Algorithmique et programmation :
Transition pédagogique de Scratch à Python au lycée

Kit Enseignant

- Des blocs au texte -
Vers une transition pédagogique de Scratch à Python au lycée

PDF interactif

Contextualisation :

Lors de leur entrée en classe de Seconde, les jeunes ont acquis des compétences de base en algorithmique via la programmation en blocs. Les programmes scolaires de mathématiques requièrent un approfondissement et une consolidation de ces compétences, avec une transition vers la découverte de la programmation en Python. Cette transition est essentielle pour permettre aux élèves de découvrir la programmation textuelle à travers le programme de Seconde, tout en consolidant les notions d'algorithmique vues au collège. Il est toutefois complexe pour les enseignants de faire l'état des lieux des compétences effectives de leurs élèves, sachant qu'elles ne sont (pour le moment) pas explicitement testées dans les tests de positionnement de Seconde.

Ce kit Enseignant propose un accompagnement pratique et pédagogique, visant à accompagner les enseignements d'algorithmique et de programmation tout au long de l'année scolaire. Il comprend à la fois une phase de positionnement préalable des compétences acquises par les élèves en programmation par blocs, ainsi que leurs représentations et croyances dans ce domaine, puis propose un cadre d'accompagnement dans la découverte et la consolidation de ces compétences tout au long de la transition vers la programmation textuelle.

Déroulé du kit Enseignant :

Cible : Enseignants de mathématiques, intervenant dans les classes de Seconde générale et technologique.

Public visé : Jeunes scolarisés en classes de Seconde Générale et Technologique.

Matériel nécessaire :

- un ordinateur connecté avec enceintes par élève (ou groupe d'élèves - précisé en fonction des activités) ;
- un vidéoprojecteur ou un TNI pour une utilisation collective.

Durée : tout au long des enseignements d'algorithmique et de programmation.

Ce kit Enseignant est modulaire : chaque module peut être utilisé de manière indépendante, en fonction du besoin de l'enseignant. Il est pensé pour une utilisation en contexte scolaire et péri-scolaire.

Objectifs pédagogiques :

- définir et évaluer les acquis préalables des élèves en algorithmique et en programmation (**Modules 1 et 2**) ;
- proposer un module de positionnement permettant d'évaluer le niveau des élèves (nombre de tentatives nécessaires pour compléter une activité) tout en favorisant une prise en main de l'interface de la ressource Citizen Code Python (**Module 2**) ;
- faciliter l'acquisition et la consolidation des notions d'algorithmique et de programmation dans le programme de mathématiques de seconde (**Module 3**).
- proposer une progression pédagogique adaptée au travers d'activités présélectionnées favorisant l'accompagnement des élèves dans leurs apprentissages tout au long de l'année scolaire (**Module 4**).

Module 1 : Rappel des acquis en programmation des élèves à l'issue du cycle 4

Attendu général des compétences supposées acquises en programmation à l'issue du cycle 4 (Source : eduscol) :

Écrire, mettre au point et exécuter un programme simple

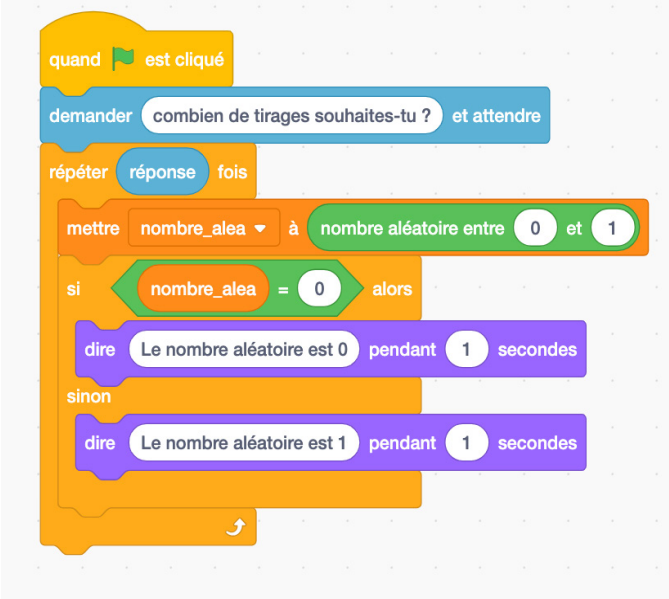
Cet attendu général atteignable à l'issue du cycle 4 peut être décomposé en trois niveaux de compétences en algorithmique et en programmation informatique (de complexité croissante), comme détaillés dans le tableau ci-dessous.

