Annexe IV
Objectifs de formation et programme d'informatique de la classe préparatoire scientifique d'ATS Ingénierie Industrielle

Objectifs de formation

Le programme d'informatique d'ATS s'inscrit entre deux continuités : en amont avec les programmes de BTS et BUT, 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 nourrie par les pratiques de la démarche scientifique, en pleine connaissance et maîtrise des techniques et des enjeux de l'informatique.

Organisation du Programme L'enseignement d'informatique en classe d'ATS s'effectue exclusivement à travers des séances de manipulation individuelle sur ordinateur, éventuellement complétées par des évaluations. À ce titre, le programme d'informatique présente des notions qui permettent d'allier les concepts fondamentaux et leur bonne appropriation.

Sur le langage et les partis pris par le programme L'enseignement repose sur le langage de programmation Python. Une annexe liste les éléments exigibles des étudiants ainsi que ceux pouvant être attendus avec documentation. L'apprentissage du langage Python vise à instaurer une bonne discipline de programmation en se concentrant sur un fragment du langage.
Le programme impose des choix de présentation et de notation privilégiant la bonne maîtrise d'un socle restreint de notions.

Mode d'emploi Le programme est structuré en trois parties. La première partie présente les bases de la programmation impérative avec Python et correspond, de manière indicative, à une moitié des séances. La deuxième partie prolonge et approfondit ces notions et la dernière partie présente les bases de l'algorithmique numérique dans l'objectif de renforcer la pratique tout en permettant des liens avec les autres disciplines scientifiques.
L'organisation de la progression relève de la responsabilité pédagogique des enseignants, avec des notions revisitées tout au long de l'année.

Programme

Introduction à la programmation impérative avec Python

Cette première partie de l'année a pour objectif de donner les bases de la programmation impérative aux élèves. Les algorithmes étudiés permettent de pratiquer cette programmation afin de renforcer cet apprentissage, mais l'analyse théorique de ceux-ci est hors programme.

Notions	**Commentaires**
Expressions, variables. Instructions et effet de bord.	On présente principalement les valeurs numériques, type `int` et `float` dans le but de réaliser des calculs, ainsi que les booléens, type `bool`, dans le but d'écrire des conditions. La présentation des chaînes de caractères s'effectue dans le but de faciliter les entrées et les sorties.

Instructions conditionnelles.	Afin d'introduire un comportement dynamique en l'absence de fonctions, on peut dans un premier temps présenter un appel comme `random.randint` au sein d'une condition sans préciser à ce stade la notion de module.
Boucles inconditionnelles bornées.	On présente uniquement les boucles de la forme `for compteur in range(debut, arret):` à ce stade.
Boucles inconditionnelles.	On réserve l'usage des boucles inconditionnelles `while` au seul cas où le nombre d'itérations ne peut être connu à l'avance.
Fonctions.	Les fonctions sont introduites dans un premier temps comme le mode principal de structuration des programmes. La validation des données en entrée s'effectue à l'aide de l'instruction `assert` en évitant toute technicité excessive. Les fonctions récursives sont hors programme.
Tableaux de taille fixe. Notion d'indice et de valeur. Itération sur un tableau par indice et par valeur.	On présente le type `list` dans le but de manipuler des tableaux de taille fixe. À ces fins, on limite l'usage de la méthode `append` à la construction ou à l'initialisation des tableaux. Les méthodes `pop` et assimilées ainsi que la notion d'extraction de tranche sont hors programme. La construction de listes par compréhension ou par multiplication d'une liste singleton est hors programme. On ne soulève aucune difficulté sur la notion de méthode qui est vue comme une syntaxe particulière d'appel de fonction.
Algorithmes élémentaires portant sur les tableaux.	Recherche de l'indice d'un élément, détermination de l'indice ou de la valeur d'un maximum d'un tableau non vide, calcul de la somme des éléments, renversement d'un tableau par échanges successifs. On insiste sur la différence entre indice et valeur dans un tableau dans les paramètres et les valeurs de retour de ces algorithmes.

Approfondissement et exploration

Cette partie du programme permet aux élèves d'approfondir leurs connaissances sans introduire de notions trop éloignées des précédentes.

