Le Guide canadien de la pensée computationnelle et de la programmation

Une introduction à la littératie algorithmique par Kids Code Jeunesse

Par Juliet Waters

Avec un financement du

Canada

v. 1.2 - Dernière mise à jour : 17/04/2020

v. 1.2 - Dernière mise à jour : 17/04/2020

Introduction

Lorsque j'ai décidé, un beau jour de janvier, de participer à un défi appelé *Code Year* (Année de programmation) dans le cadre de mes résolutions du Nouvel An, je n'étais même pas certaine de savoir ce qu'était la programmation. Mais je voulais en apprendre plus. Une entreprise naissante à mission pédagogique proposait des tutoriels et défis simples pendant 52 semaines. Étant mère d'un préadolescent, les algorithmes qui régissaient de plus en plus notre vie et mon manque de connaissances dans ce domaine m'inquiétaient.

Après quelques minutes à essayer de donner du sens à la ponctuation et à la logique contre-intuitives d'un tutoriel pour écrire « Bonjour le monde » en JavaScript, j'étais complètement dépassée. Sans l'aide de mon fils Ben, j'aurais tout abandonné à ce moment-là. Ben a seulement onze ans, son cerveau s'attache donc moins aux règles de syntaxe et il peut en apprendre de nouvelles plus facilement et rapidement. Grâce à ses conseils, j'ai persévéré malgré ma confusion et ma frustration. Ensuite, j'ai réalisé petit à petit que la programmation pouvait être une activité amusante.

Finalement, j'ai pu mettre à profit mes qualités d'adulte, notamment la persévérance, que j'ai développée durant ma carrière de journaliste. En effet, j'ai retrouvé dans la programmation plusieurs des joies de l'écriture. Les élégantes boucles de logique, le souci du détail, le besoin de maximiser l'impact avec le minimum de mots, le sentiment de pouvoir transformer quelques embryons d'idées en une œuvre complète et fonctionnelle : voilà des joies et des frustrations que je connaissais bien. Une fois les obstacles initiaux derrière moi, je suis rapidement devenue meilleure que Ben pour résoudre des problèmes de programmation plus complexes. Début février, Ben avait déjà abandonné la programmation et s'était tourné vers d'autres disciplines. J'imagine que programmer n'avait plus l'air très *cool* maintenant que sa mère était capable de le faire.

De mon côté, cependant, il n'était plus question d'abandonner. J'avais pris goût à la programmation et je m'améliorais de semaine en semaine. Quelques mois plus tard, alors qu'arrivait l'été, j'ai découvert l'existence des bibliothèques de code. J'ai alors réalisé que la rédaction et la programmation avaient une importante différence : en programmation, je n'avais pas besoin de faire d'efforts pour être originale. En effet, des programmeur·euse·s du monde entier mettent leurs propres programmes à la disposition de tous. Grâce à ces nombreux répertoires de code libres et ouverts, apprendre à programmer devient en quelque sorte une carte de bibliothèque. Une fois

qu'on a appris les protocoles de base et le système, et qu'on sait où sont rangés les artefacts, on a alors accès à un énorme répertoire d'expertise accumulée qu'on peut utiliser à son gré. Dans les trois derniers mois de cette année-là, j'ai appris les bases du langage Python, l'un des langages de programmation ouverts les plus populaires au monde. Je n'ai jamais eu le désir de devenir programmeuse professionnelle, mais j'ai découvert que ce que j'adorais par-dessus tout, c'était d'essayer de comprendre pourquoi les éléments fondamentaux de la programmation étaient si difficiles à apprendre et à enseigner. Je sais maintenant que toute personne qui réussit à passer au travers de ces quelques premières heures déroutantes peut en apprendre suffisamment sur la programmation pour comprendre qu'il ne s'agit que d'une langue comme les autres. Je dirais même que c'est l'une des langues les plus simples à apprendre.

Près d'une décennie plus tard, je suis fière d'avoir pu contribuer au développement de Kids Code Jeunesse, un organisme pancanadien à but non lucratif qui a enseigné la programmation à des centaines de milliers d'enfants et aux adultes qui les soutiennent (enseignant·e·s, parents, bibliothécaires, organisateur·rice·s communautaires). En passant, que sa mère sache programmer n'a pas découragé mon fils à tout jamais. L'année dernière, il a réussi haut la main son premier cours collégial de Python et ne comprenait tout simplement pas pourquoi ses collègues de classe trouvaient le cours si difficile. Apprendre la programmation, ne serait-ce qu'un tantinet, lorsqu'il était jeune a fait qu'il n'a jamais perdu cette compétence, tout comme faire du vélo. Ces connaissances de base lui ont donné la confiance nécessaire pour replonger dans le domaine. De plus, en voyant sa mère apprendre à programmer, il a pu constater qu'on n'est jamais trop vieux pour apprendre quoi que ce soit!

Mais qu'en est-il de ces adultes, et tout particulièrement de ces femmes, qui n'ont jamais été exposés à la programmation lorsqu'ils et elles étaient jeunes? *Le Guide canadien de la pensée computationnelle et de la programmation* a pour but de rappeler à tous ses lecteurs et toutes ses lectrices, sur un ton amical et encourageant, qu'ils et elles ont les compétences, la force et les facultés nécessaires pour apprendre tout au long de leur vie, et ce, peu importe leur âge, leur sexe ou leur domaine de formation. Ce livre existe pour vous rappeler que si un enfant de onze ans peut apprendre quelque chose, un adulte le peut aussi.

Juliet Waters
Responsable des connaissances
Kids Code Jeunesse

Chapitre 1 : Programmer pour apprendre

Pourquoi apprendre à programmer?

La plupart des éducateur·rice·s ont déjà entendu tous les arguments en faveur de l'apprentissage de la programmation, mais demeurent plus nerveux que curieux. Un tel apprentissage permettrait aux enfants d'être mieux préparés pour affronter les emplois et les tâches qui les attendent en ce 21e siècle, leur a-t-on dit. Ils et elles ont vu leurs collègues plus intrépides ou plus à l'aise avec les technologies s'essayer à la programmation. Peut-être même ont-ils ou elles tenté la chose eux-mêmes avec un langage visuel de programmation par blocs, tel que le populaire langage Scratch, conçu par le MIT (eh oui, Scratch est un *vrai* langage de programmation!) Peut-être que cette expérience fut très enrichissante, ou peut-être que les résultats furent plutôt équivoques. Peut-être qu'ils ou elles n'ont pas su quoi faire lorsque leurs élèves ont rencontré un mur, ou lorsque les aptitudes technologiques des élèves ont dépassé les leurs. En apprendre assez à propos de la programmation pour pouvoir l'enseigner est une tâche qui peut paraître ardue, voire impossible, pour de nombreux éducateur·rice·s.

Ces mêmes éducateur·rice·s ont également entendu tous les arguments contre l'apprentissage de la programmation. La programmation, disent certains, empiéterait sur le temps alloué à d'autres matières plus créatives, telles que la musique ou les arts visuels. D'autres soutiennent que la programmation est une compétence obsolète puisque, grâce à l'arrivée de l'intelligence artificielle (IA), les ordinateurs se programmeront bientôt eux-mêmes. D'autres encore ont pour opinion que tout langage de programmation appris aujourd'hui n'existera peut-être plus lorsque leurs élèves quitteront l'école. Tous ces éducateur·rice·s ont leurs propres raisons très valables pour justifier leur décision d'attendre avant de se lancer dans l'apprentissage de la programmation. La plus importante de ces raisons est souvent le manque de temps et de soutien pour les éducateur·rice·s qui veulent apprendre. Avec toutes les responsabilités qu'ils et elles cumulent en tant qu'enseignant·e·s, parents et citoyen·ne·s, est-ce vraiment possible d'enseigner la programmation à tous et à toutes? Est-ce vraiment important?

Apprendre à programmer est à la fois possible et essentiel

Pour répondre d'abord à cette dernière question, réfléchissez à ceci : dans les quelques années qui se sont écoulées entre le moment présent et celui où la majorité des adultes ont acheté leur premier téléphone intelligent, une génération entière est née, une génération qui *ne connaîtra jamais un monde qui n'est pas régi par des algorithmes informatiques!* Les algorithmes, ces séquences de commandes qui constituent un programme informatique, peuvent nous aider à exploiter l'incroyable puissance de calcul des ordinateurs. Cependant, les algorithmes influencent également les goûts et les décisions de cette nouvelle génération. Les algorithmes pourront recueillir des données et en apprendre plus sur cette jeune génération que sur aucune autre, le plus souvent à son insu. Les algorithmes ont le potentiel d'enfermer les jeunes dans des cycles routiniers de distractions infinies.

Nous pourrions toujours confisquer leurs appareils électroniques, mais pendant combien de temps? **Les algorithmes ne sont pas près de disparaître.** Non seulement vont-ils continuer de régir notre monde, mais, alors que la programmation s'articule de plus en plus autour de données et de machines plutôt qu'autour de décisions humaines, ils sont d'ores et déjà en train de devenir plus puissants, plus complexes et plus difficiles à comprendre. Il est donc compréhensible d'avoir peur des algorithmes. Ce qui se passe derrière l'écran de notre ordinateur nous semble souvent occulte et complexe.

Mais à quel point est-il facile pour tous et toutes d'apprendre à programmer? Si vous êtes conscient·e que vous n'essayez pas d'apprendre la programmation dans le cadre d'une carrière en informatique, mais plutôt afin de mieux comprendre la chose et de pouvoir aller plus loin que la littératie numérique pour atteindre la sagesse numérique, **vous pouvez probablement apprendre les bases de la programmation avec seulement 12 à 18 heures d'études.** Vous n'avez pas besoin d'y passer une année entière, comme je l'ai fait. Une fois ces bases apprises, vous serez surpris·e de voir à quel point programmer peut être amusant!

La programmation enrichit nos vies

La programmation ne nous empêche pas de jouer de la musique, de créer des œuvres d'art ou de raconter des histoires. Au contraire! Toutes ces activités peuvent être enrichies par la programmation et explorées par le biais de cette dernière. Le nombre

d'artistes qui travaillent dans le domaine de la programmation et les diverses façons qu'ils et elles ont trouvées pour fusionner leur profession et leur passion ne cessent de me surprendre. Fondamentalement, **apprendre à programmer nous apprend à apprendre**. Et qu'est-ce qu'apprendre, sinon la plus importante aptitude qu'un·e citoyen·ne du 21e siècle puisse posséder?

Au fur et à mesure que les systèmes complexes sur lesquels reposent l'intelligence artificielle et l'apprentissage automatique prendront de la place dans nos vies, la volonté de continuer à apprendre tout au long de notre vie deviendra d'autant plus importante. Apprendre à programmer renforce ce qu'on appelle notre « pensée computationnelle », c'est-à-dire la capacité de réfléchir de façon efficace et créative aux activités routinières qui composent nos vies. De nombreux adultes, à commencer par moi-même, commencent à programmer « pour le bien des enfants », mais se rendent vite compte qu'ils et elles le font pour leur propre bien.

Ce qui importe vraiment, ce n'est pas « pourquoi » apprendre à programmer, mais « quand ». Il est facile de répondre à cette dernière question : maintenant.

Quel est le but de ce livre?

Le Guide canadien de la pensée computationnelle et de la programmation contient toutes les informations nécessaires pour atteindre les trois objectifs suivants, essentiels pour acquérir une littératie algorithmique :

1. Comprendre ce que sont les algorithmes
2. Utiliser un algorithme dans le cadre d'un projet
3. Posséder le langage nécessaire pour pouvoir en apprendre plus sur les algorithmes et explorer les fondements des algorithmes avec d'autres personnes

Ce guide contient des activités et des projets qui aideront les lecteur·rice·s à concevoir des algorithmes et ainsi créer leurs propres « artefacts numériques », tels que des animations, des jeux vidéo et des sites web. Il aidera les lecteur·rice·s à comprendre les

concepts de base de la programmation, de manière à ce qu'ils et elles puissent effectuer un retour sur leurs projets, expliquer leur façon de faire et poser des questions à leurs collègues. Il leur permettra également d'observer le monde qui les entoure et d'évaluer comment les algorithmes pourraient améliorer celui-ci ou lui nuire. Par-dessus tout, ce livre cherche à donner à ses lecteur·rice·s les outils nécessaires pour joindre ou démarrer ces communautés de pratique dont nous avons tous besoin pour continuer à apprendre tout au long de notre vie.

Les activités pratiques et les schémas d'apprentissage contenus dans ce livre ont été soigneusement conçus et sélectionnés afin de favoriser la compréhension de **la logique qui sous-tend les algorithmes**. Ensemble, ils balisent le chemin à prendre pour apprendre, mettre en pratique et assimiler les concepts et méthodes élémentaires qui pourront ensuite être utilisés avec n'importe quel langage de programmation. Cette façon de faire permet d'enseigner les bases tout en évitant de causer une surcharge cognitive en raison d'un trop grand vocabulaire à apprendre. Ce que nous espérons surtout, c'est que ce guide donnera à ses lecteur·rice·s le goût d'aller plus loin.

Non, votre âge n'a *absolument pas* d'importance!

L'une des croyances les plus préjudiciables à la création d'une société de citoyen·ne·s numériques avisés est celle selon laquelle la programmation ne peut s'apprendre que lorsqu'on est jeune. Je vous en prie, croyez-moi lorsque je vous dis que **je n'ai rien d'exceptionnel.** Si j'ai pu apprendre la programmation, vous le pouvez aussi!

En étudiant le concept de « période critique », soit l'idée que certaines choses sont impossibles à apprendre à moins de commencer jeune, le journaliste Gary Marcus s'est rendu compte que la science derrière ce concept était beaucoup moins robuste que ce qu'on pourrait croire. Certains chercheurs ont d'ailleurs récemment mené des expériences qui remettent en question les croyances actuelles dans ce domaine.

Obtenir la sagesse du hibou

Si certains scientifiques utilisent des rats, les chercheur·euse·s se penchant sur le concept de période critique utilisent plutôt des hiboux. Une expérience réalisée à l'université Stanford a démontré que les jeunes hiboux apprenaient effectivement plus rapidement à se déplacer dans une grange simulant une forme de « réalité virtuelle » grâce à des miroirs et des prismes déroutants. Cependant, les hiboux plus âgés pouvaient également

apprendre à se déplacer à l'intérieur de cette grange, à condition qu'ils apprennent petit à petit plutôt qu'en une seule séance. Voilà qui explique comment j'ai pu apprendre les mêmes concepts de bases que mon fils de 11 ans, mais plus lentement. Voilà également qui explique pourquoi je l'ai en fin de compte dépassé. Le hibou âgé que je suis a connu beaucoup plus de cycles d'apprentissage par étapes que mon jeune hibou. De plus, j'avais assez de sagesse pour savoir que je pouvais apprendre et continuer à apprendre.

Dans le domaine de la pensée computationnelle, **cette stratégie consistant à fragmenter des défis apparemment insurmontables en petites tâches plus simples à réaliser se nomme « décomposition »**. À l'âge adulte, nous utilisons cette stratégie continuellement, que ce soit en dressant des listes de choses à faire ou en décidant de ne lire que quelques sections de ce livre plutôt que de le lire dans son entièreté en une seule séance. Chez KCJ, nous avons « décomposé » le défi d'apprendre la programmation pour tous les Canadiens et Canadiennes, de Gander à Terre-Neuve à Prince Rupert en Colombie-Britannique. Nous avons enseigné les algorithmes à des enseignant·e·s d'Iqaluit, des élèves de Whitehorse, des bibliothécaires de Montréal et des parents de Toronto. Nous avons également enseigné à des programmeur·euse·s à communiquer avec des non-programmeur·euse·s dans un langage clair et intuitif.

Ce que les enseignant·e·s pensent de KCJ

Depuis sa fondation en 2013, KCJ travaille au côté des enseignant·e·s dans les classes afin de concevoir des plans de cours qui aident ces enseignant·e·s à apprendre la programmation en même temps que leurs élèves. L'équipe de KCJ a enseigné la programmation à des centaines de milliers d'enfants au pays et a formé des enseignant·e·s à la demande de ministères provinciaux de l'Éducation, de grands conseils ou commissions scolaires, et du programme CodeCan du gouvernement fédéral. En 2018, nous avons formé plus de 6000 enseignant·e·s au Canada. En partenariat avec le camp d'entraînement à la programmation de Lighthouse Labs, nous avons également offert de très populaires journées de formation pour enseignant·e·s intitulées Coder Créer Éduquer.

Nous demandons toujours aux enseignant·e·s de remplir un court sondage. Voici ce qu'ils et elles nous ont répondu :

89 %
ont répondu que les activités correspondent aux besoins des enseignant·e·s et de leurs classes.

87 %
ont répondu qu'ils et elles intégreraient la programmation à leurs projets en classe.

92 %
voient la programmation comme une chance d'essayer de nouvelles façons d'enseigner et d'apprendre.

91 %
croient que la programmation pour les aider à motiver leurs élèves et enrichi leur expérience d'apprentissage.

Alors, finies les excuses! Il est temps de découvrir les **secrets de la pensée computationnelle**.

Chapitre 2 : Développer sa pensée computationnelle

Découvrez le secret de la pensée computationnelle en cinq minutes

Prenez cinq minutes pour dessiner une maison. Votre dessin peut être aussi enfantin et rudimentaire que vous le désirez, mais assurez-vous de dessiner pendant l'ensemble des cinq minutes. Si vous le désirez, vous pouvez dessiner votre maison ici même sur cette page.

Une fois que vous avez terminé, tournez la page pour découvrir **le secret de la pensée computationnelle**.

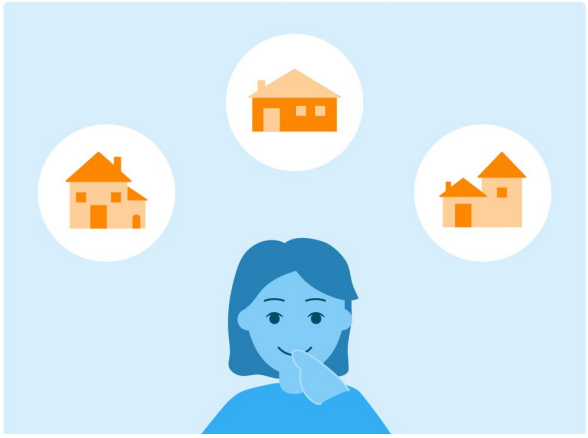
Vous l'exercez déjà!

La bonne nouvelle, c'est que vous exercez votre pensée computationnelle depuis aussi loin que remonte votre mémoire, et même plus loin encore! Lorsque vous n'étiez qu'un bébé et que vous construisiez et remplissiez cette merveilleuse base de données que nous appelons le cerveau, vous n'aviez pas les mots pour nommer ce processus « pensée computationnelle ». Maintenant que vous avez ces mots, l'une des manières les plus simples de maîtriser la pensée computationnelle est d'explorer comment vous l'avez utilisée par le passé. Nous allons donc l'explorer par le biais d'une expérience d'apprentissage parmi les plus basiques et universelles : dessiner une maison.

