ALGORITHMIQUE APPLIQUÉE

L'objectif de ce module est de construire des algorithmes, et de les programmer dans un langage informatique, dans le but de résoudre des problèmes de niveau raisonnable.
Les thématiques abordées lors de l'étude de ce module sont très ouvertes, mais l'objectif visé, à l'intérieur du processus de conception, est ciblé. On veille à ce que les situations proposées soient mathématiquement achevées. Alors qu'une solution, peut être décrite de manière très libre, textuelle ou graphique, par formules ou symboles, par l'exemple ou de manière inachevée, on s'attache ici à les exprimer en utilisant les outils algorithmiques usuels, pour les rendre directement convertibles et exécutables sur machine.
Afin de faciliter la compréhension des mécanismes et la détection d'éventuelles erreurs, il est impératif de concrétiser les algorithmes par l'emploi d'un langage de programmation et de conduire l'étudiant à réaliser des tests. La tâche inverse, consistant à comprendre un algorithme, présente également un grand intérêt pour l'assimilation des mécanismes et lors d'opérations de maintenance.
Les sujets empruntés à la vie courante peuvent être utilisés à chaque fois qu'ils permettent d'illustrer un mécanisme simple avec pertinence. Sinon, on préfère utiliser des sujets dérivés directement d'autres modules mathématiques et, avec un certain équilibre, des sujets associés à des thématiques informatiques (par exemple : codage, cryptage et décryptage, redondance de sécurité, tri itératif et tri récursif, parcours de graphes). Ces sujets peuvent être abordés afin d'illustrer des concepts fondamentaux d'algorithmique sans qu'aucune connaissance spécifique ne soit exigible de l'étudiant dans ces derniers domaines.

Ce module vise à développer les compétences spécifiques suivantes :

– comprendre un algorithme et expliquer ce qu'il fait ;
– modifier un algorithme existant pour obtenir un résultat différent ;
– concevoir une procédure, un algorithme simple ;
– transcrire un algorithme dans un langage informatique ;
– s'interroger sur l'efficacité d'un algorithme.

Les contrôles d'exécution constituent le cœur des mécanismes algorithmiques de base. À ce titre, on attache un soin tout particulier à leur étude progressive mais détaillée. Leur maîtrise pratique est essentielle et les évaluations doivent être centrées sur eux.

Pour l'écriture des algorithmes, une représentation textuelle convenablement indentée avec des indicateurs de début et de fin explicites facilite la lecture. Pour aider à la compréhension, il est utile également d'indiquer un en-tête composé d'un nom, d'un rôle, de l'indication des données d'entrée et de sortie. Des commentaires sont ajoutés, notamment pour préciser le rôle des variables ou fournir des indications méthodologiques.

Un algorithme est indépendant de tout langage de programmation ; aussi aucun langage n'est imposé, mais il convient de s'assurer qu'il est accepté par le centre d'examen. On privilégie un langage de programmation simple d'utilisation et libre d'installation. L'existence d'une communauté d'utilisateurs et de bibliothèques facilite le développement.

La présentation linéaire du programme, avec une entrée par les contenus n'indique pas d'ordre dans sa mise en œuvre. Aussi les concepts fondamentaux (algorithme, finitude, modularité, identifiant, type, constante, variable, fonction, procédure, expression numérique, expression conditionnelle et plus généralement booléenne…) sont acquis par l'usage, sans faire appel à des définitions formelles préalables.

CONTENUS	CAPACITÉS ATTENDUES	COMMENTAIRES
Types de données Types simples : entier naturel, entier relatif, réel, booléen. Chaîne de caractères.	• Connaître les modes de codage et gérer les différences entre données mathématiques et informatiques : – domaine de valeurs ; – représentation exacte ou approchée, précision.	Il n'est pas nécessaire de connaître la représentation exacte des données en machine, notamment en ce qui concerne les flottants. On évite de considérer une chaîne comme un tableau de caractères.
Tableaux de données : – de type homogène à une ou deux dimensions ; – à deux dimensions dans lequel, soit les lignes soit les colonnes, peuvent être de types différents. Procédure et fonction : – paramètres d'entrée ; – valeur(s) retournée(s) par une fonction ; – variables globales ou locales.	• Construire un tableau. • Traiter les données d'un tableau : – accéder à ses différents éléments en lecture et en écriture ; – traiter les éléments d'une ligne ou d'une colonne ; – copier un tableau. • Gérer les transferts en entrée seule, en sortie seule, en entrée et sortie. • Utiliser les variables globales et locales.	On adapte la construction et l'exploitation de ces tableaux aux possibilités de l'outil informatique utilisé. Les structures de données et les objets ne sont pas au programme de mathématiques : ils figurent dans ceux des enseignements professionnels. Sans aborder la programmation objet, les concepts de modularité et d'interface doivent être connus.
Instructions élémentaires Lecture, écriture. Affectation, affectation récursive. **Opérateurs** Opérateurs numériques : addition, soustraction, multiplication, division, exponentiation, quotient et reste de la division entière, signe. Fonctions mathématiques usuelles.	• Saisir une donnée depuis le clavier, manipuler les variables et afficher sur l'écran. • Gérer la chronologie des valeurs contenues dans les variables et produire la trace d'un algorithme. • Transformer une expression mathématique en expression numérique en ligne et réciproquement.	Les fichiers ne sont pas au programme de mathématiques. En standard, la mise en forme des affichages est limitée à l'utilisation de séparateurs et de changements de ligne. Sinon, les outils nécessaires sont fournis et décrits. La gestion des pointeurs n'est pas au programme. « Fonctions mathématiques usuelles » doit être entendu au sens informatique et inclure les fonctions d'arrondi, ainsi qu'un générateur de nombres pseudo-aléatoires uniforme dans un intervalle.

