Traitement

- Logique
- Boucles
- Mathématiques
- Textes
- Listes
- Utilisation
- Variables
- Fonctions

Logique

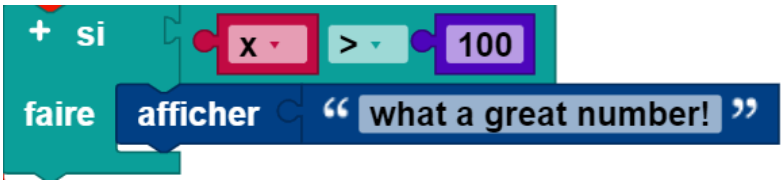
Instructions conditionnelles

Les instructions conditionnelles sont essentielles pour la programmation. Elles permettent de formuler des distinctions de cas telles que :

- S'il y a un moyen de tourner à gauche, rotation à gauche.
- Si le score = 100, imprimer « Bien joué ! ».

Blocs **si**

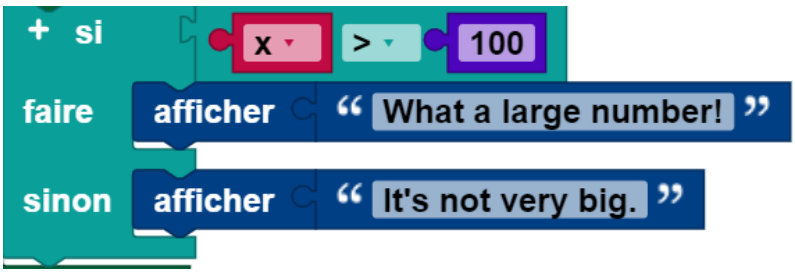
La condition la plus simple est un bloc **si** :



Lorsqu'il est exécuté, la valeur de la variable **x** est comparée à 100. Si elle est supérieure, « Quel grand nombre ! » est édité. Sinon, il ne se passe rien.

Blocs **si-sinon**

Il est également possible d'indiquer que quelque chose doit se produire si la condition n'est pas vraie, comme dans cet exemple :

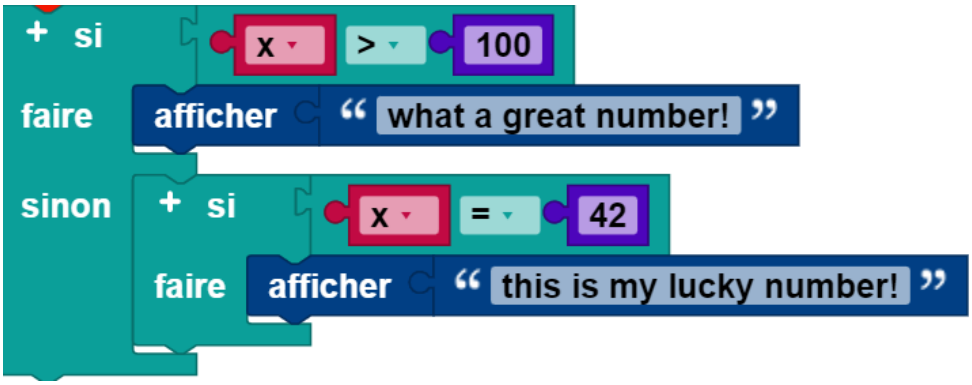


Comme pour le bloc précédent, « Quel grand nombre ! » est émis lorsque **x** > 100. Dans le cas contraire, « Ce n'est pas très grand » est édité.

Un bloc **si** peut avoir une section **sinon**,mais pas plus d'une.

Blocs **si-sinon-si**

Il est également possible de tester plusieurs conditions avec un seul bloc **si** en ajoutant des clauses **sinon**-**si** :

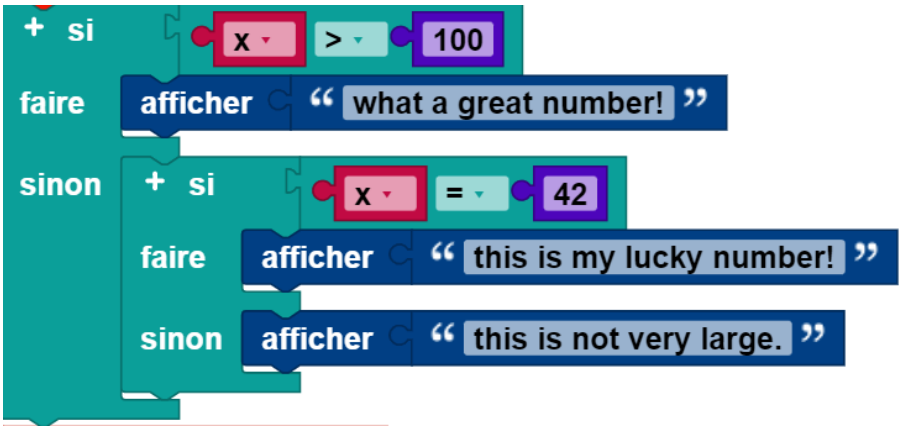


Le bloc vérifie d'abord si **x** > 100 et émet « Quel grand nombre ! » si c'est le cas. Si ce n'est pas le cas, il continue à vérifier si **x** = 42. Si oui, il édite « C'est mon chiffre porte-bonheur ! ». Sinon, il ne se passe rien.

Un bloc **si** peut avoir un nombre quelconque de sections **sinon**-**si**. Les conditions sont évaluées de haut en bas jusqu'à ce qu'une condition soit remplie ou jusqu'à ce qu'il ne reste plus aucune condition.

Blocs **si-sinon-si-sinon**

Les blocs **si** peuvent comporter aussi bien des sections **sinon**-**si** que des sections **sinon** :



La section **sinon** garantit qu'une action sera exécutée même si aucune des conditions précédentes n'est vraie.

Une autre section **sinon** peut se produire après n'importe quel nombre de sections **sinon**-**si**, y compris zéro, pour obtenir un bloc **si**-**sinon** normal.

Modification de bloc

La barre d'outils affiche uniquement le bloc **si** simple et le bloc **si**-**sinon** :

Pour ajouter des clauses **si-sinon** et **sinon**, cliquez sur l'icône (+). L'icône (-) permet de supprimer à nouveau les clauses **sinon-si** :

Remarquez que les formes des blocs permettent d'ajouter un nombre quelconque de sous-blocs **sinon-si**, mais seulement jusqu'à un bloc **si**.