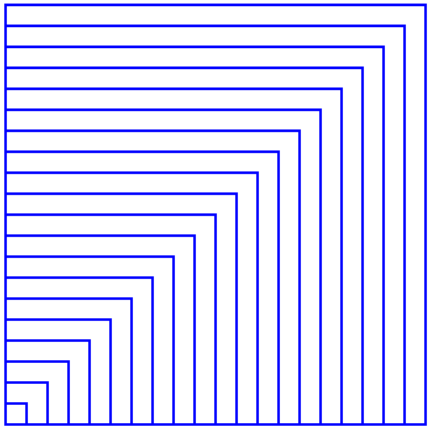
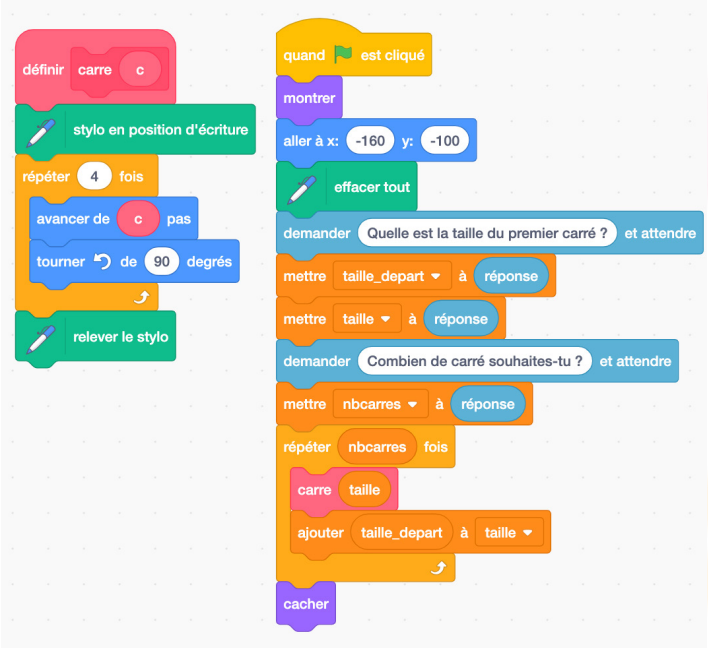
En théorie, le programme d'algorithmique et programmation de Seconde reprend à partir des acquis développés par les élèves tout au long du collège. L'enseignant de mathématiques devrait ainsi pouvoir s'appuyer sur ces compétences préalables pour appliquer le programme. Toutefois, il est important de prendre en considération quelques nuances majeures :

- le niveau attendu est un idéal théorique et ne correspond donc pas nécessairement au niveau effectif des élèves - des disparités de niveau peuvent également être observées entre les élèves d'une même classe ;
- le changement de paradigme — marqué par le passage de la programmation par blocs à la programmation textuelle (Python) — peut occulter des compétences algorithmiques pourtant présentes chez les élèves dans un contexte de programmation par blocs ;
- la transition de la programmation par blocs à la programmation textuelle requiert également des ajustements de connaissances ainsi que l'acquisition d'une nouvelle syntaxe.

Ce parcours exploratoire permet de répondre à ces considérations, afin de bien démarrer l'année de Seconde.

Tableau récapitulatif des niveaux de compétences progressifs en programmation par blocs sous-jacent à l'attendu en fin de cycle 4

Niveau	Compétences	Exemple d'activité mathématique	Exemple de programme informatique sur Scratch
Niveau 1	L'élève réalise des activités débranchées d'algorithmique, organise des blocs fournis pour créer un programme simple (ex. : déplacement géométrique avec Scratch).	L'élève peut utiliser Scratch pour produire seul un programme de construction d'un triangle équilatéral, en utilisant la boucle : 	Il pourra alors produire ce programme sur Scratch. 
Niveau 2	L'élève est capable de gérer et déclencher des événements, de créer un programme comprenant une séquence d'instructions conditionnelles dans une boucle de répétition et d'intégrer des variables.	L'élève programme des tirages aléatoires dont le nombre est choisi par l'utilisateur en gérant un affichage conditionnel. Il pourra alors produire sur Scratch un programme comme celui-ci.	

Niveau	Compétences	Exemple d'activité mathématique	Exemple de programme informatique sur Scratch
Niveau 3	Décomposition de problèmes complexes, création de blocs personnalisés, utilisation de boucles imbriquées et conditions complexes.	L'élève reproduit une frise donnée reproduisant un motif grâce à un bloc personnalisé. 	Il pourra alors produire sur Scratch un programme comme celui-ci. 

Module 2 : *Parcours exploratoire* pour les Élèves de Seconde

Objectifs du parcours exploratoire :

Ce parcours exploratoire permet d'adresser des objectifs complémentaires entre les enseignants et leurs élèves.

Objectifs à destination de l'enseignant

- déterminer le degré des compétences algorithmiques acquises par les élèves à l'entrée en Seconde, ainsi que leur profondeur.
- identifier le positionnement et les représentations préalables des élèves vis-à-vis du numérique et de la programmation informatique (intérêt, sentiment de compétence, motivation, accessibilité).

