L'extension pour LaTeX

v 0.11

10 mars 2019

Christian Tellechea
unbonpetit@netc.fr

Cette extension permet de dessiner des empilements de blocs similaires à ceux que l'on trouve dans le logiciel de programmation *visuelle* « scratch v3 [1] ».

1. Le logiciel que l'on peut utiliser en ligne à https://scratch.mit.edu/

1 Avant propos

La présente extension intitulée scratch3, est une évolution de l'extension scratch qui imitait la version 2 du logiciel. Depuis janvier 2019, le logiciel est passé à la version 3 et l'extension scratch est donc devenue obsolète et non maintenue.

Cette version de scratch3 étant la première, elle est susceptible d'être immature, c'est-à-dire de contenir des bugs, des incohérences ou des effets non désirés. Si cela se produit, je remercie les utilisateurs de bien vouloir me le faire savoir par email ou *via* le « bugtracker » de gitlab à l'adresse

https://framagit.org/unbonpetit/scratch3/issues

Les utilisateurs les plus atentifs auront sans oute remarqué que seuls les blocs de type « stylo » sont disponibles parmi ceux qui sont dorénavant considérés comme « extension ». Il s'agit, pour l'instant, d'un choix délibéré...

Attention Il est fortement déconseillé de charger les packages scratch et scratch3 en raison de plusieurs définitions communes qu'ils partagent et qui conduiraient inévitablement à des dysfonctionnements. L'erreur qui survient — et le refus de charger le deuxième — lorsqu'on tente de charger ces deux packages ne doivent pas être désactivés par une modification de leurs codes.

L'extension scratch3 requiert les extensions simplekv et tikz qui sont automatiquement chargées si cela s'avère nécessaire.

Fidèle à mes convictions, la documentation de cette extension n'est disponible qu'en français.

2 L'environnement scratch

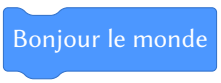
Pour dessiner un programme comme le fait scratch, il faut ouvrir un environnement « scratch » et écrire dans cet environnement les macros correspondant aux *blocs* que l'on veut y mettre :

```
\begin{scratch}
    macros pour dessiner des blocs
\end{scratch}
```

Comme le savent ceux qui enseignent l'algorithmique et la programmation avec le très-à-la-mode [2] logiciel « scratch », les programmes sont construits avec des briques, appelés « blocs », qui peuvent s'emboiter les uns sur les autres. Ces blocs sont de plusieurs couleurs, chacune correspondant à un type d'instruction que l'on retrouve dans les menus de scratch.
J'ai pris le parti d'écrire des macros ayant comme argument le texte qui figure dans le bloc. Ce faisant, on a plus de liberté que dans scratch où les blocs ont des textes prédéfinis, mais cette liberté permet d'utiliser cette extension quelle que soit la langue dans laquelle on écrit.
Enfin, j'ai cherché le bon compromis entre complexité du code et qualité des dessins obtenus avec cette extension : ils *ressemblent* à ceux du logiciel scratch, mais le but de cette extension n'est *pas* la ressemblance absolue au pixel près !

3 Les blocs normaux

Ces blocs sont les plus courants et possèdent une encoche d'emboitement, femelle en haut et mâle en bas. Les macros permettant de dessiner ces blocs ont des noms de la forme \block⟨*suffixe*⟩ et ont un seul argument obligatoire qui est le texte que l'on souhaite mettre dans le bloc. Par exemple, un bloc bleu (correspondant au menu « mouvement ») a un suffixe **move**, et est dessiné grâce à la macro \blockmove{⟨*texte*⟩}. Ainsi, dans l'environnement scratch, écrire \blockmove{Bonjour le monde} donne



La police d'écriture dans chaque bloc est la police « sans serif » qui est définie dans le document au moment où l'environnement est appelé : pratiquement, cela signifie que la macro \sffamily est exécutée avant que le texte des blocs ne soit composé. La clé « pre text » (voir page 8), modifiable par l'utilisateur, contient par défaut le code exécuté avant que du texte ne soit affiché, c'est-à-dire \sffamily. Dans cette documentation, la police sans serif est « biolinum ».
La plupart des dimensions des blocs sont proportionnelles à la taille de la police en cours. On peut donc jouer sur la taille de la police (via les classiques macros \small, \large, \footnotesize, etc) pour modifier la taille des blocs [3] .
Voici un inventaire des tous les blocs disponibles, empilés les uns sous les autres :

2. Je ne déteste rien de plus comme langage de programmation que ce *truc* vaguement graphique, ultra limité et contre-productif dans l'apprentissage du codage qu'est scratch et que l'éducation nationale veut à tout prix imposer. Je ne compte bien évidemment pas me plier à ce nouveau dogme ridicule et ne l'utiliserai ni ne l'enseignerai ; je considère que les élèves méritent mieux que scratch– ADA, java ou lua par exemple – comme entrée dans le monde de la programmation.
3. Il y a aussi la clé « scale » pour mettre le dessin à l'échelle que l'on souhaite, voir page 8

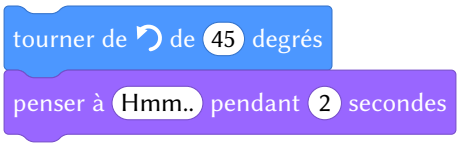
```
Voici un algorithme bizarre : \begin{scratch}
  \blockmove{bloc de mouvement}
  \blocklook{bloc d'apparence}
  \blocksound{bloc de son}
  \blockpen{bloc de stylo}
  \blockvariable{bloc de variable}
  \blocklist{bloc de liste}
  \blockevent{bloc d'événement}
  \blockcontrol{bloc de contrôle}
  \blocksensing{bloc de capteur}
\end{scratch}
```

