□□

MicomScript_Windows 版 リファレンスマニュアル 1.0.0alpha

□□

目次

１．使い方

1.1 Visual Studio のダウンロード

https://visualstudio.microsoft.com/ja/downloads/#

・上の URL から、Visual Studio 2017 をダウンロードします。

1.2 実行

・Visual Studio のコマンドプロンプト上の src 内でコンパイル(cc と入力することでも可)、続けて「mscript」を入力することで実行されます。

・Visual Studio がダウンロードされていない場合、bin フォルダより、MicomScript.exe を実行します。

1.3 アンインストール

インストール先のフォルダ内のファイルをすべて削除するだけです。

２．注意事項

●記号 "＄" は、入力促進記号を表します。

●データ型について
入力値は、"int 型"、"float 型"、"string 型"、"list 型"、"bool 型"に分けられます。
異なる型の演算時には、演算子の左側の型に統一されて演算結果が出力されます。

●文字列の長さ
入力できる一行の長さは、半角 255 文字です。
256 文字以上入力すると不具合を起こします。

●変数について
変数宣言は、var の後ろに宣言したい変数名を記述します。
変数に何も代入されていない場合、変数の値は 0(integer 型)として演算されます。
　C言語と異なり、変数の型は実行時に動的に決まるため、変数宣言時に型名を決める必要はありません。
変数名は半角のみ使用可能です。
1 文字目には、a～z、A～Z、_を使用することができます。
2 文字目には、a～z、A～Z、_に加えて 0～9 の数字を使用できます。
変数名はアルファベットの大文字と小文字を区別して扱われます。
予約語、定数名を変数名に使用することは出来ません。
　(宣言はできますが予約語や定数が優先的に扱われるため、定数と同じ名前の変数は扱うことができません。)

●ソースコードの長さについて
作成できるプログラムの行数に上限はありませんが、メモリ容量が足りなくなった場合、正常に動作しなくなります。

３．演算子

数値…integer 型、float 型のことを指します。
数字…0～9 までの値を含む文字列のことを指します。
　返される値が、演算子の左側と同じデータ型になるように、演算される値は一部変換され、演算処理がされます。
　list 型の値の要素を指定するとき、その要素と同じデータ型の規則で演算されるものとします。
　list 型の値が演算子の左側であるとき、list 型の値は、その要素それぞれが演算の対象になります。
その際、演算の規則は要素と同じデータ型の値と、同じ規則で演算されます。

3.1 加算 （＋）

　数値が演算子の左側で、文字列(string 型)が演算子の右側の場合、文字列の先頭が数字ならその数字が数値として扱われ、それ以外の場合は 0(integer 型)として演算されます。
　文字列が演算子の左側のとき、演算子の右側のデータ型が integer 型,float 型の場合、その数値が数字として扱われ、list 型の場合"List"(string 型)として文字列結合されます。

```
$ 1 + 2.1 + "string"
3                    → 1 + 2 + 0

$ 1.5 + 1 + "string"
2.5                  → 1.5 + 1 + 0

$ "string" + 1 + 2.1
string12.1           → "string" + "1" + "2.1"

$ "Micom" + "Script"
MicomScript

$ 1.1 + "1.1apple"
2.2                  → 1.1+1.1

$ "apple" + [1,2,3]
"appleList"          → ”apple” + “List”
```

3.2 減算 （ - ）

integer 型または float 型の値が演算子の左側であるとき、
文字列(string 型)が演算子の右側の場合、文字列の先頭が数字であればその数字の部分のみ数値として扱われ、それ以外の場合は 0(integer 型)として演算されます。
string 型の値が演算子の左側であるときの演算はできません。

```
$ 2 - 1.1
1                    → 2 – 1

$ 1.5 - 1
0.5                  → 1.5 – 1.0

$ 2.3 - "0.3"
2                    → 2 .3– 0.3

$ 12 - "11apple"
1                    → 12 – 11

$ 12 - "apple"
12                   → 12 – 0
```