Objectifs à destination des élèves de Seconde

- mobiliser des compétences de programmation acquises tout au long du cycle 4 ;
- découvrir la ressource Citizen Code et prendre en main son interface et ses paramètres de bases.

Prérequis pédagogiques : **Aucun**

Ce parcours se prête parfaitement à une introduction à l'algorithmique et à la programmation en Seconde. Les élèves s'appuient sur leurs connaissances du collège.

Matériel nécessaire :

- Les élèves peuvent bénéficier **d'un ordinateur (ou d'une tablette) chacun** afin que l'enseignant puisse obtenir des informations personnalisées pour chacun de ses élèves (voir des exemples d'animation plus bas).
- La ressource Citizen Code Python doit être activée par l'enseignante ou l'enseignant RUPN de l'établissement via le Médiacentre de Lycée Connecté.

Remarques :

- En amont de la séance, pensez à assigner le parcours à votre classe en utilisant le guide fourni dans la **fiche synthétique** (*https://api.futureengineer.fr/sites/default/files/fiche_synthetique_v2.pdf*). À noter qu'avec le **Médiacentre**, il vous suffit de demander aux élèves de se connecter une première fois à la ressource pour les voir apparaître ainsi que vos classes.
- Pensez également à faire rejoindre l'ensemble des élèves avant de leur assigner les activités.

Étape 1 : Positionnement et représentations préalables des élèves en amont des activités (5 min)

En amont de la séance, vous pouvez faire compléter le questionnaire Le Code et moi à vos élèves. Ce questionnaire vise à déterminer les représentations préalables des jeunes vis-à-vis du numérique et de la programmation (motivation, accessibilité de la programmation, sentiment de compétence, intérêt pour le numérique et la programmation), mais également d'estimer leur niveau de familiarité avec la programmation textuelle (Python).
Pour accéder au questionnaire, partagez ce lien à vos élèves : *https://survey.tralalere.com/index.php/599358?lang=fr*

Étape 2 : Parcours exploratoire sur la ressource Citizen Code Python (environ 50 min)

Le Parcours Exploratoire utilise le mode **Blockly** de la ressource **Citizen Code Python**, comparable à Scratch. Il propose de mobiliser et d'estimer les compétences de base en programmation par blocs (instructions, boucles, conditions) de vos élèves au travers de cinq activités suivant une pente pédagogique croissante, correspondant aux trois niveaux d'**attendus à l'issue du cycle 4**. Les élèves réalisent les activités assignées **en autonomie et à leur rythme**. L'objectif sous-jacent est de rendre l'élève actif dans la prise en main de l'outil tout en permettant à l'enseignant d'observer ses acquis.

Le détail des activités et des notions de programmation mobilisées est disponible dans le tableau ci-dessous.

Exemple d'animation : Le parcours peut se réaliser en **une séance de 50 minutes**, intégrant un peu de temps pour la connexion en amont.

- **Vous avez un ordinateur par élève.** (séance en groupe, grande salle informatique, BYOD) Il suffit de proposer à vos élèves de se laisser guider par le parcours.
- **Vous n'avez pas un ordinateur par élève.** Il est possible de proposer le Parcours Exploratoire à une moitié de la classe quand l'autre travaille sur une feuille d'exercices. On peut suggérer ici que le travail donné en "débranché" ne soit pas lié à la programmation afin de laisser les deux groupes sur un même pied d'égalité lorsqu'ils démarreront le parcours. Vous pouvez alors inverser les groupes l'heure suivante ou lors de la prochaine séance.

Conseils :

- **Partagez avec vos élèves les deux objectifs de ce parcours :** pour eux, il s'agit de reprendre contact avec la programmation et les notions vues au collège. Pour vous, il s'agit de prendre en compte leurs compétences, mais aussi leurs éventuelles difficultés pour poursuivre les apprentissages de la classe de Seconde.
- **Mettez-les en confiance :** vous pouvez leur rappeler qu'il n'est pas grave d'être bloqué sur une activité. Bien que le parcours soit progressif, vous pouvez proposer à vos élèves d'explorer une autre activité s'ils bloquent trop longtemps.
- **Encouragez vos élèves à persévérer** et rappelez-leur si nécessaire que toutes les programmeuses et tous les programmeurs font face à des "bogues" et que "déboguer" fait partie intégrante de l'expérience.

- **N'hésitez pas à aider** tout élève qui serait bloqué trop longtemps sur une activité en favorisant **un tutorat par un autre élève**, de façon très ponctuelle et avec l'unique objectif de débloquer la situation problématique.

Suivi des productions et évaluation des compétences :

- **Évaluation des compétences algorithmiques :** Les résultats de chacun de vos élèves sont donnés par activité. En particulier, est précisé le nombre d'erreurs, c'est-à-dire, le nombre de tentatives qui leur a été nécessaire pour réussir la mission.
- **Remarque :** Il est normal de s'y prendre à plusieurs reprises lorsque l'on programme. Cependant, un grand nombre de tentatives peut indiquer une difficulté de compréhension de la notion.

