CONSEIL SUPÉRIEUR DES PROGRAMMES

Numérique et sciences informatiques

Classe terminale, enseignement de spécialité, voie générale

Mai 2019

Sommaire

Préambule

L'enseignement de spécialité de numérique et sciences informatiques du cycle terminal de la voie générale vise l'appropriation des fondements de l'informatique pour préparer les élèves à une poursuite d'études en les formant à la pratique d'une démarche scientifique et en développant leur appétence pour des activités de recherche.

L'objectif de cet enseignement général est l'appropriation des concepts et des méthodes qui fondent l'informatique, dans ses dimensions scientifiques et techniques. Il s'appuie sur l'universalité de quatre concepts fondamentaux et la variété de leurs interactions :

- les **données**, qui représentent sous une forme numérique unifiée des informations très diverses : textes, images, sons, mesures physiques, sommes d'argent, etc. ;
- les **algorithmes**, qui spécifient de façon abstraite et précise des traitements à effectuer sur les données à partir d'opérations élémentaires ;
- les **langages**, qui permettent de traduire les algorithmes abstraits en **programmes** textuels ou graphiques de façon à ce qu'ils soient exécutables par les machines ;
- les **machines**, et leurs systèmes d'exploitation, qui permettent d'exécuter des programmes en enchaînant un grand nombre d'instructions simples, assurent la persistance des données par leur stockage et gèrent les communications. Y sont inclus les **objets connectés** et les **réseaux**.

À ces concepts s'ajoute un élément transversal : les **interfaces** qui permettent la communication, la collecte des données et la commande des systèmes.

Cet enseignement prolonge les enseignements d'informatique dispensés à l'école primaire, au collège en mathématiques et en technologie et, en classe de seconde, l'enseignement commun Sciences numériques et technologie. Il s'appuie aussi sur l'algorithmique pratiquée en mathématiques en classe de seconde. Il approfondit les notions étudiées et les compétences travaillées en classe de première dans l'enseignement de spécialité. Il autorise tous les choix de couplage avec les autres spécialités.

L'enseignement de spécialité de numérique et sciences informatiques permet de développer les compétences suivantes, constitutives de la pensée informatique :

- analyser et modéliser un problème en termes de flux et de traitement d'informations ;
- décomposer un problème en sous-problèmes, reconnaître des situations déjà analysées et réutiliser des solutions ;
- concevoir des solutions algorithmiques ;
- traduire un algorithme dans un langage de programmation, en spécifier les interfaces et les interactions, comprendre et réutiliser des codes sources existants, développer des processus de mise au point et de validation de programmes ;
- mobiliser les concepts et les technologies utiles pour assurer les fonctions d'acquisition, de mémorisation, de traitement et de diffusion des informations ;
- développer des capacités d'abstraction et de généralisation.

Cet enseignement se déploie en mettant en activité les élèves, **sous des formes variées** qui permettent de développer des compétences transversales :

- faire preuve d'autonomie, d'initiative et de créativité ;
- présenter un problème ou sa solution, développer une argumentation dans le cadre d'un débat ;
- coopérer au sein d'une équipe dans le cadre d'un projet ;
- rechercher de l'information, partager des ressources ;
- faire un usage responsable et critique de l'informatique.

La progression peut suivre un rythme annuel construit autour de périodes spécifiques favorisant une alternance entre divers types d'activités.

L'enseignement de numériques et sciences informatiques permet l'acquisition des compétences numériques qui font l'objet d'une certification en fin de cycle terminal. Comme tous les enseignements de spécialité, il contribue au développement des compétences orales à travers notamment la pratique de l'argumentation. Celle-ci conduit à préciser sa pensée et à expliciter son raisonnement de manière à convaincre. Elle permet à chacun de faire évoluer sa pensée, jusqu'à la remettre en cause si nécessaire, pour accéder progressivement à la vérité par la preuve.

Démarche de projet

Un enseignement d'informatique ne saurait se réduire à une présentation de concepts ou de méthodes sans permettre aux élèves de se les approprier en développant des projets.

Un tiers au moins de l'horaire total de la spécialité est réservé à la conception et à l'élaboration de projets conduits par des petits groupes d'élèves.

