Des **outils ludiques** pour tous les âges

Lorsque l'on pense à la programmation on se dit que son apprentissage va être très compliqué et semé d'embûches. Grâce aux outils existants, pour les enfants et adolescents, la programmation s'avère être un jeu d'enfant !

• Nicolas Decoster
Co-fondateur de **La Compagnie du Code** *et animateur*
Informaticien scientifique chez Magellium
@nnodot

Youmna Ovazza
Fondatrice,
Teen-Code
(Teen-Code : stages et ateliers d'initiation à la programmation pour ados
www.teen-code.com*)*

Aurélie Vache
Lead Developer chez
atchikservices
Duchess France Leader
@aurelievache

PROGRAMMATION PAR BLOCS

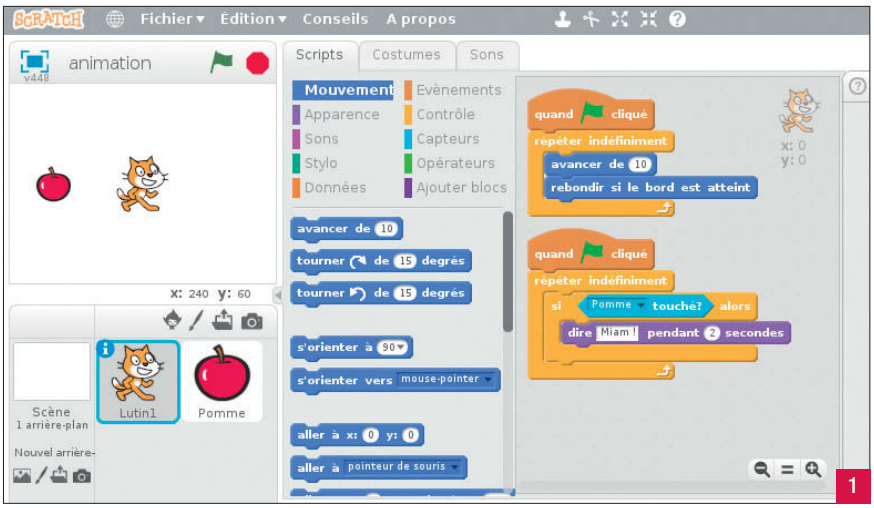
Lorsqu'on parle d'initiation à la programmation pour les enfants il est difficile de ne pas parler de Scratch. Il s'agit d'un outil de programmation visuelle par blocs qui a plus de dix ans. Il est développé par le MIT et remporte un très grand succès, auprès des petits et des grands, surtout depuis la version 2 qui est disponible en ligne sur un site communautaire qui promeut l'échange, le partage et le remix de projets (http://scratch.mit.edu). Sa syntaxe visuelle qui emboîte des blocs de textes comme des lego a inspiré de très nombreux outils similaires dont Snap! (un logiciel libre en ligne développé par Berkeley, une autre université aux USA) et Blockly (développé par Google). D'ailleurs le MIT et Google ont décidé de travailler ensemble pour réécrire Scratch, heuuu… from scratch ;-) entièrement en HTML et en JavaScript (alors que Scratch 2 est écrit en ActionScript et nécessite donc d'avoir un plugin flash pour fonctionner). Leur idée est de partir du moteur de rendu de Blockly et d'y brancher un moteur d'exécution Scratch. Le travail est en cours, affaire à suivre !

Scratch, la référence

Mais que peut-on faire avec Scratch ? En fait, il s'agit d'un langage tellement expressif que le mieux est d'en parler à l'aide d'un exemple. La figure ci-dessous montre l'interface de Scratch avec un exemple de programme. Si Captain Obvious était là, il dirait que ce programme fait se déplacer le personnage en permanence en le faisant rebondir sur les bords et en lui faisant dire "Miam !" dès qu'il touche une pomme. Voilà, un premier programme simple à faire mais qui produit déjà une première animation. [1]