Comment le dessin nous aide à comprendre la pensée computationnelle

Remémorez-vous le lieu où vous avez appris à dessiner une maison pour la première fois. Il s'agissait peut-être d'une poussette ou d'un siège de voiture. C'est dans cet endroit que votre cerveau a commencé à emmagasiner assez d'exemples de maisons pour pouvoir en reconnaître certains éléments caractéristiques, tels que les murs, les fenêtres et le toit. Ce faisant, vous employiez une puissante stratégie cognitive et computationnelle appelée :

RECONNAISSANCE DES SCHÉMAS



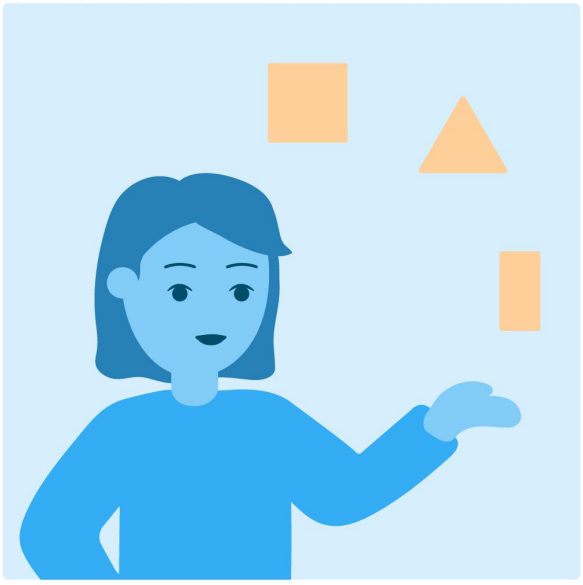
Ce processus ne vous demande guère d'efforts, car il est intuitif chez les humains. C'est en analysant et mémorisant des formes et des motifs que nous apprenons à parler une langue, à reconnaître les visages et à nous orienter au sein d'une ville.

Lorsque vous tentez pour la première fois de dessiner une maison, vous ne commencez pas en dessinant une réplique parfaite d'un château, d'une cathédrale, ni même de votre propre maison. Vous ne dessinez pas directement à partir de votre mémoire. Vous dessinez plutôt à partir de formes encore plus simples, c'est-à-dire à partir d'autres dessins de maisons que vous avez aperçus dans des livres ou sur des affiches, à l'école ou à la maison. Vous reconnaissez ces images comme étant des représentations symboliques simples des réelles maisons que vous connaissez, des représentations qui sont faciles à reconnaître ne possèdent pas trop de détails. Ce faisant, vous employez une stratégie liée à la pensée computationnelle et appelée :

ABSTRACTION

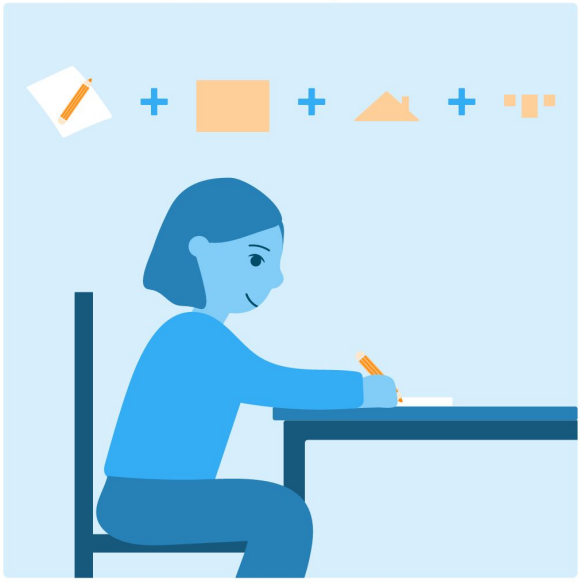
De plus, à moins d'être un génie précoce, il est peu probable que vous dessiniez cette première maison d'un seul trait continu. Au contraire, vous (ou la personne qui vous enseigne à dessiner) allez séparer votre modèle cognitif d'une maison en diverses composantes plus faciles à dessiner et ensuite réunir ces composantes pour créer un tout, c'est-à-dire un dessin représentant une maison. Dans le domaine de la pensée computationnelle, ce processus s'appelle :

DÉCOMPOSITION



Lorsque vous dessinez une maison, commencez-vous par esquisser les fenêtres, le toit ou la cheminée? Probablement pas. En général, vous commencez plutôt par esquisser le rectangle ou le carré qui représente le bâtiment lui-même, puis vous y ajoutez des variables plus interchangeables, comme un toit. Cette série d'étapes, cette routine répétée un nombre incalculable de fois par autant d'enfants partout dans le monde depuis des siècles, exemplifie le plus important concept sous-tendant la pensée computationnelle :

PENSÉE ALGORITHMIQUE



Les algorithmes sont des séries d'étapes routinières qui garantissent que le résultat obtenu sera sensiblement le même à chaque exécution. Il est possible d'introduire quelques variables, telles que la taille des composantes, les couleurs utilisées ou les détails ajoutés pour simuler un matériel de construction (brique, bois, etc.). Notre algorithme s'assurera quand même que le résultat final ressemblera toujours à une maison.

Littératie algorithmique

L'une des plus importantes raisons d'acquérir une meilleure maîtrise de la pensée computationnelle est d'être mieux à même de créer des algorithmes et de comprendre leur fonctionnement. Cette compétence, nommée littératie algorithmique, est le début d'une aventure dont l'objectif est d'acquérir une compréhension plus intuitive des algorithmes. Avec le temps, vous commencerez à repérer et à déchiffrer les algorithmes qui vous entourent, ainsi qu'à concevoir de meilleurs algorithmes qui façonneront le monde selon ce qui vous tient à cœur.

Mais vous devez tout d'abord comprendre les fondements du décryptage et de la programmation d'algorithmes élémentaires. Vous aurez également besoin d'acquérir un vocabulaire de base afin de comprendre les différents éléments avec lesquels vous travaillerez.

Grosso modo, la littératie traditionnelle est la capacité de lire à un niveau assez élevé pour être capable d'améliorer vos aptitudes en lecture, de façon plus ou moins autonome, en lisant davantage. Une fois que vous pouvez déchiffrer le texte sur une page, et une fois que vous pouvez reconnaître et saisir le sens d'un nombre suffisant de mots, le texte que vous lisez devient alors votre plus important outil pour continuer à apprendre. De la même manière, le simple fait de pouvoir discerner la présence des algorithmes dans notre vie et de pouvoir reconnaître le rôle que ceux-ci jouent dans nos interactions avec le monde qui nous entoure est important, mais pas suffisant. Personne ne définirait la littératie comme le simple fait de savoir que les livres existent. Vous devez également être en mesure de démontrer que vous pouvez lire.

Ce livre définit la littératie algorithmique comme suit : connaissance suffisante des algorithmes pour permettre de déchiffrer des séquences simples et comprendre les concepts rudimentaires et fondamentaux qui sous-tendent tous les algorithmes informatiques, peu importe le langage de programmation utilisé pour les créer.

Choisis ta propre aventure

Apprendre à écrire fait de nous de meilleur·e·s lecteur·rice·s, même si nous ne devenons pas des écrivain·e·s professionnel·le·s. Par la même logique, apprendre à programmer des algorithmes peut faire de nous des utilisateur·rice·s plus avertis d'outils numériques. Nous ne devons pas nous contenter de la simple littératie numérique. Nous devons plutôt cultiver notre désir d'en toujours apprendre plus sur ce qui se passe derrière notre écran. Apprendre les concepts et les pratiques liés à la pensée computationnelle et à la programmation peut nous donner le courage nécessaire pour faire une telle chose.

Il est important de rappeler que la pensée computationnelle et la programmation sont deux choses différentes. Malgré tout, **la programmation, c'est à dire s'entraîner à traduire une routine en un langage assez simple pour que l'ordinateur puisse la comprendre**, est une excellente façon d'accroître votre curiosité et de renforcer votre pensée computationnelle. Il n'y a rien de mal à se lancer d'emblée dans la programmation avant de développer davantage sa pensée computationnelle, que ce soit en termes de compétences ou de vocabulaire. Beaucoup d'entre nous ont appris à composer de petits récits simples avant d'apprendre les règles de grammaire régissant les textes plus complexes.

Si vous en avez le courage, ou si vous êtes résistant·e de nature aux concepts abstraits et préférez apprendre d'abord par la pratique, n'hésitez pas à arrêter ici votre lecture de ce chapitre et à passer directement au chapitre 3 : Apprendre à programmer. Certaines personnes préfèrent se jeter directement dans l'eau froide plutôt que de s'y glisser doucement.

Il n'y a bien sur rien de mal à préférer la seconde méthode. Si vous voulez en apprendre plus sur la manière d'intégrer la pensée computationnelle à votre programme scolaire sans vous attaquer tout de suite à la programmation, vous n'avez qu'à lire ce chapitre jusqu'à la toute fin.

Si vous décidez de passer directement au chapitre 3, nous vous recommandons tout de même de lire le reste de ce chapitre ultérieurement pour passer en revue les concepts et pratiques à la base de la pensée computationnelle. Nous recommandons également de prendre le temps d'essayer quelques activités débranchées. Même les programmeur·euse·s professionnel·le·s doivent, à l'occasion, délaisser la programmation et travailler sur des routines « débranchées » pour recharger leurs piles cognitives.

Le ping-pong à la rescousse

Si vous voulez exercer votre pensée computationnelle et celle de vos élèves immédiatement, sans ordinateurs, essayez **Ping Pong à la rescousse**. KCJ a conçu cette activité débranchée dans le cadre de modules éducatifs créés pour le ministère de l'Éducation de la Colombie-Britannique, en partenariat avec le camp d'entraînement de Lighthouse Labs. Nous avons utilisé ce plan de cours pour aider les enseignants de partout au pays à aborder en douceur la pensée computationnelle. Vous n'avez besoin que d'une balle de ping-pong, d'un bandeau et du plan de cours **Ping Pong à la rescousse** qui contient une courte grille d'évaluation simple.

Pour passer de débranché à branché

Écrire une « aventure dont vous êtes le héros » est une autre excellente façon d'explorer la manière dont les algorithmes peuvent emprunter des chemins différents en réaction aux choix effectués. Une fois que l'histoire a été créée, les élèves auront le matériel nécessaire pour programmer en utilisant un outil simple comme Twine. Essayez également cet autre plan de cours expérientiel que nous avons conçu avec Lighthouse Labs à la demande du ministère de l'Éducation de la Colombie-Britannique. **Activité débranchée : Choisis ta propre aventure.**

Gander, Terre-Neuve. Des enseignantes en pleine partie de Ping Pong Rescue

La pensée computationnelle dans le programme scolaire

Comme nous l'avons constaté, la pensée computationnelle est une « compétence cognitive » que nous pouvons utiliser pour résoudre des problèmes ou créer des artefacts. À l'occasion, on l'appelle également la « **pensée algorithmique** ». Elle peut être utilisée sans ordinateurs, par exemple lorsque nous rédigeons une recette étape par étape, lorsque nous procédons à un exercice d'incendie, lorsque nous démontrons comment résoudre un problème mathématique ou, comme nous l'avons vu lors de la première activité, lorsque nous tentons de créer une œuvre d'art. Jumelée à un ordinateur, la pensée computationnelle nous permet d'automatiser les composantes d'une tâche, de concevoir un jeu vidéo génial, ou de créer une animation numérique.

Comme toutes les autres compétences cognitives, telles que l'esprit critique et la pensée créative, la pensée computationnelle peut se développer, devenir plus naturelle avec de l'entraînement, et s'améliorer au fur et à mesure que nous l'utilisons dans des contextes variés. Nous développons notre esprit critique en nous exerçant à l'écriture ou à la communication orale. Nous développons notre pensée créative en apprenant les arts visuels, la danse, ou la musique. De la même manière, s'exercer à la programmation informatique est une excellente façon de développer sa pensée computationnelle.

Le calcul fait partie de la nature humaine

Notons une fois de plus que la programmation n'est pas la seule façon de développer sa pensée computationnelle. Les humains font preuve de remarquables aptitudes pour le calcul et l'utilisaient déjà pour résoudre des problèmes et créer des artefacts bien avant que soient inventés les premiers ordinateurs. **C'est grâce au calcul que nous avons appris à compter avec un boulier.** C'est également grâce au calcul que nous avons créé des jeux aussi complexes que les échecs, où un même résultat peut-être obtenu de multiples façons. L'invention au 19e siècle du métier Jacquard révolutionna l'industrie du textile et du tissage à l'aide d'une série de cartes perforées permettant de créer différents motifs. Ces cartes établirent également les fondements sur lesquels s'appuient toujours les ordinateurs et les langages de programmation modernes.

C'est au mathématicien victorien Charles Babbage qu'on attribue l'invention de l'ordinateur programmable, vers la fin du 19e siècle, même si aucun modèle réel ne fut construit avant le 20e siècle. Le titre de première programmeuse revient cependant à Ada

Byron Lovelace, fille du poète Lord Byron. La science poétique qu'elle a imaginée, décrite et formulée pour la « machine analytique » de Babbage met en œuvre les mêmes principes algorithmiques fondamentaux que les langages que nous utilisons pour programmer les ordinateurs d'aujourd'hui.

Ada Byron Lovelace, la première programmeuse informatique

Les ordinateurs sont des outils qui nous aident à faire usage de ces aptitudes millénaires. Ils nous aident, individuellement et collectivement, à exploiter à son maximum notre vaste puissance de calcul. Mais ce ne sont que des outils, et leur efficacité n'est égale qu'à nos capacités de gestion et d'optimisation.

Alors que nous faisons nos premiers pas dans l'ère de l'intelligence artificielle, être compétent avec les algorithmes nécessite une connaissance toujours plus grande de la manière dont les algorithmes peuvent être gérés par des données ainsi que par des séries d'étapes que nous avons programmées pour eux. Plus nos algorithmes deviennent sophistiqués, plus les ordinateurs apprennent par eux-mêmes à convertir en algorithmes leur capacité à reconnaître des formes. **Cela ne signifie pas que nous pouvons ou devons arrêter d'essayer d'acquérir une littératie algorithmique ou une compétence en la matière.** Il est toujours important de pouvoir déterminer quels algorithmes gagnent à être utilisés avec un ordinateur, et quels algorithmes font le meilleur usage de nos talents d'humains.

Pourquoi la pensée computationnelle est-elle importante?

La puissance de leur pensée computationnelle permet aux humains de créer des logiciels très sophistiqués qui peuvent ensuite être utilisés par un ordinateur. Ce dernier offre la force brute et la persévérance nécessaires pour accomplir rapidement et indéfiniment des tâches simples, ce qui peut s'avérer extrêmement utile lorsque l'ordinateur accomplit la bonne tâche de la bonne manière. Malheureusement, lorsqu'il accomplit la mauvaise tâche de la mauvaise manière, un ordinateur peut causer des dégâts majeurs, voire désastreux, beaucoup plus rapidement qu'aucun humain. En résumé : plus grande est notre maîtrise de la pensée computationnelle, meilleur·e·s sommes nous lorsque vient le temps d'utiliser correctement un ordinateur ou de lui donner les bonnes instructions.

En apprenant à reconnaître les différentes utilisations que nous faisons de la pensée computationnelle dans nos vies et dans le programme scolaire, même sans avoir recours aux ordinateurs ou aux dernières technologies éducatives, les élèves, les enseignant·e·s, les parents et l'ensemble des éducateur·rice·s dans la communauté peuvent **encourager leur curiosité, leur enthousiasme et leur désir d'exploration**. Ensemble, ils et elles peuvent déterminer le meilleur moment de commencer à apprendre la programmation et la meilleure manière d'intégrer la programmation à leurs activités et projets.

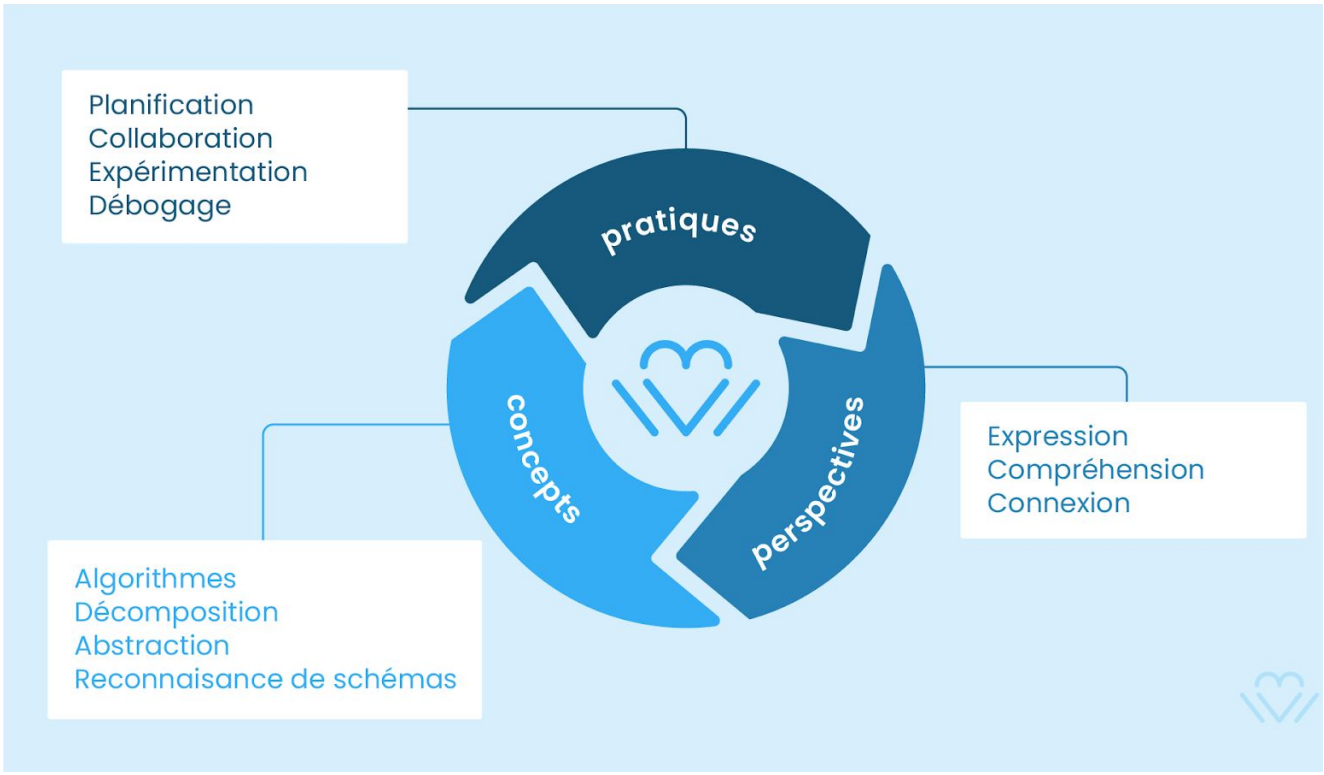
La littératie algorithmique a également comme objectif majeur d'utiliser la pensée computationnelle et la programmation pour **renforcer la certitude et les compétences de base des élèves** afin qu'ils et elles puissent apprendre de manière plus approfondie,

et ce, tout au long de leur vie. La pensée computationnelle est complémentaire de la pensée créatrice et de l'esprit critique. Elle permet aux élèves de résoudre des problèmes complexes, d'aborder les défis avec des perspectives différentes afin d'apporter de possibles solutions, et d'adopter des pratiques les aidant à assimiler des compétences et accroître leur compréhension.