3.3 乗算 （ * ）

string 型が演算子の左側の場合、右側のデータ型は integer 型に統一され、文字列の重複回数になります。

演算子の右側のデータ型が string 型の場合、文字列(string 型)の先頭が数字であればその数字の部分のみが数値として扱われます。

"12aaaaaaa"のようなときは 12(integer 型)となり、文字のみもしくは文字列の先頭が文字のときはすべて 1(integer 型)となります。

$ 1 * 1.5
1　　→ 1 * 1

$ 1.5 * 1
1.5　　→ 1.5 * 1.0

$ 1.1 * "2.0"
2.2　　→ 1.1 * 2.0

$ 1 * "string"
0　　→ 1 * 0

$ "string" * 2.3
stringstring　　→ “string” * 2

$ "hello" * "world"
hello　　→ “hello” * 1

3.4 除算 （/）

割り切れない演算の場合は、小数第 7 位を四捨五入して返します。
　integer 型または float 型の値が演算子の左側であるとき、右側にある string 型の値は、文字列の先頭が数字の場合、その数字の部分のみが数値として扱われ、それ以外の string 型の値が含まれるとき、エラーとなります。
string 型の値が先頭であるときの演算はできません。

$ 4.4 / 2
2.2　　→ 4.4 / 2.0

$ 2.0 / 3
0.666667　　→ 2.0 / 3.0

$ 2.2 / "2.2"
1　　→ 2.2 / 2.2

3.5 剰余（％）

" a ％ b "において、a を b で割ったときの余りを求める演算子です。
　integer 型または float 型の値が演算子の左側であるとき、右側にある string 型の値は、文字列(string 型)の先頭に数字がある場合、その数字の部分のみが数値として扱われ、それ以外の string 型の値が含まれる場合は、エラーとなります。
string 型の値が先頭であるときの演算はできません。
list 型の値が演算子の左側であるとき、演算はできません。
list 型どうしの剰余演算はできません。

```
$ 6 % 2.5
0                    → 6 % 2

$ 6.0 % 2.5
1                    → 6.0 % 2.0

$ "18 % "8"
2                    → 18 % 8

$ 18.0 % "8.5"
1                    → 18.0 % 8.5
```

3.6 べき乗 （**）

" a ** b "において、a の b 乗の演算結果が返されます。
　integer 型または float 型の値が演算子の左側にあるとき、
演算子の右側にある string 型の値は、文字列の先頭に数字がある場合、その数字のみが数値として扱われます。
　文字のみもしくは文字列の先頭が文字の場合は、その値は数値の 0 に変換され、演算されます。
string 型の値が先頭であるときの演算はできません。

$ 2 ** 2.5
4 → 2^2

$ 2.0 ** 2.5
5.65685 → $2.0^{2.5}$

$ 2 ** "2.0"
4 → 2^2

3.7 代入演算子（ = , += , -= , *= , /= , %= , **= ）

代入演算子は、演算と代入を簡単に表記するための演算子です。

a += b ←→ a = a + b
a -= b ←→ a = a - b
a *= b ←→ a = a * b
a /= b ←→ a = a / b
a %= b ←→ a = a % b
a **= b ←→ a = a ** b
a &= b ←→ a = a & b
a |= b ←→ a = a | b
a ^= b ←→ a = a ^ b
a ~= b ←→ a = a ~ b
a <<= b ←→ a = a << b
a >>= b ←→ a = a >> b

3.8 インクリメント/デクリメント（ ++ , -- ）

++、--を使う場合、記述する位置に注意しましょう。
" ++a "のように変数の前に記述した場合、a の値をインクリメントしてから値を返します。
逆に" a++ "のように変数の後ろに記述した場合、値を返したあとにインクリメントを行います。
string 型のインクリメント、デクリメントはできません。

```
$ var a = 1
println(a++)
1
$ println(a)
2
$ println(a++)
2
$ println(a)
3
```

```
$ var a = 1
$ println(++a)
2
$ println(a)
2
$ println(++a)
3
$ println(a)
3
```

3.9 ビット演算子(& , | , ^ , ~ , >> , <<)

整数をビット単位で演算します。
数字以外の文字を含む string 型の値は 0 として扱われます。

3.9.1 AND (&)

両方のビットが 1 の場合は 1、それ以外の場合は 0 を返す演算子。

```
$ 0b0110 & 0b0101
4

$ 0b1110 & 0b0111
6
```

3.9.2 OR (|)

両方のビットが 0 の場合 0、それ以外の場合は 1 を返す演算子。

```
$ 0b0110 | 0b0101
7

$ 0b1110 | 0b0111
15
```

3.9.3 XOR (^)

両方のビットが異なる値の場合 1 、それ以外の場合は 0 を返す演算子。

```
$ 0b0110 ^ 0b0101
3

$ 0b1110 ^ 0b0111
9
```

