Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

MINISTÈRE
DE L'ENSEIGNEMENT
SUPÉRIEUR,
DE LA RECHERCHE
ET DE L'INNOVATION

Classes préparatoires aux grandes écoles

Filière scientifique

Voie Technologie, physique et chimie (TPC)

Annexe 4

Programmes d'informatique

1ère et 2nde années

Table des matières

Introduction au programme

Les objectifs du programme Le programme d'informatique de TSI et TPC s'inscrit entre deux continuités : en amont avec les programmes rénovés du lycée, en aval avec les enseignements dispensés dans les grandes écoles, et plus généralement les poursuites d'études universitaires. Il a pour objectif la formation de futurs ingénieures et ingénieurs, enseignantes et enseignants, chercheuses et chercheurs et avant tout des personnes informées, capables de gouverner leur vie professionnelle et citoyenne en pleine connaissance et maîtrise des techniques et des enjeux de l'informatique et en la nourrissant par les habitudes de la démarche scientifique. Le présent programme a pour ambition de poser les bases d'un enseignement cohérent et mesuré d'une science informatique encore jeune et dont les manifestations technologiques connaissent des cycles d'obsolescence rapide. On garde donc à l'esprit :

— de privilégier la présentation de concepts fondamentaux pérennes sans s'attacher outre mesure à la description de technologies, protocoles ou normes actuels ;
— de donner aux futurs diplômées et diplômés les moyens de réussir dans un domaine en mutation rapide et dont les technologies qui en sont issues peuvent sauter brutalement d'un paradigme à un autre très différent ;
— de préparer les étudiantes et étudiants à tout un panel de professions et de situations de la vie professionnelle qui les amène à remplir tour à tour une mission d'expertise, de création ou d'invention, de prescription de méthodes ou de techniques, de contrôle critique des choix opérés ou encore de décision en interaction avec des spécialistes ;
— que les concepts à enseigner sont les mêmes dans toutes les filières mais que le professeur d'informatique de chaque classe peut adapter la façon de les transmettre et les exemples concrets sur lesquels il s'appuie au profil de ses élèves et aux autres enseignements qu'ils suivent.

Compétences visées Ce programme vise à développer les six grandes compétences suivantes :

analyser et modéliser un problème ou une situation, notamment en utilisant les objets conceptuels de l'informatique pertinents (table relationnelle, graphe, dictionnaire, etc.) ;

imaginer et concevoir une solution, décomposer en blocs, se ramener à des sous-problèmes simples et indépendants, adopter

une stratégie appropriée, décrire une démarche, un algorithme ou une structure de données permettant de résoudre le problème ;

décrire et spécifier les caractéristiques d'un processus, les données d'un problème, ou celles manipulées par un algorithme ou une fonction ;

mettre en œuvre une solution, par la traduction d'un algorithme ou d'une structure de données dans un langage de programmation ou un langage de requête ;

justifier et critiquer une solution, que ce soit en démontrant un algorithme par une preuve mathématique ou en développant des processus d'évaluation, de contrôle, de validation d'un code que l'on a produit ;

communiquer à l'écrit ou à l'oral, présenter des travaux informatiques, une problématique et sa solution ; défendre ses choix ; documenter sa production et son implémentation.

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. Les exemples ou les exercices d'application peuvent être choisis au sein de l'informatique elle-même ou en lien avec d'autres champs disciplinaires.

Sur les partis pris par le programme Ce programme impose aussi souvent que possible des choix de vocabulaire ou de notation de certaines notions. Les choix opérés ne présument pas la supériorité de l'option retenue. Ils ont été précisés dans l'unique but d'aligner les pratiques d'une classe à une autre et d'éviter l'introduction de longues définitions récapitulatives préliminaires à un exercice ou un problème. De même, ce programme nomme aussi souvent que possible l'un des algorithmes possibles parmi les classiques qui répondent à un problème donné. Là encore, le programme ne défend pas la prééminence d'un algorithme ou d'une méthode par rapport à un autre mais il invite à faire bien plutôt que beaucoup.

Sur les langages et la programmation L'enseignement du présent programme repose sur un langage de manipulation de données (SQL) ainsi que le langage de programmation Python, pour lequel une annexe liste de façon limitative les éléments qui sont exigibles des étudiants ainsi que ceux auxquels les étudiants sont

familiarisés et qui peuvent être attendus à condition qu'ils soient accompagnés d'une documentation. La poursuite de l'apprentissage du langage Python est vue en particulier par les étudiants pour adopter immédiatement une bonne discipline de programmation tout en se concentrant sur le noyau du langage plutôt que sur une API pléthorique.

