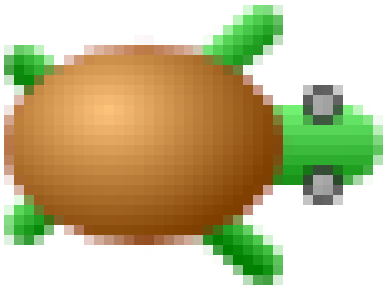
http://irem.univ-reunion.fr/spip.php?article974

La programmation au brevet des collèges 2018

- Algorithmique et programmation
- CoffeeScript
- Sofus, ou la programmation visuelle par blocs en collège (et au lycée)
-



Date de mise en ligne : vendredi 22 juin 2018

Le retour des programmes de calcul permet de mettre le flow programming en situation.

Pondichéry

Le sujet comportait, en exercice 4, un programme de calcul, et en exercice 5, un calcul de Ï€ par la méthode de Monte-Carlo.

Exercice 4

En fait il y avait deux programmes de calcul et l'objet de l'exercice était essentiellement la recherche des nombres donnant le même résultat avec ces deux programmes.

Programme A	**Programme B**
" Choisir un nombre " Soustraire 3 " Calculer le carré du résultat obtenu	" Choisir un nombre " Calculer le carré de ce nombre " Ajouter le triple du nombre de départ " Ajouter 7

Avant de réutiliser Sofus pour programmer A et B en calcul formel, on va d'abord montrer, sur ces deux programmes, comment mettre en oeuvre flow974

Préliminaires

Lorsqu'on démarre flow974, il y a déjà une fonction, dont on n'a pas besoin pour ce corrigé. On a donc un peu de nettoyage préliminaire à faire (clic droit sur un bloc puis *remove node* pour l'enlever, pour ne garder qu'un des nombres présents [1]. On en profite pour ajouter un affichage pour connaître le « résultat des programmes » ; du point de vue fonctionnel, il suffit d'utiliser la fonction *identité* qui ne modifie rien mais affiche sa valeur. On a donc laissé un *nombre*, et, si on effectue un clic droit sur la partie vide du plan de travail, le menu qui apparaît alors permet, parmi les fonctions, de choisir l'identité :

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH295/10000000000001c30000014c4a921596-b82bf.png]

Une fois que c'est fait, l'identité est en jaune parce qu'elle vient juste d'être sélectionnée, et il reste encore à effectuer un branchement :

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH146/10000000000001a600000099d9faf4c4-e6954.png]

Pour cela, on clique sur la sortie du nombre et, sans lâcher le bouton gauche de la souris, on mène celle-ci vers l'entrée de la fonction :

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH142/10000000000001ae00000098e95966ef-e70aa.png]

Une fois qu'on est arrivé et que le bouton gauche de la souris a été lâché, le branchement est effectué :

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH134/10000000000001a70000008dbebdf3cc-f551b.png]

On s'en rend compte parce que la modification du nombre en entrée se répercute immédiatement à la sortie.

1. Corinne

Le programme A modélise une fonction, que l'on va noter A(x) par la suite : Il est clair, selon l'énoncé, que le nombre obtenu en sortie dépend du nombre choisi en entrée, et c'est exactement ce que signifie le mot « fonction ».
Pour programmer proprement, on se propose de créer une fonction, ce qui se fait en cliquant quelque part entre l'entrée et la sortie, et en naviguant dans le menu apparu, jusqu'à trouver la fonction générique. Comme c'est une fonction, elle se trouve parmi les fonctions.

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH306/10000000000001ad00000148b7a91060-89bd1.png]

Mais elle va se nommer « module » après sa création :

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH98/100000000000026a00000097d4ae149c-f0e3e.png]

Pour la programmer et la renommer, il suffit alors de cliquer sur le bouton "Edit", ce qui fait entrer dans la fonction.

Une fonction se doit d'avoir une entrée : clic gauche puis choix de celle-ci dans "fonctions" :

[http://irem.univ-reunion.fr/local/cache-vignettes/L232xH310/10000000000000e80000013678d8a72e-798a5.png]

Une fonction se doit aussi d'avoir une sortie :

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH271/10000000000001d30000013c8bd3731d-a5dd7.png]

Comme l'étape suivante est de « soustraire 3 » on insère un module de soustraction. Il est dans les opérations :

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH217/10000000000002400000001386fc8e734-b1cd2.png]

C'est à l'entrée qu'on soustrait 3, d'où ce branchement :

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH126/100000000000029b000000d2a150e74e-83743.png]

Mais que soustraire ? Un nombre !

