DE *SCRATCH* À *PYTHON*, UNE TRANSITION DOUCE…

Michel CHEVALLIER[1]
IREM de Rouen

Jean-Luc DE SÉEGNER[2]
IREM de Rouen

Christelle PAISNEL[3]
IREM de Rouen

Résumé. Les logiciels de programmation utilisés dans les programmes de mathématiques appliqués en troisième et en seconde, à savoir *Scratch* et *Python*, sont très différents dans leurs syntaxes et dans leurs utilisations. Le passage de l'un à l'autre pose un véritable problème didactique. Nous analysons ces difficultés et proposons un retour réflexif sur la pédagogie à mettre en place pour envisager cette transition à l'entrée au lycée. Peu de publications récentes ayant été faites à ce sujet, nous nous appuyons sur un travail de recherche mené dans le cadre de notre groupe IREM, composé d'enseignants de mathématiques du second degré affectés en collège et en lycée. Nous avons construit, expérimenté en classe et analysé de nombreuses activités pour travailler l'algorithmique et la programmation en collège et en lycée, dans le cadre du cours de mathématiques.

Mots-clés. Transition, Scratch, Python, programmation, algorithme, variable.

Abstract. *Scratch* and *Python*, the two software programs used in the mathematics syllabuses in secondary schools, have very different syntaxes and uses. *Scratch* is meant for pupils in Year 10 who attend French collèges whereas *Python* is meant for Years 11 who attend their first year in lycées. Having to shift from *Scratch* to *Python* poses a real teaching problem. We analyse these difficulties and suggest some feedback about tools which can be put into action to ease the transition from one to the other. As very few recent publications on this subject are available, our IREM group which is composed of secondary school mathematics teachers working in collèges and lycées decided to carry out research. Thus, we built, tested and analysed numerous activities intended to work on algorithms and programming as part of the mathematics syllabuses in secondary schools.

Keywords. Transition, Scratch, Python, programming, algorithm, variable.

Introduction

L'introduction de l'apprentissage de l'algorithmique et de la programmation au lycée (MEN, 2009) a été antérieure à l'arrivée de la programmation au collège (MEN, 2015). Bien que les élèves de collège aient été confrontés à des algorithmes en mathématiques (algorithme d'Euclide, de construction en géométrie, opératoires, etc.), voire à leur implémentation dans un tableur, un logiciel de géométrie dynamique ou la calculatrice, aucun lien formel entre algorithme et programmation n'avait été établi et présenté aux élèves jusqu'en 2016. En 2010, les élèves arrivaient donc en seconde vierges de toutes connaissances dans ce domaine. Les enseignants de mathématiques disposaient ainsi de toute latitude pour introduire ces notions nouvelles.

Ces modifications de programmes de 2015 au collège ont conduit bien évidemment les collègues

[1] michel.chevallier@ac-normandie.fr

[2] jean-luc.de-seegner@ac-normandie.fr

[3] christelle.paisnel@ac-normandie.fr

de mathématiques à adapter leurs pratiques pédagogiques pour enseigner l'algorithmique et la programmation en seconde. Au collège, les élèves sont souvent invités à programmer des jeux comme le soulignent les programmes (MEN, 2018) : *« Exemples d'activités possibles : jeux dans un labyrinthe, jeu de Pong, bataille navale, jeu de nim, tic tac toe, jeu du cadavre exquis »*. Cependant, pour le Diplôme National du Brevet (DNB), les exercices proposés portent principalement sur des constructions géométriques ou des programmes de calcul.

En mathématiques, les programmes mis en place à la rentrée 2019 en seconde (MEN, 2019) ont bien pris en compte ce nouvel apprentissage mené au collège :

> *Au cycle 4, en mathématiques et en technologie, les élèves ont appris à écrire, mettre au point et exécuter un programme simple. Une consolidation des acquis du cycle 4 est proposée autour de deux idées essentielles :*
>
> *- La notion de fonction ;*
>
> *- La programmation comme production d'un texte dans un langage informatique.*

Ces programmes du lycée renforcent le lien entre mathématiques et informatique :

> *L'algorithmique a une place naturelle dans tous les champs des mathématiques et les problèmes ainsi traités doivent être en relation avec les autres parties du programme (fonctions, géométrie, statistiques et probabilité, logique)…*

Cette consolidation des acquis du cycle 4 affichée dans le programme de seconde s'accompagne cependant d'un changement de langage informatique. Le passage du logiciel *Scratch*, langage de programmation visuel et par briques, au langage *Python*, langage informatique textuel, nous semble constituer une première rupture qu'il est nécessaire de prendre en compte dans les apprentissages des élèves. Comment peut-on faciliter ce passage d'un langage à l'autre ? Utiliser un langage algorithmique intermédiaire constitue notre première hypothèse de travail. Cet article propose d'analyser les points d'appui et les points de vigilance pour assurer un passage en douceur. Les connaissances attendues en algorithmique et programmation au collège et au lycée sont sensiblement les mêmes : variable informatique, boucles, structure conditionnelle. Cependant le traitement de ces connaissances est fortement lié au langage utilisé. Différents points nécessitent une vigilance particulière et des activités adaptées : le typage de la variable avec *Python*, l'indentation, le traitement des boucles bornée et non bornée, le fonctionnement du bloc sous *Scratch* et de la fonction informatique de *Python*. Le choix d'un langage algorithmique proche de la langue naturelle mais adapté à cette transition peut servir de pont entre les deux langages. Nous n'étudions pas dans le cadre de cette transition les spécificités du langage *Scratch* (programmation événementielle, exécution en parallèle, …) qui n'apparaissent pas avec *Python* dans le cadre des programmes de seconde.

Les thèmes servant de supports à la programmation en troisième et en seconde constituent une autre rupture. Si le dessin est le principal support pour programmer avec *Scratch* au collège, ce n'est plus le cas au lycée avec le langage *Python* : en seconde, la géométrie étudiée est principalement analytique. Existe-t-il des thèmes mathématiques à privilégier pour introduire le langage *Python* ? Ce choix de thèmes adaptés à la transition constitue notre deuxième hypothèse de travail.

Avant de développer ces deux points, nous commençons par une analyse succincte des manuels scolaires de seconde afin d'observer sous quelle forme cette transition est proposée et nous présentons quelques éléments bibliographiques qui nous ont aidés pour cet article.

1. Des ressources qui ont alimenté notre réflexion

1.1. Des manuels scolaires de seconde

Lors de la lecture des ouvrages de seconde, nous nous sommes intéressés à la façon dont leurs auteurs ont appréhendé la transition entre *Scratch* et *Python*. Nous avons étudié la version papier de neuf manuels de seconde édités en 2019 suite aux changements des programmes du lycée et provenant de sept éditeurs différents.

Dans trois d'entre eux [1][2][3][4], la transition entre *Scratch* et *Python* est totalement absente et les parties traitant du thème de l'algorithmique débutent directement en langage *Python*. Un des manuels [1] ne consacre pas de chapitre spécifique à l'algorithmique et à la programmation mais des exercices sont proposés en lien avec les différents chapitres de mathématiques, s'appuyant certainement sur les préconisations des programmes qui indiquent que *« l'algorithmique a une place naturelle dans tous les champs des mathématiques »* (MEN, 2019). Pour les professeurs qui utilisent ce manuel, un travail de synthèse sur les notions vues en algorithmique et programmation dans les différents chapitres sera indispensable. Dans les deux autres manuels [2] [3], un chapitre complet est consacré à l'algorithmique et à la programmation. On y retrouve, sous la forme d'exemples, des algorithmes écrits en langage naturel à côté de programmes *Python* mais l'ensemble reste proche d'un catalogue d'instructions.

Dans deux autres manuels, le logiciel *Scratch* est évoqué très rapidement soit sur une page spécifique de rappels [4] mais sans lien avec le langage *Python*, soit dans la partie cours du chapitre d'algorithmique [5] pour faire cohabiter l'instruction *Python* étudiée, l'instruction *Scratch* déjà connue des élèves et le langage naturel. Une fois ces rappels très brefs effectués, tout le reste des activités et exercices est proposé en langage *Python*.

Dans les quatre derniers manuels, un réel effort pour traiter cette transition peut être relevé. Plusieurs activités de transition entre les deux langages sont proposées mais avec des états d'esprit différents. Dans un de ces manuels [6], cette transition s'effectue par le biais du logiciel *PyBlock*. Ce logiciel permet de programmer avec des briques ressemblant à celles de *Scratch*, mais qui sont sensiblement différentes, et d'obtenir la « traduction » du programme en *Python*. Le travail sur l'algorithme apparaît par la suite dans ce manuel. Dans un autre de ces manuels [7], des activités partent de programmes *Scratch* pour arriver à des programmes équivalents en *Python* avec une introduction progressive des différentes instructions. Cependant, le passage par l'écriture de l'algorithme en langage naturel n'est pas prévu dans les activités contrairement aux derniers manuels [8][9] qui prévoient des activités de transition en trois temps : le programme *Scratch*, écriture de l'algorithme en langage naturel, le programme en langage *Python*.

Nous constatons aussi que certains manuels [2][6][7] se démarquent en proposant un travail sur les entrées/sorties avant d'introduire la notion de fonction informatique tandis que d'autres [3][5] [8] travaillent immédiatement avec des fonctions. Des manuels proposent des activités intéressantes sur les types de variables [3][4][5] ou encore sur l'indentation [5][7] qui sont des nouveautés du langage *Python*. Le travail sur la logique, bien qu'explicitement présent dans les programmes scolaires, est peu développé dans les manuels. Pourtant celui-ci pourrait être nécessaire pour aborder le type booléen inscrit dans les programmes de seconde et le passage du « répéter jusqu'à … » de *Scratch* à l'instruction « while … » de *Python*. De plus, le langage *Python* utilise des bibliothèques, sujet peu abordé dans les manuels mais pourtant indispensable pour écrire certaines lignes de code.

[4] Voir l'annexe pour les références des manuels scolaires analysés.

1.2. Des résultats issus de la recherche en didactique des mathématiques

Il existe peu de recherches scientifiques sur l'enseignement de l'algorithmique et de la programmation en milieu scolaire mises à la disposition des enseignants. Pour cet article, nous nous sommes appuyés sur différentes parutions dont en voici une liste non exhaustive.

