COURS

4e

Programmation — Boucles, conditions et événements

Programme de mathématiques du cycle 4 — arrêté du 17 juillet 2020, BO n° 31 du 30 juillet 2020 — MENE2018714A

Un ordinateur ne devine rien : il exécute, une par une et dans l'ordre, les instructions qu'on lui donne. Trois outils suffisent pour lui en dire beaucoup : **la boucle répète, la condition choisit, l'événement déclenche.** Et une variable garde en mémoire ce qui change.

1 Algorithme, programme, séquence

DÉFINITION

Un **algorithme** est une suite finie d'instructions précises qui résout un problème. Il peut s'écrire en français, sans ordinateur : c'est l'**algorithmique débranchée**. Un **programme** le traduit pour une machine ; dans Scratch, des blocs emboîtés forment des **scripts**, qu'exécute un **lutin**. Les instructions d'une **séquence** s'exécutent **une par une, de haut en bas** : changer leur ordre change, en général, le résultat.

ÉCRIRE UN ALGORITHME DE DÉPLACEMENT

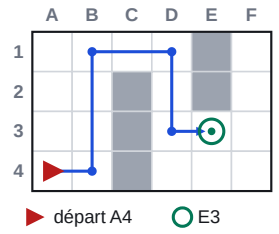
1. **Suivre le trajet case par case** avec trois instructions : « avancer de n cases », « tourner à gauche », « tourner à droite » (un quart de tour **sur place**), en lisant chaque rotation **du point de vue du robot**.
2. **Exécuter l'algorithme obtenu** depuis le départ, et vérifier l'arrivée.

EXEMPLE

Écrire un algorithme qui mène le robot de A4 à E3 sans entrer dans une case grise. Il regarde vers la droite.

1. Une case jusqu'à B4 (C4 est grise) ; à gauche, 3 cases vers le haut jusqu'à B1 ; à droite, 2 cases jusqu'à D1 (E1 est grise).
2. À droite : le robot regarde vers le **bas**, avance de 2 cases jusqu'à D3. Pour aller vers la droite de la feuille, il tourne à **sa gauche** et avance d'une case. Cases parcourues :
$$1 + 3 + 2 + 2 + 1 = 9$$

avancer de 1 · tourner à gauche · avancer de 3 · tourner à droite · avancer de 2 · tourner à droite · avancer de 2 · tourner à gauche · avancer de 1



Le trajet ; les points bleus marquent les rotations.

PIÈGE

En D3, écrire « tourner à droite » parce que E3 est à droite sur la feuille. — le robot descend : sa droite est alors la **gauche** de la feuille, et il partirait vers C3, une case grise.

Ce que ça coûte : toute la fin de l'algorithme est fausse.

Le réflexe : tourner la feuille dans le sens de la marche avant d'écrire chaque rotation.

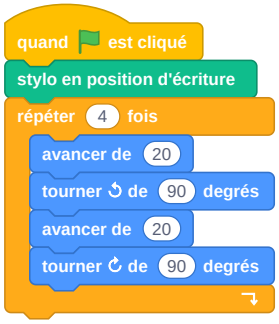
2 La boucle « répéter … fois »

DÉFINITION

`répéter 4 fois` exécute les blocs qu'il contient **4 fois de suite** ; chaque exécution est un **passage**. Ensuite seulement, le programme passe au bloc placé **sous** la boucle. Elle remplace une séquence où un même groupe d'instructions, le **motif**, est écrit plusieurs fois. Rappel : `tourner ↺` fait pivoter le lutin **vers sa gauche**, `tourner ↻` **vers sa droite** ; l'orientation 90 est vers la droite, 0 vers le haut.

REMPLACER UNE SÉQUENCE PAR UNE BOUCLE

1. Repérer le **plus petit groupe** d'instructions qui revient à l'identique : le motif. Compter ses retours.
2. Mettre le motif **une seule fois** dans la boucle ; placer **avant** elle ce qui ne se fait qu'une fois : position, orientation, stylo.
3. Contrôler : passages × blocs du motif = instructions de la séquence. Ci-contre, 16 instructions en une boucle :
$$4 \times 4 = 16$$



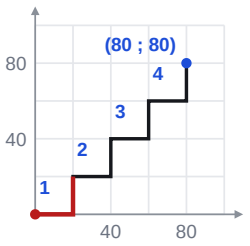
Le motif est exécuté 4 fois.

EXEMPLE

Le lutin part de $(0\,;0)$, orienté à 90. Que trace le script ci-dessus ? Où arrive-t-il ?