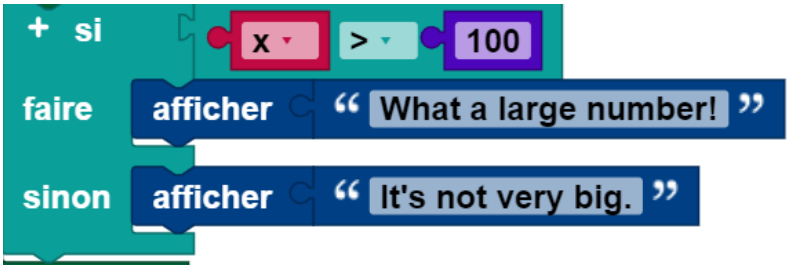
Logique booléenne

La logique booléenne est un système mathématique simple qui a deux valeurs :

- **vrai**
- **faux**

Les blocs logiques dans ROBO Pro Coding sont généralement destinés à contrôler les conditions et les boucles.

Voici un exemple :



Si la valeur de la variable x est supérieure à 100, la condition est vraie et le texte « Quel grand nombre ! » est édité. Si la valeur de x n'est pas supérieure à 100, la condition est fausse et « Ce n'est pas très grand » est édité. Les valeurs booléennes peuvent également être stockées dans des variables et transmises à des fonctions, de

même que les nombres, le texte et les valeurs de liste.

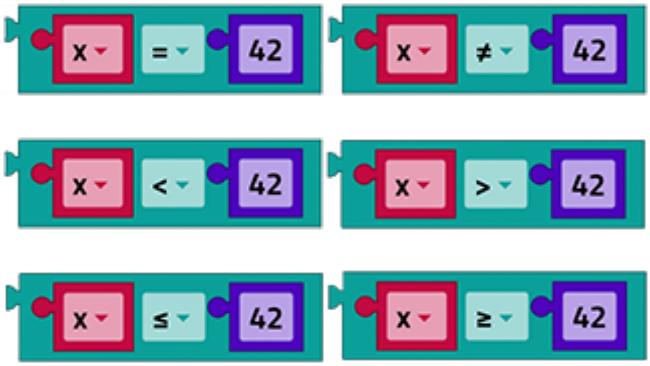
Si un bloc attend une valeur booléenne comme entrée, une entrée manquante est interprétée comme **incorrecte** . Les valeurs non booléennes ne peuvent pas être insérées directement là où des valeurs booléennes sont attendues, bien qu'il soit possible (mais non conseillé) d'enregistrer une valeur non booléenne dans une variable et de l'insérer ensuite dans l'entrée de condition. Cette méthode n'est pas recommandée et son comportement peut changer dans les versions futures de ROBO Pro Coding.

Valeurs

Un seul bloc avec une liste déroulante indiquant soit **vrai**, soit **faux** peut être utilisé pour extraire une valeur booléenne :

Opérateurs de comparaison

Il y a six opérateurs de comparaison. Deux entrées (normalement deux nombres) sont transmises à chacune et l'opérateur de comparaison renvoie **vrai** ou **faux**, selon la manière dont les entrées sont comparées.



Les six opérateurs sont : égal, non égal, inférieur, supérieur, inférieur ou égal, supérieur ou égal.

Opérateurs logiques

Le bloc **et** transmet alors et seulement alors le signal **vrai** si ses deux valeurs d'entrée sont vraies.

Le bloc **ou** transmet le signal **vrai** si au moins une de ses deux valeurs d'entrée est vraie.

pas

Le bloc **pas** transforme une saisie booléenne en sa contrepartie. Par exemple, le résultat de :

faux.

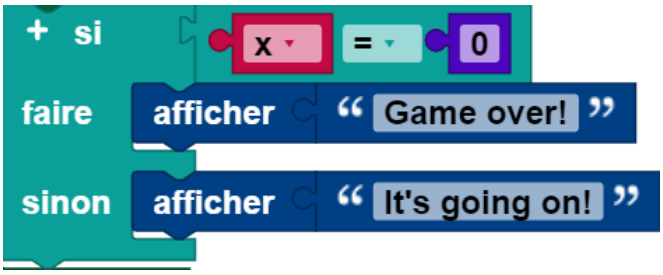
En l'absence de saisie, la valeur **vraie** est enregistrée, de sorte que le bloc suivant génère la valeur **faux** :

Toutefois, il n’est pas recommandé de laisser une entrée vide.

Triple opérateur

Le triple opérateur se comporte comme un bloc **si-sinon** miniature. Il prend trois valeurs d’entrée, la première valeur d’entrée étant la condition booléenne à tester, la deuxième valeur d’entrée étant la valeur à restituer si le test est **vrai**, la troisième valeur d’entrée étant la valeur à restituer, si le test est faux. Dans l’exemple suivant, la variable **Couleur** est définie sur le rouge si la variable **x** est inférieure à 10, sinon la variable **Couleur** est définie sur le vert.

Un bloc triple peut toujours être remplacé par un bloc **si-sinon**. Les deux exemples suivants sont exactement identiques.



Boucles

La zone « Commande » contient des blocs qui contrôlent si d'autres blocs placés à l'intérieur de celle-ci sont exécutés. Il y a deux types de blocs de contrôle : les blocs **si-sinon**-Blöcke (décrits sur une page) et les blocs qui contrôlent le nombre de fois où leur intérieur est exécuté. Ces derniers sont appelés boucles, car leur intérieur, également appelé corps ou corps de boucle, est répété (éventuellement) plusieurs fois. Chaque passage d'une boucle est appelé itération.

Blocs pour créer des boucles

Répétition permanente

Le bloc **Répétition permanente** exécute le code dans son corps jusqu'à la fin du programme.

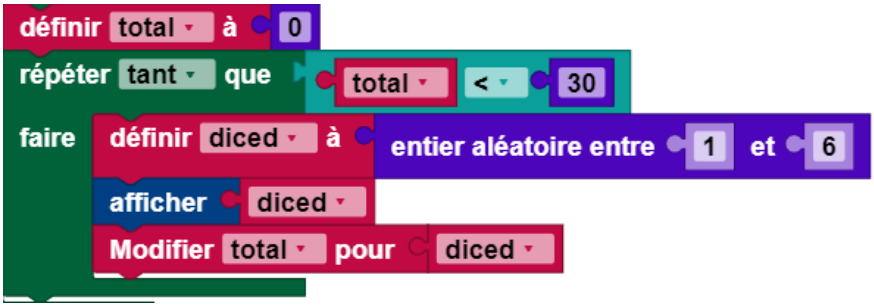
Répéter

Le bloc **Répétition** exécute le code dans son corps autant de fois que prévu, par exemple dix fois « Bonjour ! ». Le bloc suivant émet par exemple dix fois « Salut ! » :

Répétition tant que

Imaginez un jeu où un joueur lance un dé et additionne toutes les valeurs lancées tant que la somme est inférieure à 30. Les blocs suivants exécutent cette partie :

