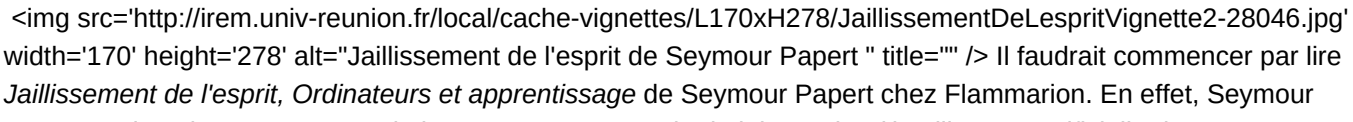
http://irem.univ-reunion.fr/spip.php?article258

Quel langage de programmation pour l'algorithmique en Seconde ?

- Algorithmique et programmation
- Activités algorithmiques en Seconde
-

Date de mise en ligne : dimanche 8 novembre 2009



Il faudrait commencer par lire *Jaillissement de l'esprit, Ordinateurs et apprentissage* de Seymour Papert chez Flammarion. En effet, Seymour Papert est l'un des concepteurs du langage LOGO au sein du laboratoire d'intelligence Artificielle du M.I.T. (Massachussetts Institute of Technology).

« Papert jette un regard sans complaisance sur la façon dont sont actuellement enseignées les mathématiques, et il démontre, en une brillante synthèse mariant les mathématiques à la théorie cognitive, comment la "phobie des mathématiques" prend naissance dans nos classes, et comment les ordinateurs pourraient en venir à bout. »

Le livre date de 1980. Le problème est que l'enseignement des mathématiques n'a pas encore vraiment changé dans nos classes ! Peut-être que l'arrivée de l'algorithmique dans les programmes officiels va enfin y changer quelque chose.

Remarquons par ailleurs que ce n'est pas vraiment l'arrivée de l'algorithmique qui marque nos nouveaux programmes, car elle était déjà présente partout dans les programmes de la rentrée 2000.

Voici quelques pistes de réflexions autour de l'algorithmique en seconde pour vous aider à choisir un langage de programmation afin d'implémenter en classe des algorithmes avec vos élèves.
Ces réflexions sont menées à l'aide de la carte mentale ci-dessous :

Vous pouvez naviguer de manière dynamique sur cette carte mentale à l'adresse :
http://nathalierun.net/PenseeLibre/MindMapping
Suivre alors le lien *Applications*, puis déplier le noeud *Choisir* pour cliquer sur le noeud *Quel langage pour l'algorithmique en seconde ?*.

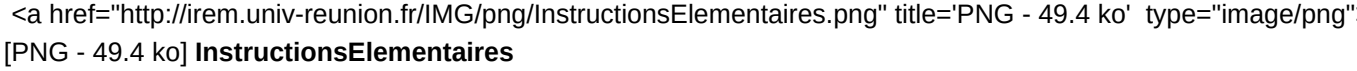
Ces quelques réflexions vont poser les questions suivantes :

1. De quoi est fait un algorithme ?
2. Quelle syntaxe pour l'algorithmique ?
3. Pourquoi implémenter un algorithme dans un langage ?
4. Quelques langages utilisables, quel choix pour un langage ?
5. Comment choisir un langage ? Quelques critères de choix.

De quoi est fait un algorithme ?

De quoi est fait un algorithme ?

Instructions élémentaires



InstructionsElementaires

[-] déclaration : faire la liste des variables
[-] affectation : initialiser et affecter les variables
[-] instructions de calcul : écrire une formule permettant un calcul
â€” Calculer l'image d'un nombre par une fonction
â€” Calcul du discriminant d'une équation du second degré
[-] instructions d'entrées-sorties : entrer les données et restituer les résultats

Instructions conditionnelles

<a href="http://irem.univ-reunion.fr/IMG/png/InstructionsConditionnelles.png" title='PNG - 36.5 ko' type="image/png">[PNG - 36.5 ko] **InstructionsConditionnelles**

[-] Si ... Alors
[-] Si ... Alors ... Sinon
â€” Calcul des solutions d'une équation du second degré
â€” Algorithme de calcul de la fonction Valeur Absolue, d'une fonction avec valeur interdite
â€” Algorithmes de logique

