

# Scratch : Guide de prise en main rapide

*Pourquoi scratch ?*

*Malgré son apparence ludique, Scratch est un vrai langage de la programmation. C'est un langage gratuit, graphique (on déplace des blocs, on ne tape pas du texte), dont l'aspect et le résultat obtenu plaît aux élèves, avec une programmation de type événementielle.*

Scratch peut s'utiliser directement en ligne : <https://scratch.mit.edu/>.

En cliquant en haut sur « Créer » on arrive directement sur l'interface.

Pour plus de confort et pour pouvoir utiliser Scratch hors ligne, on peut également l'installer gratuitement. Pour cela il faut se rendre ici : <https://scratch.mit.edu/scratch2download/>

Pensez bien à télécharger et installer en premier « Adobe Air » puis ensuite « Scratch ».

A la première ouverture, l'interface est en Anglais, mais on peut très facilement changer de langue en cliquant sur le « Globe » en haut à gauche.

Voilà, vous êtes prêts à utiliser Scratch et à programmer vos premiers algorithmes !

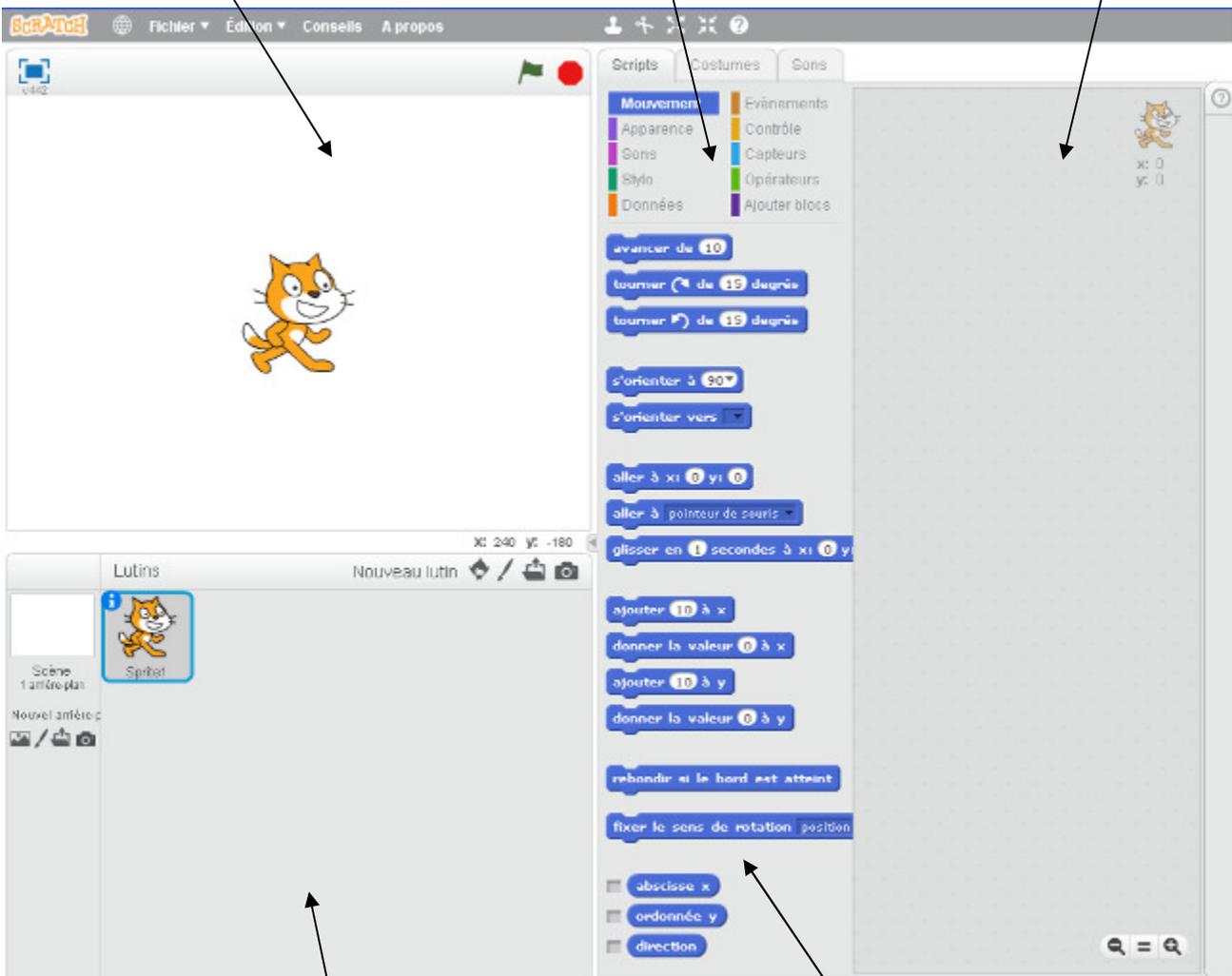


# L'espace de travail

La **scène** permet d'exécuter le programme.

Dans ces **menus** se trouvent les instructions à faire glisser dans la zone de script.

C'est dans la **zone de script** que l'on assemble les instructions du programme.



Zone de gestion et de création des **lutins** et/ou des **arrière-plans**.

Chaque menu propose des instructions à compléter et à faire glisser dans la zone de script.

# La programmation :

Une instruction est représentée par une « pièce de puzzle ».

La programmation se fait en déplaçant des instructions. Un script peut être déclenché en cliquant dessus, ou, plus souvent, suite à un événement (clic du souris, appui sur une touche, bruit, passage devant la caméra et très souvent et plus simplement, un clic sur le drapeau vert).



On trouve les événements sous forme de chapeaux beiges dans la « boîte » **Evènements**.