Les projets réalisés par les élèves, sous la conduite du professeur, constituent un apprentissage fondamental tant pour l'appropriation des concepts informatiques que pour l'acquisition de compétences. En classe de première comme en classe terminale, ils peuvent porter sur des problématiques issues d'autres disciplines et ont essentiellement pour but d'imaginer des solutions répondant à un problème ; dans la mesure du possible, il convient de laisser le choix du thème du projet aux élèves. Il peut s'agir d'un approfondissement théorique des concepts étudiés en commun, d'une application à d'autres disciplines telle qu'une simulation d'expérience, d'exploitation de modules liés à l'intelligence artificielle et en particulier à l'apprentissage automatique, d'un travail sur des données socioéconomiques, du développement d'un logiciel de lexicographie, d'un projet autour d'un objet connecté ou d'un robot, de la conception d'une bibliothèque implémentant une structure de données complexe, d'un problème de traitement d'image ou de son, d'une application mobile, par exemple de réalité virtuelle ou augmentée, du développement d'un site *Web* associé à l'utilisation d'une base de données, de la réalisation d'un interpréteur d'un mini-langage, de la recherche d'itinéraire sur une carte (algorithme A*), d'un programme de jeu de stratégie, etc.

La conduite d'un projet inclut des points d'étape pour faire un bilan avec le professeur, valider des éléments, contrôler l'avancement du projet ou en adapter les objectifs, voire le redéfinir partiellement, afin de maintenir la motivation des élèves.

Les professeurs veillent à ce que les projets restent d'une ambition raisonnable afin de leur permettre d'aboutir.

Modalités de mise en œuvre

Les activités pratiques et la réalisation de projets supposent que chaque élève ait un accès individuel à un équipement relié à internet.

Un langage de programmation est nécessaire pour l'écriture des programmes : un langage simple d'usage, interprété, concis, libre et gratuit, multiplateforme, largement répandu, riche de bibliothèques adaptées et bénéficiant d'une vaste communauté d'auteurs dans le monde éducatif est à privilégier. Au moment de la conception de ce programme, le langage choisi est Python version 3 (ou supérieure).

L'expertise dans tel ou tel langage de programmation n'est cependant pas un objectif de formation.

Éléments de programme

Le programme, organisé en six rubriques, ne constitue pas un plan de cours. Il appartient aux professeurs de choisir leur progression, sans faire de chaque partie un tout insécable et indépendant des autres. Les mêmes notions peuvent être développées et éclairées dans différentes rubriques et leurs interactions mises en évidence.

Histoire de l'informatique

Cette rubrique transversale se décline dans chacune des cinq autres.

Comme tous les concepts scientifiques et techniques, ceux de l'informatique ont une histoire et ont été forgés par des personnes. Les algorithmes sont présents dès l'Antiquité, les machines à calculer apparaissent progressivement au XVII^{e} siècle, les sciences de l'information sont fondées au XIX^{e} siècle, mais c'est en 1936 qu'apparaît le concept de machine universelle, capable d'exécuter tous les algorithmes, et que les notions de machine, algorithme, langage et information sont pensées comme un tout cohérent. Les premiers ordinateurs ont été construits en 1948 et leur puissance a ensuite évolué exponentiellement. Parallèlement, les ordinateurs se sont diversifiés dans leurs tailles, leurs formes et leurs emplois : téléphones, tablettes, montres connectées, ordinateurs personnels, serveurs, fermes de calcul, méga-ordinateurs. Le réseau internet, développé depuis 1969, relie aujourd'hui ordinateurs et objets connectés.

Contenus	Capacités attendues	Commentaires
Événements clés de l'histoire de l'informatique.	Situer dans le temps les principaux événements de l'histoire de l'informatique et leurs protagonistes. Identifier l'évolution des rôles relatifs des logiciels et des matériels.	Ces repères viennent compléter ceux qui ont été introduits en première. Ces repères historiques sont construits au fur et à mesure de la présentation des concepts et techniques.

