□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

MicomScript_ESP32 版 リファレンスマニュアル 1.0.0beta

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

# 目次

# １．はじめに

令和 4 年度、当科にて「マイコンで動作するスクリプト言語 MicomScript」が開発され、2023 年 3 月に α 版が公開されました。

マイコンで動作するスクリプト言語としては、既に Python をマイコン向けに最適化した MicroPython が存在し、月間インターフェース等でも度々紹介されている点から、徐々に利用者が増えているものと思われます。

それに対し、当校で開発された MicomScript は、実用的なスクリプト言語に成長しつつあるものの、知名度が全くない状態であり、追加すべき仕様が多く残され、今なお発展途上にありました。

そこで私たちは、MicomScript に機能追加とバグ修正を行い、より有益な言語に進化させると共に普及活動を行い、１０年後にはある程度の知名度のあるプログラミング言語に育てることを目標とし開発に励みました。

本文書では、複雑な定義を読んで理解するよりも実際のプログラム例を見ることで理解を深められると考え、実際の使用例を多く示すことで実用的なリファレンスマニュアルを目指し作成しました。

MicomScript は C 言語、C++に寄せて作った言語なので比較的馴染みやすい言語です。

本文書をきっかけに少しでもプログラミングに興味を持つ方が増えれば幸いです。

# ２．諸注意事項

本文書記載内容はESP32-S3-DevKitC-1とM5Stamp Pico DIY Kitの両方のマイコンに対応させて実例と共に解説を進めています。
※M5Stamp Pico DIY Kitに関しては動作確認が全て行えておらず、外部接続がない状態であればESP32と同等な動作可能です。

## 2.1 MicomScriptの動作環境設定

ホームページの方にダウンロードページが用意されているのでそこからバイナリ版をダウンロードし、使用してください。

USBでマイコンを接続し、Tera Termでシリアルポートを選択した後、以下の設定を行ってから動作させてください。

■シリアルポート設定

通信速度：　921600
データ長：　8 bit
パリティ：　none
ストップビット：1bit
フロー制御：none
ローカルエコー：OFF
送信遅延：１ミリ秒／行

■端末設定

改行コード：　受信LF　送信LF

## 2.2 “$”記号について

記号 "＄" は、入力促進記号を表します。

## 2.3 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。

## 2.4 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。

## 2.5 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。

何も代入されていない変数が演算に使用される場合、変数の値は 0(integer)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。

変数名は半角のみが使用可能となります。

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います。

予約語、定数名を変数名に使用することはできません。

※宣言はできますが、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。

## 2.6 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（整数型）、float 型（浮動小数点型）のことを表します。
数字　…　0～9 までの値を含む文字列のことを表します。

返される値が先頭のデータと同じデータ型になるように、演算処理の段階で演算される値が一部変換されます。

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ます。

list 型の値が先頭であるとき、list 型の値は、その要素それぞれが演算の対象になります。

※演算の規則は、要素と同じデータ型の値が演算時に先頭にある場合と同じ規則で演算されます。

## 3.1 加算 （＋）

演算子の左側が数字で、演算子の右側が文字列(string 型)である場合、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。

演算子の左側が文字列(string 型)で、演算子の右側も文字列(string 型)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ます。

演算子の右側が list 型の場合”List”(string 型)が結合されます。

＜プログラム記述例＞

```
$ 3 + 2.1 + “string”
5

$ 1.5 + 2 + “string”
3.5

$ ”string” + 5 + 2.1
“string52.1”

$ “Micom” + “Script”
“MicomScript”

$ 1.1 + “1.1apple”
2.2

$ “apple” + [1,2,3]
“appleList”
```

```
//3 + 2 + 0


//1.5 + 2.0 + 0.0


  //”String” + “5” + “2.1”


  //”Micom” + “Script”


  //1.1 + 1.1


  //”apple” + “List”
```

※演算子の左側が文字列であり、配列も文字列に揃えるので”List”となります。

## 3.2 減算 （ - ）

integer 型または float 型の値が演算子の左側であり、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。
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。
＜プログラム記述例＞

```
$ 2 – 1.1
1

$ 6.5 – 2
4.5

$ 2.3 – “0.3”
2

$12 – “11apple”
1

$ 12 – “apple”
12
```

```
//2 – 1


//6.5 – 2


//2.3 – 0.3


//12 – 11


//12 - 0
```

## 3.3 乗算 （*）

演算子の左側が string 型の場合、演算子の右側のデータ型が float 型や string 型であっても integer 型に統一され、文字列の重複回数になります。

この際、演算子の右側のデータ型が string 型であり、かつ文字列の先頭に数値が含まれる場合、文字の直前までの数値が数字として読み取られます。

演算子の左側が integer 型もしくは float 型で、演算子の右側が string 型の場合、文字の直前までの数値が数字として読み取られます。

この際、文字列の先頭に数字を含まない場合には 1 として演算されます。

“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 0 となります。

＜プログラム記述例＞

```
$ 3 * 1.5
3

$ 1.5 * 5
7.5

$ 1.1 * “2.0”
2.2

$ 3 * “string”
0

“string” * 2.3
“stringstring”
```

```
// 3 * 1


//1.5 * 5


//1.1 * 2.0


//3 * 0

//”string”を 2 回重複する意味
※演算子の右側の 2.3 は integer 型に統一される
```

## 3.4 除算 （/）

割り切れない演算の場合は、少数第７位を四捨五入して返します。

integer 型または float 型のデータ型の値が演算子の左側であり、演算子の右側が string 型の場合、文字列の先頭が数値であればその数値が数字として扱われます。

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不可です。

＜プログラム記述例＞

```
$ 4 / 2.0
2

$ 2.0 / 3
0.666667

$ 2.2 / “2.2”
1
```

```
//4 / 2


//2.0 / 3.0


//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 型のデータ型の値が演算子の左側である時、

演算子の右側にある string 型の値は､数字のみの場合その数値が数字として扱われます。

また、演算子の右側が string 型で、文字列の先頭が数値の場合、文字の直前までの数値が数字として読み取られます。

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 型の値が演算子の左側にある時の演算は出来ません。

list 型の値が演算子の左側である時、演算子の右側が integer 型、float 型、string 型のいずれであってもエラーとなります。

※list 型どうしの剰余演算はできません。

＜プログラム記述例＞

```
$ 6 % 2.5
0

$ 6.0 % 2.5
1

$ 18 % “8”
2

$ 18.0 % “8.5”
1

$ [3,6,9] % 3
Unimplemented

$ [1,2,”a”] % 2
Unimplemented
```

```
//6 % 2


//6.0 % 2.5


//18 % 8


//18.0 % 8.5


//List 型の演算は不可


//List 型の演算は不可
```

## 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、

演算子の右側が文字列の先頭に数字をもつ string 型の場合、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には、その値は数字の 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。

＜プログラム記述例＞

```
$ 2 ** 2.5
4

$ 2.0 ** 2.5
5.65685

$ 2 ** “2.0”
4
```

```
// 2 ** 2


// 2.0 ** 2.5


// 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。
＜プログラム記述例＞

```
$ var a = 1                    //初期値を a = 1 に設定
$ println(a++)                 //初期値の表示後、a += 1 している
1

$ println(a)                   //一つ前の演算結果を表示している
2

$ println(a++)                 //現在の値を表示し、その後 a += 1
2                              している

$ println(a)                   //一つ前の演算結果を表示している
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 を返す演算子です。
＜プログラム記述例＞

```
$ println(++a)
2

$ println(a)
2

$ println(++a)
3

$ println(a)
3
```

```
$ 0b0110 & 0b0101
4

$ 0b1110 & 010111
6
```

```
//0b0100 となり 10 進数の 4 を返す

//0b0110 となり 10 進数の 6 を返す
```

```
//初期値 a に a+=1 し、その演算結果を表示している

//現在の値を表示している

//前の値 a に a+=1 した値を表示している

//現在の値を表示している
```

### 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 を返す演算子です。
＜プログラム記述例＞

```
$ 0b0110 | 0b0101
7

$0b1110 | 0b0111
15
```

```
//0b0111 となり 10 進数の 7 を返す


//0b1111 となり 10 進数の 15 を返す
```

### 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 を返す演算子です。
＜プログラム記述例＞

```
$ 0b0110 ^ 0b0101
3

$ 0b1110 ^ 0b0111
9
```

```
//0b0011 となり 10 進数の 3 を返す



//0b1001 となり 10 進数の 9 を返す
```

### 3.9.4 NOT ( ~ )

値を反転させる演算子です。
＜プログラム記述例＞

```
$ ~1
-2

$ ~2
-3
```

```
//2 進数 0001 を反転させる
//2 進数 1110 は 10 進数の-2

//2 進数 0010 を反転させる
//2 進数 1101 は 10 進数-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です。
＜プログラム記述例＞

```
$ 8 >> 1
4

$ 8 >> 2
2
```

```
//2 進数 1000 を右に 1bit ずらす
//2 進数 0100 は 10 進数の 4

//2 進数 1000 を右に 2bit ずらす
//2 進 0010 は 10 進数の 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です。

＜プログラム記述例＞

```
$ 2 << 1
4

$ 2 << 2
8
```

```
//2 進数 0010 を左に 1bit ずらす
//2 進数 0100

//2 進数 0010 を左に 2bit ずらす
//2 進数 1000
```

## 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 の値です。

### 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 を返します。
＜プログラム記述例＞

```
$ 1 && 1
true

$ 1 && 0
false
```

```
//両辺とも 1 なので true を返す

//両辺が 1 でないので false を返す
```

### 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。
＜プログラム記述例＞

```
$ 1 || 1
true

$ 1 || 0
true

$ 0 || 0
false
```

```
//両辺 1 なので true を返す

//両辺が等しくないので true を返す

//両辺 0 なので false を返す
```

### 3.10.3 否定（!）

1 は初期状態が true とし、 0 は初期状態が false とします。
true ならば false に、false ならば true を返します。
＜プログラム記述例＞