L'interface de Scratch est basique. Il y a une scène à gauche, avec la liste des lutins en dessous, au milieu il y a les blocs utilisables regroupés par catégories et la partie programmation se trouve à droite. Pour programmer on glisse tout simplement les blocs avec l'aide de la souris et on les assemble comme des Lego. Nous n'allons pas rentrer dans les détails du fonctionnement de Scratch, il y a beaucoup de ressources en ligne pour cela et même de plus en plus de livres. Indiquons juste que Scratch permet d'utiliser beaucoup de concepts de base de la programmation : les séquences d'instructions, les exécutions conditionnelles, les boucles, les variables, les événements, les interactions avec l'utilisateur, l'affichage, l'encapsulation (on peut définir de nouveaux blocs), les messages, la création dynamique d'objets. Et, au final, les notions de base essentielles qu'utilisent les développeurs sont quasiment toutes là.

Cependant lorsque l'on commence à avoir une pratique un peu poussée de Scratch on atteint certaines limites : on ne peut pas définir de blocs qui retournent quelque chose, on ne peut pas attacher des informations aux messages que l'on envoie, on ne peut pas accéder aux informations ou aux actions des autres lutins, etc. Ces limitations qui sont réellement handicapantes lorsque l'on veut réaliser un projet un peu complexe, sont pleinement assumées par l'équipe de développement de Scratch : la volonté est de garder quelque chose de simple pour que cela reste un outil très accessible dédié à la découverte de la programmation, tout ce qui est un peu complexe à appréhender a été volontairement laissé de côté.



1

Snap! pour aller plus loin

Snap! reprend les principes et les fonctions de Scratch, et les deux outils sont très similaire d'aspect. Par contre Snap! offre plein de nouvelles possibilités qui pallient des manques, qui facilitent la vie du développeur, qui lui permettent d'explorer des concepts plus avancés ou qui lui ouvrent de nouvelles possibilités.

Au final, Scratch et Snap! sont tellement simples et amusants à utiliser que l'on pourrait penser qu'ils sont de simples jouets qui ne font qu'effleurer ce qu'est réellement la programmation. Mais cela est trompeur, au-delà de la simple découverte du code, il y a moyen de construire des choses ambitieuses et on peut aller très loin dans l'exploration de la programmation!

App Inventor

Curieusement méconnu en France, App Inventor (AI) est certainement une des meilleures

manières de s'initier à la programmation mobile, et à la programmation tout court, pour débutants. Conçu en 2009 par une équipe conjointe de Google et du MIT (Massachussets Institute of Technology) et depuis développé par ce dernier, App Inventor est un logiciel de programmation visuelle sous Android, (comme Scratch par exemple pour les enfants), basé quant à lui sur Blockly, open-source et accessible gratuitement en ligne en plusieurs langues.

Quelques chiffres clés pour en présenter l'impact international avant de détailler ses avantages : en 2015, la communauté d'utilisateurs de AI 3 était de millions d'utilisateurs issus de 195 pays différents. Plus de 100 000 utilisateurs actifs / semaine, ont construit à ce jour plus de 7 millions d'applis Android.

App Inventor, pour qui et pour quoi faire ?

Grâce à sa double interface de création, d'interface utilisateur d'une part, et de programmation d'autre part, AI permet de s'initier, bien au-delà de la programmation "pure", à l'ensemble de la logique qui préside à la création d'un projet : l'utilisateur est au cœur du projet, le design et l'ergonomie sont en permanence aussi importants et visibles que le code lui-même, et en cela, c'est un outil excellent pour s'initier à la programmation ou sensibiliser des profils non-techniques aux différentes dimensions qui y sont associées. [2]

App Inventor est également un outil qui permet la réalisation complète d'un projet, de zéro à l'installation d'une application fonctionnelle sur son téléphone ou sa tablette, au partage avec d'autres personnes voire à la publication sur le Google Play Store. C'est donc un outil qui n'est pas un simple bac à sable mais un véritable moyen de création d'une réalisation concrète qu'on transporte et manipule avec soi, dans sa poche, ce qui situe bien les enjeux de l'apprentissage en prise directe avec les usages quotidiens des utilisateurs.