Un script est constitué d'instructions collées. Si des blocs ne sont pas collés, ils sont indépendants.



Certaines instructions ont besoin d'avoir des informations complémentaires.

Les nombres sont représentés sous forme de ronds ou de rectangles arrondis.

Les blocs rectangulaires représentent des textes ou des nombres.

Les blocs hexagonaux représentent des tests qui peuvent être vrai ou faux.



## Le lutin

(Traduction de *sprite*). On peut avoir plusieurs lutins. Chaque lutin peut avoir plusieurs costumes.

Un costume est un dessin pour représenter le lutin. Un changement de costume peut permettre de

simuler un déplacement ou une animation. **basculer sur costume**



Deux costumes pour un sprite (fourni par scratch). En passant alternativement d'un costume à l'autre, on donne l'impression que la chauve-souris vole.

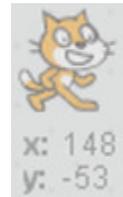
Chaque lutin peut contenir plusieurs scripts et peut donc réagir à plusieurs événements.

# Repère

Pour déplacer un lutin, on utilise notamment ses coordonnées dans un repère qui n'est pas affiché. L'origine du repère est au centre de la zone, l'orientation est l'orientation classique (de gauche à droite et de bas en haut).

x correspond à l'abscisse et y à l'ordonnée.

Les coordonnées vont de -240 à +240 pour l'abscisse et -180 à 180 pour l'ordonnée.



## Petite visite des instructions

Ci-dessous, quelques exemples pour donner une idée des instructions qui existent. Vous en découvrirez plusieurs au fur et à mesure de vos explorations.

### • Instructions de base

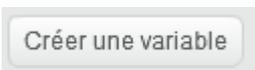


### • Boucles

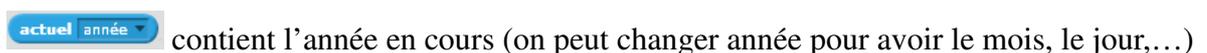
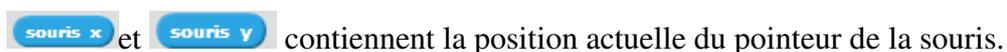
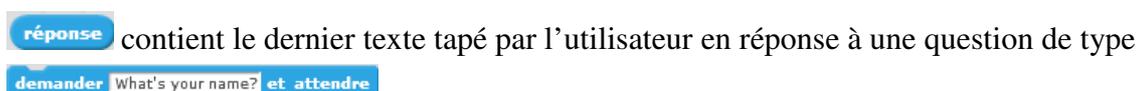


### • Variables, affectation, modification



(Remarque : au départ, la zone [Données] ne contient que  et . Elle commence à se remplir lorsqu'on a cliqué sur un bouton et créé une variable ou une liste. Pour les exemples ci-dessus, on a créé une variable nommée « longueur ».