```
$ !1
false

$ !0
true
```

```
//1(true)を否定するので false


//0(false)を否定するので 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。
＜プログラム記述例＞

```
$ 1 == 1
true

$ 1 == 0
false

$ “string” == “string”
true
```

```
//右辺と左辺が等しいので true を返す


//右辺と左辺が等しくないので false を返す

//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。
＜プログラム記述例＞

```
$ 1 != 1
false

$ 1 != 0
true
```

```
//左辺と右辺が等しいので false を返す


//左辺と右辺が等しいので true を返す
```

### 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。
＜プログラム記述例＞

```
$ 3 < 2
false
$ 6 < 8
true
```

```
//左辺より右辺が大きいので false を返す

//左辺が右辺未満なので true を返す
```

### 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。
＜プログラム記述例＞

```
$ 1 <= 2
true

$ 4 <= 3
false
```

```
//左辺が右辺以下なので true を返す


//左辺が右辺以上なので false を返す
```

### 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。
＜プログラム記述例＞

```
$ 6 > 9
false

$ 4 > 2
true
```

```
//左辺が右辺未満なので false を返す


//左辺が右辺より大きいので true を返す
```

### 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。
＜プログラム記述例＞

```
$ 5 >=1
true

$ 2 >= 4
false
```

```
//左辺が右辺以上なので true を返す


//左辺が右辺以下なので false を返す
```

## 3.12 演算子の優先順位

※この表において番号が小さいほど優先順位が高くなります。

| 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することもできます。

### 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。
＜プログラム記述例＞

```
$ var i = 5
$ if(i < 5){
        println(“i < 5”)
 }else if(i == 5){
        println(“i == 5”)
 }else {
        println(“i > 5”)
 }
i == 5
```

```
//初期値を i= 5 に設定
//もし i<5 ならば
        // i < 5 と表示
//もし i == 5 ならば
        // i == 5 と表示
//もしそれ以外ならば
        // i > 5 と表示

//初期値は i = 5 だったので i == 5 と表示
```

## 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、{ }内の処理を実行します。
　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。
＜プログラム記述例＞

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

```
//変数宣言
//変数 sum = 0 を宣言
//i を 1 から 10 まで 1 ずつカウントアップ
//その合計を sum に入れる

//合計 sum を表示
```

●変数宣言を引数 1 でする場合
※上記プログラムと動作は同じ
＜プログラム記述例＞

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

```
//変数 sum = 0 をここで宣言
//for 文の中で変数 i を宣言し、i を 1 から 10
まで 1 ずつカウントアップ
//その合計を sum に入れる
//合計 sum を表示
```

## 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。
＜プログラム記述例＞

```
$ var i = 1                          //変数 i = 1 を宣言
$ var sum = 0                        //変数 sum = 0 を宣言
$ while(i <= 10){                    //i <= 10 以下である間
        Sum += i                     //sum に i ずつ増やした合計を入れる
        i++                          //i を 1 ずつ増やす
 }
$ println(sum)                       //合計 sum を表示する
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。
＜プログラム記述例＞

```
$ var i = 1                          //変数 i = 1 を宣言
$ do {
        println(i)                   //i を表示
        i++                          //i を 1 ずつ増やす
 }while(I <= 3)                      //i <= 3 である間
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 : 終了値です。
引数 3 : 増分指定です。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。
＜プログラム記述例＞

```
$ var i                          //変数 i を宣言
$ repeat(i = 1.5){               //i を 1 から 5 まで繰り返す
        println(i)               //i を表示する
 }
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。
＜プログラム記述例＞

```
$ times(5){                      //5 回繰り返す
        println(“hello”)         //hello と回数分表示する
 }
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。
＜プログラム記述例＞

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

```
//変数 a を宣言
//list = [1,2,3,4]を宣言
//list に含まれる値が変数 a に代入される
//変数 a に代入された値を表示する
```

# 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。

＜プログラム記述例＞

```
$ var a = [1,2,3]
a.append(5)
4

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

$ a
[ 1, 2, 3, 5, 7 ]
```

```
//a = [1,2,3]を宣言する
//5 を末尾に追加する
//a の中にある要素数を返す


//a の中身を表示
// 7 を末尾に追加する
//a の中にある要素数を返す


// a の中身を表示
```

## 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。

＜プログラム記述例＞

```
$ a = [0,1,2,4]                    //a を宣言
[ 0, 1, 2, 4 ]                     //a の中身を表示

$ a.insert(3.8)                    //配列の 3 番目に 8 を挿入
5                                  //要素数表示

a
[ 0, 1, 2, 8, 4 ]                  //a の中身を表示

$ a.insert(5,9)                    //配列の 5 番目に 9 を挿入
6                                  //要素数表示

$ a
[ 0, 1, 2, 8, 4, 9 ]               //a の中身を表示
```

## 5.3 配列名.clear()

配列名の要素を全て空にします。

＜プログラム記述例＞

```
$ a = [1,2,3]
[ 1, 2, 3 ]

$ a
[1, 2, 3 ]

$ a.clear()
0

$ a
[ ]
```

```
//a を宣言
//a の中身を表示



// a の中身を表示

//a の中身をクリアする
//要素が 0 になっている

//現段階の a の中身を表示
//クリアされて要素なし
```

### 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。
＜プログラム記述例＞

```
$ a = [1,2,3,4]
[ 1, 2, 3, 4 ]

$ a.remove(3)
3

$ a
[ 1, 2, 4 ]
```

```
//a を宣言
//a の中身を表示

//a の要素から 3 を取り除く


//a の要素を表示
```

### 5.5 配列名.len()

配列の要素数を返します。
＜プログラム記述例＞

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

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

```
//list の宣言
//list 配列の要素数を返す


//要素のない list を作る

//要素を表示
```

# 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。
＜プログラム記述例＞

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

$ a
“apple,orange”
```

```
//変数 a を宣言し、”apple”を代入
変数 a の文字列の後ろに”,orange”を追加
//変数に代入されている文字列の要素数を返す

//変数 a の要素を返す
```

## 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。
＜プログラム記述例＞

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

```
//変数 a を宣言し、”apple”を代入
//変数 a の 1 番目の文字から 3 文字分返す
```

## 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。

＜プログラム記述例＞

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

```
//変数 a を宣言し、”     apple”を代入

//変数 a の要素を返す
//変数 a に代入された要素”     apple”から空白を取り除いたものを表示する
```

## 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。

＜プログラム記述例＞

```
$ var a = “apple”
$ a.len()
5
```

```
//変数 a を宣言し、”apple”を代入
//変数 a に含まれる文字列の文字数を返す
```

# 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、string 型、list 型の評価方法と同様に扱われます。
＜プログラム記述例＞

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

$ cos(60.0 * pi / 180)
0.5

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

```
//sin(30.0 * 3.14159 / 180)


//cos(60.0 * 3.14159 / 180)


//tan(45.0 * 3.14159 /180)
```

## 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、string 型、list 型の評価方法と同様に扱われます。

＜プログラム記述例＞

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

$ acos(1) * 180 / 3.14
0

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

```
//アークサインを返す


//アークコサインを返す


//アークタンジェントを返す
```

## 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、string 型、list 型の評価方法と同様に扱われます。

＜プログラム記述例＞

```
$ sinh(1)
1.1752

$ cosh(1)
1.54308

$ tanh(1)
0.761594
```

```
//ハイパボリックサインを返す


//ハイパボリックコサインを返す


//ハイパボリックタンジェントを
返す
```

7.4

### 切り捨て floor()

引数の数値の切り捨てを行う関数です。
string 型は 0 として扱います。
＜プログラム記述例＞

```
$ floor(13.9)
13

$ floor(13.4)
13

$ floor(“apple”)
0

$floor(“5.8apple”)
5
```

```
//13.9 を切り捨てる


//13.4 を切り捨てる


//”apple”は string 型なので 0 扱い


//”5.8apple”は 5.8 のみ数値として扱い、切り捨てられる
```

### 7.5 切り上げ ceil()

引数の数値の切り上げを行う関数です。
string 型は 0 として扱います。
＜プログラム記述例＞

```
$ ceil(14.2)
15

$ ceil(14.6)
1

$ ceil(“apple”)
0

$ ceil(“13.1apple”)
14
```

```
//14.2 を切り上げ


//14.6 を切り上げ


//”apple”は string 型なので 0 扱い


//”13.1apple”は 13.1 のみ数値として扱い、切り上げられる
```

## 7.6 四捨五入 round()

引数の数値の四捨五入を行う関数です。
string 型は 0 として扱います。
＜プログラム記述例＞

```
$ round(4.6)
5

$ round(3.4)
3

$ round(“apple”)
0

$ round(“1.2apple”)
1
```

```
//4.6 を四捨五入する


//3.4 を四捨五入する


//”apple”は string 型なので 0 扱い


//”1.2apple”は 1.2 のみ数値として扱い、
四捨五入される
```

## 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 の数値を指数、引数 2 の数値を底とした時のべき乗を返す関数です。
＜プログラム記述例＞

```
$ pow(2,3)
9

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

```
//3²


(-3)²
```

## 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。
＜プログラム記述例＞

```
$ log(2)
0.693147

$ log(1)
0

$ log(E)
1
```

```
//log(2)の値を返す


//log(1)の値を返す


//log(E)の値を返す
```

## 7.9 常用対数 log10()

底が 10 の対数(常用対数)を返す関数です。
＜プログラム記述例＞

```
$ log10(10)
1

$ log10(1)
0

$ log10(1000)
3
```

```
//log10 10 の値を返す


//log10 1 の値を返す


//log10 1000 の値を返す
```

## 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。
＜プログラム記述例＞

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

```
//e^1
```

## 7.11 平方根 sqrt()

指定した値の平方根を返す関数です。
＜プログラム記述例＞

```
$ sqrt(4)
2

$ sqrt(2)
1.41421
```

```
//√4 の値を表示


//√2 の値を表示
```

## 7.12 立方根 cbrt()

指定した値の立方根を返す関数です。
＜プログラム記述例＞

```
$ cbrt(8)
2

$ cbrt(64)
4
```

```
//8 = 2^3


//64 = 4^3
```

## 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。
＜プログラム記述例＞