Calcul itératif

<a href="http://irem.univ-reunion.fr/IMG/png/CalculIteratif.png" title='PNG - 92.8 ko' type="image/png">[PNG - 92.8 ko] **CalculIteratif**

[-] Boucles avec un nombre d'itérations donné au départ
(Faire le parallèle indice/compteur). Exemples :
â€” Tableau de valeurs d'une fonction
â€” Algorithme de calcul de la fonction Puissance
â€” Construction de Polygones réguliers : triangle équilatéral, hexagone régulier, polygône régulier à n côtés
â€” Suites récurrentes : Calcul de u_n : Fibonacci

[-] Boucle avec fin de boucle conditionnelle
â€” la condition est donnée au départ de la boucle
â€” la condition est donnée à la fin de la boucle
â€” SIMULER des jets de dés et calcul des fréquences

[PNG - 20.9 ko] **Lancers de dés**	[PNG - 23.3 ko] **Manipulation de tableaux**

â€” PROMENADES ALEATOIRES (sauts de puce sur une droite, sur un triangle, sur un tétraèdre)
<dl class='spip_document_2512 spip_documents spip_documents_right' style='float:right;'>[PNG - 20.8 ko] **PromenadeAleatoireSurUnTetraedre**

[-] Boucle infinie (la boucle ne s'arrête que par une interruption brutale du programme par l'utilisateur)
â€” LE LIEVRE ET LA TORTUE
â€” Suite divergente

Quelle syntaxe pour l'algorithmique ?

<a href="http://irem.univ-reunion.fr/IMG/png/QuelleSyntaxePourLalgorithmique.png" title='PNG - 61.3 ko' type="image/png">[PNG - 61.3 ko] **QuelleSyntaxePourLalgorithmique**

En français (pseudo-code)

<a href="http://irem.univ-reunion.fr/IMG/png/Pseudo-Code.png" title='PNG - 74.6 ko' type="image/png">[PNG - 74.6 ko] **Pseudo Code**

[-] Affectation

```
Attribuer à toto la valeur 5
toto <-- 5
```

[-] Les instructions d'entrées-sorties

```
 Entrer A
Afficher A*A
```

[-] Instruction Conditionnelle

```
 Si ConditionVraie Alors Faire
...
Sinon Faire
...
FinSi
```

[-] Calcul itératif

- dont on connait le nombre d'itérations

```
 Pour i allant de ... à ... Faire :
...
...
FinPour
```

- associé à une condition

La condition est au début de la boucle

```
 TantQue ConditionVraie Faire :
...
...
FinTantQue
```

La condition est à la fin de la boucle

```
 Faire :
...
...
TantQue ConditionVraie
FinFaire
```

Algobox

Algobox ne me sert qu'à écrire des algorithmes simples que je construis avec les élèves. Je montre alors ces algorithmes écrits avec Algobox aux élèves sous forme d'images.
Je leur demande alors d'expliquer l'algorithme et de suivre les variables.

Le logiciel Algobox est idéal pour écrire des algorithmes.

Mais Algobox n'est pas un langage de programmation. Il a été écrit spécifiquement pour écrire des algorithmes avec les élèves. Son éditeur n'est absolument pas ergonomique et développer un programme avec est assez fastidieux.
Algobox n'est capable d'exécuter que des algorithmes simples.

Pourquoi implémenter un algorithme dans un langage ?

<a href="http://irem.univ-reunion.fr/IMG/png/PourquoiImplementerUnAlgorithmeDansUnLangage.png" title='PNG - 53.4 ko' type="image/png">[PNG - 53.4 ko] **PourquoiImplementerUnAlgorithmeDansUnLangage**

[-] **pour tester son algorithme**
[-] **pour débogger**
afin d'effectuer une gestion des erreurs. L'exécution en mode pas à pas est la première étape du déboggage.
[-] **pour mettre la machine au service de l'être humain**
[-] **pour vérifier que l'algorithme est bien UNIVERSEL**
Le traduire et l'exécuter sur ordinateur et sur calculatrice lorsque cela est possible.
[-] **parce que c'est ludique**
L'élève éprouve une réelle satisfaction de voir que son algorithme est correct puisque le programme Â« tourne Â».
[-] **pour préparer nos élèves à un environnement numérique**

Quelques langages utilisables, quel choix pour un langage ?

