□□

MicomScript_ESP32 Editon reference manual 1.0.0alpha

□□

Table of contents

1. How to use 5

２．Notes 9

３．Operator 10

3.1 addition (+) 10
3.2 subtraction (-) 11
3.3 multiplication (*) 11
3.4 division (/) 12
3.5 surplus (%) 13
3.6 exponentiation (**) 13
3.7 assignment operator （ = , += , -= , *= , /= , %= , **=) 14
3.8 increment/decrement （ ++ , --) 15
3.9 bitwise operator(& , | , ^ , ~ , >> , <<)エラー! ブックマークが定義されていません。
3.9.1 AND (&) 16
3.9.2 OR (|) 16
3.9.3 XOR (^) 16
3.9.4 NOT (~) 17
3.9.5 right shift operator(>>) 17
3.9.6 left shift operator(<<) 17
3.10 logical operations （ && , || , !) 18
3.10.1 logical AND (&&) 18
3.10.2 logical sum (||) 18
3.10.3 denial (!) 19
3.11 comparison operator （ == , != , < , <= , > , >=) 19
3.11.1 equality operator(==) 19
3.11.2 inequality operator(!=) 19

3.11.3 smaller (<) 20
3.11.4 less than equal (<=) 20
3.11.4 big (>) 21
3.11.5 greater than equal (>=) 21
3.12 operation priority 22

4． Control statement 23

4.1 if(argument){} 23
4.2 for(){} 24
4.3 while(){} 24
4.4 do{ } while() 25
4.5 repeat(){ } 25
4.6 times(){} 26
4.7 foreach(){} 26

5． List operation function 27

5.1 array name.append() 27
5.2 array name.insert() 27
5.3 array name.clear() 28
5.4 array name.remove() 28
5.5 array name.len() 29

6． String manipulation functions 30

6.1 array name.append() 30
6.2 array name.substr() 30
6.3 array name.trim() 31
6.4 variable name.len() 31

7． Math functions 32

7.1 trigonometric function　sin(),cos(),tan() 32
7.2 inverse trigonometric functions　asin(),acos(),atan(),atan2() 33
7.3 hyperbolic function　sinh(),cosh(),tanh() 33
7.4 truncation　floor() 34
7.5 round up　ceil() 34
7.6 rounding off　round() 34
7.7 exponentiation　pow() 35

7.8 exponential logarithm　log() 35
7.9 common logarithm　log10() 36
7.10 power　exp() 36
7.11 square root　sqrt() 36
7.12 cube root　cbrt() 37
7.13 sum of squares　hypot() 37
7.14 random number　random() 38
7.15 random number seeds　srand() 38

8．Time function 39

8.1 time() 39
8.2 millis() 39
8.3 micros() 39
8.4 delay() 39

9．Other library fuctions ..... エラー! ブックマークが定義されていません。

9.1 abs() ..... エラー! ブックマークが定義されていません。
9.2 bool() ..... エラー! ブックマークが定義されていません。
9.3 fabs() ..... エラー! ブックマークが定義されていません。
9.4 float() ..... エラー! ブックマークが定義されていません。
9.5 input() ..... エラー! ブックマークが定義されていません。
9.6 int() ..... エラー! ブックマークが定義されていません。
9.7 max() ..... エラー! ブックマークが定義されていません。
9.8 min() ..... エラー! ブックマークが定義されていません。
9.9 print() ..... エラー! ブックマークが定義されていません。
9.10 printf() ..... エラー! ブックマークが定義されていません。
9.11 println() ..... エラー! ブックマークが定義されていません。
9.12 string() ..... エラー! ブックマークが定義されていません。
9.13 sum() ..... エラー! ブックマークが定義されていません。
9.14 type() ..... エラー! ブックマークが定義されていません。

１０．User-defined function..... エラー! ブックマークが定義されていません。

10.1 Creating a user-defined function ..... エラー! ブックマークが定義されていません。

１１．Command..... エラー! ブックマークが定義されていません。

11.1 help.....エラー! ブックマークが定義されていません。
11.2 help　command name .....エラー! ブックマークが定義されていません。
11.3 vlist.....エラー! ブックマークが定義されていません。
11.4 undef　variable name .....エラー! ブックマークが定義されていません。
11.5 clist.....エラー! ブックマークが定義されていません。
11.6 flist .....エラー! ブックマークが定義されていません。
11.7 undef　function name .....エラー! ブックマークが定義されていません。
11.8 exit.....エラー! ブックマークが定義されていません。

