

Langage AL

AL est un nouveau langage (signifiant Algoid Language). Il a été inspiré par java, python, smalltalk et lua, mais **il n'est pas** ces langages.

L'intérêt d'Algoid Language est d'être simple, mais complet. Son but est d'aider à apprendre comment programmer et par la suite comment utiliser les différents paradigmes :

Impératif, procédural, fonctionnel et récursif, objet orienté prototype (héritage multiple) et programmation orientée aspect.

Il intègre des idiomes puissants comme le protocole meta-objet (inspiré de python) et cascade (inspiré de smalltalk).

Et parce qu'un jour vous allez, je l'espère, travailler avec un langage standard de l'industrie (très certainement proche du c et du java) AL a été conçu aussi proche de leurs syntaxes que possible.

Cette syntaxe avec ses accolades et ses symboles peut sembler très compliquée, mais on s'y fait vite et elle a l'avantage d'être concise.

En outre, les idiomes fondamentaux d'AL sont :

- Toutes les fonctions sont des expressions.
- Tous les objets sont des expressions.
- Toutes les expressions sont des objets.
- Donc toutes les fonctions sont des meta-fonctions et tous les objets sont des meta-objets.

Le reste est inspiré des différents langages énoncés plus haut.

Index

Language structure

instruction
block
expression
statement

Language primitives

while
do
loop
for
if
function
lambda
object
array

[dynamic] type

.getType
.is
.isNull
.ifNull
.ifNotBreak

.ifNullBreak
.equals
.toString
.add
.addAll
.onChanged

[dynamic] void

[dynamic] boolean

.not
.and
.or
.xor
.ifTrue
.ifFalse
.whileDo
.untilDo

[dynamic] number

.isInteger
.isReal
.isNan
.isInfinite

.toInteger
.minus
.increment
.decrement
.addition
.subtract
.multiply
.divide
.modulo
.greaterThan
.smallerThan
.greaterOrEquals
.smallerOrEquals
.between
.decodePoint
.loopFor

[dynamic] string

.isEmpty
.length
.getChar
.contains
.concat
.indexOf
.count
.upper
.lower
.append

- .appendSep
- .subString
- .subStringOf
- .replace
- .replaceAt
- .remove
- .split
- .splitAt
- .trim
- .create
- .encodePoint
- .each

[dynamic] array

- .isEmpty
- .length
- .getItem
- .setItem
- .getFirst
- .getLast
- .clone
- .clear
- .contains
- .remove
- .pop
- .indexOf
- .count
- .swap

.decodePoint
.find
.create
.each
.eachOnRow
.eachOnCol
.eachItem
.filter
.sort
.min
.max
.join
.merge

[dynamic] function

.parameterExists
.getParametersNames
.setParameter
.setParameters
.removeParameter
.concat
.decorate

[dynamic] object

.clone
.attributeExists

.getAttributesNames
.isA
.setAttribute
object.toString
.merge
.removeAttribute

Racine de l'API

print

al

al.allObjects
al.allLocalObjects
al.clock

al.order

al.order.ascending
al.order.descending
al.order.random
al.order.reverse

al.combine

al.combine.sum
al.combine.product
al.combine.concat

al.types

al.types.VOID
al.types.BOOL
al.types.NUMBER
al.types.STRING
al.types.ARRAY
al.types.FUNCTION
al.types.OBJECT

util

util.eval
util.wait
util.pulse
util.notice
util.pullOn
util.clearTasks
util.log
util.warn
util.err

math

math.E
math.PI
math.abs
math.acos
math.aim
math.asin
math.atan
math.ceil
math.cos
math.dbl
math.diagonal
math.exp
math.floor
math.log
math.max
math.min
math.pow
math.random
math.round
math.sin
math.sqrt
math.tan

ui

ui.message
ui.fullScreen
ui.halfScreen

ui.miniScreen
ui.showText
ui.showAlgo
ui.showScope
ui.showLog
ui.clearLog
ui.showMenu
ui.hideMenu

text

text.clear
text.output
text.inputText
text.inputNumber

algo.color

algo.color.TRANSP
algo.color.BLACK
algo.color.DARK_BLUE
algo.color.DARK_GREEN
algo.color.DARK_CYAN
algo.color.DARK_RED
algo.color.DARK_MAGENTA
algo.color.BROWN
algo.color.GRAY
algo.color.DARK_GRAY

algo.color.BLUE
algo.color.GREEN
algo.color.CYAN
algo.color.RED
algo.color.MAGENTA
algo.color.YELLOW
algo.color.WHITE

algo

algo.setColor
algo.setBgColor
algo.setAlpha
algo.setFontSize
algo.setStroke
algo.setStack
algo.getX
algo.getY
algo.getAngle
algo.getTop
algo.getBottom
algo.getLeft
algo.getRight
algo.getWidth
algo.getHeight
algo.show
algo.hide
algo.clear
algo.autoClear

algo.removeFirst
algo.removeLast
algo.goTo
algo.lineTo
algo.go
algo.jump
algo.turnRight
algo.turnLeft
algo.rotateTo
algo.circle
algo.disc
algo.square
algo.box
algo.rect
algo.plane
algo.oval
algo.platter
algo.path
algo.poly
algo.curve
algo.curvedPoly
algo.text
algo.onClick
algo.onRelease
algo.onMove
algo.onDrag
algo.onWheel
algo.onKey
algo.onTap
algo.onTouch

- algo.onUp
- algo.onGravity
- algo.onAcceleration
- algo.onGyroscope
- algo.onOrientation
- algo.onProximity
- algo.onTemperature
- algo.onLight

algo.stamp

- algo.stamp.id
- algo.stamp.width
- algo.stamp.height
- algo.stamp.clone
- algo.stamp.delete
- algo.stamp.draw

instruction

L'élément unitaire du langage AL est l'instruction.

En AL, le point-virgule est totalement optionnel. Il existe parce qu'AL est inspiré du C et qu'en C il est obligatoire. Donc vous pouvez écrire votre programme avec ou sans. Préférez avec parce que la majorité des langages le demande. Ca devient une habitude.

SYNTAX : `instruction [';']`

Les instructions en AL se divisent en deux familles : les expressions (ou valeurs du programme) et les instructions de contrôle (issu de l'impératif).

block

Un bloc est une suite d'instructions encapsulées dans une portée locale.

La portée signifie que toute variable déclarée à l'intérieur n'est accessible que depuis cette portée et qu'elle sera détruite dès la sortie de celle-ci.

SYNTAX : `'{' {[instruction]} '}'`

expression

Toute valeur du langage s'appelle une expression. AL comprend plusieurs types d'expressions : VOID, BOOLEAN, NUMBER, STRING, FUNCTION, OBJECT, ARRAY.

AL est un langage dynamique, il ne nécessite pas de manipulation des types. Le trans-typage ce fait automatiquement, selon le besoin et les opérateurs utilisés.