Plus notre société devient dépendante des ordinateurs, plus les enjeux sont grands et plus il devient primordial d'être en mesure de comprendre comment ces ordinateurs fonctionnent si nous voulons continuer de les utiliser de façons sécuritaire et éthique. Chaque fois que nous conduisons une voiture, notre sécurité dépend non seulement de nos aptitudes de conduites, mais également de la qualité du logiciel. Cela deviendra de plus en plus vrai à mesure que nous progresserons dans l'ère de l'IA et de l'apprentissage automatique. Plus la compréhension des outils numériques par les citoyen·ne·s ordinaires qui les utilisent sera grande, plus ces citoyen·ne·s seront prêt·e·s à affronter le monde du travail et à prendre les responsabilités nécessaires à la création d'un monde plus sécuritaire. Cela dit, un des moyens les plus rapides pour développer la curiosité envers la pensée computationnelle est d'amener les gens à réaliser que celle-ci peut être amusante, malgré les difficultés qu'elle peut parfois poser.

Le cadre pédagogique simple de Kids Code Jeunesse

Nous avons développé ce cadre pour les éducateur·rice·s. Il est basé sur les travaux de la chercheuse canadienne Karen Brennan en collaboration avec Mitch Reznick, cofondateur de Scratch et membre du *Lifelong Kindergarten Group* au MIT. Le cadre pédagogique sur la pensée computationnelle de KCJ comprend les concepts fondamentaux et rudimentaires nécessaires pour commencer à acquérir une meilleure maîtrise de la pensée computationnelle.



La pensée computationnelle

Concepts et pratiques liés à la pensée computationnelle

Comme nous l'avons abordé au début de ce chapitre, lorsqu'ils et elles exercent leur pensée computationnelle, les élèves utilisent l'un ou plusieurs des concepts suivants :

CONCEPT	DÉFINITION
Décomposition	Séparer un objet en ces différentes composantes
Reconnaissance des formes	Reconnaître les similarités entre les objets
Abstraction	Représenter un objet avec un minimum de détails
Algorithmes	Séquence d'événements et de commandes

Les concepts qui sous-tendent la pensée computationnelle sont utiles dans tous les domaines d'apprentissage, mais se manifestent différemment selon la matière. Intégrer ce vocabulaire à votre programme éducatif et repérer les endroits où vous utilisez déjà ces concepts sont d'excellents moyens pour commencer à acquérir les connaissances dont vous et vos élèves aurez besoin pour être confiant·e·s lorsque vous relèverez les défis posés par la programmation.

Par exemple, les élèves peuvent utiliser la décomposition pour morceler un problème mathématique et ainsi le rendre plus simple à résoudre. Ils et elles pourraient également utiliser le même concept de décomposition pour séparer les différentes composantes d'une histoire ou d'un texte et ainsi approfondir leur compréhension de certaines formes et de certains genres. Ces stratégies sont souvent à leur plus passionnante lorsqu'intégrées aux cours d'éducation physique, de musique ou d'arts plastiques.

Pour appliquer concrètement les concepts liés à la pensée computationnelle, les élèves peuvent utiliser l'une ou plusieurs des pratiques suivantes :

Pratiques	**Applications**
Planification	Esquisser ou ébaucher de possibles solutions
Collaboration	Travailler en groupe et demander l'aide de ses pairs pour tester des idées ou des artefacts
Expérimentation	Essayer une solution plusieurs fois afin d'améliorer les résultats à chaque essai
Débogage	Réparer des erreurs

Lorsque mise en pratique, la pensée computationnelle aide les élèves à résoudre des problèmes en trouvant des solutions qui peuvent être reproduites par ces mêmes élèves et par d'autres. Elle permet également aux élèves non seulement d'utiliser, mais également de maîtriser, les ordinateurs, ou toute autre technologie, dans le but d'exprimer leurs idées et de partager leurs solutions. Plus important encore, le processus de mise en pratique de la pensée computationnelle est hautement itératif, ce qui signifie

que les élèves peuvent constater les avantages d'un processus qui utilise les répétitions productives pour atteindre un but.

Les exercices et défis permettant d'exercer la pensée computationnelle doivent offrir une atmosphère de confiance permettant aux élèves d'échouer, d'accroître leur résilience, et de soutenir leur esprit de croissance. Les élèves feront inévitablement des erreurs lors de l'écriture de leurs algorithmes, mais, plus crucialement, ils et elles seront capables de rapidement voir le résultat de leur travail, y apporter des changements, et répéter ce processus jusqu'à ce qu'ils ou elles obtiennent le résultat souhaité.

Préparez les élèves avec des activités débranchées

Les activités débranchées sont une des meilleures façons de préparer les élèves, puisqu'elles nécessitent d'utiliser la pensée computationnelle de diverses manières qui s'appliquent également à la programmation. Voici quelques-unes des activités qui demandent l'utilisation de stratégies que les élèves utilisent déjà naturellement dans leur vie quotidienne et qui se trouvent être des fondements de la programmation informatique!

- Faire semblant d'être un robot aveugle qui a besoin que ses pairs lui fournissent des instructions pour pouvoir trouver un objet;
- Créer un jeu avec des consignes précises;
- Écrire des histoires en mettant l'accent sur la manière dont nous pouvons structurer l'intrigue en utilisant des séquences répétables (début-milieu-fin).
- Créer des histoires non linéaires, telles que des aventures dont vous êtes le héros.

Les différentes pratiques de la pensée computationnelle

Planification

La planification peut être particulièrement intimidante pour ceux et celles qui commencent à apprivoiser la pensée computationnelle. Par conséquent, planifiez des activités de réchauffement, de bidouillage et de préplanification afin de stimuler la

formulation d'idées. Avant de commencer à travailler sur un jeu ou une animation, encouragez les élèves à réfléchir à leur projet et à en esquisser un prototype à l'aide d'une feuille de papier et d'un crayon. Demandez aux élèves de séparer le projet en petites tâches faciles à accomplir, mais qui sont logiquement reliées aux objectifs du projet.

Expérimentation

La programmation est un processus itératif, ce qui signifie qu'un·e programmeur·euse doit essayer quelque chose plusieurs fois. Cela signifie également que, pour s'améliorer, les élèves doivent être prêt·e·s à échouer. Les élèves ont besoin de temps pour réfléchir et explorer des pistes de solutions. Pour encourager ce type d'apprentissage, concentrez votre évaluation sur le processus plutôt que sur le résultat. Les élèves peuvent utiliser des feuilles ou cahiers de conception pour noter les obstacles rencontrés et les démarches entreprises pour atteindre un objectif. Certain·e·s enseignant·e·s donnent même une partie des points en cas « d'échec. » Un échec peut être défini par le nombre d'essais effectués ou par l'atteinte d'un résultat plus intéressant que prévu, mais obtenu accidentellement. Encouragez les élèves à ajuster leur plan lorsque nécessaire.

Collaboration

Les programmeur·euse·s travaillent rarement seul·e·s. Concevoir un logiciel est généralement un travail d'équipe, et les programmeur·euse·s ont besoin de l'opinion et des conseils de leurs collègues et mentors, même lorsqu'ils ou elles travaillent sur des projets individuels. Encouragez la collaboration en demandant à vos élèves de noter sur une feuille de conception l'aide reçue et l'aide procurée aux autres. Vous pouvez également prendre le temps d'organiser une activité « réfléchir-joindre-partager » dans le cadre de laquelle les élèves réfléchissent à leur projet, se joignent à un·e coéquipier·ère et partagent leurs problèmes ou solutions. Assurez-vous d'offrir des opportunités de mentorat par les pairs, comme un système de notes autocollantes apposées sur les ordinateurs. Orange pourrait signifier « commentaires demandés! », par exemple, alors que vert signifierait plutôt « j'ai atteint mon objectif et je suis prêt·e à aider les autres ».

Débogage

Pour enseigner le débogage, rien de mieux que de faire soi-même des erreurs! Faire des erreurs devant les élèves, demander l'aide des élèves pour déboguer une séquence, et souligner ce que vous avez appris de vos propres erreurs n'est pas la preuve d'un

manque de connaissances ou compétences de votre part. C'est plutôt la preuve que TOUS les programmeurs et TOUTES les programmeuses doivent s'adonner au débogage, peu importe leur niveau d'aptitude. Les programmeur·euse·s professionnels·le·s passent une grande partie de leur temps à tenter de débusquer les bogues dans leurs programmes. Plus un programme est complexe, plus les chances sont grandes que des bogues s'y trouvent! Vous devez donc apprendre à transformer le fait d'être coincé en une opportunité pour votre cerveau de ralentir et de réfléchir.

Déboguer avec un canard en plastique

L'une de nos méthodes de débogage préférées est ce qu'on appelle le « rubber duck debugging », ou le débogage avec un canard en plastique. Cette méthode consiste littéralement à expliquer notre algorithme à un canard en plastique. Bien souvent, le simple fait d'expliquer notre problème et notre stratégie à un apprenant passif peut forcer notre cerveau à ralentir juste assez pour débusquer le bogue.

Dans les ateliers Coder Créer Éduquer que nous avons offerts en partenariat avec Lighthouse Labs, nous avons présenté aux enseignant·e·s tous nos outils de prédilection : du papier et des stylos pour la planification et la création de prototypes, des bandeaux pour jouer au robot, des notes autocollantes pour la collaboration et, bien sûr, des canards en plastique!

Perspectives développées par la pensée computationnelle

L'intégration de concepts et de pratiques liées à la pensée computationnelle, à l'intérieur comme à l'extérieur de la classe, ne fait pas que préparer les élèves à programmer tout

au long de leurs études postsecondaires et sur le marché du travail. Elle les prépare également à affronter les défis toujours changeants qui jalonnent notre monde numérique et à saisir les opportunités offertes par ce même monde.

Cependant, le but ultime de l'enseignement de la pensée computationnelle n'est pas de créer une nouvelle classe de programmeur·euse·s professionnel·le·s. Il s'agit plutôt d'élever une nouvelle génération de citoyen·ne·s qui aura la certitude d'être capable d'apprendre tout ce qu'elle doit connaître afin de pouvoir agir dans une société dominée par le numérique.

Alors que vous progressez vers ce but, il peut s'avérer utile d'encourager et de favoriser le développement de trois perspectives essentielles dans le domaine de la pensée computationnelle et de vous exercer à les mettre en application à l'intérieur des expériences d'apprentissage des élèves. Voici ces perspectives dans un tableau simplifié :

Perspective	**Applications**
S'exprimer	Reconnaître que les algorithmes sont des moyens de créer (des dessins, des histoires, etc.) ainsi que de résoudre des problèmes
Comprendre	Reconnaître que les programmes et les algorithmes influencent les comportements, autant à l'extérieur qu'à l'intérieur d'un environnement numérique
Connecter	Reconnaître que les programmes et les algorithmes peuvent tisser des liens entre les individus et faciliter la collaboration

Explorons maintenant plus en détail *pourquoi* ces perspectives jouent un rôle inestimable, autant en programmation que dans la vie de tous les jours.

S'exprimer

Plus nous apprendrons à voir la programmation comme un outil aussi utile pour développer notre créativité que pour résoudre des problèmes, plus grande sera notre

capacité à agir et à innover. Rechercher des façons amusantes de programmer et de créer des artefacts qui nous importent est une excellente façon de transformer notre curiosité en enthousiasme et, ultimement, de cultiver la passion d'apprendre.

Comprendre

Les algorithmes sont des moyens de structurer les comportements. Ils font partie de nos activités quotidiennes. Par exemple, lorsque nous effectuons une recherche en ligne, nous faisons confiance au moteur de recherche et à ses algorithmes pour trouver les meilleures informations, sur la base desquelles nous prendrons ensuite des décisions. Choisir le prochain film que nous allons regarder en nous appuyant sur les recommandations de notre plateforme de diffusion préférée implique la même confiance. Les algorithmes façonnent nos comportements en rendant les choses à un tel point intuitives que nous oublions que ces algorithmes ont d'abord été créés par quelqu'un, voire par nous-mêmes. Pouvoir reconnaître, remettre en question, examiner et analyser les algorithmes qui partagent notre quotidien est le début, mais certainement pas la fin, d'une vie entière passée à appliquer la pensée computationnelle au monde qui nous entoure. Accepter que les algorithmes sont et seront inévitables est la première étape sur le chemin qui mène à la littératie algorithmique, et la plus importante.

Connecter

Partager nos algorithmes, harmoniser nos routines avec celles des autres, répondre collectivement à un problème commun, tel un exercice d'incendie, ou bien à un enjeu collectif, tel que la collecte de données sur les changements climatiques : voilà un ensemble de façons dont le développement d'algorithmes solides peut faire naître un monde meilleur pour tous et pour toutes. La nécessité de recueillir et de partager des données tout en respectant notre vie privée et celle d'autrui est un autre contexte dans lequel les algorithmes peuvent renforcer la connectivité entre les individus. Cette connectivité est une bonne chose, mais comme toute relation interpersonnelle, ces liens nécessitent l'adhésion à des valeurs et des règles de conduite communes afin d'assurer le bien de tous.

Pour lire ou explorer davantage

"New frameworks for studying and assessing the development of computational thinking" (Nouveaux cadres de référence pour étudier et évaluer le développement de la pensée computationnelle), par Karen Brennan et Mitchel Resnick, MIT Media Lab

"A Pedagogical Framework for Computational Thinking" (Cadre pédagogique pour la pensée computationnelle), par Donna Kotsopoulos, Lisa Floyd, Steven Khan, Immaculate Kizito Namukasa, Sowmya Somanath, Jessica Weber et Chris Yiu

« Computational Thinking and CS Unplugged »
(La pensée computationnelle et CS Unplugged), par CS Unplugged, un organisme à but non lucratif basé en Nouvelle-Zélande.

Chapitre 3 : Apprendre à programmer

Comme nous venons de l'établir, vous utilisez déjà la pensée computationnelle, et ce depuis longtemps. Ce n'est probablement pas le cas de la programmation, puisque sinon, vous ne liriez pas ce livre. Avant d'aborder les concepts fondamentaux et quelques exercices de base, attaquons-nous d'abord à l'un des plus imposants obstacles à l'intégration de la programmation dans votre classe. Que faire lorsque les compétences

des élèves dépassent les nôtres ou lorsque ces mêmes élèves rencontrent des problèmes auxquels nous ne connaissons pas la solution?
En d'autres mots :

Comment enseigner une matière lorsque nous ne connaissons pas toutes les réponses?

Dans le cadre de nos ateliers Coder Créer Éduquer, j'ai posé cette question à des milliers d'enseignant·e·s de partout au Canada. Il s'agit d'une question très large. Il n'y a pas de bonne ni de mauvaise réponse. Il est donc utile de diviser cette question en trois sous-questions plus précises : (question surprise : quelle stratégie provenant de la pensée computationnelle utilisons-nous lorsque nous divisons une grosse idée en plusieurs idées plus petites? *)

- Qu'est-ce que nous enseignons lorsque nous ne connaissons pas les réponses?
- Pourquoi voudrions-nous enseigner de cette manière?
- Est-ce que quelqu'un a déjà enseigné de cette manière auparavant?

Ci-dessous, vous trouverez les réponses les plus courantes, selon les enseignant·e·s ayant participé à nos ateliers. J'y explique également pourquoi il est important que les éducateur·rice·s gardent ces réponses en tête.

Question surprise

Quelle stratégie provenant de la pensée computationnelle utilisons-nous lorsque nous divisons une grosse idée en plusieurs idées plus petites?

Réponse : La déconstruction, bien sûr!

Qu'est-ce que nous enseignons lorsque nous ne connaissons pas les réponses?

Comment apprendre. C'est de loin la réponse la plus courante. Notre société insiste trop souvent pour que les enseignant·e·s se présentent comme des expert·e·s dans un domaine spécifique, plutôt que ce qu'ils et elles doivent être, c'est-à-dire des expert·e·s en apprentissage. Savoir comment programmer n'est certainement aussi important que de savoir comment aider les élèves à déterminer où se trouvent les réponses. Quelques fois, ces réponses se trouvent à même une plateforme de programmation populaire telle que Scratch. À d'autres moments, la réponse proviendra d'un·e autre élève qui semble apprendre la matière plus rapidement. Très souvent, bien sûr, les élèves se tourneront vers Google. Après tout, c'est ce que font également les vrai·e·s programmeur·euse·s! Il n'y a aucune honte à utiliser l'apprentissage de la programmation comme une occasion de s'exercer à utiliser efficacement les moteurs de recherche.

Comment composer avec l'incapacité de trouver une réponse. Cette autre réponse courante souligne que l'incapacité de trouver une réponse à un problème dans l'immédiat n'est pas vraiment un échec, mais plutôt un élément inévitable de la condition humaine. C'est pourquoi la programmation est souvent présentée comme une excellente façon de développer sa persévérance, sa détermination et son esprit de croissance. Ces attributs sont essentiels à notre époque dominée par Google, car celle-ci nous a habitué·e·s à trouver des réponses en quelques clics seulement. Parfois, la meilleure réaction au problème de ne pas avoir de réponses, c'est d'aller dormir et de réessayer le lendemain.

Comment admettre avec grâce que nous ne connaissons pas la réponse. Cette réponse est une de mes favorites. On insiste beaucoup trop pour que les enseignant·e·s se présentent comme des expert·e·s, même quand ce n'est pas le cas. Lorsque j'étais journaliste, j'ai interviewé plusieurs expert·e·s et me suis aperçue que, règle générale, un·e vrai·e expert·e sait ce qu'il ou elle ne sait pas et a rarement peur de l'avouer. La programmation est un champ de compétence qui offre en permanence l'occasion de conjuguer expertise et humilité. Même les programmeur·euse·s professionnel·le·s passent une grande partie de leur temps à chercher des réponses. Plus vous serez à l'aise avec l'humilité, mieux vous apprendrez et enseignerez la programmation, et moins vos élèves auront peur de l'apprendre.

Pourquoi voudrions-nous enseigner de cette manière?

Parce que nous mettons ainsi l'accent sur le processus d'apprentissage plutôt que sur son résultat. Il existe de très bons arguments expliquant pourquoi il vaut parfois mieux enseigner quelque chose que nous ne connaissons pas très bien. Au minimum, vous êtes moins à risque de priver vos élèves de l'expérience de véritablement trouver la réponse par leurs propres moyens. C'est pourquoi un·e enseignant·e qui vient tout juste de commencer à programmer est souvent mieux placé·e pour enseigner la programmation qu'un·e expert·e dans le domaine.

