BYOB Manuel de Référence Version 3.0.7

BYOB is an extension to Scratch (http://scratch.mit.edu) that allows you to Build Your Own Blocks. It also features first class lists and first class procedures (explained below). This document is a very terse explanation of the main features, probably not a good tutorial. It assumes that you are already familiar with Scratch.

BYOB (http://byob.berkeley.edu/)

I. Building a Block

A. Simple blocks

In the Variables palette, at the bottom, is a button labeled “Make a block.” (A palette is one of the eight menus of blocks you can select in the leftmost column of the BYOB window.)

Clicking this button will display a dialog window in which you choose the block’s name, shape, and palette/color. You also decide whether the block will be available to all lutins, or only to the current lutin. Note: You can also enter the “Make a block” dialog by right-click/control-click on the script area background and then choose “Make a block” from the menu that appears.

For the most part, there is one color per palette, e.g., all Motion blocks are blue. But the Variables palette includes the orange variable-related blocks and the red list-related blocks. Both colors are available, along with an “Other” option that makes grey blocks in the Variables palette for blocks that don’t fit any category.

http://scratch.mit.edu/forums/viewtopic.php?id=34284&

BYOB est une extension de Scratch (http://scratch.mit.edu) qui vous permet de créer vos propres blocs. Il comporte également des listes de première classe et des procédures de première classe(explications ci-dessous). Ce document est une explication très succincte des principales caractéristiques, probablement pas un bon tutoriel. Il suppose que vous êtes déjà familier avec Scratch.

I. Construire un Bloc

A. Des Blocs simples

Dans la palette des Variables, tout en bas, il y a un bouton «Créer un bloc» *(Une palette est l'un des huit menus associés aux 8 catégories différentes de blocs que vous pouvez sélectionner dans la colonne de gauche de la fenêtre BYOB.)*

Cliquer sur ce bouton vous permet d'afficher une fenêtre de dialogue dans

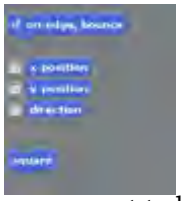
laquelle vous choisissez le nom du bloc, sa forme et sa couleur. Vous pouvez également décider si le bloc sera disponible pour tous les lutins, ou seulement à l'image-objet en cours. Note: vous pouvez également accéder à la fenêtre de dialogue «Créer un bloc» en faisant un clic-droit/contrôle-clic dans l’aire des scripts, puis choisir "Créer un bloc» dans le menu qui s'affiche.

Dans la plupart des cas , une couleur particulière est associée à chaque palette ; par exemple, tous les blocs de la catégorie Mouvement sont bleu. Mais la palette Variables comprend des blocs de couleur orange associés à la variable et des blocs de couleur rouge associés aux listes. Les deux couleurs sont disponibles, ainsi que pour une option «Autre». Cette option crée des blocs de couleur grise dans la palette Variables : blocs qui ne correspondent à aucune des 8 catégories .

There are three block shapes, following a convention that should be familiar to Scratch users: The jigsaw-puzzle- piece shaped blocks are Commands, and don't report a value. The oval blocks are Reporters, and the hexagonal blocks are Predicates, which is the technical term for reporters that report Boolean (true or false) values.

Suppose you want to make a block named "square" that draws a square. You would choose Motion, Command, and type the word "square" into the name field. When you click OK, you enter the Block Editor. This works just like making a script in the lutin's scripting area, except that the "hat" block at the top, instead of saying something like "when Lutin1 clicked," has a picture of the block you're building. This hat block is called the prototype of your custom block. You drag blocks under the hat to program your custom block:

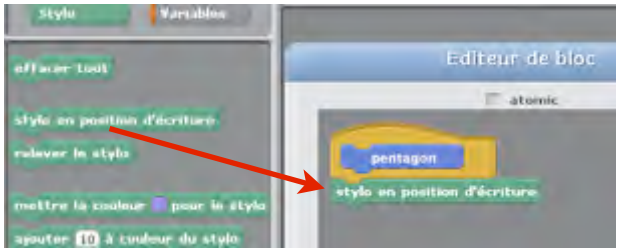
Your block appears at the bottom of the Motion palette. Here's the block and the result of using it:



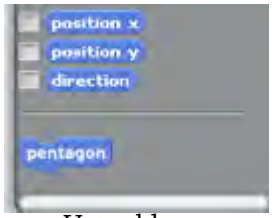
But suppose you want to be able to draw squares of different sizes. Control-click or right-click on the block, choose "edit," and the Block Editor will open. If you hover the mouse over the word "square" in the prototype in the hat block, you'll see two circle-plus signs appear next to it.

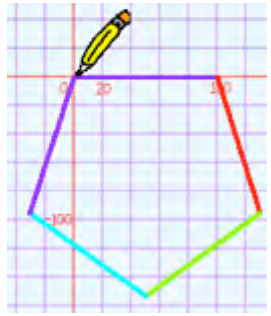
Il existe trois formes de blocs ; c'est une convention familière pour les utilisateurs de Scratch. Les blocs en forme de pièce de puzzle sont des blocs de commande, et ne renvoient pas de valeur. Les blocs en forme d' «ovale» sont dits blocs à valeurs (*qui renvoient une valeur*), et les blocs en forme d' «hexagone» sont des prédicats : c'est le terme technique utilisé pour les blocs à valeurs qui renvoient les valeurs : vrai (1)ou faux (0).

Supposons que vous vouliez créer un bloc "pentagon" qui dessine un pentagone. Vous devriez choisir un bloc de commande Mouvement .Après avoir écrit le mot « pentagon» dans le champ Nom, vous allez cliquer sur OK, et l'éditeur de blocs s'ouvre. *(Cela fonctionne exactement comme pour faire un script dans l'aire des scripts d'un lutin)*. La différence, c'est que le bloc "chapeau" en haut, au lieu d'exécuter l'instruction «quand Lutin1 cliqué»", contient une image du bloc que vous venez de construire. Ce bloc chapeau est appelé le ***prototype*** de votre bloc personnalisé. Vous pouvez faire glisser



des blocs sous le bloc chapeau pour programmer votre bloc personnalisé:





Votre bloc apparaît au bas de la palette : Mouvement. Voici le bloc et le résultat de l'exécution du script:

Mais supposons que vous vouliez être en mesure de dessiner des pentagones de différentes tailles. Clic-droit/ contrôle-clic sur le bloc, choisissez éditer, et l'éditeur de blocs s'ouvre.

Si vous passez la souris sur le mot «pentagon» dans le prototype du bloc de chapeau, vous verrez apparaître deux cercle-plus de chaque côté du nom.

Click on the circle-plus on the right. You will then see the “input name” dialog:

Type in the name “size” and click OK. There are other options in this dialog; you can choose “title text” if you want to add words to the block name, so it can have text after an input slot, like the “move ( ) steps” block. Or you can select a more extensive dialog with a lot of options about your input name. But we’ll leave that for later. When you click OK, the new input appears in the block prototype:

You can now drag the orange variable down into the script, then click okay:

En cliquant sur le cercle-plus sur la droite, vous allez ouvrir la boîte de dialogue "Créer un nom d’entrée":

Entrez le nom «taille» dans le champ disponible et cliquez sur OK. Il y a d'autres options dans cette boîte de dialogue. Vous pouvez choisir "Texte du titre" si vous voulez ajouter des mots au nom du bloc, de sorte qu'il peut avoir du texte après une zone de saisie circulaire, comme pour le bloc de commande mouvement «avancer de () pas». Vous pouvez, aussi, sélectionner une boîte de dialogue plus large avec beaucoup d'options pour votre argument en cliquant sur la flèche noire. Mais nous allons laisser cela de côté pour plus tard.

Lorsque vous cliquez sur OK, le nouvel argument «taille» apparaît dans le

prototype de bloc.Vous pouvez maintenant faire glisser la variable d'orange vers le bas dans le script, puis cliquez sur ok:

Votre bloc apparaît maintenant dans la palette Mouvement avec une zone de saisie rectangulaire:

Your block now appears in the Motion palette with an input box:

You can draw any size square by entering the length of its side in the box and running the block as usual, by double-clicking it or by putting it in a script.

At the top of the Block Editor window is a check box labeled atomic. When this box is checked, the block's entire script is carried out in a single BYOB execution cycle. (As in Scratch, the way multiple scripts are able to operate more or less in parallel is that each script gets to carry out one step (one command, more or less) and then each script gets another turn.) The advantage of atomicity is that the script runs faster. The disadvantage is that if the script draws a picture, or does a dance, the user won't see the individual steps happen but will instead see a longish pause followed by the appearance of the complete picture at once, or the last step of the dance. (You can make atomic custom blocks run non-atomically temporarily by holding down the ESC key.) The default is that the box is checked for reporter blocks, but not for command blocks.

B. Recursion

Since the new custom block appears in its palette as soon as you start editing it, you can write recursive blocks (blocks that call themselves) by dragging the block into its own definition:

If recursion is new to you, here are a few brief hints: It's crucial that the recursion have a base case, that is, some small(est) case that the block can handle without using recursion. In this case, it's the case depth=0, for which the block does nothing at all, because of the enclosing if. Without a base case, the recursion would run forever, calling itself over and over. Don't try to trace the exact sequence of steps that the computer follows in a recursive program. Instead, imagine that inside the computer there are many small people, and if Theresa is drawing a tree of size 100, depth 6, she hires Tom to make a tree of size 70, depth 5, and later hires Theo to make a tree of size 60, depth 5. Tom in turn hires Tammy and Tallulah, and so on. Each little person has his or her own local variables size and depth, each with different values.

4

Vous pouvez dessiner un carré de toute taille en entrant la longueur de son côté dans la zone de saisie et utiliser le bloc comme d'habitude, en double-cliquant dessus ou en le mettant dans un script.

En haut et au milieu de la fenêtre Editeur de bloc se trouve une case à cocher correspondant à la fonction appelée : **atomic**. Lorsque cette case est cochée, la totalité du script du bloc est réalisée dans un seul cycle d' exécution par BYOB. (Comme dans Scratch, la seule façon pour que plusieurs scripts soient en mesure de fonctionner de manière plus ou moins en parallèle, est que chaque script (une seule commande, plus ou moins) soit exécuté l'un après l'autre, chacun son tour . L'avantage de l'atomicité est que le script s'exécute plus rapidement. L'inconvénient est que si le scénario dresse un tableau, ou bien une danse, l'utilisateur ne verra pas les différentes étapes se produire successivement, mais va plutôt voir une pause assez longue avant de voir l'apparition de l'image complète, ou la dernière étape de la danse. (Vous pouvez désactivé temporairement l'atomicité des blocs personnalisés en appuyant sur la touche ESC.)