Opérateurs de comparaison : =, <> ou !=, <, <=, >, >=. Opérateurs booléens : non, et, ou, oux.	• Traduire la condition (éventuellement composée) d'une itération (tant que / répéter jusqu'à ce que) ou d'une alternative (si) sous forme d'expression booléenne.	On compare soit des valeurs numériques, soit des chaînes de caractères.
Opérateurs booléens bit à bit.	• Appliquer des opérations booléennes sur les bits.	On interprète notamment en termes de masque, de mise à un, de mise à zéro, de changement d'état.
Opérateur de chaînes : concaténation. Fonctions permettant l'extraction en début, milieu ou fin, la recherche d'un motif.	• Construire et manipuler des chaînes et construire les messages affichés à l'écran.	L'usage d'expressions régulières simples est possible, mais l'étude des expressions régulières est hors programme.
Transtypage	• Gérer les différents types de données et effectuer des conversions d'un type vers un autre.	D'autres instructions, fonctions ou procédures peuvent être introduites dans l'écriture d'algorithmes. Les descriptions sémantiques et syntaxiques précises sont alors mises à disposition de l'étudiant.
Structures de contrôle et d'exécution. Exécution séquentielle. Exécution à structure conditionnelle (si-alors-sinon). Exécution à structure itérative (pour) et (tant que / répéter jusqu'à ce que).	• Mettre en œuvre ces structures.	Afin d'en maîtriser le fonctionnement, les structures d'exécution sont elles-mêmes présentées sous forme d'algorithmes.
Construction des structures itératives : raisonnement par récurrence, initialisation, mise à jour itérative, calcul itératif, mise en forme finale.	• Gérer une itération en distinguant la préparation, l'itération elle-même et la mise en forme finale.	Le raisonnement par récurrence n'a pas à être évalué pour des démonstrations. Il est introduit pour servir de base à une construction des itérations. Le calcul itératif est souvent récursif

Symboles $\sum_{i=}$ et $\prod_{i=}$, traduction algorithmique.	• Gérer une somme ou un produit d'un nombre variable d'opérandes dépendant d'un paramètre.	Généralement la variable affectée récursivement est initialisée à l'élément neutre de l'opération utilisée, avant d'entrer dans l'itération.
Structures imbriquées	• Gérer des structures imbriquées.	On traite également des exemples où les éléments de contrôle des structures internes dépendent de ceux des structures externes. Le nombre d'imbrications n'est pas limité, sauf pour les itérations en dépendance, où on se limite à deux. On évite les excès de complexité, ainsi que les constructions ne correspondant pas à un besoin concret.
	• Traiter un produit matriciel.	La formule de calcul des coefficients d'un produit est donnée.
Récursivité. Nécessité d'un test. Nécessité de cas particuliers résolus sans appel à la récursivité. Finitude.	• Mettre en œuvre ou exploiter une fonction ou une procédure récursive simple.	On peut traiter des exemples de récursivité terminale, de conversion en algorithme non récursif, de récursivité non terminale. On ne présente pas de récursivité mutuelle entre plusieurs procédures.
Analyse d'algorithmes		Ce paragraphe se veut être une simple sensibilisation aux notions de complexité, de correction, de recherche d'erreur et de finalité d'un algorithme. En cas d'évaluation portant sur l'un de ces thèmes, on apporte suffisamment d'indications pour limiter les prérequis.
Notions de complexité temporelle et spatiale.	• Calculer un nombre minimal ou maximal d'opérations significatives ou la taille globale de données choisies.	On présente des variantes produisant les mêmes résultats avec des complexités très différentes. Aucune connaissance théorique n'est exigible.
Validation et débogage.	• Procéder à un suivi de variables par la production d'une trace ou l'utilisation de jeux d'essai pour le test d'un algorithme et la recherche d'erreur.	On propose des algorithmes délibérément erronés dont les défauts sont repérés puis débogués. Selon le langage choisi, on peut montrer les fonctions permettant

		le suivi de variables, le débogage pas à pas**.** Afin de mieux sensibiliser à certains risques, on peut présenter des cas d'effets indésirables (effets de bord, évaluation partielle lors de calcul d'expressions booléennes, débordements ou approximations numériques, transtypage, utilisation d'indices hors domaine,…) et leurs conséquences spectaculaires. Aucune théorie n'est au programme.
Interprétation d'algorithmes.	• Savoir reconstituer le rôle d'un algorithme.	L'ajout d'une ou plusieurs procédures ou fonctions à un ensemble interdépendant peut constituer une excellente base. On se limite à des cas simples.