<a href="http://irem.univ-reunion.fr/IMG/png/QuelquesLangagesPourLalgorithmique-png800px.png" title='PNG - 143.9 ko' type="image/png">[PNG - 143.9 ko]

Langages orientés WEB

Faire programmer les élèves avec un langage orienté WEB serait idéal puisque c'est ce dont ils auront principalement besoin dans leur vie numérique d'adulte face au Net.

[-] PHP
Les variables en PHP doivent impérativement commencer par $. Exemple de code PHP :

```
<?
for($i=1;$i<=10;$i++)
{echo "$i.Mon premier script PHP!
\n";}
```

```
?>
```

PHP est interprété côté serveur et il faut donc installer un serveur Web local sur sa machine. Il est idéal pour travailler avec des bases de données (on parle du couple PHP/MySQL). Il présente l'avantage majeur de pouvoir s'inclure directement dans des pages HTML.
[-] Java
Java est un langage de programmation informatique orienté objet qui provient du C++. Sa syntaxe lui ressemble beaucoup.
Il faudrait un cours spécifique de programmation pour mettre du Java en oeuvre au lycée...
[-] Javascript

```
var n=demander("Calcul de la somme des naturels entre 0 et n. Entrez la valeur de n");
var i;
var somme=0;
for (i=1 ; i<=n ; i++)
 somme=somme+i;
afficher("La somme des entiers de 0 à ", n, " est : ", somme);
```

Emmanuel Ostenne a proposé un éditeur JavaScript en ligne à cette adresse. Rien n'empêche de prendre quelques heures de demi-groupe avec les élèves pour y faire fonctionner de petits algorithmes très simples.

D'autres langages

Etant ingénieur d'origine, j'ai touché à de nombreux langages. J'ai commencé par apprendre à programmer sur des cartes perforées à l'école, puis en Fortran 77 sur un PDP11-34 de chez Digital Equipment en arrivant dans le monde du travail pour écrire mon programme de thèse qui a servi aux règles Neige et Vent 1985 du Bâtiment. Ensuite j'ai appris le C dans un bureau d'étude pour développer de gros projets prévisionnels pour EDF. Puis, au sein de l'Education Nationale, j'ai appris le Basic et le Pascal pour écrire de petits programmes sur mon ordinateur personnel et communiquer avec les collègues.
Je me suis ensuite mise au C++ (juste pour la syntaxe, mais je n'ai pas fait de programmation orientée objet) pour donner quelques TP d'informatique à l'IUT.

De cette expérience, je classifie ainsi les langages :

Langages obsolètes
[-] Fortran 77
[-] Basic
[-] Pascal

Langages universel : C et C++, langages de noblesse
[-] C ayant été écrit pour développer le système Unix. La syntaxe du C est pure et réduite à sa plus simple expression. Les fonctions sont entre parenthèses. Beaucoup de syntaxes actuelles s'en inspirent (Python, Java, Javascript). Voir C sur Wikipedia.

```
int factorielle(int n)
{
```

```
 if (n > 1) return n * factorielle(n - 1);
 else return 1;
}
```

[-] C et C++ sont des langages dont les programmes peuvent être totalement portables, si l'on respecte les règles de portabilité (norme ISO).

```
#include <stdio.h>

int main(void)
{
 printf("hello, world\n");
 return 0;
}
```

Langages « graphiques »

[-] Logo

_ Sorti du Massachusetts Institute of Technology (MIT) dans les années 1960, il permet de déplacer la fameuse tortue qui peut alors dessiner des figures géométriques, mais il permet aussi d'écrire des programmes complets et récursifs. Logo est un langage puissant - bien plus que son contemporain Basic - qui a été (et est toujours) sous-estimé par les matheux de par son côté ludique et puéril.