```
$ hypot(3,4)
5

$ hypot(1,1)
1.41421

$hypot(1.41421,1.41421)
1.99999
```

```
//√(3²+4²)


//√(1²+1²)


//√(1.41421²+1.41421²)
```

## 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。
※初期状態では srand(1)になっています。
＜プログラム記述例＞

```
$ random()
0.690001

$ random()
0.505418

$ random()
0.591491
```

```
//無造作に選んだ数値を出力


//無造作に選んだ数値を出力


//無造作に選んだ数値を出力
```

## 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。
※初期状態では srand(1)になっています。
＜プログラム記述例＞

```
$ srand(2)

$random()
0.380002

$ random()
0.320836

$random()
0.677563

$random()
0.518079
```

```
//乱数の種を初期設定(srand(1))からsrand(2)に変更
//無造作に選んだ数値を出力


//無造作に選んだ数値を出力


//無造作に選んだ数値を出力


//無造作に選んだ数値を出力
```

```
$ srand(3)

$ random()
0.0700031

$ random()
0.136253

$ random()
0.763635

$ random
0.481374
```

```
//乱数の種を初期設定(srand(1))からsrand(3)に変更

//無造作に選んだ数値を出力


//無造作に選んだ数値を出力


//無造作に選んだ数値を出力


//無造作に選んだ数値を出力
```

```
$ srand(4)

$random()
0.760004

$ random()
0.95167

$ random()
0.849707

$random()
0.444668
```

```
//乱数の種を初期設定(srand(1))からsrand(4)に変更
//無造作に選んだ数値を出力


//無造作に選んだ数値を出力


//無造作に選んだ数値を出力


//無造作に選んだ数値を出力
```

```
$ srand(5)

$random()
0.450005

$ random()
0.767088

$ random()
0.935779

$random()
0.673004
```

```
//乱数の種を初期設定(srand(1))からsrand(5)に変更
//無造作に選んだ数値を出力


//無造作に選んだ数値を出力


//無造作に選んだ数値を出力


//無造作に選んだ数値を出力
```

# ８．時間関数

## 8.1 time()

MicomScript を起動してからの秒数を返す関数です。

＜プログラム記述例＞

```
$ time()
10467

$ time()
10474
```

```
//起動してから 10467 秒経過


//起動してから 10474 秒経過
```

## 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。

＜プログラム記述例＞

```
$ millis()
10955486
```

```
//プログラムの実行を開始した時から現在までの時間
```

## 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。

＜プログラム記述例＞

```
$ micros()
-1920661074
```

```
//プログラムの実行を開始した時から現在までの時間
```

## 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。
＜プログラム記述例＞

| | |
|---|---|
| $ delay(100) | //100ms 止める処理 |

# ９．画面表示関数

## 9.0 LCD ピンについて

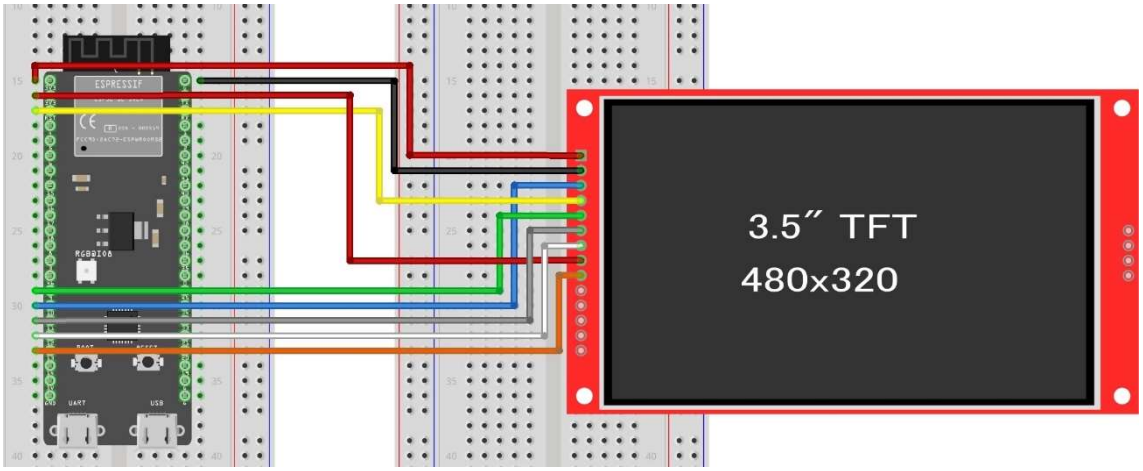
バイナリ版の場合 LCD のピンは固定されています。

LCD と ESP32 のピン設定

| LCD のピン | ESP32 のピン | 配線図の線の色 |
|---|---|---|
| VCC | 3.3V ピン | 赤色 |
| GND | GND ピン | 黒色 |
| CS | 10 番ピン | 青色 |
| RESET | RST ピン | 黄色 |
| D/C | 9 番ピン | 緑色 |
| SDI(MOSI) | 11 番ピン | 灰色 |
| SCK | 12 番ピン | 白色 |
| LED | 3.3V ピン | 赤色 |
| SDO(MISO) | 13 番ピン | 橙色 |

となっています。

下に配線例を示します



## 9.1 setColor()

setColor(引数 1,引数 2)
関数で使用される色を指定する関数です。図形は引数 1 に入力した色で描画されます。
初期値は引数 1 が黒、引数 2 が白で変更するまでその色に固定されます。
色は 16 ビットカラーで指定しますが、以下の色は文字で指定することができます。

| | | | | |
|---|---|---|---|---|
| BLACK | WHITE | RED | BLUE | GREEN |
| YELLOW | PINK | ORANGE | CYAN | NAVY |
| PURPLE | VIOLET, | MAROON | OLIVE | DARKGREEN |
| BROWN | GOLD | SILVER | SKYBLUE | GREENYELLOW |
| MAGENTA | DARKCYAN | DARKGREY | LIGHTGREY | LIGHTPINK |

＜プログラム記述例＞

```
$ setColor(RED)
$ setColor(RED,BLUE)
```

```
//赤に設定
//文字を赤に設定し、背景を青に設定
```

## 9.2 textSize()

textSize(引数)
文字のサイズをドット数指定で変更する関数です。初期値は 1 です。
＜プログラム記述例＞

```
$ textSize(10)
```

```
//文字のサイズを 10 に設定
```

## 9.3 clearScreen()

画面全体を白く塗りつぶす関数です。
＜プログラム記述例＞

```
$ drawRect(10,10,10,10)
$ clearScreen()
```

```
//座標指定
//画面全体を白く塗りつぶす
```

## 9.4 fillScreen()

画面全体を引数１で入力した色で塗りつぶす関数です。
＜プログラム記述例＞

```
$ setColor(RED)      //赤に設定
$ fillScreen()       //画面を設定色に塗りつぶす
```

## 9.5 drawLine()

drawLine(x1,y1,x2,y2)
座標(x1,y1)から,座標(x2,y2)に直線を描画する関数です。
＜プログラム記述例＞

```
$ drawLine(10,10,200,200)

//座標(10,10)から座標(200,200)に直線を描画
```

## 9.6 drawPixel()

drawPixel(x,y)
座標(x,y)に点を描画する関数です。
＜プログラム記述例＞

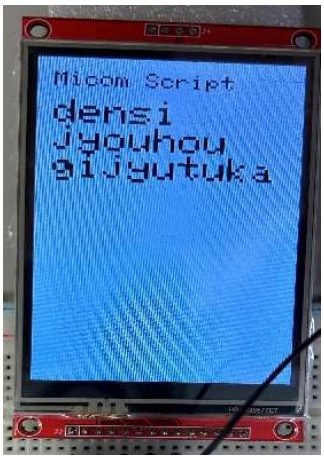
```
$ drawPixel(100,100)      //(100,100)に点を描画
```

## 9.7 drawString()

drawString(str,x,y)
座標(x,y)に文字列 str を描画する関数です。文字の背景は引数 2 に入力した色で塗りつぶされます。
＜プログラム記述例＞

```
$ drawstring(“Micom Script”,25,25)
$ textsize(10)
$ drawstring(“densi”,25,100)
$ drawstring(“jyouhou”,25,150)
$ drawstring(“gijyutuka”25,200)
```

```
//座標(25,25)に”Micom Script”と表示
//フォントサイズを 10 に設定
//座標(25,100)に”densi”と表示
//座標(25,150)に”jyouhou”と表示
//座標(25,200)に”gijyutuka”と表示
```



## 9.8 drawRect()

drawRect(x,y,w,h)
四角形の左上座標(x,y)、幅 w、高さ h を指定し四角形を描画する関数です。
＜プログラム記述例＞

```
$ drawRect(100,100,20,20)
```

```
//四角形の左上の座標を(100,100)にし、幅を 20、高さを 20 とした四角形を描画
```

## 9.9 fillRect()

fillRect(x,y,w,h)
四角形の左上座標(x,y)、幅 w、高さ h を指定し、
塗りつぶされた四角形を描画する関数です。
＜プログラム記述例＞

```
$ fillRect(200,200,25,25)
```

```
//四角形の左上の座標を(200,200)にし、幅を 25、高さを 25 と塗りつぶされた四角形を描画
```

## 9.10 drawCircle()

drawCircle(x,y,r)
中心点の座標(x,y)と半径 r を指定し、円を描画する関数です。
＜プログラム記述例＞

```
$ drawCircle(100,100,10)
```

```
//中心座標(100,100)で半径 10 の円を描画
```

## 9.11 fillCircle()

fillCircle(x,y,r)
中心点の座標(x,y)、半径 r を指定し、
塗りつぶされた円を描画する関数です。
＜プログラム記述例＞