Et ce nombre est 3 :

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH143/100000000000028f000000ea1bb85a94-c948d.png]

Enfin il faut calculer le carré du résultat :

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH182/10000000000003070000016091379 0b4-6f7e6.png]

Après les branchements nécessaires, le contenu du module A(x) est le suivant :

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH153/100000000000033b0000013b53e06e57-a4975.png]

Au passage on a renommé le module en haut. Après avoir cliqué sur "Enregistrer le module" on est de retour au plan de travail, et là on peut brancher le nombre choisi à l'entrée de A(x) et la sortie de A(x) à l'identité qui sert d'affichage :

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH106/10000000000002610000000a18b514f84-d6248.png]

On vérifie en positionnant l'entrée sur 1, que Corinne obtient bien 4 en sortie (à droite de l'écran) :

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH107/10000000000002650000000a37a7c388e-7317f.png]

2. Tidjane

Pour répondre à la question de Tidjane, il faut créer un nouveau module nommé B(x) et commencer par y placer le module « carré » puisqu'on a besoin du carré du nombre de départ.

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH171/10000000000001c9000000c3e56cb120-26b0a.png]

Mais on ne trouve pas de module « triple ». Qu'à cela ne tienne, on va en créer un !
On va donc créer une fonction à l'intérieur d'une fonction : On la nomme « triple » et voici son contenu :

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH163/10000000000002e40000012dadce9bcb-b01dc.png]

En effet le triple d'un nombre, c'est le produit de 3 par ce nombre.

Avec ce module on peut facilement faire le programme B(x) :

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH125/10000000000003fa0000013ea343db62-0bb7a.png]

Ci-dessus la partie en jaune correspond à l'expression « ajouter le triple du nombre de départ ». Ce module B(x) permet de répondre à la question de Tidjane : Le nombre trouvé est 17.

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH144/100000000000028d000000eb8a6ed65c-9c556.png]

3. Lina

La formule saisie est `=B3^2+3*B3+7`

Pour étoffer un peu cette partie, on propose un prolongement : On verra dans la partie 4 que le but final de l'exercice est de savoir pour quel nombre choisi en entrée, les deux programmes A et B donnent le même résultat. Ce n'est pas difficile en flow programming : il suffit de mettre les deux modules A(x) et B(x) sur un même plan de travail, et d'y injecter le même nombre :

Si le nombre x est au moins 1, B(x) est plus grand que A(x) :

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH250/10000000000002cc000001be21208b8c-70251.png]

Si le nombre x est négatif ou nul, B(x) est plus petit que A(x) :

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH241/1000000000000338000001ef27bf1295-e311b.png]

Comme les nombres ne prennent que des valeurs entières, on n'a ici que la même information donnée par le tableur : La solution de l'équation A(x)=B(x) est quelque part entre 0 et 1. Ceci dit, dans l'exercice, tous les nombres qui interviennent dans les questions 1, 2 et 3 étant entiers, il faut aller jusqu'à la résolution de l'équation dans la question suivante, pour comprendre que dans cet exercice, « nombre » signifie « réel » et pas nécessairement « entier ».

Pour gagner en précision, on peut ajouter un module de division par 10 au nombre d'entrée, et explorer la situation avec une précision accrue :

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH210/1000000000000360000001c411dbb5c3-4ebc2.png]

4. Zoé

Le vocabulaire de Sofus n'est pas très similaire à celui de l'énoncé ; notamment

- Quand on soustrait 3, il faut dire *à qui* on soustrait 3 ;
- Lorsqu'on calcule un carré, il faut dire *quel est* le nombre à élever au carré ;
- Lorsqu'on finit un calcul, il faut dire *quoi faire* du résultat : L'afficher !
- Et il faut savoir que si le nombre de départ s'appelle « x », son triple s'écrit « 3×x »

Mais on s'y fait vite. Avec l'aide de Sofus, Zoé

- **prouve** que le programme A donne bien x^2-6x+9 :

[http://irem.univ-reunion.fr/local/cache-vignettes/L305xH354/10000000000001310000016216eba9b5-24448.png]

- **trouve** que le programme B donne x^2+3x+7 :

[http://irem.univ-reunion.fr/local/cache-vignettes/L373xH400/10000000000001760000192c66a7b61-21e15.png]

- **résout** l'équation $x^2-6x+9=x^2+3x+7$:

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH206/10000000000001c7000000ea2fabbc3f-8c0e0.png]