Dans sa thèse, Modeste (2012) propose dans un premier temps une analyse épistémologique du concept d'algorithme. Puis, dans un second temps, il s'intéresse à la transposition didactique à travers les instructions officielles, les documents d'accompagnement, les manuels scolaires de lycée et les ressources des sites IREM. Son constat est sans appel : en mathématiques, l'algorithme n'est considéré que comme outil au service de la discipline. Il propose enfin des pistes pour créer des situations qui doivent permettre d'étudier l'algorithme en tant qu'objet.

La thèse de Briant (2013) porte sur l'articulation entre algèbre et algorithmique et la place des algorithmes dans la résolution d'équations. S'appuyant sur de nombreux exemples, elle montre que l'algorithmique et la programmation devraient constituer un nouveau registre de représentations pour les objets de l'algèbre à condition de redéfinir précisément les savoirs à enseigner.

Davion (2019), dans son mémoire, pointe certaines difficultés liées au passage de *Scratch*, utilisé au collège, au langage informatique plus formel introduit en seconde. En particulier, elle remarque que l'utilisation et le fonctionnement de la variable informatique, constituent, pour ses élèves le premier problème qu'ils rencontrent dans l'écriture d'un programme. Elle analyse les différents rôles joués par une variable informatique et aborde le problème du typage, nouveauté en seconde. Elle présente quelques activités expérimentées en classe, activités qui s'appuient systématiquement sur l'écriture d'un algorithme en français pour passer du programme *Scratch* à celui traduit en *Python*.

S'appuyant sur leurs travaux de recherches en didactique des mathématiques et en psychologie cognitive, Lagrange et Rogalski (2017) s'interrogent sur les difficultés conceptuelles liées à l'enseignement de l'algorithme. Ils questionnent les concepts communs en mathématiques et algorithmique et pointent la variable informatique comme notion cruciale dans l'apprentissage de la programmation. Ils proposent une analyse fine de situations d'apprentissage et montrent ainsi la difficile articulation entre ces différentes « disciplines ».

Couderette (2016) met en évidence, à partir d'analyses de cas et de travaux de recherches antérieurs (Knuth, 1968), que la nature hybride de l'objet algorithmique n'est généralement pas perçue pour bon nombre de professeurs et leurs élèves. S'appuyant sur des activités analysées, l'auteure note que l'enseignante observée distingue l'aspect mathématique de l'aspect informatique. Quant aux élèves, ils privilégient le volet technique de l'aspect informatique. Elle rejoint en cela la position de Modeste (2012) et plaide pour que l'algorithme devienne un véritable objet d'enseignement.

Les travaux sur la transition de *Scratch* à *Python* sont bien plus rares encore. Les recherches menées par Branthôme (2021) portent sur les différences entre les deux langages *Scratch* et *Python* et analysent les niveaux de congruence entre les deux registres sémiotiques (Duval, 1993) liés à ces langages. Nous approfondissons cette comparaison dans la suite de l'article mais il nous semble important de souligner dès maintenant que les connaissances antérieures des élèves sur *Scratch* peuvent parfois constituer un obstacle dans l'utilisation de certains outils de *Python*.

2. Le rôle que pourrait jouer un langage algorithmique dans cette transition

2.1. Notre première analyse des deux langages

Scratch est un langage de programmation événementielle, en parallèle, orienté objet : le logiciel *Scratch* est visuel et utilise un ensemble de briques permettant d'accéder aux différentes instructions qui existent en plusieurs langues dont le français, habituellement utilisé avec les élèves de collège. Nous préférons employer le mot « brique » plutôt que « bloc » qui est réservé à un ensemble d'instructions sous *Scratch*. *Python*, quant à lui, est un langage textuel de programmation principalement séquentielle dans l'utilisation qui est indiquée dans les programmes de mathématiques de seconde : les logiciels qui permettent de programmer en *Python* ne sont pas, de prime abord, visuels bien que l'on puisse accéder à une fenêtre graphique spécifique. Ce langage utilise un ensemble d'instructions sous forme de mots clés principalement en anglais. Contrairement à *Scratch*, toutes ces instructions ne sont pas disponibles par défaut, certaines étant regroupées dans des bibliothèques ou modules à importer.

Un élève de seconde doit donc passer du registre des briques en français au registre de lignes de codes utilisant un lexique principalement en anglais. Il doit également connaître et mémoriser les instructions *Python* alors que toutes les briques de *Scratch* sont immédiatement disponibles via différents menus. De plus, certaines instructions symboliques ne sont pas forcément identiques, par exemple la puissance, la racine carrée ou le reste de la division euclidienne. S'ajoute à ces éléments une congruence souvent faible entre certaines commandes *Scratch* et *Python* (les entrées/sorties, la boucle non bornée, etc.) qui sera développée dans la suite de cet article.

Une première piste pour favoriser la transition serait d'utiliser un langage intermédiaire permettant d'aplanir les difficultés rencontrées. Des logiciels tels que *PyBlock* ou *Blockly* proposent des traductions automatiques d'un langage par briques qui ressemble à *Scratch* en des lignes de codes dans le langage *Python* : il n'y a pas alors d'apprentissage réel de *Python* et la prise en main nécessaire d'un nouveau logiciel peut présenter une difficulté supplémentaire pour certains élèves. Nous avons envisagé et expérimenté dans le cadre d'une formation l'utilisation du logiciel *Algobox* pour assurer la transition entre ces deux langages mais, à nouveau, la prise en main de ce logiciel a davantage constitué un obstacle qu'une véritable aide. Briant (2013), dans sa thèse, indique que *« si l'instrumentation n'est pas suffisamment avancée, le passage entre les environnements papier-crayon et informatique risque d'être délicat »* (Briant, 2013, p. 235). Elle souligne l'importance de la genèse instrumentale à construire pour surmonter ces difficultés. *Scratch*, par son usage du français, se rapproche du langage algorithmique tout comme *Algobox* : nous avons émis l'hypothèse que l'utilisation d'un langage algorithmique pourrait faciliter la transition de *Scratch* à *Python*. Gommer l'impact de la langue par un passage de *Scratch* en anglais ne réduit pas les difficultés de congruence entre les deux langages informatiques comme nous pourrons le vérifier en détail par la suite. Il n'est cependant pas question d'avoir recours à un langage algorithmique trop formel : l'utilisation de la langue naturelle constitue sans doute un compromis acceptable pour envisager la première étape d'une transition entre *Scratch* et *Python*. Cependant, nous avons orienté le choix de la forme du langage algorithmique utilisé de manière à favoriser autant que possible le passage d'un programme *Scratch* vers un algorithme ou d'un algorithme vers un programme en langage *Python*. Dans cet article, l'algorithmique envisagée est un outil au service de la transition entre les deux langages informatiques et non un objet d'enseignement proprement dit : nous nous focalisons sur le passage d'une programmation réalisée avec *Scratch* au collège à une programmation réalisée avec *Python* en seconde, dans le cadre du cours de mathématiques. Comme le souligne Modeste (2012) qui conclut dans sa thèse à propos de la place de

l'algorithme dans les instructions officielles :

> *L'étude des programmes de mathématiques du lycée et du document ressource pour la seconde révèlent une conception de l'algorithme assez éloignée de celle du savoir-savant. En particulier, nous avons montré que l'algorithme n'est pas considéré en tant qu'objet dans ces documents. Il est réduit à un rôle de mise en œuvre des procédures rencontrées dans les différents chapitres (effectivité). La notion de problème, avec différentes instances à résoudre, n'est pas toujours mise en jeu (algorithmes instanciés) et les algorithmes proposés ne sont pas toujours représentatifs du concept algorithme (programmes de modélisation-simulation par exemple). De plus, l'algorithme se voit attribuer une écriture extrêmement formelle mettant en jeu des éléments de programmation (programme-papier). Cela révèle une conception de l'algorithme comme étant orienté vers l'écriture de programmes (Algorithme Informatique Outil)* (Modeste, 2012, p. 235).

L'enseignement de l'objet algorithmique est un autre sujet que nous n'abordons pas ici.

Les instructions de contrôle de *Scratch*, par exemple, se présentent sous la forme de « pinces » matérialisant le début et la fin de l'instruction conditionnelle ou de la boucle. Le bloc se caractérise par un script détaché du script principal avec un début et une fin, facilement identifiables. *Python* a recours à ce qu'on appelle l'indentation : cette écriture s'appuie sur un « deux points » et une mise en retrait du texte lors du passage à la ligne suivante. La fin de l'instruction conditionnelle, de la boucle ou de la fonction se matérialise par la suppression de la mise en retrait du texte lors du passage à la ligne suivante.

Lors du travail de transition entre les deux langages, nous proposons une étape intermédiaire en langage algorithmique comme dans l'illustration ci-dessous afin d'aider les élèves à repérer et prendre conscience du début et de la fin d'une instruction de contrôle.

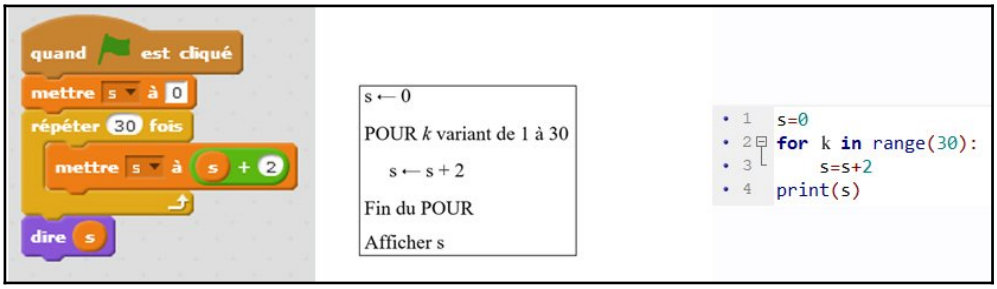


Figure 1 *: Traduction en langage algorithmique d'un script* Scratch *contenant une boucle bornée.*

Nous avons choisi d'y indiquer l'instruction « Fin du POUR » pour rappeler la brique en forme de « pince » utilisée dans *Scratch*. Ce choix permet une congruence forte avec le programme *Scratch* (excepté le « répéter 30 fois » qui est remplacé par « POUR »). Par contre, cette congruence s'affaiblit lorsque l'on passe du langage algorithmique à *Python*. En effet, la ligne indiquant la fin de l'instruction ne correspond pas à une ligne de code mais à la fin de l'indentation, difficilement perceptible pour un élève habitué à la « pince » de *Scratch* qui marque automatiquement la fin de la boucle ou de la structure conditionnelle. Ici, il faut instaurer un nouvel automatisme : « fin de l'instruction = fin de l'indentation ».