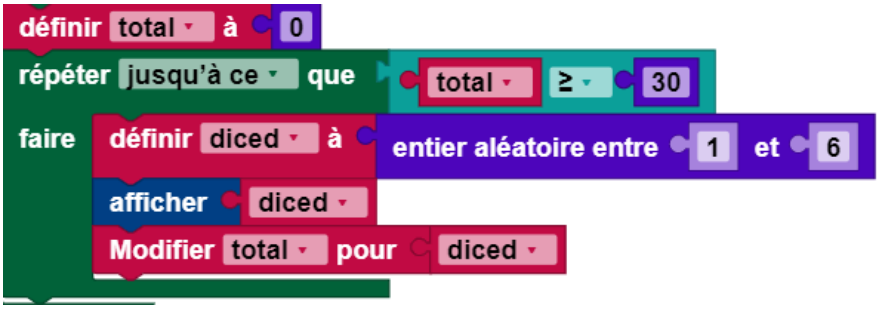
1. Une variable nommée **total** obtient une valeur initiale de 0.
2. 2. La boucle commence par vérifier si le **total** est inférieur à 30. Si c'est le cas, les blocs passent dans le corps.
3. 3. Un nombre aléatoire compris entre 1 et 6 est généré (pour simuler un lancer de dé) et stocké dans une variable nommée **dés**.
4. Le nombre en dés est édité.
5. La variable **totale** est augmentée de la valeur **en dés**.
6. Une fois la fin de la boucle atteinte, le contrôleur revient à l'étape 2.



Après la fin de la boucle, tous les blocs suivants (non représentés) sont passés. Dans l'exemple, le passage en boucle se termine après qu'un certain nombre de nombres aléatoires se situent dans la plage de 1 à 6, et la variable **total** a alors comme valeur la somme de ces nombres qui est d'au moins 30.

Répétition jusqu'à

Les boucles **Répétition tant que** répètent leur corps, **tant que** une condition est remplie. Les boucles **Répétition jusqu'à** sont similaires, à la différence qu'elles répètent leur corps **jusqu'à** ce qu'une condition définie soit remplie. Les blocs suivants sont équivalents à l'exemple précédent, car la boucle est exécutée jusqu'à ce que le **total** soit supérieur ou égal à 30.



Compter-de-à

La boucle **Compter-de-à**augmente la valeur d'une variable en commençant par une première valeur, se terminant par une deuxième valeur et par incréments d'une troisième valeur, le corps étant exécuté une fois pour chaque valeur de la variable. Le programme suivant donne par exemple les chiffres 1, 3 et 5.

Comme le montrent les deux boucles suivantes, qui émettent respectivement les nombres 5, 3 et 1, cette première valeur peut être supérieure à la seconde. Le comportement est le même, que le montant incrémental (troisième valeur) soit positif ou négatif.

Pour chacun

Le bloc **pour chacun** est similaire à celui de la boucle **Compter-de-à**, sauf qu'il utilise les valeurs d'une liste à la place de la variable de boucle dans un ordre numérique. Le programme suivant expose chaque élément de la liste « alpha », « bêta », « gamma » :

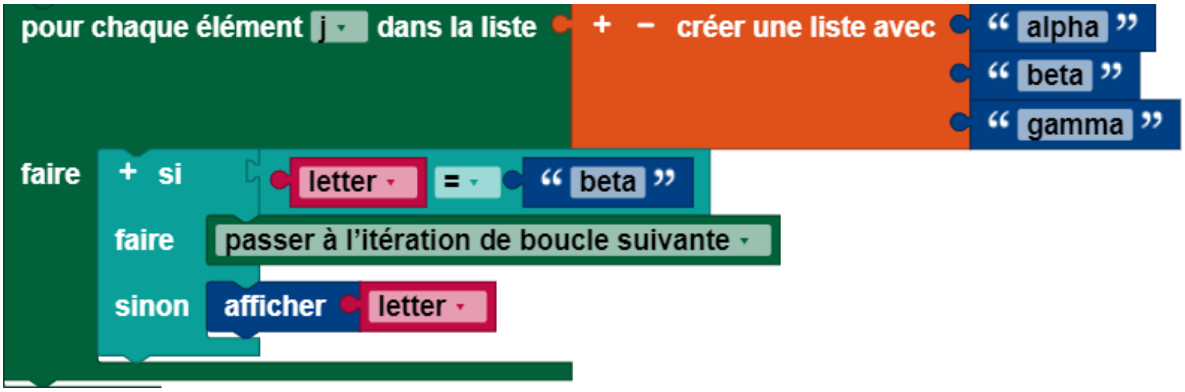
Blocs de rupture de boucles

La plupart des boucles sont exécutées jusqu'à ce que la condition d'interruption soit satisfaite (pour les blocs **Répéter**) ou jusqu'à ce que toutes les valeurs de la variable de boucle soient acceptées (pour les boucles **Compter avec** et pour les boucles **pour chacun**). Deux blocs rarement nécessaires mais parfois utiles offrent des possibilités supplémentaires de contrôle du comportement de la boucle. Ils peuvent être utilisés pour n'importe quel type de boucle, même si les exemples suivants montrent leur utilisation pour les boucles **pour chacun**.

Poursuivre avec la prochaine itération

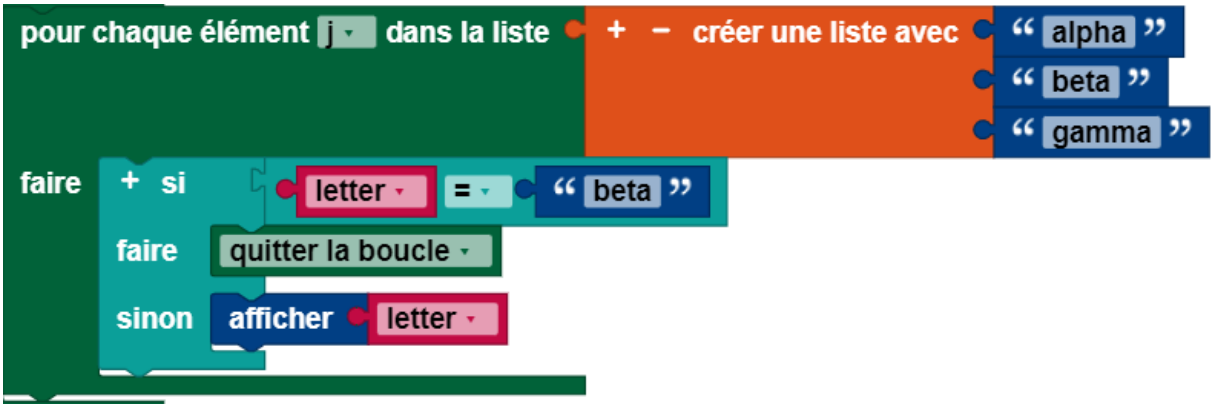
La boucle **Poursuivre avec la prochaine itération** permet de passer les blocs restants dans le corps de boucle et de commencer l'itération suivante de la boucle.