1 2.　File operation commands....エラー! ブックマークが定義されていません。

12.1 files.....エラー! ブックマークが定義されていません。
12.2 files “search word”.....エラー! ブックマークが定義されていません。
12.3 save “/file name” .....エラー! ブックマークが定義されていません。
12.4 load “/file name” .....エラー! ブックマークが定義されていません。
12.5 remove “/file name” .....エラー! ブックマークが定義されていません。
12.6 show “/file name”.....エラー! ブックマークが定義されていません。

1． How to use

1.1 Install the driver

https://jp.silabs.com/developers/usb-to-uart-bridge-vcp-drivers?tab=downloads

・Download the corresponding OS driver for "CP210x_Universal_Windows_Driver" from the above URL.

・Unzip the downloaded ZIP file

From Device Manager、「Ports」 → 「Silicon Labs CP210x USB to UART Bridge(COMx)」 and select 「Update Driver」

From “Browse my computer” → “Search for drivers”, select “Extracted folder”.

Finally, reboot and you’re done.

1 .2 Installing Arduino IDE

https://www.arduino.cc/en/software

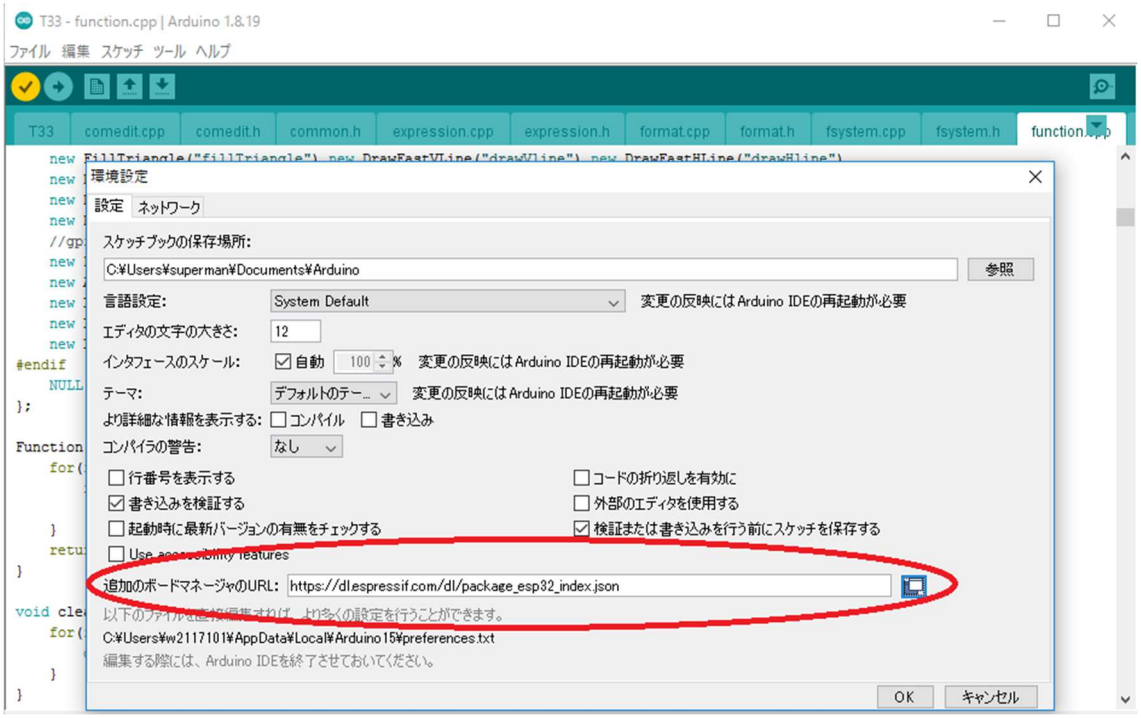
・Download the appropriate installer from the URL above. At this time, select "JUST DOWNLOAD", but if you use automatic translation, the text will become unreadable, so cancel automatic translation.

1.3 Arduino IDE settings

https://github.com/espressif/arduino-esp32

・From the URL above, open the "Arduino core for the ESP32" page on "GitHub".

・Open "README" → "Documentation" → "Installing (Windows, Linux and macOS)" and copy the URL under "Stable release link:".
From Arduino's "File", open "Basic Settings (Environment Settings)",
Click the button on the far right of "Additional board manager URLs",
Paste the URL you copied under "Enter additional URLs, one line at a time" and select OK.