Certaines variables sont prédéfinies (elles existent déjà). On peut les lire mais pas les modifier :



- **Instructions conditionnelles**



- **Opérateurs nombres, textes ou logiques**



- **Communication entre les lutins**



- **Bloc d'instruction (sous programme).**

Ceci est utile quand une liste d'instruction apparaît à plusieurs endroits du programme à l'identique (ou presque). Dans ce cas, on peut regrouper ces instructions dans un bloc et dans le programme, mettre un appel au bloc à la place des instructions.

Les blocs sont également pratiques pour structurer et découper un problème en sous-problèmes. Ainsi le programme final sera mieux partitionné et plus digeste.

Exemple : Définition d'un bloc permettant de dessiner un carré dont on doit préciser la longueur.



Si dans un programme, on met cet appel au bloc, il exécutera le bloc et dessinera un carré de côté 10.

## Les fiches d'exercices pour élèves

Il nous semble contre-productif de donner des fiches trop détaillées avec des sous questions et la liste des instructions qu'ils devront utiliser. Ni de montrer ce que le programme doit faire car les élèves risquent d'avoir du mal à se détacher de ce qu'ils ont vu. Il nous semble préférable de donner ce qu'on veut programmer (un tic tac toe, une discussion entre lutins, un programme de calcul), éventuellement leur éviter un ou deux écueils, puis les laisser se débrouiller, explorer, expérimenter. Ils ne feront pas tous le même programme, et c'est tant mieux. Tout ce qu'on doit tester à la fin, c'est si le programme fait bien ce qui est demandé.

## Comment déboguer

C'est souvent l'angoisse des collègues quand ils maîtrisent peu le logiciel. Mais c'est aux élèves de trouver leur erreur. Il est donc plutôt préférable de leur donner des conseils.

Note : en recherche d'erreurs, plutôt que de supprimer un bloc douteux : On peut simplement le décrocher. En effet, ainsi il ne s'exécutera plus et surtout, il faut savoir que toute suppression d'un bloc d'instruction est définitive. Il faudra le recréer.

Isoler un bloc permet aussi de l'exécuter séparément avec un simple clic.

On peut aussi conseiller aux élèves d'afficher certaines variables pour contrôler leurs valeurs.

On peut poser des arrêts dans le programme avec les instructions :



Un clic droit sur une instruction permet d'avoir une aide (en anglais) sur cette instruction.

On peut modifier « à la main » les valeurs des variables pour tester comment réagit le bloc avec ces valeurs.

Quelques sites internet permettent d'avoir des informations, faq, exemples, cartes tutoriel, ...

## Lexique Scratch

Une instruction est représentée par une « pièce de puzzle ». Elle fait quelque chose (déplace le lutin, affiche un message, lance un son, modifie une variable...).

Événements sous forme de chapeau beige dans la « boîte » [événement].

Un script est constitué d'instructions collées. Si des blocs ne sont pas collés, ils sont indépendants. Quand on lance un script (en cliquant dessus ou en activant l'événement « chapeau » au dessus), il exécute les instructions les unes après les autres.

Lutin : Un lutin peut revêtir plusieurs costumes (dessin). Quand on affiche, ou déplace le lutin, on manipule à l'écran son costume. Pour faire agir un lutin, on lui associe des scripts. On peut utiliser des lutins de la bibliothèque fournie, les dessiner, récupérer un lutin dans un fichier, ou utiliser un fichier image.

Costume : C'est un dessin. Un lutin a, à tout moment, un costume qui le représente. Pour avoir les costumes d'un lutin, cliquer sur l'onglet costume. On peut rajouter un costume (un dessin de la bibliothèque, un dessin qu'on fait avec l'éditeur de dessin, un fichier image ou provenant d'une webcam). Les costumes peuvent permettre d'exprimer quelque chose ou de réaliser des animations.