Voici un algorithme bizarre :

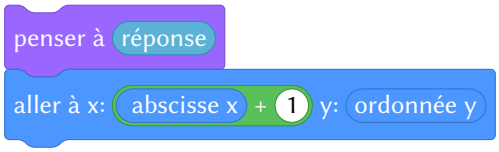
Il faut donc retenir cette logique : les suffixes **move**, **look**, **sound**, **pen**, **variable**, **list**, **event**, **control** et **sensing** correspondent aux couleurs des blocs. Il existe aussi le suffixe **operator** qui n'a pas été montré précédemment puisqu'aucun bloc n'existe pour la fonction « opérateurs ».

4 Les ovales

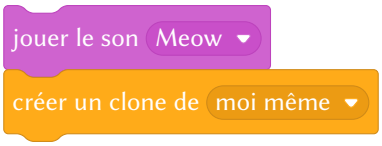
Les « ovales » sont, selon le code graphique de `scratch`, censés contenir des variables que l'utilisateur spécifie lui-même (comme des nombres ou du texte) :



Les variables peuvent être prédéfinies dans `scratch` ou créées par l'utilisateur et dans ce cas, les ovales sont de couleur identique à la couleur du thème auquel appartient la variable :



Ces variables peuvent également être spécifiques aux blocs concernés, sélectionnées parmi plusieurs possibilités, auquel cas l'ovale est de couleur plus sombre que le bloc et se termine avec une flèche de sélection :

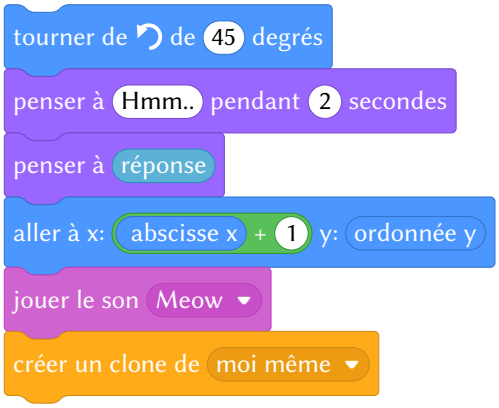


Au niveau des macros utilisées :

— la macro \ovalnum{⟨*nombre*⟩} dessine un ovale à fond blanc ;
— la macro \oval⟨*suffixe*⟩ trace un ovale de la couleur de ⟨*suffixe*⟩ ;
— en version étoilée, \oval⟨*suffixe*⟩* trace un ovale de couleur plus sombre avec une flèche de sélection ;
— les macros \turnleft et \turnright dessinent des flèches de rotation dans les blocs \blockmove.

```
Ovales de toutes sortes : \begin{scratch}
    \blockmove{tourner de \turnleft{} de \ovalnum{45} degrés}
    \blocklook{penser à \ovalnum{Hmm..} pendant \ovalnum{2} secondes}
    \blocklook{penser à \ovalsensing{réponse}}
    \blockmove{aller à x: \ovaloperator{\ovalmove{ abscisse x} + \ovalnum{1}} y: \ovalmove{ordonnée y}}
    \blocksound{jouer le son \ovalsound*{Meow}}
    \blockcontrol{créer un clone de \ovalcontrol*{moi même}}
\end{scratch}
```

Ovales de toutes sortes :



Toutes les macros de la forme \oval⟨*suffixe*⟩ sont utilisables *en dehors* de l'environnement scratch :

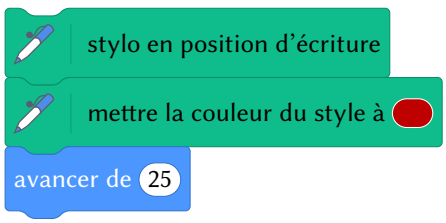
```
Voici une variable simple « \ovalmove{direction} »,
une variable sélectionnée « \ovalsound*{Meow} ».
```

Voici une variable simple « direction », une variable sélectionnée « Meow ».

Il faut signaler que les blocs de suffixes **operator**, **variable** , **list** ou **moreblocks** ne peuvent avoir d'ovales de sélection. N'ayant pas de sens pour scratch, une erreur sera émise si une macro étoilée \oval⟨*suffixe*⟩* est utilisée, et visuellement, cela se traduira par une couleur de remplissage du bloc en rouge.

Un ovale rempli de couleur et accessible avec la macro \pencolor{⟨*couleur*⟩} permet de sélectionner la couleur du stylo :

```
\begin{scratch}
  \blockpen{stylo en position d'écriture}
  \blockpen{mettre la couleur du style à \pencolor{red!75!black}}
  \blockmove{avancer de \ovalnum{25}}
\end{scratch}
```



5 Les blocs de début

Ces blocs sont de la couleur **event** pour la plupart (macro \blockinit), mais il existe aussi un bloc de début de couleur **control** (macro \blockinitclone). Le drapeau vert est dessiné avec la macro \greenflag.

```
Voici un début :
\begin{scratch}
  \blockinit{quand \greenflag est cliqué}
```

```
  \blockmove{suite de l'algorithme...}
\end{scratch}
et un autre :
\begin{scratch}
  \blockinitclone{quand je commence comme un clone}
  \blocklook{enfiler le costume de clown !}
\end{scratch}
```