2.2. La variable informatique

La variable informatique joue un rôle primordial dans l'écriture et le traitement d'un programme informatique. Cependant, cette notion, introduite au collège, entre en conflit avec la notion de variable mathématique qui occupe une place importante en seconde. Davion (2019), dans les conclusions de son mémoire, précise les différents rôles de la variable informatique qui diffèrent

du fonctionnement de la variable mathématique :

> *Les élèves en début de seconde arrivent à créer des programmes simples dans lesquels la variable joue le rôle de donnée mais dès que le rôle de la variable change, cela génère des difficultés : les élèves ont du mal à créer une variable jouant le rôle d'intermédiaire dans la programmation et dont l'existence n'est nécessaire qu'à cause du système informatique, ils ont également du mal quand la variable a le rôle de compteur et qu'il faut penser à l'initialiser. On peut penser que cela est dû au fait que les élèves n'ont pas une bonne représentation de ce qu'est une variable et de la manière dont le système informatique fonctionne* (Davion, 2019, p. 20).

À propos de ces difficultés, elle ajoute :

> *En ce qui concerne la représentation de la variable, on peut proposer la représentation boîte/enveloppe aux élèves qui a l'avantage de lier l'algorithmique à la manipulation pratique. En ce qui concerne le fonctionnement du système informatique, une remédiation que l'on peut proposer est de détailler avec les élèves chaque étape effectuée lors de la mise en œuvre d'un programme informatique par un ordinateur* (*ibid.*, p. 20).

Dans la même idée, nous proposons un jeu théâtral sur la manipulation des variables dans l'environnement *Scratch* au collège[5] : il s'appuie sur la manipulation de boîtes étiquetées et d'ardoises représentant respectivement les variables et leurs valeurs lors de l'exécution d'un programme *Scratch*. Les activités que nous proposons permettent de mettre en scène et de vivre le déroulement du programme pas à pas.

De plus, le changement de langage informatique entre le collège et le lycée peut rendre plus délicat encore le travail autour de cette notion. En effet, comme nous allons le voir, le fonctionnement de la variable informatique sous *Scratch* (déclaration, saisie, affectation…) diffère sensiblement de celui dans *Python*. Une nouvelle fois, nous allons envisager un passage par le choix d'une forme de l'écriture algorithmique en langage naturel pour faciliter cette transition. Elle s'appuie sur la déclaration des variables et de leur type.

Un nouvel aspect de la variable informatique doit être introduit en seconde : le *type* (entier, flottant, booléen, chaîne de caractères). Bien que les élèves de collège aient été confrontés au choix entre « Créer une variable » et « Créer une liste » dans le menu « Données » de *Scratch 2* et qu'ils aient rencontré éventuellement des types lors de la création de blocs, aucun travail autour du typage n'est au programme. Ce langage distingue d'ailleurs variable et liste alors que bon nombre d'autres, dont *Python*, considèrent qu'une liste est une variable particulière[6]. Un travail nécessaire sur le typage doit donc être réalisé en classe de seconde.

Nous proposons d'aborder ce contenu du programme de mathématiques de seconde à partir de la saisie de données. Cette situation est fréquemment rencontrée au collège à l'aide de l'instruction « demander <message> et attendre ». La valeur saisie est alors affectée à l'attribut de classe (pseudo-variable) « réponse » sans avoir recours à la création d'une variable. Avec *Python*, la valeur obtenue par la saisie à l'aide de l'instruction « input(…) » est nécessairement affectée à une variable. Si plusieurs données doivent être saisies sous *Scratch*, le recours à des variables s'impose en utilisant l'instruction d'affection « mettre <variable> à réponse ». Il faut donc deux instructions avec *Scratch* pour affecter une valeur saisie à une variable alors qu'une seule suffit avec *Python* : « variable = input("message") ».

---

[5] https://irem.univ-rouen.fr/tuic/algocollege

[6] Nous pouvons à ce propos regretter que les listes ne soient pas au programme de seconde alors qu'elles peuvent être employées dans les programmes *Scratch* au collège

Lors de la saisie d'une valeur à l'aide de l'instruction « input » avec *Python*, cette valeur est par défaut typée chaîne de caractères. Si l'on veut saisir un nombre entier ou décimal, il convient d'utiliser les fonctions « int(…) » ou « float(…) » pour convertir la valeur saisie.

L'exemple de la figure 2 présente les instructions de saisie habituellement utilisées dans les deux langages. On perçoit bien une congruence faible de la forme avec le passage de deux briques à une seule ligne de code utilisant deux fonctions imbriquées. La difficulté implicite liée au typage de la variable a avec *Python* vient s'ajouter à cette rupture visuelle rendant la transition plus délicate.

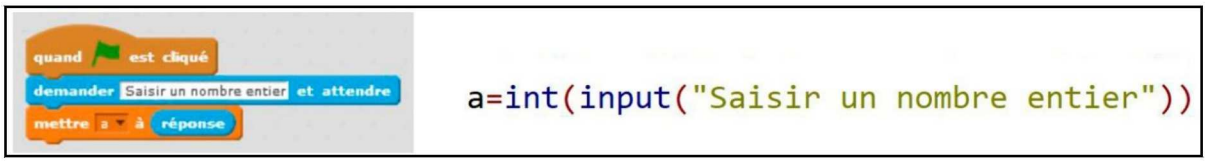


Figure 2 *: Traduction en* Python *d'un script* Scratch *contenant une instruction de saisie.*

Nous proposons donc de passer par une écriture algorithmique qui inclut la déclaration initiale des variables et précise leurs types (*cf.* figure 3).

Figure 3 *: Traduction intermédiaire en langage algorithmique du script* Scratch *de la figure 2.*

Nous retrouvons ici la nécessité de déclarer (sous *Scratch* : « créer ») la variable et nous introduisons la notion de type (entier, flottant, chaîne de caractères). La ligne « Début » permet de dissocier la déclaration des variables du script de l'algorithme proprement dit comme dans *Scratch* où l'action de « créer la variable » ne constitue pas une instruction présente dans le script. Il convient bien sûr d'indiquer la fin du script (complet) en langage algorithmique par le mot « Fin ».

Avec *Python*, l'instruction d'affectation « a = … » permet à la fois de créer (déclarer) et d'affecter une valeur à la variable a. Le type (entier) de la variable a est traduit à l'aide de la fonction « int(…) » (*cf.* figure 4).

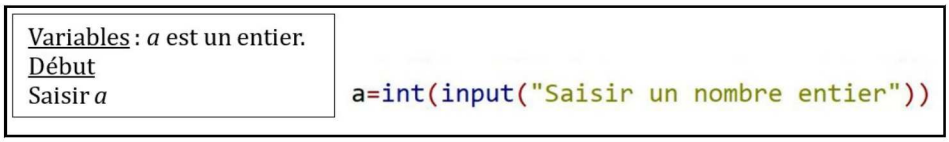


Figure 4 *: Traduction en* Python *de l'algorithme de la figure 3.*

On pourrait objecter qu'en *Python*, deux instructions distinctes, comme indiquées sur la figure 5, permettraient d'éviter l'imbrication et de rapprocher les deux langages informatiques.

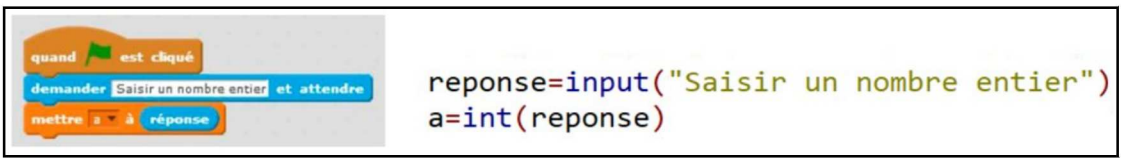


Figure 5 *: Traduction alternative en* Python *du script* Scratch *de la figure 2 utilisant « réponse ».*

On note que l'on voit deux lignes d'instructions. Cependant, aucune n'est la traduction exacte de l'autre : il n'y a ni congruence syntaxique, ni congruence sémantique. En effet, dans *Scratch*, le « demander » est une méthode qui permet d'attribuer automatiquement et de manière implicite une valeur à l'attribut de classe « réponse » qui n'est pas une variable informatique d'un lutin. Ensuite, la variable a est affectée de la valeur de cet attribut. Avec *Scratch*, il semble délicat de déterminer alors le type de cette variable (nombre ou chaîne suivant les instructions qui suivront). En *Python*, l'affectation de la variable « réponse » permet de déclarer la variable informatique et de lui affecter une chaîne de caractères (et pas un nombre entier comme indiqué). Ensuite, on affecte à la variable a la valeur de la conversion en un entier, obtenu à partir de la chaîne de caractères saisie.

Scratch ne prévoit pas de typage des variables (excepté pour les listes et éventuellement les entrées lors de la création de blocs) et renvoie peu de messages d'erreur, ce qui peut poser problème lors de l'écriture de certains programmes. Ainsi, une chaîne de caractères composée de chiffres peut se comporter comme un nombre (*cf.* figure 6). Il n'y a pas de distinction dans l'écriture d'une chaîne de caractères et d'un nombre : il n'y a pas de guillemets pour matérialiser une chaîne de caractères.

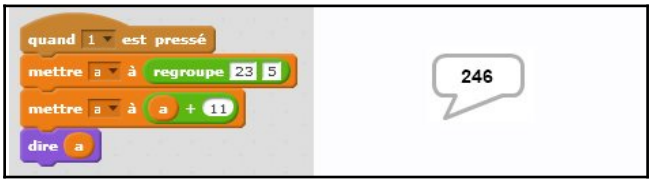


***Figure 6** : Script* Scratch *montrant l'absence de typage des valeurs utilisées.*

Si la chaîne contient des lettres, la valeur renvoyée est un nombre (0) ou une chaîne de caractères suivant les opérateurs utilisés (« + » ou « regroupe »), comme dans les deux exemples de la figure 7 :

***Figure 7** : Variables identiques et résultats différents selon le type imposé par le traitement des variables.*

On obtient un message d'erreur (« NaN » : not a number) lorsqu'on souhaite ajouter un nombre à une chaîne de caractères avec l'instruction « ajouter à … … » (*cf.* figure 8) :

***Figure 8** : Message d'erreur lorsqu'une chaîne non numérique est traitée comme un nombre.*

Comme *Scratch*, *Python* est un langage au typage dit dynamique, que montre la figure 9.