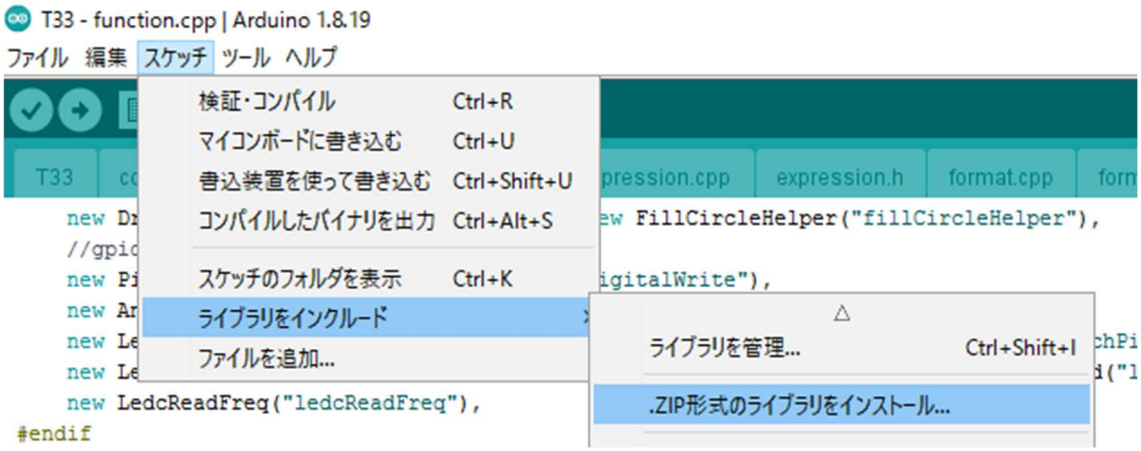


・Open “Tools” → “Board” → “Board Manager” from the menu bar,
Search for "ESP32" in the search field.
Install "ESP32 by Espressif System".

・From Arduino IDE's "File" → "Open", open the downloaded MicomScript.
Select the "src" file.

From "Tools", set as shown in the image. Select "Board", "Partition Scheme", and "Serial Port" according to the device.

・Select "Sketch" → "Include library" → "Install zipped library" in Arduino IDE, and select "TFT_eSPI-master.zip" from "lib" in MicomScript. (Do not unzip this file.)



1.4Write to the microcontroller board with Arduino IDE

By clicking on the ○ part of the image, it will be compiled and written to the microcontroller board.1.5 TeraTerm settings

・Select "Serial".

・Select "Settings" → "Serial Port" and set "Speed: 115200"

・From "Settings" → "Terminal", set "Receive: LF".

・Click Edit → Clear Screen to clear the screen.

1.6 Perform program work with TeraTerm

1.7 Uninstall

Just delete all files in the installation folder.

2. Notes

●The symbol " $ " represents a prompt symbol.

●About data types
Input values are divided into int, float, string, list, and bool types.
The operation result is output unified to the type on the left side of the operator.

●String length
The length of one line that can be entered is 255 half-width characters.
If you enter more than 256 characters, a problem will occur.

●About variables
To declare a variable, write the name of the variable you want to declare after var.
If nothing is assigned to a variable, the value of the variable is calculated as 0 (integer type).
Unlike the C language, the type of a variable is determined dynamically at runtime, so there is no need to decide the type name at the time of variable declaration.
Only half-width characters can be used for variable names.
The first character can be a-z, A-Z, _.
The second character can be numbers 0-9 in addition to a-z, A-Z, and _.
Variable names are treated as case sensitive.
Reserved words and constant names cannot be used as variable names.
(Although declarations can be made, reserved words and constants are treated preferentially, so variables with the same name as constants cannot be handled.)
●About source code length
There is no upper limit to the number of lines of a program that can be created, but if there is insufficient memory capacity, the program will not function properly.

3. Operator

Numerical value: Refers to integer type and float type.
Number: Refers to a string containing values from 0 to 9.

The value being operated on is partially converted and processed so that the returned value has the same data type as the left side of the operator.

When specifying an element of a list type value, it is assumed that the operation will be performed using the same data type rules as that element.

When a list type value is on the left side of an operator, each element of the list type value is subject to the operation. In this case, the calculation rules are calculated using the same data type value as the element and the same rules.

3.1 Addition (+)

If a number is on the left side of the operator and a string (string type) is on the right side of the operator, if the string starts with a number, that number is treated as a number, otherwise it is operated as 0 (integer type).