Par défaut la case est cochée pour les blocs à valeurs (reporters), mais pas pour les blocs de commande.

B. Récursivité

Depuis le nouveau bloc personnalisé apparaît dans sa palette dès que vous commencez à éditer, vous pouvez écrire des blocs récursifs (blocs qui s'appellent eux-mêmes) en faisant glisser le bloc dans le script qui se situe en dessous du prototype, donc dans sa propre définition:

Si la récursivité est nouveau pour vous, voici quelques conseils bref: Il est essentiel que la récursion ont un cas de base, qui est, certains petits (HNE) le cas où le bloc peut gérer sans l'aide de récursion. Dans ce cas, c'est la profondeur = 0, pour

lequel le bloc ne fait rien du tout, en raison de l'enveloppant, si. Sans un cas de base, la récursion irait jamais, se demandant encore et encore. Ne pas essayer de retracer la séquence exacte des étapes que l'ordinateur s'inscrit dans un programme récursif. Au lieu de cela, imaginer que l'intérieur de l'ordinateur il ya beaucoup de petites gens, et si Thérèse est l'élaboration d'un arbre de taille 100, profondeur 6, elle engage Tom pour faire un arbre de taille 70, profondeur 5, et embauche plus tard, Théo pour faire un arbre de taille 60, profondeur 5. Tom à son tour embauche Tammy et Tallulah, et ainsi de suite. Chaque petite personne a son propre ou sa taille variables locales et de la profondeur, chacune avec des valeurs différentes.4

You can also write recursive reporters, like this block to compute the factorial function:

Note the use of the report block. When a reporter block uses this block, the reporter finishes its work and reports the value given; any further blocks in the script are not evaluated. Thus, the if else block in the script above could have been just an if, with the second report block below it instead of inside it, and the result would be the same, because when the first report is seen in the base case, that finishes the block invocation, and the second report is ignored. There is also a stop block block that has a similar purpose, ending the block invocation early, for command blocks. (By contrast, the stop script block stops not only the current block invocation, but the entire toplevel script that called it.)

For more on recursion, see *Thinking Recursively* by Eric Roberts.

II. First Class Lists

A data type is “first class” in a programming language if data of that type can be

the value of a variable an input to a procedure the value returned by a procedure a member of a data aggregate

In Scratch 1.4, numbers and text strings are first class. You can put a number in a variable, use one as the input to a block, write a reporter that reports a number, or put a number into a list.

But Scratch’s lists are not first class. You create one using the “Make a list” button, which requires that you give the list a name. You can’t put the list into a variable, into an input slot of a block, or into a list item—you can’t have lists of lists. None of the Scratch reporters reports a list value. (You can use a reduction of the list into a text string as input to other blocks, but this loses the list structure; the input is just a text string, not a data aggregate.)

A fundamental design principle in BYOB is that all data should be first class. If it’s in the language, then we should be able to use it fully and freely. We believe that this principle avoids the need for many special-case tools, which can instead be written by BYOB users themselves.

5

Vous pouvez également écrire aux journalistes récursive, comme ce bloc de calculer la fonction factorielle:

Notez l'utilisation du bloc rapport. Quand un bloc journaliste utilise ce bloc, le journaliste termine son travail et les rapports de la valeur donnée; tous les blocs plus loin dans le script ne sont pas évalués. Ainsi, le bloc si ailleurs dans le script ci-dessus aurait pu être juste un cas, avec le bloc deuxième rapport en dessous au lieu de l'intérieur, et le résultat serait le même, parce que quand le premier rapport est vu dans le cas de base, qui termine l'invocation bloc, et le deuxième rapport est ignoré. Il ya aussi un bloc de butoir qui a un but similaire, mettant fin à l'appel de bloc au début, pour les blocs de commande. (En revanche, le bloc de script stop arrête non seulement l'invocation bloc courant, mais l'ensemble du script toplevel qui l'a appelé.)

Pour en savoir plus sur la récursivité, voir Pensée récursivement par Eric Roberts.

II. Listes de première classe

Un type de données est «première classe» dans un langage de programmation si les données de ce type peuvent être

la valeur d'une variable d'entrée à une procédure de la valeur retournée par une procédure d'un membre d'un agrégat de données

En Scratch 1.4, nombres et les chaînes de texte sont de première classe. Vous pouvez mettre un nombre dans une variable, utilisez l'une comme l'entrée d'un bloc, un journaliste d'écrire que les rapports d'un certain nombre, ou mettre un numéro dans une liste.

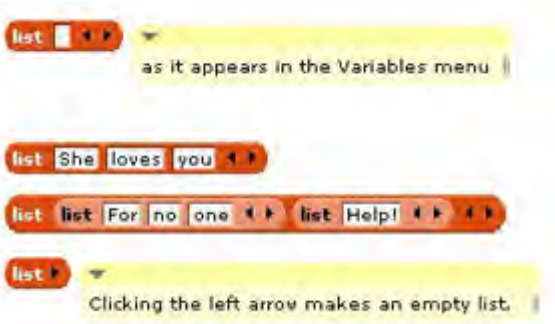
Mais les listes Scratch ne sont pas de première classe. Vous créer un en utilisant le «Faire une liste", qui exige que vous donniez la liste un nom. Vous ne pouvez pas mettre la liste dans une variable, dans une fente d'entrée d'un bloc, ou dans un élément de la liste, vous ne pouvez pas avoir des listes de listes. Aucun des journalistes Scratch fait état d'une valeur de liste. (Vous pouvez utiliser une réduction de la liste en une chaîne de texte comme entrée pour d'autres blocs, mais cela perd la structure de liste;. L'entrée est simplement une chaîne de texte, pas un agrégat de données)

Un principe fondamental de conception dans BYOB est que toutes les données devraient être de première classe. Si c'est dans la langue, alors nous devrions être capable de l'utiliser pleinement et librement. Nous croyons que ce principe évite le besoin d'outils spéciaux de nombreux cas, ce qui peut au contraire être écrits par les utilisateurs eux-mêmes BYOB. 5

A. The list block

At the heart of providing first-class lists is the ability to make an “anonymous” list—to make a list without simultaneously giving it a name. The list reporter block does that.

At the right end of the block are two left-and-right arrowheads. Clicking on these



changes the number of inputs to list, i.e., the number of elements in the list you are building.

You can use this block as input to many other blocks:

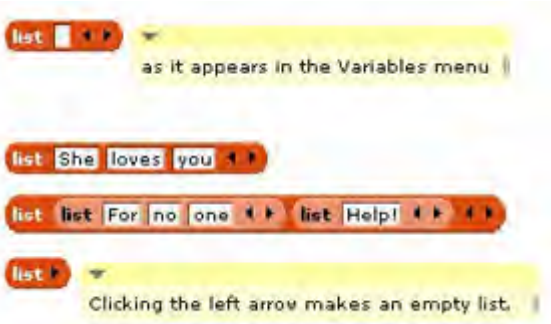
Note that the Scratch list blocks use a pulldown menu with the names of all the “Make a list”-type lists. BYOB retains this notation for compatibility, but you can drag any expression whose value is a list over the pulldown menu:

6

A. Le bloc liste

Au cœur de fournir des listes de première classe est la capacité de faire un "anonyme" liste à dresser une liste sans simultanément en lui donnant un nom. Le bloc de reporter la liste qui le fait.

A l'extrémité droite du bloc de deux pointes de flèches gauche et droite. En



cliquant sur ces changements, le nombre d'entrées à la liste, à savoir, le nombre d'éléments dans la liste que vous générez.

Vous pouvez utiliser ce bloc comme entrée pour de nombreux autres blocs:

Notez que la liste des blocs Scratch utiliser un menu déroulant avec les noms de tous les "Faites une liste des« listes de type. BYOB conserve cette notation pour la compatibilité, mais vous pouvez faire glisser n'importe quelle expression dont la valeur est une liste sur le menu déroulant: 6

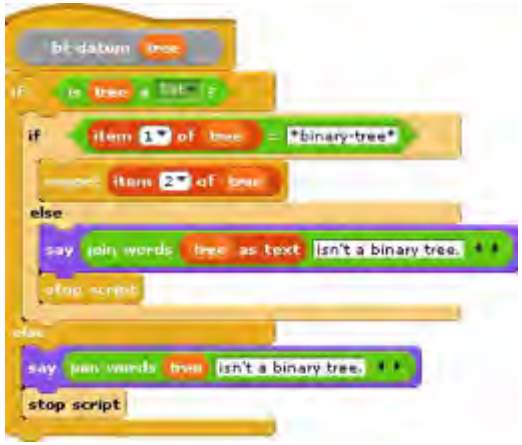
B. Lists of lists

Lists can be inserted as elements in larger lists. We can easily create ad hoc structures as needed:

We can also build any classic computer science data structure out of lists of lists,

by defining constructors, selectors, and mutators as needed. Here we create binary trees with type-checking selectors; only one selector is shown but the ones for left and right children are analogous.

III. Typed inputs



A. Scratch's type notation

Scratch block inputs come in two types: Text-or-number type and Number type. The former is indicated by a rectangular box, the latter by a rounded box:

The BYOB type system is an expanded version to include Procedure and List

types.

7

B. Liste des listes

Les listes peuvent être insérées en tant qu'éléments dans les grandes listes. Nous pouvons facilement créer des structures ad hoc selon les besoins:

Nous pouvons aussi construire une classique de la science informatique de

données sur la structure des listes de listes, la définition par les constructeurs, des sélecteurs et des mutateurs au besoin. Ici, nous créons des arbres binaires avec des sélecteurs de type de vérification, un seul sélecteur est représenté, mais ceux pour les enfants gauche et droit sont analogues.

III. Entrées typées

A. Scratch 's notation de type

entrées du bloc Scratch sont de deux types: le type de texte-ou-et le numéro de type. Le premier est indiqué par une boîte rectangulaire, celui-ci par une boîte arrondie:

Le système de type BYOB est une version élargie pour inclure la procédure et des types de liste.

B. The BYOB Input Type dialog

In the input name dialog, there is a right-facing arrowhead after the “Input name” option:

Clicking that arrowhead opens the “long” input name dialog:

There are eleven input-type shapes, plus three mutually exclusive categories, listed in addition to the basic choice between title text and an input name. The default

type, the one you get if you don’t choose anything else, is “Any,” meaning that this input slot is meant to accept any value of any type.

