez) qui vise à devenir une plateforme générique de l'utilisation du langage Python dans le domaine des sciences.

```
In [1]: from math import *

In [2]: cos(pi/3)**2+sin(pi/3)**2
Out[2]: 1.0

In [3]: 1j*1j
Out[3]: (-1+0j)
```

Note: La première étape d'un programme Python consiste à charger des modules spécialisés. Ceux-ci sont très nombreux et couvrent un très large éventail de besoins. **L'objectif de la formation sera de comprendre les modules importants pour le calcul scientifique**

Par exemple dans l'exemple précédent le module *math* donne accès aux fonctions de la librairie standard *math* du langage C. Sans cette importation, il serait impossible d'accèder aux fonctions $sin(x)$ et $cos(x)$.

Note: Une fonction se reconnait à son prototype qui utilise obligatoirement les parenthèses *unnomdefunction()*

```
In [4]: exp(1j*pi)
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-4-860da8d890b6> in <module>()
----> 1 exp(1j*pi)

TypeError: can't convert complex to float
```

L'interpréteur du langage renvoie un message d'erreur indiquant qu'il ne sait pas calculer une exponentielle avec un argument complexe.

C'est gênant.

Fort heureusement, il existe d'autres modules, plus complets, qui peuvent prendre le relais. Parmi ceux-ci le module *numpy*.

```
In [5]: from numpy import *

In [6]: exp(1j*pi)
Out[6]: (-1+1.2246063538223773e-16j)
```

C'est mieux ! Mais viens alors la question : Comment fait Python pour savoir quelle exponentielle choisir, puisqu'il a le choix entre l'exponentielle du module *math* et l'exponentielle du module *numpy* ?

C'est effectivement un problème. Il vient de se créer un conflit entre espaces de noms : celui du module *math* et celui de *numpy*. Un même nom est partagé par plusieurs modules.

On peut régler ce probléme en important le module différemment. Il y a trois possibilités :

- importer le module (sans son espace de noms)
- importer le module * (avec son espace de noms)
- importer le module avec un alias (souvent court par commodité)

```
In [7]: import numpy

In [8]: from numpy import *

In [9]: import numpy as np
```

La troisième solution est la bonne pratique.

L'inconvénient, c'est qu'il faut alonger le nom des fonctions appartenant au module en les faisant précéder de **np.**, (**numpy.** dans le premier cas), ce qui nuit (un peu) à la lisibilité du code.

Le grand avantage est que l'on règle ainsi le problème du conflit entre espaces de noms. Celui ci peut être très gênant et souvent source d'erreurs pénibles à découvrir (*la fonction appelée n'étant pas la fonction que l'on croit être appelée*). Ceci peut facilement se produire, car quel développeur peut prétendre avoir mémorisé l'intégralité des fonctions de tous les modules dont il a l'usage ? C'est trop gros, trop vaste, il faut cloisonner, et Python permet cela.

Bien sûr si l'usage que l'on a de python implique très peu de module l'import * est raisonable.

Les librairies importantes ont des alias génériques adoptés par tous qu'il convient d'employer.

```
In [10]: import numpy as np

In [11]: import scipy as sp

In [12]: import scipy.io as ios

In [13]: import scipy.linalg as la
```

```
In [14]: import matplotlib.pyplot as plt
```

11.2 Les bases du langage Python

11.3 Manipulation des chaines de caractères et des fichiers

11.4 Algorithmique et notions d'objects

11.5 Numpy : Créer et manipuler des données numériques où Python et les Nombres

numpy chez scipy lectures

11.6 Séance 5 : Tracé de graphes avec Matplotlib

scipy-lectures

11.7 Scipy et le calcul scientifique de haut niveau

scipy

11.8 Une application pratique de synthèse

CHAPTER

TWELVE

AUTRES

12.1 Liens Utiles

scipy-lectures pep8