When a string is on the left side of an operator, if the data type on the right side of the operator is an integer type or float type, the number is treated as a number, and if it is a list type, it is string-combined as "List" (string type).

```
$ 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 ubtraction　(-)

When a value of type integer or float is on the left side of an operator,
When a string (string type) is on the right side of an operator, if the first number in the string is a number, only that number is treated as a number, otherwise it is calculated as 0 (integer type).
Operations cannot be performed when a string type value is on the left side of the operator.

```
$ 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 Multiplication　(*)

If the string type is on the left side of the operator, the data type on the right side is unified to integer type, which is the number of times the string is duplicated.

If the data type on the right side of the operator is string type, if the string (string type) starts with a number, only that number part will be treated as a number.

For example, "12aaaaaaa", it will be 12 (integer type), and if there are only characters or the first character in the string, it will be all 1 (integer type).

$ 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 Division (/)

If the calculation is not divisible, the number is rounded to the seventh decimal place and returned.
When an integer or float value is on the left side of an operator, a string value on the right side is
If the string starts with a number, only that number will be treated as a number, and if it contains other string type values, an error will occur.
Operations cannot be performed when the string type value is the first value.
$ 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 Surplus (%)

" a % b " is an operator that calculates the remainder when a is divided by b.

When an integer or float value is on the left side of an operator, a string value on the right side is If there is a number at the beginning of a string (string type), only that number will be treated as a number, and if it contains other string type values, an error will occur.

Operations cannot be performed when the string type value is the first value.
An operation cannot be performed when a list type value is on the left side of an operator.
Remainder operations between list types are not possible.

```
$ 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 Exponentiation　(**)

" a ** b " returns the result of a raised to the b power.

When a value of type integer or float is on the left side of an operator,
If the string value on the right side of the operator has a number at the beginning of the string, only that number is treated as a number.

If there are only characters or the first character in the string is a character, the value is converted to a numeric value of 0 and calculated.

Operations cannot be performed when the string type value is the first value.

$ 2 ** 2.5
4 → 2^2

$ 2.0 ** 2.5
5.65685 → $2.0^{2.5}$

$ 2 ** "2.0"
4 → 2^2

3.7 Assignment operator（ = , += , -= , *= , /= , %= , **= ）

Assignment operators are operators that simplify operations and assignments.

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 Increment/Decrement（ ++ , -- ）

When using ++ and --, be careful where you write them.
When written before a variable like " ++a ", the value of a is incremented and then the value is returned.
Conversely, if you write it after a variable like "a++", it will be incremented after returning the value.
Incrementing or decrementing string type is not possible.

```
$ 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 Bitwise operator(& , | , ^ , ~ , >> , <<)

Operates on integers bit by bit.
A string value containing non-numeric characters is treated as 0.

3.9.1 AND (&)

An operator that returns 1 if both bits are 1, and 0 otherwise.

```
$ 0b0110 & 0b0101
4

$ 0b1110 & 0b0111
6
```

3.9.2 OR (|)

Operator that returns 0 if both bits are 0, 1 otherwise.

```
$ 0b0110 | 0b0101
7

$ 0b1110 | 0b0111
15
```

3.9.3 XOR (^)

An operator that returns 1 if both bits have different values, and 0 otherwise.

```
$ 0b0110 ^ 0b0101
3

$ 0b1110 ^ 0b0111
9
```

3.9.4 NOT (~)

Operator that inverts the value.

```
$ ~1
-2

$ ~2
-3
```

3.9.5 Right shift operator (>>)

Operator that shifts an integer value to the right by a specified number of bits.

```
$ 8 >> 1
4

$ 8 >> 2
2
```

3.9.6 Left shift operator (<<)

Operator that shifts an integer value to the left by a specified number of bits.

```
$ 2 << 1
4

$ 2<< 2
8
```

3.10 Logical operations （ && , || , !)

Non-zero values include int, float, string, and list types.

3.10.1 Logical AND (&&)

Logical AND returns true if both sides are 1 (true), otherwise returns false.

$ 1 && 1
true

$ 1 && 0
false

$ 0 && 0
false

3.10.2 Logical Sum (||)

If both sides are 0 (false), returns false, otherwise returns true.

$ 1 || 1
true

$ 1 || 0
true

$ 0 || 0
false

3.10.3 Denial(!)

Returns false if true, true if false.