“Any” versus “Text”: In Scratch, every block that takes a Text-type input has a default value that makes the rectangles for text longer than tall. The blocks that aren’t specifically about text either are of Number type or have no default value, so those rectangles are taller than wide. At first we thought that Text was a separate type that always had a wide input slot; it turns out that this isn’t true in Scratch (delete the default text and the rectangle narrows), but we thought it a good idea anyway, so we allow Text-shaped boxes even for empty input slots.

B. L'entrée BYOB Type de dialogue

Dans la boîte de dialogue nom de l'entrée, il y a une pointe de flèche dirigée vers la droite après le «nom d'entrée" option:

En cliquant sur ce flèche ouvre le "long" de dialogue nom de l'entrée:

Il y a onze formes de type entrée, ainsi que trois catégories mutuellement exclusives, énumérés à l'Outre le choix de base entre le texte du titre et un nom

d'entrée. Le type par défaut, celui que vous obtenez si vous ne choisissez pas autre chose, est "Tout", ce qui signifie que cette fente d'entrée est destiné à accepter n'importe quelle valeur de tout type.

"Tout" et "Texte": Dans Scratch, chaque bloc qui prend une entrée de type texte a une valeur par défaut qui rend les rectangles de texte plus long que haut. Les blocs qui ne sont pas spécifiquement sur le texte sont soit de type Number ou n'ont pas de valeur par défaut, ces rectangles sont plus hautes que larges. Au début, nous avons pensé que ce texte est un type distinct qui a toujours eu une fente d'entrée gamme, il s'avère que ce n'est pas vrai dans Scratch (supprimer le texte par défaut et le rectangle

étroit), mais nous avons pensé ce une bonne idée de toute façon, nous permettons à des zones de texte en forme de même pour les slots d'entrée vide.

List type: The red rectangles inside the input slot are meant to resemble the appearance of lists as Scratch displays them on the stage: each element in a red rectangle. The ability to use lists as inputs is new in BYOB.

Procedure-type inputs: These are also new in BYOB. A “procedure” is a block or a script. (Of course blocks and scripts aren’t new in BYOB; what’s new is the ability to use them as data — as an input to a block.) The slots are grey, not white, to indicate that a procedure is expected as the input value. We’ll say more about these input types later in the manual. There are three procedure types: Command (the jigsaw-puzzle-piece shaped blocks), Reporter (the oval-shaped blocks), and Predicate (the hexagonal blocks). For Command-type inputs there is the additional choice of an input slot on the title line or a “C-shaped” block in which the input goes inside the C.

Boolean vs. Predicate types: “Boolean” is the name for a true or false value. It’s not so different from “Number” as a type that only accepts certain values, except that Scratch provides no way for a user to enter a Boolean value directly into a block, the way you can type a number into a Number-type input slot. (BYOB provides two predicates named true and false that output constant Boolean values.) “Predicate” is the name for a block that reports a Boolean value.

The “Any (unevaluated)” and “Boolean (unevaluated)” types: See Section IV.D, Special Forms, below.

The “Single input” option: All Scratch inputs are in this category. There is one input slot in the block as it appears in its palette. If a single input is of type Any, Number, or Text, then you can specify a default value that will be shown in that slot in the palette, like the “10” in the move 10 steps block. In the prototype block at the top of the script in the Block editor, an input with name “size” and default value 10 looks like this:

The “Multiple inputs” option: The list block introduced earlier accepts any number of elements for the new input. To allow this, BYOB introduces the arrowhead

notation ([¥]and [¥]) that expands and contracts the block, adding and removing input slots. Custom blocks made by the BYOB user have that capability, too. If you choose the “Multiple inputs” button, then arrowheads will appear after the input slot in the block. More or fewer slots (as few as zero) may be used. When the block runs, all of the values in all of the slots for this input name are collected into a list, and the value of the input as seen inside the script is that list of values:

The ellipsis (...) in the orange input slot name box in the prototype indicates a Multiple input.

Le type de liste: Les rectangles rouges à l'intérieur de la fente d'entrée sont conçus pour ressembler à l'apparition de listes Scratch les affiche sur la scène: chaque élément dans un rectangle rouge. La possibilité d'utiliser des listes d'intrants qui est nouveau dans BYOB.

entrées procédure-type: Ce sont aussi de nouveaux BYOB. Une «procédure» est un bloc ou un script. (Des blocs de cours et les scripts ne sont pas nouveaux dans BYOB;. Ce qui est nouveau, c'est la capacité de les utiliser comme des données - en tant que contribution à un bloc) Les fentes sont gris et non blanc, pour indiquer qu'une procédure est prévue comme entrée valeur. Nous dirons plus sur ces types d'entrées plus tard dans le manuel. Il existe trois types de procédures: Command (le puzzle-puzzle-pièce en forme de blocs), Rapporteur (les blocs de forme ovale), et du prédicat (les blocs hexagonaux). Pour les entrées de commande de type il ya le choix supplémentaire d'une fente d'entrée sur la ligne de titre ou d'un "C-forme de" bloc dans lequel l'entrée est à l'intérieur du C.

Boolean vs types de prédicat: "Boolean" est le nom d'une valeur vraie ou fausse. Ce n'est pas si différent de "Number" comme un type qui accepte que certaines valeurs, sauf que Scratch ne fournit aucun moyen pour un utilisateur d'entrer une valeur booléenne directement dans un bloc, la façon dont vous pouvez taper un nombre dans une fente d'entrée Nombre de type . (BYOB fournit deux prédicats vrai et le faux nom que la production constante de valeurs booléennes.) "Principale" est le nom d'un bloc qui présente une valeur booléenne.

Le "Tout (non évaluée)" et "Boolean (non évaluée)" types: Voir la section IV.D, des formes particulières, ci-dessous.

Le "entrée unique" option: Toutes les entrées sont Scratch dans cette catégorie. Il ya une fente d'entrée dans le bloc tel qu'il apparaît dans sa palette. Si une seule entrée est de type un nombre quelconque, ou du texte, vous pouvez spécifier une valeur par défaut qui sera affiché à cet emplacement dans la palette, comme le "10" dans le déplacement de bloc 10 étapes. Dans le bloc de prototype au début du script dans

l'éditeur de bloc, une entrée avec le nom de «taille» et la valeur 10 par défaut ressemble à ceci:

Les "entrées multiples" option: Le bloc de liste introduit plus tôt accepte n'importe quel nombre d'éléments pour l'entrée de nouveaux. Pour permettre cela, BYOB introduit la notation fléchée ([¥] [¥] et) qui dilate et se contracte le bloc, ajouter et supprimer des slots d'entrée. Blocs personnalisés par l'utilisateur BYOB ont cette capacité, aussi. Si vous choisissez l'option "entrées multiples", puis des pointes de flèche apparaîtra après la fente d'entrée dans le bloc. fentes Plus ou moins (aussi peu que zéro) peut être utilisé. Lorsque le bloc fonctionne, toutes les valeurs dans tous les créneaux horaires pour ce nom de l'entrée sont rassemblées dans une liste, et la valeur

de l'entrée comme on le voit à l'intérieur du script, c'est que la liste des valeurs:

Les points de suspension (...) dans la zone de saisie du nom d'orange fente dans le prototype indique une entrée multiples.

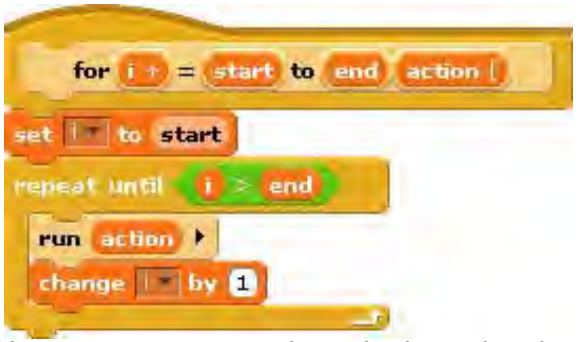
The third category, “Make internal variable visible,” isn’t really an input at all, but rather a sort of output from the block to its user. An upward-pointing arrow indicates this kind of input name in the prototype:

The variable i can be dragged from the for block into the blocks used in its command slot. Also, by clicking on the orange i, the user can change the name of the variable as seen in the script (although the name hasn’t changed inside the block’s definition).

IV.Procedures as Data

A. Procedure input types

In the for block example above, the input named action has been declared as type “Command (C-shaped)”; that’s why the finished block is C-shaped. But how does the block actually tell BYOB to carry out the commands inside the C-slot? Here is a simple version of the block script:



This is simplified because it assumes, without checking, that the ending value is greater than the starting value; if not, the block should (depending on the designer’s purposes) either not run at all, or change the variable by –1 instead of by 1.

The important part of this script is the run block near the end. This is a BYOB built-in block that takes a Command-type value as its input, and carries out its instructions. There is a similar call block for invoking a Reporter block. The call and run blocks are at the heart of BYOB’s first class procedure feature; they allow scripts and blocks to be used as data — in this example, as an input to a block — and eventually carried out under control of the user’s program.

10

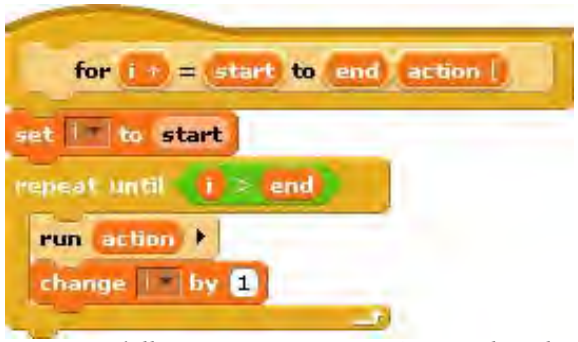
La troisième catégorie, «Faire variable interne visible», n'est pas vraiment une entrée à tous, mais plutôt une sorte de sortie du bloc à son utilisateur. Une flèche pointant vers le haut indique que ce type de nom de l'entrée dans le prototype:

La variable i peut être glissés depuis la pour le bloc dans les blocs utilisés dans la fente de commande. De plus, en cliquant sur le i orange, l'utilisateur peut changer le nom de la variable comme on le voit dans le script (bien que le nom n'a pas changé à l'intérieur de la définition du bloc).

IV.Procédures en tant que données d'entrée de

A.procédure types A.