Le programme suivant donne « alpha » à la première itération de la boucle. Lors de la deuxième itération, le bloc **continue avec la prochaine intégration**, ce qui fait sauter la sortie de « beta ». Lors de la dernière itération, « gamma » est imprimé.



Rupture de boucle

Le bloc **Rupture de boucle** permet une sortie prématurée d'une boucle. Le programme suivant donne « alpha » à la première itération et interrompt la boucle à la seconde itération si la variable de boucle est égale à « beta ». Le troisième point de la liste n'est jamais atteint.


pour chaque élément j dans la liste
+ − créer une liste avec
" alpha "
" beta "
" gamma "
faire
+ si
letter
=
" beta "
faire
quitter la boucle
sinon
afficher
letter

Mathématiques

Les blocs de la catégorie Mathématiques sont utilisés pour effectuer des calculs. Les résultats des calculs peuvent, par exemple, être utilisés comme valeurs pour les variables. La plupart des blocs mathématiques font référence à des calculs mathématiques généraux et devraient être explicites.

Blocs

Chiffres

Utilisez le bloc Chiffres pour ajouter n'importe quel nombre à votre programme ou pour assigner ce nombre à une variable en tant que valeur. Ce programme attribue le chiffre 12 à la variable **Âge** :

Calculs simples

Ce bloc a une structure valeur-opérateur-valeur. Les modes de calcul **+**, **-**, **÷**, **×** et **^** sont disponibles comme opérateurs. L'opérateur peut être sélectionné à l'aide du menu déroulant. Il peut être appliqué directement à des nombres ou à des valeurs de variables. Exemple :

Ce bloc donne le résultat 144 (12^2).

Calculs spéciaux

Ce bloc applique le type de calcul sélectionné dans le menu déroulant au nombre placé en arrière ou à la valeur de la variable placée en arrière. Les opérations de calcul disponibles sont les suivantes :

- racine carrée,
- montant,
- logarithme naturel,
- logarithme décadique,
- fonction exponentielle avec la base e (e^1, e^2,...),
- fonction exponentielle avec la base 10 (10^1, 10^2,...),
- changement de signe (multiplication par -1).

e est le nombre d'Euler. Ce bloc extrait la racine carrée de 16 et place la variable **i** sur le résultat.

Fonctions trigonométriques

Ce bloc fonctionne comme le bloc décrit ci-dessus, sauf que les opérations de calcul sont les fonctions trigonométriques sinus, cosinus, tangens et leurs fonctions inverses. Le nombre spécifié ou la valeur de la

variable spécifiée est alors inséré dans la fonction sélectionnée dans le menu déroulant et le résultat peut être traité dans le programme. Il y a en plus le bloc **arctan2 of X: ... Y: ...**, qui permet, à l'aide de deux nombres réels (à utiliser comme X et Y), d'obtenir une valeur de fonction de l'arctan2 dans la plage de 360°.

Constantes fréquemment utilisées

Ce bloc fonctionne de la même manière que le bloc de chiffres, mais on n'y indique pas le chiffre lui-même. Les constantes fréquemment utilisées (par exemple ?) sont sauvegardées. La constante peut être sélectionnée à l'aide du menu déroulant.

Reste d'une division

Le bloc **Reste de ...** est utilisé pour distribuer le reste d'une division. Ce programme attribue la variable **Reste** à la division 3:2, soit 1 :

Arrondir

Le bloc **Arrondir ...** permet d'arrondir un nombre décimal spécifié ou la valeur d'une variable spécifiée à un nombre entier. Trois options sont disponibles dans le menu déroulant :

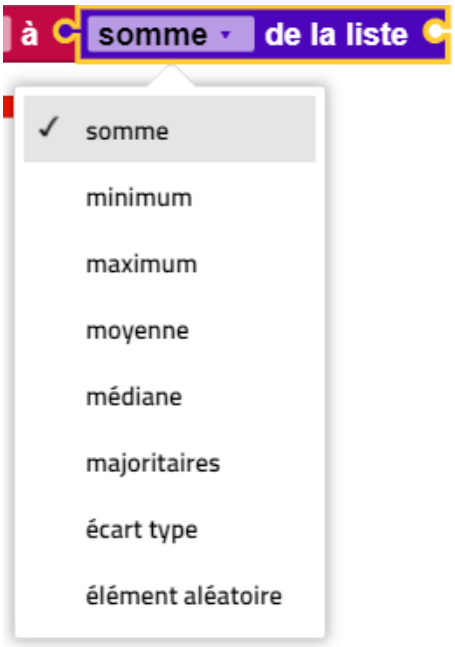
- arrondi commercial avec « arrondi » (par exemple 4,5 à 5)
- arrondi vers le haut avec « arrondi supérieur » (par exemple 5,1 à 6)
- arrondi vers le bas avec « arrondi inférieur » (par exemple 5,9 à 5).

Evaluation de listes

Le bloc **... Liste** permet d'éditer

- Avec « Somme », la somme de toutes les valeurs d'une liste,
- Avec « min », la plus petite valeur d'une liste,
- Avec « max », la plus grande valeur d'une liste,
- Avec « Valeur moyenne », la valeur moyenne de toutes les valeurs d'une liste,
- Avec « Médiane », la valeur médiane d'une liste,
- Avec « Valeur modale », la valeur la plus fréquente d'une liste,
- Avec « Ecart type », l'écart type de toutes les valeurs d'une liste,
- Avec « Valeur aléatoire », une valeur aléatoire d'une liste

. Toutes ces options peuvent être sélectionnées dans le menu déroulant du bloc :



Limiter les valeurs d’entrée

Le bloc **Restriction... de... À...** permet de restreindre les valeurs d’entrée à un intervalle défini. Avant de traiter une valeur d’entrée, on teste si elle se situe dans l’intervalle spécifié. Il y a trois options pour traiter une valeur saisie :

- La valeur est dans l’intervalle, donc elle est transmise telle quelle.
- La valeur est inférieure à la limite inférieure de l’intervalle, donc cette limite inférieure est transmise.
- La valeur est supérieure à la limite supérieure de l’intervalle, donc cette limite supérieure est transmise.

Dans cet exemple, le bloc est utilisé pour limiter la valeur de la variable **Vitesse** aux régimes supportés par le moteur :

Générer des valeurs aléatoires

Les deux blocs, **un nombre aléatoire de ...à ...** et **un nombre aléatoire** donnent une valeur aléatoire. Le bloc **nombre aléatoire de ... à ... donne ainsi** un nombre de l’intervalle défini. Le bloc **nombre aléatoire** donne au contraire une valeur entre 0,0 (inclus) et 1,0 (exclus).