Mode d'emploi Ce programme a été rédigé par semestre pour assurer une certaine homogénéité de la formation. Le premier semestre permet d'asseoir les bases de programmation vues au lycée et les concepts associés. **L'organisation de la progression au sein des deux premiers semestres relève de la responsabilité pédagogique de la professeure ou du professeur** et le tissage de liens entre les thèmes contribue à la valeur de son enseignement. Les notions étudiées lors d'un semestre précédent sont régulièrement revisitées tout au long des deux années d'enseignement.

1 Programme du premier semestre

Le programme du premier semestre poursuit les objectifs suivants :
- consolider l'apprentissage de la programmation en langage Python qui a été entrepris dans les classes du lycée ;
- mettre en place un environnement de travail ;
- mettre en place une discipline de programmation : spécification précise des fonctions et programmes, annotations et commentaires, jeux de tests ;
- introduire les premiers éléments de complexité des algorithmes : on ne présente que l'estimation asymptotique du coût dans le cas le pire ;
- introduire des outils de validation : variants et invariants.

Le tableau ci-dessous présente les thèmes qui sont abordés lors de ces séances, et, en colonne de droite, une liste, sans aucun caractère impératif, d'exemples d'activités qui peuvent être proposées aux étudiants. L'ordre de ces thèmes n'est pas impératif.

Aucune connaissance relative aux modules éventuellement rencontrés lors de ces séances n'est exigible des étudiants.

Thèmes	**Exemples d'activité. *Commentaires.***
Recherche séquentielle dans un tableau unidimensionnel. Dictionnaire.	Recherche d'un élément. Recherche du maximum, du second maximum. Comptage des éléments d'un tableau à l'aide d'un dictionnaire. *Manipulations élémentaires d'un tableau unidimensionnel. Utilisation de dictionnaires en boîte noire. Notions de coût constant, de coût linéaire.*
Algorithmes opérant sur une structure séquentielle par boucles simples ou imbriquées.	Recherche d'un facteur (ou d'un mot) dans un texte. Calcul d'une intégrale par la formule de la moyenne ou par la méthode des trapèzes. Tri à bulles. *Notion de complexité quadratique. On propose des outils pour valider la correction de l'algorithme.*
Utilisation de modules, de bibliothèques.	Lecture d'un fichier de données simples. Calculs statistiques sur ces données. Représentation graphique (histogrammes, etc.).
Algorithmes dichotomiques.	Recherche dichotomique dans un tableau trié. Calcul d'une solution de l'équation $f(x) = 0$ sur $[a, b]$ quand $f(a).f(b) < 0$. *On met en évidence une accélération entre complexité linéaire d'un algorithme naïf et complexité logarithmique d'un algorithme dichotomique. On met en œuvre des jeux de tests, des outils de validation.*
Fonctions récursives.	Version récursive d'algorithmes dichotomiques. Fonctions produisant à l'aide de `print` successifs des figures alphanumériques. Dessins de fractales. *On évite de se cantonner à des fonctions mathématiques (factorielle, suites récurrentes). On peut montrer le phénomène de dépassement de la taille de la pile.*
Tableau de pixels et images.	Algorithmes de rotation de 90 degrés, de réduction ou d'agrandissement. Modification d'une image : flou, détection de contour, etc. *Les images servent de support à la présentation de manipulations de tableaux à deux dimensions.*
Tris.	Algorithmes quadratiques : tri par insertion, par sélection. Tri par partition-fusion. Tri par comptage. *On fait observer différentes caractéristiques (par exemple, stable ou non, en place ou non, comparatif ou non, etc).*

2 Programme du second semestre

2.1 Méthodes de programmation et analyse des algorithmes

Même si on ne prouve pas systématiquement tous les algorithmes, on dégage l'idée qu'un algorithme doit se prouver et que sa programmation doit se tester.