Dans l'exemple ci-dessus le bloc, l'action d'entrée nommé a été déclaré comme type "Command (en forme de C)", c'est pourquoi le bloc fini est en forme de C. Mais comment le bloc de dire réellement BYOB pour effectuer les commandes à l'intérieur de la rainure en C? Voici une version simple de le bloc de script:



Ceci est simplifiée parce qu'elle suppose, sans vérifier que la valeur de fin est supérieure à la valeur de départ, sinon, le bloc doit (selon les besoins du concepteur) soit de ne pas fonctionner du tout, ou modifier la variable de -1 au lieu de 1.

La partie importante de ce script est le bloc de passer près de la fin. Il s'agit d'un BYOB intégré dans le bloc qui prend une valeur de commande de type en entrée, et s'acquitte de ses instructions. Il ya un blocage d'appels similaires pour invoquer un bloc Reporter. Les blocs d'appel et d'exécution sont au cœur de BYOB de fonctionnalité de classe première procédure, ils permettent des scripts et des blocs à utiliser que des données - dans cet exemple, comme une entrée à un bloc - et finalement réalisé sous le contrôle du programme de l'utilisateur.

10

Here’s another example, this time using a Reporter-type input:

Here we are calling the Reporter “multiply by 10” three times, once with each

element of the given list as its input, and collecting the results as a list. (The reported list will always be the same length as the input list.) Note that the multiplication block has two inputs, but here we have specified a particular value for one of them (10), so the call block knows to use the input value given to it just to fill the other (empty) input slot in the multiply block.

The call block (and also the run block) has a right arrowhead at the end; clicking on it adds the phrase “with inputs” and then a slot into which an input can be inserted:

If the left arrowhead is used to remove the last input slot, the “with inputs” disappears also. The right arrowhead can be clicked as many times as needed for the number of inputs required by the reporter block being called. (You’ll have noticed that “with inputs” is presented in a way that suggests there’s an alternative, but we’ll come back to that later.)

If the number of inputs given to call (not counting the Reporter-type input that

comes first) is the same as the number of empty input slots, then the empty slots are filled from left to right with the given input values. If call is given exactly one input, then every empty input slot of the called block is filled with the same value:

Voici un autre exemple, en utilisant cette fois une entrée Reporter-type:

Ici, nous appelons le Reporter "multiplier par 10" trois fois, une fois avec chaque

élément de la liste donnée en entrée, et la collecte des résultats sous forme de liste. (La liste rapporté sera toujours la même longueur que la liste d'entrée.) Notez que le bloc de multiplication a deux entrées, mais ici nous avons spécifié une valeur particulière pour l'un d'eux (10), de sorte que le bloc d'appel connaît d'utiliser l'entrée valeur donnée pour la juste pour remplir la fente d'entrée (vide) dans le bloc de multiplier.

Le bloc d'appel (et aussi le bloc de course) a une pointe de flèche à droite à la fin; cliquant dessus ajoute le membre de phrase "avec des entrées", puis une fente dans laquelle une entrée peut être insérée:

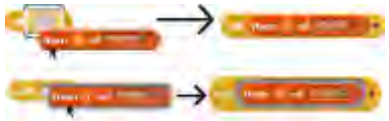
Si la flèche de gauche est utilisé pour enlever la fente d'entrée dernier, le "avec des entrées" disparaît aussi. La flèche droite peut être cliqué autant de fois que nécessaire pour le nombre d'entrées requises par le bloc rapporteur étant appelé. (Vous aurez remarqué que "avec des entrées" est présentée d'une manière qui suggère qu'il ya une alternative, mais nous y reviendrons plus tard.)

Si le nombre d'entrées donnée à appeler (sans compter l'entrée Reporter-type qui vient en premier) est le même que le nombre de slots d'entrée vide, alors les

emplacements vides sont remplis de gauche à droite avec les valeurs d'entrée. Si l'appel est donné exactement une entrée, puis tous les fente d'entrée à vide du bloc appelé est rempli avec la même valeur:

An even more important thing to notice about these examples is the thick grey border around the blocks that we've put in the Reporter-type input slots to call and map above. This notation, not seen in Scratch, indicates that the block itself, not the number or other value that the block would report when called, is the input. By contrast, inside the for and map scripts, we actually want to evaluate a block, the orange one that reports the procedure's input value:

How do we distinguish these two cases as we're building scripts? There's a long way and a short way. The long way, to be described below, uses the the block and the script blocks. The short way uses the distance from the input slot to control whether a Procedure-type input slot's actual input block is evaluated, or taken as the input without evaluation. As you drag a block toward a Procedure-type input slot, you see either the usual Scratch white halo around the input slot, or a white horizontal line inside the input slot and a grey halo around the dragged block, as appropriate.



B. Writing Higher Order Procedures

A "higher order procedure" is one that takes another procedure as an input, or that reports a procedure. In this document, the word "procedure" encompasses scripts, individual blocks, and nested reporters. (Unless specified otherwise, "reporter" includes predicates. When the word is capitalized inside a sentence, it means specifically oval-shaped blocks. So, "nested reporters" includes predicates, but "a Reporter-type input" doesn't.)

Although an Any-type input slot (what you get if you use the small input-name dialog box) will accept a procedure input, it won't do the automatic grey-border input technique described above. So the declaration of Procedure- type inputs makes the use of your custom block much more convenient.

Why would you want a block to take a procedure as input? This is actually not an obscure thing to do; the Scratch conditional and looping blocks (the C-shaped ones in the Control palette) are taking a script as input. Scratch users just don't usually think about it in those terms! We could write the repeat block as a custom block this way, if Scratch didn't already have one:

Une chose encore plus importante à noter à propos de ces exemples est la bordure grise épaisse autour des blocs que nous avons mis dans les fentes d'entrée Reporter-type d'appel et la carte ci-dessus. Cette notation, pas vu dans Scratch, indique que le bloc lui-même, pas le nombre ou la valeur d'autres que le bloc ferait rapport lorsqu'il est appelé, est l'entrée. En revanche, l'intérieur de la carte et scripts, nous voulons vraiment évaluer un bloc, l'orange que les rapports valeur d'entrée de la procédure:

Comment peut-on distinguer ces deux cas que nous construisons scripts? Il ya un

long chemin et un chemin court. Le long chemin, qui sera décrite ci-dessous, utilise le bloc et les blocs de script. Le chemin court utilise la distance entre la fente d'entrée pour contrôler si bloquer un slot d'entrée de procédure-type de saisie réelle est évaluée, ou considérée comme l'entrée sans évaluation. Lorsque vous faites glisser un bloc vers un slot d'entrée de procédure-type, vous voyez le halo blanc autour d'habitude Scratch la fente d'entrée, ou une ligne blanche horizontale à l'intérieur de la fente d'entrée et un halo gris autour du bloc traîné, le cas échéant.

B. Rédaction des procédures ordre supérieur

Une "procédure d'ordre supérieur" est celui qui prend une autre procédure comme

une entrée, ou que les rapports d'une procédure. Dans ce document, le mot «procédure» comprend les scripts, des blocs individuels, et les journalistes imbriqués. (Sauf indication contraire, "reporter" comprend des prédicats. Lorsque le mot est en majuscule dans une phrase, cela veut dire précisément blocs de forme ovale. Ainsi, "les journalistes imbriqués» comprend des prédicats, mais "une entrée Reporter de type" ne fonctionne pas.)

Même si une fente d'entrée Tout type-(ce que vous obtenez si vous utilisez la petite boîte de dialogue d'entrée-nom) accepter une entrée de la procédure, il ne fera pas la technique d'alimentation automatique gris-frontière décrite ci-dessus. Ainsi, la déclaration des entrées de type procédure rend l'utilisation de votre bloc personnalisé beaucoup plus pratique.

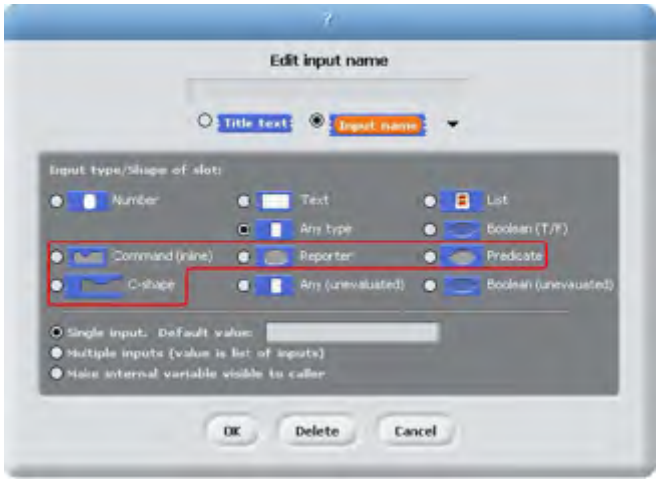
Pourquoi voudriez-vous un bloc de prendre une procédure en entrée? Il s'agit en fait pas une chose obscure à faire, le Scratch avec sursis et des blocs de boucle (ceux en forme de C dans la palette Contrôle) prennent un script comme entrée. Scratch utilisateurs n'ont tout simplement pas souvent penser en ces termes! Nous pourrions écrire le bloc de répéter comme un bloc personnalisé cette façon, si Scratch n'a pas déjà un:

12

The bracket [next to action in the prototype indicates that this is a C-shaped block, and that the script enclosed by the C is the input named action in the body of the script. The only way to make sense of the variable action is to understand that its value is a script.

To declare an input to be Procedure-type, open the input name dialog as usual, click on the arrow:

Then, in the long dialog, choose the appropriate Procedure type. There are two choices for Command inputs; in addition to the usual C-shaped script slot, it is possible to include a jigsaw-piece shaped input inline with the title text. The difference is entirely one of formatting; in either case the input name is associated with a script that can be used as input to run. Indeed, if a variable, an item <#> of <list> block, or a custom Reporter block is dropped onto a C-shaped slot, it turns into an inline slot, as in the repeater block's recursive call above. (Other built-in Reporters can't report scripts, so they aren't accepted in a C-shaped slot.)



Le support [à côté de l'action dans le prototype indique qu'il s'agit d'un bloc en forme de C, et que le script ci-joint par le C est l'entrée nommée action dans le corps du script. La seule façon de donner un sens à l'action variable est de comprendre que sa valeur est un script.