$!1
false

$!0
true

3.11 Comparison operator （ == , != , < , <= , > , >=)

When comparing strings, compare the values ??one by one starting from the left and return the results.
Returns the result as a boolean value (true or false).
The size relationship when comparing strings is
1 < 2 < ... < 8 < 9 < a < b < ... < y < z
It looks like.
If a string value contains only numbers, it is treated as a number.

3.11.1 Equality operator(==)

Returns true if the values ??on the left and right sides are equal; otherwise returns false.

$ 1 == 1
true

$ 1 == 0
false

$ "string" == "string"
true
↑ Similar to numbers, the string type also returns true when the values ??on both sides match.

3.11.2 Inequality operator(!=)

Returns true if the values on the left and right sides are not equal; otherwise returns false.

$ 1 != 1
false

$ 1 != 0
true

3.11.3 Smaller (<)

Returns true if the left side is less than the right side, otherwise returns false.

$ 0 < 1
true

$ 1 < 0
false

3.11.4 Less than equal (<=)

Returns true if the right side is greater than or equal to the left side, otherwise returns false.

$ 0 <= 1
true

$ 1 <= 0
false

3.11.4 Big (>)

Returns true if the right side is less than the left side, otherwise returns false.

$ 1 > 0
true

$ 0 > 1
false

3.11.5 Greater than equals (>=)

Returns true if the left side is greater than or equal to the right side, otherwise returns false.

0 >= 1
false

1 >= 0
true

3.12 Operator precedence

※The lower the number, the higher the priority.

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

4． Control statement

If there is only one process to be executed within { }, you can write the process after the () without adding { }.

4.1 if(argument){ }

A function that executes conditional branching.
When the conditional expression of the argument is true, the processing in the immediately following { } is executed.
If you want to write other conditional branches, you can also use else if, else.
If you want to specify another conditional branch, immediately write the conditional expression in the argument of else if (argument) and the process in { } immediately after.

```
$ 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(){}

Repeats the process inside { } while the conditional expression in argument 2 is true.

for(argument 1;argument 2;argument 3){
① .Argument 1 is an initial value that is executed only once at the beginning.
②. If the conditional expression of argument 2 is true, execute the process inside { }. If the conditional expression is false, the for processing ends.
③. After executing the process inside { }, execute the process for argument 3 once, and return to ②.

Argument 1 can be written as var variable name ...
Variables declared within the arguments remain.

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

4.3 while(){}

Repeats the process inside { } while the argument conditional expression is true.

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

4.4 do{ } while()

Repeats the process inside { } while the argument conditional expression is true.
Unlike the while statement, the conditional expression is evaluated after the processing inside { } is completed, so The processing inside { } is always performed once.

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

4.5 repeat(){ }

Repeat the process inside { } until argument 1 becomes the value of argument 2.

repeat(argument 1, argument 2, argument 3){
Argument 1: Initial value. It is executed only once, the first time repeat() is executed.
Argument 2: End value.
Argument 3: Increment specification. If not specified, the increment will be 1.

repeat() is different from for() in that it can be repeated with a single test.

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

4.6 times(){}

Executes the process inside { } the number of times specified in the argument.

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

4.7 foreach(){}

foreach(Argument in array name){
Write the arguments as above. Iterates for the number of elements in the array, and each time the element in the array is repeated, the value of the element in the array is assigned to the variable.

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

5． list operation function

5.1 array name.append()

Adds the value specified in the argument to the end of the array.

$ 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 array name.insert()

array name.insert(argument 1, argument 2)
Write the arguments as above, and insert the element specified in argument 2 at the index specified in argument 1.
In this case, elements after the specified index are moved backward one by one before the new element is inserted.
For argument 1, you can specify an existing array element and the next index from both ends of that element to insert().
$ 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 array name.clear()

Empty all elements of array name.

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

5.4 array name.remove()

Removes one element from the array that has the same element as the element specified in the argument.
If there are two specified elements, the element closest to the beginning will be deleted.
If you specify a value that is not in the array as an argument, an error will occur.

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

5.5 array name.len()

Returns the number of elements in the array.

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

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

6． string manipulation functions

6.1 Variable name.append()

Adds the string specified as an argument to the end of the string assigned to the specified variable.

Only variables to which only strings are assigned can be specified.

The string specified as an argument must be enclosed in " ".

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

6.2 Variable name.substr()

Variable name.substr(argument 1, argument 2)

Write the arguments as above, and return characters equal to "argument 2" from the "argument 1" character, starting with the character 0 at the beginning of the string assigned to the specified variable.

The value of the specified variable remains unchanged.

Only variables to which only strings are assigned can be specified.

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

6.3 Variable name.trim()

Returns the string assigned to the specified variable with any blank spaces removed.
Only variables to which only strings and spaces are assigned can be specified.

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

6.4 Variable name.len()

Returns the number of characters in a string contained in the specified variable.
Only variables to which only strings are assigned can be specified.

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

7． math functions

Numerical values (integer type, float type) shall be written in the arguments of mathematical functions.
A string value containing only numbers is treated as a number.
If nothing is entered in (), the default argument is nan or the value of the last argument entered in the function.

7.1 Trigonometric functions sin(), cos(), tan()

This function returns the sine, cosine, and tangent of the value specified as an argument.
If you specify a list type value as an argument, returns the sine, cosine, and tangent of the sum of the elements in the array.
Enter the argument in radial degree [rad].
Let the variable pi = 3.14159.
Values in arrays are also treated in the same way as float, str, and list types.

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

$ cos(60.0 * pi / 180)
0.5

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

7.2 inverse trigonometric functions asin(),acos(),atan(),atan2()

This function returns the arcsine, arccosine, and arctangent of the numerical value of the argument.
If a list type value is specified as an argument, returns the arcsine, arccosine, and arctangent of the sum of the elements in the array.
The units of the returned value are radial degrees [rad]
Values in arrays are also treated in the same way as float, str, and list types.

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

$ acos(1) *180/3.14
0

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

7.3 hyperbolic function sinh(),cosh(),tanh()

This function returns the hyperbolic sine, hyperbolic cosine, and hyperbolic tangent of the argument.
If a list type value is specified as an argument, the hyperbolic sine of the sum of the elements in the array,
Returns the hyperbolic cosine and hyperbolic tangent.
Values in arrays are also treated in the same way as float, str, and list types.