3.9.4 NOT (~)

値を反転させる演算子。

```
$ ~1
-2

$ ~2
-3
```

3.9.5 右シフト演算子(>>)

整数値を指定したビット分だけ右へずらす演算子。

```
$ 8 >> 1
4

$ 8 >> 2
2
```

3.9.6 左シフト演算子(<<)

整数値を指定したビット分だけ左へずらす演算子。

```
$ 2 << 1
4

$ 2<< 2
8
```

3.10 論理演算（ &&, ||, ! ）

0 以外の値として含まれるのは、int 型、float 型、string 型、list 型 の値。

3.10.1 論理積（ && ）

論理積は両辺ともに 1(true)の場合 true を返し、それ以外は false を返す。

```
$ 1 && 1
true

$ 1 && 0
false

$ 0 && 0
false
```

3.10.2 論理和（ || ）

両辺ともに 0(false)の場合、false を返し、それ以外は true を返します。

```
$ 1 || 1
true

$ 1 || 0
true

$ 0 || 0
false
```

3.10.3 否定（!）

true ならば false に、false ならば true を返します。
$!1
false

$!0
true

3.11 比較演算子（ == , != , < , <= , > , >=）

文字列を比較するときは、比較する値の左側から一つずつ比較し、結果を返します。
結果を真偽値(true または false)で返します。
文字列比較のときの大小関係は、
1 < 2 < ... < 8 < 9 < a < b < ... < y < z
のようになります。
string 型の値が数字のみの場合、その値は数値として扱われます。

3.11.1 等価演算子(==)

左辺と右辺の値が等しい場合、true、それ以外は false を返します。

$ 1 == 1
true

$ 1 == 0
false

$ "string" == "string"
true
↑string 型も数値と同様に両辺の値が一致したとき、true を返す。

3.11.2 不等価演算子(!=)

左辺と右辺の値が等しくない場合、true、それ以外は false を返します。

$ 1 != 1
false

$ 1 != 0
true

3.11.3 小なり(<)

左辺が右辺未満の場合、true、それ以外は false を返します。

$ 0 < 1
true

$ 1 < 0
false

3.11.4 小なりイコール(<=)

右辺が左辺以上の場合、true、それ以外は false を返します。

$ 0 <= 1
true

$ 1 <= 0
false

3.11.4 大なり(>)

右辺が左辺未満の場合、true、それ以外は false を返します。

```
$ 1 > 0
true

$ 0 > 1
false
```

3.11.5 大なりイコール(>=)

左辺が右辺以上の場合、true、それ以外は false を返します。

```
0 >= 1
false

1 >= 0
true
```

3.12 演算子の優先順位

※番号が小さいほど、優先順位が高いです。

優先順位	演算子
1	( ) [ ] . " ++ --
2	+ - ~ !
3	**
4	* / %
5	+ -
6	<< >>
7	< <= > >=
8	== !=
9	&
10	^
11	|
12	&&
13	||
14	= += -= *= /= %= &= **= |= ^= ~= <<= >>=

４．制御文

{ }内で実行する処理が一つである場合は、{ }を付けずに、()の後ろに処理を記述してもよいです。

4.1 if(引数){}

条件分岐を実行する関数です。
引数の条件式が真であるとき、直後の{ }内の処理を実行します。
その他の条件分岐を記述したい場合は else if, else を用いることも可能です。
他の条件分岐を指定したい場合、直後に else if(引数)の引数に条件式、直後の{ }内に処理を記述します。

```
$ var i = 3$
$ if(i < 3){
                println("i < 3")
  }else if(i == 3){
                println("i == 3")
  }else{
                println("i > 3")
  }
i == 3
```

4.2 for(){}

引数 2 に入る条件式が真である間、{ }内の処理を繰り返します。

for(引数 1;引数 2;引数 3){
①.引数 1 は、初めに一度だけ実行される初期値です。
②.引数 2 の条件式が真ならば、{ }内の処理を実行する。条件式が偽ならば、for の処理を終了します。
③.{ }内の処理を実行後、引数 3 の処理を一度実行し、②に返ります。

引数 1 では var 変数名 ... と記述可能です。
引数内で宣言した変数は以降も残ります。

```
$ var i
$ var sum = 0
$ for(i = 1;i <= 10;i++){
                        sum += i
  }
$ println(sum)
55
```