Pour déclarer une entrée à la procédure-type, ouvrez la boîte de dialogue nom de l'entrée comme d'habitude, cliquez sur la flèche:

Puis, dans la boîte de dialogue de long, choisir le type approprié de la procédure. Il ya deux choix pour les entrées de commande, en plus de la fente en forme de C script d'habitude, il est possible d'inclure un puzzle-pièce en forme d'entrée en ligne avec le texte du titre. La différence est tout à fait une de mise en forme; dans les deux cas le nom de l'entrée est associée à un script qui peut être utilisé comme entrée pour fonctionner. En effet, si une variable, un <#> élément du bloc <list>, ou un bloc personnalisé Reporter est tombé sur une fente en forme de C, il se transforme en une fente en ligne, comme dans l'appel récursif bloc répéteur ci-dessus. (Autres intégré Reporters ne peut pas signaler scripts, de sorte qu'ils ne sont pas acceptés dans une fente en forme de C.)

Why would you ever choose an inline Command slot rather than a C shape? Other than the run block discussed below, the only case I can think of is something like the C/C++/Java for loop, which actually has three command script inputs, only one of which is the “featured” loop body:

Okay, now that we have procedures as inputs to our blocks, how do we use them? We use the blocks run (for commands) and call (for reporters). The run block’s script input is inline, not C-shaped, because we anticipate that it will be rare to use a specific, literal script as the input. Instead, the input will generally be a variable whose value is a script.

The run and call blocks have arrowheads at the end that can be used to open slots for inputs to the called procedures. How does BYOB know where to use those inputs? If the called procedure (block or script) has empty input slots, BYOB “does the right thing.” This has several possible meanings:

1. If the number of empty slots is exactly equal to the number of inputs provided, then BYOB fills the empty slots from left to right:

2. If exactly one input is provided, BYOB will fill any number of empty slots with it:

3. Otherwise, BYOB won’t fill any slots, because the user’s intention is unclear.

If the user wants to override these rules, the solution is to use the block or the script with explicit input names that can be put into the given block or script to indicate how inputs are to be used. This will be discussed more fully below.

14

Pourquoi voudriez-vous choisir un emplacement en ligne de commande plutôt qu'une forme C? Autres que le bloc exécuter le verra plus loin, le seul cas que je peux penser est quelque chose comme le C / C / Java pour la boucle, qui a fait trois entrées script de commande, dont une seule est le corps "vedette" de boucle:

Bon, maintenant que nous avons des procédures comme intrants dans nos blocs, comment pouvons-nous les utiliser? Nous utilisons le terme blocs (pour les commandes) et appel (pour les journalistes). Le bloc d'entrée script exécuter est en ligne, pas en forme de C, parce que nous prévoyons qu'elle sera rare d'utiliser un spécifique, littéral de script comme entrée. Au lieu de cela, l'entrée sera généralement une variable dont la valeur est un script.

Les blocs de courir et d'appel ont des pointes de flèches à la fin qui peut être utilisé pour ouvrir des fentes pour les entrées aux procédures appelées. Comment savoir où BYOB d'utiliser ces entrées? Si la procédure appelée (bloc ou script) présente des fentes d'entrée vide, BYOB "fait la bonne chose." Cela a plusieurs significations possibles:

1. Si le nombre de logements vides est exactement égal au nombre d'entrées prévues, puis BYOB remplit les espaces vides de gauche à droite:

2. Si une entrée est exactement prévu, BYOB remplira n'importe quel nombre de cases vides avec elle:

3. Sinon, BYOB ne sera pas remplir les slots, parce que l'intention de l'utilisateur n'est pas claire.

Si l'utilisateur veut modifier ces règles, la solution est d'utiliser le bloc ou le script d'entrée avec un nom explicite qui peut être mis dans le bloc ou l'alphabet pour indiquer comment les ressources doivent être utilisées. Ce point sera discuté plus en détail ci-dessous.14

The text “with inputs” is in a pulldown menu that has one other choice: “with input list.” This variant is used only when making a recursive call to a block that takes a variable number of inputs:

This block will take any number of numbers as inputs, and will make the lutin grow and shrink accordingly:

The user of this block calls it with any number of individual numbers as inputs. But inside the definition of the block, all of those numbers form a list that has a single input name, numbers. This recursive definition first checks to make sure there are any inputs at all. If so, it processes the first input (item 1 of the list), then it wants to make a recursive call with all but the first number. (All but first of isn’t built into BYOB, but is in the tools project distributed with BYOB.) But sizes doesn’t take a list as input; it takes numbers as input! So these would be wrong:15

Le texte "avec des entrées" est dans un menu déroulant qui a un autre choix: ". Avec liste d'entrée" Cette variante est utilisée uniquement lors d'un appel récursif à un bloc qui prend un nombre variable d'entrées:

Ce bloc aura un certain nombre de numéros d'entrées, et fera le lutin grandir et diminuer en conséquence:

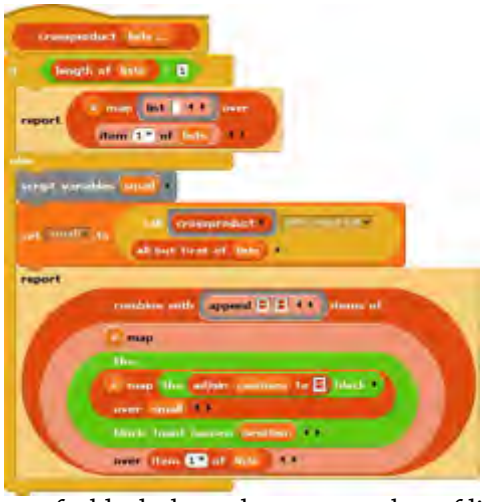
L'utilisateur de ce bloc l'appelle avec un certain nombre de numéros individuels comme intrants. Mais à l'intérieur de la définition du bloc, tous ces chiffres sous forme d'une liste qui a un nom d'entrée unique, numéros. Cette définition récursive vérifie d'abord s'assurer qu'il n'y ait ou non des intrants. Si oui, il traite la première entrée (point 1 de la liste), alors il veut faire un appel récursif avec tous, mais le premier numéro. (Tous les mais d'abord n'est pas incorporé dans BYOB, mais il est dans les outils du projet distribué avec BYOB.) Mais tailles ne prend pas une liste en entrée, il prend le nombre en entrée! Donc, ces serait erroné: 15

The “with input list” option replaces the variable-numbered Any-type input slots with one List-type slot:

The items in the list are taken individually as inputs to the script. Since numbers is a list of numbers, each individual item is a number, just what sizes wants.

C. Procedures as Data

Sometimes the automatic encapsulation of procedures in grey borders as you drag them onto input slots isn’t adequate. This can happen for two reasons: Either you need more control over the use of inputs to an encapsulated procedure, or you want to use a procedure as the input for an input slot that is not declared to be of Procedure type (because other inputs are also meaningful in that context). Those both sound abstract and confusing, so here are two examples. First, the need for finer control over the use of input data:



This is the definition of a block that takes any number of lists, and reports the list of all possible combinations of one item from each list. The important part for this discussion is that near the bottom there are two nested calls to map, a higher order function that applies an input function to each item of an input list. In the inner block, the function being mapped is adjoin, and that block takes two inputs. The second, the empty List-type slot, will get its value in each call from an element of the inner map’s list input. But there is no way for the outer map to communicate values to empty slots of the adjoin block. We must give an explicit name, newitem, to the value that the outer map is giving to the inner one, then drag that variable into the adjoin block.16

La "liste d'entrée avec" option remplace la variable-numérotés slots d'entrée Tout type avec un slot de type liste:

Les éléments de la liste sont pris individuellement comme intrants pour le script. Depuis nombres est une liste de numéros, chaque article est un nombre, ce que veut tailles.

C. Procédures en tant que données

Parfois, l'encapsulation automatique des procédures dans des bordures en gris comme vous les faites glisser sur slots d'entrée n'est pas adéquate. Cela peut se produire pour deux raisons: soit vous avez besoin de plus de contrôle sur l'utilisation d'intrants à une procédure encapsulé, ou si vous souhaitez utiliser une procédure que l'entrée d'une fente d'entrée qui n'est pas déclarée comme étant de type de procédure (en raison d'autres intrants sont aussi un sens dans ce contexte). Ces deux sons abstraits et confus, mais voici deux exemples. Tout d'abord, la nécessité d'un contrôle

plus précis sur l'utilisation des données d'entrée:

Telle est la définition d'un bloc qui prend un certain nombre de listes, et les rapports de la liste de toutes les combinaisons possibles d'un élément de chaque liste. La partie importante de cette analyse est que près du fond, il ya deux appels imbriqués à carte, une fonction d'ordre supérieur qui applique une fonction d'entrée de chaque élément d'une liste d'entrée. Dans le bloc interne, la fonction en cours de mappage est contigu, et ce bloc a deux entrées. La seconde, l'emplacement vide de type liste, obtiendra sa valeur dans chaque appel d'un élément de la liste d'entrée la carte interne de. Mais il n'existe aucun moyen pour la carte externe à communiquer les valeurs d'emplacements vides du bloc contigu. Nous devons donner un nom explicite newitem, à la valeur que la carte externe est de donner à l'intérieure, puis faites glisser la variable dans le bloc contigu.

16

If that example is too confusing, here is a simpler but more contrived one:

Here we just want to put one of the inputs into two different slots. If we left all

three slots empty, BYOB would not fill any of them, because the number of inputs provided (2) would not match the number of empty slots.

By the way, once the called block provides names for its inputs, BYOB will not automatically fill empty slots, on the theory that the user has taken control. In fact, that's another reason you might want to name the inputs explicitly: to stop BYOB from filling a slot that should really remain empty.

Here's an example of the other situation in which a procedure must be explicitly marked as data:

Here, we are making a list of procedures. But the list block accepts inputs of any type, so its input slots are not marked as Procedure type, so there is no option of

dropping the block or script with an automatic grey border. We must say explicitly that we want the block itself as the input, rather than whatever value would result from evaluating the block.

In any of these situations, we use the the block block or the the script block. Both of these mark their input as data; the former takes blocks and the latter takes scripts. Both have right arrows that can be used to supply explicit names for expected inputs. Clicking the arrow exposes an orange oval containing the default name #1 for the first such input, #2 for the second, and so on. These default names can be changed to

something more meaningful by clicking on the orange oval. (Don't drag by accident.) These variables can be dragged into the block or script being encapsulated.