SYNTAX :
`nil`
`true | false`
`nan | infinity | number`
`number`
`''' string '''`
`function | object`
`ident '[' expr '']'`

Les opérateurs binaires sont les suivant :

symbole	déscription
&&	(booléen) et : true (vrai) et true (vrai) donne true (vrai)
	(booléen) ou : true (vrai) ou false (faux) donne true (vrai)
==	est égale
!=	est différent
<	est plus petit que
>	est plus grand que
<=	est plus petit ou égale à
>=	est plus grand ou égale à
+	(number) plus
-	(number) moins
*	(number) multiplie
/	(number) divise
%	(number) modulo

..	(string) concatène
->	(function) puis

Les opérateurs unaires sont les suivants :

symbole	description
!	(boolean) not (non) : !true (non vrai) donne false (faux)
-	(number) moins : -i est équivalent à 0 - i
++	incrementation : i++ équivaut à i = i + 1 ou i += 1
--	decrement : i-- équivaut à i = i - 1 ou i -= 1

exemple:

insert into ide

```
1      ui.showLog();
2      set i = 100 / 2 + 4 * -2;
3      i++;
4      set b = true || false;
5      util.log("result : " .. i .. " and 1 or 0 : " .. b);
6
7
```


AL supporte également plusieurs valeurs constantes : true, false, nil, nan, infinity :

- true et false sont les deux valeurs booléenne
- nil représente la valeur null
- nan signifie "not a number" (résultat d'une opération non autorisée)
- infinity représente un nombre infini (utiliser -infinity pour la valeur négative)

statement

Déclare une variable dans la portée courante.

Les variables déclarées peuvent être utilisées par la suite dans la même portée ou dans une portée imbriquée la portée courante.

SYNTAX : 'set' ident [symbol expr] ';'

Les symboles sont les suivants :

symbole	déscription
=	égale $i = n$;
+=	plus égale, équivaut à $i = i + n$
-=	moins égale, équivaut à $i = i - n$
*=	multiplie égale, équivaut à $i = i * n$
/=	divise égale, équivaut à $i = i / n$

<code>%=</code>	modulo égale, équivaut à <code>i = i % n</code>
<code>..=</code>	concatène égale (pour les strings), équivaut à <code>s = s .. t</code>
<code>->=</code>	puis égale (pour les fonctions), équivaut à <code>f = f -> g</code>

La déclaration est elle même une expression.

exemple:

insert into ide

```
1      ui.showLog();
2      set i = 10;
3      i += 20;
4      util.log("result: " .. i);
5
6
```

while

While est la première instruction impérative. Elle boucle tant que sa condition renvoie true (vrai).

SYNTAX : `'while' '(' condition ')' block`

exemple:

insert into ide

```
1 ui.showLog();
2 set i = 10;
3 while (i-- > 0) {
4   util.log("result: " .. i);
5 }
6
7
```

do

Boucle jusqu'à ce que la condition renvoie false (faux).

SYNTAX : 'do' block 'until' '(' condition ')'

exemple:

insert into ide

```
1 ui.showLog();
2 set i = 10;
3 do {
4   util.log("result: " .. i);
5 } until (i-- <= 0)
6
7
```

loop

Boucle le nombre de fois indiqué par le paramètre limite.

SYNTAX : 'loop' (limit) block

exemple:

insert into ide

```
1      ui.showLog();
2      set a = 0;
3      loop (10) {
4          util.log("loop: " .. a);
5          a++;
6      }
```

for

La boucle for initialise une variable et répète le bloc d'instruction tant que la progression de la variable respecte la condition.

SYNTAX : 'for' ([initialization] ';' [condition] ';' [progression]) block

exemple:

insert into ide

```
1      ui.showLog();
2      for (set i = 0; i < 10; i++) {
3          util.log("loop: " .. i);
4      }
```

if

IF exécute un bloc d'instruction si sa condition est true (vrai) sinon, c'est le bloc Else qui est exécuté. Elseif exécuté si la condition précédente n'est pas true (vrai) et si la sienne l'est.

SYNTAX : 'if' '(' condition ')' block {'elseif' '(' condition ')' block} ['else' block]

exemple:

insert into ide

```
1      ui.showLog();
2      for (set i = 0; i<4; i++) {
3          if (i == 0) {
4              util.log("i is 0");
5          } elseif (i == 1) {
6              util.log("then 1");
7          } else {
8              util.log("then others");
9          }
10     }
11
12 }
```

En langage AL, IF est aussi un expression. Elle peut être utilisée pour décrire une valeur conditionnelle.

exemple:

insert into ide

```
1 ui.showLog();
  set i = 0;
2 set s = if (i==0) "zero" elseif (i == 1) "one" else "other";
  util.log ("if i=" .. i .. ", s=" .. s);
3
4
5
6
```

function

Définit un bloc d'instruction réutilisable dans une portée propre avec des paramètre en entrée et un paramètre de sortie.

La particularité d'AL, c'est que ses fonctions (ainsi que ses objets) sont considérés comme des valeurs. Ce sont des expressions et donc, peuvent se terminer par un point-virgule.

SYNTAX : 'function' ['(' [arg [{' , ' arg}]] ')'] '{' instruction '}'

exemple:

insert into ide

```
1 set f = function (x) {
2   ui.message("x parameter is " .. x);
3 };
4 f (10);
5
6
```

lambda

Une expression lambda est une fonction avec une écriture simplifiée. Elle ne peut-être déclarée que comme paramètre d'une autre fonction. Elle a été créée pour simplifier l'écriture des foncteurs de la programmation fonctionnelle.

SYNTAX : ['(' [arg [' , ' arg]] ')] '{' instruction '}'

exemple:

insert into ide

```
1      util.pulse({  
2          algo.go(10);  
3      }, 50);  
4  
5
```

object

Définit un ensemble de déclarations réutilisables dans leur portée propre.
Les déclarations peuvent être : des attributs (variables), des méthodes (fonctions) et des objets imbriqués.

SYNTAX : 'object' '(' {parent [' , ' parent]} ')' '{' declarations '}'

exemple:

insert into ide

```
1 ui.showLog ();
2 set o = object () {
3   set a = 0;
4   set b = "my b attribut";
5 };
6
7 util.log ("o.a = " .. o.a);
8 util.log ("o.b = " .. o.b);
9
```

Toute fonction déclarée dans un objet est appelé méthode. Les méthodes constituent les comportements de l'objet.

Une méthode désignée pour accéder à un attribut est appelé un accesseur.

Une méthode désignée pour modifier un attribut est appelé un mutateur.

exemple:

[insert into ide](#)

```
1 set o = object () {
2   set a = 0;
3   // the a setter
4   set setA = function(a) {
5     this.a = a;
6   };
7   // the a getter
8   set getA = function() {
9     return this.a;
10  };
11  // a method
12  set doubleA = function() {
13    this.a = this.a * this.a;
14  };
15 }
```



```

16         };
17
18     };
19
20     o.setA(2);
21     ui.message("o.a parameter is " .. o.getA());
22     o.doubleA();
23     ui.message("and its double is " .. o.getA());

```

Les objets peuvent être dupliqués en conservant la même structure interne. Cela s'appelle cloner un objet. En AL les objets sont construits et peuvent être clonés. Il existe deux façons pour cloner; l'instruction `new` duplique l'objet et ses états actuels et la méthode `clone`. Cette dernière permet d'attribuer de nouvelles valeurs aux états de l'objet en les lui passant en paramètre.

exemple:

[insert into ide](#)

```

1     ui.showLog();
2
3     set o = object () {
4         set a = 7;
5         set b = "my attribute b";
6         set toString = function () {
7             return "o {a=" .. a .. ", b=" .. b .. "};"
8         };
9     };
10    set p = o.clone (8, "another parameter");
11    util.log (o);
12    util.log (p);
13

```

This est une référence de l'objet à lui même.

exemple:

insert into ide

```
1      ui.showLog();
2      set o = object () {
3          set a;
4          set setA = function (a) {
5              this.a = a;
6          };
7          set getA = function () {
8              return this.a;
9          };
10     };
11     o.setA (7);
12     util.log (o.getA());
13
14
```

Supers[n] est un tableau qui permet d'accéder à la liste des super objets (héritage multiple).

exemple:

insert into ide

```
1      set q = object() {
2          set ret7 = function () {
3              return 7;
4          };
5      };
6
7
```

```

8      set p1 = object() {};
9
10     set p2 = object(q) {};
11
12     set o = object (p1, p2) {
13         set test = function () {
14             return this.supers[1].supers[0].ret7();
15         };
16     };
17
18     ui.message("o.test result is " .. o.test());
19

```

array

Définit un tableau d'éléments ou un dictionnaire (liste associative) indexé par n'importe quelle expression. Lors de la définition de tableaux imbriqués, le mot clé 'array' n'est obligatoire que pour le premier tableau (le root). Il devient optionnel ensuite.