Tableau récapitulatif des activités sélectionnées mises en lien avec leurs objectifs pédagogiques et les notions de programmation mobilisées.

La liste des activités assignées est détaillée dans le tableau ci-dessous. Chaque activité permet de mobiliser des notions de programmation de complexité croissante tout en les mettant en perspective avec leurs objectifs pédagogiques.

Activité	Objectifs pédagogiques	Notions de programmation mobilisées
Le gratte-ciel (saison 1, épisode 1) *Consigne : terminer la construction d'un immeuble.*	• prendre en main l'interface ; • découvrir les dispositifs accompagnant la résolution du problème ; • (re)donner confiance à vos élèves.	• séquences d'instructions
Le métro (saison 1, épisode 2) *Consigne : compléter la pièce manquante du métro.*	• tester la maîtrise de la notion de boucle de répétition (for) ; • provoquer une situation de débogage.	• boucle de répétition (ou boucle bornée) • répétition d'une instruction isolée
Le totem (saison 1, épisode 3) *Consigne : réordonner les éléments du totem.*	• résoudre un problème avec l'utilisation d'une boucle de répétition (for).	• boucle de répétition (ou boucle bornée) • répétition d'une séquence d'instructions
La grue (saison 1, épisode 6) *Consigne : compléter la construction d'un immeuble dont le haut est caché.*	• traduire un énoncé textuel en une série d'instructions conditionnelles.	• conditions et instructions conditionnelles • séquences d'instructions
Les enseignes (saison 1, épisode 7) *Consigne : replacer les enseignes des bâtiments au bon endroit.*	• combiner les notions abordées dans les activités précédentes.	• séquences d'instructions • boucles de répétition (for) • conditions et instructions conditionnelles

Étape 3 : Positionnement et représentations des élèves à l'issue du parcours exploratoire (5 min)

À l'issue du parcours exploratoire, vous pouvez faire compléter le questionnaire Le Code et moi à vos élèves. Ce questionnaire vise à estimer l'évolution des représentations des jeunes vis-à-vis du numérique et de la programmation (motivation, accessibilité de la programmation, sentiment de compétence, intérêt pour le numérique et la programmation).
Pour accéder au questionnaire, partagez ce lien à vos élèves : *https://survey.tralalere.com/index.php/599358?lang=fr*

Module 3 : *Activités pour Travailler les Notions de Seconde*

Objectif du module :

Ce module vise à expliciter les correspondances entre les notions de mathématiques du programme de Seconde générale et technologique et les activités proposées dans la ressource Citizen Code permettant de les aborder.

Un récapitulatif des **notions à travailler** en mathématiques et les **activités Citizen Code Python** associées, réparties sur une pente pédagogique, est proposé dans le tableau ci-après.

Pistes d'animation

- **Phase de réflexion en langage naturel :** Pour chaque activité sélectionnée, vous pouvez proposer une phase préliminaire de réflexion à vos élèves en les invitant à **écrire l'algorithme proposé en langage naturel** sur une feuille de papier. Les élèves peuvent alors se lancer dans une phase de recherche et se poser les questions suivantes :
  - Que dois-je obtenir ?
  - Quels objets ai-je à ma disposition pour y arriver ?
  - Comment obtenir ce que je souhaite ?
  - Comment l'écrire pour qu'un ordinateur me comprenne ?

- **Phase de compilation du programme informatique (en Blockly ou en Python) :** La réponse à la dernière question peut permettre de générer l'**algorithme en langage naturel** qu'il suffira alors d'écrire dans la ressource Citizen Code Python pour résoudre l'activité ciblée. Le programme pourra alors montrer des erreurs, mais reposant sur la base d'une première réflexion approfondie des élèves à partir de laquelle ils pourront développer leurs **capacités de résolution de problèmes de façon algorithmique.**

Éléments de progressivité et d'évaluation des compétences :

Vous pouvez proposer aux élèves de **réaliser une activité avec Blockly (en langage blocs) puis avec Python (en langage textuel)**.
NB : si vous assignez les activités à vos élèves, pensez à assigner d'abord une activité avec Blockly puis également la même activité avec Python.

Pour évaluer les connaissances théoriques et pratiques de vos élèves tout au long de l'année scolaire, vous trouverez des questionnaires proposant des courtes résolutions de problèmes, inspirées de la **formation Harward CS50**. Vous pouvez déployer chaque questionnaire comme transition en amont de l'introduction de nouvelles notions.
Ces questionnaires en ligne seront disponibles courant 2025.