1. Un passage trace **une marche** de 20 sur 20 et laisse le lutin tourné vers la droite : chaque passage ajoute 20 à x et à y, et 40 pas de trait.
   $4 \times 20 = 80$
   et 4 × 40 = 160 pas de trait

un escalier de 4 marches ; le lutin arrive en (80 ; 80)



En rouge, le premier passage.

RÈGLE

À savoir écrire seul. Le lutin doit faire un tour complet. Carré : `répéter 4 fois [ avancer de 100 · tourner ↻ de 90 degrés ]`. Triangle équilatéral : `répéter 3 fois [ avancer de 100 · tourner ↻ de 120 degrés ]` — à chaque sommet, il tourne de $180 - 60 = 120$ degrés, pas de 60.

PIÈGE

Écrire `répéter 8 fois` pour l'escalier, parce qu'on voit 8 segments. — le motif trace **deux** segments : avec 8 passages, le lutin trace 8 marches.

Ce que ça coûte : une figure deux fois trop grande.

Le réflexe : passages = segments du dessin ÷ segments du motif, ici $8 \div 2 = 4$.

3 Les variables

DÉFINITION

Une **variable** est un emplacement de la mémoire, repéré par un **nom**, qui contient **une seule valeur** à la fois. `mettre a à 7` range 7 dans `a` : **l'ancienne valeur est effacée**. `ajouter 3 à a` augmente de 3 la valeur actuelle. Ce que tape l'utilisateur après `demander … et attendre` est rangé dans `réponse`. Dans une boucle, `avancer de côté` puis `ajouter 10 à côté` trace des segments de plus en plus longs. Scratch note la multiplication `*`, écrite × ici.

SUIVRE UNE VARIABLE DANS UN TABLEAU

1. Une **colonne par variable**, une **ligne par instruction exécutée** (3 passages de boucle : 3 groupes de lignes).
2. Calculer chaque ligne avec les valeurs de la **précédente** ; la réponse est sur la **dernière**.

EXEMPLE

`demander Choisis un nombre et attendre · mettre a à réponse · mettre a à a + 3 · mettre a à a × a · dire a.` Qu'affiche ce script pour 7 ? pour −5 ? Que calcule-t-il ?

1. Pour 7 : `a` vaut 7, puis 10, puis $a \times a$ avec la valeur **actuelle**.
   $10 \times 10 = 100$
2. Pour −5 :
   $(-5+3) \times (-5+3) = (-2) \times (-2) = 4$
3. Pour un nombre x : `a` vaut x, puis $x + 3$, puis $(x+3) \times (x+3) = (x+3)^2$.

le lutin dit 100 pour 7 et 4 pour −5 ; le script calcule $(x+3)^2$

PIÈGE

Lire `mettre a à a + 3` comme l'équation « $a = a + 3$ », impossible. — ce bloc est une **affectation** : on calcule $a + 3$ avec l'ancienne valeur de `a`, puis on range le résultat dans `a`.

Ce que ça coûte : le programme de calcul n'est pas exécuté : la question est perdue.

Le réflexe : lire « `a` **devient** $a + 3$ », et écrire la valeur avant et après chaque bloc.

4 Les instructions conditionnelles

DÉFINITION

Une **condition** vaut **vrai** ou **faux** au moment où elle est testée : `réponse < 14`, `n = 0`, `touche espace pressée ?`, `toucher Souris ?`. Les comparaisons < et > sont **strictes** : $14 < 14$ est faux. `si condition alors [ A ]` exécute A **seulement si** elle est vraie. `si condition alors [ A ] sinon [ B ]` exécute **A ou B, jamais les deux, jamais aucun**.

EXÉCUTER UNE CONDITION À LA MAIN

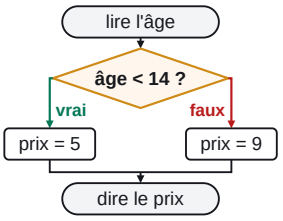
1. Remplacer chaque variable par sa valeur **à cet instant** ; décider vrai ou faux en regardant de près la **valeur frontière**.
2. Suivre une seule branche, **barrer l'autre**, puis reprendre sous le bloc.

EXEMPLE

Un cinéma fait payer 5 € aux moins de 14 ans, 9 € aux autres. `demander Ton âge ? et attendre` · `si réponse < 14 alors [ mettre prix à 5 ] sinon [ mettre prix à 9 ]` · `dire prix`. Que dit-il pour 13, 14, 15 ?

