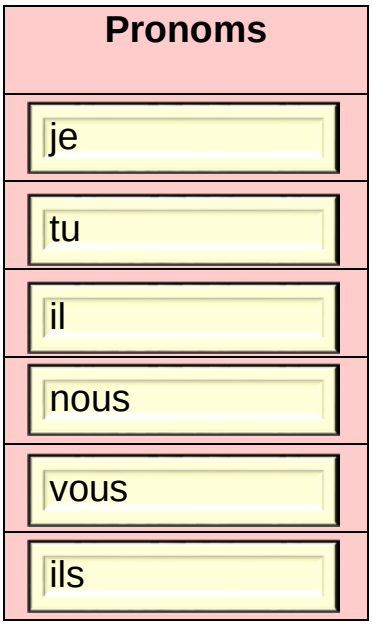
Les listes

Les listes dans Scratch

Une liste est une **variable**, avec **plusieurs emplacements** (plusieurs tiroirs).

Une liste a un **nom**, un certain nombre **d'emplacements de stockage**, une **longueur** égale à ce nombre d'emplacements.



Nous avons ici une liste nommée « Pronoms », de longueur égale à 6, dans laquelle sont mémorisés six pronoms personnels.

Pour accéder à un élément stocké dans la liste, il faut indiquer sa position dans la liste (son numéro de tiroir). La position est indiquée par « **un indice** » de stockage.

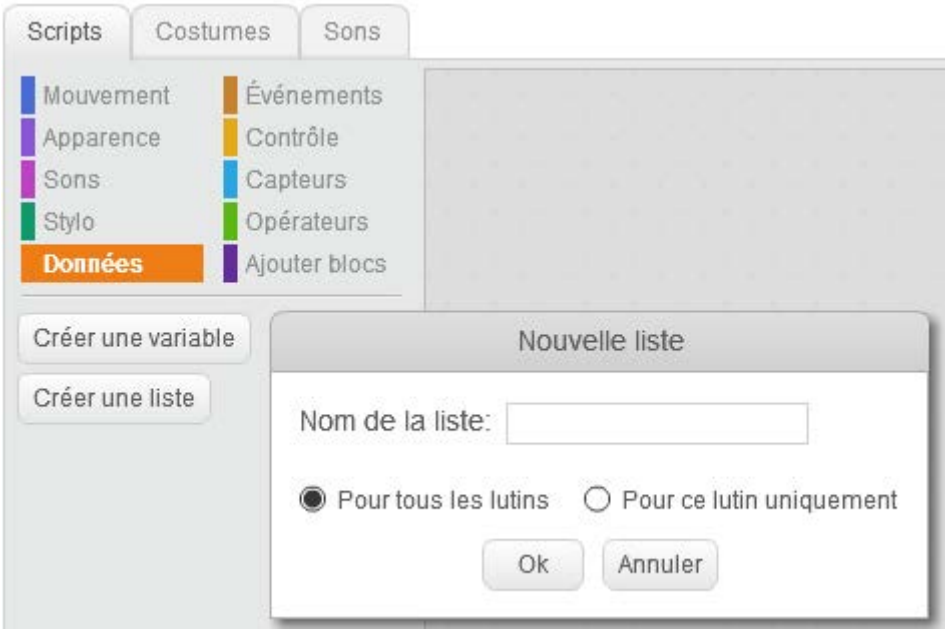
Dans Scratch, le premier élément de la liste est désigné par l'indice 1, le deuxième élément par l'indice 2, etc.

Par exemple, l'élément « nous » est désigné par l'indice 4 dans la liste « Pronoms ».

Créer et travailler avec des listes

La création d'une liste est identique à la création d'une variable.

Dans la palette, cliquer sur « Données » puis sur « Créer une liste »



Entrer le nom de la liste, et choisir sa portée :

- « pour tous les lutins » crée une liste globale, visible par tous les lutins du projet.

- « pour ce lutin uniquement » crée une liste locale, qui appartient uniquement au lutin sélectionné. Seul ce lutin pourra modifier, lire le contenu de la liste.

En cliquant sur « Ok » la liste est créée et elle apparait sur la scène. Les blocs spéciaux permettant de travailler avec la liste, apparaissent également dans la palette.

Ce bloc ajoute un élément en fin de liste « Pronoms »

Ce bloc ajoute un élément à la position indiquée dans la liste. Nous avons aussi la possibilité

d’indiquer que l’insertion se fait après le dernier élément de la liste, ou au hasard.