Évolution du positionnement et des représentations des élèves à l'issue de l'année scolaire

À l'issue de l'issue scolaire (ou à mi-chemin), vous pouvez faire compléter le questionnaire Le Code et moi à vos élèves. Ce questionnaire vise à estimer l'évolution des représentations des jeunes vis-à-vis du numérique et de la programmation (motivation, accessibilité de la programmation, sentiment de compétence, intérêt pour le numérique et la programmation).
Pour accéder au questionnaire à compléter à 6 mois ou à l'issue de l'année scolaire, partagez ce lien à vos élèves : *https://survey.tralalere.com/index.php/565355?lang=fr*

Tableau récapitulatif des correspondances entre les notions de mathématiques et les activités associées sur la ressource Citizen Code Python

Notions à Travailler	Activités suggérées sur Citizen Code Python	Conseils
Séquences d'instructions	*Saison 1 - épisode 1 (découverte)* *Saison 1 - épisode 2 (approfondissement)*	Ces épisodes ne sont pas nécessaires si vous avez proposé à vos élèves le *parcours exploratoire* présenté ci-dessus. Vous pouvez cependant les proposer à vos élèves avec la consigne de réaliser les missions cette fois en Python (voir les pistes d'animation sous ce tableau) pour une première introduction à ce langage.
Boucles de répétition (for)	*Saison 1 - épisode 3 (découverte)* *Saison 1 - épisode 4 (approfondissement)* *Saison 1 - épisode 5 (consolidation)*	L'épisode 5 propose une complexité plus élevée, non nécessairement attendue en fin de classe de seconde. Réservez-le aux élèves les plus rapides pour mieux gérer l'hétérogénéité de la classe.
Conditions et Instructions conditionnelles	*Saison 1 - épisode 6*	Les deux activités peuvent être réalisées indépendamment. La deuxième est proposée dans le parcours exploratoire. À noter qu'elles font déjà appel à la notion de variable (hauteur de la tour).
Utilisation de variables	*Saison 3 - épisode 1*	Le jeu de l'oie permet d'introduire la notion de variable (et d'assignation de valeurs à une variable) et de mobiliser les notions de programmation introduites précédemment (fonctions, instructions conditionnelles, imbrications et boucles de répétition).
Fonctions et modularité	*Saison 2 - épisode 2 (par exemple "les statues grecques")* *Saison 3 - épisode 1* *Saison 3 - épisode 2*	L'activité "les statues grecques" propose deux niveaux de réalisation. Le premier niveau consiste uniquement en un copier-coller et permet de faire comprendre l'idée de fonction informatique comme un "sous-programme" que l'on peut appeler dans un programme principal. Les activités proposées dans la saison 3 requièrent la mobilisation de l'ensemble des notions abordées préalablement : chaque activité requiert de l'utilisateur qu'il crée sa propre fonction, réutilisable d'activité en activité. Vous pouvez réserver ces activités aux élèves les plus avancés.

Éléments de progressivité et transition progressive de la programmation par blocs à la programmation textuelle :

Lorsque les élèves se sentent prêts, ils peuvent alors progressivement ne réaliser que les missions proposées en Python (la lourdeur des blocs pouvant apparaître à la longue, ce qui est une bonne chose du point de vue de la transition que l'on souhaite les voir réaliser vers le langage textuel).

Au cours de ce module — et tout au long de l'année scolaire —, un tableau de correspondances entre le langage naturel et le langage Python pourra émerger et être complété à chaque nouvelle notion abordée. Ci-dessous, un exemple de ce type de tableau de correspondances vous est proposé

Je veux...	En langage naturel	En langage Python

utiliser **des opérations mathématiques spéciales**	On *appelle* la bibliothèque "math"	`from math import *`
		ou si l'on ne veut importer que la racine, par exemple :
		`from math import sqrt`

multiplier	x	*
diviser	÷	/
mettre en **puissance**	a^n	a**n
écrire une racine carrée	√a	sqrt(a)
acquérir un nombre n de l'utilisateur	demander n	`n = float(input("[message à l'utilisateur]"))`

définir une **fonction**	*Soit f(variable) = [instructions] retourner ...*	`def nom_de_la_fonction(nom_de_la_variable):`
		`  [instructions]`
		`  return [formule]`

afficher quelque chose dans la console (le "shell")	Afficher [ce que l'on veut afficher]	`print("un texte",une variable,...)`
donner une valeur à une variable a	a <- [valeur souhaitée] par exemple a <- a + 1 permet d'ajouter 1 à a	a = [valeur souhaitée] a = a + 1
écrire une égalité	a = b	a == b
écrire une non-égalité	a ≠ b	a != b

Module 4 : *Proposition de progression pédagogique d'algorithmique autour du programme de mathématiques*

Objectif du module :

Ce module vise à proposer une progression pédagogique pour accompagner les enseignants de mathématiques à associer les notions d'algorithmiques de manière spécifique et filée au sein de chacun des chapitres de mathématiques tout en respectant la flexibilité des progressions individuelles.