Textes

Voici quelques exemples de textes :

« Objet 1 »
« 12 mars 2010 »
«» (texte vide)
Le texte peut contenir des lettres (en minuscules ou en majuscules), des chiffres, des signes de ponctuation, d'autres symboles et des espaces.

Blocs

Création de texte

Le bloc suivant génère le texte « Bonjour » et l'enregistre dans la variable intitulée **Salutations :** :

Le bloc **Générer du texte à partir de** combine la valeur de la variable **Salutations** et le nouveau texte « Monde » avec le texte « Bonjour le monde ». Notez qu'il n'y a pas d'espace entre les deux textes, car il n'y en avait pas dans les deux textes originaux.

Pour augmenter le nombre de saisies de texte, cliquez sur l'icône (+). Pour supprimer la dernière édition, cliquez sur l'icône (-).

Modification du texte

Le bloc **Joindre à ...** ajoute le texte indiqué à la variable spécifiée. Dans cet exemple, il change la valeur de la variable **Salutations** de « Bonjour » en « Bonjour à vous ! » :

Longueur du texte

Le bloc **Longueur de** compte le nombre de caractères (lettres, chiffres, etc.) qui sont contenus dans un texte. La longueur de « Nous sommes #1 ! » est 12 et la longueur du texte vide est 0.

Vérifier si le texte est vide

Le module **Vide** vérifie si le texte spécifié est vide (la longueur est de 0). Le résultat est **vrai** dans le premier exemple et **faux** dans le second.

Recherche de texte

Ces blocs peuvent être utilisés pour vérifier si un texte est présent dans un autre texte et, dans l’affirmative, où il est présent. Par exemple, on demande la première apparition de « a » dans « Salut », et le résultat est 2 :

Ceci pose la question de la dernière apparition de « a » dans « Salut », ce qui fait également 2 :

Que la première ou la dernière occurrence soit sélectionnée, ce bloc renvoie le résultat 0, car « Salut » ne contient pas de « z ».

Extraction de texte

Extraction d’un caractère unique

Ceci renvoie « b », la deuxième lettre dans « abcde » :

Ceci renvoie « d », l’avant-dernière lettre de « abcde » :

Ceci renvoie « a », la deuxième lettre dans « abcde » :

Ceci obtient « e », la dernière lettre dans « abcde » :

Ceci obtient chacune des 5 lettres « abcde » avec la même probabilité :

Aucun d'eux ne modifie le texte extrait.

Extraction d'une zone de texte

Le bloc **dans le texte ... Fournit la chaîne de caractères** permet d'extraire une zone de texte qui commence avec :

- Lettre #
- Lettre # de la fin
- Première lettre

et termine par :

- Lettre #
- Lettre # de la fin
- Dernière lettre

Dans l'exemple suivant, « abc » est extrait :

Mise en majuscules/minuscules du texte

Ce bloc génère une version du texte d'entrée qui est soit

- en MAJUSCULES (toutes les lettres en majuscules) ou
- en minuscules (toutes les lettres en minuscules) ou
- Substantive (première lettre en majuscule, les autres lettres en minuscules).

Le résultat du bloc suivant est « BONJOUR ».

Les caractères non alphabétiques ne sont pas concernés. Remarquez que ce bloc ne fonctionne pas sur des textes dans des langues sans majuscules et minuscules, comme le chinois par exemple.

Ajuster (supprimer) les espaces

Supprimer le bloc suivant, en fonction de ce qui est paramétré dans le menu déroulant (petit triangle), espaces :

- au début du texte
- à la fin du texte
- des deux côtés du texte

Le résultat du bloc suivant est « Bonjour vous ».

Les espaces au milieu du texte ne sont pas concernés.

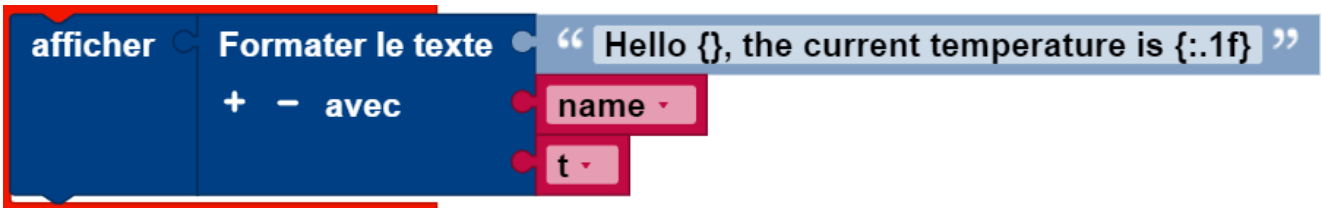
Éditer du texte

Le bloc **Editer** a pour effet d'éditer la valeur de saisie dans la fenêtre de la console :

En aucun cas, il ne sera envoyé à l’imprimante, comme son nom le laisse supposer.

Éditer du texte avec formatage

Le bloc **Formater le texte** permet d’éditer des sorties de texte formatées avec un contenu variable. Tous les caractères de remplacement **{}** dans le texte sont remplacés par le contenu des variables jointes après le texte. Un formatage peut être donné entre parenthèses. Le formatage **{:.1f}** ne donne par exemple que la première décimale du nombre de virgules dans la variable **t**.



Listes

Comme dans le langage courant, ROBO Pro Coding contient une liste d'éléments ordonnés, tels qu'une liste « à faire » ou une liste d'achats. Les éléments d'une liste peuvent être de n'importe quel type et la même valeur peut apparaître plusieurs fois dans une liste.

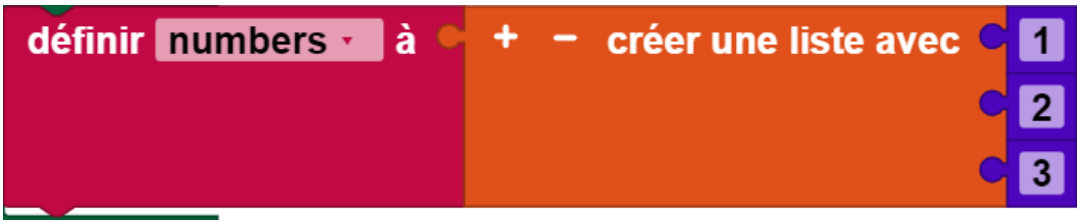
Création d'une liste

Créer une liste avec

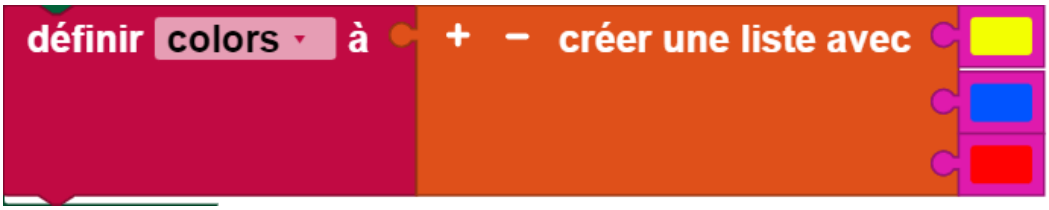
Le bloc **Créer une liste avec** permet de saisir les valeurs initiales dans une nouvelle liste. Dans cet exemple, une liste de mots est créée et stockée dans une variable appelée **lettres** :

Nous appelons cette liste [« alpha », « bêta », « gamma »].