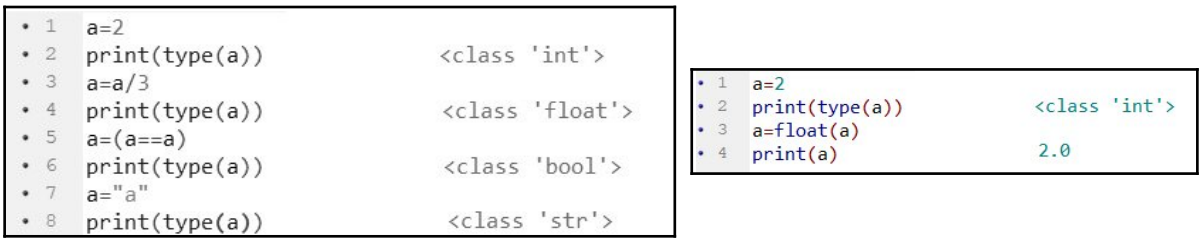


Figure 9 *: Typage dynamique dans* Python *: le type d'une variable peut changer au cours du script.*

Cependant, sous *Python*, le type de variable est défini lors de l'affectation et est conservé lors de l'utilisation de l'opérateur « + ». Ainsi, contrairement à ce qui se passe dans *Scratch*, une chaîne de caractères composée exclusivement de chiffres ne pourra être ajoutée à un nombre.

Le type de la variable a ci-dessus s'adapte aux affectations successives. La fonction « type(…) » permet de renvoyer le type de la variable. *Scratch* ne dispose pas d'instruction de ce genre et ne permet donc pas d'accéder au type de la variable.

Nous constatons aussi, dans les exemples précédents, que les deux langages proposent des instructions d'affectation très proches : « mettre <variable> à valeur » pour *Scratch* et « variable = valeur » pour *Python*. La difficulté réside cependant dans ce signe « = » qui n'a pas le même sens dans ces deux langages : test d'égalité pour l'un et symbole d'affectation pour l'autre. Le test d'égalité en *Python* requiert l'utilisation d'un double signe égal. C'est une difficulté syntaxique supplémentaire relevée par Davion (2019).

Sous *Scratch*, une autre instruction d'affectation est disponible : « ajouter à variable <valeur> ». Cette instruction est souvent privilégiée au collège car plus naturelle pour les élèves, en particulier dans des programmes de jeux comme, par exemple, « ajouter à compteur 1 » ou « ajouter à vies −1 ». Dans le programme *Scratch* proposé dans la partie 2.1., l'instruction « mettre s à s+2 » serait habituellement remplacée par « ajouter à s 2 » plus simple.

Dans l'instruction « mettre s à s+2 », la lettre s, qui apparaît deux fois, joue des rôles différents et constitue une difficulté réelle pour les collégiens : la première lettre s rencontrée lors de la lecture désigne une variable alors que la seconde désigne la valeur contenue dans la variable avant le traitement de l'instruction. Visuellement, l'élève voit deux lettres s identiques alors qu'il y a deux signifiés pour un seul signifiant. Néanmoins, le fonctionnement de la variable informatique doit constituer un objectif vers lequel il faut tendre en 3e afin de faciliter cette transition vers *Python*.

L'instruction en *Python* « s=s+2 » présente aussi cette difficulté autour de la variable informatique et sa valeur. Mais s'ajoute également le conflit avec l'écriture mathématique identique dans laquelle la lettre s, présente deux fois, constitue la même inconnue ou variable : un signifié pour un signifiant. L'utilisation du signe « = » renforce la proximité avec les écritures mathématiques et accroît les difficultés : un nouveau sens de ce symbole (affectation) s'ajoute aux différents sens déjà rencontrés en mathématiques (Chevallier, Paisnel & De Séegner, 2018). Un travail doit donc être mené en parallèle sur les statuts des lettres et du signe « = » en mathématiques et sur la notion de variable informatique. L'inversion de chronologie entre l'écriture (et la lecture) de l'instruction et son traitement informatique constitue enfin une autre difficulté : alors que la lecture s'effectue de gauche à droite, l'instruction s'exécute en effectuant le calcul s+2 avant d'affecter le résultat à la variable s. Ici encore, l'écriture de l'algorithme peut faciliter le passage d'un langage à l'autre. En effet, l'écriture « s←s+2 », proche de l'instruction *Scratch*, gomme, dans un premier temps, l'utilisation du signe « = » et la

flèche (symbole recommandé dans les instructions officielles) oriente la lecture de l'affectation.

On pourrait éventuellement passer directement de l'instruction *Scratch* « ajouter à s 2 » à l'écriture *Python* « s+=2 ». Nous pensions *a priori* que cette traduction poserait problème aux élèves car elle n'apparaîtrait pas naturelle avec la juxtaposition des symboles « + » et « = ». En réalité, nos expérimentations ont montré que cette écriture était acceptée assez facilement. En effet, elle présente l'avantage de gommer l'un des deux « s » et sa lecture « la valeur de s augmente de 2 » est plus simple que « s prend la valeur de s+2 ». Mais cette traduction est purement technique et comporte des implicites : on se focalise visuellement sur le calcul de s+2 et on passe au second plan l'affectation du résultat dans la variable s. L'écriture algorithmique « s←s+2 » permet de construire les deux composantes de cette instruction particulière (calcul puis affectation). Il convient sans doute de faire usage avec les élèves des deux notations « s=s+2 » et « s+=2 » dans la suite de l'apprentissage de la programmation en *Python*.

2.3. La structure conditionnelle

Dans l'utilisation d'une structure conditionnelle, on constate une congruence forte entre les écritures de cette structure avec *Scratch*, l'algorithme et *Python*, comme le montre l'exemple de la figure 10.

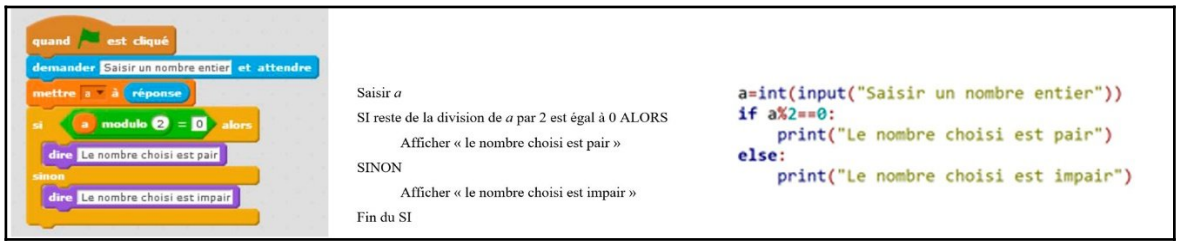


Figure 10 *: Congruence entre les trois écritures (*Scratch*, algorithme,* Python*) dans le cas d'une structure conditionnelle.*

On retrouve la vision en forme de « pince » déjà évoquée au début de cet article. Cette congruence s'affaiblit du fait du passage à l'indentation avec *Python* qui efface la fin de la structure conditionnelle. Cette manière de marquer le « alors » de la condition et la fin de la structure conditionnelle reste peu visible pour un élève : elle n'est pas repérable tant que l'on reste en débranché. Suivant le logiciel utilisé, une erreur d'indentation peut ou non être repérée lors de l'écriture du script (surlignée ou soulignée en rouge) ou lors de l'évaluation syntaxique du script lorsque l'exécution est demandée (message d'erreur). Malgré la congruence forte, l'indentation reste un obstacle important (*cf.* figure 11).

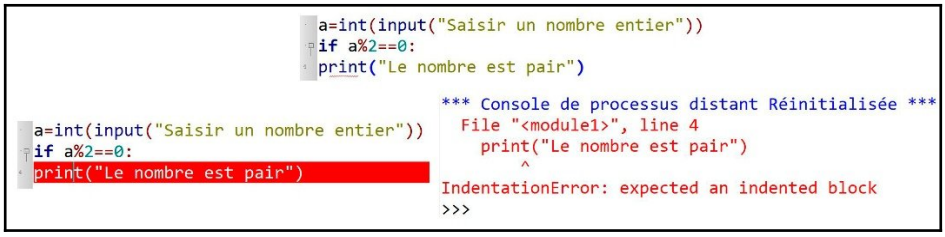


Figure 11 *: L'indentation : une source d'erreur.*

La condition utilisée est *a priori* un booléen. Avec *Scratch*, la forme des briques impose l'utilisation d'un opérateur booléen de type égalité, infériorité, supériorité, connecteurs logiques d'opérateurs booléens (ET, OU, NON). Le langage textuel *Python* n'a pas cette contrainte : il est

sous-entendu que la condition écrite par l'utilisateur doit être de type booléen et on pourrait s'attendre à ce que *Python* renvoie un message d'erreur dans le cas contraire. Pourtant, l'exemple de la figure 12 est assez révélateur d'un certain dysfonctionnement suite à une erreur d'écriture de la condition (comme pourrait le faire un élève).

Lorsqu'on exécute ce programme pour n'importe quel entier, le message affiché est toujours le même : « le nombre choisi est pair ». *Python* ne renvoie pas de message d'erreur alors que la condition n'est pas un booléen. Bien évidemment, cette chaîne de caractères qui sert de condition n'est, *a priori*, ni vraie, ni fausse (ce n'est pas un booléen).

```
a=int(input("Saisir un nombre entier"))
if "le reste de la division de a par 2 est égal à 0":
    print("le nombre choisi est pair")
else:
    print("le nombre choisi est impair")
```

Figure 12 *: Utilisation dans une condition d'une phrase en langage naturel, interprétée comme un booléen « vrai » par* Python.