Trop souvent, un·e expert·e ne peut s'empêcher d'être un·e expert·e, ce qui signifie parfois qu'il ou elle ne sait pas quand se retenir et laisser aux élèves de l'espace pour apprendre. Les expert·e·s ont cette fâcheuse manie de donner trop d'informations et d'ainsi créer une surcharge cognitive qui rend plus difficile pour les élèves de reconnaître et d'assimiler les concepts fondamentaux. Les enseignant·e·s sont formé·e·s pour comprendre cette situation. Ils et elles s'abstiendront donc souvent de donner la réponse afin de donner aux élèves l'occasion de résoudre le problème à leur rythme. Parfois, il est même plus facile de faire ainsi lorsque l'enseignant·e·ne connaît pas la réponse.

Est-ce que quelqu'un a déjà enseigné de cette manière auparavant?

En fait, je préfère enseigner de cette manière! Jamais je n'ai posé cette question à une salle remplie d'enseignant·e·s sans obtenir cette réponse au moins une fois. Il existe une théorie pédagogique, très populaire dans le domaine des technologies, selon laquelle les enseignant·e·s devraient enseigner moins de contenu et favoriser plutôt l'apprentissage à l'aide de plans de cours axés sur la création utile et orientés par projets. Cette théorie se nomme le constructionnisme et fut avancée pour la première fois par Seymour Papert, professeur au MIT.

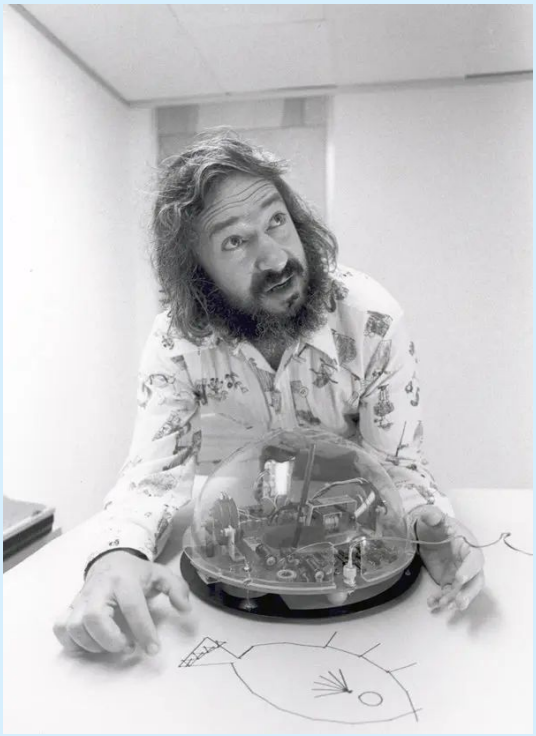
Le constructionnisme est à la base de certains mouvements qui préconisent de « programmer pour apprendre », tel que le mouvement *maker*. Papert est l'auteur de *Jaillissement de l'esprit*, l'un des ouvrages ayant eu le plus d'influence sur ce courant pédagogique. L'idée principale du livre est que la programmation est l'un des meilleurs moyens d'apprendre à apprendre, indépendamment des compétences acquises en programmation. En entraînant les ordinateurs à faire que nous voulons qu'ils fassent, nous n'avons pas d'autres choix que de réfléchir à la manière dont nous apprenons.

Papert a découvert qu'en apprenant à programmer, les enfants renforçaient leur numératie et apprenaient même les concepts à la base de formules complexes utilisées en mathématique et en ingénierie. Les élèves atteignaient également une zone que l'un·e d'entre eux a décrite comme une zone « d'amusement difficile », c'est-à-dire cette zone où le développement des compétences cognitives devient à la fois exigeant et gratifiant.

,

Seymour Papert : Une icône dans le monde de l'éducation

Si vous faites une recherche sur Seymour Papert, vous trouverez un amas de photos, de mèmes et de citations sages. Ces dernières rivalisent en nombre avec celles de Lao Tseu, ma préférée étant celle-ci : **« Le rôle de l'enseignant est de créer les conditions de l'inventivité plutôt que de fournir des réponses ».** Par « les conditions de l'inventivité », Papert n'entendait pas nécessairement l'ordinateur, le iPad, le micro:bit et le nombre incalculable d'autres appareils qui ont été créés depuis. Il parlait également des outils cognitifs, ces concepts que les enfants doivent absorber pour apprendre à construire et à inventer.



Cynthia Solomon/MIT Media Lab

Votre boîte à outils cognitive

Les outils cognitifs les plus élémentaires qu'un·e élève ou un·e éducateur·rice doit posséder sont quatre concepts avec lesquels vous avez déjà fait connaissance si vous avez essayé l'une ou l'autre des activités débranchées que nous vous avons suggérées dans le chapitre précédent. Sinon, vous les apprivoiserez rapidement lorsque vous commencerez à programmer.

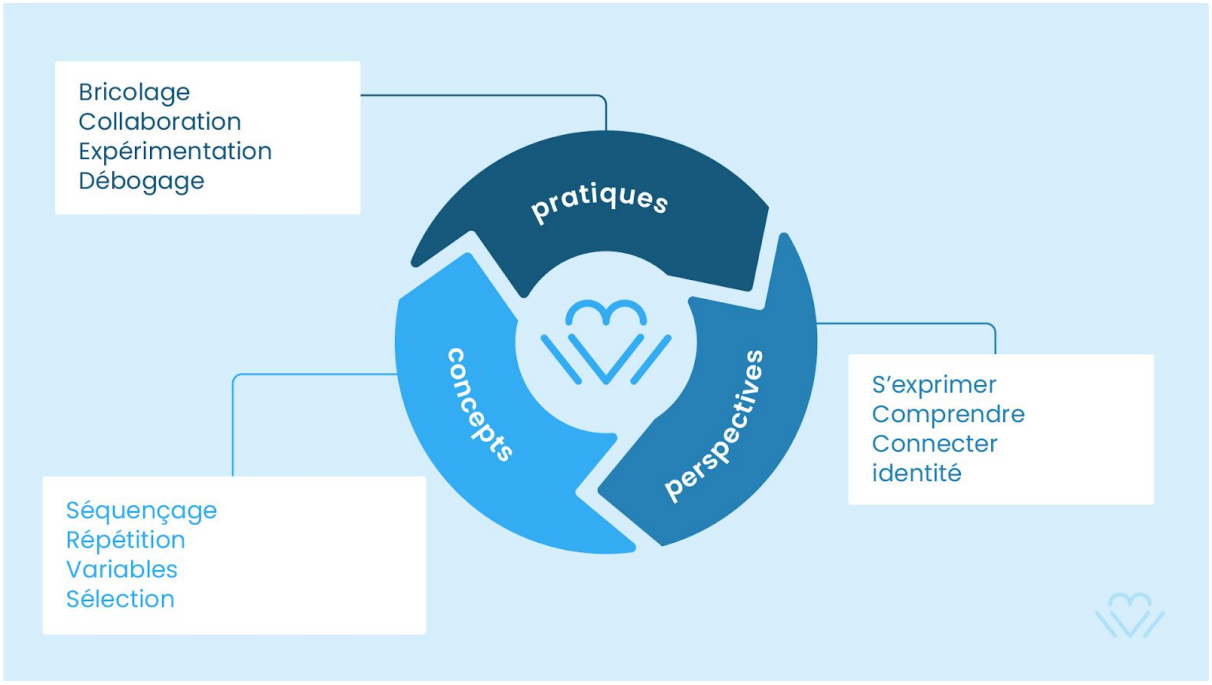
CONCEPTS DE PROGRAMMATION
Séquencement
Répétition
Variables
Sélection

Le **séquencement** est facile à conceptualiser, mais pas toujours aussi facile à mettre en pratique. Lorsque vous explorez les algorithmes ou que vous créez tout autre ensemble de commandes, vous apprenez le séquencement. Les ordinateurs, cependant, ont besoin de séquences différentes de celles des algorithmes débranchés ordinaires. Cela deviendra très évident lorsque vous commencerez à programmer. La programmation consiste essentiellement à apprendre à écrire des algorithmes que les ordinateurs pourront ensuite exécuter.

La **répétition** entre en scène au moment où vous décidez de rendre votre séquence plus concise grâce à une commande du genre « répété 3 fois ». C'est la première étape d'une boucle de programmation. Tout comme le séquencement, les boucles peuvent paraître faciles à conceptualiser, mais les mettre en pratique est une tout autre histoire! Les ordinateurs répètent très rapidement leurs actions, ce qui peut donner l'illusion qu'ils savent ce qu'ils font. Une grande partie du processus de débogage consiste à ralentir ses actions afin de voir précisément ce que l'ordinateur répète vraiment.
Les **variables** sont utilisées pour personnaliser un algorithme. Lorsque nous définissons une variable dans un algorithme, nous introduisons un moyen de changer les entrées de nos algorithmes afin d'obtenir des résultats semblables, mais différents. Par exemple, si je veux créer une maison avec un ordinateur, je peux créer une variable appelée « porte, » pour laquelle je peux ensuite entrer une couleur ou une taille de porte différente chaque fois que j'utilise mon programme pour dessiner une maison. Encore une fois, les ordinateurs peuvent changer une variable extrêmement rapidement. S'y habituer peut prendre du temps, et le programme qui en résulte peut ne pas fonctionner correctement si les variables n'ont pas été clairement définies ou mises dans le bon ordre.

La **sélection** est quant à elle utilisée pour créer des algorithmes qui peuvent emprunter différentes voies. Nous utilisons la sélection à l'aide des traditionnelles commandes SI/ALORS. Vous en avez croisé une au chapitre 2 lorsque nous vous avons demandé de choisir votre méthode apprentissage. SI vous vouliez vous lancer directement dans la programmation, ALORS vous êtes passé·e directement à ce chapitre-ci. SINON, vous avez lu le chapitre 2 jusqu'à la toute fin. Ou peut-être avez-vous décidé d'essayer d'abord une activité débranchée. SI vous vous êtes dirigé·e directement vers ce chapitre-ci, mais que la programmation s'est avérée plus ardue que ce que vous pensiez, ALORS vous pouvez arrêter votre lecture et essayer une activité débranchée afin de ralentir un peu le processus jusqu'à ce que vous soyez à l'aise pour continuer.

Le cadre pédagogique de Kids Code Jeunesse pour apprendre la programmation



Apprendre à programmer

Le séquencement, la répétition, les variables, et la sélection ne sont certainement pas les seuls concepts de programmation qui existent, mais ce sont les plus élémentaires. **La programmation est l'art de mettre ces concepts, et plusieurs autres, en pratique pour communiquer avec les ordinateurs.** Si nous pouvions assimiler ces concepts simplement en lisant à leur propos, nous n'aurions pas besoin d'apprendre à programmer.

La pratique est la seule façon de transformer ces concepts théoriques en véritable compréhension. Cela dit, les cadres pédagogiques peuvent nous servir de guides de travail. Notre cadre pédagogique pour apprendre la programmation s'appuie sur le cadre pédagogique sur la pensée computationnelle abordé au chapitre précédent. Comme vous pouvez le constater, les pratiques et les perspectives sont sensiblement les mêmes. La planification a été remplacée par le bidouillage, qui est une approche plus concrète, mais sinon, la programmation s'apprend sensiblement de la même manière que n'importe quelle autre compétence.

Pratiquer modifie nos perspectives

Mettre des concepts abstraits en pratique modifie notre perspective, ce qui, en retour, enrichit davantage nos concepts et nos pratiques. Par exemple :

- Créer quelque chose à l'aide de la programmation nous aide à reconnaître celle-ci comme un véhicule autant pour **l'expression** que pour la résolution de problème.
- La programmation nous aide à développer une meilleure **compréhension** des algorithmes et d'autres produits conçus à partir d'algorithmes, tels que l'Internet.
- La programmation nous aide à développer des **connexions** non seulement avec d'autres programmes, mais aussi avec les gens qui conçoivent et utilisent ces programmes.
- Plus encore que la pensée computationnelle, la programmation nous aide à développer une nouvelle identité, c'est-à-dire qu'elle nous aide à passer d'un sentiment d'impuissance et d'incompréhension face aux technologies à celui d'être une personne qui sait programmer et qui réalise que le langage de la technologie peut s'apprendre.

Les cinq meilleures stratégies pour se sortir d'une impasse

Maintenant que vous avez notre cadre pédagogique pour apprendre à programmer bien en tête, explorons quelques autres réponses à la question que nous avons posée au début de ce chapitre : comment enseigner une matière lorsque nous ne connaissons pas toutes les réponses? Ces réponses-ci sont plus spécifiques à la programmation.

En 2018, la chercheuse canadienne Karen Brennan, affiliée à la Harvard Graduate School of Education, a sondé des programmeur·euse·s Scratch expérimenté·e·s et âgé·e·s de moins de 17 ans à propos des stratégies qu'ils et elles utilisent pour se sortir d'une impasse. Voici les cinq réponses les plus courantes. **Notez ces réponses sur une fiche et vous aurez la réponse à presque toutes les questions qu'un·e programmeur·euse débutant·e pourrait vous poser.** Ce ne sera peut-être pas la réponse que votre élève voudra entendre, mais ce sera tout de même une réponse.

Les 5 stratégies préférées des enfants pour se sortir d'une impasse.

1. Lisez votre code au complet
2. Expérimentez, bidouillez et jouez avec votre code
3. Recherchez des exemples
4. Travaillez avec quelqu'un d'autre
5. Soyez persistant, mais sachez à quel moment prendre une pause

Sondage effectué auprès de la communauté Scratch par Harvard Edu

Avec ces réponses à portée de main et une meilleure idée de ce que vous apprendrez et enseignerez réellement, vous avez maintenant en votre possession **tous les outils nécessaires pour partir à l'aventure dans le monde la programmation!**

Programmation visuelle par blocs

Seymour Papert a créé plus que la théorie pédagogique connue sous le nom de constructionnisme. Il a également créé le premier langage de programmation adapté aux enfants : Logo. Logo utilisait la programmation pour déplacer une tortue à travers un écran de façon à dessiner des formes simples et des motifs étonnamment complexes. En programmant avec Logo, les enfants modélisaient visuellement et mentalement le fonctionnement d'une boucle de programmation, la vitesse stupéfiante à laquelle un ordinateur peut changer les variables, et l'ajustement nécessaire pour adapter des séquences à l'ordinateur et à sa compréhension limitée du monde qui l'entoure. Ces **modèles cognitifs de base sont aussi importants que l'apprentissage de la syntaxe**

d'un langage de programmation qui sera peut-être en voie d'être obsolète avant même que vos élèves aient terminé leurs études secondaires.

Apprendre grâce à Scratch

Vingt ans plus tard, le groupe Lifelong Kindergarten au MIT Media Lab s'est basé sur langage de Papert pour élaborer un nouveau langage de programmation adapté aux enfants et nommé Scratch. La tortue fut remplacée par un chat qui peut à son tour être remplacé par un nombre illimité de « sprites » provenant d'une énorme bibliothèque d'images. La programmation textuelle, et tous les obstacles que son apprentissage implique, autant pour les enfants que pour les adultes, ont été remplacés par des blocs qui ressemblent à des blocs Lego et s'assemblent de la même manière.

Les enfants peuvent utiliser la même méthode d'exploration par le jeu qu'ils et elles utilisent depuis des générations grâce aux célèbres blocs Lego pour créer, démonter et recréer des programmes. Ils et elles peuvent donc transférer les compétences liées à la pensée computationnelle qu'ils et elles ont déjà acquises par le jeu pour maîtriser et absorber des concepts de programmation élémentaires, intermédiaires et même avancés, sans avoir à se soucier des erreurs de syntaxes.

Bien qu'originellement pensé pour les activités parascolaires et les programmeur·euse·s débutant·e·s de 8 ans et plus, le langage est assez robuste et sophistiqué pour être utilisé dans le cours d'introduction à l'informatique de Harvard. Le langage Scratch est maintenant supporté par une plateforme puissante et sous modération constante : Scratch.mit.edu. Les blocs sont offerts en plus de 25 langues, et près de cent millions de projets ont été créés avec Scratch par des enfants du monde entier. Scratch est devenu **l'une des plus grandes communautés de pratique pour les enfants, et certainement la plus étendue géographiquement**. Le langage se taille également une place de plus en plus grande dans les écoles. De nombreux·euses enseignant·e·s l'utilisent pour enseigner des compétences numériques telles que l'animation, sans même se rendre compte qu'il s'agit d'un langage de programmation!

Programmer avec Scratch dans votre classe

Cela dit, introduire Scratch dans votre classe n'est pas aussi facile que de déverser une pile de blocs Lego sur le plancher. Pour reprendre une idée d'Ollie Bray, *Global Director of*

Play (Directeur mondial du jeu) chez Lego, **la technologie ne rend pas l'enseignement plus facile, elle rend l'enseignement différent.**

Un excellent exemple est le défi de garder une classe assise et concentrée pendant toute la durée des tutoriels autodirigés de Scratch. Créés en partenariat avec Google en 2018, ils sont disponibles à même la plateforme. L'idée derrière ces tutoriels est de libérer l'enseignant·e de l'obligation de savoir quoi que ce soit à propos de la programmation. Les enfants sont plutôt encouragé·e·s à apprendre ce qu'ils ont besoin de savoir en suivant les tutoriels.

L'un de mes moyens favoris d'enseigner Scratch à des éducateur·rice·s est de les guider étape par étape à travers le premier tutoriel autodirigé, comme si nous étions dans un monde idéal où les enfants suivent parfaitement les tutoriels tels des robots. Puis je les guide à travers ce qui arrive généralement quand on demande à 25 enfants de suivre le tutoriel en même temps. Parce que les enfants sont évidemment le véritable contraire des robots. Si les enfants sont programmé·e·s pour quelque chose, c'est pour être curieux et avoir le goût de voir ce qui arrive lorsqu'ils ou elles ne suivent pas les instructions.

Les enfants n'ont absolument aucun problème à faire fonctionner un programme simple qui fait bouger puis miauler le chat sur la « scène » de Scratch.

Les ennuis commencent dans les quelques minutes suivantes, lorsque les enfants s'aperçoivent qu'il est possible de placer cette séquence très simple à l'intérieur d'un bloc « répéter » pour faire bouger et miauler le chat plusieurs fois de suite. C'est incroyable à quelle vitesse ils et elles constatent qu'il est possible de faire bouger le chat un millier de fois ou, encore mieux, de mettre la séquence qui fait bouger et miauler le chat dans un bloc « répéter indéfiniment ».