Besides the list block in the example above, other blocks into which you may want to put procedures are set (the value of a variable), say and think (to display a procedure to the user), and report (for a reporter that reports a procedure):17

Si cet exemple est trop confus, voici un simple, mais plus ménagé une:

Ici nous voulons simplement mettre l'une des entrées en deux emplacements

différents. Si nous avons laissé les trois emplacements vides, BYOB ne remplirait pas l'un d'eux, parce que le nombre d'entrées prévues (2) ne serait pas correspondre au nombre de logements vides.

Soit dit en passant, une fois le bloc appelé donne les noms de ses entrées, BYOB ne sera pas automatiquement de remplir les espaces vides, sur la théorie que l'utilisateur a pris le contrôle. En fait, c'est une autre raison pour laquelle vous pouvez décider de renommer les entrées explicitement: pour arrêter BYOB de remplir un créneau qui devrait vraiment rester vide.

Voici un exemple de la situation d'autres dans lesquelles une procédure doit être clairement marqué comme étant des données:

Ici, nous faisons une liste de procédures. Mais le bloc de la liste accepte les entrées de tout type, de sorte que son slots d'entrée ne sont pas marqués en tant que type de procédure, il n'ya donc pas possibilité de faire descendre le bloc ou d'un script avec une

bordure grise automatique. Nous devons dire clairement que nous voulons que le bloc lui-même que l'entrée, plutôt que quelle que soit la valeur résulterait de l'évaluation du bloc.

Dans l'une de ces situations, nous utilisons le bloc le bloc ou le bloc de script du. Ces deux marques d'entrée leur en tant que données, le premier prend blocs et celle-ci prend des scripts. Les deux ont des flèches droit qui peut être utilisé pour fournir des noms explicites pour les contributions attendues. Cliquez sur la flèche expose un ovale orange contenant le nom de # 1 par défaut pour la première entrée par exemple, # 2 pour le second, et ainsi de suite. Ces noms par défaut peut être changé pour quelque

chose de plus significatif en cliquant sur l'ovale orange. (Ne pas traîner par accident.) Ces variables peuvent être glissés dans le bloc de script ou être encapsulé.

Outre la liste de blocage dans l'exemple ci-dessus, d'autres blocs dans lequel vous pouvez mettre ces procédures sont définies (la valeur d'une variable), disent et pensent (à afficher une procédure à l'utilisateur), et le rapport (pour un journaliste que les rapports une procédure): 17

When you are using the block to control the encapsulated block's use of its inputs with variable names, be careful not to put a grey border around the block itself, which would encapsulate the block instead of whatever block you used as input to it:

Although unusual, it is possible that you'd want to make a list of blocks that includes the block. That's why the automatic grey border isn't ruled out.

D. Special Forms

Scratch has an if else block that has two C-shaped command slots and chooses one or the other depending on a Boolean test. Because Scratch doesn't emphasize functional programming, it lacks a corresponding reporter block to choose between two expressions. We could write one:

Our block works for these simple examples, but if we try to use it in writing a recursive operator, it'll fail:

The problem is that when any Scratch block is called, all of its inputs are computed (evaluated) before the block itself runs. The block itself only knows the values of its inputs, not what expressions were used to compute them. In particular, all of the inputs to our if then else block are evaluated first thing. That means that even in the base case, factorial will try to call itself recursively, causing an infinite loop. We need our if then else block to be able to select only one of the two alternatives to be evaluated.

We have a mechanism to allow that: declare the then and else inputs to be of type Reporter rather than type Any. Then, when calling the block, enclose those inputs in the () block so that the expressions themselves, rather than their values, become the inputs:18

Lorsque vous utilisez le bloc de contrôle de l'utilisation du bloc encapsulé de ses entrées avec des noms de variables, faire attention de ne pas mettre une bordure grise autour du bloc lui-même, qui encapsulent le bloc au lieu de tout le bloc que vous avez utilisé comme entrée pour le:

Bien que rare, il est possible que vous voudriez faire une liste de blocs qui comprend le bloc. C'est pourquoi la bordure grise automatique n'est pas exclu.

D. Formes spéciales

Scratch est un bloc if else qui dispose de deux emplacements de commande en forme de C et choisit l'une ou l'autre en fonction d'un test booléen. Parce que Scratch ne met pas l'accent de la programmation fonctionnelle, il lui manque un bloc journaliste correspondant à choisir entre deux expressions. Nous pourrions écrire un:

Notre bloc des œuvres de ces exemples simples, mais si nous essayons de l'utiliser dans la rédaction d'un opérateur récursif, il va échouer:

Le problème est que quand un bloc Scratch est appelé, toutes ses entrées sont calculés (évalué) avant que le bloc lui-même fonctionne. Le bloc lui-même ne connaît que les valeurs de ses entrées, pas ce que les expressions ont été utilisées pour les calculer. En particulier, toutes les entrées à notre première chose que si alors sinon les blocs sont évaluées. Cela signifie que même dans le cas de base, factorielle vais essayer de lui-même de manière récursive, ce qui provoque une boucle infinie. Nous avons besoin de notre if then else bloc pour être en mesure de choisir une seule des deux solutions de rechange à être évalués.

Nous avons un mécanisme pour permettre que: déclarer à l'époque et d'autre des entrées à des Reporter type plutôt que de type Any. Puis, quand l'appel du bloc, mettez ces entrées dans le () bloc de sorte que les expressions elles-mêmes, plutôt que de leurs valeurs, deviennent les entrées: 18

In this version, the program works, with no infinite loop. But we've paid a heavy price: this reporter-if is no longer as intuitively obvious as the Scratch command-if. You have to know about procedures as data, about grey borders, and about constant functions. (The id block implements the identity function, which reports its input. We need it because the block only takes reporters as input, not numbers.) What we'd like is a reporter-if that behaves like this one, delaying the evaluation of its inputs, but looks like our first version, which was easy to use except that it didn't work.

Such blocks are indeed possible. A block that seems to take a simple expression as input, but delays the evaluation of that input by wrapping a hidden the block around it (and, if necessary, an id-like transformation of constant data into constant functions) is called a special form. To turn our if block into a special form, we edit the block's prototype, declaring the inputs yes and no to be of type "Any (unevaluated)" instead of type Reporter. The script for the block is still that of the second version, including the use of call to evaluate either yes or no but not both. But the slots appear as white Any-type rectangles, not Reporter-type ovals, and the factorial block will look like our first attempt.

Special forms trade off implementor sophistication for user sophistication. That is, you have to understand all about procedures as data to make sense of the special form implementation of if then else. But any Scratch programmer can use if then else without thinking at all about how it works internally.

Special forms are actually not a new invention in BYOB. Many of Scratch's conditional and looping blocks are really special forms. The hexagonal input slot in the if block is a straightforward Boolean value, because the value can be computed once, before the if block makes its decision about whether or not to run its action input. But the forever if, repeat until, and wait until blocks' inputs can't be Booleans; they have to be of type "Boolean (unevaluated)," so that Scratch can evaluate them over and over again. Since Scratch doesn't have custom blocks, it can afford to handwave away the distinction between evaluated and unevaluated Booleans, but BYOB can't. The pedagogic value of special forms is proven by the fact that no Scratcher ever notices that there's anything strange about the way in which the hexagonal inputs in the Control blocks are evaluated.19

Dans cette version, le programme fonctionne, sans boucle infinie. Mais nous avons payé un lourd tribut: ce journaliste-si n'est plus aussi intuitivement évident que le Scratch commande si. Vous devez savoir au sujet des procédures que des données, sur les bordures en gris, et sur les fonctions constantes. (Le bloc ID met en œuvre la fonction de l'identité, qui rend compte de ses commentaires. Nous en avons besoin parce que le bloc ne prend que des journalistes comme entrée, pas de chiffres.) Ce que nous souhaitons, c'est un journaliste-qui se comporte comme si celui-ci, ce qui retarde l'évaluation des ses entrées, mais ressemble à notre première version, ce qui était facile à utiliser, sauf que cela n'a pas fonctionné.

Ces blocs sont en effet possibles. Un bloc qui semble prendre une expression simple en entrée, mais des retards de l'évaluation de cette entrée en enroulant une caché le bloc autour de lui (et, si nécessaire, une transformation id-comme des données constantes en fonctions constantes) est appelée une forme spéciale . Pour activer notre si le bloc dans une forme spéciale, nous éditons prototype du bloc, de déclarer les entrées oui et non à être de type "Tout (non évaluée)" au lieu de type Reporter. Le script pour le bloc est toujours celle de la deuxième version, y compris l'utilisation de l'appel d'évaluer ni oui ni non, mais pas les deux. Mais les fentes apparaissent en blanc rectangles Tout type, pas ovales Reporter-type, et le bloc factoriel ressemblera notre première tentative.

Les formes spéciales compromis sophistication réalisateur pour la sophistication de l'utilisateur. C'est, il faut comprendre toutes les données sur les procédures que pour donner un sens à la mise en œuvre sous forme spéciale des if then else. Mais tout programmeur Scratch pouvez utiliser if then else sans penser à tous sur la façon dont il fonctionne.

Des formulaires spéciaux sont en fait pas une nouvelle invention dans BYOB. Un grand nombre de blocs conditionnels et les boucles de Scratch sont des formes très spéciales. La fente d'entrée hexagonale dans le bloc, si est une valeur booléenne simple, parce que la valeur peut être calculé une fois, avant le bloc if ne rende sa décision quant à savoir si ou non d'exécuter son entrée d'action. Mais le toujours si, répéter jusqu'à ce que, et attendre que les entrées des blocs ne peut pas être booléens, ils doivent être de type "Boolean (non évalués)," de sorte que Scratch peut les évaluer encore et encore. Depuis Scratch n'a pas des blocs personnalisés, il peut se permettre de handwave loin la distinction entre les opérations booléennes évaluées et non évaluées, mais ne peut pas BYOB. La valeur pédagogique de formes particulières est prouvé par le fait qu'aucune Scratcher jamais remarque qu'il ya quelque chose d'étrange la façon dont les entrées hexagonale dans les blocs de contrôle sont evaluated.19

IV. Object Oriented Programming

Object oriented programming is a style based around the abstraction “object”: a collection of data and methods (procedures, which from our point of view are just more data) that you interact with by sending it a “message” (just a name, maybe in the form of a text string) to which it responds by carrying out a method, which may or may not report a value back to the asker. Reasons for using OOP vary; some people emphasize the “data hiding” aspect (because each object has local variables that other objects can access only by sending messages to the owning object) while others emphasize the “simulation” aspect (in which each object abstractly represents something in the world, and the interactions of objects in the program model real interactions of real people or things).