Voici un début : 

et un autre : 

6 Les rectangles

Un menu déroulant contenant des valeurs prédéfinies est symbolisé par un rectangle dont la couleur reprend celle du bloc dans lequel il se trouve. Pour ce faire, la macro `\selectmenu{`⟨*texte*⟩`}` doit être exécutée :

```
\begin{scratch}
  \blockinit{Quand je reçois \selectmenu{message 1}}
  \blocksound{ajouter \ovalnum{10} à l'effet \selectmenu{hauteur}}
  \blocklook{mettre l'effet \selectmenu{couleur} à \ovalnum{0}}
  \blockvariable{mettre \selectmenu{ma_variable} à \ovalnum{10}}
\end{scratch}
```

7 Les losanges et les blocs de test

Dans la symbolique graphique de `scratch`, les losanges contiennent des valeurs booléennes ayant vocation à se retrouver dans un bloc de test. Pour dessiner de tels objets booléens, les macros `\bool`⟨*suffixe*⟩`{`⟨*texte*⟩`}` sont utilisées où les ⟨*suffixes*⟩ représentent les couleurs correspondant à la fonction du booléen tracé : **list**, **sensing** ou **operator**.
Les blocs de test sont de deux types,selon qu'ils possèdent ou pas une branche « else ».

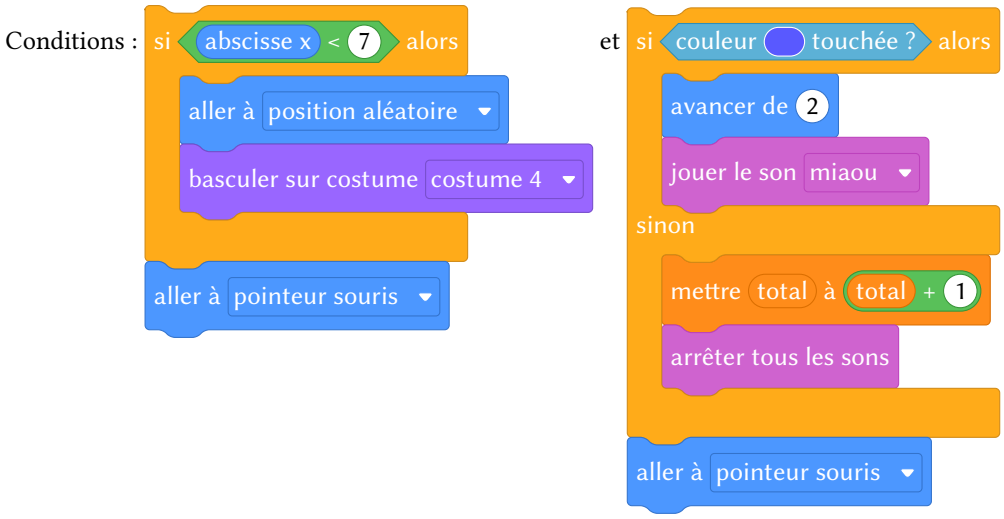
```
\blockif{⟨texte du test⟩}
    {⟨instructions si test vrai⟩}
```

et

```
\blockifelse{⟨texte du test⟩}
    {⟨instructions si test vrai⟩}
    {⟨instructions si test faux⟩}
```

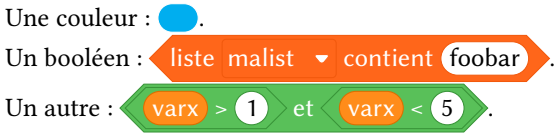
```
Conditions : \begin{scratch}
  \blockif{si \booloperator{\ovalmove{abscisse x} < \ovalnum{7}} alors}
    {\blockmove{aller à \selectmenu{position aléatoire}}
     \blocklook{basculer sur costume \selectmenu{costume 4}}
    }
  \blockmove{aller à \selectmenu{pointeur souris}}
\end{scratch}
et
\begin{scratch}
  \blockifelse{si \boolsensing{couleur \pencolor{blue!65} touchée ?} alors}
```

```
    {\blockmove{avancer de \ovalnum{2}}
    \blocksound{jouer le son \selectmenu{miaou}}
    }
    {\blockvariable{mettre \ovalvariable{total} à \ovaloperator{\ovalvariable{total} + \ovalnum{1}}}
    \blocksound{arrêter tous les sons}
    }
  \blockmove{aller à \selectmenu{pointeur souris}}
\end{scratch}
```



La macros `\pencolor` et celles de la forme `\bool`⟨*suffixe*⟩ sont utilisables *en dehors* d'un environnement `scratch` :

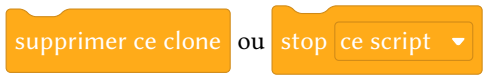
```
Une couleur : \pencolor{cyan}.\par
Un booléen : \boollist{liste \selectmenu{malist} contient \ovalnum{foobar}}.\par
Un autre : \booloperator{\booloperator{\ovalvariable{varx} > \ovalnum{1}}
et \booloperator{\ovalvariable{varx} < \ovalnum{5}}}.
```



8 Les blocs de fin

Ces blocs sont susceptibles de clore un algorithme et n'ont donc pas d'encoche mâle dans leur partie basse. Ils ne peuvent être que du type **control** et sont dessinés avec la macro `\blockstop{`⟨*texte*⟩`}`

```
\begin{scratch}\blockstop{supprimer ce clone}\end{scratch}
ou
\begin{scratch}\blockstop{stop \selectmenu{ce script}}\end{scratch}
```