Je n'ai jamais chronométré le temps nécessaire avant que les chats commencent à sortir de la scène, ne laissant plus que le bout de leur queue visible, mais en continuant sans cesse de miauler à un rythme effréné. En revanche, je sais qu'il ne faut guère de temps pour qu'un·e enseignant·e se retrouve à gérer un troupeau de 25 chats assourdissants, autant en chair et en os que derrière l'écran. Peu d'enseignant·e·s s'arrêtent, savourent ce moment, et se disent « Que c'est merveilleux! Mon début de migraine m'indique que mes élèves viennent d'apprendre l'un des plus puissants concepts de programmation : la répétition! »

Fabriquez votre propre bouton de redémarrage, et bien plus, avec la pensée computationnelle

Comme la plupart des gens qui ont été formés pour être des utilisateur·rice·s plutôt que des concepteur·rice·s de logiciels, les éducateur·rice·s et les élèves essaient d'abord de trouver la commande de redémarrage ou d'annulation. En général, ils s'aperçoivent rapidement qu'il n'y en a pas. La plateforme Scratch ressemble à n'importe quel autre logiciel en ligne, mais **le but de Scratch est de pouvoir interagir avec la plateforme en tant que programmeur·euse plutôt qu'en tant qu'utilisateur·rice.** La principale différence entre les programmeur·euse·s et ceux et celles qui ne le sont pas est que ces dernier·ère·s cherchent un bouton de redémarrage alors que les programmeur·euse·s savent comment créer le leur.

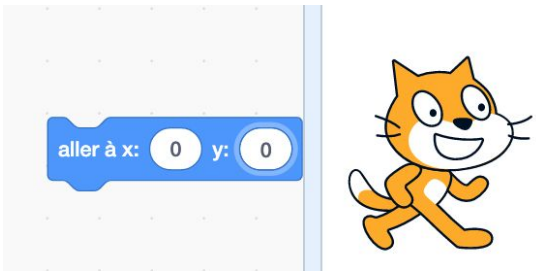
En fait, programmer un bouton de redémarrage est très facile à faire avec Scratch. Premièrement, vous devez reconnaître que vous avez besoin d'une séquence de redémarrage. Il suffit ensuite de décomposer au maximum le problème en plus petites tâches. (Si vous voulez d'abord ouvrir Scratch et essayer de le faire vous-même, la

prochaine section sera peut-être plus facile à comprendre. Mais il vous sera peut-être utile de lire d'abord la description de la démarche qu'emprunte la pensée computationnelle. Vous pouvez également suivre les instructions étape par étape fournies dans le plan de cours se trouvant à la fin de cette section.)

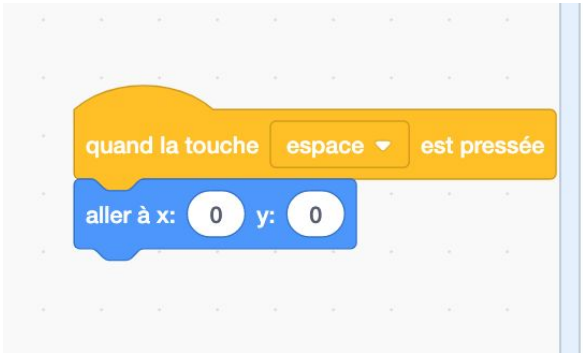
Dans une boucle de programmation, vous voulez toujours établir d'abord la fonction de l'objet de programmation. Par exemple, si vous conceviez un algorithme pour permettre à un robot de se rendre à une destination précise pour récupérer une balle de ping-pong, vous voudriez alors commencer par établir la position initiale du robot. Sinon, le robot terminerait toujours son mouvement dans un endroit différent, et non pas là où se trouve la balle de ping-pong.

Introduction aux boucles de programmation avec Scratch

Tous les « sprites » utilisés par Scratch se déplacent sur une scène divisée selon une grille cartésienne (ce graphique de base à deux axes, X et Y, que nous avons tous appris à l'école, même si nous ne souvenons pas de son nom). Vous n'avez pas besoin d'expliquer ce qu'est une grille cartésienne à vos élèves. En fait, je vous déconseille même de le faire dans ce cours-ci. Les élèves en apprendront déjà beaucoup aujourd'hui. Reportez donc cette information à un cours ultérieur. Vous avez seulement besoin de leur montrer le bloc « aller à x:0 y:0". Lorsque vous cliquerez sur celui-ci, le sprite retournera au milieu de la scène.



Un ensemble de blocs « Événement » jaunes agissent comme des déclencheurs pour chaque séquence que vous programmez. Choisissez celui qui permet de déclencher une séquence lorsque la barre d'espacement du clavier est pressée. Insérez-y votre bloc de positionnement et voilà, vous avez un bouton de redémarrage.



En programmation, c'est ce qu'on appelle **« initialiser » votre boucle**. L'un des premiers et plus précieux algorithmes qu'un·e éducateur·rice peut apprendre et enseigner est celui qui crée une séquence d'initialisation qui réglera les paramètres par défaut de toute boucle créée avec un bloc « répéter ». En effet, en programmation textuelle, il est impossible pour une boucle de fonctionner correctement sans déterminer quand la boucle commence et s'arrête. Par conséquent, le plus tôt vous apprendrez ce concept dans Scratch, plus vous y gagnerez.

Cette séquence permet donc aux élèves qui ont programmé trop de répétitions et fait sortir le sprite de la scène de ramener ce dernier au centre de la scène.

Malheureusement, le sprite est toujours coincé dans une boucle « répéter indéfiniment ». Il continuera donc de sortir de la scène, sauf si vous lui imposez une condition qui lui permettra de se déplacer indéfiniment sans sortir de la scène.

Voici un bloc qui à lui seul introduit une logique de sélection :

Insérez-le à l'intérieur d'un bloc « répéter indéfiniment ». Vous avez maintenant imposé une condition à votre sprite : s'il atteint le bord de la scène, il va se retourner et continuer à avancer dans la direction opposée. (Il est hautement recommandé de demander respectueusement à vos élèves d'enlever tout bloc faisant miauler le chat de toute boucle qui se répète indéfiniment, et ce, pour toujours.)

Il est bon de noter que dans ce programme tout simple, nous avons abordé les quatre concepts contenus dans notre boîte à outils cognitive :

1. Nous avons utilisé **le séquencement** pour créer le programme.
2. Nous avons utilisé **la répétition** pour créer une boucle de programmation.
3. Nous avons utilisé **des variables** pour déterminer le point de départ du sprite.
4. Nous avons utilisé **la sélection** pour imposer une condition au sprite avec la commande « rebondir si le bord (de la scène) est atteint ».

Votre mission, si vous l'acceptez, est d'explorer plus abondamment ces concepts en jouant davantage.

Vous trouverez à l'annexe 1 plusieurs plans de cours simples tournant autour de ces quatre concepts.

Si vous voulez explorer Scratch comme un outil pour créer des animations, ce plan de cours a été testé dans des centaines de classes sur le territoire de la Commission scolaire de Montréal (CSDM), l'une des plus importantes commissions scolaires au Québec : *Let's move* (On bouge).

Mais avant de vous lancer tête la première dans la brousse pour explorer le potentiel d'animation infini de Scratch, peut-être aimeriez-vous d'abord essayer Scratch avec une compétence que vous possédez déjà : le dessin.

Créer des conditions propices à l'innovation

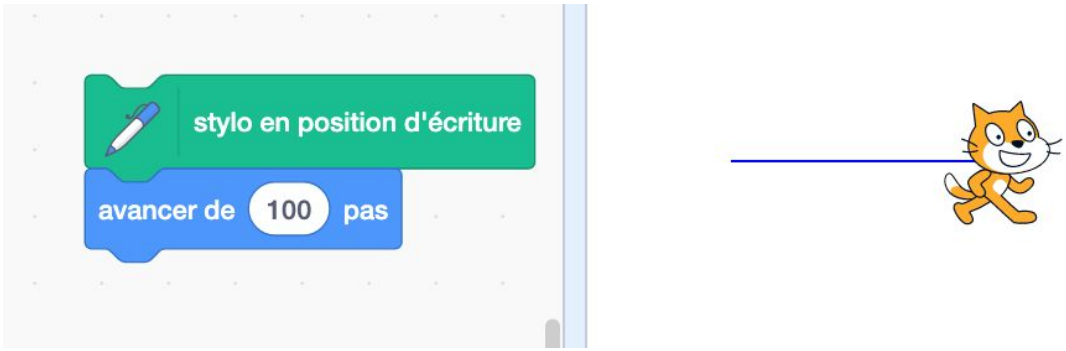
Ce qui est bien avec Scratch, c'est qu'il s'agit à la fois d'un langage de programmation complet et d'une plateforme d'animation, offrant ainsi aux enfants une multitude de façons de créer. Mais cela est également le plus grand défi posé par Scratch. Avec autant d'options, les enfants peuvent prendre tellement de chemins différents qu'il devient difficile pour un seul vieux hibou comme vous et moi de suivre leur rythme d'apprentissage. D'excellents guides d'utilisation de Scratch, tel que « Creative Computing » (Informatique créative) de Karen Brennan, suggèrent de mettre les enfants au défi de créer des programmes avec Scratch en utilisant seulement un nombre limité de blocs. Un tel défi délimite une « aire de jeu » qui maintiendra les débutant·e·s loin des blocs plus puissants et complexes pendant un certain temps. Il s'agit d'un très bon choix pour tout groupe constitué d'enfants qui n'ont jamais utilisé Scratch. La popularité croissante du langage, cependant, rend une telle occasion de plus en plus rare.

Scratch : un véritable terrain de jeu

Selon nos observations, rapprocher Scratch de son ancêtre, Logo, grâce à l'extension « stylo » est l'une des meilleures façons de délimiter une « aire de jeu » dans Scratch sans limiter pour autant le potentiel des enfants qui sont déjà habitué·e·s à créer des animations avec Scratch et qui veulent donc explorer ses fonctions plus avancées.

Une fois qu'un bloc « stylo » est utilisé, tous les blocs « mouvement » deviennent des blocs « dessin ». Les élèves peuvent ainsi apprendre à dessiner une ligne droite à l'aide d'un

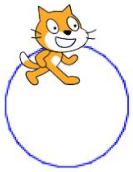
algorithme simple construit à partir des blocs de mouvements les plus élémentaires. (J'ai réduit le chat ci-dessous à 30 % de sa taille normale, sinon il serait impossible de voir la ligne, qui mesure seulement 10 pixels.)



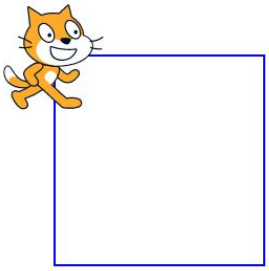
Ajoutez un bloc « tourner de 15 degrés » et un bloc « répéter » pour transformer cette ligne droite en courbe.

En expérimentant avec le bloc « répéter », les enfants peuvent apprendre à dessiner un cercle dès l'âge de 7 ans. Nul besoin de géométrie!

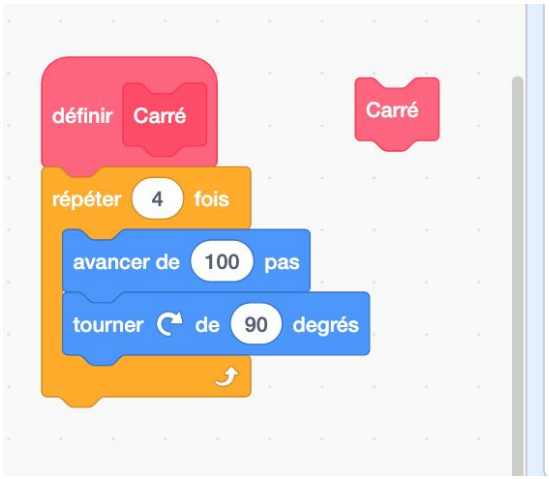
Une séquence de redémarrage utilisant le bloc « effacer tout » replacera le sprite à sa position et son angle de départ (90 degrés à l'horizontale, afin de maintenir le chat debout). Les élèves peuvent expérimenter jusqu'à ce qu'ils et elles déterminent le nombre de répétitions nécessaires pour dessiner un cercle complet.

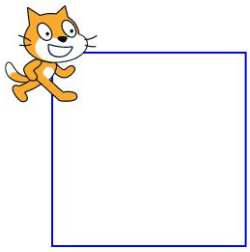


Vous pouvez également essayer des formes différentes.



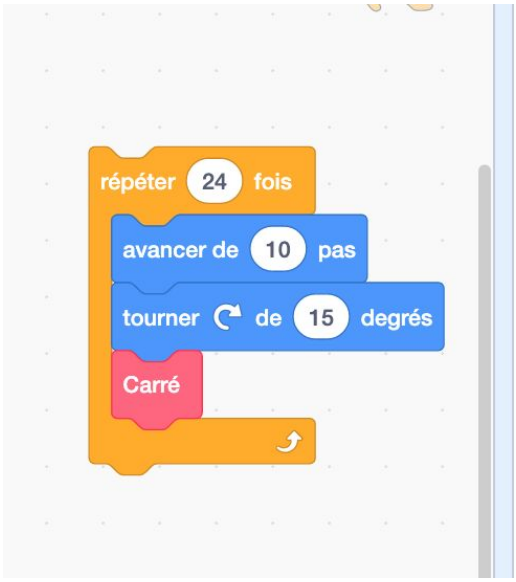
Une fois que les formes ont été créées, nous pouvons utiliser les blocs roses de la catégorie « Mes Blocs » pour simplifier encore davantage nos algorithmes en les nommant. Nommer nos séquences afin de pouvoir les réutiliser sans avoir à réécrire minutieusement les commandes chaque fois est le principe de base ce qu'on appelle la programmation modulaire. C'est la forme avancée de la programmation de séquences pour débutant·e·s.
Apprendre la programmation modulaire tôt encourage les enfants à maintenir leurs algorithmes concis et lisibles, ce qui facilite ensuite le débogage.

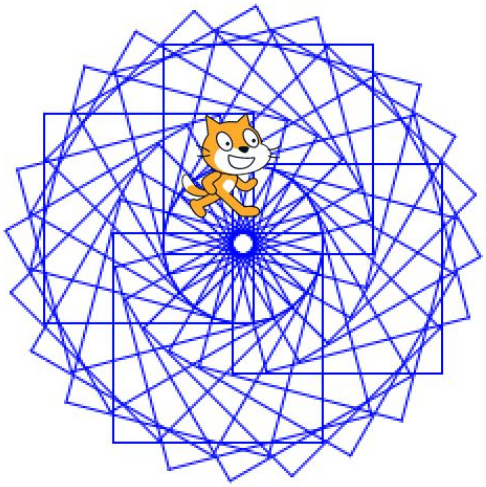




Maintenant, nous pouvons réellement nous amuser!

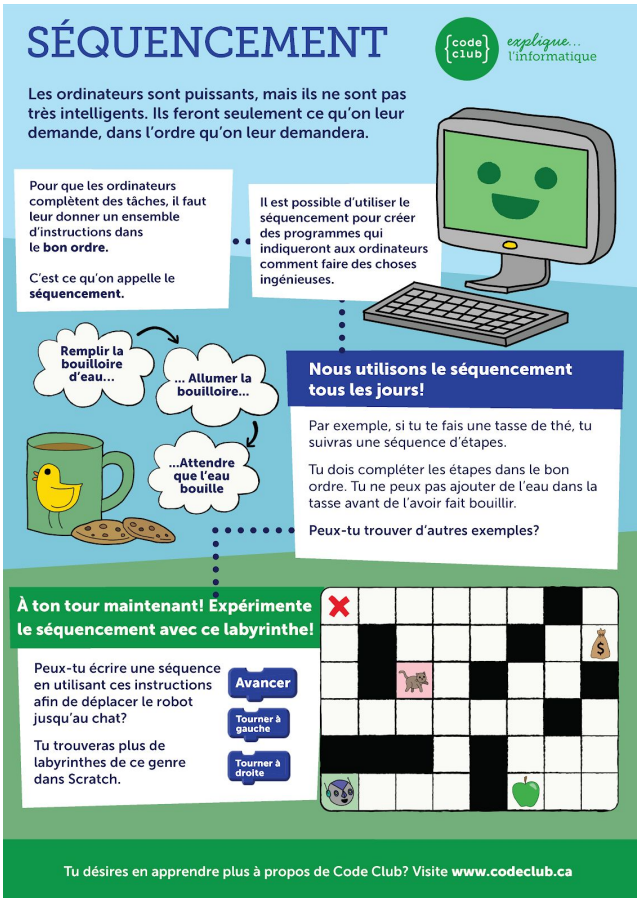
En glissant un bloc « carré » à l'intérieur de l'algorithme utilisé pour dessiner un cercle, les enfants vont enfin découvrir la raison pour laquelle il peut être plus amusant de programmer des dessins que de dessiner à la main.

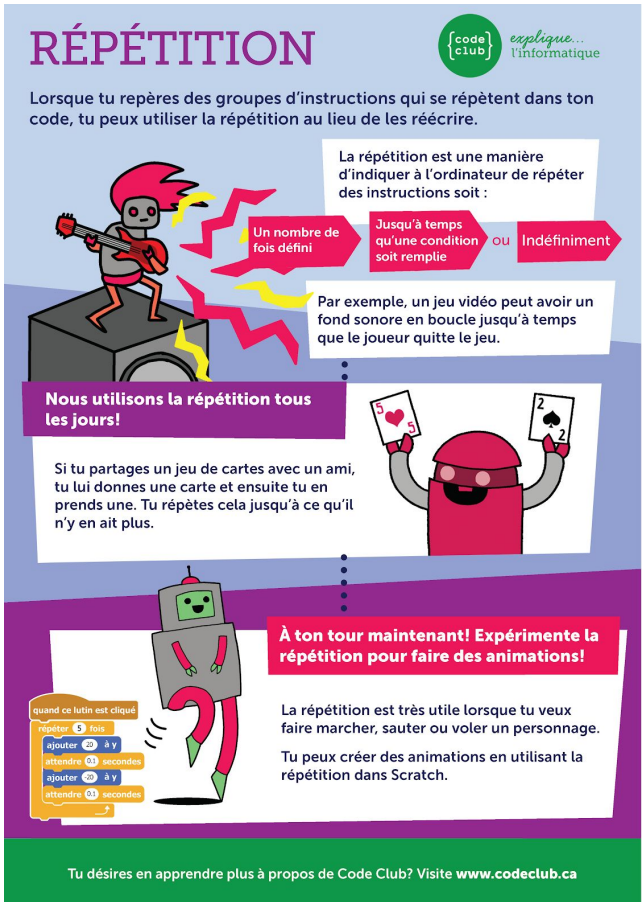


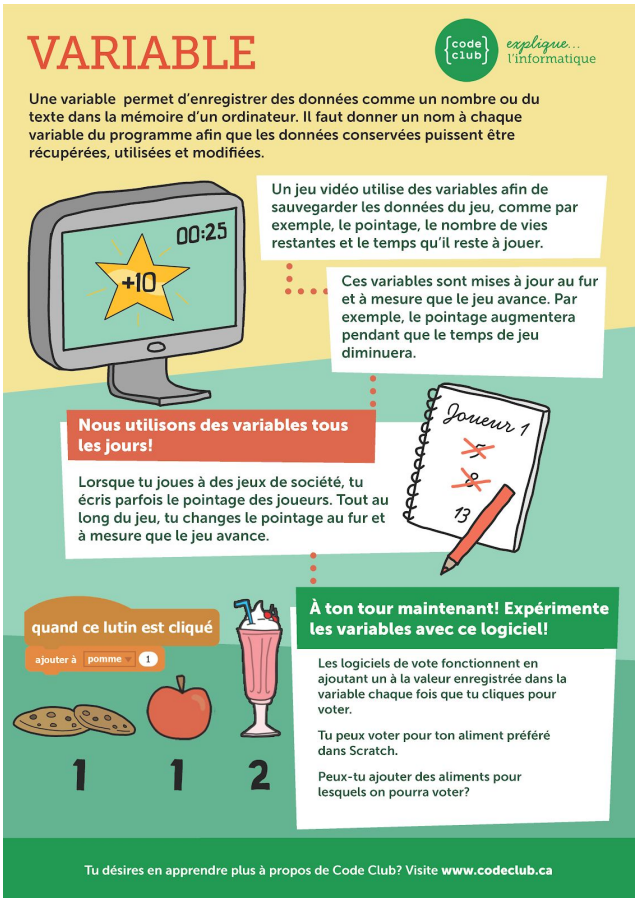


Essayez-le vous-même!