Structures de données

L'écriture sur des exemples simples de plusieurs implémentations d'une même structure de données permet de faire émerger les notions d'interface et d'implémentation, ou encore de structure de données abstraite. Le paradigme de la programmation objet peut être utilisé pour réaliser des implémentations effectives des structures de données, même si ce n'est pas la seule façon de procéder.

Le lien est établi avec la notion de modularité qui figure dans la rubrique « langages et programmation » en mettant en évidence l'intérêt d'utiliser des bibliothèques ou des API *(Application Programming Interface)*.

Contenus	Capacités attendues	Commentaires
Structures de données, interface et implémentation.	Spécifier une structure de données par son interface. Distinguer interface et implémentation. Écrire plusieurs implémentations d'une même structure de données.	L'abstraction des structures de données est introduite après plusieurs implémentations d'une structure simple comme la file (avec un tableau ou avec deux piles).
Vocabulaire de la programmation objet : classes, attributs, méthodes, objets.	Écrire la définition d'une classe. Accéder aux attributs et méthodes d'une classe.	On n'aborde pas ici tous les aspects de la programmation objet comme le polymorphisme et l'héritage.
Listes, piles, files : structures linéaires. Dictionnaires, index et clé.	Distinguer des structures par le jeu des méthodes qui les caractérisent. Choisir une structure de données adaptée à la situation à modéliser. Distinguer la recherche d'une valeur dans une liste et dans un dictionnaire.	On distingue les modes FIFO *(first in first out)* et LIFO *(last in first out)* des piles et des files. Les listes n'existent pas de façon native en Python.
Arbres : structures hiérarchiques. Arbres binaires : nœuds, racines, feuilles, sous-arbres gauches, sous-arbres droits.	Identifier des situations nécessitant une structure de données arborescente. Évaluer quelques mesures des arbres binaires (taille, encadrement de la hauteur, etc.).	On fait le lien avec la rubrique « algorithmique ».
Graphes : structures relationnelles. Sommets, arcs, arêtes, graphes orientés ou non orientés.	Modéliser des situations sous forme de graphes. Écrire les implémentations correspondantes d'un graphe : matrice d'adjacence, liste de successeurs/de prédécesseurs. Passer d'une représentation à une autre.	On s'appuie sur des exemples comme le réseau routier, le réseau électrique, internet, les réseaux sociaux. Le choix de la représentation dépend du traitement qu'on veut mettre en place : on fait le lien avec la rubrique « algorithmique ».

Bases de données

Le développement des traitements informatiques nécessite la manipulation de données de plus en plus nombreuses. Leur organisation et leur stockage constituent un enjeu essentiel de performance. Le recours aux bases de données relationnelles est aujourd'hui une solution très répandue. Ces bases de données permettent d'organiser, de stocker, de mettre à jour et d'interroger des données structurées volumineuses utilisées simultanément par différents programmes ou différents utilisateurs. Cela est impossible avec les représentations tabulaires étudiées en classe de première.

Des systèmes de gestion de bases de données (SGBD) de très grande taille (de l'ordre du pétaoctet) sont au centre de nombreux dispositifs de collecte, de stockage et de production d'informations. L'accès aux données d'une base de données relationnelle s'effectue grâce à des requêtes d'interrogation et de mise à jour qui peuvent par exemple être rédigées dans le langage SQL (*Structured Query Language*). Les traitements peuvent conjuguer le recours au langage SQL et à un langage de programmation.

Il convient de sensibiliser les élèves à un usage critique et responsable des données.

Contenus	Capacités attendues	Commentaires
Modèle relationnel : relation, attribut, domaine, clef primaire, clef étrangère, schéma relationnel.	Identifier les concepts définissant le modèle relationnel.	Ces concepts permettent d'exprimer les contraintes d'intégrité (domaine, relation et référence).
Base de données relationnelle.	Savoir distinguer la structure d'une base de données de son contenu. Repérer des anomalies dans le schéma d'une base de données.	La structure est un ensemble de schémas relationnels qui respecte les contraintes du modèle relationnel. Les anomalies peuvent être des redondances de données ou des anomalies d'insertion, de suppression, de mise à jour. On privilégie la manipulation de données nombreuses et réalistes.
Système de gestion de bases de données relationnelles.	Identifier les services rendus par un système de gestion de bases de données relationnelles : persistance des données, gestion des accès concurrents, efficacité de traitement des requêtes, sécurisation des accès.	Il s'agit de comprendre le rôle et les enjeux des différents services sans en détailler le fonctionnement.
Langage SQL : requêtes d'interrogation et de mise à jour d'une base de données.	Identifier les composants d'une requête. Construire des requêtes d'interrogation à l'aide des clauses	On peut utiliser DISTINCT, ORDER BY ou les fonctions d'agrégation sans utiliser les clauses GROUP BY et HAVING.