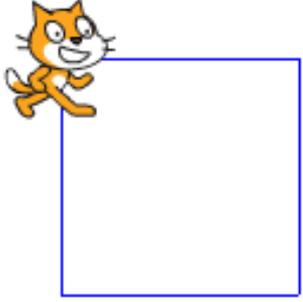
Pour changer de costume par programme, utiliser l'instruction



# Premiers programmes :

Pour une activité, le premier « programme » à faire doit être simple, peu ergonomique et l'utilisateur peut dans un premier temps faire une partie du travail. Par exemple, pour l'inégalité triangulaire, rentrer les nombres dans l'ordre, puis, pour ceux qui ont fini, leur demander d'améliorer le programme.

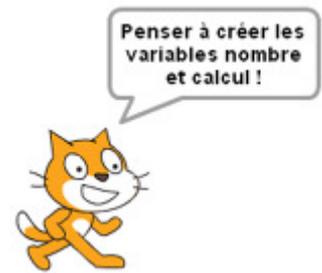
- Tracer un carré

Un premier programme	Qui donne ceci	Mais qui peut rapidement être amélioré
		

- Le magicien !

```

quand [drapeau vert] pressé
demander "Donne-moi un nombre entier, et attends"
mettre [nombre] à [réponse]
mettre [calcul] à [nombre + nombre + 1]
dire "J'ajoute ce nombre à celui qui le suit, pendant 2 secondes"
dire [regroupe Je trouve calcul] pendant 2 secondes
mettre [calcul] à [calcul + 9]
dire "J'ajoute 9 à ce résultat, pendant 2 secondes"
dire [regroupe Je trouve calcul] pendant 2 secondes
mettre [calcul] à [calcul / 2]
dire "Je divise ce résultat par 2, pendant 2 secondes"
dire [regroupe Je trouve calcul] pendant 2 secondes
mettre [calcul] à [calcul - nombre]
dire "Je soustrais le nombre de départ, pendant 2 secondes"
dire [regroupe Je trouve calcul] pendant 2 secondes
  
```



Mouvement	Evènements
Apparence	Contrôle
Sons	Capteurs
Stylo	Opérateurs
<b>Données</b>	Ajouter blocs

Créer une variable

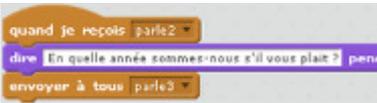
Créer une liste

- Choisir un nombre et le chercher par propositions et réponses : Plus grand/Plus petit

```
quand flag verte pressée
  cacher la variable nombre
  mettre compteur à 0
  mettre proposition à 0
  mettre nombre à nombre aléatoire entre 1 et 100
  répéter jusqu'à proposition = nombre
    demander Proposer un nombre et attendre
    mettre proposition à réponse
    ajouter à compteur 1
    si proposition < nombre alors
      dire Ton nombre est trop petit !! pendant 2 secondes
    sinon
      si proposition > nombre alors
        dire Ton nombre est trop grand !! pendant 2 secondes
      sinon
        dire regroupe BRAVO ! Tu as réussi en regroupe compteur tentatives!
```

The image shows a Scratch script for a number-guessing game. The script starts with a 'when green flag is clicked' event. It then performs several initialization steps: hides the 'nombre' variable, sets 'compteur' to 0, sets 'proposition' to 0, and sets 'nombre' to a random number between 1 and 100. A 'repeat until' loop is used to find the correct number. Inside the loop, the user is prompted to 'Proposer un nombre' (Propose a number), the input is stored in 'proposition', and the 'compteur' is incremented by 1. Two conditional checks follow: if 'proposition' is less than 'nombre', the user is told 'Ton nombre est trop petit !!' (Your number is too small !!) for 2 seconds; if 'proposition' is greater than 'nombre', the user is told 'Ton nombre est trop grand !!' (Your number is too large !!) for 2 seconds. If neither condition is met, the user is congratulated with 'BRAVO ! Tu as réussi en [compteur] tentatives!' (BRAVO ! You succeeded in [compteur] attempts!).

- Discussion entre deux lutins

		
	 	
	 	
	 	
	 	
<p>Etc...</p>		