Cette proposition comprend une progression pédagogique tenant compte à la fois d'une progressivité dans les notions mathématiques abordées, mais également dans les notions algorithmiques mises en jeu. Chaque enseignant peut adapter ces suggestions à sa progression personnelle, en ajoutant ou modifiant les activités en fonction de ses besoins. Un tableau récapitulatif, présenté ci-dessous, permet de relier les chapitres mathématiques et problèmes mathématiques aux notions d'algorithmique et de programmation, accompagnées par des exemples simples de programmes en Python et des applications au sein de la ressource Citizen Code Python.

Chapitre de mathématiques	Problème mathématique	Activité algorithmique Proposée	Notions algorithmiques mises en jeu	Exemples d'algorithmes	Entraînement sur les concepts algorithmiques avec Citizen Code Python
Nombres et calculs	NC1 - Calculer des puissances et racines carrée	Automatiser le calcul des puissances pour une valeur donnée	Entrées-sorties Séquences d'instructions	base = ***int***(***input***("Entrez la base : ")) exposant = ***int***(***input***("Entrez l'exposant : ")) resultat = base ** exposant ***print***(f"{base} à la puissance {exposant} est égal à {resultat}")	Saison 1, Épisode 1 : Séquences d'instructions.
	NC2 -Déterminer si un nombre est un carré parfait.	Demander un nombre à l'utilisateur et vérifier s'il est un carré parfait en comparant la racine carrée arrondie au nombre initial.	Entrées-sorties Instructions conditionnelles	***import*** math nombre = ***int***(***input***("Entrez un nombre : ")) racine = ***int***(***math.sqrt***(nombre)) ***if*** racine * racine == nombre: ***print***(f"{nombre} est un carré parfait.") ***else***: ***print***(f"{nombre} n'est pas un carré parfait.")	Saison 1, Épisode 6 : Instructions conditionnelles.
Introduction aux fonctions	F1 – Introduire les fonctions à deux variables en calculant des volumes.	Créer une fonction Python volume_boite (rayon, hauteur) qui renvoie le volume d'une boîte cylindrique	Fonctions	***def*** volume_boite(rayon, hauteur): ***return math.pi*** * (rayon ** 2) * hauteur rayon = ***float***(***input***("Entrez le rayon : ")) hauteur = ***float***(***input***("Entrez la hauteur : ")) ***print***(f"Le volume de la boîte est : {volume_boite(rayon, hauteur):.2f} cm^3")	Saison 2, Episode 2 : Les fonctions Saison 3, Épisode 1 : Les fonctions.
	F2 – Optimiser le volume d'une boîte créée en découpant des carrés aux quatre coins d'un rectangle.	Créer une fonction volume_boite(x) pour calculer le volume du pavé pour différentes valeurs de x. Afficher les résultats pour permettre à l'utilisateur de repérer le volume maximal.	Fonctions Boucles de répétition (for)	***def*** volume_boite(L, l, x): ***return*** (L - 2 * x) * (l - 2 * x) * x L = ***float***(***input***("Longueur du rectangle : ")) l = ***float***(***input***("Largeur du rectangle : ")) ***print***("Volumes pour différentes valeurs de x :") ***for*** x in range(1, int(min(L, l) / 2)): ***print***(f"x = {x} -> Volume = {volume_boite(L, l, x):.2f} cm^3")	Saison 2, Episode 2 : Les fonctions ou Saison 3, Épisode 1 : Les fonctions.

Chapitre de mathématiques	Problème mathématique	Activité algorithmique Proposée	Notions algorithmiques mises en jeu	Exemples d'algorithmes	Entraînement sur les concepts algorithmiques avec Citizen Code Python
Statistiques descriptives et proportions	S1 – Calculer la moyenne de plusieurs valeurs et déterminer si chaque valeur est au-dessus ou en dessous de la moyenne.	Parcourir un ensemble de données pour calculer la somme puis la moyenne. Utiliser des instructions conditionnelles pour vérifier si chaque valeur est supérieure ou inférieure à la moyenne.	Boucles de répétition (for) Instructions conditionnelles	valeurs = [10, 20, 30, 40, 50] total = 0 ***# Calcul de la moyenne*** ***for*** valeur in valeurs: total ***+=*** valeur moyenne = total / ***len***(valeurs) ***# Vérification de chaque valeur par rapport à la moyenne*** ***for*** valeur in valeurs: ***if*** valeur > moyenne: ***print***(f"{valeur} est au-dessus de la moyenne.") ***else***: ***print***(f"{valeur} est en dessous de la moyenne.")	Saison 1, Épisode 6 : Instructions conditionnelles.
	S2 – Calculer la médiane d'un ensemble de valeurs.	Trier les valeurs et déterminer la médiane.	Conditions (déterminer la médiane en fonction du nombre de valeurs pair ou impair)) Listes (tri)	valeurs = [30, 10, 20, 50, 40] ***valeurs.sort***() ***# Tri des valeurs*** n = ***len***(valeurs) ***if*** n % 2 == 0: mediane = (valeurs[n//2 - 1] + valeurs[n//2]) / 2 ***else***: mediane = valeurs[n//2] ***print***(f"La médiane est : {mediane}")	Saison 1, Épisode 6 : Instructions conditionnelles.
Géométrie repérée et distance	R1 – Calculer la distance entre deux points dans un repère orthonormé.	Calculer la distance entre plusieurs paires de points.	Boucle dans une liste Séquences d'instructions	***import*** math points = [(1, 2, 4, 6), (0, 0, 3, 4), (-1, -1, 1, 1)] ***for*** x1, y1, x2, y2 in points: distance = ***math.sqrt***((x2 - x1) ** 2 + (y2 - y1) ** 2) ***print***(f"La distance entre les points ({x1}, {y1}) et ({x2}, {y2}) est de {distance:.2f}")	Saison 1, Épisode 2 : Répéter une instruction