	du langage SQL : SELECT, FROM, WHERE, JOIN. Construire des requêtes d'insertion et de mise à jour à l'aide de : UPDATE, INSERT, DELETE.	

Architectures matérielles, systèmes d'exploitation et réseaux

La réduction de taille des éléments des circuits électroniques a conduit à l'avènement de systèmes sur puce (*SoCs* pour *Systems on Chips* en anglais) qui regroupent dans un seul circuit nombre de fonctions autrefois effectuées par des circuits séparés assemblés sur une carte électronique. Un tel système sur puce est conçu et mis au point de façon logicielle, ses briques électroniques sont accessibles par des APIs, comme pour les bibliothèques logicielles.

Toute machine est dotée d'un système d'exploitation qui a pour fonction de charger les programmes depuis la mémoire de masse et de lancer leur exécution en leur créant des processus, de gérer l'ensemble des ressources, de traiter les interruptions ainsi que les entrées-sorties et enfin d'assurer la sécurité globale du système.

Dans un réseau, les routeurs jouent un rôle essentiel dans la transmission des paquets sur internet : les paquets sont routés individuellement par des algorithmes. Les pertes logiques peuvent être compensées par des protocoles reposant sur des accusés de réception ou des demandes de renvoi, comme TCP.

La protection des données sensibles échangées est au cœur d'internet. Les notions de chiffrement et de déchiffrement de paquets pour les communications sécurisées sont explicitées.

Contenus	Capacités attendues	Commentaires
Composants intégrés d'un système sur puce.	Identifier les principaux composants sur un schéma de circuit et les avantages de leur intégration en termes de vitesse et de consommation.	Le circuit d'un téléphone peut être pris comme un exemple : microprocesseurs, mémoires locales, interfaces radio et filaires, gestion d'énergie, contrôleurs vidéo, accélérateur graphique, réseaux sur puce, etc.
Gestion des processus et des ressources par un système d'exploitation.	Décrire la création d'un processus, l'ordonnancement de plusieurs processus par le système. Mettre en évidence le risque de l'interblocage *(deadlock)*.	À l'aide d'outils standard, il s'agit d'observer les processus actifs ou en attente sur une machine. Une présentation débranchée de l'interblocage peut être proposée.

Protocoles de routage.	Identifier, suivant le protocole de routage utilisé, la route empruntée par un paquet.	En mode débranché, les tables de routage étant données, on se réfère au nombre de sauts (protocole RIP) ou au coût des routes (protocole OSPF). Le lien avec les algorithmes de recherche de chemin sur un graphe est mis en évidence.
Sécurisation des communications.	Décrire les principes de chiffrement symétrique (clef partagée) et asymétrique (avec clef privée/clef publique). Décrire l'échange d'une clef symétrique en utilisant un protocole asymétrique pour sécuriser une communication HTTPS.	Les protocoles symétriques et asymétriques peuvent être illustrés en mode débranché, éventuellement avec description d'un chiffrement particulier. La négociation de la méthode chiffrement du protocole SSL (*Secure Sockets Layer*) n'est pas abordée.

Langages et programmation

Le travail entrepris en classe de première sur les méthodes de programmation est prolongé. L'accent est mis sur une programmation assurant une meilleure sûreté, c'est-à-dire minimisant le nombre d'erreurs. Parallèlement, on montre l'universalité et les limites de la notion de calculabilité.

La récursivité est une méthode fondamentale de programmation. Son introduction permet également de diversifier les algorithmes étudiés. En classe terminale, les élèves s'initient à différents paradigmes de programmation pour ne pas se limiter à une démarche impérative.