Notions	Commentaires
Instruction et expression. Effet de bord.	On peut signaler par exemple que le fait que l'affectation soit une instruction est un choix des concepteurs du langage Python et en expliquer les conséquences.
Spécification des données attendues en entrée, et fournies en sortie/retour.	On entraîne les étudiants à accompagner leurs programmes et leurs fonctions d'une spécification. Les signatures des fonctions sont toujours précisées.
Annotation d'un bloc d'instructions par une précondition, une postcondition, une propriété invariante.	Ces annotations se font à l'aide de commentaires.
Assertion.	L'utilisation d'assertions est encouragée par exemple pour valider des entrées. La levée d'une assertion entraîne l'arrêt du programme. Ni la définition ni le rattrapage des exceptions ne sont au programme.
Explicitation et justification des choix de conception ou programmation.	Les parties complexes de codes ou d'algorithmes font l'objet de commentaires qui l'éclairent en évitant la paraphrase. Le choix des collections employées (par exemple, liste ou dictionnaire) est un choix éclairé.
Terminaison. Variant. Invariant.	On montre sur plusieurs exemples que la terminaison peut se démontrer à l'aide d'un variant de boucle. Sur plusieurs exemples, on explicite, sans insister sur aucun formalisme, des invariants de boucles en vue de montrer la correction des algorithmes.
Jeu de tests associé à un programme.	Il n'est pas attendu de connaissances sur la génération automatique de jeux de tests ; un étudiant doit savoir écrire un jeu de tests à la main, donnant à la fois des entrées et les sorties correspondantes attendues. On sensibilise, par des exemples, à la notion de partitionnement des domaines d'entrée et au test des limites.
Complexité.	On aborde la notion de complexité temporelle dans le pire cas en ordre de grandeur. On peut, sur des exemples, aborder la notion de complexité en espace.

2.2 Représentation des nombres

On présente sans formalisation théorique les enjeux de la représentation en mémoire des nombres. Ces notions permettent d'expliquer certaines difficultés rencontrées et précautions à prendre lors de la programmation ou de l'utilisation d'algorithmes de calcul numérique dans les disciplines qui y recourent.

Notions	Commentaires
Représentation des entiers positifs sur des mots de taille fixe.	La conversion d'une base à une autre n'est pas un objectif de formation.
Représentation des entiers signés sur des mots de taille fixe.	Complément à deux.
Entiers multi-précision de Python.	On les distingue des entiers de taille fixe sans détailler leur implémentation. On signale la difficulté à évaluer la complexité des opérations arithmétiques sur ces entiers.
Distinction entre nombres réels, décimaux et flottants.	On montre sur des exemples l'impossibilité de représenter certains nombres réels ou décimaux dans un mot machine

Représentation des flottants sur des mots de taille fixe. Notion de mantisse, d'exposant.	On signale la représentation de 0 mais on n'évoque pas les nombres dénormalisés, les infinis ni les NaN. Aucune connaissance liée à la norme IEEE-754 n'est au programme.
Précision des calculs en flottants.	On insiste sur les limites de précision dans le calcul avec des flottants, en particulier pour les comparaisons. On n'entre pas dans un comparatif des différents modes d'arrondi.

2.3 Bases des graphes

Il s'agit de définir le modèle des graphes, leurs représentations et leurs manipulations.
On s'efforce de mettre en avant des applications importantes et si possibles modernes : réseau de transport, graphe du web, réseaux sociaux, bio-informatique. On précise autant que possible la taille typique de tels graphes.

Notions	**Commentaires**
Vocabulaire des graphes.	Graphe orienté, graphe non orienté. Sommet (ou nœud) ; arc, arête. Boucle. Degré (entrant et sortant). Chemin d'un sommet à un autre. Cycle. Connexité dans les graphes non orientés. On présente l'implémentation des graphes à l'aide de listes d'adjacence (rassemblées par exemple dans une liste ou dans un dictionnaire) et de matrice d'adjacence ainsi que les différences en terme d'occupation mémoire. On n'évoque ni multi-arcs ni multi-arêtes.
Notations.	Graphe $G = (S,A)$, degrés $d(s)$ (pour un graphe non orienté), $d_+(s)$ et $d_-(s)$ (pour un graphe orienté).
Pondération d'un graphe. Étiquettes des arcs ou des arêtes d'un graphe.	On motive l'ajout d'information à un graphe par des exemples concrets.
Fonctions de manipulation.	Obtention du nombre de sommets, ajout/suppression/test d'existence d'un arc ou d'une arête, construction de la liste des voisins d'un sommet, etc. On présente l'importance de construire des fonctions de manipulation élémentaires en vue d'une programmation modulaire ainsi que l'impact du choix d'une représentation en terme de complexité temporelle.
Parcours d'un graphe.	On introduit à cette occasion les piles et les files ; on souligne les problèmes d'efficacité posés par l'implémentation des files par les listes de Python et l'avantage d'utiliser un module dédié tel que `collections.deque`. Détection de la présence de cycles ou de la connexité d'un graphe non orienté.