A. Lutins as Objects

Scratch comes with things that are basically objects: its lutins. Each lutin can own local variables; each lutin has its own scripts (methods). A Scratch animation is plainly a simulation of the interaction of characters in a play. There are only two ways in which Scratch lutins are less versatile than the objects of an OOP language. First, message passing is primitive in three respects. Messages can only be broadcast, not addressed to an individual lutin; messages can’t take inputs; and methods can’t return values to their caller. Second, an OOP language generally includes the idea of “inheritance,” which means that an object (or a type of objects) can be declared to be identical to some other object except for a few specified differences, and this is not possible for lutins.

BYOB addresses the first of these deficiencies through an extension to the <var> of <lutin> block in the Sensing palette. This block was meant to allow one lutin to find out some numeric attribute of another lutin, such as its x or y position. But if you assign a procedure to a lutin’s local variable, then the of block will report that method, which can be used as input to call or run, whichever is appropriate. This allows direct invocation of one lutin’s methods by another lutin. The method can be given arguments, and if it’s a reporter, it reports a value to the invoking lutin.

As a convenience, BYOB includes a launch block that’s identical to run except that it calls the method as a separate script, so the calling script can meanwhile do something else. Launch can be thought of as an improved broadcast, while run of another lutin’s method is more analogous to broadcast and wait.

Creating lutin methods by saying

is a bit awkward; it’s easier to define methods in the Block Editor. Therefore, BYOB also includes lutin-local (“for this lutin only”) blocks in the pulldown list of local state in the of block.

We have not addressed the question of lutin inheritance. Scratch users have long wanted a clone block that would create a new lutin sharing behavior with an existing lutin. This feature was deliberately left out of Scratch for fear of a “sorcerer’s apprentice” script that would make huge numbers of lutins before the user could stop it. Perhaps a later version of BYOB will provide this capability in some form.20

IV. Programmation Orientée Objet

Programmation orientée objet est un style basé autour de l'abstraction "objet": une collection de données et méthodes (procédures, ce qui de notre point de vue sont simplement plus de données) que vous interagissez avec lui envoyant un "message" (juste un nom, peut-être sous la forme d'une chaîne de texte) à laquelle elle répond en procédant à une méthode, qui peut ou peut ne pas rapporter une valeur à l'demandeur. Raisons d'utiliser la POO varient; certaines personnes l'accent sur la «dissimulation de données" aspect (parce que chaque objet a des variables locales que d'autres objets peuvent accéder que par l'envoi de messages à l'objet propriétaire), tandis que d'autres insistent sur la "simulation" aspect (dans lequel chaque objet abstrait représente quelque chose dans le monde, et les interactions des objets dans les interactions modèle de programme réel de vraies personnes ou des choses).

A. Lutins en tant qu'objets

Scratch est livré avec des choses qui sont essentiellement des objets: ses lutins. Chaque lutin peut propres variables locales; chaque lutin a ses propres scripts (méthodes). Une animation Scratch est tout simplement une simulation de l'interaction des personnages dans une pièce. Il n'ya que deux façons dont les lutins Scratch sont moins polyvalents que les objets d'un langage orienté objet. Tout d'abord, le passage de messages est primitif à trois égards. Les messages peuvent être diffusés, ne s'adresse pas au lutin individuels; messages ne peuvent pas prendre des intrants et les méthodes ne peuvent pas retourner les valeurs à leur appelant. Deuxièmement, un langage OOP comprend généralement l'idée de «l'héritage», ce qui signifie qu'un objet (ou un type d'objets) peut être déclaré identique à un autre objet à l'exception de quelques différences précisées, et ce n'est pas possible pour les lutins .

BYOB répond au premier de ces carences grâce à une extension de la <var> du bloc <lutin> dans la palette de détection. Ce bloc était destiné à permettre à un lutin de trouver quelque attribut numérique d'un autre lutin, tels que sa position x ou y. Mais si vous assignez une procédure à une variable locale lutin, alors le bloc du rapport de cette méthode, qui peut être utilisée comme entrée d'appeler ou d'exécuter, selon le cas. Cela permet à l'invocation directe de méthodes à un lutin par un autre lutin. La méthode peut être donné des arguments, et si c'est un journaliste, il rapporte une valeur à la invoquant lutin.

Par souci de commodité, BYOB comprend un bloc de lancement qui est identique à courir, sauf qu'il appelle la méthode comme un script séparé, de sorte que le script d'appel peut en attendant de faire autre chose. Lancement peut être considéré comme une émission améliorée, tout en terme de méthode d'un autre lutin s'apparente davantage à diffuser et d'attendre.

Création de méthodes de lutin en disant

est un peu maladroit, il est plus facile de définir des méthodes dans l'éditeur de blocs. Par conséquent, BYOB comprend également lutin-local («pour ce lutin seulement") des blocs dans la liste déroulante de l'Etat dans les locaux du bloc.

Nous n'avons pas abordé la question de l'héritage lutin. Scratch utilisateurs ont longtemps voulu un bloc clone qui créerait un comportement lutin nouveau partage avec un lutin existants. Cette fonctionnalité a été délibérément laissés de côté Scratch de peur de «l'apprenti sorcier" un script qui ferait un très grand nombre de lutins avant que l'utilisateur ne pouvait l'arrêter. Peut-être une version ultérieure de BYOB fournira cette capacité dans certains form.20

This script implements an object class, a type of object, namely the counter class. In this first simplified version there is only one method, so no explicit message-passing is necessary. When the make a counter block is called, it reports a procedure, created by the the script block inside its body. This procedure implements a specific counter object, an instance of the counter class. When invoked, a counter instance increases and reports its count variable. Each counter has its own local count:

This example will repay careful study, because it isn't obvious why each instance has a separate count. From the point of view of the make a counter procedure, each invocation causes a new, block-associated count variable to be created. Usually such "block variables" are temporary, going out of existence when the procedure ends. But this one is special, because the procedure returns another procedure that makes reference to the count variable, so it remains active. (The script variables block makes variables local to a script. It can be used in a lutin's script area or in the Block Editor. Script variables can't be dragged outside the script in which they are created, but they can be "exported" indirectly by being used in a reported procedure, as here.)

In this approach to OOP we are representing both classes and instances as procedures. The make a counter block represents the class, while each instance is represented by a nameless script created each time make a counter is called. The script variables created inside the make a counter block but outside the the script block are instance variables, belonging to a particular counter.21

Ce script met en œuvre une classe d'objet, un type d'objet, à savoir la classe counter. Dans cette première version simplifiée n'y a qu'une seule méthode, donc pas explicite de transmission de messages est nécessaire. Lorsque le faire un bloc compteur est appelé, il signale une procédure, créée par le bloc de script à l'intérieur de son corps. Cette procédure met en œuvre un objet spécifique contre, une instance de lutte contre la classe. Lorsqu'elle est invoquée, une instance de compteur augmente et rend compte de ses variable de comptage. Chaque compteur est le nombre de ses propres

locaux.

Cet exemple va rembourser une étude attentive, car il n'est pas évident de comprendre pourquoi chaque instance dispose d'un chef d'accusation distinct. Du point de vue de la faire une procédure contre, chaque invocation provoque une nouvelle variable nombre de blocs associés à créer. Habituellement, ces «variables de bloc" sont temporaires, sortir de l'existence lorsque la procédure se termine. Mais celui-ci est spéciale, parce que la procédure retourne une autre procédure qui fait référence à la variable de comptage, de sorte qu'il reste actif. (Le bloc de script permet variables variables locales à un script. Il peut être utilisé dans la zone de script d'un lutin ou dans l'éditeur de blocs. Variables de script ne peut pas être traîné en dehors de l'écriture dans laquelle ils sont créés, mais ils peuvent être «exportés» indirectement par être utilisé dans une procédure rapporté, comme ici.)

Dans cette approche à la POO nous représentons les deux classes et instances que des procédures. Le faire un bloc compteur représente la classe, tandis que chaque instance est représentée par un script sans nom créé à chaque fois faire un compteur est appelé. Les variables de script créé à l'intérieur du faire un bloc compteur, mais en dehors du bloc de script sont des variables d'instance, appartenant à un particulier counter.21

In the simplified class above, there is only one method, and so there are no messages; you just call the instance to carry out its one method. Here is a more detailed version that uses message passing:

Again, the make a counter block represents the counter class, and again the script creates a local variable count each time it is invoked. The large outer the script block represents an instance. It is a dispatch procedure: it takes a message (just a text word) as input, and it reports a method. The two smaller the script blocks are the methods. The top one is the next method; the bottom one is the reset method. The latter requires an input, named value.

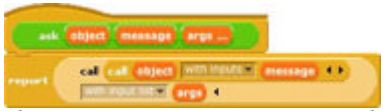
In the earlier version, calling the instance did the entire job. In this version, calling the instance gives access to a method, which must then be called to finish the job. We can provide a block to do both procedure calls in one:

The ask block has two required inputs: an object and a message. It also accepts optional additional inputs, which BYOB puts in a list; that list is named args inside the block. The block has two nested call blocks. The inner one calls the object, i.e., the dispatch procedure. The dispatch procedure always takes exactly one input, namely the message. It reports a method, which may take any number of inputs; note that this is the situation in which we need the “with input list” option of call.22

Dans la classe simplifié ci-dessus, il n'y a qu'une seule méthode, et ainsi il n'ya pas de messages; vous suffit d'appeler l'instance à s'acquitter de sa seule méthode. Voici une version plus détaillée qui utilise le passage de messages:

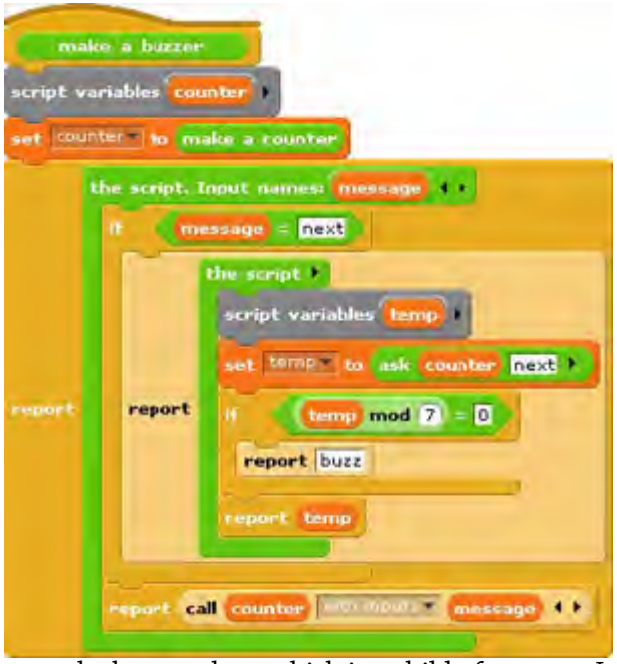
Encore une fois, la marque d'un bloc compteur représente la classe au comptoir, et encore une fois le script crée une variable locale compter chaque fois qu'il est invoqué. Le grand bloc de script externe représente une instance. Il s'agit d'une procédure d'envoi: il faut un message (juste un mot du texte) comme entrée, et il rend compte d'une méthode. Les deux plus petits blocs de script sont les méthodes. Celui du haut est la méthode suivante; celui du bas est la méthode de réinitialisation. Ce dernier nécessite une entrée, valeur nommée.