1. 13 : $13 < 14$ est vrai, branche « alors » : 5.
2. 14 : $14 < 14$ est faux, branche « sinon » : 9 — juste, car à 14 ans on n'a plus moins de 14 ans. 15 : faux, 9.

le script respecte le tarif : 5 €, 9 €, 9 €



Un test, deux chemins qui se rejoignent.

PIÈGE

Écrire `si réponse > 14 alors [ mettre prix à 9 ] sinon [ mettre prix à 5 ]` pour « 9 € à partir de 14 ans ». — $14 > 14$ est faux : à 14 ans, on paierait 5 €. Essayé avec 10 et 20, le script semble juste : seule la frontière révèle l'erreur.

Ce que ça coûte : un script faux pour la valeur que le correcteur teste en premier.

Le réflexe : tester toujours la valeur frontière elle-même.

PIÈGE

Remplacer `si n < 10 alors [ ajouter 5 à n ] sinon [ ajouter -10 à n ]` par deux `si` à la suite. — le second test est **toujours** fait, avec la valeur modifiée : pour $n = 8$, on obtient $8 + 5 - 10 = 3$ au lieu de 13.

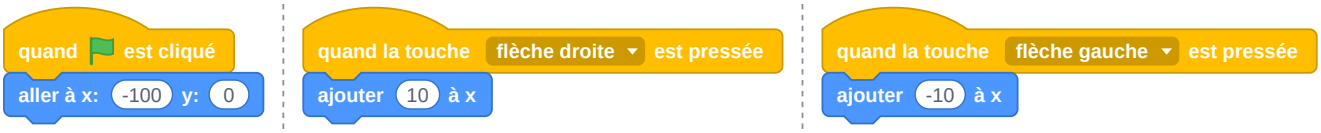
Ce que ça coûte : une valeur finale fausse, sans alerte.

Le réflexe : deux cas qui s'excluent : un seul `si … alors … sinon`.

5 Déclencher un script par un événement

DÉFINITION

Un **événement** est ce qui arrive pendant que le programme fonctionne. Le premier bloc d'un script, arrondi en haut, est son **déclencheur** : `quand le drapeau vert est cliqué`, `quand la touche espace est pressée`, `quand ce lutin est cliqué`, `quand je reçois …`. Le script s'exécute **chaque fois** que son événement se produit ; **sans déclencheur, il ne démarre jamais**.



Trois scripts d'un même lutin, chacun lancé par son événement.

PROGRAMMER UN DÉPLACEMENT AU CLAVIER

1. **Un script par touche** : `ajouter 10 à x` vers la droite, `ajouter -10 à x` vers la gauche. Ce bloc bleu change l'**abscisse** ; l'orange `ajouter 10 à score`, une **variable**.
2. Un script `quand le drapeau vert est cliqué` qui **initialise** : position, orientation, variables.

EXEMPLE

On clique sur le drapeau, puis on appuie 7 fois sur la flèche droite et 2 fois sur la flèche gauche. Où est le lutin ?

1. Le drapeau place le lutin en $(-100\,;0)$; chaque appui relance le script de sa touche, et aucun ne modifie y.
   $-100 + 7 \times 10 - 2 \times 10 = -50$

le lutin est en (−50 ; 0)

PIÈGE

Mettre `si touche flèche droite pressée ? alors [ ajouter 10 à x ]` sous le drapeau, sans boucle. — le test est fait **une seule fois**, au clic sur le drapeau, quand personne n'appuie encore : la flèche ne fait rien.

Ce que ça coûte : le jeu ne réagit pas au clavier.

Le réflexe : un déclencheur de touche, ou le test dans `répéter indéfiniment`, qui le refait sans arrêt : c'est ainsi qu'un lutin réagit quand il en touche un autre.

6 Mettre au point : tester, corriger

METTRE AU POINT UN PROGRAMME

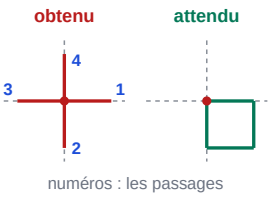
1. **Prévoir** sur un cas simple ce que le programme doit produire.
2. **Exécuter et comparer** ; chercher le **premier** bloc où l'exécution s'écarte de la prévision.
3. **Corriger ce seul bloc, puis retester**, valeurs frontières comprises.