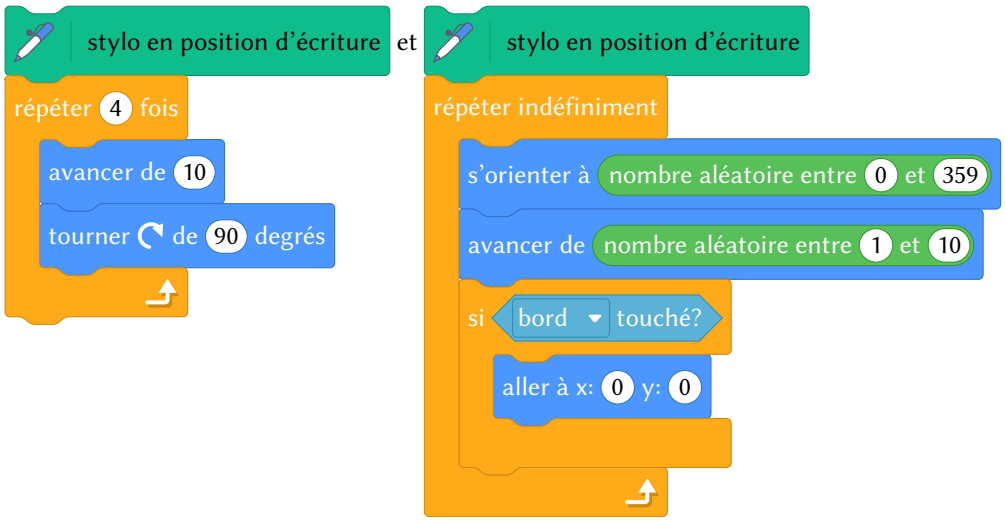


9 Les blocs de répétition

Ces blocs sont de deux types, selon que la répétition est prévue pour s'arrêter ou pas (boucle infinie). Ils seront dessinés par les macros `\blockrepeat` et `\blocinfloop` ayant chacune *deux* arguments : le premier étant le ⟨*texte*⟩ du bloc et le second la suite d'instructions à répéter.

```
\begin{scratch}
  \blockpen{stylo en position d'écriture}
  \blockrepeat{répéter \ovalnum{4} fois}
    {
    \blockmove{avancer de \ovalnum{10}}
    \blockmove{tourner \turnright{} de \ovalnum{90} degrés}
```

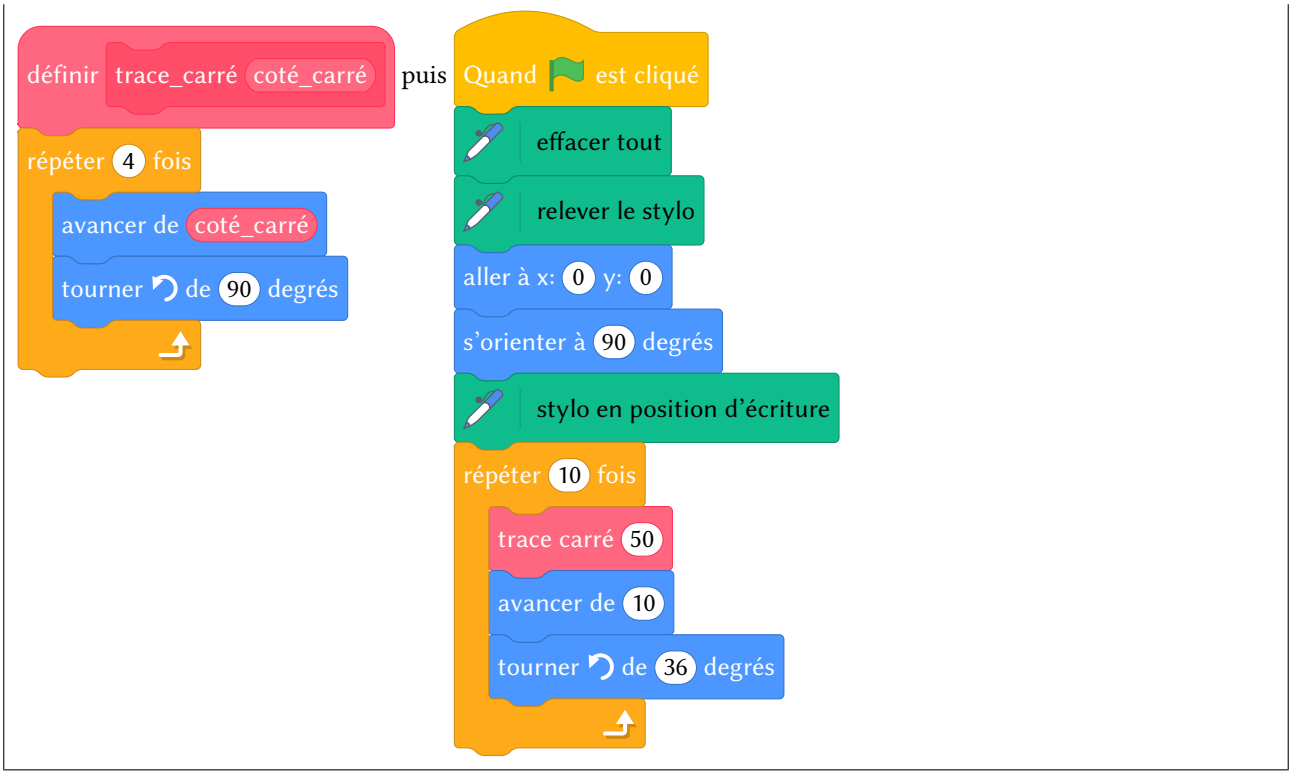
```
    }
\end{scratch}
et
\begin{scratch}
  \blockpen{stylo en position d'écriture}
  \blockinfloop{répéter indéfiniment}
    {
    \blockmove{s'orienter à \ovaloperator{nombre aléatoire entre \ovalnum{0} et \ovalnum{359}}}
    \blockmove{avancer de \ovaloperator{nombre aléatoire entre \ovalnum{1} et \ovalnum{10}}}
    \blockif{si \boolsensing{\selectmenu{bord} touché?}}
      {
      \blockmove{aller à x: \ovalnum{0} y: \ovalnum{0}}
      }
    }
\end{scratch}
```