Dans la version antérieure, appelant l'instance a fait le travail ensemble. Dans cette version, appelant l'instance donne accès à une méthode, qui doit alors être appelé à achever le travail. Nous pouvons fournir un bloc de faire les deux appels de procédure à la fois:



Le bloc de demander a deux entrées nécessaires: un objet et un message. Il accepte également en option des entrées supplémentaires, qui BYOB met dans une liste; cette liste est nommé args l'intérieur du bloc. Le bloc a deux blocs imbriqués appel. Le intérieure appelle l'objet, à savoir, la procédure d'expédition. La procédure de répartition a toujours exactement une entrée, à savoir le message. Il rend compte d'une méthode, qui peut prendre n'importe quel nombre d'entrées; noter que c'est la situation dans laquelle nous avons besoin de "liste d'entrée avec" option de call.22

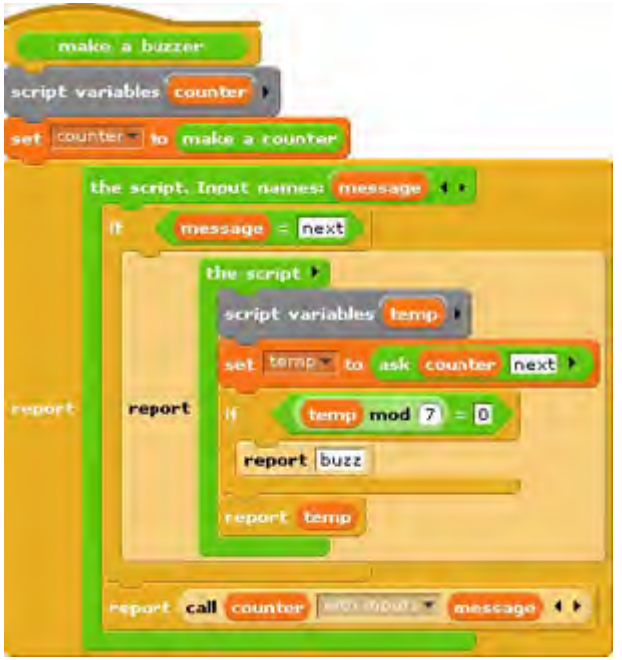
So, our objects now have local state variables and message passing. What about inheritance? We can provide that capability using the technique of delegation. Each instance of the child class contains an instance of the parent class, and simply passes on the messages it doesn't want to specialize:



This script implements the buzzer class, which is a child of counter. Instead of having a count (a number) as a local state variable, each buzzer has a counter (an object) as a local state variable. The class specializes the next method, reporting what the counter reports unless that result is divisible by 7, in which case it reports "buzz." If the message is anything other than next, though, then the buzzer simply invokes its counter's dispatch procedure. So the counter handles any message that the buzzer doesn't handle explicitly.23

Ainsi, nos objets ont maintenant des variables d'état locales et le passage de message. Qu'en est-il l'héritage? Nous pouvons fournir cette capacité en utilisant la technique de la délégation. Chaque instance de la classe des enfants contient une instance de la classe parente, et passe tout simplement sur les messages qu'il ne veut pas se spécialiser:

Ce script implémente la classe buzzer, qui est un enfant du comptoir. Au lieu d'avoir un compte (un nombre) comme une variable d'état local, chaque vibreur a un



compteur (un objet) comme une variable d'état locale. La classe spécialisée de la méthode suivante, information ce que le compteur indique moins que ce résultat est divisible par 7, dans ce cas, il rend compte de «buzz». Si le message est autre chose que la prochaine, si, alors le buzzer invoque simplement son compteur procédure d'envoi. Ainsi, le compteur gère tout message que le buzzer ne gère pas explicitly.23

V. Miscellaneous features

BYOB 3 also introduces a few blocks that aren't about first class lists or procedures:

These blocks report the constant values true and false, getting around the need to use text strings as Boolean values.

This block is used to check the type of any value. The pulldown lists all the available types: number, text, Boolean, list, command, reporter, predicate. Any value has exactly one of these

types.

These blocks convert between single-character text strings (the character can be a letter, digit, or punctuation mark even though the block name says "letter") and numeric codes used

to represent the characters internally (the ASCII code). They're useful, for example, when you want to use a character to index into a list of values; if you have a list containing the words "Alpha," "Bravo," "Charlie," "Delta," and so on, you can turn a letter of the alphabet into the corresponding word by taking the ASCII for that letter minus the ASCII for "A," plus 1.

This block takes a list and reports its value as a text string. It is provided for compatibility with Scratch 1.4 projects; the Scratch list block reports the contents of a list as a single text

string with all the items concatenated. When reading a Scratch project, BYOB replaces any list block in a script with this block.

The copy of block makes a new list containing the same items as the input list, more quickly than copying elements one by one. This block does not make copies of any sublists (lists

that are items of the overall list); the same sublist appears in the original input and in the copy, so changing an item of a sublist in one of these lists affects the other.

The script variables block can be used in any script, whether in a lutin's script area or in the Block Editor. It creates one or more variables that are local to that script; they can be used

as temporary storage during the running of the script, or may become permanent (but still only usable within the body of the script) if the script reports a procedure that makes reference to them. For scripting area blocks, this is a new capability; for the Block Editor it's an alternative to the (deprecated) Make a Variable button inside the Block Editor.

BYOB 3 introduit également quelques pâtés de maisons qui ne sont pas sur les listes de première classe ou de procédures:

Ces blocs rapport les valeurs constantes vrai et le faux, se déplacer sur la nécessité

d'utiliser des chaînes de texte comme des valeurs booléennes.

Ce bloc est utilisé pour vérifier le type de valeur. Le menu déroulant répertorie tous les

types disponibles: nombre, texte, valeur booléenne, liste, commande, journaliste, prédicat. Toute valeur a exactement un de ces types.

Ces blocs de conversion entre les chaînes de texte à caractère unique (le personnage peut

être une lettre, chiffre ou signe de ponctuation, même si le nom du bloc dit «lettre») et les codes numériques utilisés pour représenter les caractères à l'interne (le code ASCII). Ils sont utiles, par exemple, lorsque vous souhaitez utiliser un caractère à l'index dans une liste de valeurs, si vous avez une liste contenant les mots «Alpha», «Bravo», «Charlie», «Delta», et ainsi de suite , vous pouvez transformer une lettre de l'alphabet dans le mot correspondant en prenant l'ASCII pour cette lettre, moins le ASCII pour «A», plus 1.

Ce bloc a une liste et des rapports de sa valeur comme une chaîne de texte. Il est fourni

pour la compatibilité avec Scratch 1.4 Projets; le bloc de la liste des rapports Scratch le contenu d'une liste comme une chaîne de texte unique avec tous les éléments concaténés. Lors de la lecture d'un projet Scratch, BYOB remplace n'importe quel bloc de liste dans un script avec ce bloc.

La copie de bloc permet une nouvelle liste contenant les mêmes éléments que la liste

d'entrée, plus rapidement que copier des éléments un par un. Ce bloc ne pas faire de copies de toute sous-listes (listes qui sont des éléments de la liste globale); même la sous-liste apparaît dans l'entrée d'origine et dans la copie, donc la modification d'un élément d'une sous-liste dans une de ces listes affecte l'autre.

Le bloc de script variables peuvent être utilisées dans n'importe quel script, que ce soit

dans la zone de script d'un lutin ou dans l'éditeur de blocs. Il crée une ou plusieurs variables qui correspondent à ce script, ils peuvent être utilisés comme stockage temporaire pendant le déroulement du script, ou peut devenir permanent (mais toujours utilisable que dans le corps du script) si le scénario fait état d'une procédure qui fait référence à eux. Pour les blocs de zone de script, il s'agit d'une nouvelle capacité, car l'éditeur de blocs c'est une alternative à la (obsolète) Faire un bouton de la variable à l'intérieur de l'éditeur de blocs.24

V. Divers caractéristiques

<var> of <lutin> · 20
A
anonymous list · 6 Any (unevaluated) type · 9, 19 Any type · 8 arrow, upward-pointing · 10 arrowheads · 6, 9 as ascii · 24 as text · 24 ascii code of · 24 atomic · 4
B
binary tree · 7 Block Editor · 2 block, making a · 1 Boolean vs. Predicate types · 9 bracket · 13
C
call · 10, 14 Command type · 9 Command type, C-shaped · 9 Command type, inline · 9 copy of · 24 crossproduct · 16 C-shaped block · 13 C-shaped Command type · 9
D
data structure · 7 data types · 7 default value · 9 delegation · 23 dialog, input name · 3 dispatch procedure · 22
E
empty input slots, filling · 11 ESC key · 4
F
false · 24 first class data type · 5
G
grey border · 12 grey halo · 12
H
higher order procedure · 12
I
if else · 18 inheritance · 23 inline Command type · 9 input name dialog · 3, 8 input-type shapes · 8 internal variable · 10 is a · 24
L
launch · 20 list · 6 List type · 9
M
Make a block · 1 make internal variable visible · 10 map · 11 Multiple inputs · 9
N
Number type · 8
ellipsis · 9

O
object oriented programming · 20 objects, building explicitly · 21
P
palette · 1 Predicate type · 9 Procedure type · 9 prototype · 2
R
recursion · 4 Reporter type · 9 reporters, recursive · 5 run · 10, 14
S
script variables · 24 Single input · 9 special form · 19
Special Form · 18 lutins as objects · 20 square bracket · 13
T
Text type · 8 the block · 14, 17 the script · 14, 17 true · 24
U
upward-pointing arrow · 10
V
variable number of inputs · 15
W
with input list · 16