Chapitre de mathématiques	Problème mathématique	Activité algorithmique Proposée	Notions algorithmiques mises en jeu	Exemples d'algorithmes	Entraînement sur les concepts algorithmiques avec Citizen Code Python
	R2 – Déterminer si trois points sont alignés dans le repère.	Calculer les pentes entre les points pour vérifier l'alignement.	Fonctions Booléens Instructions conditionnelles	***def*** sont_alignes(x1, y1, x2, y2, x3, y3): ***# Calculer les pentes pour vérifier si elles sont égales*** ***if*** (x2 - x1) == 0 or (x3 - x2) == 0: # Éviter la division par zéro ***return*** False pente1 = (y2 - y1) / (x2 - x1) pente2 = (y3 - y2) / (x3 - x2) ***return*** pente1 == pente2 x1, y1 = 1, 2 x2, y2 = 2, 4 x3, y3 = 3, 6 ***if*** sont_alignes(x1, y1, x2, y2, x3, y3): ***print***("Les points sont alignés.") ***else***: ***print***("Les points ne sont pas alignés.")	Saison 2, Episode 2 : Les fonctions Saison 2, Episode 4 : les booléens. Saison 3, Épisode 1 : Combinaison des deux
Fonctions de Référence et Optimisation	FR1 – Représenter la fonction f(x) = x^2 et observer le comportement de la fonction pour différentes valeurs.	Générer des valeurs de f(x) à partir de plusieurs valeurs de x.	Fonctions Librairie matplotlib	***import*** matplotlib.pyplot as plt ***import numpy*** as np ***def*** fonction_carre(x): ***return*** x ** 2 x = ***np.linspace***(-10, 10, 100) y = fonction_carre(x) ***plt.plot***(x, y, ***label***="f(x) = x^2", ***color***='green') ***plt.xlabel***('x') ***plt.ylabel***('f(x)') ***plt.title***('Courbe de la fonction carrée') ***plt.legend***() ***plt.grid***() ***plt.show***()	Saison 2, Episode 2 : Les fonctions

Chapitre de mathématiques	Problème mathématique	Activité algorithmique Proposée	Notions algorithmiques mises en jeu	Exemples d'algorithmes	Entraînement sur les concepts algorithmiques avec Citizen Code Python
	FR2 – Optimiser la surface d'un rectangle en fonction de son périmètre fixé.	Utiliser une technique de balayage pour identifier la largeur *lmax* qui maximise la surface. Parcourir les valeurs de largeur possibles, calculer la surface correspondante, et mettre à jour *lmax* si la surface est supérieure à celle rencontrée jusqu'à présent.	Boucles bornées : utiliser une boucle pour parcourir les largeurs Conditions : mettre à jour lmax lorsque la surface augmente Variables : Utiliser des variables pour stocker la surface maximale et la largeur correspondante	***def*** surface_rectangle(longueur, largeur): ***return*** longueur * largeur perimetre = 20 lmax = 0 surface_max = 0 ***print***("Calcul de la surface maximale pour un périmètre de 20 cm :") ***for*** largeur in range(1, perimetre // 2): longueur = (perimetre / 2) - largeur surface = surface_rectangle(longueur, largeur) ***if*** surface > surface_max: surface_max = surface lmax = largeur ***print***(f"La surface maximale approchée est {surface_max:.2f} cm^2, obtenue pour une largeur de {lmax} cm.")	Saison 3, Épisode 1 : Les fonctions
Vecteurs et colinéarité	V1 – Vérifier si deux droites sont parallèles en utilisant la colinéarité de vecteurs.	Calculer le déterminant des vecteurs et vérifier s'il est nul.	Conditions Fonctions	***def*** sont_paralleles(A, B, C, D): ***# Calculer les composantes des vecteurs AB et CD*** AB_x, AB_y = B[0] - A[0], B[1] - A[1] CD_x, CD_y = D[0] - C[0], D[1] - C[1] ***# Calculer le déterminant*** determinant = AB_x * CD_y - AB_y * CD_x ***return*** determinant == 0 ***# Exemples de points*** A = (1, 2) B = (3, 4) C = (2, 3) D = (4, 5) ***if*** sont_paralleles(A, B, C, D): ***print***("Les droites (AB) et (CD) sont parallèles.") ***else***: ***print***("Les droites (AB) et (CD) ne sont pas parallèles.")	Saison 1, Épisode 6 : Instructions conditionnelles Saison 3, Épisode 1 : Les fonctions