10 Les blocs de définition

Les « procédures », ayant le suffixe **moreblocks**, permettent d'étendre les maigres possibilités données au programmeur en scratch.

```
\begin{scratch}
  \initmoreblocks{définir \namemoreblocks{trace_carré \ovalmoreblocks{coté_carré}}}
  \blockrepeat{répéter \ovalnum4 fois}
    {\blockmove{avancer de \ovalmoreblocks{coté_carré}}
    \blockmove{tourner \turnleft{} de \ovalnum{90} degrés}
    }
\end{scratch}
puis
\begin{scratch}
  \blockinit{Quand \greenflag est cliqué}
  \blockpen{effacer tout}
  \blockpen{relever le stylo}
  \blockmove{aller à x: \ovalnum0 y: \ovalnum0}
  \blockmove{s'orienter à \ovalnum{90} degrés}
  \blockpen{stylo en position d'écriture}
  \blockrepeat{répéter \ovalnum{10} fois}
    {
    \blockmoreblocks{trace carré \ovalnum{50}}
    \blockmove{avancer de \ovalnum{10}}
    \blockmove{tourner \turnleft{} de \ovalnum{36} degrés}
    }
\end{scratch}
```



À l'intérieur d'un `\namemoreblocks`, on peut mettre `\ovalmoreblocks` pour spécifier un argument de type nombre ou texte ou bien `\boolmoreblocks` pour un argument de type booléen.

11 Bloc invisible

Bien que ce genre de bloc n'existe pas avec `scratch`, cette fonctionnalité peut s'avérer utile. On insère un espace vide avec `\blockspace[`⟨*coeff*⟩`]`. L'espace verticale insérée est égale à la hauteur normale d'un bloc multipliée par le ⟨*coeff*⟩, valeur optionnelle qui vaut 1 par défaut.

```
\begin{scratch}
  \blockmove{ci-dessous, une espace d'un bloc}
  \blockspace
  \blockmove{ci dessous, une espace égale à la moitié d'un bloc}
  \blockspace[0.5]
  \blockmove{la suite}
\end{scratch}
```

ci-dessous, une espace d'un bloc

ci dessous, une espace égale à la moitié d'un bloc

la suite

12 Personnalisation

Plusieurs ⟨*paramètres*⟩ peuvent être réglés par l'utilisateur selon la syntaxe ⟨*clé*⟩=⟨*valeur*⟩. Cas paramètres peuvent être spécifiés dans :

- l'argument optionnel de l'environnement `\begin{scratch}[`⟨*paramètres*⟩`]` auquel cas les ⟨*paramètres*⟩ ne s'appliquent qu'à cet environnement ;
- l'argument de la macro `\setscratch{`⟨*paramètres*⟩`}` pour spécifier des ⟨*paramètres*⟩ pour les environnements `scratch` à venir ;

— l'argument de `\setdefaultscratch{⟨paramètres⟩}` pour spécifier des ⟨*paramètres*⟩ *par défaut*.

Il existe la macro `\resetscratch` qui remet à leur valeur par défaut tous les ⟨*paramètres*⟩ de `scratch`, pour annuler les effets d'une macro `\setscratch`.

Voici les ⟨*paramètres*⟩ disponibles :

`else word=⟨caractères⟩` (*Défaut* : `sinon`)
Représente est le mot qui est inséré dans la branche « else » d'un bloc de test.

`pre text=⟨code⟩` (*Défaut* : `\sffamily `)
C'est le code qui est exécuté avent que du texte ne soit affiché par `scratch3`.

`x sep=⟨dimension⟩` (*Défaut* : `0.33333em`)
Représente l'espace horizontale insérée entre les bords droit et gauche du texte du bloc et les bords droits et gauche du bloc. La valeur est ramenée si nécessaire dans l'intervalle [3pt ; 1em].

`y sepsup=⟨dimension⟩` (*Défaut* : `3pt`)
Représente l'espace verticale insérée entre le bas de l'encoche femelle et le bord supérieur du texte du bloc. La valeur est ramenée si nécessaire dans l'intervalle [3pt ; 3ex].

`y sepinf=⟨dimension⟩` (*Défaut* : `5pt`)
Représente l'espace verticale insérée entre le bas du bloc et le bord inférieur du texte du bloc. La valeur est ramenée si nécessaire dans l'intervalle [3pt ; 3ex].

`line width=⟨dimension⟩` (*Défaut* : `.4pt`)
Représente l'épaisseur des lignes de relief des blocs et le double des lignes de relief des losanges booléens. La valeur est ramenée si nécessaire dans l'intervalle [0pt ; 5pt].