EXEMPLE

Pour un carré de côté 50, Nora écrit, stylo baissé, `répéter 4 fois [ aller à x:0 y:0 · avancer de 50 · tourner ↻ de 90 degrés ]`, lutin en $(0\,;0)$ orienté à 90. Elle obtient une croix. Corriger.

1. Passage 1 : le lutin va en $(50\,;0)$ et se tourne vers le bas. Passage 2 : `aller à x:0 y:0` **le ramène au centre**, il descend en $(0\,;-50)$ au lieu de $(50\,;-50)$: premier écart.
2. On place `aller à` **avant** la boucle. Retest : le carré se ferme.
   $4 \times 90 = 360$

`aller à x:0 y:0` était dans la boucle ; placé avant elle, le script trace le carré



numéros : les passages

La croix de Nora et le carré attendu.

PIÈGE

Modifier trois blocs d'un coup, sans retester entre deux. — on ne sait plus laquelle a réparé, ou cassé, le programme.

Ce que ça coûte : un programme juste par hasard, impossible à justifier.

Le réflexe : une correction, un test.

7 Plus loin : répéter jusqu'à, blocs, scripts en parallèle

Cette partie va au-delà des attendus de fin de 4e : le **niveau 3** est attendu en fin de 3e. Elle sert à prendre de l'avance ou à revoir.

DÉFINITION

`répéter jusqu'à ce que condition [ A ]` teste la condition **avant chaque passage** : tant qu'elle est fausse, A est exécuté ; dès qu'elle est vraie, on sort. On s'en sert quand on ne sait pas d'avance combien de passages il faudra.

EXEMPLE

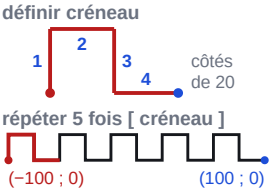
Inès a 40 € et en ajoute 15 chaque semaine : `mettre somme à 40 · mettre semaines à 0 · répéter jusqu'à ce que somme > 200 [ ajouter 15 à somme · ajouter 1 à semaines ] · dire semaines`. Que dit le lutin ?

1. Après 10 passages, `somme` vaut $40 + 15 \times 10 = 190$, et $190 > 200$ est faux. Après 11, c'est vrai : on sort.
   $40 + 15 \times 11 = 205$

le lutin dit 11 : il faut 11 semaines pour dépasser 200 €

DÉFINITION

Dans « Mes blocs », on **crée un bloc**, par exemple `créneau`, et on écrit sous `définir créneau` les instructions d'un **motif** : monter de 20, avancer de 20, descendre de 20, avancer de 20. On **décompose** ainsi un problème : chaque sous-problème devient un bloc. `répéter 5 fois [ créneau ]` depuis $(-100\,;0)$ mène le lutin en $(100\,;0)$, car $-100 + 5 \times 40 = 100$, et trace $5 \times 4 \times 20 = 400$ pas.



Le motif, puis la frise.

RÈGLE

Plusieurs scripts peuvent s'exécuter en même temps : tous ceux qui ont le même déclencheur démarrent ensemble, chez un ou plusieurs lutins ; `envoyer à tous fin` lance les scripts `quand je reçois fin`. **Pour simuler le hasard**, `nombre aléatoire entre 0 et 1` donne 0 ou 1 avec la même chance : 100 tirages et un `si … sinon` comptent les 0 et les 1.

PIÈGE

Écrire `répéter jusqu'à ce que somme = 200` dans le script d'Inès. — `somme` vaut 40, 55, …, 190, 205 : jamais 200, car $200 - 40 = 160$ n'est pas un multiple de 15. La boucle ne s'arrête jamais.

Ce que ça coûte : un programme qui tourne sans fin.

Le réflexe : une condition d'arrêt avec `>` ou `<`, qu'on est sûr d'atteindre.

À VÉRIFIER

- Un **déclencheur** par script ? Ce qui ne se fait qu'une fois **avant** la boucle ? Gauche et droite **du point de vue du lutin** ?
- Chaque variable calculée avec sa valeur **actuelle** ? La **valeur frontière** testée ? Un **retest** après correction ?

À RETENIR

- Les instructions s'exécutent **une par une, dans l'ordre**.
- Ce qui ne se fait qu'une fois se place **avant** la boucle.
- `mettre a à a + 3` calcule avec l'**ancienne** valeur de `a`.
- `si … alors … sinon` suit **une seule** branche ; on teste la valeur frontière.
- Un script ne démarre que sur son **événement** ; on le teste, on le corrige, on le reteste.