Annexe

Comme Python est de plus en plus un langage de programmation fonctionnelle, voici, à titre de comparaison avec les flow-programmes ci-dessus, les fonctions A et B programmées en Python :

```
def A(x):
 x -= 3
 x **= 2
 return x
def B(x):
 def triple(nombre):
 return 3*nombre
 y = x**2
 y += triple(x)
 y += 7
 return y<div class='code_download'   style='text-align: right;'> <a
href='http://irem.univ-reunion.fr/local/cache-code/04ed8a39f65c3ff0a113bec6b1e7cfb8.txt'   style='font-family: verdana,
arial, sans; font-weight: bold; font-style: normal;'>Télécharger
```

Exercice 5

La question 1 portait sur le théorème de Pythagore : Un triangle rectangle était d'ailleurs dessiné sur la copie d'écran de Scratch. La question 2 était une question de cours, portant sur la définition d'un disque. Mais la question 3 s'est montrée très synthétique, en associant le test sur la distance et le calcul de celle-ci par Pythagore. Le script aurait pu être simplifié en laissant Scratch effectuer le calcul de distance et en faisant un dessin à la place. En effet tout lutin de Scratch connaît la distance le séparant d'un autre lutin, et peut s'estampiller sur la scène.

Préparation

Dans Sofus, la tortue numéro 1 peut connaître à quelle distance se trouve la tortue numéro 0. On commence donc par créer la tortue 0, qui restera à l'origine ; puis on cache les deux tortues :

[http://irem.univ-reunion.fr/local/cache-vignettes/L271xH271/pondi1-3-f96cd.png]

Le script, avec la tortue 0 placée à l'origine du repère, est quand même plus simple que celui de l'énoncé :

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH176/pondi3-ff2e0.png]

Avec SofusPy

On remarque que le script de l'énoncé ne comprend aucun affichage de la variable *score* et qu'il est fait allusion aux « lignes 5, 6 et 7 du programme » donc à de la programmation textuelle.

En reprogrammant le script avec SofusPy, on obtient ce genre de blocs :

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH194/pondi6-2-deea1.png]

Pas très différents de ceux de l'énoncé. Mais SofusPy donne automatiquement la version Python, laquelle (après quelques retouches comme remplacer les Â« 0 Â» par des pointillés) ressemble à ceci :

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH103/pondi7-2-0f413.png]

Et là, on a bien des lignes de programme et pas des blocs à compter...

Nuage de points

Une petite variante du programme simplifié permet de dessiner en rouge les points à l'intérieur du disque et en bleu les autres :

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH250/pondi2-2-84ac2.png]

On obtient alors ce genre de nuage de points :

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH250/pondi4-2-57d17.png]

La variable *score* possède alors une signification : C'est le nombre de points rouges dans le nuage.

Avec SofusPy, les couleurs n'étant pas encore implémentées, on peut faire le choix de ne dessiner le point que lorsque la variable *score* est incrémentée et affichée sa valeur à la fin du script. Celui-ci peut être copié-collé vers SofusPy :

```
from turtle import * ; from math import *; from random import *
score = 0; reset()
for _ in range(120):
 setposition(randint(-100, 100), randint(-100, 100))
 Carre_de_OF = xcor()**2 + ycor()**2
 distance = sqrt(Carre_de_OF)
 if distance < 100:
 score = score + 1
 dot(10)
setposition(120,0)
write(score)<div class='code_download'   style='text-align: right;'> <a
href='http://irem.univ-reunion.fr/local/cache-code/4983a6f1790521558ac6f541fbaf8a64.txt'   style='font-family:
verdana, arial, sans; font-weight: bold; font-style: normal;'>Télécharger
```

Il produit ce genre de dessin où on devine, en pointillés, le disque de rayon 100 (la tortue est à droite, en forme de chevron) :

[http://irem.univ-reunion.fr/local/cache-vignettes/L283xH253/pondi8-865b3.png]

USA

Le sujet comportait deux exercices de programmation : Le 4 avec des dessins de carrés par Scratch et le 5, avec un dessin de frise par « un logiciel de géométrie dynamique ».

Exercice 4

L'exercice 4 démarre par la définition d'une procédure « carré ». Le côté du carré est une variable globale *côté*, comme le montre la version Python ci-dessous, obtenue avec la version péï de SofusPy, et plus précisément la deuxième ligne de la fonction [2] *carré()* : Â« global côté Â» qui explique à Python le caractère global de cette variable.

La définition en Blockly est très similaire :

[http://irem.univ-reunion.fr/local/cache-vignettes/L295xH228/us1-4a9b7.png]

La différence essentielle est que Scratch met le *stylo en position d'écriture* sans préciser quelle est cette position,

alors que Sofus *baisse le stylo*, la tortue étant vue d'au-dessus, et devant donc baisser son stylo pour que celui-ci laisse une trace de ses mouvements.