On ne peut s'empêcher de penser qu'elle est considérée ou « évaluée » comme vraie par le langage, et pourtant quelques manipulations confirment que cette chaîne de caractères n'est ni vraie, ni fausse (*cf.* figure 13).

```
>>> "le reste de la division de a par 2 est égal à 0"==True
False
>>> "le reste de la division de a par 2 est égal à 0"==False
False
```

Figure 13 *: Tests permettant de constater qu'une chaîne de caractères ne correspond pas, en* Python*, à une valeur booléenne « vrai » ou « faux ».*

En réalité, la machine ne compare pas à « vrai » ou à « faux » mais elle effectue une comparaison par rapport à 0, c'est-à-dire à faux. En fait, elle effectue une comparaison avec « n'est pas faux » au lieu de « vrai ». La chaîne de caractères n'est pas « fausse » et donc le programme exécute uniquement l'instruction qui suit le « if ».

La condition est souvent assimilée à tort à une proposition avec une valeur de vérité « vrai » ou « faux » alors que le langage réalise une comparaison à « n'est pas faux » : la logique mathématique et celle utilisée en informatique diffèrent, affaiblissant la congruence. L'absence d'un message d'erreur est regrettable. Cependant, il convient sans doute de continuer à parler de condition vraie ou de condition fausse en classe en veillant à écrire des conditions de type booléen.

Ainsi une condition peut être considérée comme une proposition au sens de la logique. Cependant dans l'écriture de l'algorithme, une telle condition s'apparente davantage à un prédicat dont la valeur de vérité dépend de la valeur des variables utilisées. Lors de l'exécution d'un programme, ces variables sont instanciées et le prédicat devient bien une proposition. On constate à nouveau une distorsion entre logique mathématique et logique informatique.

Nous pensons que tous ces éléments un peu délicats doivent être connus de l'enseignant mais ne sont pas nécessairement à présenter aux élèves de troisième et de seconde, quel que soit le langage utilisé : ils lui permettent d'expliquer certaines erreurs rencontrées dans les travaux des élèves.

Il reste une autre différence entre *Scratch* et *Python* autour de la structure conditionnelle. Lorsque plusieurs conditions sont à programmer, les deux langages, tout comme l'algorithme, permettent l'imbrication des tests conditionnels. Le langage *Python*, quant à lui, propose une écriture simplifiée utilisant l'instruction « elif » suivie d'une nouvelle condition et d'une indentation. Néanmoins, une transition avec des instructions imbriquées sous *Python* est nécessaire pour introduire « elif ».

2.4. La boucle bornée

Pour les boucles bornées, la congruence est faible entre *Scratch* et *Python* : on doit passer d'une instruction « répéter … fois » à une instruction « for ». La nécessité de passer par le langage algorithmique s'impose pour aplanir certaines difficultés : nous avons choisi d'utiliser une instruction « POUR » dans l'algorithme au lieu d'un « Répéter » avec l'introduction d'une variable compteur — variable pourtant absente de l'instruction dans *Scratch* — pour se rapprocher de l'instruction en *Python*. Reprenons pour cela un extrait du premier exemple proposé dans cet article pour analyser le passage de *Scratch* à l'écriture algorithmique (*cf.* figure 14) :

***Figure 14** : Traduction en langage algorithmique d'un script* Scratch *contenant une boucle bornée.*

L'instruction de contrôle « répéter … fois » se présente sous forme d'une pince qu'on retrouve dans le langage algorithmique avec la ligne de début (« POUR … ») et la ligne de fin (« Fin du POUR »). Par contre, la congruence sémantique est faible. *Scratch* utilise une variable transparente qui compte les tours de boucle (on ne la déclare pas et on n'y a pas accès) alors qu'il faut déclarer ce compteur dans les autres langages. De plus, dans un algorithme ou en langage *Python*, cette variable k se trouve incrémentée automatiquement à chaque boucle sans que soit écrite l'instruction « k = k + 1 ».

Les trois lignes de l'algorithme se traduisent en *Python* par deux lignes de code : nous retrouvons ici (*cf.* figure 15) le problème de l'indentation qui représente une difficulté réelle pour les élèves.

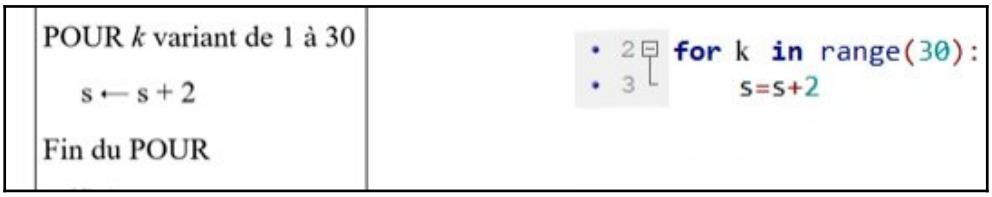


***Figure 15** : Différences entre le langage algorithmique et* Python *: deux lignes au lieu de trois et indentation nécessaire.*

Par exemple, un exercice, comme celui de la figure 16, proposé en classe, permet de mettre en évidence les effets de l'indentation dans l'écriture d'un programme : l'exécution de ces deux programmes produit des résultats éclairants.

```
a=5                          a=5
s=0                          s=0
for k in range(30):          for k in range(30):
    s=s+a                        s=s+a
print(s)                         print(s)
```

***Figure 16** : Effets de l'indentation sur deux scripts* Python *presque identiques.*

La congruence sémantique entre les deux langages pour la boucle bornée est ici moyenne. Certes, nous retrouvons le « POUR k » traduit par « for k ». Par contre, le « range(30) » ne permet pas d'affecter à la variable des valeurs de 1 à 30 mais les valeurs de 0 à 29 inclus. Dans cet exemple, le compteur de tours de boucle n'est pas repris dans les instructions contenues dans la boucle, on peut donc évacuer dans un premier temps cette difficulté. En effet, ce « range(30) » est proche de l'expression « 30 fois » contenue dans l'instruction *Scratch* et permet bien d'effectuer 30 tours de boucle.

Par contre, un travail ciblé sur la fonction « range(…) » s'impose pour pouvoir utiliser les valeurs du compteur dans les instructions contenues dans la boucle. Si l'on souhaite par exemple calculer la somme des 100 premiers entiers naturels non nuls (*cf.* figure 17), il faudra définir le « range(…) » en tenant compte des nombres de départ et d'arrivée.

```
1  s=0
2  for i in range(1,101):
3      s=s+i
4  print(s)
```

***Figure 17** : Boucle utilisant la fonction range(...)*
avec utilisation de la variable dans de la boucle.

Dans le langage algorithmique, on écrirait : « POUR i allant de 1 à 100 ». Certains manuels [4] [6] proposent un travail ciblé sur cette fonction « range(…) » qu'il ne faut pas éluder.

On peut une nouvelle fois regretter l'absence de la notion de listes au programme de seconde, notion qui pourrait éclairer le fonctionnement d'une boucle bornée dans le langage *Python* et les paramètres de la fonction « range(…) ». Les deux instructions de la figure 18, par exemple, sont équivalentes.

```
for i in range(1,10,2):  for i in [1,3,5,7,9]:
```

***Figure 18** : Deux écritures différentes d'une boucle*
dont l'indice va de 1 à 9 avec un pas de 2.

Il nous semble important de travailler dans un premier temps la boucle bornée dans le cadre d'activités qui n'utilisent pas la valeur du compteur dans les instructions de la boucle. On peut ensuite envisager des activités qui utilisent astucieusement cette valeur au sein de la boucle.

2.5. La boucle non bornée

Le décalage le plus important entre les deux langages informatiques concerne les boucles non bornées. Le passage de l'instruction *Scratch* « répéter jusqu'à … » au « Tant que … Faire » du langage algorithmique puis au « while … : » est délicat.

Cette transition cumule nombre de difficultés déjà rencontrées pour l'utilisation des structures

conditionnelles et boucles bornées :

- la « pince » *Scratch* est traduite par l'indentation de certaines lignes de code ;
- les conditions d'arrêt sont des propositions informatiques car elles utilisent des variables instanciées mais se présentent comme des prédicats mathématiques ;
- ces propositions ont comme valeurs de vérité « faux » ou « pas faux », le « pas faux » ne signifiant pas nécessairement vrai en informatique (contrairement à la logique bivalente des mathématiques).

S'ajoute à ces nombreuses difficultés le changement des conditions d'arrêt lorsqu'on passe de l'instruction « répéter jusqu'à … » au « Tant que … Faire » et au « while … » de *Python*. Certains logiciels, comme *PyBlock*, traduisent la condition d'arrêt de *Scratch* « jusqu'à <proposition> » par la condition sous *Python* « while not(<proposition>) ». Le passage du « répéter jusqu'à … » au « while … » peut en effet s'effectuer dans un premier temps par la négation de la proposition utilisée dans *Scratch*. Un travail autour de la logique peut ensuite être engagé en traduisant les négations de propositions contenant des relations comme « = », « ⩽ », « < », etc. On peut saisir ici l'opportunité qui nous est offerte de mener, en amont, un travail un peu plus large autour de quelques éléments de la logique (proposition, prédicat, valeur de vérité, quantificateurs, négation et connecteurs logiques) : ce travail s'appliquerait alors à la programmation et pourrait trouver un écho utile dans d'autres champs du cours de mathématiques.

Les élèves doivent également changer de point de vue sur la condition utilisée dans une boucle non bornée lorsqu'on passe d'un langage à l'autre. En effet, sous *Scratch*, la condition du « répéter jusqu'à … » représente un point d'arrêt, une borne : on sort de la boucle dès que la condition est vraie pour la première fois. Dans *Python*, on effectue les instructions de la boucle aussi longtemps que la condition reste vraie : c'est un ensemble de propositions vraies qui est concerné. L'élève doit ainsi passer d'une vision ponctuelle à une vision élargie d'« intervalle ». On peut rapprocher cela du passage de l'équation du premier degré avec unique solution étudiée au collège à l'inéquation du premier degré qui admet un intervalle comme ensemble de solutions abordée en seconde.

À ce propos, pour *Scratch*, l'expression « boucle non bornée » peut prêter à confusion pour les élèves : la condition utilisée constitue pour eux une borne. Il paraît utile de préciser la définition d'une boucle non bornée : on ne connaît pas *a priori* le nombre de tours de boucle à effectuer.