`loop width=⟨dimension⟩` (*Défaut* : `3ex`)
Représente est la largeur de la barre verticale des blocs de répétition ou de test. La valeur est ramenée si nécessaire dans l'intervalle [3pt ; 3em].

`loop height=⟨dimension⟩` (*Défaut* : `1.75ex`)
Représente est l'épaisseur des barres horizontales « else » et inférieure des blocs de répétition ou de test. La valeur est ramenée si nécessaire dans l'intervalle [3pt ; 3ex].

`corner=⟨dimension⟩` (*Défaut* : `0.66667ex`)
Représente la dimension des chanfreins des blocs. La valeur est ramenée si nécessaire dans l'intervalle [0.33333ex ; 1ex].

`notch=⟨dimension⟩` (*Défaut* : `1em`)
Représente la largeur des encoches. La valeur est ramenée si nécessaire dans l'intervalle [0.33333em ; 3em].

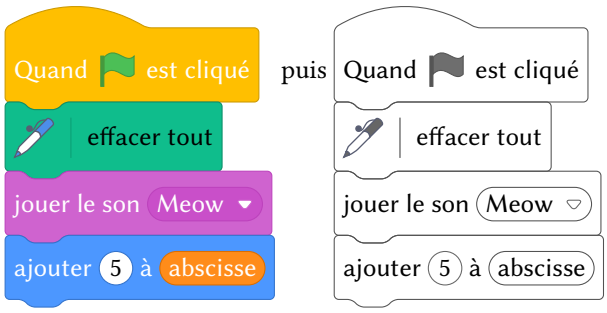
`scale=⟨coefficient⟩` (*Défaut* : `1`)
Représente l'échelle à laquelle est représenté le dessin. La valeur est ramenée si nécessaire dans l'intervalle [0.2 ; 5].

`init arcangle=⟨angle⟩` (*Défaut* : `30`)
Représente l'angle avec l'horizontale de l'arc de cercle tracé dans la partie haute des blocs de départ. La valeur est ramenée si nécessaire dans l'intervalle [20 ; 40].

`init arclength=⟨dimension⟩` (*Défaut* : `5em`)
Représente la longueur horizontale de l'arc de cercle tracé dans la partie haute des blocs de départ. La valeur est ramenée si nécessaire dans l'intervalle [3em ; 8em].

`print=⟨booléen⟩` (*Défaut* : `false`)
Lorsque ce booléen est vrai, les dessins se font en noir et blanc de façon à pouvoir être dirigés vers une impression en noir et blanc.

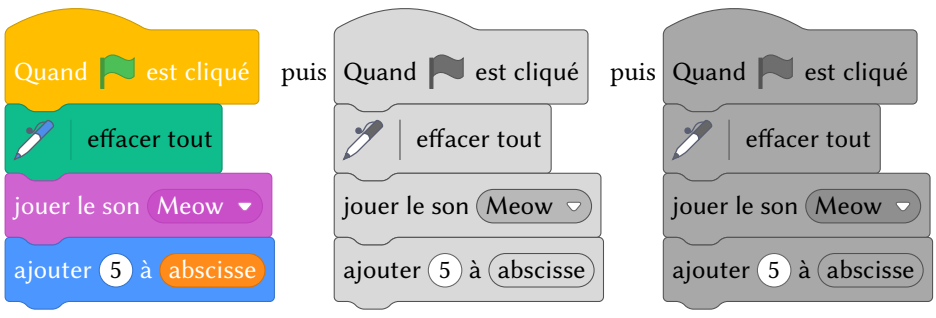
```
\begin{scratch}
  \blockinit{Quand \greenflag est cliqué}
  \blockpen{effacer tout}
  \blocksound{jouer le son \ovalsound*{Meow}}
  \blockmove{ajouter \ovalnum{5} à \ovalvariable{abscisse}}
\end{scratch} puis
\begin{scratch}[print]
  \blockinit{Quand \greenflag est cliqué}
  \blockpen{effacer tout}
  \blocksound{jouer le son \ovalsound*{Meow}}
  \blockmove{ajouter \ovalnum{5} à \ovalvariable{abscisse}}
\end{scratch}
```



fill blocks=⟨booléen⟩ (*Défaut* : `false`)
Ce booléen n'est pris en compte que lorsque le booléen `print` est vrai. Si `fill blocks` est vrai, tous les dessins (sauf les ovales contenant des nombres) seront remplis avec un gris choisi avec la clé suivante.

fill gray=⟨taux de gris⟩ (*Défaut* : `0.85`)
Lorsque `fill blocks` est vrai, ce taux de blanc dans le gris (nombre compris entre 0 pour noir et 1 pour blanc) est utilisé pour définir une couleur de remplissage des dessins. La valeur est ramenée si nécessaire dans l'intervalle $[0\,;1]$.

```
\begin{scratch}
  \blockinit{Quand \greenflag est cliqué}
  \blockpen{effacer tout}
  \blocksound{jouer le son \ovalsound*{Meow}}
  \blockmove{ajouter \ovalnum{5} à \ovalvariable{abscisse}}
\end{scratch} puis
\begin{scratch}[print,fill blocks]
  \blockinit{Quand \greenflag est cliqué}
  \blockpen{effacer tout}
  \blocksound{jouer le son \ovalsound*{Meow}}
  \blockmove{ajouter \ovalnum{5} à \ovalvariable{abscisse}}
\end{scratch} puis
\begin{scratch}[print,fill blocks,fill gray=0.66]
  \blockinit{Quand \greenflag est cliqué}
  \blockpen{effacer tout}
  \blocksound{jouer le son \ovalsound*{Meow}}
  \blockmove{ajouter \ovalnum{5} à \ovalvariable{abscisse}}
\end{scratch}
```