Ce bloc permet de modifier le contenu d’un emplacement d’indice x et d’y mettre une nouvelle valeur. Nous avons aussi la possibilité d’indiquer que le remplacement se fait sur le dernier élément de la liste, ou au hasard.

Ce bloc supprime l’élément d’indice x de la liste. Nous avons aussi la possibilité d’indiquer que cette suppression se fait sur le dernier élément de la liste, ou que l’on supprime tous les éléments de la liste.

Ce bloc permet de lire la valeur de l’élément d’indice x de la liste. Nous avons aussi la possibilité d’indiquer que nous lisons le dernier élément de la liste, ou nous lisons un élément pris au hasard dans la liste.

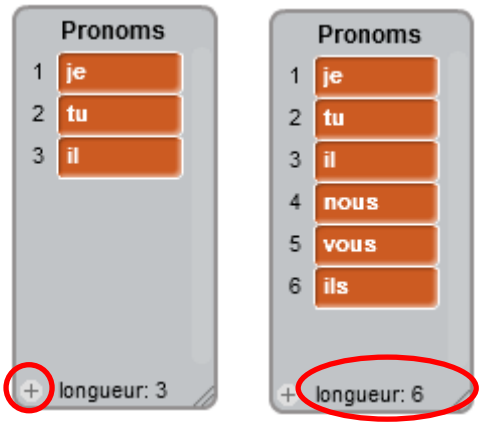
Ce bloc fournit la longueur de la liste, son nombre d’éléments.

Ce bloc permet de tester si un des éléments de la liste est égal au paramètre fourni : ici je teste si ma liste « Pronoms » a un élément égal à « ils ».

Remplissage manuel d'une liste

Nous pouvons aussi entrer des éléments dans une liste à la main. Il suffit de cliquer sur le (+) situé en bas à gauche de la liste.

Ici en cliquant sur le (+) j'accède au premier élément de la liste et il suffit ensuite de le remplir. Un appui sur la touche « entrée » passe à l'élément suivant.



Exercice

Nous allons repartir du projet « Conjugaison1 » vu dans les exercices sur les chaines de caractères.

- Modifier le projet en créant un bloc « **ExtraireRadical** » qui reçoit un verbe du premier groupe en paramètre, extrait le radical de ce verbe, et mémorise le résultat, dans une variable « **radical** », **locale** au lutin.

  On utilisera directement la variable « réponse » qui contient le verbe tapé par l'utilisateur. Tester.

- Ajouter au projet, 2 listes **locales** au lutin : « **Pronoms** », « **Terminaisons** » et une liste **globale** « **Verbes** ». Vous remplirez ces trois listes à la main.

  Pour les verbes, vous pouvez mettre la liste de verbes suivante :

  brûler, cacher, casser, couler, briller, chanter, fermer, marcher, monter, montrer, porter, poser, rouler, rêver, travailler, tromper, voler.

  Nous évitons dans cette liste de mettre les verbes présentant quelques exceptions comme manger, nettoyer, ou des verbes commençant par une voyelle, ou des verbes pronominaux .

- Modifier le programme pour que le verbe à conjuguer soit tiré au hasard dans la liste des verbes. Ensuite le lutin récite la conjugaison de ce verbe. Tester.

- Le projet final : nous rajoutons un deuxième lutin (le fantôme)

- Le fantôme tire un verbe au hasard dans la liste des verbes, place ce verbe dans une variable **globale** « **AConjuguer** », affiche le verbe à conjuguer et demande au lutin « Nano » de conjuguer.
- Nano conjugue le verbe et lorsqu'il a terminé, envoie au fantôme un message lui indiquant qu'il a terminé. Le fantôme en recevant ce message se cache.

Les listes numériques

Dans Scratch, nous pouvons créer des listes alphabétiques ou numériques. Dans Algobox, nous ne pouvons créer que des listes numériques.

Les listes numériques sont utilisées dans de très nombreux problèmes en mathématiques.

Nous allons les utiliser sur deux exemples simples.

Exercice 1 : Trouver tous les diviseurs d'un nombre entier.

Un nombre b est le diviseur d'un nombre entier a, si le reste de la division euclidienne de a par b vaut 0.