Grâce à la grande variété des composants précodés déjà installés, AI se prête à de multiples sujets d'application, du dessin aux jeux vidéo ou à la photo en passant par la géolocalisation, la traduction, la gestion des SMS, les échanges d'information via Internet, etc. ce qui en fait un cadre flexible et ouvert aux centres d'intérêt du plus grand nombre, et notamment de tous les "non-geek" qui n'en utilisent pas moins un téléphone tous les jours !

Du cadre scolaire à la formation pour adultes, AI permet d'intégrer les bases de la logique et des bonnes pratiques de développement, en réalisant de vrais projets, sans y passer des mois : que demander de plus pour s'y mettre ?

PROGRAMMATION EN MODE TEXTE

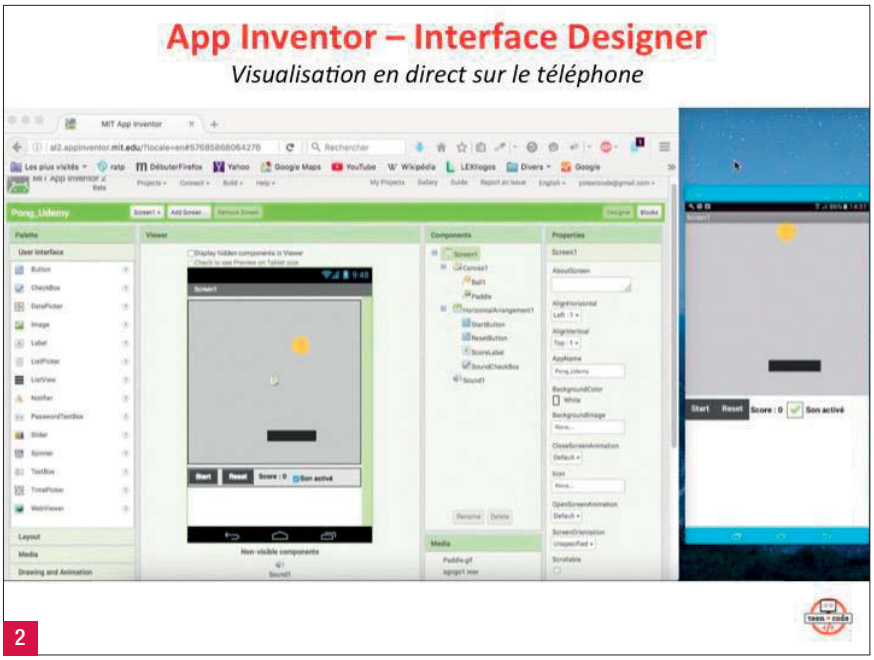
On l'a vu, les outils de programmation par bloc sont des candidats sérieux pour découvrir et approfondir la programmation et ses concepts. Cependant, actuellement le métier de développeur est essentiellement centré sur les langages de programmation en mode texte. De nos jours les logiciels, les applis, les sites Web font appel à des programmes écrits en C, C++, Java, JavaScript, Python, C#, bash, etc. L'attrait pour ces "vrais" langages de programmation par les enfants et les adolescents est donc tout naturel, ils ont envie d'entrer dans la cour des grands et de pratiquer les langages qui ont construit les logiciels qu'ils utilisent. Mais quels sont les langages textuels qui sont adaptés pour eux dans l'optique d'apprendre la programmation ? Et quels sont les avantages, les inconvénients et les spécificités par rapport à la programmation par blocs ?

En fait chaque langage a ses particularités qui fait qu'il sera plus ou moins adapté à l'apprentissage. Certains sont plus complexe, d'autres sont moins accessibles et d'autres sont des langages obligés pour un usage particulier. Passons en revue certain d'entre eux.