Exercez-vous à ce type de programmation Scratch avec ce plan de cours simple pour créer un spirographe. Ou bien développez d'autres compétences grâce aux projets contenus dans la banque de projets de Code Club Canada, que nous avons rendue accessible pour les bénévoles de partout au pays. Si vous mettez sur pied un club de programmation Code Club dans votre école, vous aurez également accès à un grand nombre d'affiches abordant des concepts essentiels. En voici quelques exemples!







Programmation textuelle

Introduire la programmation textuelle en classe est un défi de taille, même si vous êtes un·e expert·e en programmation. Cependant, lorsqu'ils ou elles ont les bonnes ressources, certain·e·s élèves et enseignant·e·s la préfèrent à la programmation par blocs.

Bien que Scratch est un excellent choix pour les apprenant·e·s visuel·le·s, des études ont démontré que la programmation active les zones du cerveau associées au langage encore plus que celles associées aux nombres et aux chiffres. Les enfants qui sont naturellement attiré·e·s par les textes et la lecture trouvent particulièrement passionnant l'apprentissage d'une langue qui ressemble beaucoup à de l'anglais, mais qui en diffère assez pour qu'ils et elles aient l'impression d'apprendre quelque chose de nouveau.

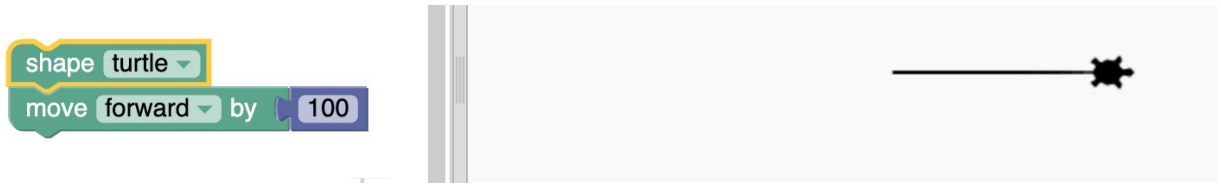
Fait amusant concernant la programmation textuelle : Les filles, qui n'ont peut-être pas eu la chance de grandir avec leurs propres jouets de construction (Lego ou autre), semblent apprivoiser particulièrement rapidement le langage **Python**. Python est un langage de programmation conçu pour suivre le plus fidèlement possible la syntaxe normale de l'anglais tout en utilisant les mêmes concepts élémentaires communs à tous les langages de programmation.

Effectuer la transition entre les blocs et le texte.

KCJ a eu l'immense chance de découvrir très tôt un outil spécifiquement créé pour décomposer ce défi auquel font face les enseignant·e·s.

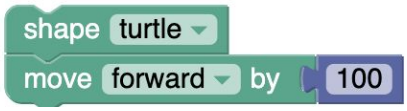
Trinket.io est une plateforme (en anglais seulement) conçue pour faciliter la transition entre la programmation visuelle par blocs et la programmation textuelle. Elle utilise une « bibliothèque de code » contenant des commandes semblables à celles utilisées par la tortue Logo et précodées en Python. Les enfants utilisent les commandes dans cette bibliothèque pour dessiner, de la même manière que nous avons utilisé un sprite pour dessiner dans Scratch.

Voici une séquence de base qui utilise les blocs de code de Trinket pour déplacer un stylo en forme de tortue :



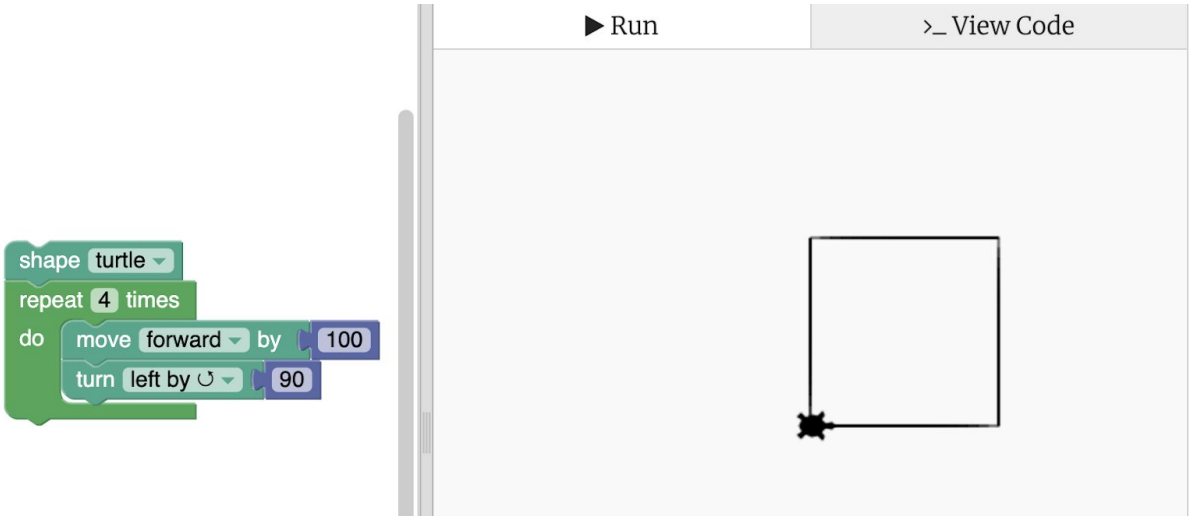
Trinket permet à ses utilisateur·rice·s de basculer rapidement entre les blocs de code et le code textuel. Ce que nous avons programmé avec des blocs est instantanément transformé en code textuel lisible. C'est un peu comme apprendre à lire les mots associés aux images avant d'affronter le défi d'apprendre à écrire.

Voici à quoi ressemble la séquence en langage Python :

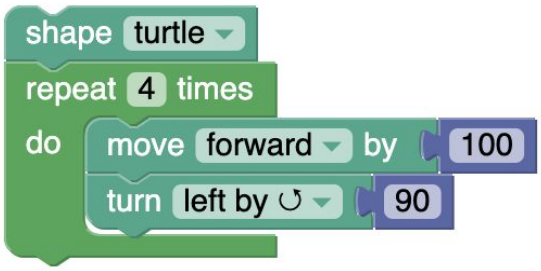



```
turtle.shape("turtle")
turtle.forward(100)
```

Examinons maintenant un carré dessiné grâce aux blocs de code de Trinket. Comme vous pouvez le voir, cela ressemble beaucoup à ce que nous avons programmé sur Scratch :



Maintenant, voyons à quoi ressemble la boucle de programmation en langage Python :




```
turtle.shape("turtle")
for count in range(4):
  turtle.forward(100)
  turtle.left(90)
```

Dans ce morceau de code, la ligne la plus complexe est la boucle commençant par « *for* ». Celle-ci ordonne à la tortue de répéter son mouvement quatre fois afin de former un carré. Cela dit, si vous avez un peu d'expérience dans la création de spirographes avec Scratch, il n'est pas trop difficile de deviner comment fonctionne cette boucle. Elle ordonne simplement à l'ordinateur de répéter les commandes contenues dans la séquence suivant le mot « for » autant de fois que nécessaire pour atteindre un compte (*count*) de quatre, puis de s'arrêter ensuite.

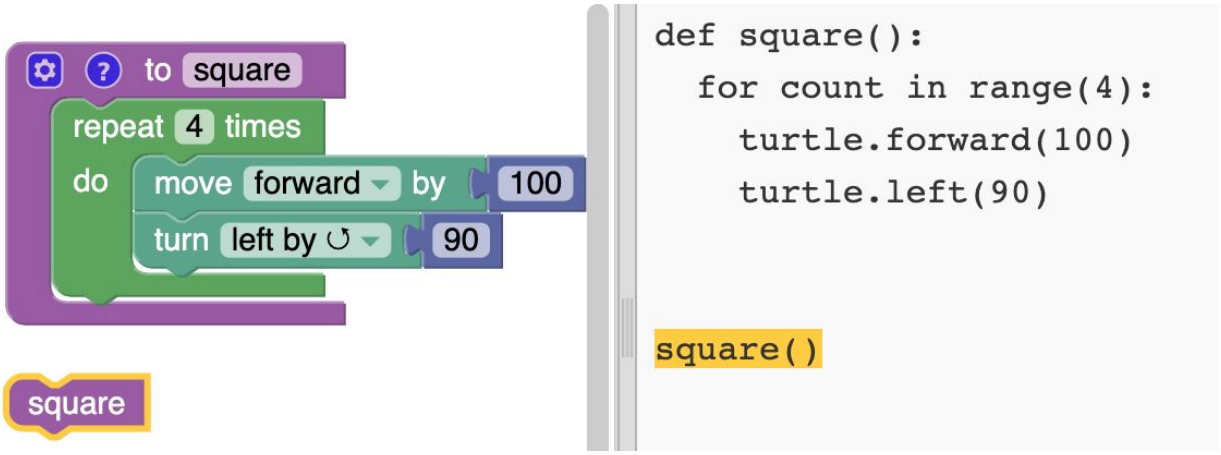
Une autre façon d'aborder la programmation en Python

Imaginez l'ordinateur comme une créature qui met énormément de temps à comprendre qu'elle doit faire quelque chose quatre fois. Elle doit donc compter à voix haute chaque fois. Puisque cette créature exécute la commande très rapidement, nous avons l'impression qu'elle comprend la commande mieux que nous. En fait, elle doit toujours silencieusement compter à voix haute.

En réalité, cette créature-ordinateur ne comprend pas ce qu'elle fait et ne le comprendra jamais. Nous devons donc écrire une séquence qui lui explique ce qu'elle doit faire, étape par étape. Dans ce cas-ci, cette séquence est : faire avancer la tortue, tourner de 90 degrés vers la gauche.

Définir un carré

Comme dans Scratch, nous pouvons utiliser les blocs de codes pour créer un bloc « carré ». Nous pouvons ainsi gagner du temps et nous éviter la tâche fastidieuse d'avoir à réécrire le programme au complet.



En programmation textuelle, nous appelons cela créer une « fonction ». Une fonction est tout simplement le nom qu'on donne à une séquence de commandes. Une fois que nous avons fait cela, nous n'avons plus besoin de réécrire la séquence.

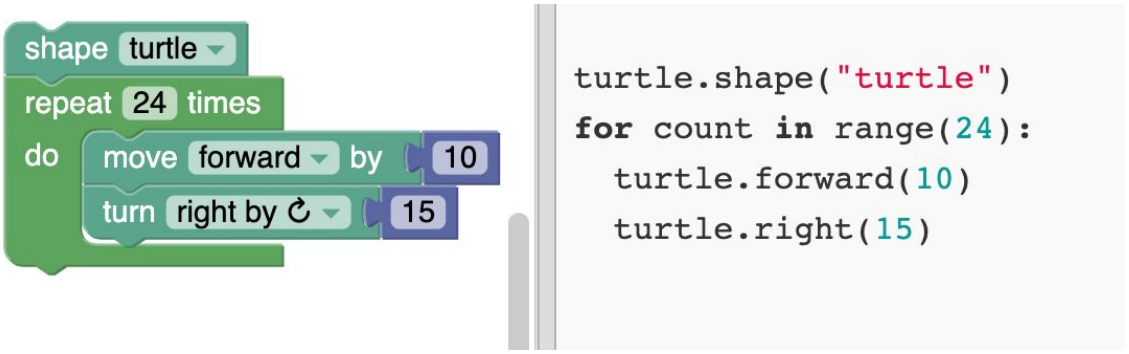
Comme vous pouvez le voir dans le code Python ci-dessous, quand nous donnons à l'ordinateur la commande « carré () » (« *square ()* »), l'ordinateur exécute l'ensemble de la séquence.

Nous obtenons ainsi une séquence que nous pouvons utiliser pour programmer le même spirographe que celui que nous avons programmé dans Scratch.

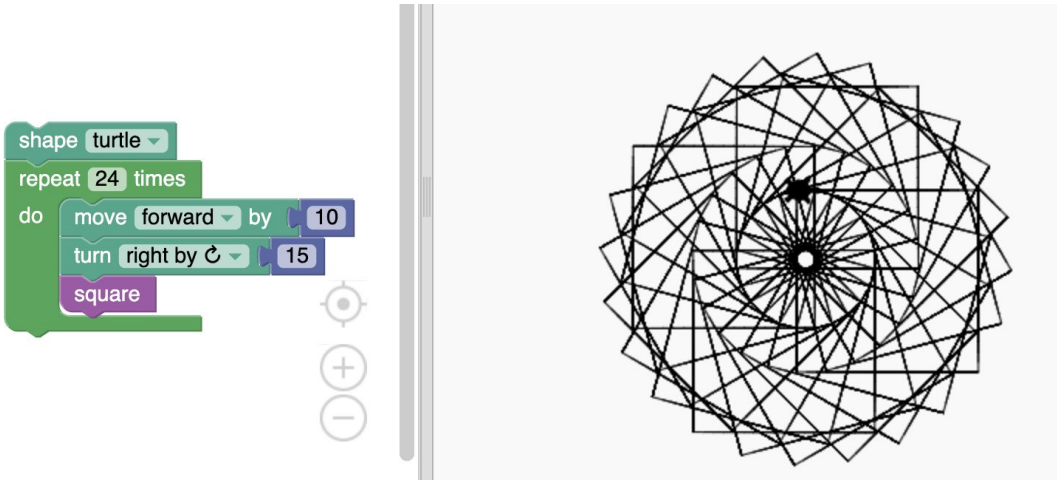
Programmer un spirographe en Python

Pour programmer le même spirographe que nous avons programmé dans Scratch, nous devons également programmer un cercle. Programmons-le de la même manière qu'avec Scratch, c'est-à-dire en utilisant les blocs « avancer » (« *move forward by* ») et « tourner » (« *turn right by* »).

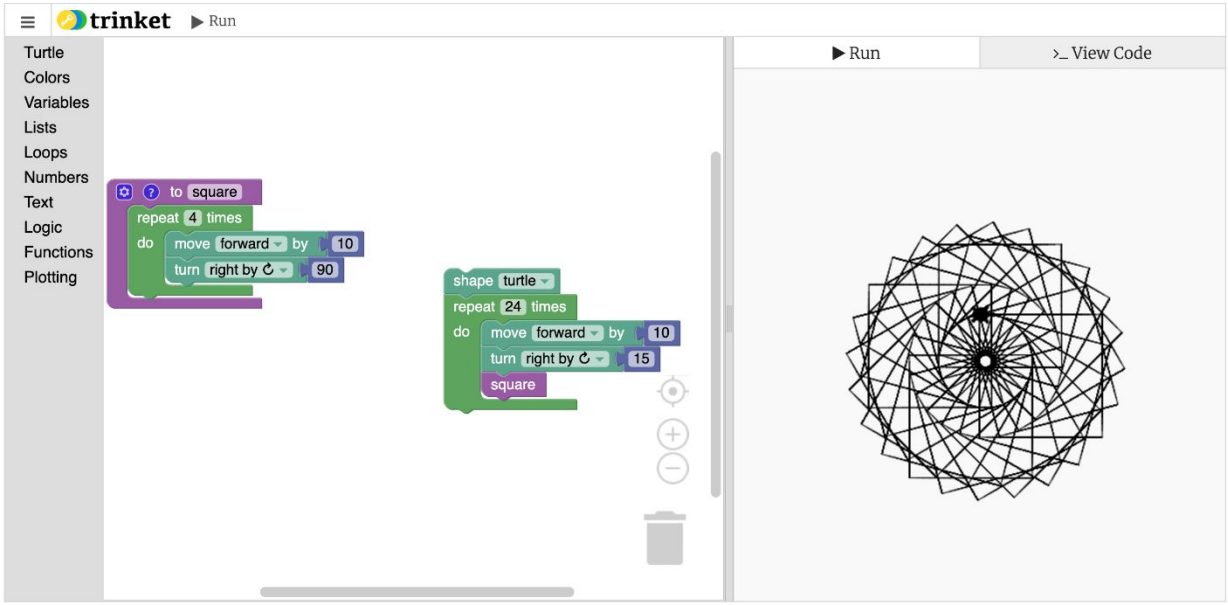
Ce qui nous donne ceci en Python :



Plaçons maintenant notre bloc « carré » à l'intérieur de notre boucle de programmation permettant de dessiner un cercle afin d'obtenir un spirographe!



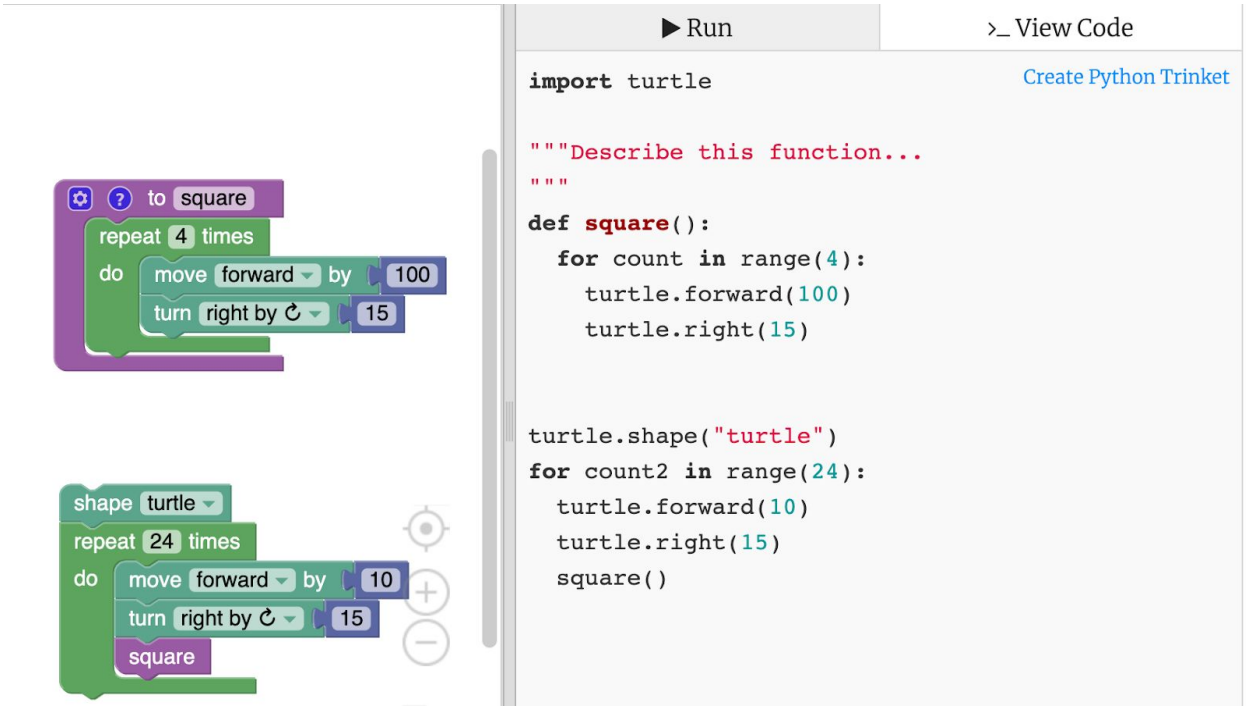
Voici à quoi ressemble ce spirographe sur la plateforme Trinket.io :



En langage humain, ces commandes se traduisent ainsi :

- Utiliser le stylo en forme de tortue
- Compléter la séquence suivante vingt-quatre fois :
    - Déplacer la tortue de dix pixels vers l'avant.
    - Faire pivoter la tortue de quinze degrés vers la droite.
    - Dessiner un carré tel que défini dans la fonction « carré »

Voyons à quoi ressemble le code du spirographe en Python :



Complexe, mais pas si compliqué.