Exemple : Diviseurs de 36

$36 \div 1 = 36$	$36 \div 9 = 4$
$36 \div 2 = 18$	$36 \div 12 = 3$
$36 \div 3 = 12$	$36 \div 18 = 2$
$36 \div 4 = 9$	$36 \div 36 = 1$
$36 \div 6 = 6$	

La liste des diviseurs de 36 est : 1, 2, 3, 4, 6, 9, 12, 18, 36

Diviseurs de 45

$45 \div 1 = 45$	$45 \div 9 = 5$
$45 \div 3 = 15$	$45 \div 15 = 3$
$45 \div 5 = 9$	$45 \div 45 = 1$

La liste des diviseurs de 45 est : 1, 3, 5, 9, 15, 45

L'algorithme :

Dans l'ensemble des nombres entiers $\mathbb{N}$, si b divise a alors $b \leq a$.

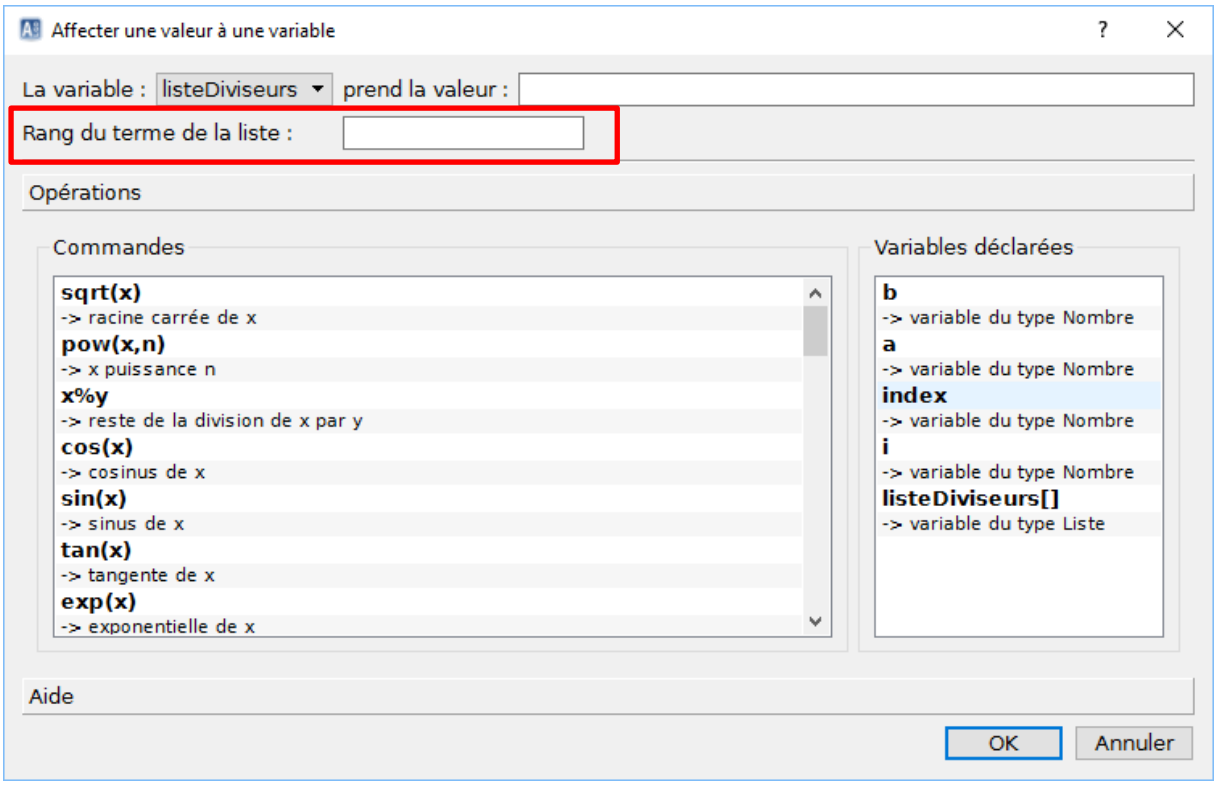
Pour trouver tous les diviseurs de a, on examine tous les nombres entiers b, inférieurs ou égaux à a.

Pour chacun de ces nombres b, on teste si le reste de la division a÷b est égal à zéro.

Si oui b est un diviseur de a et nous le mémoriserons dans une liste, sinon ce n'est pas un diviseur de a.

Écrire cet algorithme dans Algobox.

- Dans Algobox, une variable liste est déclarée de type « liste ».
- Pour obtenir le reste d'une division euclidienne, on utilise la fonction modulo (%) : a%b
- Pour affecter une valeur à un élément de la liste, il faut indiquer son rang dans la liste. Ce rang commence à 1.



- Algobox ne fournit pas la taille d'une liste.