Flocon de Von Koch

```
POUR CARRE :X
AVANCE :X
TD 90
AVANCE :X
TD 90
AVANCE :X
TD 90
AVANCE :X
TD 90
FIN
POUR VONKOCH :X
SI :X < 5  AVANCE :X [STOP]
VONKOCH :X / 3 TG 60
VONKOCH :X / 3 TD 120
VONKOCH :X / 3 TG 60
VONKOCH :X / 3
FIN
```

[-]

[PNG - 7.5 ko] **Logo de Scratch**

Scratch est pour moi en quelque sorte le successeur de Logo. En pleine phase de développement, il sort aussi des laboratoires du MIT, ne gère pas encore la récursivité mais permet déjà d'adapter de nombreux algorithmes niveau lycée.

Avec Scratch, on peut, comme avec Logo, dessiner très facilement une figure géométrique en indiquant une série d'instructions à affectuer à un petit lutin.

Il est possible de trouver un grand nombre d 'exemples sur un espace de partage, ce qui rend son apprentissage très aisé. On peut lire sur Wikipédia : « Le slogan de Scratch est *Â« Imagine·Programme·Partage ! Â»* . Le partage est en effet un fondamental de la pédagogie de Scratch. »

Un article donnant des exemples d'algorithmes en seconde avec Scratch et Algobox se trouve ici :
Initiation à l'algorithmique avec Scratch et Algobox.

Langage associé à de la géométrie dynamique

[-] CarMetal et son don d'ubiquité

Implémentant du Javascript avec les *CarScripts*, CarMetal permet d'allier géométrie dynamique et programmation, ce qui ouvre un champ de possibilités presque sans limite pour écrire des algorithmes au lycée.

<dl class='spip_document_2556 spip_documents spip_documents_right' style='float:right;'>[JPEG - 41.1 ko]

Cardioïde

```
a=Point("O",0,0);
b=Point("B",5,-2);
pas=40;
// pas=Input("Pas ?");
c=Circle("",a,b);
ro=Math.sqrt(X(b)*X(b)+Y(b)*Y(b)); //rayon du cercle c
//Println(ro);
m=Point("M",1,-2);
//FixedCircle("",m,ro);
for (n=1; n<=pas; n++){
 teta=4*Math.PI/n;
 x_r=ro*Math.cos(teta);
 y_r=ro*Math.sin(teta);
 r=Point("R",x_r,y_r);
 c1=Circle("",r,m);
 SetColor(c1,"red");
 }
```

Dans la rubrique « Algorithmique et CarScripts », Yves Martin et Alain Busser développent actuellement le thème de travail : « Utilisation de javascript avec l'éditeur incorporé à CaRMetal pour enseigner l'algorithmique au lycée ».

On peut s'initier aux CarScripts avec l'article de Yves Martin : « Algorithmique - Introduction aux CarScripts de CaRMetal »

Alain Busser propose les corrections de 24 épreuves pratiques au Bac S 2009 avec CarMetal dans cette rubrique : Corrigés de l'épreuve pratique du bac S 2009.

Langages scientifiques

[-] Scilab, écrit par une équipe d'ingénieurs (une version Scilab lycée a été spécifiquement développée),

Une version a été développée de manière spécifique pour le lycée : Scilab pour le lycée

```
for n=1 : 100 ;
u(n)=(1+1/n)^n ; end
n=1:100';plot2d(n,u(n),style=-1),y=%e,plot(n,y)
```

[-] bc, très puissant mais en ligne de commande,

```
define fibonacci (n) {
if (n==0) return (0);
if (n==1) return (1);
return (fibonacci(n-2)+fibonacci(n-1));
}
```

Voir bc sur Melusine.

[-] R, adapté aux statistiques,

```
fregresssion <-function()
{
par(mfrow=c(2,2))
x <-rnorm(12,6)
y <- rnorm(21,9)
plot(x,y)
abline(lm(y~x))
hist(x)
hist(y)
}
```

[-] Ocaml, utilisé en classes préparatoires

Il y a aussi
[-] Python

```
def factorielle(x):
if x == 0:
return 1
else:
return x * factorielle(x-1)
```

[-] Visual Basic
Visual Basic est un langage propriétaire créé par Microsoft et dérivé du Basic.

Ce sont à priori les plus connus. Et il y en a beaucoup d'autres bien sûr...