```
$ fillCircle(100,200,10)
```

```
//中心座標(100,200)で半径 10 の塗りつぶされた円を描画
```

## 9.12 drawTriangle()

drawTriangle(x1,y1,x2,y2,x3,y3)
座標(x1,y1)、座標(x2,y2)、座標(x3,y3)を指定し、三角形を描画する関数です。
＜プログラム記述例＞

```
$ drawTriangle(100,100,150,100,100,150)
```

```
//座標(100,100)、座標(150,100)、座標(100,150)の三角形を描画
```

## 9.13 fillTriangle()

fillTriangle(x1,y1,x2,y2,x3,y3)
座標(x1,y1)、座標(x2,y2)、座標(x3,y3)を指定し、
塗りつぶされた三角形を描画する関数です。
＜プログラム記述例＞

```
$ fillTriangle(200,200,250,200,200,250)
```

```
//座標(200,200)、座標(250,200)、座標(200,250)の塗りつぶされた三角形を描画
```

## 9.14 drawVline()

drawVline(x,y,h)
座標(x,y)と高さ h を指定し、垂直線を描画する関数です。
線は指定した座標から下方向に描画されます。
＜プログラム記述例＞

```
$ drawVline(10,10,10)
```

```
//座標(10,10)と高さ 10 を結ぶ垂直線を描画
```

## 9.15 drawHline()

drawHline(x,y,v)
座標(x,y)と横幅 v を指定し、水平線を描画する関数です。
線は指定した座標から右方向に描画されます。
＜プログラム記述例＞

```
$ drawHline(10,10,10)
```

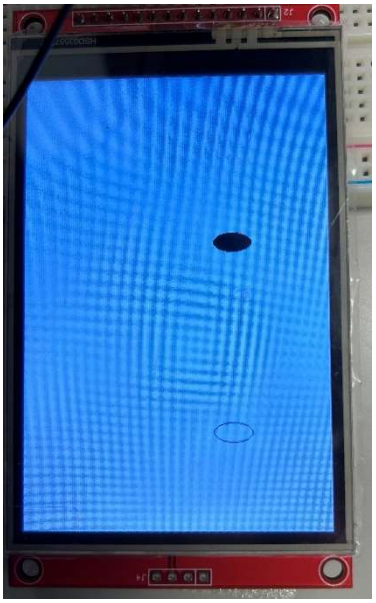
```
//座標(10,10)と幅 10 を結ぶ水平線を描画
```

## 9.16 drawEllipse()

drawEllipse(x,y,rx,ry)
中心座標(x,y)、x 軸の半径 rx,y 軸の半径 ry を指定し、楕円の線を描画する関数です。
＜プログラム記述例＞

```
$ drawEllipse(100,100,20,10)
```

```
//中心座標(100,100),x 軸の半径 20,y 軸の半径 10 の楕円を描画する
```



## 9.17 fillEllipse()

fillEllipse(x,y,rx,ry)
中心座標(x,y)、x 軸の半径 rx,y 軸の半径 ry を指定し、
塗りつぶされた楕円を描画する関数です。
＜プログラム記述例＞

```
$ fillEllipse(100,300,20,10)
```

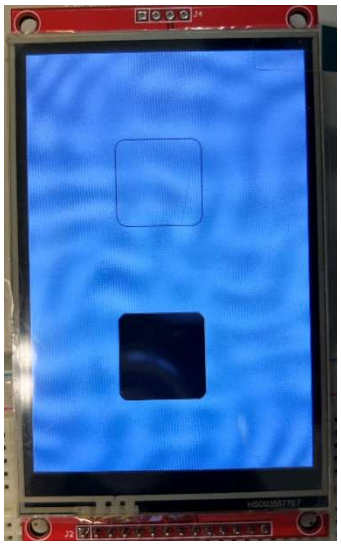
```
//中心座標(100,300),x 軸の半径 20,y 軸の半径 10 の塗りつぶされた楕円を描画する
```

## 9.18 drawRoundRect()

drawRoundRect(x,y,w,h,r)
左上隅の座標(x,y)、横幅 w、高さ h、丸角の半径 r を指定し、
角の丸い四角を描画する関数です。
＜プログラム記述例＞

```
$ drawRoundRect(100,100,10,10,5)
```

```
//左上の座標を(100,100)とし、横幅 10、高さ 10、
丸角の半径を 5 とした角の丸い四角を描画する
```



## 9.19 fillRoundRect()

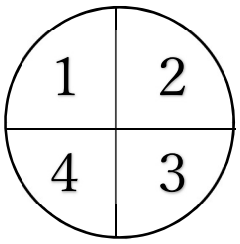
fillRoundRect(x,y,w,h,r)
左上隅の座標(x,y)、横幅 w、高さ h、丸角の半径 r を指定し、
塗りつぶされた角の丸い四角形を描画する関数です。
＜プログラム記述例＞

```
$ fillRoundRect(100,300,10,10,5)
```

```
//左上の座標を(100,300)とし、横幅 10、高さ 10、
丸角の半径を 5 とした角の丸い四角を描画する
```

## 9.20 drawCircleHelper()

drawCircleHelper(x,y,r,c)
座標(x,y)、円の半径 r、描画したい円弧の範囲 c を指定し、円弧を描画する関数です。
c は 4 ビットの 2 進数として扱います。
ビットが 1 になっている位の範囲に円弧を描画します。




c = 1 のとき　0001→ 1 桁目が 1 → 1 の範囲に円弧を描画します。
c = 6 のとき　0110→ 2 , 3 桁目が 1 → 2 , 3 の範囲に円弧を描画します。
c = 11 のとき　1011→ 1 , 2 , 4 桁目が 1 → 1 , 2 , 4 の範囲に円弧を描画します。

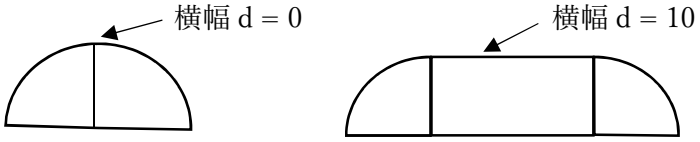
＜プログラム記述例＞

```
drawCircleHelper(200,200,100,3)
```

```
//座標(200,200)、円の半径 100、描画範囲 c = 3 とし、円弧を描画
```

## 9.21 fillCircleHelper()

fillCircleHelper(x,y,r,c,d)
座標(x,y)、円の半径 r、描画したい円弧の範囲 c(1:上半分, 2:下半分, 3:両方)、横幅 d を指定し、描画する関数です。




＜プログラム記述例＞

```
fillCircleHelper(150,100,100,1,20)
```

```
//座標(150,150)、円の半径 100、描画範囲 c = 1 とし、
横幅 20 として円弧を描画
```

# １０．GPIO 関数

pin　　→ ピン番号
io　　→ INPUT / OUTPUT / INPUT_PULLUP
HL　　→ HIGH / LOW (1/0)
freq　　→ 周波数
bit　　→ bit 数
ch　　→ チャンネル
tones　→ デューティー比

## 10.1 pinMode()

pinMode(pin , io)
上のように引数を記述します。
ピンの番号とピンの動作を入力か出力に設定する関数です。
＜プログラム記述例＞

```
pinMode(25,OUTPUT)　　//25 番ピン出力設定
```

## 10.2 digitalWrite()

digitalWrite(pin , HL)
上のように引数を記述します。
HIGH または LOW を、指定したピンに出力する関数です。
＜プログラム記述例＞

```
digitalWrite(25,HIGH)　　//25 番ピン HIGH 出力設定
```

## 10.3 analogRead()

analogRead(pin)
上のように引数を記述します。
指定したアナログピンから値を読み取る関数です。
＜プログラム記述例＞

```
var a = analogRead(25)　　//25 番ピンからアナログ値を読み取る
```

## 10.4 digitalRead()

digitalRead(pin)
上のように引数を記述します。
指定したピンの値を読み取る関数。その結果は HIGH または LOW になります。
＜プログラム記述例＞

```
var a = digitalRead(25)
```

```
//25 番ピンの値を読み取り変数 a に代入
```

## 10.5 ledcSetup()

ledcSetup(ch , freq , bit)
上のように引数を記述します。
指定したチャンネル(ch)に周波数(freq)とデューティー比を表す bit 数を決める関数です。
チャンネルは 0〜15 まで指定できる。
＜プログラム記述例＞

```
ledcSetup(1,12000,16)
```

```
//チャンネル 1 、周波数 12000、16bit を設定
```

## 10.6 ledcAttachPin()

ledcAttachPin(pin , ch)
上のように引数を記述します。
LED PWM で利用するピンとチャネルを結びつける関数です。
＜プログラム記述例＞

```
ledcAttachPin(25,1)
```

```
//25 番ピンとチャンネル 1 をつなげる
```

## 10.7 ledcDetachPin()

ledcDetachPin(pin)
上のように引数を記述します。
LED PWM で利用するピンとチャネルの結びつきを解除する関数です。
＜プログラム記述例＞

```
ledcDetachPin(25)
```

```
//25 番ピンとチャンネルの結びつきを解除
```

## 10.8 ledcWrite()

ledcWrite(ch , tones)
上のように引数を記述します。
LED PWM を利用して、指定したデューティー比で PWM 出力する関数です。
＜プログラム記述例＞

```
ledcWrite(1,500)
```

## 10.9 ledcWriteTone()

ledcWrite(ch , freq)
上のように引数を記述します。
LED PWM を利用して、指定した周波数を PWM 出力する関数です。
＜プログラム記述例＞

```
ledcWriteTone(1,6000)
```

## 10.10 ledcRead()

ledcRead(ch)
上のように引数を記述します。
LED PWM のチャネルに指定したデューティー比を取得する関数です。
＜プログラム記述例＞

```
var a = ledcRead(ch)
```

## 10.11 ledcReadFreq()

ledcReadFreq(ch)
上のように引数を記述します。
LED PWM のチャネルに指定した周波数を取得する関数です。
＜プログラム記述例＞