SYNTAX :

item : [expr ':'] expr

array : ['array']1 '{' item [{ ',' item }] '}'

exemple:

insert into ide

```

1      ui.showLog();
2      set a = array {7, 8, 9, 10, 11};
3      for (set i=0; i < a.length(); i++) {
4

```

```

5      util.log("a[" .. i .. "] is " .. a[i]);
6  }
7  // or more elegant
8  a.each (function (item, index) {
9      util.log("a[" .. index .. "] is " .. item);
10 });
11

```

Les tableaux peuvent être utilisés comme des tableaux associatifs.

Ils gardent leur comportement de tableau, mais leurs valeurs sont référencées par une expression clé :

exemple:

insert into ide

```

1      ui.showLog();
2      set a = array {"a" : 7, "b" : 8, "c" : 9, "d" : 10, "e" : 11};
3      a.each (function (item, index, key) {
4          util.log("a[" .. key .. "] is " .. item);
5      });
6
7      util.log ("Find a[c] = " .. a["c"]);
8
9

```

Le framework Algoid

Un langage c'est une somme de structures de contrôle et de primitives qui permettent de manipuler les états. Le framework est une librairie additionnelle (composé de fonctions et d'objets) qui met à disposition une somme d'outils.

Une particularité du langage AL est que chaque fois qu'une valeur est utilisée, AL met à disposition des méthodes pour agir sur elle.

Cet idiome a été inspiré de Smalltalk (cascade) et de Python (magic methods).

type

En AL, toutes les variables héritent de l'objet type. Lorsqu'une variable est déclarée, AL crée un objet approprié qui est aussi un objet type. Type a des propriétés et des méthodes qui sont donc communes à tous les types de variables.

property

*.getType () [insert into ide](#)

Renvoie le type AL de la donnée. Voir [Types AL](#) pour la référence complète.

*.is (type) [insert into ide](#)

Vérifie que la donnée soit du type indiqué comme paramètre. Voir [Types AL](#) pour la référence complète.

`*.isNull ()`

insert into ide

Vérifie si le type de la variable est VOID (égale à nil).

`*.ifNull (value)`

insert into ide

Vérifie si le type de la variable est VOID (égale à nil). Si tel est le cas, la fonction retourne la valeur en paramètre.

`*.ifNotBreak (type)`

insert into ide

Vérifie que la donnée soit du type indiqué comme paramètre. Si tel n'est pas le cas, la fonction renvoie nil et termine la chaîne d'appel de fonctions. Voir [Types AL](#) pour la référence complète.

`*.ifNullBreak ()`

insert into ide

Vérifie si le type de la variable est VOID (égale à nil). Si tel est le cas, la fonction retourne la valeur en paramètre.

`*.equals (b)`

insert into ide

Vérifie si boolean est égale à b.

```
*.toString ()
```

insert into ide

Renvoie l'expression de la donnée sous forme de texte. Dans le cas d'un type complexe (comme array, fonction ou object), renvoie l'ensemble des valeurs du type sous forme de texte.

method

```
*.add (item [, index])
```

insert into ide

Index est un paramètre optionnel.

Si index est absent, il crée un tableau avec la valeur en première position et y ajoute un élément à sa fin. Sinon, il crée le tableau et ajoute l'élément à la position indiquée.

```
*.addAll (array [, index])
```

insert into ide

Index est un paramètre optionnel.

Si index est absent, il crée un tableau avec la valeur en première position et y ajoute tous les éléments à la fin. Sinon, il crée le tableau et ajoute les éléments à la position indiquée.

event

```
*.onChanged ( function(value){} )
```

insert into ide

Un évènement qui se lève chaque fois que la valeur est modifiée dans le programme.

void

Quand une variable sans valeur est déclaré, AL créer un objet VOID. Cet objet à des propriétés et des méthodes. Voir [l'objet Type](#) pour les méthodes et les propriétés communes aux types.

boolean

Quand un booléen est déclaré, AL créer un objet. Cet objet à des propriétés et des méthodes. Voir [l'objet Type](#) pour les méthodes et les propriétés communes aux types.

property

`boolean.not ()` [insert into ide](#)

Retourne la valeur inverse du booléen.

`boolean.and (b)` [insert into ide](#)

Retourne le résultat de l'opération boolean && b.

`boolean.or (b)` [insert into ide](#)

Retourne le résultat de l'opération boolean || b.


```
boolean.xor (b)
```

```
insert into ide
```

Retourne le résultat de l'opération $(\text{boolean} \parallel !b) \&\& (!\text{boolean} \parallel b)$. En d'autres termes, boolean différent de b

```
boolean.isTrue (function () {})
```

```
insert into ide
```

Exécute la fonction en paramètre si la valeur booléenne est vraie.
Équivalent fonctionnel de l'instruction If.

```
boolean.isFalse (function () {})
```

```
insert into ide
```

Exécute la fonction en paramètre si la valeur booléenne est fausse.
Équivalent fonctionnel de l'instruction Else.

```
boolean.untilDo (function () {})
```

```
insert into ide
```

Exécute la fonction en paramètre tant que la valeur booléenne est vraie.
Équivalent fonctionnel de l'instruction While.

```
boolean.untilDo (function () {})
```

```
insert into ide
```

Exécute la fonction en paramètre tant que la valeur booléenne est vraie.
Équivalent fonctionnel de l'instruction While.

number

Quand un nombre est déclaré, AL créer un objet. Cet objet à des propriétés et des méthodes. Voir [l'objet Type](#) pour les méthodes et les propriétés communes aux types.

Attention, les parenthèses doivent être employés avec ce type parce que le point est réservé aux décimales. ex : (7).isNumber();

```
number.isInteger ()
```

[insert into ide](#)

Vérifie si le type de la variable est de type NUMBER et entière.

```
number.isReal ()
```

[insert into ide](#)

Vérifie si le type de la variable est de type NUMBER et réel.

```
number.isNan ()
```

[insert into ide](#)

Vérifie si la valeur n'est pas un nombre, càd si le résultat de l'opération est invalide.

```
number.isInfinite ()
```

[insert into ide](#)

Vérifie si la valeur est infinie.

method

`number.toInteger ()` [insert into ide](#)

Retourne la valeur entière du nombre.

`number.minus ()` [insert into ide](#)

Retourne la valeur inverse du nombre, équivalent de -n.

`number.increment ([n])` [insert into ide](#)

Retourne le nombre incrémenté, Équivalent de $\text{number} + n$. Si n n'est pas paramétré, il est égale à 1

`number.decrement ()` [insert into ide](#)

Retourne le nombre décrémenté, Équivalent de $\text{number} + n$. Si n n'est pas paramétré, il est égale à 1

`number.addition (n)` [insert into ide](#)

Equivaut à l'opération $\text{number} + n$.

`number.substract (n)` [insert into ide](#)

Equivaut à l'opération $\text{number} - n$.

`number.multiply (n)` [insert into ide](#)

Equivalut à l'opération $\text{number} * n$.

`number.divide (n)` [insert into ide](#)

Equivalut à l'opération number / n .

`number.modulo (n)` [insert into ide](#)

Equivalut à l'opération $\text{number} \% n$.