Notions	**Commentaires**
Import et utilisation de modules.	On présente l'import global `import ...` ainsi que l'import qualifié `from ... import ...` L'usage de `from ... import *` est proscrit. Les modules `math`, `statistics`, `csv` et `matplotlib` peuvent être supports d'exemples. Dans le cas de l'utilisation du module `csv`, on se limite à la lecture d'un fichier donné à l'aide d'une expression `open(nom_fichier)` sans soulever de difficulté sur la notion de fichiers.
Tableaux bi-dimensionnels.	Manipulations élémentaires, somme par ligne, par colonne, somme totale. Les images en niveaux de gris sont un cas d'utilisation de tableaux bi-dimensionnels pouvant servir d'exemples tout en permettant de les visualiser simplement avec `matplotlib`. On peut ainsi présenter des exemples simples de traitements par modification des éléments, comme le négatif d'une image, ou par permutation des éléments, comme le miroir d'une image. Les tableaux bi-dimensionnels sont définis exclusivement à l'aide d'ajouts de lignes par `append`, ces lignes étant elles-mêmes construites de la même manière.

Tri de tableaux.	On présente l'intérêt des tris indépendamment de leur implémentation à l'aide de la méthode `.sort()`. Des exemples d'implémentations peuvent être donnés comme le tri par sélection ou le tri par insertion. On ne soulève aucune difficulté concernant la correction de ces algorithmes.
Introduction expérimentale à la complexité.	On présente ici des mesures de temps associées à des programmes étudiés auparavant. On pourra utiliser la fonction `time.time` afin de mesurer les temps.

Algorithmique numérique

L'algorithmique numérique fournit ici des exemples de programmes que les élèves retrouvent soit directement, soit à travers des bibliothèques de calcul, dans les autres disciplines scientifiques. La résolution numérique n'étant pas l'objectif premier, l'efficacité des solutions est moins importante que la pratique raisonnée des notions vues précédemment. À ce titre, l'usage de structures de données autres que celles introduites dans ce programme est à proscrire. En particulier, `numpy` est hors programme. L'étude de ces algorithmes ne doit pas faire l'objet de développements mathématiques, et se contente d'un point de vue expérimental pour comparer la qualité des solutions.

Notions	**Commentaires**
Calculs en virgule flottante.	On présente les enjeux spécifiques liés au calcul en virgule flottante. Notamment, on présente les questions de précisions des calculs, de comparaisons de nombres et d'erreurs de calculs.
Représentation de fonction. Dérivée discrète.	On compare visuellement la représentation de fonction mathématique sous la forme d'une fonction Python de calcul ou sous une forme tabulée. Par convention, le calcul de la dérivée discrète s'effectue par différence avant.
Calcul approché d'intégrales.	Méthodes des rectangles, méthode des trapèzes. On compare expérimentalement les méthodes.
Recherche approchée des zéros d'une fonction.	Dichotomie, méthode de Newton. On peut présenter les algorithmes avec les deux représentations possibles de fonctions.

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.
- Évaluation et affectation.

Types de base

- Opérations sur les entiers (`int`) : `+`, `-`, `*`, `//`, `**`, `%` avec des opérandes positives.
- Opérations sur les flottants (`float`) : `+`, `-`, `*`, `/`, `**`.
- Opérations sur les booléens (`bool`) : **`not`**, **`or`**, **`and`** (et leur caractère paresseux).
- Chaînes de caractères (`str`) : conversion vers un entier ou un flottant.
- Comparaisons : `==`, `!=`, `<`, `>`, `<=`, `>=`.

Listes

- Création explicite de listes finies `[`e_1`, ..., `e_n`]`.
- Création par `append` successifs.
- Fonction `len`, accès par indice positif valide.

Structures de contrôle

- Instruction d'affectation avec `=`.
- Instruction conditionnelle : **`if`**, **`elif`**, **`else`**.
- Boucle **`while`** (sans **`else`**). **`return`** dans un corps de boucle.
- Boucle **`for`** (sans **`else`**) et itération sur `range(`*debut*`, `*arrêt*`)`, une liste.
- Définition d'une fonction **`def`** f`(`p_1`, ..., `p_n`)`, **`return`**.

Divers

- Introduction d'un commentaire avec **`#`**.
- Utilisation simple de `print` pour afficher une ou plusieurs expressions, sans paramètre optionnel.
- Importation de modules avec
  `import` *module*, **`import`** *module* **`as`** *alias*, **`from`** *module* **`import`** $f, g, \ldots$
- Assertion : **`assert`** (sans message d'erreur).