```
$ sinh(1)
1.1752

$ cosh(1)
1.54308

$ tanh(1)
0.761594
```

7.4 round down floor()

This function truncates the numerical value of the argument.
The str type is treated as 0.

$ floor(13.9)
13

$ floor(13.4)
13

7.5 round up ceil()

This function rounds up the numerical value of the argument.

$ ceil (14.2)
15

$ ceil(14.6)
15

7.6 Rounding off round()

This function rounds off the numerical value of the argument.

$ round(4.6)
5

$ round(3.4)
3

7.7 exponentiation pow()

pow(argument 1, argument 2)
This is a function that writes the arguments as above, and returns the power when the number of argument 1 is the exponent and the number of argument 2 is the base.

```
$ pow(2,3)
9

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

7.8 exponential logarithm log()

This is a function that returns the logarithm of the specified value to the base e (Napier's number).

```
$ log(2)
0.693147

$ log(1)
0

$ log(E)
1
```

7.9 common logarithm log10()

This is a function that returns the logarithm (common logarithm) with a base of 10.

```
$ log10(10)
1

$ log10(1)
0

$ log10(1000)
3
```

7.10 power　exp()

This is a function that returns the power of the number of the argument to the base e.

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

7.11 square root sqrt()

A function that returns the square root of the specified value.

```
$ sqrt(4)
2

$ sqrt(2)
1.41421
```

7.12 cube root cbrt()

This function returns the cube root of the specified value.

$ cbrt(8)
2

$ cbrt(64)
4

7.13 sum of squares hypot()

hypot(argument 1, argument 2)
This is a function that writes the arguments as above and returns the square root of the sum of squares of the numbers in argument 1 and argument 2.

$ hypot(3,4)
5

$ hypot(1,1)
1.41421

$ hypot(1.41421 , 1.41421)
1.99999

7.14 random number random()

This is a function that outputs a randomly selected number.
Values greater than 0 and less than 1 to the 6th decimal place are output.

$ random()
0.531663

$ random()
0.571184

$ random()
0.601764

7.15 random number seeds srand()

This is a function that changes the random number seed by specifying a value as an argument.

8． time function

8.1 time()

This function returns the number of seconds since startup.

8.2 millis()

This function returns the time in milliseconds since the start of program execution.

8.3 micros()

This function returns the time in microseconds since the start of program execution.

8.4 delay()

This function stops the program for a specified period of time.
The unit is milliseconds (ms).

9. Other library functions

9.1 print()

●This is a function that writes out the value of the argument.
After execution, there will be no line break at the end.