`number.greaterThan (n)` [insert into ide](#)

Equivalut à l'opération $\text{number} > n$.

`number.smallerThan (n)` [insert into ide](#)

Equivalut à l'opération $\text{number} < n$.

`number.greaterOrEquals (n)` [insert into ide](#)

Equivalut à l'opération $\text{number} \geq n$.

`number.smallerOrEquals (n)` [insert into ide](#)

Equivaut à l'opération `number <= n`.

`number.between (min, max)` [insert into ide](#)

Retourne vrai si `number` est entre `min` et `max`. C'est l'Équivalent de `number >= min && number <= max`.

`number.decodePoint ()` [insert into ide](#)

Renvoie le caractère correspondant au code codepoint.

`number.loopFor (function (index) {} [, init, [, increment]])` [insert into ide](#)

Equivaut à l'instruction impérative `for (set index = init; index < number; index += increment)`. Exécute la fonction depuis 0 (ou `init`, optionel) jusqu'au nombre en incrémentant de 1 (ou `increment`, optionel) chaque fois.

string

Quand une chaîne de caractère est utilisée, AL créer un objet `STRING`. Cet objet à des propriétés et des méthodes. Voir [l'objet Type](#) pour les méthodes et les propriétés communes aux types.

property

`string.isEmpty ()` [insert into ide](#)

Vérifie si la chaîne est vide ("").

`string.length ()` [insert into ide](#)

Retourne le nombre de caractères de la chaîne.

`string.getChar (index)` [insert into ide](#)

Retourne le caractère à la position index dans la chaîne.

method

`string.contains (subString)` [insert into ide](#)

Retourne vrai si la chaîne contient la sous-chaîne.

`string.concat (string)` [insert into ide](#)

Concatène la chaîne de caractères avec celle en paramètre.

`string.indexOf (subString, index)` [insert into ide](#)

Retourne la position de la sous-chaîne contenue dans la chaîne.
Index est optionnel et indique à quelle position la recherche est commencé.

```
string.count (subString) insert into ide
```

Retourne le nombre de sous chaîne trouvés dans la chaîne.

```
string.upper () insert into ide
```

Retourne l'équivalent en majuscule de la chaîne.

```
string.lower () insert into ide
```

Retourne l'équivalent en minuscule de la chaîne.

```
string.append (substring [, index]) insert into ide
```

Retourne le résultat de la concaténation entre la chaîne et la sous-chaîne.
Index est optionnel et spécifie la position où la sous-chaîne doit être insérée.
Équivalent de l'opérateur ".."

```
string.appendSep (substring , separator) insert into ide
```

Retourne le résultat de la concaténation entre la chaîne et la sous-chaîne précédée d'un séparateur si elle

n'était pas vide.

```
string.substring (begin [, end])
```

insert into ide

Renvoie la coupe de la chaîne en une sous-chaîne entre le début et la fin.

End, le paramètre de fin est optionnel. Il indique la fin de la sous-chaîne, s'il n'est pas indiqué, alors c'est la fin de la chaîne qui est prise.

```
string.substringOf (begin [, end])
```

insert into ide

Renvoie la coupe de la chaîne en une sous-chaîne entre les sous-chaîne de début et de fin.

End est optionnel.

```
string.replace (from, to)
```

insert into ide

Renvoie la chaîne dont la sous-chaîne from a été remplacé par la sous-chaîne to.

```
string.replaceAt (substring, index)
```

insert into ide

Renvoie la chaîne dont la sous-chaîne a été remplacé à la position index.

```
string.remove (index [, length])
```

insert into ide

Renvoie la chaîne dont le caractère à la position à été remplacé.

Length est optionnel, if assigné il indique le nombre de caractère à supprimer.

```
string.split (separator) insert into ide
```

Divise la chaîne de caractère en un tableau de chaînes selon le séparateur choisi.
Utiliser le séparateur "" pour convertir une chaîne en tableau de caractères.

```
string.splitAt (array) insert into ide
```

Divise la chaîne de caractère en un tableau de chaînes selon les indexes spécifiés dans le tableau (array) choisi.

```
string.trim () insert into ide
```

Renvoie la chaîne dont les espaces inutiles au début et à la fin de la chaîne ont été supprimés.

```
string.create (subString, count) insert into ide
```

Créer une chaîne en dupliquant une sous-chaîne de 0 à "count" fois.

```
string.encodePoint () insert into ide
```

Créer un tableau dont les valeurs sont les codespoints unicode de la chaîne.

functional method

```
string.each (function (char [, index]) {} [, step])
```

[insert into ide](#)

Itère sur tous les caractères de la chaîne.

La fonction exécutée doit définir un paramètre pour récupérer le caractère à chaque itération de la boucle.

Le second paramètre index est optionnel et fournit la position du caractère dans la chaîne. Step est optionnel, il représente le nombre de caractère à sauter avant la prochaine itération.

array

Quand un tableau est déclaré, AL crée un objet. Cet objet a des propriétés et des méthodes. Voir [l'objet Type](#) pour les méthodes et les propriétés communes aux types.

property

```
array.isEmpty ()
```

[insert into ide](#)

Vérifie si le tableau est vide ({}).

```
array.length ()
```

[insert into ide](#)

Retourne le nombre d'éléments du tableau.

```
array.getItem (identity)
```

[insert into ide](#)

Obtient l'élément du tableau à la position indiquée par identity.
Identity peut-être la position ou la clé de l'élément.

`array.setItem (identity, item)` [insert into ide](#)

Définit la valeur du tableau à la position indiquée par identity.
Identity peut-être la position ou la clé de l'élément.

`array.getFirst ()` [insert into ide](#)

Obtient le premier élément du tableau.

`array.getLast ()` [insert into ide](#)

Obtient le dernier élément du tableau.

prototype

`array.clone ()` [insert into ide](#)

Duplique le tableau vers un nouveau.
Chaque modification faite à un clone n'affecte pas l'objet original.

method

```
array.clear ()
```

[insert into ide](#)

Enlève tous les éléments du tableau.

```
array.contains (item)
```

[insert into ide](#)

Teste si l'élément est contenu par le tableau.

```
array.remove (identity)
```

[insert into ide](#)

Supprime un élément du tableau à la position de identity.
Identity peut-être la position ou la clé de l'élément.

```
array.pop ([identity])
```

[insert into ide](#)

Identity est un paramètre optionnel, il peut-être la position ou la clé de l'élément.
Si identity est absent, retourne le dernier élément du tableau et le supprime.
Si identity est spécifié, retourne l'élément à la position et le supprime du tableau.
Utiliser `array.add()` et `array.pop()` pour avoir un comportement de pile LiFo (Last in First out, ou en français DEPS Dernier entré Premier sortie). Utiliser `array.add()` et `array.pop(0)` pour avoir un comportement de file FiFo (First in First out, ou en français PEPS Premier entré Premier sortie)

```
array.indexOf (item)
```

[insert into ide](#)

Retourne la position de l'élément dans le tableau.

```
array.count (item)
```

[insert into ide](#)

Retourne le nombre d'élément d'une certaine valeur trouvée dans le tableau.

```
array.swap(identity1, identity2)
```

[insert into ide](#)

Intervertie les deux éléments aux positions identity1 et identity2.
Identity1 et 2 peuvent être les positions ou les clés des éléments.

```
array.decodePoint ()
```

[insert into ide](#)

Crée une chaîne de caractères à partir de sa représentation codepoint.

functional method

```
array.find (function (item [, index [, key]] [, pos]) {})
```

[insert into ide](#)

Trouve l'élément du tableau pour lequel la fonction retourne la valeur vraie.
La fonction exécutée doit définir un paramètre pour récupérer l'élément à chaque itération de la boucle.
Le second paramètre index est optionnel et fournit la position de l'élément dans le tableau.
Le troisième paramètre est également optionnel, il fournit la clé de l'élément.
Pos est un paramètre également optionnel de la méthode find. Elle indique à partir de quel indice la recherche doit commencer.

```
array.create (count, function ([index]) {})
```

[insert into ide](#)

Créer un tableau de n éléments. Chaque élément est calculé par la fonction passée en paramètre.

```
array.each (function (item [, index [, key]]) {})
```

[insert into ide](#)

Itère sur tous les éléments du tableau.