Le script principal montre aussi deux petites différences entre Scratch et Sofus :

- Alors que Scratch *met côté à 40*, Sofus *met 40 dans côté*. Ce qui est également suggéré par le pseudocode *côté ← 40*
- Alors que Scratch *ajoute à côté 20*, Sofus *augmente côté de 20*

Par ailleurs il est rappelé dans l'énoncé que *s'orienter à 90* (sans unité !) « signifie qu'on se dirige vers la droite » (qui « on » ? Quelle droite ?)

[http://irem.univ-reunion.fr/local/cache-vignettes/L336xH207/us2-26ebd.png]

Question 1

La question 1 porte sur un programme de calcul avec répétition, puisque seule la valeur de la variable *côté* est considérée. On peut donc ne laisser que les transformations de celle-ci :

[http://irem.univ-reunion.fr/local/cache-vignettes/L281xH146/us3-1aad8.png]

La question a porte sur une affectation (initialisation) et la réponse est donc 40, à lire dans le script ; la question b trouve sa réponse dans l'affichage du script ci-dessus :

```
60
80
100
120
```

Question 2

Le réglage de l'épaisseur du stylo est une fonction experte de Scratch, on peut donc s'étonner de sa présence alors même que « l'enseignement de l'informatique au cycle 4 n'a pas pour objectif de former des élèves experts, ni de leur fournir une connaissance exhaustive d'un langage ou d'un logiciel particulier ».

Ceci dit, le seul endroit dans la boucle où il ne faut pas mettre le bloc d'augmentation de l'épaisseur du crayon, est avant le bloc *carré*.

Question 3

La programmation fonctionnelle n'est pour autant pas totalement absente de ce sujet puisque, même si la variable *côté* est globale, elle est argument de la fonction *avancer* et l'avant-dernier bloc du nouveau script est traduit en Python (voir ci-dessous) par *avancer de (côté + 30)*. La notation parenthésée montre que *avancer de* est une fonction, et que son argument n'est plus la valeur actuelle de la variable *côté* mais la somme *côté+30*.

Le nouveau script

[http://irem.univ-reunion.fr/local/cache-vignettes/L334xH207/us4-1312c.png]

donne ceci

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH140/us5-2-39d7b.png]

soit le dessin 3. Le dessin 2 est à écarter parce qu'il n'y a pas les levers de stylo, et le dessin 1 pour une raison plus subtile : Après avoir dessiné un carré, la tortue a tourné en tout de 360° et est donc à nouveau sur l'axe des abscisses, ce qui est incompatible avec le dessin 1.

Version Python

```
from turtle import *
def carre():
 global cote
 pendown()
 for _ in range(4):
 forward(cote)
 left(90)
 penup()
reset()
setposition(-200, 0)
cote = 40
for _ in range(4):
 carre()
 forward(cote + 30)
 cote = cote + 20<div class='code_download'   style='text-align: right;'> <a
href='http://irem.univ-reunion.fr/local/cache-code/9a3b4db699f8a77d5ff49fcf75d3c594.txt'   style='font-family: verdana, arial, sans; font-weight: bold; font-style: normal;'>Télécharger
```

Le pseudocode de la procédure principale est

```
téléporter la tortue vers (-200, 0)
côté • 40
pour _ allant de 0 à 3
```

```
carre( )
avancer de (côté + 30)
côté • côté + 20
```

fin du pour

Exercice 5

Le logiciel de géométrie dynamique choisi, est, ici, CaRMetal, parce qu'il permet de programmer la construction de la frise en français.

Dessin du motif 1

Un moyen simple d'avoir un triangle isocèle par script est de donner des coordonnées aux sommets d'icelui :

a = Point("A",0,1);
b = Point("B",0,-1);
c = Point("C",-3,0);
Polygone("motif1","A,B,C"); <a href='http://irem.univ-reunion.fr/local/cache-code/76d94f4de50c4d9e27579f63add93bec.txt' style='font-family: verdana, arial, sans; font-weight: bold; font-style: normal;'>Télécharger

Mais pour être complet il faudrait fixer les sommets A, B et C, sinon, sous l'effet du Monkey, le triangle ne restera pas isocèle (l'effet est néanmoins intéressant).