4.3 while(){}

引数の条件式が真である間、{ }内の処理を繰り返します。

```
$ var i = 1
$ var sum = 1
$ while(i <= 10){
                        sum += i
                        i++
  }
$ println(sum)
55
```

4.4 do{ } while()

引数の条件式が真である間、{ }内の処理を繰り返します。
while 文とは異なり、{ }内の処理がされた後、条件式の判定があるため、
一度は必ず{ }内の処理が行われます。

```
$ var i = 1
$ do {
                    println(i)
                    i++
  } while (i <= 3)
1
2
3
```

4.5 repeat(){ }

引数 1 が引数 2 の値になるまでの間、{ }内の処理を繰り返します。

repeat(引数 1,引数 2,引数 3){
引数 1 : 初期値。repeat()が実行されるとき最初に一度だけ実行されます。
引数 2 : 終了値。
引数 3 : 増分指定。指定がない場合は増分は 1 となります。

repeat()は for()とは違い、一回の判定で繰り返しができます。

```
$ var i
$ repeat(i = 1, 5){
                    println(i)
  }
1
2
3
4
5
```

4.6 times(){}

引数に指定した回数、{ }内の処理を実行します。

```
$ times(5){
                println("hello")
  }
hello
hello
hello
hello
hello
```

4.7 foreach(){}

foreach(引数 in 配列名){
上のように引数を記述します。配列の要素数分繰り返し、一回繰り返しが行われるたびに配列に含まれる要素の値が変数に代入されます。

```
$ var a
$ var list = [1,2,3,4]
$ foreach(a in list){
                println(a)
  }
1
2
3
4
```

５． list 操作関数

5.1 配列名.append()

引数に指定した値を配列の末尾に追加します。

```
$ var a = [1,2,3]
$ a.append(5)
$ a
[ 1, 2, 3, 5 ]
$ a.append(7)
5
$ a
[ 1, 2, 3, 5, 7 ]
```

5.2 配列名.insert()

配列名.insert(引数１,引数 2)
上のように引数を記述し、引数 1 で指定したインデックスに、引数 2 に指定した要素を挿入します。
このとき、指定したインデックスより後ろの要素は、新しい要素が挿入される前に一つずつ後ろに移動します。
引数１には、存在する配列の要素と、その要素の両端より一つ隣のインデックスまで insert() に指定できます。

```
$ a = [0,1,2,4]
[ 0, 1, 2, 4 ]
$ a.insert(3,8)
5
$ a
[ 0, 1, 2, 8, 4 ]
$ a.insert(5,9)
6
$ a
[ 0, 1, 2, 3, 4, 9 ]
```

5.3 配列名.clear()

配列名の要素を全て空にします。

```
$ a = [1,2,3]
[ 1, 2, 3 ]
$ a
[ 1, 2, 3 ]
$ a.clear()
0
$ a
[  ]
```

5.4 配列名.remove()

引数に指定した要素と同じ要素を持つ配列の要素を一つ削除します。
指定した要素を二つ持っている場合、先頭に近い方の要素が削除されます。
配列にない値を引数に指定するとエラーになります。

```
$ a = [1,2,3,4]
[ 1, 2, 3, 4 ]
$ a.remove(3)
3
$ a
[ 1, 2, 4 ]
```

5.5 配列名.len()

配列の要素数を返します。

```
$ var list = [ 2, 4, 6 ]
$ list.len()
3

$ list= []
[    ]
$ list.len()
0
```

６．文字列操作関数

6.1 変数名.append()

引数に指定した文字列を、指定した変数に代入されている文字列の後ろに追加します。
文字列のみが代入されている変数のみ指定することができます。
引数に指定する文字列は" " で囲む必要があります。

```
$ var a = "apple"
$ a.append(",orange")
12
$ a
"apple,orange"
```

6.2 変数名.substr()

変数名.substr(引数 1,引数 2)
上のように引数を記述し、指定した変数に代入されている文字列の先頭を文字を 0 として、"引数 1"番目の文字から、"引数 2"文字分だけの文字を返します。
指定した変数の値は変わりません。
文字列のみが代入されている変数のみ指定することができます。

```
$ var a = "apple"
$ a.substr(1,3)
"ppl"
```

6.3 変数名.trim()