La fonction exécutée doit définir un paramètre pour récupérer l'élément à chaque itération de la boucle.

Le second paramètre index est optionnel et fournit la position de l'élément dans le tableau.

Le troisième paramètre est également optionnel, il fournit la clé de l'élément.

```
array.eachOnRow (row, function (item [, index [, key]]) {})
```

[insert into ide](#)

Dans un tableau à deux dimensions, itère sur tous les éléments de la ligne du tableau.

La fonction exécutée doit définir un paramètre pour récupérer l'élément à chaque itération de la boucle.

Le second paramètre index est optionnel et fournit la position de l'élément dans le tableau.

Le troisième paramètre est également optionnel, il fournit la clé de l'élément.

```
array.eachOnCol (col, function (item [, index [, key]]) {})
```

[insert into ide](#)

Dans un tableau à deux dimensions (une table), itère sur tous les éléments de la colonne du tableau.

La fonction exécutée doit définir un paramètre pour récupérer l'élément à chaque itération de la boucle.

Le second paramètre index est optionnel et fournit la position de l'élément dans le tableau.

Le troisième paramètre est également optionnel, il fournit la clé de l'élément.

```
array.forEach (function (item [, index [, key]]) {})
```

[insert into ide](#)

Dans un tableau à plusieurs dimensions (un arbre), itère sur tous les éléments du tableau, si l'élément est un tableau, il va itérer sur les éléments du tableau récursivement.

La fonction exécutée doit définir un paramètre pour récupérer l'élément à chaque itération de la boucle.

Le second paramètre index est optionnel et fournis la position de l'élément dans le tableau.

Le troisième paramètre est également optionnel, il fournis la clé de l'élément.

```
array.filter (function (item [, index [, key]]) {})
```

[insert into ide](#)

Filtre les éléments du tableau sur la valeur de retour d'une fonction.

La fonction doit retourner un booléen.

Si la valeur du booléen est vrai, l'élément est conservé, sinon il est supprimé du tableau.

La fonction exécutée doit définir un paramètre pour récupérer l'élément à chaque itération de la boucle.

Le second paramètre index est optionnel et fournis la position de l'élément dans le tableau.

Le troisième paramètre est également optionnel, il fournis la clé de l'élément.

```
array.sort ([function (item1, item2){}])
```

[insert into ide](#)

Renvoie un tableau trié sur les éléments du tableau.

La fonction est optionnelle.

```
array.min ([function (item1, item2){}])
```

[insert into ide](#)

Retourne la valeur minimale contenue dans le tableau.

La fonction est optionnelle. Voir [sort](#) pour plus de détails;

```
array.max ([function (item1, item2){}])
```

[insert into ide](#)

Retourne la valeur maximale contenue dans le tableau.
La fonction est optionnelle. Voir [sort](#) pour plus de détails;

```
array.join (function (item1, item2 [, index [, key]]){})
```

[insert into ide](#)

Joint tous les éléments du tableau ensemble.
La fonction est nécessaire pour savoir comment joindre séquentiellement les éléments.
C'est utile pour concaténer un tableau en un string ou pour ajouter tous les éléments d'un tableau.

```
array.merge (array, function (item1, item2, index [, key]]){})
```

[insert into ide](#)

Renvoie le résultat de la fusion de deux tableaux. Chaque élément du premier tableau est fusionné avec l'élément correspondant du second tableau.
La fonction détermine le comportement de la fusion en retournant la valeur souhaitée pour chaque élément.
Note: si le second tableau est trop court, les éléments seront répétés depuis le début jusqu'à la complétion.

function

Quand une fonction est déclarée, AL crée un objet. Cet objet a des propriétés et des méthodes. Voir [l'objet Type](#) pour les méthodes et les propriétés communes aux types.

property

`function.parameterExists (name)` [insert into ide](#)

Vérifie que le paramètre existe dans la définition de la fonction.

`function.getParametersNames ()` [insert into ide](#)

Retourne un tableau contenant tous les noms des paramètres définis par la fonction.

`function.setParameter (name, value)` [insert into ide](#)

Assigne la valeur au paramètre de la fonction dont le nom correspond à name. Le paramètre est créé s'il n'existe pas. Utilisé pour assigner les paramètres avant l'appelle de la fonction (util lors du passage d'une fonction à une autre sans copie des paramètres).

`function.setParameters ({ values })` [insert into ide](#)

Assigne les valeurs aux paramètres de la fonction. Attention, les valeurs des paramètres doivent être regroupés dans un tableau (entre {}). Utilisé pour assigner tous les paramètres avant l'appelle de la fonction (util lors du passage d'une fonction à une autre sans copie des paramètres).

method

`function.removeParameter (name)` [insert into ide](#)

Supprime dynamiquement un paramètre de la définition de la fonction.

functional method

`function.concat (function)` [insert into ide](#)

Retourne une fonction résultat de la concaténation des deux fonctions. Les paramètres sont fusionnés et les traitements s'exécutent en suivant. Equivaut à l'opérateur ->

`function.decorate (function)` [insert into ide](#)

Renvoie une nouvelle fonction résultat de la décoration de la fonction par une fonction décoratrice. L'exécution de la fonction décorée est encapsulé dans la fonction décoratrice. Nécessaire au paradigme Aspect.

object

Quand un objet est déclaré, AL créer un meta-objet. Cet objet à des propriétés et des méthodes. Voir [l'objet Type](#) pour les méthodes et les propriétés communes aux types.

prototype

`object.clone ()` [insert into ide](#)

Duplique l'objet vers un nouveau.
Chaque modification faite à un clone n'affecte pas l'objet original.

C'est l'équivalent de l'opérateur new.

property

`object.attributeExists (name)` [insert into ide](#)

Vérifie que l'attribut existe dans la définition de l'objet.

`object.getAttributesNames ()` [insert into ide](#)

Retourne un tableau contenant tous les noms des attributs définis par l'objet.

`object.isA ()` [insert into ide](#)

Vérifie que l'objet soit une instance ou un sous objet de celui en paramètre.

`object.setAttribute (name, value)` [insert into ide](#)

Assigne ou ajoute (si absent) dynamiquement un attribut à la définition de l'objet. L'attribut peut-être un attribut, une méthode ou un objet imbriqué.

method

`object.toString = function () {}` [insert into ide](#)

Remplace le text par défaut de l'objet.

```
object.merge (object) insert into ide
```

Renvoie un objet résultat de la fusion de deux objets. Les définitions sont fusionnées ensemble par addition des attributs et des méthodes.

```
object.removeAttribute (name) insert into ide
```

Supprime dynamiquement un attribut de la définition de l'objet. L'attribut peut-être un attribut, une méthode ou un objet imbriqué.

API root

```
print (text) insert into ide
```

Ecrit le texte.

al

property

```
al.allObjects insert into ide
```

Retourne tous les objets et les méthodes qui peuvent être utilisés dans l'API Algoid.