Chapitre de mathématiques	Problème mathématique	Activité algorithmique Proposée	Notions algorithmiques mises en jeu	Exemples d'algorithmes	Entraînement sur les concepts algorithmiques avec Citizen Code Python
	V2 – Comparer les normes de plusieurs vecteurs pour trouver le vecteur de norme maximale.	Définir une fonction vecteur_norme_max(vecteurs) qui prend en entrée une liste de vecteurs, calcule la norme de chaque vecteur et retourne celui de norme maximale.	Fonctions Boucles de répétition Conditions Variables	***import*** math ***def*** vecteur_norme_max(vecteurs): norme_max = 0 vecteur_max = None ***for*** vx, vy in vecteurs: norme = math.sqrt(vx ** 2 + vy ** 2) ***if*** norme > norme_max: norme_max = norme vecteur_max = (vx, vy) ***return*** vecteur_max ***# Exemple d'utilisation avec une liste de vecteurs*** vecteurs = [(2, 3), (4, 1), (5, 6), (3, 7)] vecteur_maximal = vecteur_norme_max(vecteurs) ***print***(f"Le vecteur de norme maximale est {vecteur_maximal}")	Saison 3, Épisode 1 : Les fonctions. (pour aller plus loin : Saison 3, Épisode 4 : Les listes).
Probabilités et simulation	P1 - Simuler un lancer de dé jusqu'à obtenir un nombre cible de "6".	Simuler des lancers de dé jusqu'à obtenir un certain nombre de fois un "6". La boucle while sera utilisée pour effectuer les lancers jusqu'à ce que le nombre cible de "6" soit atteint. Afficher le nombre total de lancers nécessaires.	Boucle *while* (Tant que) Conditions	***import random*** cible = 10 ***# Nombre cible de "6" à atteindre*** succes = 0 ***# Compteur de "6" obtenus*** lancers = 0 ***# Nombre total de lancers*** ***while*** succes < cible: lancer = ***random.randint***(1, 6) lancers ***+=*** 1 ***if*** lancer == 6: succes ***+=*** 1 ***print***(f"Nombre de lancers nécessaires pour obtenir {cible} fois un '6' : {lancers}")	Saison 2, Épisode 3 : Boucles non bornées (while)

Chapitre de mathématiques	Problème mathématique	Activité algorithmique Proposée	Notions algorithmiques mises en jeu	Exemples d'algorithmes	Entraînement sur les concepts algorithmiques avec Citizen Code Python
	P2 - Observer la loi des grands nombres en simulant un tirage de bille avec des fréquences cumulées. Pour cela, simuler le tirage d'une bille dans une urne contenant un certain nombre de billes rouges et bleues.	Utiliser une boucle *for* pour simuler un grand nombre de tirages. À chaque étape, calculer la fréquence cumulée de tirages rouges et observer sa stabilisation. Afficher les fréquences tous les 100 tirages pour visualiser la convergence.	Librairie *random* Fonction : simuler des tirages Boucles et variables : utiliser une boucle pour effectuer les tirages et une variable pour stocker la fréquence cumulative	***import random*** ***def*** simuler_tirages(tirages): rouge = 0 ***# Compteur de billes rouges*** ***for*** i in range(1, tirages + 1): bille = ***random.choice***(["rouge", "bleue"]) ***if*** bille == "rouge": rouge += 1 ***# Afficher la fréquence tous les 100 tirages*** ***if*** i % 100 == 0: frequence = rouge / i ***print***(f"Après {i} tirages, fréquence de 'rouge' : {frequence:.2f}") ***# Fréquence finale après tous les tirages*** frequence_finale = rouge / tirages ***print***(f"Fréquence finale de 'rouge' après {tirages} tirages : {frequence_finale:.2f}") ***return*** frequence_finale ***# Appel de la fonction avec 1000 tirages*** simuler_tirages(1000)	Saison 3, Épisode 1 : Les fonctions.

Sources

Programme de mathématiques du cycle 4,
Repères annuels de progression - mathématique - cycle 4,
Programme de mathématiques - niveau Seconde générale et technologique,
Guide pédagogique de la ressource Citizen Code Python,
Fiche synthétique du mode Enseignant sur Citizen Code Python.