```
var a = ledcReadFreq
```

## 10.12 Output()

Output(pin)
上のように引数を記述します。
特定の pin を出力に設定します。
＜プログラム記述例＞

```
Var led = Output(12)        //12 番ピンに LED を接
```

## 10.13 high()

特定のピンの出力を HIGH にする関数です。
以下のように記述します。
＜プログラム記述例＞

```
led.high()          //HIGH 出力
```

## 10.14 low()

特定の pin の出力を LOW にする関数です。
以下のように記述します。
＜プログラム記述例＞

```
Led.low()           //LOW 出力
```

## 10.15 toggle()

HIGH/LOW 出力を反転させる関数です。
以下のように記述します。
＜プログラム記述例＞

```
Led.toggle()        //反転
```

## 10.16 onPress()

プッシュスイッチを押したとき動作する関数です。
以下のように記述します。
※Button 関数は後に解説あり
＜プログラム記述例＞

```
Var led = Output(12)          //LED を 12 番ピンに接続する
function b1_on(){
        led.high()
}
var b1 = Botton(16)
b1.onPress(“b1_on”)           //b1 のスイッチを押すと LED が点灯する
```

## 10.17 onRelease()

プッシュスイッチを離したとき動作する関数です。
以下のように記述します。
＜プログラム記述例＞

```
Var led = Output(12)          //LED を 12 番ピンに接続する
function b2_on(){
        led.low()
}
b2.onRelease(“b2_on”)         //b2 のスイッチを押すと LED が消灯する
```

## 10.18 button()

プッシュスイッチ（ボタン）を設定する関数です。
以下のように記述します。
＜プログラム記述例＞

```
var button = Button(5)      //5 番ピンにプッシュスイッチ（ボタン）を接続
```

## 10.19 status()

以下のように記述します。
＜プログラム記述例＞

```
var s = led.status()        //状態取得
```

## 10.20 temperature() ・ humidity()

温湿度センサ AM2320 に対応した。
以下のように記述することで温湿度センサの値を読み取ることが可能である。
＜プログラム記述例＞

```
var sensor = AM2320()

var temp = sensor.temperature()
var humid = sensor.humidity()
```

```
//sensor 変数に AM2320

//temp 変数に温度の値を入力
//humid 変数に湿度の値を入力
```

# 11.タイマー割り込み・マルチタスク

## 11.1 timer() ・ setTimer()

timer()：タイマーを設定する関数です。
setTimer()：特定の関数を指定した周期で実行する関数です。

以下のように記述します。
※プログラム例から抜粋した一部
＜プログラム記述例＞

```
var led = Output(12)
function onTimer(){
        led.toggle()
}
var timer = Timer()
timer.setTimer("onTimer",250)
```

```
//LED:12pin




//タイマーセット
```

## 11.2 stop() ・ resume()

stop()：タイマーの停止
resume()： タイマーの再開
以下のように記述します。
＜プログラム記述例＞

```
timer.stop()
timer.resume()
```

```
//タイマー停止
//タイマー再開
```

# 12.Wi-Fi 通信

2 つのマイコンを同じアクセスポイントに接続することで少し離れた場所からでもマイコンでもう一方のマイコンをリモート制御することができます。
さらに、通信のリクエスト待ち状態であっても、ほかの処理と並行動作が可能になっています。

## 12.1 Wi-FI　AP との接続

下のように記述することで Wi-Fi AP との接続が可能です。
＜プログラム記述例＞

```
WiFi.begin(“ssid”,”password”)　　//Wi-Fi 接続
```

## 12.2 Wi-Fi AP との接続状態確認

下のように記述することで Wi-Fi AP との接続状態を確認できます。
＜プログラム記述例＞

```
var state = Wi-Fi.status()　　//Wi-Fi の接続状況を確認
```

戻り値として以下のものがあります

"NO_SHIELD"
"IDLE_STATUS"
"NO_SSID_AVAIL"
"SCAN_COMPLETED"
"CONNECTED"
"CONNECT_FAILED"
"CONNECTION_LOST"
"DISCONNECTED"
"UNKNOWN"

## 12.3 接続情報の表示

以下のように記入することで接続情報を確認できます。
＜プログラム記述例＞

WiFi.print()
WiFi.localIP()
WiFi.dnsIP()
WiFi.gatewayIP()
WiFi.subnetMask()
WiFi.macAddress()

## 12.4 Wi-Fi AP とのリンクを切断する

以下のように記述することで Wi-Fi AP とのリンクを切断することができます。
＜プログラム記述例＞

```
WiFi.desconnect()        //リンク切断
```

## 12.5 Wi-Fi UDP 取得

以下のように記述することで Wi-Fi UDP 取得が可能です。
＜プログラム記述例＞

```
var udp = WiFi.UDP()        // Wi-Fi UDP の取得
```

## 12.6 送信元アドレス・ポート

以下のように記述することで、送信元アドレス、ポートを指定できます。
＜プログラム記述例＞

```
var ipaddress = udp.remoteIP()          //ipaddress 宣言
var port = udp.remotePort()             //port 宣言

udp.send("IP アドレス","ポート番号","   //記入内容
メッセージ")
udp.send("172.xx.xx.xx",60000,"Hello")
```

# 13.その他のライブラリ関数

## 13.1 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。
＜プログラム記述例＞

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

```
//1 を表示
```

◆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。
＜プログラム記述例＞

```
$ print(“print”, 1)
print1$
```

```
//文字列を表示
```

◆引数が 2 進数, 16 進数の場合、10 進数として表示されます。
＜プログラム記述例＞

```
$ print(0b1100)
12$

$ print(0xc)
12$
```

```
//()内の 2 進数を 10 進数で表示


//()内の 16 進数を 10 進数で表示
```

## 13.2 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。
＜プログラム記述例＞

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

```
//表示する
```

◆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。
＜プログラム記述例＞

```
$ println(“print”, 1)
print1
```

```
//表示する
```

◆引数が 2 進数, 16 進数の場合、10 進数として表示されます。
＜プログラム記述例＞

```
$ print(0b1100)
12

$ print(0xc)
12
```

```
//2 進数表記を 10 進数表記にする


//16 進数表記を 10 進数表記にする
```

## 13.3 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。
＜プログラム記述例＞

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

```
//変数 a を宣言
//表示する
```

◆”“内で、¥n を記述すると、記述した位置で改行が行われます。
＜プログラム記述例＞

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

```
//変数 a を宣言
//表示する
```

◆変数を書き出す場合、以下のようにコードを入力する。
＜プログラム記述例＞

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

```
//変数 a を宣言
//表示する
```

■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。 |

## 13.4 max()

●引数の中で最大の要素を返す関数です。

＜プログラム記述例＞

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

```
//MAX 値を表示
```

◆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）

＜プログラム記述例＞

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

$ max(1,2,3,4,”5abc2”)
5
```

```
//最大値を表示


//最大値を表示
```

◆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。

＜プログラム記述例＞

```
$ max(1,2,3,4,”a5bc2”)
4
```

```
//最大値を表示する
```

◆引数が list 型の場合、list の中の要素が比較対象になります。

＜プログラム記述例＞

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

```
//変数 a を宣言し、配列を代入
//最大値を表示
```

## 13.5 min()

●引数の中で最小の要素を返す関数です。

＜プログラム記述例＞

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

```
//最小値を表示する
```

◆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）
＜プログラム記述例＞

| | |
|---|---|
| $ min(1,2,3,4,”5abc2”)<br>1 | //最小値を表示する |

◆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。
＜プログラム記述例＞

| | |
|---|---|
| $ min(1,2,3,4,”a5bc2”)<br>1 | //最小値を表示する |

◆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。
＜プログラム記述例＞

| | |
|---|---|
| $ var a = [1,2,3,4,1.1]<br>$ min(a)<br>1 | 変数 a を宣言し、配列を代入<br>//最小値を表示 |

## 13.6 sum()

●引数の合計値を返す関数です。

＜プログラム記述例＞

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

```
//合計を返す

//list 変数を宣言
//hist の合計を表示
```

◆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。

＜プログラム記述例＞

```
$ sum(5,”world”)
5

$ sum(5,”2world”)
7
```

```
//5 + 0


//5 + 2
```

## 13.7 type()

●引数の型名を返す関数です。

＜プログラム記述例＞

```
$ var a = 1
$ type(a)
”integer”

$ var a = 1.1
$ type(a)
”float”

$ var a = “hello”
$ type(a)
”string”
```

```
//変数 a を宣言
//型名を返す


//変数 a に 1.1 を代入
//型名を返す


//変数 a に hello を代入
//型名を返す
```

## 13.8 int()

●引数の型を integer 型に変換して返す関数です。

＜プログラム記述例＞

```
$ var a = 2.2;
$ type(a)
“float”

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

```
変数 a を宣言し、2.2 を代入
//型名を表示


//a の値を整数型を a に代入

//型名を表示
```

◆引数が string 型の場合、integer 型の 0 を返します。

＜プログラム記述例＞

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

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

```
//変数 b を宣言し、test を代入
//型名を表示


//変数 b に整数型の b を代入する

//型名を表示
```

◆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。）
＜プログラム記述例＞

```
$ var c = [1, 2, “namako”, 3.4]
$ type(c)
“list”
$ c = int(c)
6.4
$ type(c)
“integer”
```

```
//変数 c に配列を代入
//型名を表示

//変数 c に整数型の c を代入する

//型名を表示
```

## 13.9 float()

●引数の型を float 型に変換して返す関数です。
＜プログラム記述例＞

```
$ var a = 2;
$ type(a)
“integer”

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

```
//変数 a を宣言し、2 を代入
//変数 a の型名を表示


//変数 a に float 型の a を代入

//変数 a の型名を表示
```

◆引数が string 型の場合、float 型の 0 を返します。
＜プログラム記述例＞

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

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

```
//変数 b を宣言し、test を代入
//変数 b の型名を表示


//変数 b に float 型の b を代入

//変数 b の型名を表示
```