`al.allLocalObjects`

insert into ide

Retourne tous les objets et les méthodes qui peuvent être utilisés dans la portée courante.

`al.clock`

insert into ide

Retourne le temps en seconde depuis lequel Algoid a été lancé.

al.order

method

`al.order.ascending (item1, item2)`

insert into ide

Trie l'ordre du tableau de façon ascendante.

`al.order.descending (item1, item2)`

insert into ide

Trie l'ordre du tableau de façon ascendante.

`al.order.random (item1, item2)`

insert into ide

Inverse l'ordre du tableau.

```
al.order.reverse (item1, item2)
```

[insert into ide](#)

Inverse l'ordre du tableau.

al.combine

```
al.combine.sum (item1, item2)
```

[insert into ide](#)

Combine les éléments d'un tableau ensemble par sommage.

```
al.combine.product (item1, item2)
```

[insert into ide](#)

Combine les éléments d'un tableau ensemble par produit.

```
al.combine.concat (item1, item2, index, key, separator)
```

[insert into ide](#)

Combine les éléments d'un tableau ensemble par concaténation.

al.types

property

```
al.types.VOID
```

[insert into ide](#)

Le type AL VOID. Représente une valeur nulle.

`al.types.BOOL`

[insert into ide](#)

Le type AL BOOL. Représente une valeur booléenne (true, false).

`al.types.NUMBER`

[insert into ide](#)

Le type AL NUMBER. Représente une valeur numérique (1, 2, 3.5, 7).

`al.types.STRING`

[insert into ide](#)

Le type AL STRING. Représente une chaîne de caractère ("Hi, I am algoid !").

`al.types.ARRAY`

[insert into ide](#)

Le type AL ARRAY. Représente un tableau de données ({true, 5, "Hi"}).

`al.types.FUNCTION`

[insert into ide](#)

Le type AL FUNCTION. Représente une fonction.

`al.types.OBJECT`

[insert into ide](#)

Le type AL OBJECT. Représente un objet.

util

method

util.eval (code) [insert into ide](#)

Execute le code en paramètre dans la portée courante.

util.wait (milli) [insert into ide](#)

Attendre le temps indiqué en milli-secondes

util.pulse (**function** ([index]) {}, milli [, count[, after]]) [insert into ide](#)

Execute la fonction chaque fois que le temps (milli-secondes) est écoulé.

La fonction appelé reçoit un paramètre optionel appelé index qui spécifie le numéro d'appel.

Lorsque la fonction renvoie une valeur, cela termine l'evenement.

Le paramère optionel count spécifie combien de fois pulse fonctionne avant d'être arrêté. Appel la fonction after un fois pulse finis.

util.notice (**function**, milli) [insert into ide](#)

Execute la fonction après un temps donné.

```
util.pullOn (array, milli) insert into ide
```

PullOn accepte un tableau de fonction qu'il execute sequentiellement avec un delai entre chaque appel.

```
util.clearTasks (array, milli) insert into ide
```

Supprime toutes les tâches et les evenements à venir, mais pas la tâche courante.

```
util.log (msg) insert into ide
```

Ecrit le message dans le journal.

```
util.warn (msg) insert into ide
```

Ecrit le warning dans le journal (orange).

```
util.err (msg) insert into ide
```

Ecrit l'erreur dans le journal (rouge).

constant

`math.E`

insert into ide

La constante mathématique e.

`math.PI`

insert into ide

La constante mathématique PI.

method

`math.abs (number)`

insert into ide

Retourne la valeur absolue (positive) du nombre.

`math.acos (factor)`

insert into ide

Calcule l'arc cosinus d'un angle en degré.

`math.aim (number, number)`

insert into ide

Calcule l'angle pour suivre une cible depuis les coordonnées (0, 0) du plan.

```
math.asin (factor)
```

[insert into ide](#)

Calcule l'arc sinun d'un angle en degré.

```
math.atan (factor)
```

[insert into ide](#)

Calcule l'arc tangente d'un angle en degré.

```
math.ceil (number)
```

[insert into ide](#)

Renvoie la plus petite valeur entière directement supérieur au nombre : $10.9 = 11$

```
math.cos (angle)
```

[insert into ide](#)

Calcule le cosinus d'un angle en degré.

```
math.db1 (number)
```

[insert into ide](#)

Calcule le nombre à la puissance 2.

```
math.diagonal (number, number)
```

[insert into ide](#)

Calcule la longueur de la diagonale (hypothénuse) d'un rectangle.

```
math.exp (number)
```

[insert into ide](#)

Calcule la valeur exponentielle du nombre.

```
math.floor (number)
```

[insert into ide](#)

Renvoie la partie entière du nombre : 10.9 = 10

```
math.log (number[, base])
```

[insert into ide](#)

Renvoie le logarithme naturel du nombre. Si la base est assignée, renvoie le logarithme du nombre de la base donnée.

```
math.max (number, number)
```

[insert into ide](#)

Retourne le plus grand des deux nombres.

```
math.min (number, number)
```

[insert into ide](#)

Retourne le plus petit des deux nombres.

```
math.pow (number, power)
```

[insert into ide](#)

Calcule le nombre à la puissance n.

```
math.random (factor)
```

[insert into ide](#)

Génère un nombre aléatoire entre 0 et factor à l'exception de celui-ci.

```
math.round (number)
```

[insert into ide](#)

Arrondi le nombre réel en un nombre naturel : $10.9 = 11$

```
math.sin (angle)
```

[insert into ide](#)

Calcule le sinun d'un angle en degré.

```
math.sqrt (number)
```

[insert into ide](#)

Calcule la racine carrée du nombre.

```
math.tan (angle)
```

[insert into ide](#)

Calcule la tangente d'un angle en degré.

```
ui.message (msg)
```

[insert into ide](#)

Affiche une fenêtre popup de message dans l'application.

```
ui.fullScreen ()
```

[insert into ide](#)

Montre la fenêtre d'exécution en mode plein écran.

```
ui.halfScreen ()
```

[insert into ide](#)

Montre la fenêtre d'exécution en mode demi écran.

```
ui.miniScreen ()
```

[insert into ide](#)

Cache la fenêtre d'exécution et montre le code source.

```
ui.showText ()
```

[insert into ide](#)

Montre la fenêtre d'invite en mode demi écran.

```
ui.showAlgo ()
```

[insert into ide](#)

Fait défiler les fenêtres jusqu'à celle d'algo.

```
ui.showScope ()
```

[insert into ide](#)

Fait défiler les fenêtres jusqu'à celle du debugger.

```
ui.showLog ()
```

[insert into ide](#)

Fait défiler les fenêtres jusqu'à celle du journal.

```
ui.clearLog ()
```

[insert into ide](#)

Vide le journal.

```
ui.showMenu ()
```

[insert into ide](#)

(Seulement sur android !) Montre le menu glissant à la droite de la fenêtre algo.

```
ui.hideMenu ()
```

[insert into ide](#)

(Seulement sur android !) Cache le menu glissant à la droite de la fenêtre algo.

text

```
text.clear ()
```

[insert into ide](#)

Vide la fenêtre d'invite.

```
text.output (msg) insert into ide
```

Ecrit une nouvelle ligne dans la fenêtre d'invite.

```
text.inputText (msg) insert into ide
```

Ecrit une nouvelle ligne de saisie utilisateur (texte) dans la fenêtre d'invite.
Retourne une chaîne de caractère.

```
text.inputNumber (msg) insert into ide
```

Ecrit une nouvelle ligne de saisie utilisateur (nombre) dans la fenêtre d'invite.
Retourne un nombre.

algo.color

constant

```
algo.color TRANSP insert into ide
```

La constante de couleur transparente (-1). Voir la [table des couleurs](#).


```
algo.color.BLACK
```