Contenus	Capacités attendues	Commentaires
Notion de programme en tant que donnée. Calculabilité, décidabilité.	Comprendre que tout programme est aussi une donnée. Comprendre que la calculabilité ne dépend pas du langage de programmation utilisé. Montrer, sans formalisme théorique, que le problème de l'arrêt est indécidable.	L'utilisation d'un interpréteur ou d'un compilateur, le téléchargement de logiciel, le fonctionnement des systèmes d'exploitation permettent de comprendre un programme comme donnée d'un autre programme.
Récursivité.	Écrire un programme récursif. Analyser le fonctionnement d'un programme récursif.	Des exemples relevant de domaines variés sont à privilégier.

Modularité.	Utiliser des API (*Application Programming Interface*) ou des bibliothèques. Exploiter leur documentation. Créer des modules simples et les documenter.	
Paradigmes de programmation.	Distinguer sur des exemples les paradigmes impératif, fonctionnel et objet. Choisir le paradigme de programmation selon le champ d'application d'un programme.	Avec un même langage de programmation, on peut utiliser des paradigmes différents. Dans un même programme, on peut utiliser des paradigmes différents.
Mise au point des programmes. Gestion des bugs.	Dans la pratique de la programmation, savoir répondre aux causes typiques de bugs : problèmes liés au typage, effets de bord non désirés, débordements dans les tableaux, instruction conditionnelle non exhaustive, choix des inégalités, comparaisons et calculs entre flottants, mauvais nommage des variables, etc.	On prolonge le travail entrepris en classe de première sur l'utilisation de la spécification, des assertions, de la documentation des programmes et de la construction de jeux de tests. Les élèves apprennent progressivement à anticiper leurs erreurs.

Algorithmique

Le travail de compréhension et de conception d'algorithmes se poursuit en terminale notamment via l'introduction des structures d'arbres et de graphes montrant tout l'intérêt d'une approche récursive dans la résolution algorithmique de problèmes.

On continue l'étude de la notion de coût d'exécution, en temps ou en mémoire et on montre l'intérêt du passage d'un coût quadratique en n^2 à $n \log_2 n$ ou de n à $\log_2 n$. Le logarithme en base 2 est ici manipulé comme simple outil de comptage (taille en bits d'un nombre entier).

Contenus	Capacités attendues	Commentaires
Algorithmes sur les arbres binaires et sur les arbres binaires de recherche.	Calculer la taille et la hauteur d'un arbre. Parcourir un arbre de différentes façons (ordres infixe, préfixe ou suffixe ; ordre en largeur d'abord). Rechercher une clé dans un arbre de recherche, insérer une clé.	Une structure de données récursive adaptée est utilisée. L'exemple des arbres permet d'illustrer la programmation par classe. La recherche dans un arbre de recherche équilibré est de coût logarithmique.

Algorithmes sur les graphes.	Parcourir un graphe en profondeur d'abord, en largeur d'abord. Repérer la présence d'un cycle dans un graphe. Chercher un chemin dans un graphe.	Le parcours d'un labyrinthe et le routage dans internet sont des exemples d'algorithme sur les graphes. L'exemple des graphes permet d'illustrer l'utilisation des classes en programmation.
Méthode « diviser pour régner ».	Écrire un algorithme utilisant la méthode « diviser pour régner ».	La rotation d'une image bitmap d'un quart de tour avec un coût en mémoire constant est un bon exemple. L'exemple du tri fusion permet également d'exploiter la récursivité et d'exhiber un algorithme de coût en $n \log_2 n$ dans les pires des cas.
Programmation dynamique.	Utiliser la programmation dynamique pour écrire un algorithme.	Les exemples de l'alignement de séquences ou du rendu de monnaie peuvent être présentés. La discussion sur le coût en mémoire peut être développée.
Recherche textuelle.	Étudier l'algorithme de Boyer-Moore pour la recherche d'un motif dans un texte.	L'intérêt du prétraitement du motif est mis en avant. L'étude du coût, difficile, ne peut être exigée.