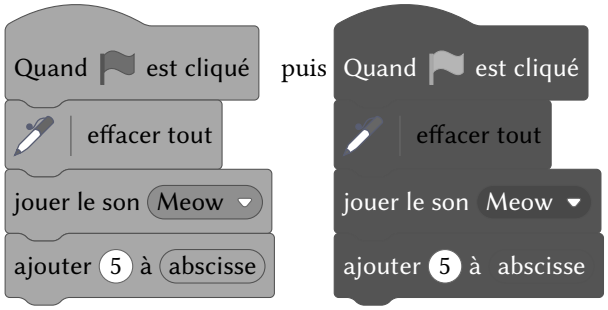


contrast=⟨entier⟩ (*Défaut* : 20)
Lorsque l'option print est vraie, cet entier, compris entre 0 et 100 inclus, qualifie la différence de teinte entre les lignes de relief et la teinte de gris de ces blocs, spécifiée par la clé fill gray. L'entier 0 sélectionne la couleur spécifiée par fill gray tandis que 100 trace les lignes en noir. La valeur est ramenée si nécessaire dans l'intervalle [0 ; 100].

text color=⟨couleur⟩ (*Défaut* : black)
Lorsque fill blocks est vrai, cette couleur sera utilisée pour le texte des blocs.

flag gray=⟨taux de gris⟩ (*Défaut* : 0.33)
Lorsque print est vrai, ce taux de gris est utilisé pour la couleur du drapeau tracé avec \greenflag ainsi que pour la flèche se trouvant au bas des blocs de répétition. La valeur est ramenée si nécessaire dans l'intervalle [0 ; 1].

```
\begin{scratch}[print,fill blocks,fill gray=0.66]
  \blockinit{Quand \greenflag est cliqué}
  \blockpen{effacer tout}
  \blocksound{jouer le son \ovalsound*{Meow}}
  \blockmove{ajouter \ovalnum{5} à \ovalvariable{abscisse}}
\end{scratch} puis
\begin{scratch}[print,fill blocks,fill gray=0.33,text color=white,flag gray=0.66]
  \blockinit{Quand \greenflag est cliqué}
  \blockpen{effacer tout}
  \blocksound{jouer le son \ovalsound*{Meow}}
  \blockmove{ajouter \ovalnum{5} à \ovalvariable{abscisse}}
\end{scratch}
```



line gray=⟨taux de gris⟩ (*Défaut* : 0.4)
Lorsque print est vrai, ce taux de gris est utilisé pour la couleur des lignes de relief. La valeur est ramenée si nécessaire dans l'intervalle [0 ; 1].

num blocks=⟨booléen⟩ (*Défaut* : false)
Lorsque ce booléen est vrai, les blocs sont numérotés.

num sep=⟨dimension⟩ (*Défaut* : 3pt)
Cette clé contient l'espacement entre les numéros de blocs et leur frontière gauche. La valeur est ramenée si nécessaire dans l'intervalle [0pt ; 1.5em].

num start=⟨entier⟩ (*Défaut* : 1)
Cette clé contient le premier numéro du bloc.
La macro \numblock est chargée d'imprimer les numéros de ligne. Elle admet un argument (le numéro de ligne) et permet, lorsqu'elle est redéfinie, de personnaliser la numérotation : choix de la couleur, de la police, de sa taille et des effets applicables à son argument. Par défaut, cette macro est définie par

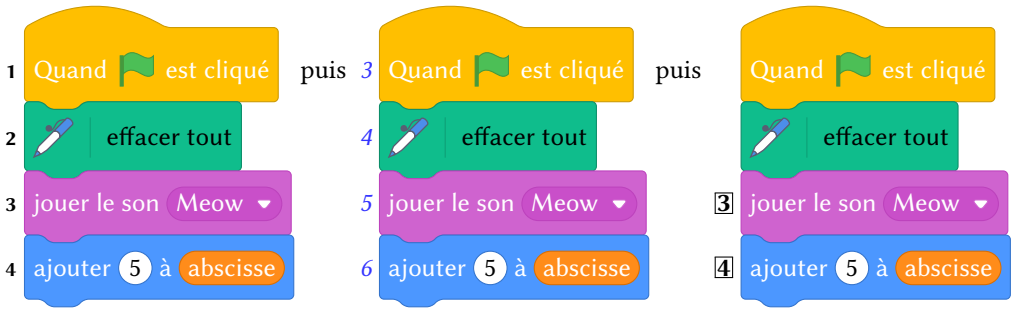
```
\newcommand*\numblock[1]{\color{black}\footnotesize\bfseries#1}
```