[insert into ide](#)

La constante de couleur noire (0). Voir la [table des couleurs](#).

```
algo.color.DARK_BLUE
```

[insert into ide](#)

La constante de couleur bleu foncé (1). Voir la [table des couleurs](#).

```
algo.color.DARK_GREEN
```

[insert into ide](#)

La constante de couleur vert foncé (2). Voir la [table des couleurs](#).

```
algo.color.DARK_CYAN
```

[insert into ide](#)

La constante de couleur cyan foncé (3). Voir la [table des couleurs](#).

```
algo.color.DARK_RED
```

[insert into ide](#)

La constante de couleur rouge foncé (4). Voir la [table des couleurs](#).

```
algo.color.DARK_MAGENTA
```

[insert into ide](#)

La constante de couleur violet foncé (5). Voir la [table des couleurs](#).

```
algo.color.BROWN
```

[insert into ide](#)

La constante de couleur marron (6). Voir la [table des couleurs](#).

```
algo.color.GRAY
```

[insert into ide](#)

La constante de couleur gris (7). Voir la [table des couleurs](#).

```
algo.color.DARK_GRAY
```

[insert into ide](#)

La constante de couleur gris foncé (8). Voir la [table des couleurs](#).

```
algo.color.BLUE
```

[insert into ide](#)

La constante de couleur bleu (9). Voir la [table des couleurs](#).

```
algo.color.GREEN
```

[insert into ide](#)

La constante de couleur vert (10). Voir la [table des couleurs](#).

```
algo.color.CYAN
```

[insert into ide](#)

La constante de couleur cyan (11). Voir la [table des couleurs](#).

`algo.color.RED`

[insert into ide](#)

La constante de couleur rouge (12). Voir la [table des couleurs](#).

`algo.color.MAGENTA`

[insert into ide](#)

La constante de couleur violet (13). Voir la [table des couleurs](#).

`algo.color.YELLOW`

[insert into ide](#)

La constante de couleur jaune (14). Voir la [table des couleurs](#).

`algo.color.WHITE`

[insert into ide](#)

La constante de couleur blanc (15). Voir la [table des couleurs](#).

algo

property

`algo.setColor (color)`

[insert into ide](#)

Détermine la couleur des futures formes d'Algo.

couleur	valeur	couleur	valeur
transp.	-1		
noir	0	gris sombre	8
bleue sombre	1	bleue	9
vert sombre	2	vert	10
cyan sombre	3	cyan	11
rouge sombre	4	rouge	12
violet sombre	5	violet	13
marron	6	jaune	14
gris clair	7	blanc	15

`algo.setBgColor (color)`

insert into ide

Détermine la couleur de fond d'Algo.

`algo.setAlpha (color)`

insert into ide

Détermine la transparence de la couleur des futures formes. La valeur doit être entre 0 et 1.

`algo.setFontSize (size)` [insert into ide](#)

Détermine la taille du futures text d'Algo.

`algo.setStroke (size)` [insert into ide](#)

Détermine l'épaisseur du trait des futures formes et chemins d'Algo.

`algo.setStack ()` [insert into ide](#)

Détermine les nombre d'éléments simultanément visible dans Algo.

`algo.getX ()` [insert into ide](#)

Obtient la coordonné X d'Algo.

`algo.getY ()` [insert into ide](#)

Obtient la coordonné Y d'Algo.

`algo.getAngle ()` [insert into ide](#)

Obtient l'angle d'Algo.

```
algo.getTop ()
```

[insert into ide](#)

Obtient la coordonné du haut de la fenêtre Algo (inférieur à 0).

```
algo.getBottom ()
```

[insert into ide](#)

Obtient la coordonné du bas de la fenêtre Algo (supérieur à 0).

```
algo.getLeft ()
```

[insert into ide](#)

Obtient la coordonné de la gauche de la fenêtre Algo (inférieur à 0).

```
algo.getRight ()
```

[insert into ide](#)

Obtient la coordonné de la droite de la fenêtre Algo (supérieur à 0).

```
algo.getWidth ()
```

[insert into ide](#)

Obtient la largeur de la fenêtre Algo.

```
algo.getHeight ()
```

[insert into ide](#)

Obtient la hauteur de la fenêtre Algo.

method

`algo.show ()` [insert into ide](#)

Montre la tortue.

`algo.hide ()` [insert into ide](#)

Cache la tortue.

`algo.clear ()` [insert into ide](#)

Vide Algo de tous ses éléments visibles.

`algo.autoClear ()` [insert into ide](#)

Vide Algo de tous ses éléments visibles sans effet de scintillement. Util pour la programmation d'animation et de jeux vidéos. A placer au début de la boucle de rendu.

`algo.removeFirst ()` [insert into ide](#)

Supprimer le premier élément dessiné dans Algo.

```
algo.removeLast ()
```

[insert into ide](#)

Supprimer le dernier élément dessiné dans Algo.

```
algo.goTo (x, y)
```

[insert into ide](#)

Déplace la tortue aux coordonnées absolues x, y sur la grille.

```
algo.lineTo (x, y)
```

[insert into ide](#)

Dessine une ligne depuis la position actuelle vers les coordonnées x, y.

```
algo.go (len)
```

[insert into ide](#)

Fait avancer la tortue en dessinant une ligne.

```
algo.jump (len)
```

[insert into ide](#)

Fait avancer la tortue sans dessiner de ligne.

```
algo.turnRight (angle)
```

[insert into ide](#)

Tourne la tortue à droite.

```
algo.turnLeft (angle)
```

insert into ide

Tourne la tortue à gauche.

```
algo.rotateTo (angle)
```

insert into ide

Tourne la tortue vers un angle absolu.

```
algo.circle (diameter)
```

insert into ide

Dessine un cercle à la position courante.

```
algo.disc (diameter)
```

insert into ide

Dessine un cercle plein (un disque) à la position courante.

```
algo.square (size)
```

insert into ide

Dessine un carré à la position courante.

```
algo.box (size)
```

insert into ide

Dessine un carré plein (une boîte) à la position courante.

```
algo.rect (width, height) insert into ide
```

Dessine un rectangle à la position courante.

```
algo.plane (width, height) insert into ide
```

Dessine un rectangle plein (un plan) à la position courante.

```
algo.oval (width, height) insert into ide
```

Dessine un ovale à la position courante.

```
algo.platter (width, height) insert into ide
```

Dessine un ovale plein à la position courante.

```
algo.path (array[, closed = true]) insert into ide
```

Dessine un chemin fermé. Le paramètre optionel closed définit si la forme est fermé, sa valeur par défaut est vrai. Attention, nécessite un tableau de paires x, y en entrée. Le premier point est toujours la position courante.

```
algo.poly (array[, closed = true])
```

[insert into ide](#)

Dessine un chemin fermé plein (un polygone). Le paramètre optionel closed définit si la forme est fermée, sa valeur par défaut est vrai. Attention, nécessite un tableau de paires x, y en entrée. Le premier point est toujours la position courante.

```
algo.curve (array[, closed = true])
```

[insert into ide](#)

Dessine un chemin courbe. Le paramètre optionel closed définit si la forme est fermée, sa valeur par défaut est vrai. Attention, nécessite un tableau de paires x, y en entrée. Le premier point est toujours la position courante.

```
algo.curvedPoly (array[, closed = true])
```

[insert into ide](#)

Dessine un chemin plein courbé (un polygone). Le paramètre optionel closed définit si la forme est fermée, sa valeur par défaut est vrai. Attention, nécessite un tableau de paires x, y en entrée. Le premier point est toujours la position courante.

```
algo.text (text)
```