Tout d'abord le langage le plus disponible : JavaScript. Ce langage est présent partout, il suffit d'avoir un navigateur Web et vous pouvez faire du JavaScript. Vous ouvrez votre éditeur de texte préféré, vous mettez quelques lignes de code dans un fichier et vous l'exécutez avec votre navigateur. De plus, la plupart des navigateurs embarquent des outils de développement intégrés (console, débugger…). C'est le standard Web reconnu et beaucoup d'efforts sont fait pour disposer de moteurs performants : Google, Mozilla, Microsoft et Apple sont tous en compétition sur ce terrain. On dispose même de moteur JavaScript en dehors des navigateurs Web (le projet node.js est un moteur en ligne de commande). Cela fait que JavaScript est un langage avec lequel il faudra compter encore longtemps et sa communauté est l'une des plus bouillonnantes du moment.

Ensuite, parlons d'un vieux langage qui est le socle de beaucoup de choses : le langage C. Beaucoup de logiciels existants sont écrit en C, comme de grands pans du système Linux par exemple. C'est également un langage de prédilection pour de l'informatique embarquée car il est bas niveau et très performant. En particulier, c'est le langage qui est généralement utilisé pour programmer les cartes Arduino, et il existe tout un outillage pour s'y mettre facilement. Un avantage du langage C, qui peut être aussi un inconvénient, est qu'il est plus bas niveau que beaucoup d'autres, en particulier sur les problématiques de gestion de mémoire : c'est instructif sur le fonctionnement interne d'un programme mais c'est plus délicat à programmer. Ce langage à l'avantage, ou l'inconvénient ;-), d'imposer au développeur de se frotter aux joies de la compilation et de son lot d'erreurs.

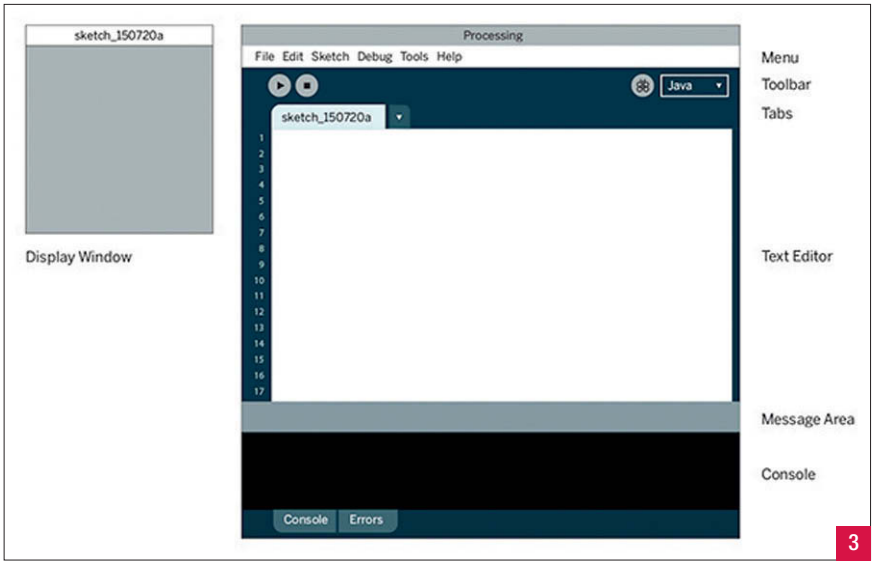
Un langage qui est un peu moins bas niveau est Python. C'est un langage interprété ce qui, en particulier, évite les problèmes de compilation. Il est très populaire et offre un écosystème très



2

riche. Quel que soit le domaine qui vous intéresse vous trouverez des modules python pour vous faciliter la vie : jeux, calcul scientifique, serveur Web, visualisation de données, communication avec du matériel, etc.

Enfin, parlons de Java, langage très utilisé en entreprise. C'est également un langage compilé (un peu comme le C mais avec quelques nuances), il est portable (comme JavaScript et Python) et dispose également d'une très grande communauté (mais peut-être avec un écosystème moins riche que pour Python). C'est également un langage très présent dans des problématiques d'actualité comme la Big Data. Java est le langage de la bibliothèque processing qui est très adapté à l'informatique créative et qui permet de découvrir la programmation de manière ludique (voir les figures ci-contre). [3]



Interface de développement de processing

```
void setup() {
  size(480, 120);
}

void draw() {
  if (mousePressed) {
    fill(0);
  } else {
    fill(255);
  }
  ellipse(mouseX, mouseY, 80, 80);
}
```

[4]

Exemple de code processing et le résultat associé

Rappelons également que le Java est utilisé pour programmer des applications mobiles sous Android.