Nous utiliserons une variable « **index** », pour désigner à chaque instant le nouvel emplacement de la liste, où il faut déposer le diviseur trouvé.

Lorsqu'on a trouvé tous les diviseurs, index pointe vers un emplacement vide.

Pour afficher le contenu de la liste des diviseurs, il faut donc la parcourir du rang 1 au rang (index-1).

Ci-dessous, l'algorithme Algobox de recherche des diviseurs d'un nombre. Cet algorithme est commenté. Prenez-le temps de bien l'étudier et de le comprendre.

```
1   VARIABLES
2     b EST_DU_TYPE NOMBRE
3     a EST_DU_TYPE NOMBRE
4     listeDiviseurs EST_DU_TYPE LISTE
5     index EST_DU_TYPE NOMBRE
6     i EST_DU_TYPE NOMBRE
7   DEBUT_ALGORITHME
8     //Nombre dont on cherche les diviseurs
9     LIRE a
10
11    //diviseur potentiel
12    b PREND_LA_VALEUR 1
13
14    //index désigne la place où il faut enregistrer le diviseur dans la liste
15    index PREND_LA_VALEUR 1
16
17    //On va examiner tous les diviseurs potentiels <= a
18    TANT_QUE (b <=a) FAIRE
19      DEBUT_TANT_QUE
20
21      //On examine le reste de la division de a par b
22      SI ((a%b)==0) ALORS
23        DEBUT_SI
24        //On a trouvé un diviseur, on le mémorise dans la liste
25        listeDiviseurs[index] PREND_LA_VALEUR b
26
27        //index est incrémenté pour dédigner l'emplacement suivant dans la liste
28        index PREND_LA_VALEUR index+1
29        FIN_SI
30
31      //On passe au diviseur potentiel suivant
32      b PREND_LA_VALEUR b+1
33      FIN_TANT_QUE
34
35    //Affichage du contenu de la liste
36    POUR i ALLANT_DE 1 A index-1
37      DEBUT_POUR
38      AFFICHER listeDiviseurs[i]
39      AFFICHER ", "
40      FIN_POUR
41
42  FIN_ALGORITHME
```

Implémenter cet algorithme dans Scratch

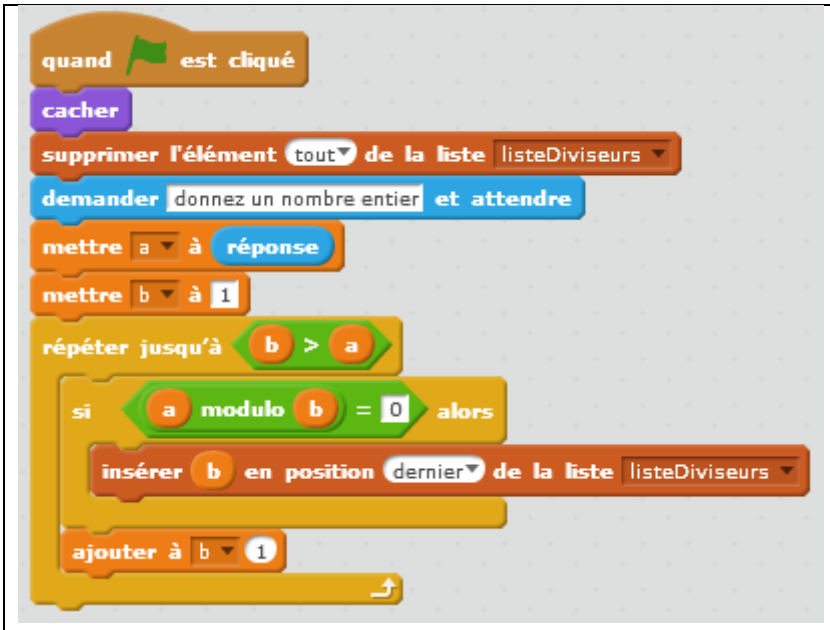
Avec Scratch nous n'avons besoin que de deux variables « a » et « b » et de la liste « listeDiviseurs ».

Pour connaitre tous les diviseurs d'un nombre, il suffit d'afficher la liste. Elle apparait sur la scène.

Au début du script, nous effaçons tout le contenu de la liste, pour pouvoir relancer plusieurs fois le programme, et calculer les deviseurs d'un autre nombre.

Par rapport à l'algorithme Algobox, nous devons traduire la structure « **Tant que** condition vraie **Faire** » par une structure « **Répéter jusqu'à ce que** condition vraie ».