[insert into ide](#)

Dessine du texte à la position courante.

event

```
algo.onClick (function (x, y))
```

[insert into ide](#)

(Seulement sur desktop !) Exécute la fonction définit lorsque l'utilisateur clique avec la souris sur algo. Lorsque la fonction renvoie une valeur, cela termine l'événement.

```
algo.onRelease (function (x, y)) insert into ide
```

(Seulement sur desktop !) Exécute la fonction définit lorsque l'utilisateur a fini de cliqué Algo avec la souris. Lorsque la fonction renvoie une valeur, cela termine l'événement.

```
algo.onMove (function (x, y)) insert into ide
```

(Seulement sur desktop !) Exécute la fonction définit lorsque l'utilisateur glisse la souris sur algo. Lorsque la fonction renvoie une valeur, cela termine l'événement.

```
algo.onDrag (function (x, y)) insert into ide
```

(Seulement sur desktop !) Exécute la fonction définit lorsque l'utilisateur clique et glisse la souris sur algo. Lorsque la fonction renvoie une valeur, cela termine l'événement.

```
algo.onWheel (function (rotation)) insert into ide
```

(Seulement sur desktop !) Exécute la fonction définit lorsque l'utilisateur utilise la roue de la souris sur algo. Retourne 1 pour bas et -1 pour haut. Lorsque la fonction renvoie une valeur, cela termine l'événement.

```
algo.onKey (function (key))
```

[insert into ide](#)

(Seulement sur desktop !) Exécute la fonction définit lorsque l'utilisateur tape une touche du clavier. Lorsque la fonction renvoie une valeur, cela termine l'evenement.

```
algo.onTap (function (x, y))
```

[insert into ide](#)

Exécute la fonction définit lorsque l'utilisateur tape avec son doigt sur algo. Lorsque la fonction renvoie une valeur, cela termine l'evenement.

```
algo.onTouch (function (x, y))
```

[insert into ide](#)

Exécute la fonction définit lorsque l'utilisateur glisse son doigt sur algo. Lorsque la fonction renvoie une valeur, cela termine l'evenement.

```
algo.onUp (function (x, y))
```

[insert into ide](#)

Exécute la fonction définit lorsque l'utilisateur à finit de taper Algo avec son doigt. Lorsque la fonction renvoie une valeur, cela termine l'evenement.

```
algo.onGravity (function (x, y, z))
```

[insert into ide](#)

(Seulement sur android !) Exécute la fonction définit lorsque l'utilisateur tourne l'appareil. x, y et z sont les axes 3d sur lesquels s'exerce la force de gravité en m/s. Lorsque la fonction renvoie une valeur, cela termine l'evenement.

```
algo.onAcceleration (function (x, y, z))
```

[insert into ide](#)

(Seulement sur android !) Exécute la fonction définit lorsque l'utilisateur déplace l'appareil. x, y et z sont les axes 3d sur lesquels s'exerce la force d'accélération en m/s.
Lorsque la fonction renvoie une valeur, cela termine l'événement.

```
algo.onGyroscope (function (x, y, z))
```

[insert into ide](#)

(Seulement sur android !) Exécute la fonction définit lorsque l'utilisateur tourne l'appareil. x, y et z sont les axes 3d sur lesquels s'exerce la force de rotation en degré/s.
Lorsque la fonction renvoie une valeur, cela termine l'événement.

```
algo.onOrientation (function (z, x, y))
```

[insert into ide](#)

(Seulement sur android !) Exécute la fonction définit lorsque l'utilisateur tourne l'appareil par rapport au champ magnétique de la terre. z, x et y sont l'Azimut, le tangage et le roulis et leur unité est le en degré.
Lorsque la fonction renvoie une valeur, cela termine l'événement.

```
algo.onProximity (function (dist))
```

[insert into ide](#)

(Seulement sur android !) Exécute la fonction définit lorsque l'utilisateur se rapproche ou s'éloigne de l'appareil. Distance est en cm.
Lorsque la fonction renvoie une valeur, cela termine l'événement.

`algo.onTemperature (function (t))` [insert into ide](#)

(Seulement sur android !) Exécute la fonction définit lorsque la température ambiante est modifiée. t est en °C.

Lorsque la fonction renvoie une valeur, cela termine l'événement.

`algo.onLight (function (l))` [insert into ide](#)

(Seulement sur android !) Exécute la fonction définit lorsque la lumière ambiante est modifiée. l est en lx.
Lorsque la fonction renvoie une valeur, cela termine l'événement.

algo.stamp

Un tampon est un objet graphique bitmap qui peut-être dessiné par duplication (tamponné) dans algo.

property

`algo.stamp.id` [insert into ide](#)

L'identifiant interne du tampon dans algo.

`algo.stamp.width` [insert into ide](#)

La largeur du tampon.

`algo.stamp.height`

[insert into ide](#)

La hauteur du tampon.

prototype

`algo.stamp.clone (colors, size)`

[insert into ide](#)

Crée un nouveau tampon avec les informations bitmap et la taille.

method

`algo.stamp.delete ()`

[insert into ide](#)

Supprime le tampon dans algo

`algo.stamp.draw ()`

[insert into ide](#)

Duplique et dessine le tampon dans algo.

Forme de Backus-Naur d'Algoid Language

// expressions et précédence des opérateurs

assign ::= var **ident** ("=" | "+=" | "-=" | "*=" | "/=" | "%=" | "..=" | "->=") **expr** [";"]

expr ::= **concat** | **assign**


```

concat ::= or [ ( "." | "->" ) or ]
or ::= and [ "|" and ]
and ::= compar [ "&&" compar ]
compar ::= oper [ "==" | "!=" | "<" | "<=" | ">" | ">=" oper ]
oper ::= mult [ "+" | "-" mult ]
mult ::= unary [ "*" | "/" | "%" unary ]
unary ::= ( ["-" | "!" | new ] ident ) | ( ident [ "++" | "--" ] )

```

// identifiants

```

ident ::= call [ "." call ]
call ::= index "(" [ expr [ { "," expr } ] ] ")"
index ::= value [ { "[" expr "]" } ]

```

// valeurs

```

boolean ::= true | false
number ::= nan | infinity | ("0"-"9") [ { ("0"-"9") } ]
string ::= "" [ { .. caractère ASCII .. } ] ""
value ::= "(" expr ")" | object | lambda | function | array | if | this | supers | ident | nil | boolean | number | string

```

// structures

```

assign ::= set ident "=" expr [ ";" ]
lambda ::= [ "(" [ ident ] [ { "," ident } ] ")" ] "{" body "}"
function ::= function "(" [ ident ] [ { "," ident } ] ")" "{" body "}"
return ::= return expr [ ";" ]
array ::= array<1*> [ "(" "]" ] "{" [ item [ { "," item } ] ] "}"
item ::= [ expr ":" ] expr
object ::= object "(" [ ident [ "," ident ] ] ")" "{" [ { assign } ] "}"

```

// impératifs

```

if ::= if "(" expr ")" body [ { elseif "(" expr ")" body } ] [ else body ]
loop ::= loop "(" [ expr ] ")" body
for ::= for "(" [ assign ] ";" [ expr ] ";" [ expr ] ")" body
while ::= while "(" expr ")" body
until ::= do body until "(" expr ")"
break ::= break [ ";" ]

```

// autres

```

comment ::= ( "/" .. caractère ASCII .. fin de ligne .. ) | ( "/" .. caractère ASCII .. "*" )

```

body ::= instruction | "{" [{ instruction }] "}"

instruction ::= comment | return | loop | for | while | until | break | expr [";"] | ";" | body