◆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。）

＜プログラム記述例＞

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

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

```
//変数 c に配列を代入
//型名を表示


//変数 c に float 型の c を代入する

//型名を表示
```

## 13.10 string()

●引数の型を string 型に変換して返す関数です。

＜プログラム記述例＞

```
$ type(string(1.1))
“string”
```

```
// string(1.1)を string 型に変換する
```

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

＜プログラム記述例＞

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

```
//変数 a を宣言し、配列を代入
//引数が list 型なので string 型の文字
列”List”が返される
```

## 13.11 abs()

◆引数の値が integer 型の場合 integer 型の絶対値に変換して返します。

＜プログラム記述例＞

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

```
//-2 を integer 型の絶対値にして返す
```

◆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。

＜プログラム記述例＞

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

```
//引数 float 型なので小数点以下切り捨て
```

◆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。
＜プログラム記述例＞

```
$ abs(“Micom Script”)
0
```

```
//引数が string 型なので integer 型の
0 を返す
```

◆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。
＜プログラム記述例＞

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

```
//変数 a を宣言し、配列を代入する
//配列 a に含まれる要素の合計を返す
```

◆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。
＜プログラム記述例＞

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

```
変数 e を宣言し、配列を代入する
//配列 e に含まれる要素の合計を返す
```

## 13.12 fabs()

●引数の値を float 型の絶対値に変換して返す関数です。
＜プログラム記述例＞

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

```
//float 型の値(-2.2)を絶対値に変換して返す
```

◆引数が string 型の場合、float 型の 0 を返します。
＜プログラム記述例＞

```
$ fabs(“hello”)
0
```

```
//引数が string 型なので float 型の 0 を
返す
```

◆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。）
＜プログラム記述例＞

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

```
変数 a に配列を代入
//配列の要素の合計を返す
```

◆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。
＜プログラム記述例＞

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

```
//変数 e を宣言し、配列を代入する
//配列内の合計を float 型の絶対値
として返す
```

## 13.13 bool()

●引数に指定した値の真理値を返す関数です。

＜プログラム記述例＞

```
$ bool(3)
true

$ bool(0)
false

$ bool(3 < 4)
true

$ bool(3 > 4)
false
```

```
// 3 の真理値を返す


// 0 の真理値を返す


// 3 < 4 の真理値を返す


// 3 > 4 の真理値を返す
```

## 13.14 input()

●ユーザーに値を入力させる関数です。

＜プログラム記述例＞

```
$ input()
Hello
“Hello”
```

```
//ユーザーに値を入力してもらえる状態になる
//ユーザーが入力するところ
```

◆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。

＜プログラム記述例＞

```
$ var a
$ a = input()
6
“6”

$ println(a)
6
```

```
//変数 a を宣言
//変数 a に input()関数を代入
//input()関数に 6 を代入
//代入されたものの表示

//変数 a に代入されているものを表示
```

# 14. ユーザ定義関数

●ユーザが作成する関数です。
ユーザ定義関数名は変数名と同様に半角文字のみ使用可能です。
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。

## 14.1 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。
＜プログラム記述例＞

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

```
//関数名と引数を宣言
//内部処理を記載
```

※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。
＜プログラム記述例＞

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

```
//変数宣言
//繰り返し処理
//合計を求める処理

//sum 関数呼び出し
```

◆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。

＜プログラム記述例＞

```
$ sigma(10)
55

$ sigma(100)
5050
```

```
//初期値 1 から 10 までの合計を返す


//初期値 1 から 100 までの合計を返す
```

※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。

# 15.　コマンド

## 15.1　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                    - print Function　list
undef                      - undefine Variable/Function
exit                     - finish execution
```

## 15.2　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。

## 15.3 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。
＜プログラム記述例＞

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

```
//変数 a を宣言
//変数 b を宣言し、5 を代入
//変数の一覧を表示する
//グローバル変数があることを示している
//変数 a には数値が入っていないので null
//変数 b に 5 が代入されていると確認できる
```

## 15.4　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。
＜プログラム記述例＞

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

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

```
//変数 a,b を宣言
//宣言されている変数 a を削除
//現在ある変数を表示

//現在は変数 b のみ存在する

//変数 a=1, b=2, c=を宣言
//現在存在する変数をすべて削除
//現在ある変数を表示

//何も存在しない
```

## 15.5　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
```

## 15.6　flist

●定義されている関数をすべて表示します。

＜プログラム記述例＞

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

```
//関数 f1()を定義
//関数 f2()を定義
//定義されている関数を表示
```

## 15.7　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。

＜プログラム記述例＞

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

```
//関数 f1()を定義
//関数 f2()を定義
//関数 f1()を削除
//定義されている関数を表示

//関数 f2()が存在していることが確認できる
//関数 f1(){}を定義
//存在する関数をすべて削除
//定義されている関数を表示

//何も存在していない
```

## 15.8　exit

●プログラムを終了します。

## 15.9　reset

●リセットを行います。

## 15.10　clist_ESP

●定数リストを表示します。

```
$ 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
```

# 16. ファイル操作コマンド

## 16.1　files

●作成した全ファイルの一覧を表示します。

```
$ files
56 /s01
263 /Hanoi
```

```
//作成した全ファイルの一覧表示
//s01 というファイルがあることを表示
//Hanoi というファイルがあることを表示
```

## 16.2　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。

```
$ files “検索ワード”
```

```
//検索したいファイルが存在していいるか確認できる
```

## 16.3　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”
```

```
//hanoi というファイルの保存を行う
```

## 16.4　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
```

```
//s01 というファイルの実行を行う
```

## 16.5　remove “/ファイル名”

●指定したファイルを削除します。

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

```
//現在あるファイルを表示する


//s02 のファイルを削除
//現在あるファイルを表示する
```

グラ

＜プログラム記述例＞

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

```
//s01 ファイルのソースプログラムを表示
```

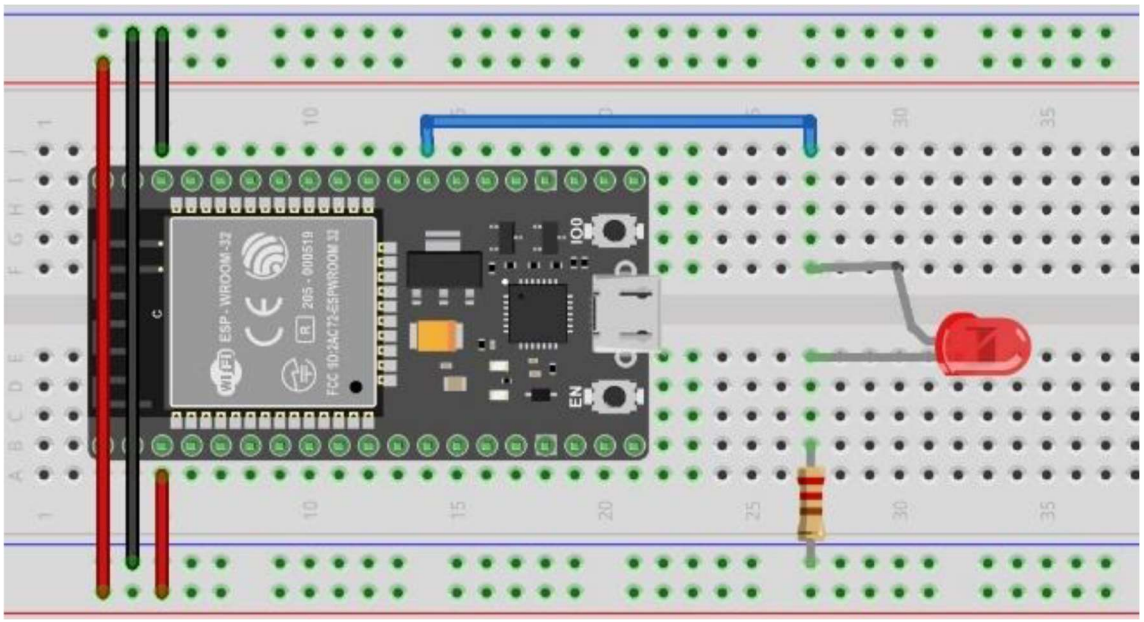
# 17.サンプルプログラム(令和４年度作成分)

いかにいくつかサンプルプログラムを記載してありますが、Micom Scriptの公式HPの方にもほかのデモプログラムを掲載してあります。

## 17.1 LED 点滅

１６pinに接続されたLEDを１秒間隔で点滅させる

回路図例

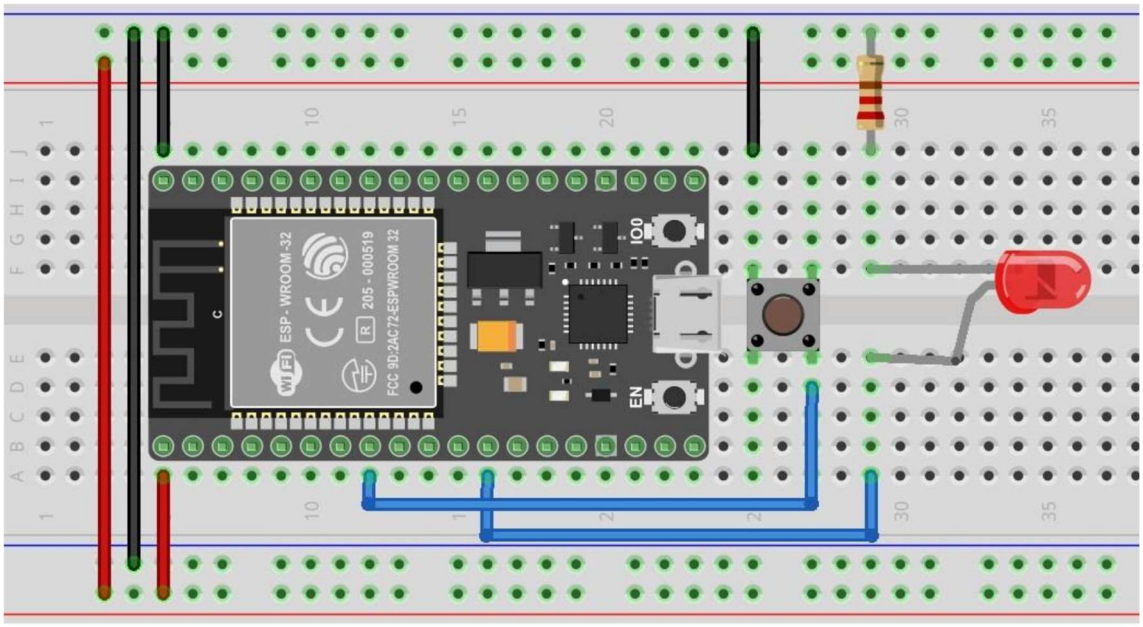


```
pinMode(16, OUTPUT)
times(10){
    digitalWrite(16, HIGH)
    delay(500)
    digitalWrite(16, LOW)
    delay(500)
}
```