Il suffit d'inverser la condition pour passer d'une structure à l'autre.



Résultat pour a = 36

Exercice 2 : Un nombre est-il premier ?

Un nombre premier est un nombre qui n'admet que deux diviseurs : 1 et lui-même.

Si vous recherchez les diviseurs de 19, avec le programme Scratch précédent, vous obtiendrez la liste :

Écrire l'algorithme Algobox, qui affiche si un nombre est premier ou non.

Nous partons de l'algorithme précédent : **si un nombre est premier, il n'y aura que deux diviseurs dans la liste**.

Il suffit donc de tester qu'elle est la longueur de la liste, lorsque tous les diviseurs y ont été inscrits, pour savoir si le nombre est premier ou non.

```
VARIABLES
  b EST_DU_TYPE NOMBRE
  a EST_DU_TYPE NOMBRE
  listeDiviseurs EST_DU_TYPE LISTE
  index EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  //Nombre dont on cherche les diviseurs
  LIRE a

  //diviseur potentiel
  b PREND_LA_VALEUR 1

  //index désigne la place où il faut enregistrer le diviseur dans la liste
  index PREND_LA_VALEUR 1

  //On va examiner tous les diviseurs potentiels <= a
  TANT_QUE (b <=a) FAIRE
    DEBUT_TANT_QUE

    //On examine le reste de la division de a par b
    SI ((a%b)==0) ALORS
      DEBUT_SI
      //On a trouvé un diviseur, on le mémorise dans la liste
      listeDiviseurs[index] PREND_LA_VALEUR b

      //index est incrémenté pour dédigner l'emplacement suivant dans la liste
      index PREND_LA_VALEUR index+1
      FIN_SI

    //On passe au diviseur potentiel suivant
    b PREND_LA_VALEUR b+1
    FIN_TANT_QUE

  //Test si le nombre est premier
  SI ((index-1) == 2) ALORS
    DEBUT_SI
    AFFICHER a
    AFFICHER " est un nombre premier"
    FIN_SI
    SINON
      DEBUT_SINON
      AFFICHER a
      AFFICHER " n'est pas un nombre premier"
      FIN_SINON
FIN_ALGORITHME
```

Seule la variable « i » a été supprimée, puisque nous n'en avons plus besoin.

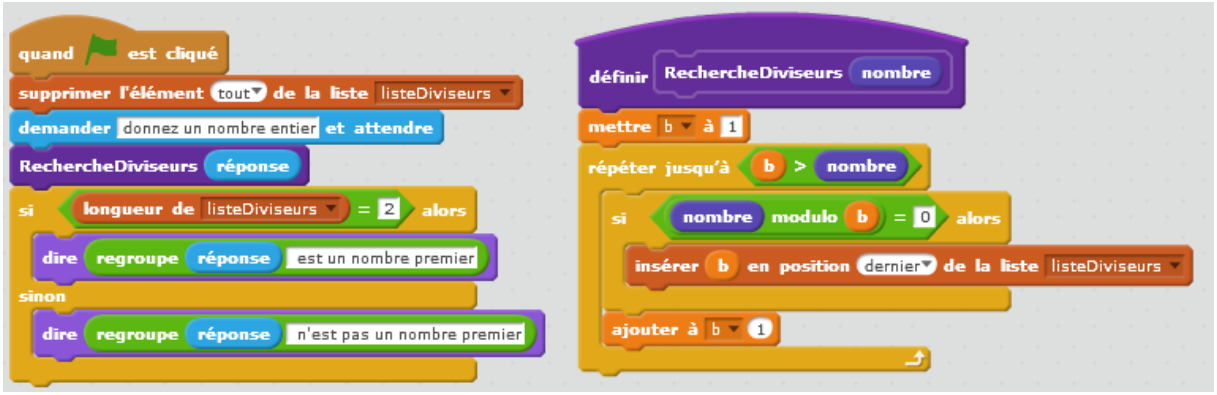
Dans l'algorithme, seul l'affichage de la fin change.

Dans Scratch

Nous plaçons la recherche des diviseurs d'un nombre, dans un bloc « RechercheDiviseurs » qui a un paramètre « nombre ».

Après avoir appelé ce bloc en lui passant le nombre fourni par l'utilisateur, nous testons la longueur de la liste « listeDiviseurs ».

Si cette longueur vaut 2, alors le nombre est premier.



Exercice : le chiffre de Cryptographie de César

Voir la fiche du projet