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

◆When outputting a string, write the string inside “ ”.
You can specify multiple values by separating them with ,.

```
$ print(“print”, 1)
print1$
```

◆If the argument is a binary or hexadecimal number, it will be displayed as a decimal number.

```
$ print(0b1100)
12$

$ print(0xc)
12$
```

9.2 println()

●This is a function that writes out the value of the argument.
After execution, a line break will be added at the end.
There is no line break within ( ).

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

◆When outputting a string, write the string inside “ ”.
You can specify multiple values by separating them with ,.

```
$ println(“print”, 1)
print1
$
```

◆If the argument is a binary or hexadecimal number, it will be displayed as a decimal number.

```
$ print(0b1100)
12
$

$ print(0xc)
12
$
```

9.3 printf()

●This is a function that writes out the string inside the argument “ ”.
After execution, there will be no line break at the end.

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

◆If you write ¥n within " ", a line break will be made at the written position.

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

◆To export variables, enter the code as shown below.

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

■Conversion specifier list

conversion specifier	explanation
%c	Outputs one character.
%s	Output a string.
%d	Outputs an integer in decimal.
%u	Outputs an unsigned integer in decimal.
%o	Outputs an integer in octal.
%x	Output an integer in hexadecimal.
%f	Outputs a real number.
%e	Output real numbers in exponential format.
%g	
%ld	Outputs a double-precision integer in decimal.
%lu	Outputs an unsigned double-precision integer in decimal format.
%lo	Outputs a double-precision integer in octal.
%lx	Outputs a double-precision integer in hexadecimal.
%lf	Outputs a double-precision real number.

9.4 max()

●A function that returns the largest element among its arguments.

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

◆If a string type element starts with a number, the first number becomes the value of that element.
(Numbers in uppercase are not treated as numbers)

```
$ max(1,2,3,4,"5abc2")
5
```

◆If an element of type string does not start with a number, it is treated as 0.

```
$ max(1,2,3,4,"a5bc2")
4
```

◆If the argument is of type list, the elements in list will be compared.

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

9.5 min()

●A function that returns the smallest element among its arguments.

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

◆If a string type element starts with a number, the first number becomes the value of that element.
(Uppercase numbers are not treated as numbers)

```
$ max(1,2,3,4,"5abc2")
1
```

◆If an element of type string does not start with a number, it is treated as 0.

```
$ max(1,2,3,4,"a5bc2")
1
```

◆If the argument is of type list, the elements in list will be compared.
Arguments and string type values within list type elements cannot be specified.

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

9.6 sum()

●A function that returns the total value of its arguments.

```
$ sum(1,2)
3

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

◆Elements of type string are treated as 0.

However, if the first number of a string type element is a number, the first number becomes the value of that element.

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

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

9.7 type()

●A function that returns the type name of the argument.

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

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

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

9.8 int()

●A function that converts the argument type to integer type and returns it.

```
$ var a = 2.2;
$ type(a)
“float”

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

◆If the argument is of type string, returns 0 of type integer.

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

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

◆If the argument is of list type, converts the total value in the array to integer type and returns it.

(The string type in the array is also treated as an integer type 0.)

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

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

9.9 float()

●This is a function that converts the argument type to float type and returns it.

```
$ var a = 2;
$ type(a)
“integer”

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

◆If the argument is of type string, returns 0 of type float.

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

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

◆If the argument is of type list, converts the total value in the array to float type and returns it.

(The string type in the array is also treated as a floating point type 0.)

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

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

9.10 string()

●A function that converts the argument type to string type and returns it.

```
$ type(string(1.1))
“string”
```

◆If the argument is of type list, the string "List" of type string is returned.

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

9.11 abs()

●A function that converts the argument value to an absolute value of integer type and returns it.

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

◆If the argument is of float type, decimal parts are rounded down and the absolute value of integer type is returned.

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

◆If the argument is of type string, returns 0 of type integer.
Also, if the string type value starts with a number, the part just before the character will be read as a number.

```
$ abs(“tanoue Kanata!”)
 0
```

◆If the argument is of list type, converts the total value in the array to an absolute value of integer type and returns it.
(The string type in the array is also treated as an integer type 0.)
Values in arrays are also treated in the same way as float, str, and list types.

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

◆Similarly, if there is an array within an array, the sum within the array is treated as the absolute value of the integer type.

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

9.12 fabs()

●A function that converts the argument value to an absolute value of float type and returns it.

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

◆If the argument is of type string, returns 0 of type float.

```
$ fabs(“hello”)
 0
```

◆If the argument is of type list, converts the total value in the array to the absolute value of float type and returns it.(The string type in the array is also treated as a float type 0.)

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

◆Similarly, if there is an array within an array, the sum in the array will be treated as the absolute value of the float type.

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

9.13 bool()

●A function that returns the truth value of the value specified as an argument.