Citons rapidement le C++ pour dire qu'il s'agit un langage très intéressant et très actif mais qui est plus difficile d'accès (on peut y venir après avoir appris le langage C), et la programmation *shell* (comme *bash*) qui est radicalement différente des autres mais qui présente un certain intérêt en tant que tel et qui est quasiment incontournable dès qu'on veut se plonger dans le fonctionnement des systèmes Unix (comme Linux). Il y a donc deux grandes grandes approches pour programmez et, donc, pour apprendre à programmer. Soit la programmation par bloc, soit la programmation en mode texte. Voyons les avantages et les inconvénients du code texte par rapport au code bloc.

Inconvénients :

- Le programmation textuelle est en général moins accessible, il faut souvent des outils à installer avant de pouvoir commencer à coder. Cependant, il y a de plus en plus de possibilités pour coder en ligne dans ces langages ;
- Souvent le résultat en mode texte est accessible moins rapidement qu'en mode bloc. Par exemple, avec Scratch ou Snap! on clique sur un bloc "avancer de 10" et le personnage avance tout de suite. Dans certains langages textuels il faut écrire le code, sauvegarder le fichier, éventuellement compiler, exécuter pour enfin voir le résultat ;
- Dans le même genre d'idée, en général il n'y a pas d'interactivité ou de *live coding*. Dans Scratch ou Snap!, pendant que le programme tourne on peut changer une valeur ou ajouter un bloc et l'exécution prend en compte la modification instantanément. Cela ouvre des possibilités d'explorations et d'expérimentations plus importantes. La plupart des langages textuels ne permettent pas cela (bien que cela commence à arriver, en particulier avec les développement Web en JavaScript) ;
- Pour développer en mode texte, même si ce n'est pas indispensable, c'est souvent mieux d'avoir un bon outillage et il peut parfois être complexe à mettre en place : il faut trouver le bon éditeur, le bon débogueur, le bon *linter*, etc. Mais on peut trouver quand même des choses bien intégrées, voire en ligne ;
- Avec un langage textuel on est confronté aux erreurs de syntaxe qui par définition ne sont pas présentes avec les blocs. Cependant un bon *linter* permet de limiter les dégâts ;
- Il en est de même avec les erreurs de compilation, c'est un frein supplémentaire, même si un bon outillage permet d'aider le développeur dans la recherche des problèmes.

Avantages :

- Comme nous l'avons dit, actuellement l'essentiel de la programmation se fait en mode texte. S'y mettre c'est se former à l'existant ;
- Un simple éditeur de texte est suffisant pour créer quelque chose, pas besoin d'une interface graphique ;
- Les langages textuels peuvent être plus puissants dans ce qu'ils permettent de faire ;
- Ils peuvent être plus bas niveau et donc offrir un contrôle plus fin de certaines choses (comme la gestion de la mémoire par exemple) ;
- Ils permettent de développer des choses complexes d'une certaine ampleur.

POUR LES PLUS PETITS

On a vu qu'il y avait des outils pour les adolescents, pour les enfants mais qu'en est-il pour les plus petits, peuvent-ils également être initiés au merveilleux monde de la programmation ?

L'initiation de la programmation n'est pas réservée qu'aux grands, les petits sont également gâtés. Avez-vous déjà entendu parler de Cubetto ou de Thymio, non ? Nous allons vous guider vers ces fabuleux outils.

Cubetto (à partir de 3 ans) [5]

Cubetto, sous ce nom se cache un mignon petit robot en forme de cube. Une campagne Kickstarter a été lancé cette année et a été plus que financée, plus de 1 596 457 $

sur les 100 000 $ initiaux ont été réunis par 6 553 contributeurs ! Lorsque l'on sait que cette campagne de crowd-founding a été soutenue