3 Programme du troisième semestre

3.1 Bases de données

On se limite volontairement à une description applicative des bases de données en langage SQL. Il s'agit de permettre d'interroger une base présentant des données à travers plusieurs relations. On ne présente pas l'algèbre relationnelle ni le calcul relationnel.

Notions	**Commentaires**
Vocabulaire des bases de données : tables ou relations, attributs ou colonnes, domaine, schéma de tables, enregistrements ou lignes, types de données.	On présente ces concepts à travers de nombreux exemples. On s'en tient à une notion sommaire de domaine : entier, flottant, chaîne ; aucune considération quant aux types des moteurs SQL n'est au programme. Aucune notion relative à la représentation des dates n'est au programme ; en tant que de besoin on s'appuie sur des types numériques ou chaîne pour lesquels la relation d'ordre coïncide avec l'écoulement du temps. Toute notion relative aux collations est hors programme ; en tant que de besoin on se place dans l'hypothèse que la relation d'ordre correspond à l'ordre lexicographique usuel. `NULL` est hors programme.
Clef primaire.	Une clef primaire n'est pas forcément associée à un unique attribut même si c'est le cas le plus fréquent. La notion d'index est hors programme. On présente la notion de clef étrangère.
Requêtes `SELECT` avec simple clause `WHERE` (sélection), projection, renommage `AS`. Utilisation des mots-clefs `DISTINCT`, `LIMIT`, `OFFSET`, `ORDER BY`.	Les opérateurs au programme sont `+`, `-`, `*`, `/` (on passe outre les subtilités liées à la division entière ou flottante), `=`, `<>`, `<`, `<=`, `>`, `>=`, `AND`, `OR`, `NOT`.
Opérateurs ensemblistes `UNION`, `INTERSECT` et `EXCEPT`, produit cartésien.	
Jointures internes T_1 `JOIN` T_2 … `JOIN` T_n `ON` ϕ. Autojointure.	On présente les jointures en lien avec la notion de relations entre tables. On se limite aux équi-jointures : ϕ est une conjonction d'égalités. On fera le lien avec la notion de clef étrangère
Agrégation avec les fonctions `MIN`, `MAX`, `SUM`, `AVG` et `COUNT`, y compris avec `GROUP BY`.	Pour la mise en œuvre des agrégats, on s'en tient à la norme SQL99. On présente quelques exemples de requêtes imbriquées. On marque la différence entre `WHERE` et `HAVING` sur des exemples.

Mise en œuvre
La création de tables et la suppression de tables au travers du langage SQL sont hors programme. La mise en œuvre effective se fait au travers d'un logiciel permettant d'interroger une base de données à l'aide de requêtes SQL. Récupérer le résultat d'une requête à partir d'un programme n'est pas un objectif. Même si aucun formalisme graphique précis n'est au programme, on peut décrire les entités et les associations qui les lient au travers de diagrammes sagittaux informels. Sont hors programme : la notion de modèle logique *vs* physique, les bases de données non relationnelles, les méthodes de modélisation de base, les fragments DDL, TCL et ACL du langage SQL, les transactions, l'optimisation de requêtes par l'algèbre relationnelle.

3.2 Dictionnaires et algorithmes pour l'intelligence artificielle

Les dictionnaires sont utilisés en boîte noire dès la première année ; les principes de leur fonctionnement sont présentés en deuxième année.
Cette partie permet aussi de revisiter les notions de programmation et de représentation de données par un graphe, qui sont vues en première année, en les appliquant à des enjeux contemporains.

Notions	**Commentaires**
Dictionnaires, clefs et valeurs.	On présente les principes du hachage, et les limitations qui en découlent sur le domaine des clefs utilisables.
Usage des dictionnaires en programmation Python.	Syntaxe pour l'écriture des dictionnaires. Parcours d'un dictionnaire.
Algorithme des k plus proches voisins avec distance euclidienne.	Matrice de confusion. Lien avec l'apprentissage supervisé.
Algorithme des k-moyennes.	Lien avec l'apprentissage non-supervisé. La démonstration de la convergence n'est pas au programme. On observe des convergences vers des minima locaux.
Jeux d'accessibilité à deux joueurs sur un graphe. Stratégie. Stratégie gagnante. Position gagnante.	On considère des jeux à deux joueurs (J_1 et J_2) modélisés par des graphes bipartis (l'ensemble des états contrôlés par J_1 et l'ensemble des états contrôlés par J_2). Il y a trois types d'états finals : les états gagnants pour J_1, les états gagnants pour J_2 et les états de match nul. On ne considère que les stratégies sans mémoire.
Notion d'heuristique.	On présente la notion d'heuristique à partir d'exemples simples comme le problème du sac à dos.