## 17.2 デジタル入出力

スイッチを押すとLEDが点灯する。

回路図例

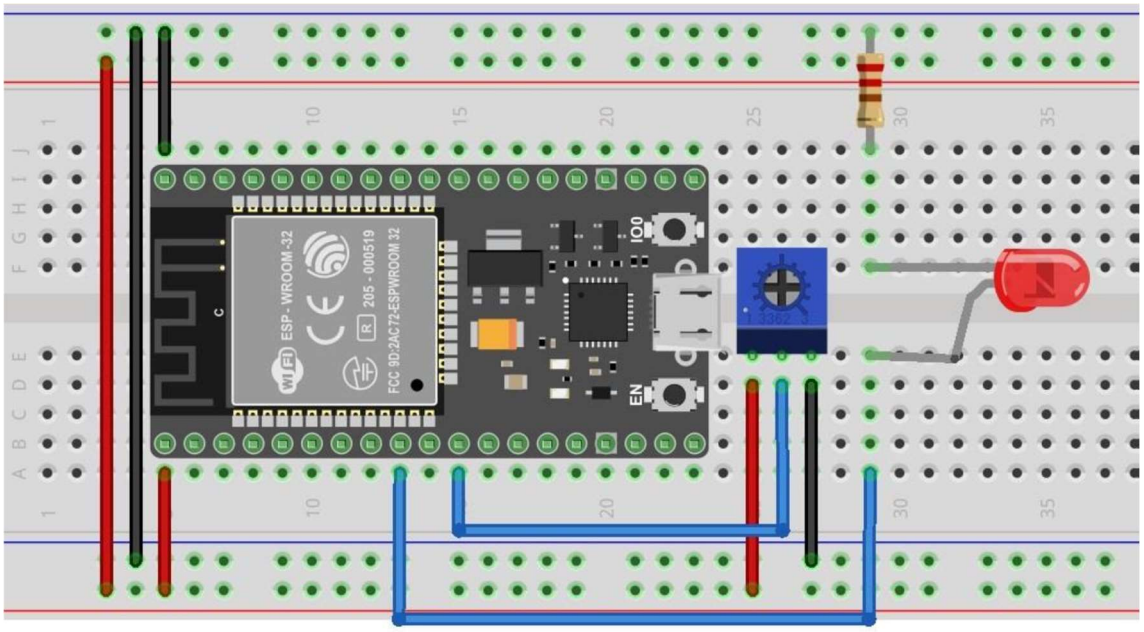



```
pinMode(21,OUTPUT)
pinMode(22,INPUT_PULLUP)
while(1){
        digitalWrite(21,!digitalRead(22))
}
```

## 17.3 アナログ入出力

可変抵抗を回すとLEDの明るさが変わる

回路図例

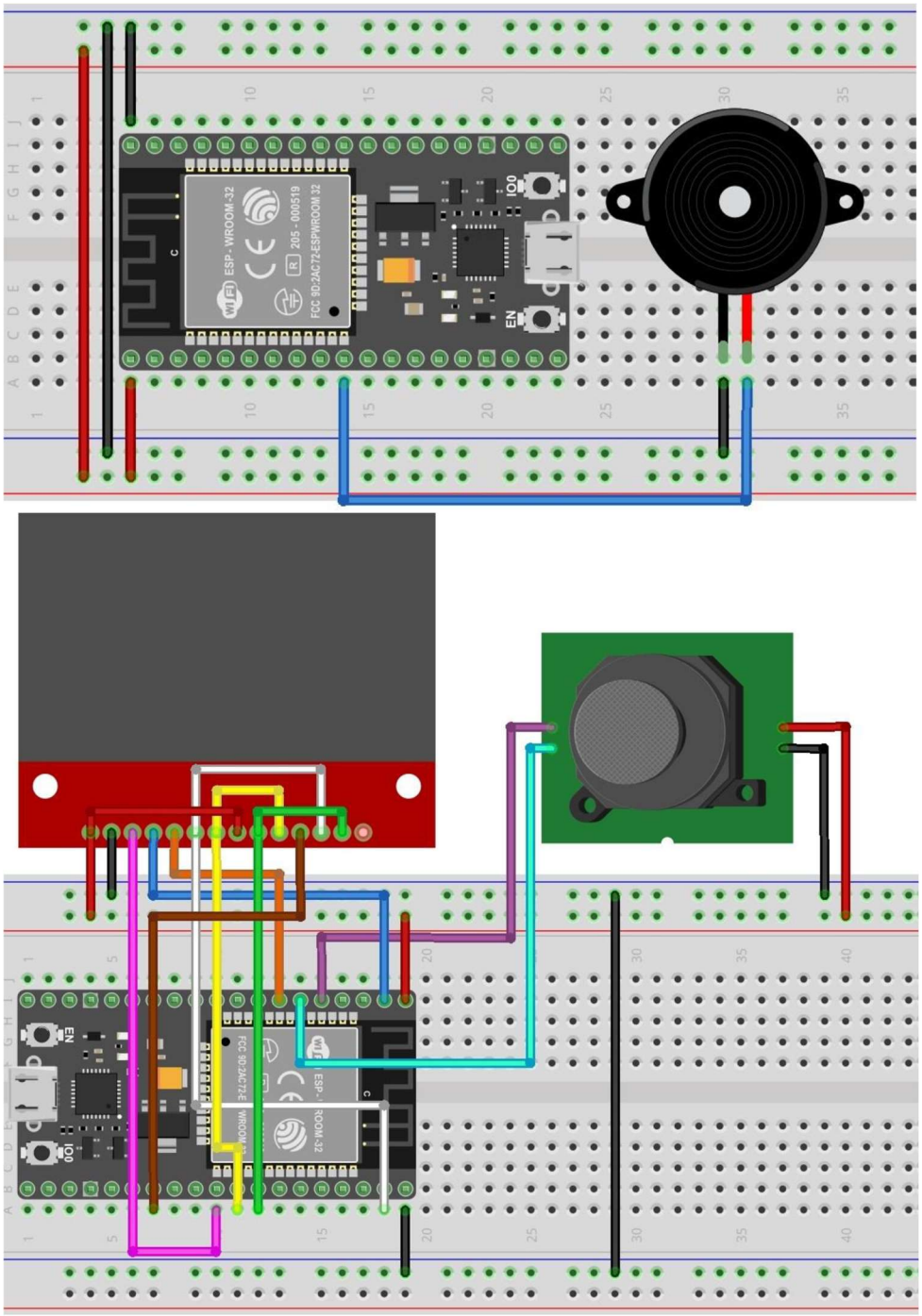



```
pinMode(25,OUTPUT)
ledcSetup(1,12800,12)
ledcAttachPin(25,1)
pinMode(27,INPUT)
while(1){
        ledcWrite(1,analogRead(27))
}
```

## 17.4 ジョイスティック操作

ジョイスティックで黒点操作

回路図例



```
var bx=120
var by=160
pinMode(34,INPUT)
pinMode(35,INPUT)
var x = analogRead(34)
var y = analogRead(35)
var prex,prey
while(1){
    x = analogRead(34)
    y = analogRead(35)
    x = adjustX(x)
    y = adjustY(y)
    if(bx<=240 && bx >= 0){
        prex=bx
        bx = bx + x
    }else if(bx>=240){
        prex=bx
        bx-=2
    }else{
        prex=bx
        bx+=2
    }
    if(by<=320 && by>=0){
        prey=by
        by = by + y
    }else if(by>=320){
        prey=by
        by-=2
    }else{
        prey=by
        by+=2
    }
    setColor(WHITE){
        fillCircle(prex,prey,10)
        setColor(BLACK)
        fillCircle(bx,by,10)
    }
```

y の値を読み取る関数

```
function adjustY(y){
   y=y-1859
   if(y/100>=1){
      return 5
   }else if(-1 >= y/100){
      return -5
   }
   return 0
}
```