Dans l'environnement scratch, entre des instructions scratch, on peut activer ou désactiver la numérotation *pour le dessin en cours* à l'aide de

```
\setscratch{num blocks=⟨booléen⟩}
```

```
\begin{scratch}[num blocks]
  \blockinit{Quand \greenflag est cliqué}
  \blockpen{effacer tout}
  \blocksound{jouer le son \ovalsound*{Meow}}
```

```
  \blockmove{ajouter \ovalnum{5} à \ovalvariable{abscisse}}
\end{scratch} puis
\renewcommand*\numblock[1]{\color{blue!80}\itshape#1}
\begin{scratch}[num blocks,num start=3]
  \blockinit{Quand \greenflag est cliqué}
  \blockpen{effacer tout}
  \blocksound{jouer le son \ovalsound*{Meow}}
  \blockmove{ajouter \ovalnum{5} à \ovalvariable{abscisse}}
\end{scratch}
puis
\renewcommand*\numblock[1]{\fboxsep=0.5pt\fbox{\bfseries#1}}
\begin{scratch}[num blocks=false]
  \blockinit{Quand \greenflag est cliqué}
  \blockpen{effacer tout}
  \setscratch{num blocks=true}
  \blocksound{jouer le son \ovalsound*{Meow}}
  \blockmove{ajouter \ovalnum{5} à \ovalvariable{abscisse}}
\end{scratch}
```

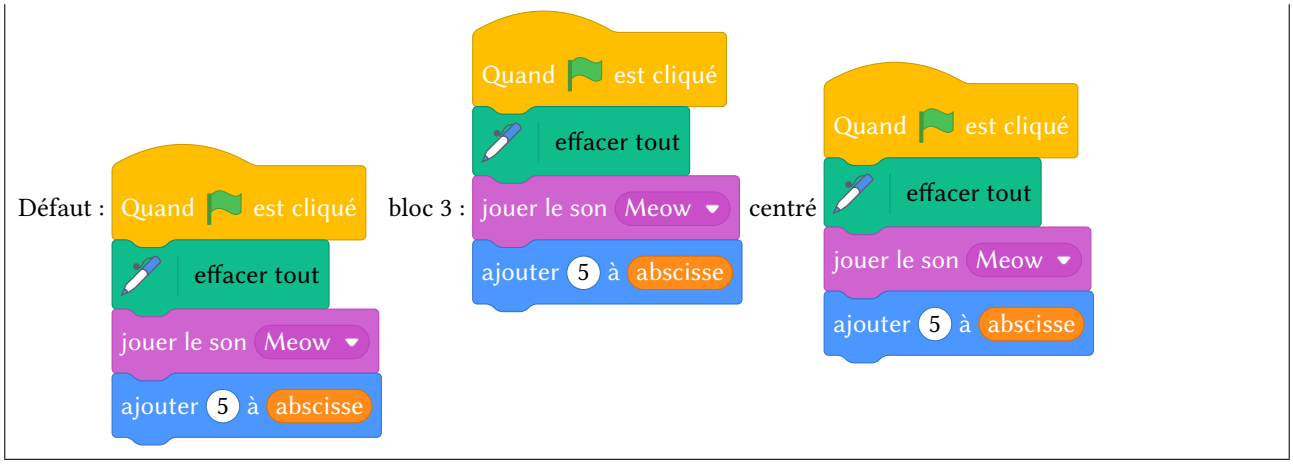


baseline=⟨alignement⟩ (*Défaut* : 1)

Cette clé contient la consigne d'alignement du dessin tout entier, c'est-à-dire :

- la lettre « c » auquel cas, le dessin est centré sur la ligne de base
- un entier qui spécifie un numéro de bloc existant : la numérotation est cohérente avec l'entier initial contenu dans la clé `num start`. Dans ce cas, la ligne de base choisie est celle du texte contenu dans le bloc portant le numéro choisi.

```
Défaut : \begin{scratch}
  \blockinit{Quand \greenflag est cliqué}
  \blockpen{effacer tout}
  \blocksound{jouer le son \ovalsound*{Meow}}
  \blockmove{ajouter \ovalnum{5} à \ovalvariable{abscisse}}
\end{scratch}
bloc 3 : \begin{scratch}[baseline=3]
  \blockinit{Quand \greenflag est cliqué}
  \blockpen{effacer tout}
  \blocksound{jouer le son \ovalsound*{Meow}}
  \blockmove{ajouter \ovalnum{5} à \ovalvariable{abscisse}}
\end{scratch}
centré
\begin{scratch}[baseline=c]
  \blockinit{Quand \greenflag est cliqué}
  \blockpen{effacer tout}
  \blocksound{jouer le son \ovalsound*{Meow}}
  \blockmove{ajouter \ovalnum{5} à \ovalvariable{abscisse}}
\end{scratch}
```



13 Mot de la fin

Le code de cette extension démontre mon immense ignorance de ti*k*z/pgf et les méthodes de programmation qui lui sont propres que, décidément, je ne comprendrai jamais ! C'est sans doute le trop grand éloignement avec la logique de TEX et la documentation de ti*k*z/pgf, aussi indigeste qu'illisible, qui explique cette incompatibilité d'humeur et mon désintérêt à l'égard de ti*k*z. Toujours est-il que cette extension fonctionne, avec une lenteur certaine que j'attribue à ma programmation hasardeuse ainsi qu'à la lenteur intrinsèque de ti*k*z.
Toute remarque, remontée de bug — je n'ose pas dire amélioration du code —, demande d'implémentation de fonctionnalité est bien évidemment bienvenue ; j'invite les utilisateurs à m'en faire part *via* email à `unbonpetit@netc.fr`

14 Historique

v0.1 19/02/2019

— Première version.

v0.11 10/03/2019

— Correction d'un bug dans `\scr_blockloop` : la couleur `\scr_current_blockcolor` y est désormais définie ;

— Correction d'un bug dans `\boolmoreblocks` : la couleur est correcte et `rounded corners` désormais est nul pour le tracé d'un losange.