Dans la boucle définissant le spirographe, la variable « count » s'appelle « count2 », afin que l'ordinateur ne confonde pas cette variable-ci avec celle définie dans la fonction « carré ».

Cela vaut la peine de noter que nous avons utilisé trois des quatre concepts élémentaires contenus dans notre cadre pédagogique pour apprendre à programmer : le séquencement, la répétition et les variables. De plus, nous avons également appris à créer une fonction (quoiqu'en vérité, nous avons appris cela précédemment lorsque nous avons créé notre propre bloc dans Scratch.)

Exercez-vous davantage à programmer en Python!

Voici un projet Roche, Papier, Ciseaux amusant qui initiera vos élèves et vous-même à l'écriture en Python, à la programmation sans blocs, ainsi qu'aux défis posés par la sélection et la programmation avec énoncés conditionnels.

Recréer le spirographe programmé précédemment dans Scratch est une façon d'introduire la programmation textuelle dans votre classe. Essayer quelques-uns des projets Python offerts par Code Club Canada en est une autre. Vous pouvez également visiter la section « Learn » (Apprendre) sur le site de Trinket.

Vous êtes maintenant en voie de devenir un·e débutant·e avancé·e. Bientôt, vous explorerez des concepts intermédiaires aux noms sophistiqués comme « opérateurs booléens » et « structures de données ». Ces concepts et plus encore peuvent être explorés sur le site de Code Club Canada, sur lequel vous trouverez également cette grille des compétences pratiques :

Scratch 1 | Scratch 2

	Projet 1: Groupe de rock	Projet 2: Perdu dans l'espace	Projet 3: chasseurs de fantômes	Projet 4: ChatBot	Projet 5: Boîte de peinture	Projet 6: Course de bateau
Séquençage	✓	✓	✓	✓	✓	✓
Répétition		✓	✓	✓	✓	✓
Variables			✓	✓	✓	✓
Sélection				✓	✓	✓
Opérateurs booléens					✓	✓
Structures de données						
Les fonctions						

Chapitre 4 : Aller plus loin

Planifier un atelier

KCJ a animé des milliers d'ateliers pour des enfants, des enseignant·e·s et des familles de partout au Canada, et nous avons appris presque tout ce que nous savons grâce à de précieux partenariats avec d'autres organisations comme Lighthouse Labs et avec des amis internationaux comme le groupe Lifelong Kindergarten du MIT Media Lab.

Des camps d'entraînements de Lighthouse Labs, nous avons appris l'importance de l'itération, c'est-à-dire l'importance d'essayer une activité le plus tôt possible en situation réelle et de l'améliorer selon les commentaires et les résultats obtenus.

Quant à Lifelong Kindergarten, ils nous ont appris **les trois principes fondamentaux d'une expérience créative d'apprentissage réussie :** des planchers bas, des plafonds hauts et de grands murs. Cela signifie que chaque atelier doit offrir : des défis assez faciles et des outils assez simples pour être accessibles à tous et toutes les apprenant·e·s (planchers bas), la flexibilité nécessaire pour permettre aux apprenant·e·s de surpasser les attentes (plafonds hauts), et des limites assez larges pour que les apprenant·e·s puissent intégrer leurs propres passions et faire preuve d'originalité, peu importe le projet (grands murs).

Nous incorporons cette pédagogie dans tous nos plans de cours et appliquons les principes de la conception inclusive pour nous assurer que tous les plans de cours sont accessibles à tous et à toutes, et qu'ils peuvent facilement être adaptés à tout type d'apprenant·e. **Nous offrons également notre** cadre pédagogique pour apprendre la programmation. Grâce à celui-ci, les enseignant·e·s peuvent obtenir plus facilement la certitude que leurs élèves atteindront des objectifs d'apprentissage qui rendront la programmation et l'art de programmer assez intuitifs pour les préparer à aborder des stratégies et des concepts plus avancés.

Utiliser les concepts, les pratiques et les perspectives

Dans l'élaboration de nos plans de cours, nous aimons mélanger concepts, pratiques et perspectives de façon à ce que les élèves aient la chance d'assimiler au maximum les compétences enseignées.

Un exemple de plan de cours

Principal *concept* de programmation abordé : répétition.
Principale *pratique* de programmation abordée : expérimentation.
Principale *perspective* à faire découvrir avec cet atelier : expression.

Un élève qui complète cet atelier sera en mesure de : comprendre comment utiliser la répétition pour améliorer un programme, démontrer qu'il ou elle a expérimenté avec la répétition dans l'écriture de son programme, savoir qu'il ou elle peut utiliser la répétition en programmation pour créer des programmes puissants, efficaces et créatifs.

Évidemment, cela ne veut pas dire que les élèves n'apprendront rien d'autre que les principaux concepts, pratiques et perspectives visés par l'atelier. Cependant, avoir des objectifs d'apprentissage et être en mesure de les communiquer aide à maintenir l'atelier sur les rails et le rend plus facile à gérer. Ceci vous permet également de déterminer plus facilement si vos planchers sont assez bas, vos plafonds assez hauts et vos murs assez grands.

À son meilleur, un atelier réussi devrait créer une variété d'artefacts prouvant l'aptitude, c'est-à-dire :

- La compréhension du concept;
- La connaissance nécessaire pour mettre ce concept en pratique;
- La reconnaissance que cette compétence peut être appliquée de façon utile à d'autres contextes.

Choisir le plancher, le plafond et les murs

Une fois que vous avez choisi vos objectifs d'apprentissage, suivez ces étapes :

1. **Préparez la classe avec une activité débranchée rapide qui réchauffera les aspects pertinents de la pensée computationnelle des élèves.** Par exemple, dans notre atelier appelé On bouge, nous recommandons que 2 élèves fassent une courte partie de robot aveugle devant la classe pendant que les autres élèves notent de quelle manière la répétition pourrait améliorer les commandes données au « robot ». Une telle activité fait germer dans l'esprit des élèves l'idée d'utiliser la répétition pour raccourcir un morceau de code et le rendre plus efficace. Elle permet également de démontrer que les erreurs font partie du processus et de présenter la programmation comme une activité sociale qui encourage les participant·e·s à rire et à s'amuser tout en essayant d'atteindre le résultat désiré. (Petit bonus, si vous rencontrez un problème technique quelconque avec une plateforme de programmation, les élèves peuvent continuer à jouer de façon débranchée pendant que vous essayez de résoudre le problème.)

2. **Choisissez trois courtes activités qui vont du plancher au plafond.** Celles-ci encourageront les élèves à intégrer leurs propres centres d'intérêt à l'intérieur de l'atelier. Voici trois exemples d'activités :

    - Activité 1 : Programmez une boucle et une condition d'initialisation très simples. Choisissez un sprite et/ou un arrière-plan qui représente un animal ou un endroit que les élèves aiment.
    - Activité 2 : Introduisez d'autres variables dans la boucle, telles que la taille ou un changement de couleur. Assurez-vous de définir une valeur par défaut pour la taille ou la couleur. De cette façon, lors de l'initialisation de la séquence, le sprite pourra revenir à son état initial.
    - Activité 3 : Ajoutez d'autres sprites. Essayez de placer des boucles à l'intérieur d'autres boucles et/ou créez un bloc qui nommera une séquence, vous permettant ainsi de simplifier votre programme.

3. **Laissez-vous du temps pour faire un retour sur l'activité.** Il est très possible que les élèves n'aient pas le temps de réaliser les trois activités, pour de nombreuses raisons. Cependant, même si vous avez seulement le temps de faire la première activité, laissez les élèves réfléchir à ce qu'ils et elles ont appris, partager les découvertes qu'ils et elles ont pu faire, et songer aux autres parties de leur vie dans lesquelles ils et elles font appel à la répétition et l'expérimentation.

4. **Établissez des connexions entre l'informatique et la société.** Songez à fournir à vos élèves des amorces de réflexions qui les encourageront à réfléchir aux différentes manières dont les ordinateurs utilisent les boucles et la répétition pour de bonnes raisons, mais pas toujours avec un résultat positif. Les jeux vidéo et les boucles de dépendance créées par plusieurs activités en ligne sont un sujet de conversation intéressant, plus encore si les élèves viennent de créer un artefact ludique. Les enfants feront peut-être des confidences qui les amèneront à moins ressentir le besoin de défendre leurs habitudes actuelles.

Animer un atelier

Cette liste des meilleures pratiques à suivre pour animer un atelier de programmation a été dressée à partir des pratiques utilisées tous les jours par les animateur·rice·s de KCJ pour appuyer les enseignant·e·s dans leur classe.

Meilleures pratiques pour la première heure

La première heure d'un atelier de programmation pour les enfants est à la fois exigeante et palpitante, mais également quelque peu angoissante si vous n'y êtes pas adéquatement préparé·e. Laissez-vous guider par le concept « d'amusement difficile » de Papert. Si vous réussissez à amener les élèves dans cette zone, ils et elles seront ravi·e·s d'affronter les défis et comprendront que ces défis font partie du processus d'apprentissage. Ces pratiques ont comme objectifs de donner le bon ton dès le départ, de maintenir les attentes à un niveau réaliste et d'établir les fondations qui vont nous permettre d'atteindre cette zone d'amusement difficile.

1. Soyez conscient de votre rôle

Au fil des années, nous avons appris à éviter de tomber dans le piège et de nous présenter comme des magicien·ne·s de fête foraine venu·e·s éblouir la classe avec des tours sensationnels. Mieux vaut mettre en place les conditions qui permettront aux élèves de créer leur propre magie. Donnez-leur des défis amusants et flexibles, maintenez leurs attentes à un niveau réaliste, et faites confiance aux élèves pour bien apprendre ensemble, voire s'apprendre les un·e·s aux autres.

2. Faites compter les activités débranchées

Utilisez des activités débranchées, même si elles sont courtes, pour donner le ton à l'atelier. Si les enfants sont détendu·e·s et intéressé·e·s dès le départ, ils et elles seront très probablement moins nerveux·euses lorsque viendra le temps d'aborder du nouveau contenu. Si un atelier Scratch demande de jouer au robot, assurez-vous d'apporter un bandeau. Les robots qui font *semblant* d'être aveugles sont beaucoup moins amusants. Peu importe la matière abordée, cette activité débranchée de dix minutes devrait encourager vos élèves à rire, à faire des erreurs et à apprendre de façon active.

3. Établissez un code de conduite

Communiquez vos attentes tôt, avant que les enfants se rendent en ligne. Donnez des consignes claires et insistez sur l'importance de protéger la vie privée, de respecter les autres participant·e·s et de publier seulement du contenu sans propos injurieux ni images violentes. Établissez un protocole pour obtenir l'attention des élèves, tel que compter jusqu'à trois et les obliger à avoir les mains sur la tête ou l'écran de l'ordinateur abaissé lorsque vous leur communiquez de l'information importante.

Établissez également un protocole pour demander de l'aide. Des notes auto-adhésives de différentes couleurs (par exemple, roses pour « j'ai besoin d'aide », vertes pour « je suis prêt·e à passer à la prochaine étape et continuer à apprendre ») permettent aux élèves de signaler qu'ils ou elles sont coincé·e·s sans avoir à arrêter de travailler et d'essayer de trouver une solution. Elles peuvent également vous aider à évaluer le meilleur moment pour passer à une autre activité. Gardez en tête qu'il s'agit ici d'un atelier d'apprentissage par la pratique. La créativité, l'exploration et, inévitablement, le bruit seront donc au rendez-vous. Pour apprendre, les élèves ont besoin de temps pour jouer. Par conséquent, encadrez-les, mais sans être rigide.

4. Ne présentez que quelques concepts à la fois.

Assurez-vous de bien doser l'ampleur du contenu et le rythme de l'atelier. Chaque nouveau concept, bloc ou conseil que vous amenez est un élément de plus qui doit être absorbé et compris à la fois par les élèves et par les éducateur·rice·s. On a longtemps cru que la mémoire moyenne pouvait retenir sept éléments d'information en même temps. Le consensus scientifique est maintenant que ce nombre tournerait plutôt autour de

quatre. Les ateliers de KCJ sont donc divisés en sections regroupant chacune quatre à cinq blocs. Ne courez pas le risque de provoquer une surcharge cognitive. Chaque nouveau bloc amène avec lui un nouvel ensemble de possibles problèmes. Maintenez donc le tout simple. Donnez du temps aux élèves pour s'exercer et jouer eux-mêmes ou elles-mêmes avec les blocs. N'essayez pas de leur enseigner tous les blocs requis d'un seul coup.

5. Programmez en direct

Soyez prêt·e à programmer devant la classe. Exercez-vous au préalable et assurez de savoir (plus ou moins) où vous devrez cliquer pour bâtir les programmes contenus dans l'atelier que vous enseignerez. Soyez à l'aise de réellement programmer *en direct.* N'ayez donc pas peur de répondre aux questions de vos élèves en insérant ou supprimant des morceaux de code au besoin. La programmation en direct, et tout ce qu'elle implique en termes de parenthèses, de pauses, d'accrocs et d'erreurs, envoient aux élèves un message subtil, mais important : la programmation est toujours un processus d'essais et d'erreurs. Un·e programmeur·euse ne travaille jamais en ligne droite du début à la fin.

Utilisez toutes les astuces existantes pour vous assurer que votre code soit lisible. Zoomez sur la zone de travail jusqu'à ce que le code soit facilement visible sur l'écran, le tableau ou le projecteur. Mentionnez que les blocs de commande sont de couleurs différentes selon la catégorie à laquelle ils appartiennent, ce qui aidera les débutant·e·s à trouver les blocs dont ils et elles ont besoin. Lorsqu'il est temps pour les élèves de programmer quelque chose, donnez des consignes claires concernant la marche à suivre. Assurez-vous que toute la classe comprend sur quel aspect concentrer ses efforts et quel objectif elle cherche à atteindre, particulièrement si vous vous lancez dans une activité avec un « plancher bas ».

6. Sachez quand braquer les projecteurs sur les apprenant·e·s plutôt que sur les éducateur·rice·s

À mesure que l'atelier avancera, un nombre grandissant d'élèves deviendront absorbé·e·s dans leurs créations naissantes et, par conséquent, moins intéressé·e·s par les défis additionnels que vous leur lancerez. Il deviendra alors peut-être plus difficile d'obtenir leur attention. Cette situation est prévisible et ne devrait pas être vue comme un échec de la part de l'enseignant·e, qui n'arriverait pas à captiver ses élèves, ou des

élèves, qui ne sauraient pas porter attention à leur enseignant·e. Au contraire, le désir de vos élèves de prendre le contrôle de leur propre apprentissage est un bon signe! C'est également le moment de demander à vos élèves d'expliquer à leurs pairs comment ils ou elles ont résolu un problème, de faire la démonstration d'une animation intéressante, ou de réfléchir à ce qu'ils et elles ont appris. Les élèves sont toujours plus à l'écoute lorsqu'ils et elles s'attendent à ce que l'attention se tourne ensuite vers eux et vers elles.

Meilleures pratiques pour la deuxième heure.

La deuxième heure d'un atelier offre son lot de nouveaux défis : comment renforcer ce qui est en train d'être appris, comment s'assurer que les élèves restent attentifs et comment gérer l'écart qui pourrait commencer à se creuser entre différents types d'apprenant·e·s.

1. Changez (métaphoriquement) de salle

Lorsque sonne la deuxième heure, commencez avec un nouveau plancher (c'est-à-dire avec un défi simple provenant d'une autre catégorie d'activité). Cela permet à ceux et celles qui apprennent plus rapidement d'essayer quelque chose de nouveau tout en donnant à ceux et celles qui apprennent plus lentement une nouvelle chance d'essayer quelque chose de facile. Si cela nécessite de réaliser une activité complexe de moins pendant la première heure, ainsi soit-il. Vous pourrez retourner à ces activités dans un cours ultérieur. Un bon atelier doit faire participer *tout le monde*, pas seulement les élèves les plus bavard·e·s, les plus rapides ou les plus expérimenté·e·s en programmation. Assurez-vous que les filles reçoivent autant d'attention, surtout celles qui sont jumelées avec un garçon sur le même ordinateur.

2. Soyez à l'affût de toute occasion de réviser et de bâtir sur des compétences acquises précédemment

Vous aiderez ainsi les élèves à solidifier la compréhension des compétences fraîchement apprises. Par exemple, demandez à vos élèves de nommer les blocs à utiliser pour créer une séquence d'initialisation ou pour tout autre technique de programmation élémentaire que vous avez abordée durant la première heure. Il n'est également pas mauvais de revenir sur le code de conduite ou les protocoles établis au début de l'atelier.

3. Enseignez aux enfants ce dont ils et elles ont besoin pour résoudre leurs propres problèmes

Ne touchez pas aux claviers de vos élèves et résistez à la tentation de leur donner la réponse immédiatement. Donnez-leur plutôt des suggestions puis laissez-les réfléchir seul·e·s aux défis présentés. Continuez d'encourager les élèves à apprendre les un·e·s des autres. Dirigez-les vers des ressources qu'ils et elles peuvent utiliser pour trouver de manière autonome les réponses recherchées. Une de ces ressources peut être la section d'aide ou le tutoriel de la plateforme utilisée.

4. Faites des erreurs

Souvent, et devant toute la classe. Apprenez comment faire des erreurs de manière élégante. En agissant ainsi, vous démontrerez que même les « expert·e·s » peuvent se tromper. Avoir la bonne attitude compensera tout manque de connaissances. N'ayez pas peur de demander à vos élèves s'ils ou elles peuvent voir le problème. Une fois que les enfants seront occupé·e·s à relever un nouveau défi, ils et elles oublieront tout malaise passager.