Comment choisir un langage ? Quelques critères de choix

<a href="http://irem.univ-reunion.fr/IMG/png/CommentChoisirUnLangage-QuelquesCriteresDeChoix.png" title='PNG - 124.9 ko' type="image/png">[PNG - 124.9 ko] **CommentChoisirUnLangage QuelquesCriteresDeChoix**

Analyse des besoins

[-] Langage de syntaxe simple, ce qui exclut PHP et Java
[-] Langage interprété
[-] Prise en main rapide
[-] Les calculs doivent être facilement mis en oeuvre
[-] Langage bien documenté avec une documentation accessible facilement
[-] Nombreux exemples en ligne [1]
[-] Interface de programmation simple
[-] Possibilité de déboggage [2]

Quelques critères

<dl class='spip_document_2538 spip_documents spip_documents_left' style='float:left;'>[JPEG - 31.3 ko] **comment choisir un langage de programmation ?**

Le livre : *Comment choisir un langage de programmation ?* de Thomas Pornin m'a aidée à choisir les critères suivants :

â€” LIBERTE du logiciel
â€” PERENNITE du langage
â€” PORTABILITE du langage
â€” DISPONIBILITE du langage
â€” ERGONOMIE de l'environnement de programmation
â€” Support
â€” Mutualisation des ressources en ligne
â€” Ludique

1. **LIBERTE du logiciel**

Un logiciel utilisé dans l'éducation Nationale doit être à mon sens libre et gratuit.
[-] gratuit pour son accessibilité aux élèves
[-] libre pour une question d'éthique
Ce point exclut Visual Basic.

2. **PERENNITE du langage**

On demande un développement en cours et une maintenance assurée. La liberté d'un logiciel augmente alors les chances de pérennité du langage.

3. **PORTABILITE du langage**

Les langages écrits en java sont par définition portables, par exemple CarMetal.

3. **DISPONIBILITE du langage**

Le langage doit posséder un ou plusieurs éditeurs accessibles facilement par téléchargement, et une interface d'exécution.

4. **ERGONOMIE de l'environnement de programmation**

Il faut au minimum un éditeur gérant la coloration syntaxique.

5. **Support**

[-] La documentation du langage doit être étoffée, avec de nombreux exemples.
[-] Existence indispensable de forums de développeurs sur lesquels on peut trouver des réponses à nos questions.

6. **Mutualisation des ressources en ligne**

On doit pouvoir trouver sur le Net un espace de partage des programmes écrits par les internautes.

7. **Ludique**

Le langage choisi et son environnement de développement doivent donner envie aux élèves de concrétiser leurs algorithmes, et même d'aller encore plus loin.
Cette approche contribuera certainement :
[-] à dédramatiser les mathématiques.
[-] faire des mathématiques autrement

Synthèse

Le langage de programmation choisi doit être adapté à nos élèves qui sont des **Digital Natives**. On pourra à ce sujet lire la page consacrée aux *Natifs Numériques* sur Wikipedia sur laquelle on trouve cette définition qui colle à merveille à nos élèves :
Un natif numérique (ou digital native en anglais) est une personne ayant grandi dans un environnement numérique comme celui des ordinateurs, Internet, les téléphones mobiles et les baladeurs MP3..
Un site en anglais leur est entièrement consacré : il est destiné à nous faire comprendre la première génération des natifs numériques.

Au sujet de ce changement des adolescents, on pourra aussi lire avec intérêt ce numéro spécial du CAFé PEDAGOGIQUE : *Enseigner avec les jeux* dans lequel on lit avec plaisir que beaucoup d'enseignants se posent la question de comment changer leur enseignement et y intégrer des notions qui parlent vraiment aux élèves puisque c'est leur domaine.
L'article sur les jeux électroniques en classe paru dans le mensuel du mois d'octobre du Café Pédagogique propose des guides sur les jeux en fonction de leur potentialité éducative.
Quelques jeux mathématiques sont proposés ici.