指定した変数に代入されている文字列の空白部分を消去した文字列を返します。
文字列と空白のみが代入されている変数のみ指定することができます。

```
$ var a = “ apple     “
$ a.trim()
"apple"
```

6.4 変数名.len()

指定した変数に含まれる文字列の文字数を返します。
文字列のみが代入されている変数のみ指定することができます。

```
$ var a = "apple"
$ a.len()
5
```

７．数学関数

数学関数の引数には、数値(integer 型、float 型)を記述するものとします。
数字のみの string 型の値は、数値として扱われます。
()に何も入力しなかった場合、デフォルト引数は nan もしくは最後にその関数で最後に入力した引数の値が入ります。

7.1 三角関数 sin(),cos(),tan()

引数に指定した値の正弦(サイン)、余弦(コサイン)、正接(タンジェント)を返す関数です。
list 型の値を引数として指定した場合、配列内の要素を合計した値の正弦、余弦、正接を返します。
引数は弧度法[rad]で入力します。
変数 pi = 3.14159 とします。
配列内の値も、float 型、str 型、list 型の評価方法と同様に扱われます。

```
$ sin(30.0 * pi / 180)
0.5

$ cos(60.0 * pi / 180)
0.5

$ tan(45.0 * pi /180)
1
```

7.2 逆三角関数 asin(),acos(),atan(),atan2()

引数の数値のアークサイン、アークコサイン、アークタンジェントを返す関数です。
list 型の値を引数として指定した場合、配列内の要素を合計した値のアークサイン、アークコサイン、アークタンジェントを返します。
返される値の単位は弧度法[rad]です
配列内の値も、float 型、str 型、list 型の評価方法と同様に扱われます。

```
$ asin(1) * 180 / 3.14
90.0456

$ acos(1) *180/3.14
0

$ atan(1) *180/3.14
45.0228
```

7.3 双曲線関数 sinh(),cosh(),tanh()

引数の数値のハイパボリックサイン、ハイパボリックコサイン、ハイパボリックタンジェントを返す関数です。
list 型の値を引数として指定した場合、配列内の要素を合計した値のハイパボリックサイン、ハイパボリックコサイン、ハイパボリックタンジェントを返します。
配列内の値も、float 型、str 型、list 型の評価方法と同様に扱われます。

```
$ sinh(1)
1.1752

$ cosh(1)
1.54308

$ tanh(1)
0.761594
```

7.4 切り捨て floor()

引数の数値の切り捨てを行う関数です。
str 型は 0 として扱います。

$ floor(13.9)
13

$ floor(13.4)
13

7.5 切り上げ ceil()

引数の数値の切り上げを行う関数です。

$ ceil (14.2)
15

$ ceil(14.6)
15

7.6 四捨五入 round()

引数の数値の四捨五入を行う関数です。

$ round(4.6)
5

$ round(3.4)
3

7.7 べき乗 pow()

pow(引数 1,引数 2)
上のように引数を記述し、引数 1 の数値を底、引数 2 の数値を指数とした時のべき乗を返す関数です。

```
$ pow(2,3)
9

$ pow(2,-3)
9
```

7.8 指数対数 log()

指定した値の e(ネイピア数)を底とする対数を返す関数です。

```
$ log(2)
0.693147

$ log(1)
0

$ log(E)
1
```

7.9 常用対数 log10()

底が 10 の対数(常用対数)を返す関数です。。

```
$ log10(10)
1

$ log10(1)
0

$ log10(1000)
3
```

7.10 累乗 exp()

e を底とする引数の数値の累乗を返す関数です。

```
$ exp(1)
2.71828
```

7.11 平方根 sqrt()

指定した値の平方根を返す関数です。

```
$ sqrt(4)
2

$ sqrt(2)
1.41421
```

7.12 立方根 cbrt()

指定した値の立方根を返す関数です。

```
$ cbrt(8)
2

$ cbrt(64)
4
```

7.13 平方和 hypot()

hypot(引数 1,引数 2)
上のように引数を記述し、引数 1、引数 2 の数値の平方和の平方根の値を返す関数です。

```
$ hypot(3,4)
5

$ hypot(1,1)
1.41421

$ hypot(1.41421 , 1.41421)
1.99999
```

7.14 乱数 random()

無作為に選んだ数値を出力する関数です。
0 より大きく、1 未満の間の小数第 6 位までの値が出力されます。

```
$ random()
0.531663

$ random()
0.571184

$ random()
0.601764
```