Nous allons maintenant revenir sur une autre différence significative entre logiques mathématique et informatique, différences déjà abordées à propos des structures conditionnelles. Par exemple, avec le connecteur logique « ou » : si p et q sont deux propositions, $p \vee q$ et $q \vee p$ sont deux propositions équivalentes en mathématiques. Ce n'est pas le cas en informatique comme nous pouvons le vérifier avec les deux programmes de la figure 19 :

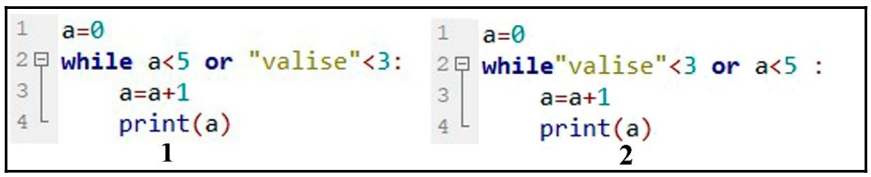


Figure 19 *: Comportements différents de scripts* Python *lorsqu'on inverse les deux propositions de la condition.*

L'exécution du programme 1 retourne les nombres 1, 2, 3, 4 et 5 avant d'afficher un message

d'erreur. Avec le programme 2, un message d'erreur apparaît immédiatement. En réalité, lors du traitement informatique de la condition $p \vee q$, si la proposition p est vérifiée, alors la proposition q est ignorée et la boucle est exécutée. Tant que p est vraie (n'est pas fausse), on effectue des tours de boucle. Lorsque la proposition p devient fausse, la valeur de vérité de la proposition q est prise en compte, ce qui déclenche ici un message d'erreur : on ne peut pas comparer une chaîne de caractères à un nombre.

Concernant ce message d'erreur, une différence de fonctionnement existe entre *Scratch* et *Python*, différence due au typage des objets utilisés dans le langage *Python*. Les relations d'ordre ($<$, $>$, $\leqslant$, $\geqslant$) ne peuvent être utilisés qu'entre objets de même type (ou presque). On peut comparer des chaînes de caractères uniquement entre elles, des listes uniquement entre elles mais des entiers, des flottants et des booléens entre eux bien qu'ils ne soient pas du même type. Par contre, la relation d'égalité peut être utilisée quel que soit le type des objets comparés.

2.6. La fonction informatique

La notion de fonction informatique est au programme de la classe de seconde. Certains manuels [2][6][7] ne l'abordent que très peu et proposent des programmes utilisant des entrées-sorties avec « input(…) » et « print(…) ». D'autres [3][5][8], au contraire, utilisent des fonctions informatiques dans tous les programmes qu'ils présentent. Nous proposons d'adopter une position intermédiaire en commençant par s'appuyer sur le « demander … et attendre » et le « dire … » pour introduire les entrées-sorties et le typage sous *Python*.

Sur le plan pédagogique, travailler la notion de fonction informatique permet de « revisiter » la notion de fonction mathématique : en effet, il convient de les distinguer autant par rapport à la notion de variable que par rapport aux représentations de la fonction en mathématiques. En informatique, on se situe davantage sur des variables instanciées contenant une unique valeur alors qu'en mathématiques, on se place sur un ensemble de valeurs prises par la variable de la fonction. L'élève doit passer d'un point de vue ponctuel en informatique à un point de vue global en mathématiques : la fonction n'est pas seulement une question d'images et d'antécédents. L'idée de « revisiter » ne doit pas conduire à tenter de faire coïncider les deux notions mais au contraire de pointer leurs différences.

Nous devons nous interroger sur l'intérêt de créer et d'utiliser des fonctions informatiques en programmation. Le premier avantage est de décomposer un programme en sous-programmes, facilitant ainsi la compréhension globale du programme. Le second avantage consiste à pouvoir appeler plusieurs fois cette fonction sans relancer l'exécution du programme ou l'utiliser à plusieurs reprises dans un même programme. Les fonctions informatiques pourront être définies en effectuant un parallèle avec les blocs de *Scratch*. Ce langage utilise des blocs avec aucun, un ou plusieurs paramètres en entrée et servent principalement pour le dessin en géométrie : en effet, on peut ainsi reproduire une même figure pour réaliser des frises, des pavages ou bien utiliser des transformations pour reproduire d'autres figures. Ces blocs n'ont pas de paramètres de sortie bien qu'ils puissent afficher quelque chose (« dire », dessin). Les paramètres en entrée sont des variables locales au bloc et privées (le bloc est lui-même privé, propre au lutin). Si on a besoin d'une variable en sortie dans le bloc, elle doit être déclarée comme une variable globale (privée ou publique).

Sous *Python*, c'est un peu différent : on a bien zéro, un ou plusieurs paramètres en entrée et toute variable utilisée au sein de la fonction est *a priori* locale. Par contre, la sortie s'effectue à l'aide de l'instruction « return » qui permet de récupérer une et une seule valeur en sortie de n'importe quel type : l'exécution de l'instruction « return » arrête la fonction et le langage prend le pas sur

l'algorithme. La valeur de sortie peut être un tuple ou une liste mais ces notions ne figurent pas au programme de seconde.

Un ensemble de fonctions peut être regroupé dans un module *Python* (fichier .py) pour être importé dans un programme principal. Une bibliothèque est une collection de modules. Le langage *Python* se caractérise par la nécessité d'importer bibliothèques ou modules existants pour avoir accès à certaines instructions prédéfinies (« sqrt(…) », « pi », « exp(…) » du module « math », « randint(…) » du module « random », instructions pour dessiner du module « turtle », etc.). Contrairement à *Scratch*, toutes les instructions ne sont pas disponibles par défaut et il convient de penser à importer les modules nécessaires à la programmation. Ceci peut constituer une difficulté supplémentaire qu'il ne faut pas négliger notamment lorsque l'on débute *Python* ou que l'on souhaite effectuer une transition avec *Scratch*. Il convient alors d'introduire sans excès et très progressivement les différents modules nécessaires pour utiliser certaines fonctions prédéfinies de *Python* lors d'activités complémentaires[7].

3. Une sélection de thèmes mathématiques susceptibles de favoriser la transition de *Scratch* à *Python*

Nous arrivons maintenant à la question du choix des thèmes pouvant servir de supports pour effectuer cette transition entre *Scratch* et *Python* dans le cadre du cours de mathématiques. Nous faisons une deuxième hypothèse : certains thèmes seraient plus favorables que d'autres pour faciliter la transition entre les deux langages.

3.1. Des thèmes mathématiques qui ne nous semblent pas adaptés

Comme nous l'avons dit dans l'introduction de cet article, le support privilégié au collège par les instructions officielles pour programmer en langage *Scratch* est le jeu. Au lycée, le lien entre mathématiques et programmation devient plus prégnant, disqualifiant le jeu comme thème de transition.

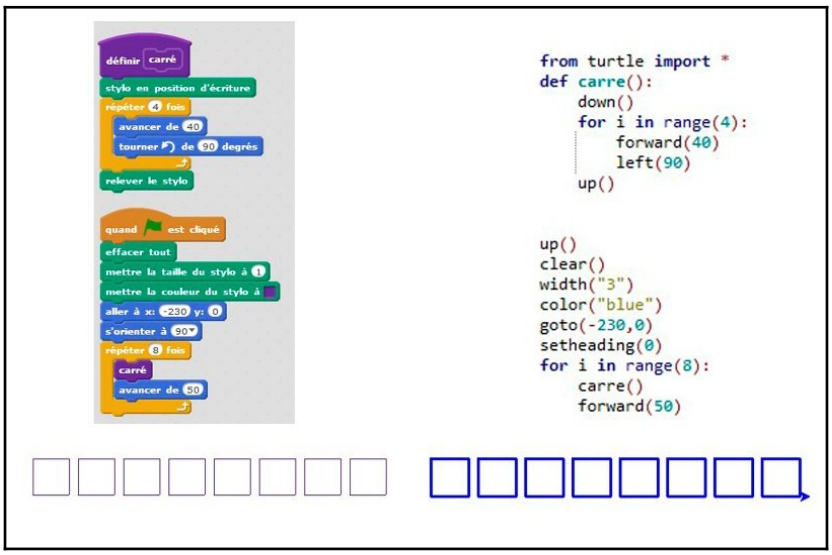


Figure 20 *: Reprise en* Python *d'un script* Scratch *qui dessine une frise avec répétition d'un motif, situation classique du collège.*

[7] https://irem.univ-rouen.fr/ressources-de-la-transition-scratch-python

Dans les sujets de mathématiques de DNB de ces dernières années, deux thèmes reviennent assez fréquemment pour servir de support aux exercices sur *Scratch* : la géométrie, où sont étudiées constructions et transformations, et les programmes de calculs.

Bien que dessiner ne soit pas un objectif du programme de seconde, la similitude apparente des instructions peut laisser penser qu'une transition serait ici possible comme dans l'exemple de la figure 20, qui reprend une situation classique au collège.Les ressemblances sont importantes mais les différences tout autant. Nous trouvons, dans le programme *Python*, une importation de module, une fonction qui ne fonctionne pas toujours comme un bloc, une boucle bornée dont nous avons parlé un peu avant dans cet article.

Initier à la programmation *Python* par le biais du module « turtle » peut être une piste lorsque l'on fait abstraction de toutes connaissances antérieures. Mais amorcer la transition entre *Scratch* et *Python* sur ce genre de programme parait ambitieux et de nombreux points de vigilance se cachent derrière l'apparente simplicité de la situation.

Le logiciel *Scratch* se prête particulièrement à la programmation d'animations visuelles avec la présence d'objets tels que les lutins ou les arrière-plans, les capteurs, les instructions de mouvements. De plus, les effets de l'ajout ou de la modification de certaines instructions peuvent être directement perçus par l'utilisateur grâce à la fenêtre de prévisualisation.

Le langage *Python*, quant à lui, est beaucoup moins adapté à ce type de tâches. Même s'il possède un module tel que « turtle » pour effectuer des dessins, il semble plus difficile à utiliser pour réaliser ce genre d'activité. De plus, la maîtrise de la langue anglaise peut constituer un frein et la terminologie de *Scratch* en langue anglaise n'est pas identique à celle de *Python*.

Nous pouvons également ajouter que la géométrie du programme de seconde est principalement analytique et nous pouvons nous interroger sur la pertinence de l'usage de *Python* en tant qu'outil de dessin.

En revanche, les programmes de calcul nous semblent être porteurs pour cette transition au même titre que l'arithmétique, les statistiques et probabilités. Nous allons détailler ces différents thèmes.