Il ne s'agit pas au lycée de faire des programmeurs en classe de mathématiques, mais on peut toutefois réfléchir à l'avenir de nos élèves, et leur donner une aptitude à apprendre un langage de programmation ainsi qu'une approche ludique de la programmation en choisissant un langage simple, ludique et coloré, qui leur inculque des bases solides d'algorithmique, sans négliger la phase pratique de la mise en oeuvre de ces algorithmes.
Si on dresse un tableau comparatif des divers langages énumérés dans cet article, avec les critères cités, et avec quelques tests faits en classe, **Scratch est idéal pour mettre en oeuvre de manière ludique et rapide des algorithmes en classe de seconde** :

[-] sa prise en main par les élèves est quasi-immédiate après une très courte démonstration au vidéo-projecteur
[-] l'environnement est simple et efficace : constitué de 3 parties, on a les instructions à disposition, la fenêtre du programme et la fenêtre d'exécution du programme dans la même fenêtre.
[-] il n'y a pas de syntaxe à connaître ni à écrire, on déplace simplement des lignes d'instruction qui s'imbriquent par aimantation.
[-] un simple double-clic sur une instruction permet de l'exécuter
[-] les élèves sont séduits par l'interface
[-] le déboggage y est très facile pour 2 raisons :
â€" exécution en mode pas à pas dont on peut régler la vitesse
â€" les blocs de programme peuvent être détachés les uns des autres et testés par simple double-clic dessus. On peut donc effectuer un déboggage par bloc.
[-] c'est un excellent logiciel pour s'initier à la programmation
[-] on peut faire travailler les élèves par groupes pour effectuer de petits projets de programmation comme des simulations de promenades aléatoires
[-] il permet de faire des rendus visuels intéressants grâce à des scènes et des costumes. On pourra alors proposer aux élèves de programmer des jeux par groupe.
[-] A partir de l'interface de développement, on peut envoyer sur un espace de partage en ligne son programme. Ce qui peut faciliter l'échange avec l'enseignant. On trouve déjà sur cet espace de partage de nombreux programmes de maths niveau lycée.
[-] Il existe un forum en français autour de Scratch sur lequel on a déjà pas mal discuté de l'algorithmique en seconde.

Les élèves seront eux-même un jour confrontés au choix d'un langage de programmation, face à la multiplicité des langages qui sont disponibles. Il n'est donc
pas interdit de leur montrer cette multiplicité et de réaliser quelques algorithmes dans plusieurs langages. Plus ils

manipulent tôt plusieurs syntaxes, plus ils auront de facilité à s'adapter à un nouveau langage.

Petite Bibliographie

Choisie parmi une cinquantaine de livres que j'ai lus ou consultés sur le sujet , je vous propose une sélection de dix ouvrages qui selon moi fournissent le plus d'exemples d'algorithmes simples :

[-] « Des Algorithmes Aux Langages, Basic, L.S.E., Logo » de Jacques Lopez chez Hachette
[-] Les documents d'accompagnement des programmes de seconde de 2000 et 2009
[-] « Algorithmique : techniques fondamentales de programmation avec des exemples en PHP » de Sébastien Rohaut (eni editions , Informatique technique)
[-] « Apprendre à programmer avec Python » de Gérard Swinnen chez Eyrolles
[-] « Javascript La Référence » (O'Reilly)
[-] « Le langage C » de Kernighan et Ritchie chez Masson
[-] « The C++ programming language » de Bjarne Stroustrup chez Addison-Wesley
[-] « Algorithmique, Applications en C » de Jean-Michel Léry chez Pearson Education
[-] « Algorithmique en C++ » de Jean-Michel Léry chez Pearson Education
[-] « De l'analyse à l'algorithme, méthodologie de programmation informatique pour l'ingénieur », de Dufour et Bialobroda chez Nathan

Dans les deux livres de Léry, les exemples sont d'abord écrits en pseudo-code avant d'être écrits en C ou en C++. Ils sont simples, nombreux et pris souvent dans le domaine des mathématiques.

---

[1] Je pars du principe que, tout comme le singe, nous apprenons beaucoup par mimétisme, surtout en informatique.

[2] Le pas à pas doit être possible, avec vitesse paramétrable et guidage possible au clavier. Mails le déboggage ne se limite pas à cela. On devrait pouvoir suivre les variables pendant le déroulement du programme.