Mise en œuvre
La connaissance dans le détail des algorithmes de cette section n'est pas un attendu du programme. Les étudiants acquièrent une familiarité avec les idées sous-jacentes qu'ils peuvent réinvestir dans des situations où les modélisations et les recommandations d'implémentation sont guidées, notamment dans leurs aspects arborescents.

3.3 Algorithmique numérique

Dans cette partie du programme, on présente des algorithmes numériques pour une approche pluri-disciplinaire. On se concentre sur la bonne compréhension de leur fonctionnement et la mise en avant de leur limites.
Ces algorithmes seront appliqués à des problématiques concrètes étudiées dans d'autres disciplines. On veillera à faire programmer par les étudiants les algorithmes étudiés.

Notions	**Commentaires**
Méthode d'Euler	Résolution approchée d'équations différentielles ordinaires d'ordre 1 et 2. Illustration de l'impact du pas sur la qualité de la solution obtenue.
Méthode de Gauss avec recherche partielle du pivot.	Résolution des systèmes linéaires inversibles. Erreurs d'arrondis et problème de la comparaison à zéro. On aura recours à une conception modulaire pour présenter cet algorithme.
Interpolation polynomiale de Lagrange.	Interpolation par morceaux.

Mise en œuvre
La connaissance dans le détail des algorithmes de cette section n'est toujours pas un attendu du programme.

A Langage Python

Cette annexe liste limitativement les éléments du langage Python (version 3 ou supérieure) dont la connaissance est exigible des étudiants. Aucun concept sous-jacent n'est exigible au titre de la présente annexe.
Aucune connaissance sur un module particulier n'est exigible des étudiants.
Toute utilisation d'autres éléments du langage que ceux que liste cette annexe, ou d'une fonction d'un module, doit obligatoirement être accompagnée de la documentation utile, sans que puisse être attendue une quelconque maîtrise par les étudiants de ces éléments.

Traits généraux

- Typage dynamique : l'interpréteur détermine le type à la volée lors de l'exécution du code.
- Principe d'indentation.
- Portée lexicale : lorsqu'une expression fait référence à une variable à l'intérieur d'une fonction, Python cherche la valeur définie à l'intérieur de la fonction et à défaut la valeur dans l'espace global du module.
- Appel de fonction par valeur : l'exécution de $f(x)$ évalue d'abord x puis exécute f avec la valeur calculée.

Types de base

- Opérations sur les entiers (`int`) : `+`, `-`, `*`, `//`, `**`, `%` avec des opérandes positifs.
- Opérations sur les flottants (`float`) : `+`, `-`, `*`, `/`, `**`.
- Opérations sur les booléens (`bool`) : **`not`**, **`or`**, **`and`** (et leur caractère paresseux).
- Comparaisons `==`, `!=`, `<`, `>`, `<=`, `>=`.

Types structurés

- Structures indicées immuables (chaînes, tuples) : `len`, accès par indice positif valide, concaténation `+`, répétition `*`, tranche.
- Listes : création par compréhension `[`*e* **`for`** *x* **`in`** *s*`]`, par `[`*e*`]` `*` *n*, par `append` successifs ; `len`, accès par indice positif valide ; concaténation `+`, extraction de tranche, copie (y compris son caractère superficiel) ; `pop` en dernière position.
- Dictionnaires : création, accès, insertion, `len`, `copy`.

Structures de contrôle

- Instruction d'affectation avec `=`. Dépaquetage de tuples.
- Instruction conditionnelle : **`if`**, **`elif`**, **`else`**.
- Boucle **`while`** (sans **`else`**). **`break`**, **`return`** dans un corps de boucle.
- Boucle **`for`** (sans **`else`**) et itération sur `range(`*a*`, `*b*`)`, une chaîne, un tuple, une liste, un dictionnaire au travers des méthodes `keys` et `items`.
- Définition d'une fonction **`def`** f`(`p_1`, ..., `p_n`)`, **`return`**.

Divers

- Introduction d'un commentaire avec `#`.
- Utilisation simple de `print`, sans paramètre facultatif.
- Importation de modules avec **`import`** *module*, **`import`** *module* **`as`** *alias*, **`from`** *module* **`import`** $f, g, \ldots$
- Manipulation de fichiers texte (la documentation utile de ces fonctions doit être rappelée ; tout problème relatif aux encodages est éludé) : `open`, `read`, `readline`, `readlines`, `split`, `write`, `close`.
- Assertion : `assert` (sans message d'erreur).