5. Prenez le temps de faire une pause et de revenir sur l'activité

Dans la deuxième heure, prendre le temps de s'arrêter et de réfléchir est primordial, tout comme le sont les idées et l'inspiration générées pendant cette pause. Les élèves ont besoin de temps pour réfléchir à ce qu'ils et elles viennent d'apprendre, faire la démonstration de leurs réussites et valider le caractère amusant de cette nouvelle activité. Réservez donc dix minutes à la fin de chaque session et donnez à la classe une à deux amorces de discussions, par exemple : les problèmes rencontrés et les solutions trouvées, ce que les élèves pensent de leurs réussites, les idées générées, ou encore la manière dont cette activité a changé le point de vue des élèves sur la programmation ou les ordinateurs.

Évaluation

L'approche de KCJ envers les évaluations

Chez Kids Code Jeunesse, notre objectif est d'apprendre aux élèves les concepts et pratiques de base dans le domaine de la pensée computationnelle et de la programmation. Nos plans de cours présentent des défis faciles et complexes qui laissent beaucoup de marge pour créer, bidouiller, expérimenter et collaborer. Le langage de programmation et la plateforme utilisés varient, allant de Scratch à des outils d'informatique physique tels que le micro:bit en passant par Python, JavaScript et plusieurs autres. Les concepts et les pratiques, cependant, sont universels.

Nous nous concentrerons donc ici sur deux suggestions de grilles pour évaluer les concepts et les pratiques. L'une est pour l'auto-évaluation des élèves et l'autre pour tout éducateur·rice se sentant assez à l'aise pour évaluer les compétences en programmation de programmeur·euse·s débutant·e·s. Les deux types d'évaluations sont essentiels à l'acquisition d'une bonne connaissance de ses propres capacités, particulièrement dans un programme scolaire axé sur les compétences. Les éducateur·rice·s qui sont eux-mêmes et elles-mêmes en train d'apprendre la programmation préféreront peut-être se fier à l'auto-évaluation tout en ayant l'objectif d'être mieux équipé·e à l'avenir pour juger le résultat d'un projet de programmation.

Comme nous en avons discuté dans la section sur les pratiques liées à la pensée computationnelle au chapitre 2, des feuilles ou cahiers de conception peuvent être utilisés par les élèves pour noter tout obstacle rencontré et toute démarche utilisée pour atteindre un résultat. Certain·e·s enseignant·e·s donnent même une partie des points en cas d'échec « productif ». Un échec peut être défini par le nombre d'essais effectués ou par l'atteinte d'un résultat plus intéressant que prévu, mais obtenu accidentellement. Encouragez les élèves à ajuster leur plan lorsque nécessaire.

L'exemple de grille d'auto-évaluation présenté ici a été conçu par les chercheur·euse·s de *Scratch Ed* à l'université Harvard. Elle s'utilise en tandem avec d'autres grilles qui évaluent d'autres pratiques reliées à la pensée computationnelle telles que le débogage et la programmation modulaire. Quant à elle, la grille d'évaluation par l'éducateur·rice a été développée par Mike Deutsh, directeur de la recherche et du développement en éducation chez KCJ.

Grille d'évaluation 1 : Auto-évaluation des pratiques liées à la pensée computationnelle

Pratiques liées à la pensée computationnelle (expérimentation et itération), auto-évaluation de l'élève

EXPÉRIMENTATION ET ITÉRATION	FAIBLES	MOYENNES	ÉLEVÉES
Décris la manière dont tu as construit ton projet.	L'élève a décrit de manière générale la création d'un projet, mais n'a pas donné de détails sur un projet précis.	L'élève a décrit de manière générale la création d'un projet précis.	L'élève a décrit de manière détaillée les différentes composantes d'un projet précis ainsi que leur développement.
Décris différentes choses que tu as essayées pendant que tu travaillais sur ton projet.	L'élève n'a pas fourni d'exemples précis de ce qu'il ou elle a essayé.	L'élève a fourni un exemple général démontrant qu'il ou elle a essayé quelque chose durant le projet.	L'élève a fourni des exemples précis de différentes choses qu'il ou elle a essayées durant projet.
Décris les corrections que tu as apportées à ton projet et la raison pour laquelle tu as fait ces corrections	L'élève a dit ne pas avoir fait de corrections, ou a dit avoir fait des corrections, mais n'a pas fourni d'exemples.	L'élève a décrit une correction qu'il ou elle a effectuée durant le projet.	L'élève a décrit toutes les corrections qu'il ou elle a effectuées durant le projet et a expliqué les raisons derrière celles-ci.
Décris un moment où tu as tenté quelque chose de nouveau.	L'élève n'a pas fourni d'exemples de moment où il ou elle a tenté quelque chose de nouveau.	L'élève a fourni un exemple général démontrant qu'il ou elle a tenté quelque chose de nouveau durant le projet.	L'élève a fourni des exemples précis de différentes choses nouvelles qu'il ou elle a tentées durant le projet.

Source : Harvard ScratchEd

Vous trouverez d'autres grilles d'évaluation de pratiques reliées à la pensée computationnelle ici (en anglais seulement) : bit.ly/CTrubric

Grille d'évaluation 2 : Évaluation par l'éducateur·rice

Concepts reliés à la pensée computationnelle, évaluation du projet de programmation d'un·e l'élève par l'éducateur·rice :

	4	3	2	1
Séquencement et boucles	Le code utilise des séquences et des boucles complexes (par exemple, en plaçant des boucles à l'intérieur d'autres boucles).	Le code utilise correctement des séquences et des boucles.	Le code est correctement séquencé, mais n'utilise pas de boucles.	Le code est incorrectement séquencé et utilise peu de répétitions ou des répétitions presque sans impacts.
Sélection et logique	La sélection (énoncés conditionnels de type si/alors) et la logique sont utilisées de manière complexe et/ou en tandem avec d'autres concepts tels que le séquencement, les boucles et les variables.	La sélection et la logique sont limpides et fonctionnent correctement	La sélection et la logique fonctionnent partiellement	La sélection et la logique ne fonctionnent pas.
Abstraction : variables	Le code utilise des variables (pour stocker, attribuer et utiliser des valeurs) de manière complexe, c'est-à-dire en combinaison avec d'autres concepts tels que le séquencement et les boucles, ou pour simplifier un morceau de code qui se répète plusieurs fois.	Le code utilise correctement des variables pour stocker, attribuer et utiliser des valeurs.	Le code utilise des variables incorrectement. Par exemple, le code stocke des valeurs, mais ne les utilise pas ou ne les met pas à jour correctement.	Le code n'utilise pas de variables.
Abstraction : algorithmes	L'algorithme utilise la programmation modulaire (les fonctions et l'attribution de noms à des séquences) de manière complexe, et/ou en tandem avec d'autres concepts.	L'algorithme est complet et utilise la programmation modulaire pour simplifier et commencer à augmenter sa complexité.	L'algorithme est complet et fait ce qu'il a été conçu pour faire, mais consiste en un seul bloc de code massif, sans aucune programmation modulaire.	L'algorithme est confus ou au moins partiellement inachevé.

Conclusion : Programmer pour le reste de votre vie

Au cours des 6 années d'existences de KCJ, nous avons reçu une foule de témoignages, mais c'est ce témoignage provenant d'une personne qui enseigne pour le district scolaire 71 de Victoria, en Colombie-Britannique, qui m'a convaincu que nous étions sur la bonne voie :

> *« Avant de participer à l'atelier, j'ai commandé deux livres sur Scratch pour la bibliothèque. Lorsque je les ai ouverts, j'ai constaté à ma grande déception que je ne comprenais absolument rien à leur contenu et je me questionnais sur l'utilité de ces ressources pour les élèves. De retour à la maison après l'atelier, j'ai à nouveau jeté un coup d'œil aux livres. J'ai alors eu l'impression d'apprendre à lire pour la première fois. Je comprenais parfaitement ce que les auteurs expliquaient aux élèves. J'ai maintenant la certitude que je peux intégrer ce matériel en classe. Merci! »*

Je comprends exactement ce que cette personne a ressenti. Ce sentiment, c'est le changement qui s'opère : le passage de l'anxiété ressentie la première fois que j'ai posé les yeux sur une ligne de code JavaScript et que j'ai essayé de terminer un tutoriel par moi-même, à un sentiment de limpidité. Personnellement, ce sentiment de limpidité a commencé à émerger lorsque j'ai réussi à déchiffrer cet entremêlement de texte avec l'aide de mon fils. Je sais maintenant qu'avec assez d'effort, petit à petit, je pourrais continuer à apprendre.

En tant que personne née bien avant que l'Internet devienne omniprésent sur toute la surface du globe, apprendre à programmer représente bien plus pour moi que l'apprentissage de nouvelles compétences. J'ai plutôt eu l'impression de passer d'immigrante au pays du numérique à citoyenne du numérique. J'ai ressenti la même chose, je crois, que ce qu'un·e immigrant·e ressent lorsqu'il ou elle apprend à lire la ou les langues officielles de son nouveau pays.

J'en suis venu à acquérir ce degré de « littératie algorithmique » grâce à une démarche dite « ascendante », soit la même démarche qu'entreprennent plusieurs

programmeur·euse·s et étudiant·e·s universitaires en informatique. J'ai entrepris cette démarche, car j'avais foi en l'utilité de ce que j'apprenais, mais il a fallu environ trois mois pour que j'acquière une vision « descendante » de la programmation. De plus, c'est seulement en apprenant le langage Python que je me suis aperçue que certains concepts étaient universels à tous les langages de programmation.

Laissez la curiosité vous guider et guider vos élèves

De plus en plus, on demande, et même oblige, les éducateur·rice·s à apprendre la programmation et à l'enseigner, par obligation envers les générations futures qui auront besoin de ces compétences pour occuper un emploi au 21e siècle. Mais la responsabilité des enseignant·e·s du réseau scolaire public n'a jamais été, et ne devrais jamais être, d'enseigner aux enfants des compétences professionnelles, surtout au primaire. De plus, ces enseignant·e·s ne reçoivent pas, et n'ont pas le luxe de se payer, 12 semaines de formation et ainsi apprendre la programmation à un tel niveau qu'ils et elles peuvent reconnaître les concepts enseignés.

Selon notre expérience, **les éducateur·rice·s sont bien plus enclins à apprendre la programmation lorsqu'ils et elles sont encouragé·e·s à le faire par curiosité**, ou pour améliorer leur propre perception de leurs aptitudes technologiques et de leur citoyenneté numérique. Les éducateur·rice·s sont d'autant plus confiant·e·s s'ils et elles ont une feuille de route qui s'intègrent aux multiples autres feuilles de route qui accompagnent les multiples programmes scolaires provinciaux. La plupart des provinces disposent d'un cadre pédagogique pour le développement des compétences basé sur une approche de type « connaître (concepts), faire (pratiques) et comprendre (perceptions) ». Nous espérons que notre guide aidera les éducateur·rice·s à intégrer ces nouvelles compétences dans leur propre programme provincial.

Changez les perceptions, changez le monde

Ce guide est simple, car nous croyons que les éducateur·rice·s ont besoin d'une feuille de route qu'ils et elles peuvent comprendre après douze *heures* d'étude, et non douze *semaines*. Un tel laps de temps peut sembler insuffisant pour démystifier quelque chose qui a été aussi occulté que la programmation, mais apprendre la programmation n'a pas besoin d'être compliqué.

Nous n'enseignons pas aux élèves toutes les compétences nécessaires pour devenir programmeur·euse·s professionnel·le·s. Nous leur enseignons plutôt toutes les compétences nécessaires pour qu'ils ou elles deviennent des programmeur·euse·s professionnel·le·s, si tel est leur désir. Le travail d'un·e éducateur·rice qui enseigne la programmation est d'abord et avant tout de trouver une manière d'apprécier la programmation puis, à l'aide d'évaluations honnêtes et de bons outils d'auto-évaluation, d'aider les élèves à acquérir la volonté et la certitude dont ils et elles ont besoin pour savoir qu'il est possible de s'améliorer.

Et si vous êtes un·e enseignant·e du primaire, votre travail est surtout de changer la perception que les élèves ont de ce à quoi ressemble un·e programmeur·euse. Tant et aussi longtemps que notre société continuera à alimenter le mythe que l'apprentissage de la programmation est, pour une raison ou pour une autre, destiné seulement aux personnes d'un sexe, groupe d'âge ou groupe socio-économique précis, nous resterons coincé·e·s dans un cercle vicieux culturel d'inégalité technologique.
Pour mettre fin à ce mythe, il est nécessaire que les éducateur·rice·s aient le courage de se présenter comme le genre de personnes qui peuvent apprendre la programmation et qui font de leur classe un espace où les filles et garçons de tous les milieux peuvent s'amuser ensemble à programmer.

Changer le monde commence par votre décision d'apprendre la programmation. Apprendre la programmation, c'est comme apprendre à faire du vélo : ça ne s'oublie jamais. Lorsque vous aurez goûté au sentiment d'émancipation que procure la programmation, vous et vos élèves ne l'oublierez jamais!

ANNEXE 1 : PLAN DE COURS DE KCJ

Plan de cours d'initiation à Scratch

On bouge

Ce plan de cours utilise Scratch pour explorer des concepts de programmation élémentaires tels que le séquencement et la répétition. Pour un défi supplémentaire, les élèves peuvent être redirigé·e·s vers des projets Code Club tels que Perdus Dans L'Espace, qui explore une variété de boucles de programmation. Puisqu'il aborde les blocs de répétitions et qu'il nécessite l'utilisation d'une grille cartésienne, il est facile de relier ce projet à des notions de base en mathématiques et en algèbre.

On dessine

Ce plan de cours aborde les mêmes concepts élémentaires, mais cette fois avec les blocs « stylo » de Scratch 3.0. Il s'agit d'une excellente façon d'imposer quelques limites sur les blocs disponibles sans pour autant limiter la quantité de défis auxquels les élèves plus expérimenté·e·s peuvent s'attaquer. Ce plan de cours peut aussi être relié ou servir d'introduction à la géométrie. Allez plus loin avec ce projet proposant de dessiner des logos que KCJ a conçu en l'honneur du 150e anniversaire du Canada.

On joue

Ce plan de cours aborde l'utilisation de données et le concept de variables en programmation. Il propose d'abord de créer un « podomètre » qui mesurera le nombre de pixels qu'un sprite peut parcourir, puis de faire le décompte des points dans le cadre d'un jeu simple. Les élèves désirant aller plus loin peuvent regarder du côté du projet Code Club S.O.S. Fantômes!

On jase

Explorer la logique, les énoncés conditionnels et de nouveaux concepts comme les « entrées » et « sorties » en créant un agent conversationnel qui vous demandera de vos nouvelles et réagira à vos réponses. Aller plus loin avec le projet Chatbot de Code Club.

Vous trouverez de nombreux autres plans de cours utilisant Scratch en visitant le site web de Kids Code Jeunesse.

Plans de cours d'initiation au langage Python

Python est un langage de programmation textuel tout usage très populaire. Grâce à Python, les élèves en apprendront davantage sur des concepts de programmation fondamentaux comme les boucles, les variables et les conditions. Ils et elles pourront créer leurs propres projets créatifs avec plus de rapidité et de puissance qu'en utilisant la programmation par blocs.

Ces plans de cours portant sur Python sont une excellente initiation à la programmation textuelle. Combinez-les avec les projets en Python de Code Club.

ANNEXE 2 : RESSOURCES COMMENTÉES (en anglais seulement)

Lifelong Kindergarten: Cultivating Creativity through Projects, Passion, Peers and Play
(Maternelle à vie : comment cultiver sa créativité grâce aux projets, à la passion, aux pairs et au jeu)
Mitchell Reznick, MIT Press, 2017.

Écrit par l'un des créateurs de Scratch, ce magnifique livre aborde la philosophie constructionniste derrière le mouvement « Programmer pour apprendre ». Face à un paysage scolaire qu'il décrit comme étant orienté sur les compétences et en train de rendre la maternelle trop semblable au reste du système scolaire (pensez aux cartes éclair, par exemple), Reznick propose plutôt de rendre le reste du système scolaire semblable à la maternelle. L'esprit créatif est mieux sollicité dans un contexte d'apprentissage par projet orienté sur les centres d'intérêt, la collaboration et, avant toute chose, le jeu.

Creative Computing Curriculum
(Programme scolaire d'informatique créative)
Harvard Graduate School of Education, 2019.

Élaboré par une équipe d'éducateur·rice·s menée par la chercheuse canadienne Karen Brennan, ce guide complet, pratique et basé sur la recherche aide à intégrer Scratch en classe. Il contient également des cadres pédagogiques et des grilles d'évaluation.

How Learning Works: Seven Research Based Principles for Smart Teaching
(Comment fonctionne l'apprentissage : Sept principes basés sur la recherche pour enseigner intelligemment)
Dirigé par Susan A. Ambrose, Michael W. Bridges, Michelle diPietro, Marcia C. Lovett et Mary K. Norman. Jossey-Bass, 2010.

Ce livre exceptionnel offre une méta-analyse des récentes études sur l'apprentissage. Chacun des sept principes est fondé sur des consensus ayant émergé au sein de la recherche en science de l'apprentissage. Certains principes, comme « les connaissances

préalables des élèves peuvent aider ou nuire à l'apprentissage », s'appliquent tout particulièrement au défi d'enseigner et d'apprendre la programmation. Le livre contient également des annexes utiles abordant l'auto-évaluation, la cartographie conceptuelle et les grilles d'évaluations.

Remerciements

Ce guide n'aurait pas été possible sans les multiples contributions des instructeur·rice·s et employé·e·s de KCJ, ni sans notre partenariat avec Lighthouse Labs, chef de file canadien des camps d'entraînement en technologie. Il n'aurait certainement pas été possible sans l'opportunité offerte par le ministère de l'Éducation de la Colombie-Britannique, une opportunité qui nous a permis de créer et de tester des modules éducatifs partout dans la province. Il n'aurait pas non plus été possible sans les opportunités offertes par le financement reçu de la part du ministère de l'Économie et de l'Innovation du Québec, ni encore sans l'important financement que nous avons reçu de la part du programme CodeCan du gouvernement du Canada pour transposer ce que nous avons appris à l'ensemble du pays.

Yasmin Ahmad a établi les bases de notre premier cadre pédagogique sur la pensée computationnelle, qui fut partiellement financé par l'Autorité canadienne pour les enregistrements Internet (ACEI). Notre instructrice-en-chef Meggie Carrier a grandement contribué à notre matériel sur la pensée computationnelle et ses meilleures pratiques. Mike Deutsch, directeur de la recherche et du développement, a conçu nos grilles d'évaluation. Wendy Hoy et Don Burks ont quant à eux conçu les activités débranchées utilisées par nos modules éducatifs en Colombie-Britannique et par nos plans de cours.

KCJ aimerait également remercier tous ceux et toutes celles qui, autant à l'interne qu'à l'externe, ont commenté et révisé ce document.

Nous sommes ouverts à toutes les questions et tous les commentaires. Faites-les parvenir à juliet@kidscodejeunesse.org.