3.2. L'arithmétique

En classe de troisième, les notions étudiées (multiples, diviseurs, nombres premiers, décomposition en produit de facteurs premiers) amènent naturellement à travailler sur quelques algorithmes simples. Une grande partie de ces notions est reprise dans le programme de mathématiques de seconde et s'accompagne de propositions d'algorithmes à étudier : *« déterminer si un entier naturel a est multiple d'un entier naturel b ; pour des entiers a et b donnés, déterminer le plus grand multiple de a inférieur ou égal à b ; déterminer si un entier naturel est premier »* (MEN, 2019).

Il est tout à fait possible de faire écrire aux élèves, dès la classe de troisième, un programme permettant d'afficher la liste des diviseurs d'un nombre ou de créer un test de primalité. Cependant, au collège, l'arithmétique ne sert qu'assez rarement de support pour programmer avec *Scratch*. En seconde, l'arithmétique occupe une place réduite dans le programme de mathématiques mais offre tout de même des possibilités de produire des scripts et des algorithmes qui peuvent servir de supports pour cette transition.

Dans les différentes situations proposées ci-dessous, l'élève doit d'abord passer du langage

Scratch au langage algorithmique. Puis il découvre le langage *Python* en traduisant l'algorithme obtenu. Le choix des scripts proposés est davantage tourné vers une diversité des exemples dans le but de découvrir le langage *Python* plutôt que d'un souci d'efficacité informatique : certaines variables qui peuvent paraître inutiles servent cependant à la compréhension mathématique.

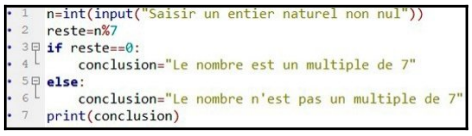
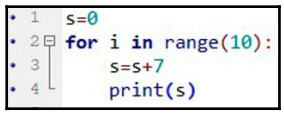
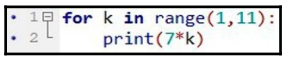
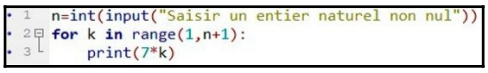
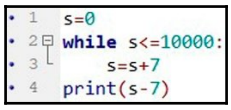
Description	**Objectifs**	**Programme final en *Python***
Entrer un nombre entier et afficher le reste de la division euclidienne par 7.	• Introduction du langage algorithmique. • Gestion des entrées-sorties. • Affectation de variables.	
Tester si un nombre est multiple de 7.	• Structure conditionnelle. • Typage des variables.	
Faire la liste des 10 premiers multiples non nuls de 7 sous forme de sommes successives de 7.	• Boucle bornée sans utilisation du compteur dans la boucle. • Introduction de la fonction « range() ».	
Faire la liste des 10 premiers multiples non nuls de 7 sous forme de produits par 7.	• Boucle bornée avec utilisation du compteur. • Fonction « range() » avec deux paramètres.	
Faire la liste des *n* premiers multiples de 7 non nuls.	• Boucle bornée avec compteur. • Fonction « range() » avec deux paramètres dont l'un est une variable.	
Déterminer le plus grand multiple de 7 inférieur ou égal à 10 000.	• Boucle non bornée avec passage de l'instruction « répéter jusqu'à » à l'instruction *Python* « while ».	

***Tableau 1** : Activités en* Python *sur le thème de l'arithmétique.*

3.3. Les programmes de calcul

Un deuxième thème, celui des fonctions numériques, fait écho au travail mené au collège sur les programmes de calcul. Il permet d'introduire les différentes fonctionnalités de *Python* en privilégiant les programmes avec entrées/sorties afin de calculer des quantités. Nous réservons l'utilisation des fonctions informatiques pour des activités complémentaires (voir les énoncés sur le site de l'IREM de Rouen). Comme pour le thème précédent, les programmes proposés contiennent des variables qui peuvent paraître inutiles mais qui sont présentes pour faciliter l'apprentissage et donner un sens mathématique.

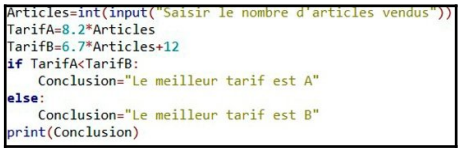
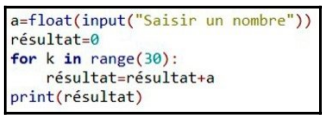
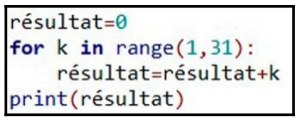
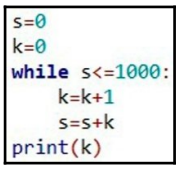
Description	Objectifs	Programme final en *Python*
Calculer le périmètre et l'aire de rectangles de largeur donnée.	• Gestion des entrées-sorties. • Affectation de variables. • Typage de variables (flottant, chaîne de caractères).	
Comparer deux tarifs dans une situation concrète.	• Typage de variables (entier, flottant, chaîne de caractères). • Structure conditionnelle et test booléen. • Variable chaîne de caractères.	
Calculer une somme avec itération d'un nombre à saisir.	• Boucle bornée sans utilisation du compteur. • Introduction de la fonction « range() ».	
Calculer la somme des 30 premiers entiers non nuls.	• Boucle bornée avec utilisation du compteur. • Fonction « range() » avec deux paramètres.	
Déterminer la valeur minimale qui permet de dépasser 1 000 dans une somme d'entiers consécutifs.	• Boucle non bornée.	

***Tableau 2** : Activités en* Python *sur le thème des programmes de calcul.*

3.4. Statistiques et probabilités

Un troisième thème, les probabilités et statistiques, permet de se placer naturellement dans le prolongement du travail débuté en collège sur les simulations d'expériences aléatoires. Ce thème permet, sur des situations assez simples, de réinvestir les instructions principales de *Python* et permet également d'aborder l'importation de modules avec l'utilisation du module « random » afin de pouvoir utiliser une instruction telle que « randint() » dont l'équivalent sous *Scratch* « nombre aléatoire entre … et … » est souvent utilisé en collège.

Au-delà de l'introduction du module « random », ces activités utilisent un ensemble important de lignes de codes pour réaliser le tirage des deux dés et calculer la somme, qui peut constituer un obstacle pour parvenir à réussir une transition douce de *Scratch* à *Python*. L'utilisation d'une fonction serait sans doute ici préférable mais là encore, cela pourrait rendre plus complexe cette transition qui est en cours de réalisation. On utilise aussi une boucle bornée qui contient une structure conditionnelle : cette imbrication introduit une double indentation très délicate à ce stade de l'apprentissage.

Ce dernier thème ne semble donc pas être le plus adapté pour introduire le langage *Python* en classe de seconde. Cependant, les activités proposées ici permettent de réinvestir un peu plus tard dans l'année les connaissances informatiques travaillées dans d'autres thèmes. Ainsi, l'algorithmique et la programmation se mettent au service des mathématiques en renforçant l'approche statistique des probabilités : l'exécution de programmes de calcul de fréquences permet d'effectuer rapidement un très grand nombre d'expériences simulées informatiquement.

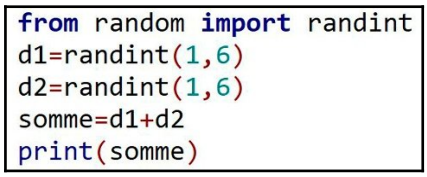
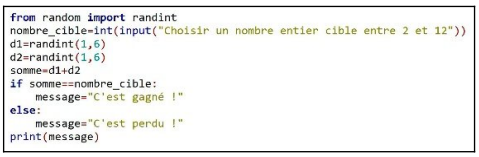
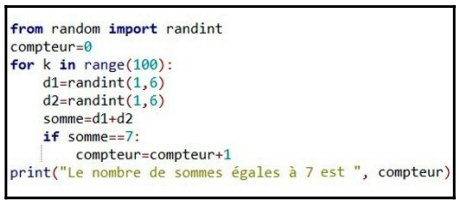
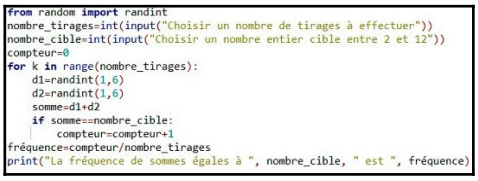
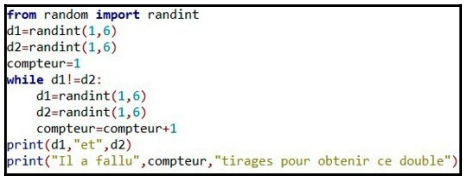
Description	Objectifs	Programme final en *Python*
Simuler l'expérience du lancer de deux dés pour calculer la somme obtenue.	• Importation d'une bibliothèque. • Utilisation de la fonction « randint() ». • Affectation de variables. • Gestion d'une sortie.	
Sous forme de jeu, proposer un nombre cible à obtenir en un lancer de deux dés.	• Gestion des entrées/sorties. • Typage de variables (entier, chaîne de caractères). • Structure conditionnelle avec « si … alors … sinon … ».	
Déterminer l'effectif de sommes égales à 7 sur 100 tirages.	• Boucle bornée sans utilisation du compteur. • Introduction de la fonction « range() ». • Structure conditionnelle simple imbriquée dans une boucle bornée.	
Calculer la fréquence de sortie d'une somme choisie de deux dés sur un nombre de tirages à saisir.	• Boucle bornée avec un nombre variable de tours de boucle. • Gestion de deux entrées et une sortie.	
Déterminer un nombre de lancers nécessaires pour obtenir un double.	• Boucle non bornée.	