Ceci montre la création d'une liste de **chiffres** :



Pour créer une liste de **couleurs** :



Il est moins courant, mais il est possible de dresser une liste de valeurs de différents types :

Modifier le nombre d'entrées

Pour modifier le nombre d'entrées, cliquez ou appuyez sur l'icône de la roue dentée. Ceci ouvre une nouvelle fenêtre. Vous pouvez faire glisser des sous-blocs d'éléments du côté gauche de la fenêtre vers le bloc de liste du côté droit pour ajouter une nouvelle entrée. :

Si le nouvel élément a été ajouté dans cet exemple, il peut être ajouté n'importe où. De même, les sous-blocs d'éléments indésirables peuvent être tirés vers la gauche à partir du bloc de liste.

Créer une liste d'éléments

Le bloc **Créer une liste avec un élément** permet de créer une liste contenant le nombre indiqué de copies d'un élément. Par exemple, les blocs suivants ajoutent des **mots** à la liste [« très », « très », « très »].

Contrôle de la longueur d'une liste

Vide

La valeur d'un bloc **vide** est **vraie** si sa saisie est la liste vide et **fausse** si c'est autre chose. Cette entrée est-elle **vraie** ? La valeur du bloc suivant serait **fausse** car la variable Couleurs n'est pas vide : elle comporte trois éléments.

Notez la similitude avec le bloc **vide** pour le texte.

Longueur de

La valeur du bloc **Longueur de** est le nombre d'éléments de la liste utilisée comme entrée. La valeur du bloc suivant serait par exemple 3, car la**couleur** comporte trois éléments :

Notez que le bloc **Longueur de** indique le nombre d'éléments contenus dans la liste et non le nombre d'éléments différents contenus dans la liste. Par exemple, ce qui suit la valeur 3, bien que les **mots** soient composés de trois copies du même texte :

Notez la similitude avec le bloc **Longueur de** pour le texte.

Recherche d'éléments dans une liste

Ces blocs trouvent la position d'un élément dans une liste. L'exemple suivant a la valeur 1 parce que la première occurrence de "très" figure en tête de la liste de mots ([« très », « très », « très »]).

Le résultat de ce qui suit est 3, parce que la dernière occurrence de « très » en **mots** est en position 3.

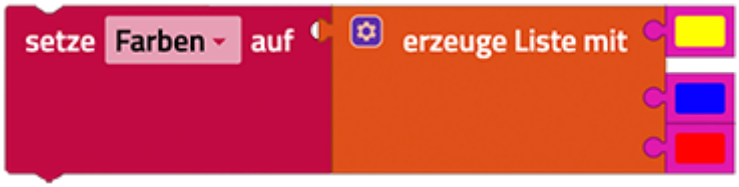
Si l'élément n'apparaît nulle part dans la liste, le résultat est 0, comme dans cet exemple :

Ces blocs se comportent de la même manière que les blocs permettant de trouver des lettres dans le texte.

Récupération d'éléments d'une liste

Récupération d'un élément unique

Rappelez-vous de la définition de la liste **Couleurs** :



Le bloc suivant reçoit la couleur bleu parce qu'il est le deuxième élément de la liste (compté à partir de la gauche) :

Celui-ci reçoit du vert parce qu'il est le deuxième élément (compté depuis l'extrémité droite) :

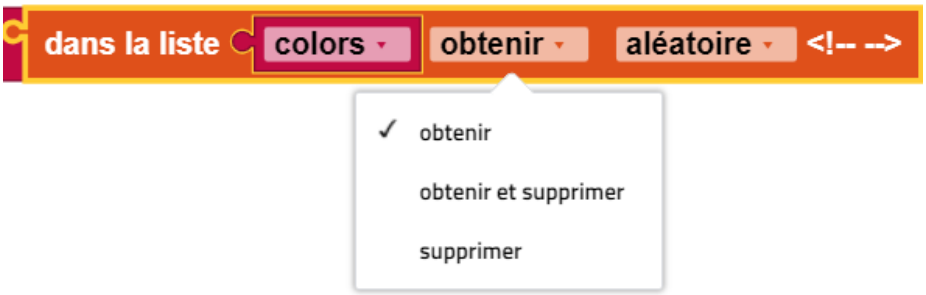
Celui-ci reçoit le premier élément, rouge :

Celui-ci reçoit le dernier élément, jaune :

Ceci sélectionne au hasard un élément de la liste en renvoyant avec la même probabilité l'un des éléments rouge, bleu, vert ou jaune.

Récupération et suppression d’un élément

Le menu déroulant permet de modifier le bloc **Récupérer dans la liste ...** dans le bloc **Récupérer dans la liste ... et supprimer** qui fournit le même résultat, mais modifie également la liste :



Cet exemple place la variable **Première lettre** sur « alpha » et laisse les lettres restantes ([« beta », « gamma »]) dans la liste.

Supprimer une entrée

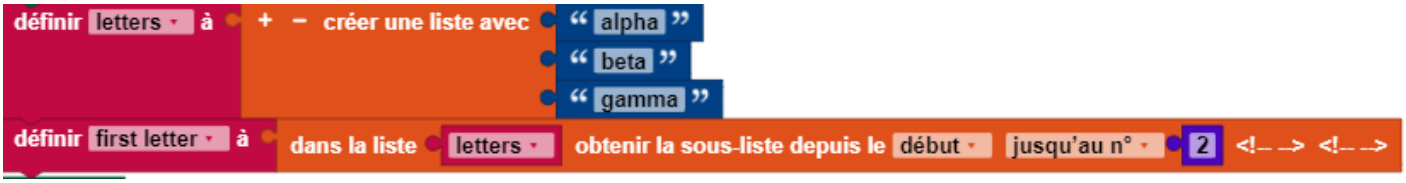
Si vous sélectionnez **Supprimer** dans le menu déroulant, le nez disparaît du bloc à gauche :

Cela supprime le premier élément des **lettres**.

Afficher une sous-liste

Le boc **Dans la liste ... Afficher la sous-liste** ressemble au bloc **Récupérer dans la liste ...** à la différence qu’il extrait une sous-liste et pas un élément individuel. Il existe plusieurs options pour indiquer le début et la fin de la sous-liste :

Dans cet exemple, une nouvelle liste de **première lettre** est établie. Cette nouvelle liste comporte deux éléments : [« alpha », « bêta »].