```
$ bool(3)
 true

$ bool(0)
 false

$ bool(3 < 4)
 true

$ bool(3 > 4)
 false
```

9.14 input()

●A function that allows the user to enter a value.

```
$ input()
Hello
“Hello”
```

◆You can assign input values to variables by assigning the input() function to variables.

```
$ var a
$ a = input()
6
“6”

$ println(a)
6
```

◆All input values are of string type.

```
$ type(a)
“string”
```

10. user-defined function

●This is a function created by the user.。

As with variable names, only half-width characters can be used in user-defined function names.

Full-width characters are not allowed.

The first character is a~z, A~Z, _.

The second character can be any number from 0 to 9 in addition to a~z, A~Z, _.

Uppercase and lowercase letters of the alphabet are distinguished.

Reserved words are not allowed. Additionally, special characters such as /are not allowed.

10.1 Create user-defined function

◆You can create a user-defined function by specifying the function name, arguments, and internal processing.

```
function Function name (argument){
        Internal processing
}
```

Variables cannot be declared within arguments.

The area inside the argument is outside the user-defined function.

◆return By writing it as a variable name, the variable specified within {} can be returned when the function is called.

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

◆When you create and call the sigma function, the sum of initial values 1 to n will be returned.

```
$ sigma(10)
55

$ sigma(100)
5050
```

Variables with the same name inside and outside the user-defined function are different variables.

Therefore, variables used within a function must be declared within the function.

11. command

11.1 help

●Display a list of commands.

```
$help
[Command  List]
vlist                  - print Variable  list
clist                  - print Constants list
flist                  - print Function  list
undef                    - undefine Variable/Function
exit                    - finish execution
```

11.2 help command name

●Displays a description of the specified command.

$ help vlist
Displays the variable name and value of the defined variable.

$ help clist
Displays the constant name and its value.

$ help flist
Displays the name of the defined function.

$ help undef
Delete a variable or function.
undef variable name removes a variable, undef function name() removes a function.

$ help exit
Finish your work and end the process.

11.3 vlist

●Display a list of variables.

If the variable does not contain a number, (null) will be displayed.

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

11.4 undef Variable name

●Deletes the variable definition of the specified variable.

By writing all in the variable section, all declared variables will be deleted.

```
$ 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

●Display constant list.

```
$ 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
```

13.6 flist

●Show all defined functions.

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

11.7　undef 関数名()

●Delete a defined function.

If there is a function with the same name, it will automatically erase and replace the old function.

All defined functions can be deleted by writing all() in the function name part.

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

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

11.8　exit

●Exit the program.

11.9　reset

●Perform a reset.

11.10　clist_ESP

●Display constant list.

```
$ clist_ESP
[Constants_ESP]
OUTPUT          : 0x0000
INPUT           : 0x0001
INPUT_PULLUP : 0x0002
BLACK           : 0x0000
NAVY            : 0x000F
DARKGREEN       : 0x03E0
DARKCYAN        : 0x03EF
MAROON          : 0x7800
PURPLE          : 0x780F
OLIVE           : 0x7BE0
LIGHTGREY       : 0xD69A
DARKGREY        : 0x7BEF
BLUE            : 0x001F
GREEN           : 0x07E0
CYAN            : 0x07FF
RED             : 0xF800
MAGENTA         : 0xF81F
YELLOW          : 0xFFE0
WHITE           : 0xFFFF
ORANGE          : 0xFDA0
GREENYELLOW   : 0xB7E0
PINK            : 0xFE19
BROWN           : 0x9A60
GOLD            : 0xFEA0
SILVER          : 0xC618
SKYBLUE         : 0x867D
VIOLET          : 0x915C
LIGHTPINK       : 0xFC9F
```

12. File operation commands

12.1 files

●Display a list of all created files.
$ files
56 /s01
263 /Hanoi

12.2 files “Search word”

●If there is a file name with the same name as the search word, that file name will be displayed.

12.3 save "/file name"

●Save the file by entering the save command. ends with ;
$ save “/hanoi”

12.4 load "/file name"

●Executes the specified file.
$ load “s01”
5050

12.5 remove "/file name"

●Delete the specified file.
$ files
56 /s01
65 /s02
$ remove “/s02”
$ files
56 /s01

12.6 show "/file name"

●Displays the source program of the specified file.

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