Tableau 3 *: Activités en* Python *sur le thème des statistiques.*

4. Expérimentations

Les activités que nous avons créées ont été expérimentées en partie en classe de seconde durant les trois dernières années, soit durant un enseignement de SNT (2020) soit durant l'enseignement de mathématiques (2021 et 2022). En classe de mathématiques, la première expérimentation de 2021 s'est portée sur le thème des programmes de calcul et s'est déroulée suivant deux temporalités : d'une part le passage de *Scratch* à l'algorithme et d'autre part le passage de l'algorithme à *Python*. Il est apparu que les élèves, lors du second temps, ne donnaient pas de sens à l'algorithme à traduire en *Python* parce que l'algorithme se trouvait déconnecté du contexte initial. Dans l'expérimentation de 2022, nous avons souhaité rapprocher les deux passages, de *Scratch* à l'algorithme et de l'algorithme à *Python*, pour éviter cet écueil et conserver le statut de simple outil à l'écriture algorithmique pour la transition. Cette expérimentation s'est faite sur le thème de l'arithmétique. Nous pouvons constater que globalement le premier thème (programmes de calcul) abordé était davantage porteur de sens pour les élèves parce qu'il faisait référence à des situations concrètes et à des variables aux noms évocateurs (Longueur, Périmètre, Aire, Tarif A, Tarif B, Articles, Résultat). Cependant, le fait de

déconnecter les deux phases de la transition a constitué un obstacle qui est venu s'ajouter aux difficultés organisationnelles. Le travail en groupe à effectif réduit en salle informatique n'a eu lieu qu'une heure par quinzaine (maximum de 15 heures sur l'année scolaire, incluant trois séances de travail sur des éléments de logique). Les contraintes matérielles ne sont pas négligeables et constituent aussi une difficulté qui monopolise l'attention de l'enseignant et des élèves. Enfin, contrairement à ce qui pourrait être attendu, la motivation des élèves pour la programmation n'est pas constante ni uniforme. Bien que le thème arithmétique choisi cette année (2022-2023) pour aborder la transition soit moins porteur de sens, le fait de rapprocher les deux phases a permis une meilleure approche de la programmation avec *Python*. Du fait de toutes ces difficultés, l'expérimentation de la transition sur ces deux années n'a pas pu porter sur les boucles non bornées.

En SNT, l'expérimentation a porté principalement sur le thème des probabilités avec des activités conçues en trois temps : une première partie sur *Scratch*, un deuxième temps avec un travail sur l'algorithme et une troisième partie de prise en main de *Python*. Nous avons pu observer une découverte assez fluide du langage *Python* à travers la progressivité des activités. Les élèves ont pu assez rapidement écrire directement des programmes en *Python*.

Au cours de ces trois années, nous avons modifié les différents protocoles et les différentes activités. Nous avons choisi de produire des énoncés formulés de manière semblable sur le même mode à travers ces trois thèmes. Vous retrouverez les énoncés complets et leurs modes de gestion, pour les trois thèmes, sur le site de l'IREM de Rouen, dans les pages de notre groupe « Images mentales et TICE ».

Il est tout à fait possible qu'un enseignant choisisse de réaliser une transition entre *Scratch* et *Python* en proposant une progression d'activités prises non pas dans l'un de ces thèmes mais dans deux voire trois des thèmes proposés : il faudrait adapter le passage de l'un à l'autre des thèmes en reprenant les activités utiles à la prise de sens.

Enfin, ces trois thèmes sont porteurs d'activités complémentaires qui doivent permettre de gérer l'hétérogénéité dans les classes, d'aider les élèves à consolider les acquis réalisés pendant cette transition et à développer des stratégies de programmation, entre autres à l'aide de fonctions informatiques. Ainsi, des travaux autour des simulations d'expériences aléatoires, des calculs de fréquences ou de moyennes peuvent leur être proposés, ou bien des activités autour des nombres premiers, « amis » ou « parfaits » en arithmétique. Nous vous renvoyons à la page de notre groupe du site de l'IREM de Rouen à ce sujet. Nous avons expérimenté ces activités au cours des deux années. Elles ont été réalisées après avoir mis en œuvre une transition de *Scratch* à *Python* : elles ne constituent pas des éléments de cette transition mais permettent de réinvestir des lignes de code en *Python* déjà rencontrées. Elles sont aussi l'occasion de varier les thèmes abordés pour la programmation.

Conclusion et perspectives

Pour conclure, la transition entre *Scratch* et *Python* au sein du cours de mathématiques est un passage délicat en classe de seconde qui mérite une attention particulière. Il est indispensable de garder en tête que les élèves ont construit, au cours de leurs années au collège, des habitudes et des compétences avec le langage *Scratch* et que celles-ci peuvent interférer avec les nouveaux apprentissages en langage *Python*. Afin d'accompagner au mieux les élèves, l'enseignant doit rester vigilant sur les quelques points détaillés tout au long de cet article : le typage des variables, le traitement des boucles bornée et non bornée, la fonction informatique. En nous appuyant sur

nos expérimentations en classe de seconde, nous considérons qu'un passage par un travail fin sur l'algorithmique et un choix de thèmes adaptés en seconde peuvent être un appui pour favoriser le passage d'un langage à l'autre.

Cependant, il nous semble primordial que cette transition soit anticipée au collège et pour cela plusieurs pistes nous semblent possibles.

Tout d'abord sur le choix des thèmes : bien que les programmes de collège insistent sur la notion de jeu, et que cet aspect ludique soit un véritable appui pour motiver les élèves à entrer dans la programmation, cela n'exclut pas d'effectuer un glissement progressif en classes de 4e et 3e vers des supports de plus en plus mathématiques tels que ceux proposés dans les épreuves de DNB. La géométrie, les programmes de calcul mais aussi l'arithmétique et les probabilités sont des thèmes qui se prêtent particulièrement bien au travail sur la programmation et l'algorithmique comme nous l'avons vu dans cet article.

D'autre part, certaines différences entre les deux langages peuvent être introduites dès le collège. Par exemple, lorsque les élèves créent un bloc utilisant un paramètre, ils doivent choisir entre « entrée nombre », « chaîne de caractères » et « entrée booléenne » : l'enseignant a ainsi l'opportunité d'aborder le typage de variables.

Un autre exemple de préparation à cette transition pourrait consister en un travail mené autour des tables de variables en appui sur les programmes de calcul et le calcul littéral. Des activités sont proposées sur le site de l'IREM de Rouen dans le parcours[8] sur la variable informatique, ainsi que quelques autres situations pour cette transition.

Ensuite, l'initiation à l'écriture d'algorithmes peut être amorcée dès le collège, que ce soit en langage naturel ou sous forme de diagrammes ou encore dans d'autres domaines que l'informatique : algorithme d'une recette de cuisine, algorithme de construction en technologie par exemple. Cela permet à l'élève de comprendre que le programme *Scratch* est une traduction dans un langage choisi d'un algorithme donné et il peut être intéressant à ce moment de faire découvrir l'existence d'autres langages informatiques sans entrer davantage dans les détails.

Pour finir, concernant les programmes de mathématiques de lycée, il nous paraîtrait plus judicieux d'introduire les listes en seconde (souvent abordées au collège) et de reporter le travail de la boucle non bornée ainsi que peut-être celui autour des fonctions informatiques en première afin de laisser du temps pour asseoir les bases du langage *Python*.

Références bibliographiques

Branthôme, M. (2021). Apprentissage de la programmation informatique à la transition collège-lycée. *STICEF (Sciences et Technologies de l'Information et de la Communication pour l'Éducation et la Formation), 28*(3), 1-35. ATIEF, 2021.

Briant, N. (2013). *Étude didactique de la reprise de l'algèbre par l'introduction de l'algorithmique au niveau de la classe de seconde du lycée français*. [Thèse de doctorat, Université Montpellier 2].

Chevallier M., Paisnel C. & De Séegner J.-L. (2018), Le logiciel Scratch au collège : un mariage de raison entre mathématiques et informatique. *Repères IREM, 110*, 5-20.

[8] https://irem.univ-rouen.fr/parcours-7

Couderette, M. (2016). Enseignement de l'algorithmique en classe de seconde : une introduction curriculaire problématique. *Annales de didactique et de sciences cognitives, 21*, 267-296.

Davion, G. (2019). *Python : une deuxième opportunité pour les élèves de comprendre la notion de variable*. [Mémoire de Master MEEF, ESPE Paris].

Duval, R. (1993). Registres de représentation sémiotique et fonctionnement cognitif de la pensée. *Annales de didactique et de sciences cognitives, 5*, 37-65.

Knuth, D. (1968). *The Art of Computer Programming. Vol 1: Fundamental Algorithms* (2e éd.).

Lagrange, J.-B. & Rogalsky, J. (2017). Savoirs, concepts et situations dans les premiers apprentissages en programmation et en algorithmique. *Annales de Didactiques et de Sciences Cognitives, 22*, 119-158. IREM de Paris. https://doi.org/10.4000/adsc.723

Modeste, S. (2012). *Enseigner l'algorithme pour quoi ? Quelles nouvelles questions pour les mathématiques ? Quels apports pour l'apprentissage de la preuve ?* [Thèse de doctorat, Université de Grenoble].

MEN (2009). *Programme d'enseignement de mathématiques de la classe de seconde générale et technologique*. https://cache.media.education.gouv.fr/file/30/52/3/programme_mathematiques_seconde_65523.pdf

MEN (2015). *Programmes d'enseignement du cycle des apprentissages fondamentaux (cycle 2), du cycle de consolidation (cycle 3) et du cycle des approfondissements (cycle 4)*. https://eduscol.education.fr/sti/sites/eduscol.education.fr.sti/files/textes/formations-college-transversal/7529-programme-26-novembre-2015.pdf

MEN (2018). *Programmes d'enseignement du cycle des approfondissements (cycle 4)*. https://cache.media.education.gouv.fr/file/30/62/8/ensel169_annexe3_985628.pdf

MEN (2019). Programme de mathématiques de seconde générale et technologique. https://cache.media.education.gouv.fr/file/SP1-MEN-22-1-2019/95/7/spe631_annexe_1062957.pdf

Annexe
Références des manuels scolaires

[1] : *Manuel de mathématiques de seconde, programme 2019*. Le livre scolaire

[2] : *Manuel de mathématiques de seconde Hyperbole, programme 2019*. Éditions Nathan

[3] : *Manuel de mathématiques de seconde Transmath, programme 2019*. Éditions Nathan

[4] : *Manuel de mathématiques de seconde, programme 2019*. Éditions Magnard

[5] : *Manuel de mathématiques de seconde Collection Math'x, programme 2019*. Éditions Didier

[6] : *Manuel de mathématiques de seconde Variations, programme 2019*. Éditions Hatier

[7] : *Manuel de mathématiques de seconde Déclic, programme 2019*. Éditions Hachette Éducation

[8] : *Manuel de mathématiques de seconde Collection Barbazo, programme 2019*. Éditions Hachette Éducation

[9] : *Manuel de mathématiques de seconde Collection Indice, programme 2019*. Éditions Bordas