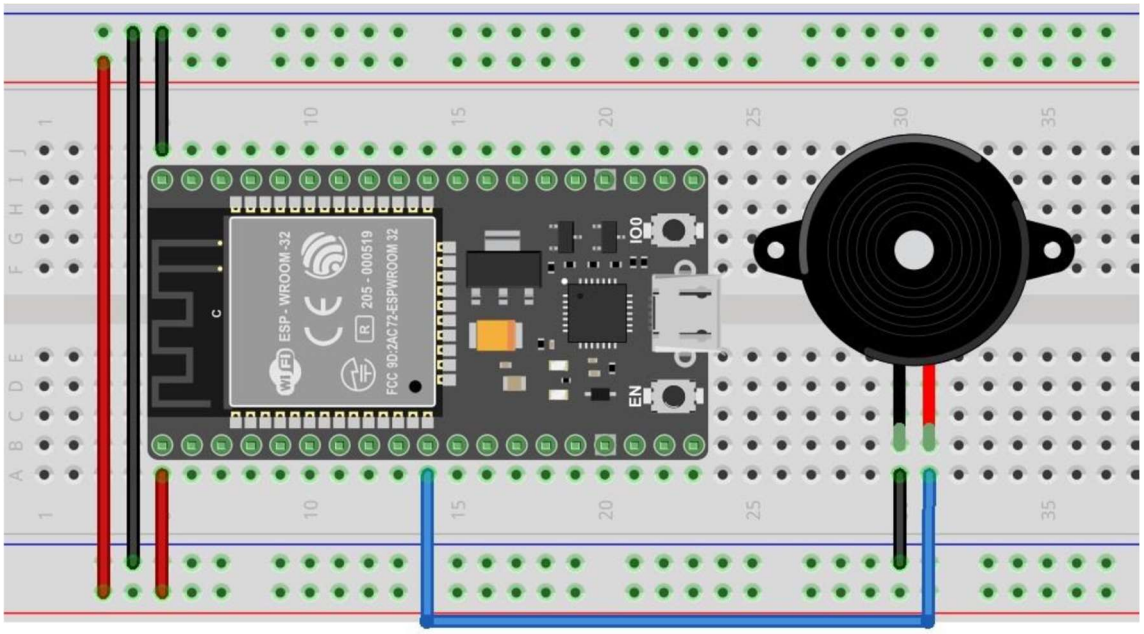
x の値を読み取る関数

```
function adjustX(x){
   x=x-1892
   if(x/100>=1){
      return 5
   }else if(-1>= x/100){
      return -5
   }
   return 0
}
```

## 17.5 スピーカー

0.5秒間隔で，ドレミファソラシドと鳴らす

回路図例




```
pinMode(26,OUTPUT)
ledcSetup(0, 12000, 12)
ledcAttachPin(26, 0)
var i = 0
var
tone=[262,294,330,349,392,440,494,523,0]
times(9){
    ledcWriteTone(0, tone[i++])
    delay(500)
}
```

# 18.デモプログラム（令和 5 年度作成分）

## 18.1 デモプログラム 1

1 番ピンに LED、 2 番ピンにプッシュスイッチを接続。

実行すると LED が 1 秒周期で点滅し始め、2 番ピンに接続したプッシュスイッチを押すと、

LED を消灯、点滅を停止させるプログラムです。

```
var led = Output(1)
var button = Button(2)

function blink(){
led.toggle()
}

function button_stop(){
        timer.stop()
        led.low()
        println("timer stop")
}

var timer = Timer()
timer.setTimer("blink", 500)

button.onPress("button_stop")
```

```
// 1 番ピンを出力に設定
// 2 番ピンをボタンに設定

//実行すると LED の出力を反転




//実行すると
//タイマーの停止
//LED の消灯
//タイマーが停止したことを表示



//タイマーを設定
// 5 0 0 ミリ秒で反転

//スイッチを押すと button_stop 関数を
実行させる
```

## 18.2 デモプログラム 2

1 番ピンにプッシュスイッチを接続。
スイッチを押した回数をカウントし、
毎度押すたびに押した回数を表示させる。

```
var button = Button(1)
var count = 0

function button_press(){
        count++
        println(count)
}

button.onPress("button_press")
```

```
// 1 番ピンをボタンに設定
//count の初期値は 0



//押した回数をカウント
//これまで押した回数を表示



//スイッチを押すと button_press()を実行
```

## 18.3 デモプログラム 3

1 番ピンに LED、2 番ピンにプッシュスイッチを接続。
スイッチを押す毎に，LED の点灯状態を反転させる。

```
var led = Output(1)
var button = Button(2)

function button_press(){
        led.toggle()
}

button.onPress("button_press")
```

```
// 1 番ピンを出力に設定
// 2 番ピンをボタンに設定

//LED の点灯状態を反転



//スイッチを押すと button_press()を実
```

## 18.4 デモプログラム 4

1 番ピンに LED を接続、 2 番、 3 番、 4 番ピンにプッシュスイッチを接続。
2 番ピンのスイッチを押すと点灯
3 番ピンのスイッチを押すと消灯
4 番ピンのスイッチを押すと点灯と消灯を反転させる。

```
var led = Output(1)

var button1 = Button(2)
var button2 = Button(3)
var button3 = Button(4)

function button1_press(){
        led.high()
}
function button2_press(){
        led.low()
}
function button3_press(){
        led.toggle()
}
```

```
// 1 番ピンを出力に設定

// 2 番ピンをボタン 1 に設定
// 3 番ピンをボタン 2 に設定
// 4 番ピンをボタン 3 に設定//LED を点灯

//LED を消灯


//LED を消灯


//LED の点灯消灯を反転
```

## 18.5 デモプログラム 5 〜サーバー側〜

```
var port = 50000                                    //ポート番号を 50000 に設定
WiFi.begin(“Access point”, "Password")              //アクセスポイントに接続

println(WiFi.status())                              //WiFI の接続状態を表示
print("IP Address = ")                              //ローカル IP アドレスを表示
println(WiFi.localIP())

var udp = WiFi.UDP()                                //UDP を取得
udp.begin(port)

var led = Output(1)                                 // 1 番ピンを出力に設定
function on_receive(){
        var text = udp.read()                       //受信したメッセージを読む
        if (text == "LED ON"){                      // LED ON を受信すると点灯
                led.high()
        } else if (text == "LED OFF"){              // LED OFF を受信すると消灯
                led.low()
        } else if (text == "LED TOGGLE"){           // LED TOGGLE を受信すると反転
                led.toggle()
        }
}
udp.onReceive("on_receive")                         //受信したら on_receive()を実行
```

## 18.6 デモプログラム 5〜クライアント側〜

```
var button1 = Button(1)                          //1 番ピンをボタン 1 に設定
var button2 = Button(2)                          //2 番ピンをボタン 2 に設定
var button3 = Button(3)                          //3 番ピンをボタン 3 に設定

var ipaddress = "xxx.xx.xx.xxx"                  // サーバー側で表示される
                                                 ローカル IP アドレスを入力
var port = 50000                                 //ポート番号を 50000 に設定

WiFi.begin(“Access point”, "Password")           //アクセスポイントに接続
println(WiFi.status())                           //WiFI の接続状態を表示
print("IP Address = ")                           //ローカル IP アドレスを表示
println(WiFi.localIP())

var udp = WiFi.UDP()                             //UDP を取得
udp.begin(port)
function button1_press(){                        //LED ON を送信
        udp.send(ipaddress, port, "LED ON")
}
function button2_press(){                        //LED OFF を送信
        udp.send(ipaddress, port, "LED OFF")
}
function button3_press(){
        udp.send(ipaddress, port, "LED TOGGLE")  //LED TOGGLE を送信
}

button1.onPress("button1_press")                 //ボタン 1 (1 番ピン)を押す
                                                 と button1_press()を実行
button2.onPress("button2_press")                 //ボタン 2(2 番ピン)を押す
                                                 と button2_press()を実行
button3.onPress("button3_press")                 //ボタン 3(3 番ピン)を押す
                                                 と button3_press()を実行
```

## 18.7 デモプログラム 6

1 番ピン 2 番ピン 3 番ピンにモーターを接続。
4 番、5 番、6 番ピンにプッシュスイッチを接続。
押したスイッチによってモーターを動作させる。

```
var motor_On_Off = Output(1)                    //1 番ピンを出力に設定
var motor_l = Output(2)                         //2 番ピンを出力に設定
var motor_r = Output(3)                         //3 番ピンを出力に設定
var button1 = Button(4)                         //4 番ピンをボタン 1 に設定
var button2 = Button(5)                         //5 番ピンをボタン 2 に設定
var button3 = Button(6)                         //6 番ピンをボタン 3 に設定
var motor_flag = 0                              //モーターの状態 0 で停止
function button1_press(){
        motor_r.high()                          //モーターを回転
        motor_l.low()
}
function button2_press(){
        motor_l.high()                          //モーターを逆回転
        motor_r.low()
}
function button3_press(){
        motor_On_Off.toggle()                   //モーターの状態を制御
        if(motor_flag == 0){
                motor_flag = 1
                println("motor ON")
        }else{
                motor_flag = 0
                println("motor OFF")
        }
        motor_r.low()
        motor_l.low()                           //ボタン 1 (4 番ピン)を押すと
}                                               button1_press()を実行
button1.onPress("button1_press")                //ボタン 2(5 番ピン)を押すと
button2.onPress("button2_press")                button2_press()を実行
button3.onPress("button3_press")                //ボタン 3(6 番ピン)を押すと
                                                button3_press()を実行
```

## 18.8 デモプログラム 7

1 番ピンにプッシュスイッチを接続。
LCD を接続し
プッシュスイッチを押すごとに LCD に表示させる色を変える。
色は　水色　紫　黄色　青　緑　の順に変わり
緑になるとカウントをリセットし次は水色に変わる。

```
var button = Button(1)
var count = 0
var colors = [0x07ff, 0x901a, 0xfd20,
0x001f, 0x07e0]

function button_press(){
        if(count == 5){
                count = 0
        }
        clearScreen()
        setColor(colors[count])
        fillScreen()
        count++
}

button.onPress("button_press")
```

```
//1 番ピンをボタン 1 に設定
//カウントの初期値を 0 に設定




//ボタンを押した回数が 5 になったらカウントをリセット

//画面を白くする
//色をセットする
//セットされた色を画面に出力する
//スイッチが押されたらカウントを 1 増やす

//ボタンを押すと button_press()を実行
```

## 18.9 デモプログラム 8

温湿度センサーを使って温度と湿度を測定する。
※AM2320 のみ

```
var sensor = AM2320()
sensor.temperature()
sensor.humidity()
```

```
//温湿度センサーを設定
//温度を表示
//湿度を表示
```

## 19.Begin license text.



川内職業能力開発短期大学校MicomScript開発チーム
＜令和４年度開発メンバー＞
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
＜令和５年度開発メンバー＞
中武 愛梨　鹿島 上幹　田上 叶　寺崎 敦　永井 光　福岡 秀晃
久美田 玲衣　原田 拓武　相川 政和