7.15 乱数の種 srand()

引数に値を指定することで乱数の種を変更する関数です。

８．時間関数

8.1 time()

UNIX 時間からの経過時間を返す関数です。
単位は秒です。
UNIX 時間は 1970-01-01 00:00:00(UTC)を指します。

8.2 millis()

プログラムの実行を開始した時から現在までの時間をミリ秒単位で返す関数です。

8.3 micros()

プログラムの実行を開始した時から現在までの時間をマイクロ秒単位で返す関数です。

8.4 delay()

プログラムを指定した時間だけ止める関数です。
単位はミリ秒(ms)です。

９．その他のライブラリ関数

数値…integer 型、float 型のことを指します。
数字…0～9 までの値を含む文字列のことを指します。

9.1 abs()

●引数の値を integer 型の絶対値に変換して返す関数です。

```
$ abs(-2)
 2
```

◆引数が float 型の場合、小数点以下は切り捨て、integer 型の絶対値を返します。

```
$ abs(2.1)
 2
```

◆引数が string 型の場合、integer 型の 0 を返します。
また、string 型の値の先頭が数字の場合、文字の直前までが数値として読み取られます。

```
$ abs(“tanoue Kanata!”)
 0
```

◆引数が list 型の場合、配列内の合計値を integer 型の絶対値に変換して返します。
（配列内の string 型も integer 型の 0 として処理されます。）
配列内の値も、float 型、str 型、list 型の評価方法と同様に扱われます。

```
$ var a = [1, 2, 3,  ”hello”]
$ abs(a)
 6
```

◆配列内に配列があった場合も同様に、配列内の合計を integer 型の絶対値として処理します。

```
$ var e = [ 1, 2, 3, [1, 2, 3] ]
$ abs(e)
 12
```

9.2 bool()

●引数に指定した値の真理値を返す関数です。

```
$ bool(3)
 true

$ bool(0)
 false

$ bool(3 < 4)
 true

$ bool(3 > 4)
 false
```

9.3 fabs()

●引数の値を float 型の絶対値に変換して返す関数です。

```
$ fabs(-2.2)
 2.2
```

◆引数が string 型の場合、float 型の 0 を返します。

```
$ fabs(“hello”)
 0
```

◆引数が list 型の場合、配列内の合計値を float 型の絶対値に変換して返します。
（配列内の string 型も float 型の 0 として処理されます。）

```
$ var a = [1, 2, 3,  ”hello”]
$ abs(a)
 6
```

◆配列内に配列があった場合も同様に、配列内の合計を float 型の絶対値として処理します。

```
$ var e = [ 1, -2.5, 3.5, [1, 2, 3] ]
$ abs(e)
8
```

9.4 float()

●引数の型を float 型に変換して返す関数です。

```
$ var a = 2;
$ type(a)
“integer”

$ a = float(a)
2
$ type(a)
“float”
```

◆引数が string 型の場合、float 型の 0 を返します。

```
$ var b =  “test”
$ type(b)
“string”

$ b = float(b)
0
$ type(b)
“float”
```

◆引数が list 型の場合、配列内の合計値を float 型に変換して返します。
（配列内の string 型も float 型の 0 として処理されます。）

```
$ var c = [1, 2, “namako”, 3.4]
$ type(c)
“list”

$c = float(c)
6.4
$ type(c)
“float”
```

9.5 input()

●ユーザーに値を入力させる関数です。

```
$ input()
Hello
“Hello”
```

◆変数に input()関数を代入する形で、入力値を変数に代入できます。

```
$ var a
$ a = input()
6
“6”

$ println(a)
6
```

◆入力値の型はすべて string 型になります。

```
$ type(a)
“string”
```

9.6 int()

●引数の型を integer 型に変換して返す関数です。

```
$ var a = 2.2;
$ type(a)
“float”

$ a = int(a)
2
$ type(a)
“integer”
```

◆引数が string 型の場合、integer 型の 0 を返します。

```
$ var b = “test”
$ type(b)
“string”

$ b = int(b)
0
$ type(b)
“integer”
```

◆引数が list 型の場合、配列内の合計値を integer 型に変換して返します。
（配列内の string 型も integer 型の 0 として処理されます。）

```
$ var c = [1, 2, “namako”, 3.4]
$ type(c)
“list”

$c = int(c)
6.4
$ type(c)
“integer”
```

9.7 max()