Le motif 2

La transformation est la symétrie d'axe (AB), mais une translation aurait aussi pu faire l'affaire :

d = SymétrieAxiale("D",Droite("l1","A","B"),"C");
Cacher("l1");
Polygone("motif2","A,B,D"); <a href='http://irem.univ-reunion.fr/local/cache-code/dafdc9753b3de523ff85139e0c430cf6.txt' style='font-family: verdana, arial, sans; font-weight: bold; font-style: normal;'>Télécharger

La première ligne signifie simplement que D est le symétrique de C par rapport à la droite (AB), mais au passage cette droite a été nommée l1 pour pouvoir, à la ligne 2, la cacher.

La translation

Voici le script engendrant la frise, on peut aisément y lire le vecteur de la translation, même si on ne connaît pas par coeur la syntaxe des CaRSCripts :

```
répéter 4 fois {
 a = Translation("C","B",a);
 b = Translation("C","B",b);
 c = Translation("C","B",c);
 d = Translation("C","B",d);
 Polygone("_a,_b,_c");
 Polygone("_a,_b,_d");
}<div class='code_download'   style='text-align: right;'> <a
href='http://irem.univ-reunion.fr/local/cache-code/6074a9926bd5ea3fdabf98a6fe8f302e.txt'   style='font-family:
verdana, arial, sans; font-weight: bold; font-style: normal;'>Télécharger
```

On voit que le script a bien engendré la frise de l'énoncé :

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH159/usa2k18-6ffd3.png]

Le pseudocode

En mettant CaRMetal en mode pseudo-code, les effets des translations successives se voient mieux :

```
répéter 4 fois {
 a õ Translation("C","B",a);
 b õ Translation("C","B",b);
 c õ Translation("C","B",c);
 d õ Translation("C","B",d);
}<div class='code_download'   style='text-align: right;'> <a
href='http://irem.univ-reunion.fr/local/cache-code/a3852539c8ae393fd20ec4605060643d.txt'   style='font-family:
verdana, arial, sans; font-weight: bold; font-style: normal;'>Télécharger
```

Maroc

Le sujet comprenait un programme de calcul dans l'exercice 1 (la troisième question d'un « vrai ou faux justifié ») et un programme Scratch à compléter dans l'exercice 6 (la partie B).

Exercice 1

Encore un programme de calcul bourré d'implicites (à qui ajouter 5 ? à qui soustraire 9 ?) :

```
Choisir un nombre;
```

```
Ajouter 5;
Multiplier le résultat obtenu par 2;
Soustraire 9.
```

On devait dire si la fonction ainsi programmée coïncide avec 2x+1 (x étant le nombre choisi au départ). La réponse affirmative est donnée par Sofus :

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH324/ce6-e047d.png]

Python

Pour aller plus loin, on peut aussi utiliser SymPy (calcul formel sous Python), qui possède une variable *n* de type formel, laquelle peut être utilisée comme valeur initiale du *nombre*, et l'affichage final de cette variable donne bien 2n+1 :

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH202/ce7-66e9c.png]

Exercice 6

Cet exercice n'est pas une bonne publicité pour Scratch : On dit que le robot est orienté vers le haut alors qu'il doit effectuer des parcours horizontaux dans le champ ; et le « costume » choisi pour le robot dans l'illustration ne le montre pas vu de haut, alors que son tracé est donné en vue aérienne.

On ne sait pas si les élèves doivent dessiner des blocs ou seulement écrire du texte, en tout cas on leur demande clairement d'« écrire un programme simple », même si c'est dans des définitions de fonctions.

Voici le corrigé avec Sofus :

[http://irem.univ-reunion.fr/local/cache-vignettes/L400xH251/ce8-2-1b481.png]

Python

Le jour où un maraîcher voudra programmer un robot pour désherber ses tomates, il est peu probable qu'il le programme en Scratch. Python ayant plus le vent en cote, voici le script produit par SofusPy à partir des blocs ci-dessus :

```
from turtle import *
def Motif_montant():
 forward(80)
```

```
 right(90)
 forward(1)
 right(90)
def Motif_descendant():
 forward(80)
 left(90)
 forward(1)
 left(90)
for count in range(24):
 Motif_montant()
 Motif_descendant()
forward(64)<div class='code_download'   style='text-align: right;'> <a
href='http://irem.univ-reunion.fr/local/cache-code/737f031c9863983d57d6805082ac2ed1.txt'   style='font-family:
verdana, arial, sans; font-weight: bold; font-style: normal;'>Télécharger
```

---

[1] La première étape de chacun des programmes A et B est de « choisir un nombre »

[2] en réalité cette fonction est une procédure, puisqu'il n'y figure pas de ligne commençant par *return*. De plus elle n'a pas d'antécédent puisque les parenthèses sont vides.