Remarquez que ce bloc ne change pas la liste d'origine.

Ajout d'éléments à une liste

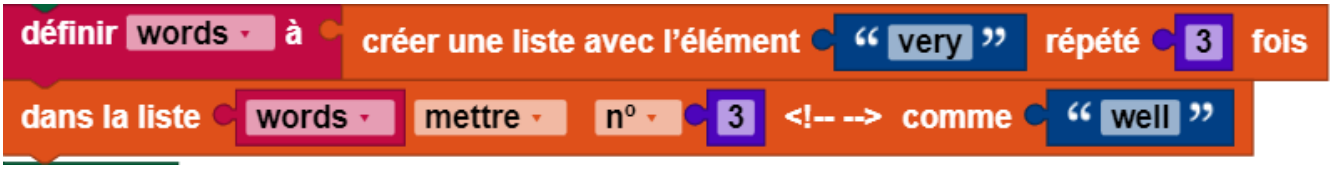
Remplacer les éléments dans une liste

Le bloc **Remplacer dans la liste ...** remplace l'élément à un emplacement défini d'une liste par un autre élément.

Vous trouverez dans la section précédente la signification des différentes options de menu déroulant.

L'exemple suivant fait deux choses :

1. La liste **mots** est composée de 3 éléments : [« très », « très », « très »].
2. Le troisième élément de la liste est remplacé par « bien ». La nouvelle valeur des **mots** est [« très », « très », « bien »]

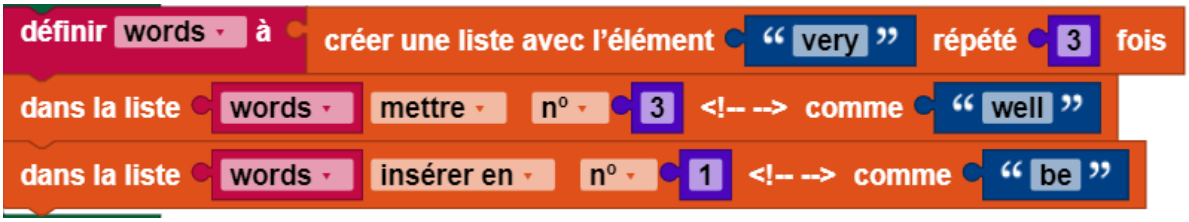


Insérer des éléments à un emplacement donné dans une liste

Le bloc **Ajouter dans la liste ... pour** est affiché via le menu déroulant du bloc **Remplacer dans la liste ...** :

Il insère un nouvel élément dans la liste à l'emplacement indiqué, devant l'élément précédemment présent à cet emplacement. L'exemple suivant (tiré d'un précédent exemple) fait trois choses :

1. La liste **mots** est composée de 3 éléments : [« très », « très », « très »].
2. Le troisième élément de la liste est remplacé par « bien ». La nouvelle valeur des **mots** est donc [« très », « très », « bien »].
3. Le mot « être » est ajouté au début de la liste. La valeur finale des **mots** est donc [« être », « très », « très », « bien »].



Diviser les chaînes et fusionner les listes

Créer une liste à partir d'un texte

Le module **Créer une liste à partir de texte** décompose le texte spécifié en parties à l'aide d'un caractère de délimitation :

Dans l'exemple ci-dessus, une nouvelle liste contenant trois morceaux de texte est renvoyée : « 311 », « 555 » et « 2368 ».

Créer un texte à partir d'une liste

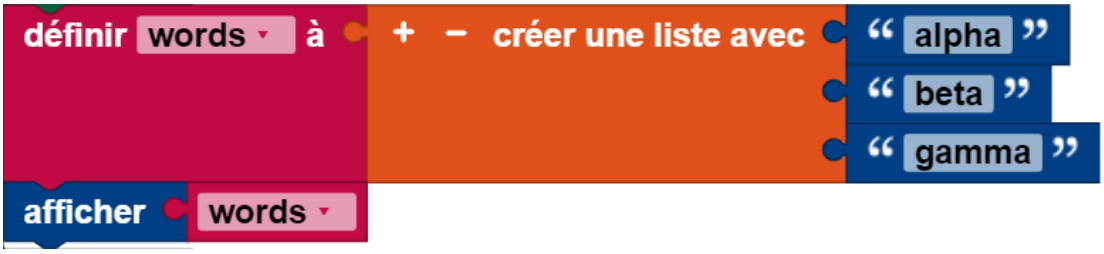
Le module **Créer un texte à partir d'une liste** fusionne une liste en un seul texte à l'aide d'un séparateur :

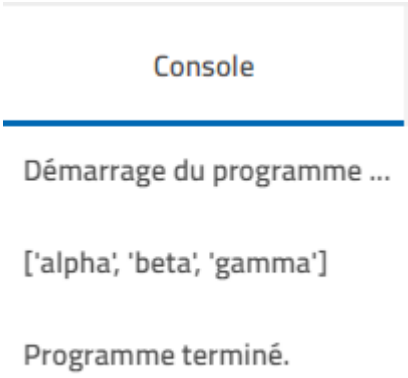
Dans l'exemple ci-dessus, un nouveau texte est renvoyé avec la valeur : « 311-555-2368 ».

Blocs apparentés

Impression d'une liste

Le module **Imprimer** de la catégorie Texte peut émettre des listes. Le résultat du programme suivant est l'édition de console représentée :





Exécuter quelque chose pour chaque élément dans une liste

Le bloc **pour chacun** dans la catégorie Commande exécute une opération pour chaque élément dans une liste. Par exemple, ce bloc imprime individuellement chaque élément de la liste :

Ceci ne supprime pas les éléments de la liste d’origine.

Voir aussi les exemples pour les blocs de rupture de boucle.

Utilisation

La catégorie Utilisation comprend, dans ROBO Pro Coding, les blocs suivants :

- Sélection de couleur
- Attendre
- Code Python
- Démarrer
- Exécution fonctionnelle

Sélection de couleur

Ce bloc sert de valeur d'entrée si une couleur est demandée (par exemple, lors de l'ajustement des couleurs par la caméra). En cliquant ou en appuyant sur la couleur, il est possible de sélectionner l'une des 70 couleurs parmi une palette de couleurs.

Attendre

Attendre que le temps soit écoulé

Le bloc **Attendre[] ...** empêche le programme de continuer pendant le temps d'attente spécifié. Le menu déroulant (petit triangle) permet de sélectionner l'unité de temps et la longueur de pause souhaitée dans le champ de saisie derrière.