●引数の中で最大の要素を返す関数です。

```
$ max(1,2,3,4,5)
5
```

◆string 型の要素の先頭が数字である場合、先頭の数字がその要素の値になります。
（数字が大文字の場合は数値として扱われません）

```
$ max(1,2,3,4,”5abc2”)
5
```

◆string 型の要素の先頭が数字ではない場合、その要素は 0 として扱われます。

```
$ max(1,2,3,4,”a5bc2”)
4
```

◆引数が list 型の場合、list の中の要素が比較対象になります。
引数、list 型の要素内の string 型の値は指定することができません。

```
$ var a = [1,2,3,4,1.1]
$ max(a)
4
```

9.8 min()

●引数の中で最小の要素を返す関数です。

```
$ min(1,2,3,4,5)
1
```

◆string 型の要素の先頭が数字である場合、先頭の数字がその要素の値になります。
（数字が大文字の場合は数値として扱われません）

```
$ max(1,2,3,4,”5abc2”)
1
```

◆string 型の要素の先頭が数字ではない場合、その要素は 0 として扱われます。

```
$ max(1,2,3,4,”a5bc2”)
1
```

◆引数が list 型の場合、list の中の要素が比較対象になります。
引数、list 型の要素内の string 型の値は指定することができません。

```
$ var a = [1,2,3,4,1.1]
$ max(a)
1
```

9.9 print()

●引数の値を書き出す関数です。
実行後、末尾は改行されません。

```
$ print(1)
1$
```

◆文字列を出力する場合、”　”の内部に文字列を記述します。
，で区切ることで複数の値を指定できます。

```
$ print(“print”, 1)
print1$
```

◆引数が 2 進数, 16 進数の場合、10 進数として表示されます。

```
$ print(0b1100)
12$

$ print(0xc)
12$
```

9.10 printf()

●引数の”　”内の文字列を書き出す関数です。
実行後、末尾は改行されません。

```
$ var a = 4
$ printf(“Hello”)
Hello
```

◆”“内で、¥n を記述すると、記述した位置で改行が行われます。

```
$ var a = 4
$ printf(“Hello¥n”)
Hello
```

◆変数を書き出す場合、以下のようにコードを入力する。

```
$ var a = 4
$ printf(“a = %d¥n”, a)
a = 4
```

■変換指定子一覧

変換指定子	説明
%c	1 文字を出力する。
%s	文字列を出力する。
%d	整数を 10 進で出力する。
%u	符号なし整数を 10 進で出力する。
%o	整数を 8 進で出力する。
%x	整数を 16 進で出力する。
%f	実数を出力する。
%e	実数を指数表示で出力する。
%g	
%ld	倍精度整数を 10 進で出力する。
%lu	符号なし倍精度整数を 10 進で出力する。
%lo	倍精度整数を 8 進で出力する。
%lx	倍精度整数を 16 進で出力する。
%lf	倍精度実数を出力する。

9.11 println()

11.2 println()

●引数の値を書き出す関数です。
　実行後、末尾が改行されます。
( )内では改行されません。

```
$ println(1)
1
```

◆文字列を出力する場合、” ”の内部に文字列を記述します。
，で区切ることで複数の値を指定できます。

```
$ println(“print”, 1)
print1
```

◆引数が 2 進数, 16 進数の場合、10 進数として表示されます。

```
$ print(0b1100)
12

$ print(0xc)
12
```

9.12 string()

●引数の型を string 型に変換して返す関数です。

```
$ type(string(1.1))
“string”
```

◆引数が list 型の場合、string 型の文字列”List”が返されます。

```
$ var a = [1,2,3]
$ string(a)
“List”
```

9.13 sum()

●引数の合計値を返す関数です。

```
$ sum(1,2)
3

$ var list = [1,2,3]
$ sum(list)
6
```

◆string 型の要素は、0 として扱われます。
ただし、string 型の要素の先頭が数字である場合、先頭の数字がその要素の値になります。

```
$ sum(5,"world")
5

$ sum(5,"2world")
7
```

9.14 type()

●引数の型名を返す関数です。

```
$ var a = 1
$ type(a)
"integer"

$ var a = 1.1
$ type(a)
"float"

$ var a = "hello"
$ type(a)
"string"
```

１０．ユーザ定義関数

ユーザが作成する関数です。
ユーザ定義関数名も変数名と同様に半角のみ使用可能です。
全角文字は使用できません。
1 文字目には、a～z、A～Z、_を使用することができます。
2 文字目には、a～z、A～Z、_に加えて 0～9 の数字を使用できます。
アルファベットの大文字と小文字を区別しています。
予約語を使用することはできません。また、¥ などの特殊文字の使用もできません。

10.1 ユーザ定義関数作成

◆関数名，引数，内部処理を指定し、ユーザ定義関数を作成できます。

```
function 関数名(引数){
        内部処理
}
```

引数内で変数宣言することはできません。
引数の内側はユーザー定義関数外の領域となります。

◆return 変数名と記述することで、{}内で指定した変数を、関数呼び出し時に返すことができます。

```
$ function sigma(n){
    var i, sum = 0
        repeat(i = 1, n){
            sum += i
        }
        return sum
}
```

◆sigma 関数を作成し、呼び出すと、初期値 1 から n までの合計値が返されます。

```
$ sigma(10)
55

$ sigma(100)
5050
```

ユーザ定義関数内外の同名の変数は別の変数となります。
そのため、関数内で使用する変数は関数内で宣言する必要があります。

１１．コマンド

11.1 help

●コマンドの一覧を表示します。

$ help
[Command List]
vlist - print Variable list
clist - print Constants list
flist - Constants list
undef - undine Variable/Fanction
exit - finish execution

11.2 help コマンド名

●指定されたコマンドの説明が表示されます。

$ help vlist
定義した変数の変数名と値を表示します。

$ help clist
定数の名前とその値を表示します。

$ help flist
定義した関数の名前を表示します。

$ help undef
変数、または関数を削除します。
undef 変数名 で変数を削除し、undef 関数名() で関数を削除します。

$ help exit
作業を終了し、処理を終了します。

11.3 vlist

●変数の一覧を表示します。

変数に数字が入っていない場合、(null)と表示されます。

```
$var a
$var b = 5
$vlist
[Global Vars]
a = null
b = 5
```

11.4 undef 変数名

●指定した変数の変数定義を削除します。

変数の部分に all を記述することで、宣言されているすべての変数を削除します。

```
$ var a,b
$ undef a
$ vlist
[Global Vars]
b = (null)


$ var a=1, b=2, c=3
$ undef all
$ vlist
[Global Vars]
```

11.5 clist

●定数リストを表示します。

```
$ clist
[Constants]
Pi              : 3.14159
E               : 2.71828
true            : true
TRUE            : true
H               : true
HIGH            : true
High            : true
false           : false
FALSE           : false
L               : false
LOW             : false
Low             : false
```

11.6 flist

●定義されている関数をすべて表示します。

```
$ function f1(){}
$ function f2(){}
$ flist
[User Functions]
f1()
f2()
```

11.7 undef 関数名()

●定義されている関数を削除します。
同名の関数がある場合、古い関数を自動的に消去して置き換えます。
関数名の部分にall()と記述することで、定義されている全ての関数を削除できます。

```
$ function f1(){}
$ function f2(){}
undef f1()
$ flist
[User Functions]
f2()

$ function f1(){}
$ function f2(){}
$ undef all()
$ flist
[User Functions]
```

11.8 exit

プログラムを終了します。

１２．ファイル操作コマンド

12.1 files

●作成した全ファイルの一覧を表示します。

$ files
56 /s01
263 /Hanoi

12.2 files “検索ワード”

●検索ワードと同じ名前のファイル名がある場合そのファイル名を表示します。

12.3 save “/ファイル名”

●save コマンドを入力し、ファイルの保存を行います。;で終了します。

$ save “/hanoi”

12.4 load “/ファイル名”

●指定したファイルの実行を行います。

$ load “s01”
5050

12.5 remove “/ファイル名”

●指定したファイル名を削除します。

$ files
56 /s01
65 /s02
$ remove “/s02”
$ files
56 /s01

12.6 show “/ファイル名”

●指定したファイルのソースを表示します。

```
$ show “/s01”
function s(n){
   var i,sum = 0
   for(i = 1; i <= n; i++){
      sum += i
   }
   return sum
}
```

---

subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial
portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
IN THE SOFTWARE.

End license text.

川内職業能力開発短期大学校 MicomScript 開発チーム
岡元 和樹　浪江 光太郎　古田 聖弥　田中 利空　軸屋 樹　池ノ上 雄登　長元 海渡
相川 政和