Attendre avec une condition

Pour le bloc **Attendre jusqu'à**, la pause n'est pas liée au temps, mais à la satisfaction d'une condition (par exemple, si un bouton-poussoir est enfoncé). La condition est ajoutée au bloc **Attendre jusqu'à**.

Code Python

Si vous souhaitez intégrer le code python existant dans le ROBO Pro Coding, vous pouvez l'insérer dans le bloc de **code python**. Le programme exécutera tout ce qui a été écrit dans le bloc dans Python.

Démarrer

Le bloc **Démarrer si** est associé à une condition. Ce n'est que lorsque cette condition est remplie que le programme se trouvant dans le bloc démarre.

Exécution fonctionnelle

La **fonction exécutée dans un fil** permet d'exécuter la fonction sélectionnée dans un fil séparé. Cette mesure peut, dans certains cas, permettre à un programme de continuer à répondre aux demandes et d'être exécuté plus rapidement.

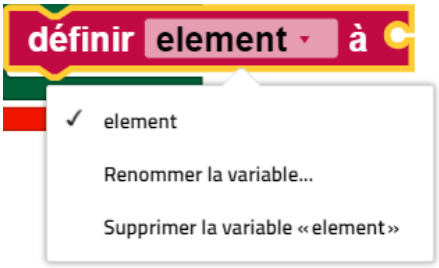
Variables

Nous utilisons le terme variable tel qu'il est utilisé en mathématiques et dans d'autres langages de programmation : une valeur nommée qui peut être modifiée (variée). Les variables peuvent être produites de différentes manières.

- Certains blocs comme **Compter avec** et **Pour chacun** utilisent une variable et définissent leurs valeurs. Un terme informatique traditionnel pour ces variables est une variable de boucle.
- Des fonctions définies par l'utilisateur (également appelées « procédures ») peuvent définir des entrées, générant ainsi des variables qui ne peuvent être utilisées que dans cette fonction. Ces variables sont traditionnellement appelées « paramètres » ou « arguments ».
- Les utilisateurs peuvent à tout moment modifier des variables avec le bloc **Définir**. Celles-ci sont traditionnellement appelées « variables globales ». Ils sont utilisables partout dans le code de ROBO Pro Coding.

Menu déroulant

Lorsque vous cliquez sur l'icône de menu déroulant (petit triangle) d'une variable, le menu suivant s'affiche :



Le menu propose les options suivantes.

- L'affichage des noms de toutes les variables existantes définies dans le programme.
- « Renommer la variable... », c'est-à-dire changer le nom de cette variable à chaque fois qu'elle apparaît dans le programme (la sélection de cette option ouvre une requête pour le nouveau nom)
- « Supprimer la variable ... », c'est-à-dire supprimer tous les blocs qui font référence à cette variable à l'endroit où elle apparaît dans le programme.

Blocs

Définir

Le bloc **Définir** assigne une valeur à une variable et définit la variable si elle n'existe pas encore. Par exemple, la valeur de la variable **âge** est fixée à 12 :

Afficher

Le bloc **Afficher à partir de** renvoie la valeur stockée dans une variable sans la modifier :

Il est possible, mais ce n’est pas une bonne idée, d’écrire un programme dans lequel un bloc **Afficher à partir de** se produit sans un bloc Définir précédent correspondant.

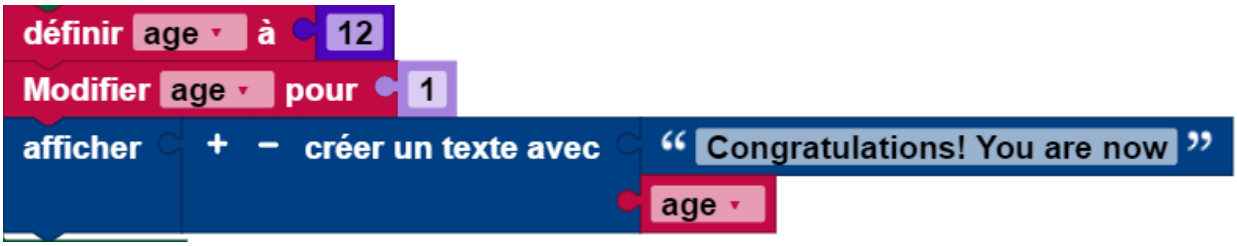
Modifier

Le bloc **Modifier** ajoute un nombre à une variable.

Le bloc **Modifier** est une abréviation pour la structure suivante :

Exemple

Regardez l’exemple de code suivant :



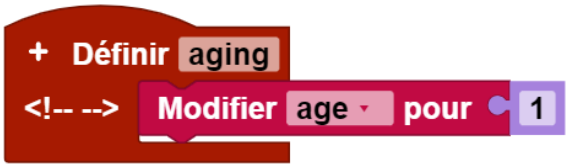
La première série de blocs génère une variable appelée **Âge** et **définit**sa valeur initiale au nombre 12. La seconde série de blocs **appelle** la valeur 12 **à partir de**, y ajoute 1 et enregistre la somme (13) dans la variable. Sur la dernière ligne, le message suivant est édité : « Félicitations ! Vous avez 13 ans à présent ».

Fonctions

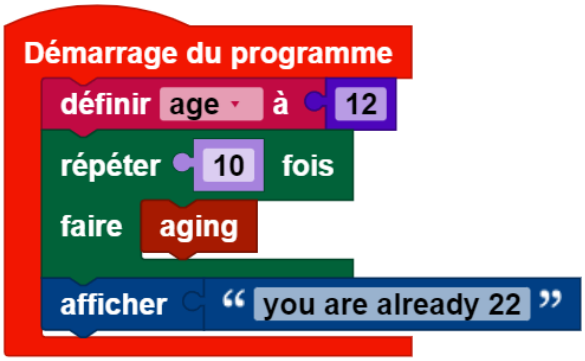
Les fonctions servent à rendre des parties du code réutilisables et donc à structurer le code dans son ensemble. Si vous remplissez un bloc de fonctions, un nouveau bloc, portant le même nom que ce bloc de fonctions, apparaît dans le menu des fonctions. Il est désormais possible de n'insérer dans le programme principal que le bloc avec le nom de la fonction. Lorsque le programme est exécuté, ce bloc passe au code dans la fonction du même nom et le traite.

Fonction simple

Le bloc de fonction simple permet de générer une fonction qui porte le nom saisi dans la zone de texte. La fonction peut contenir autant de variables que vous le souhaitez qui peuvent être ajoutées par l'icône de la roue dentée. Cette fonction **Vieillissement** ajoute 1 à la variable **Âge** :



La fonction peut alors être utilisée dans le programme principal :



Fonction avec valeur de retour

Ce bloc permet de créer une fonction avec valeur de retour. Cette valeur peut ensuite être réutilisée dans le programme principal. Voici un exemple :