□□

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

□□

目次

１．使い方

1.1 ドライバのインストール

https://jp.silabs.com/developers/usb-to-uart-bridge-vcp-drivers?tab=downloads
・上の URL から"CP210x_Universal_Windows_Driver"の対応する OS のドライバをダウンロードする。

・ダウンロードした ZIP ファイルを解凍する。
デバイスマネージャーより、ポート → Silicon Labs CP210x USB to UART Bridge(COMx)を
右クリックし、ドライバの更新を選択する。
コンピュータを参照 → ドライバを検索 から、解凍したフォルダを選択する。
最後に、再起動する。

１.2 Arduino IDE のインストール

https://www.arduino.cc/en/software
・下の URL から、適当なインストーラをダウンロードする。このとき、"JUST DOWNLOAD"を選択するが、自動翻訳を使うと文字が読めなくなるので、自動翻訳を解除する。

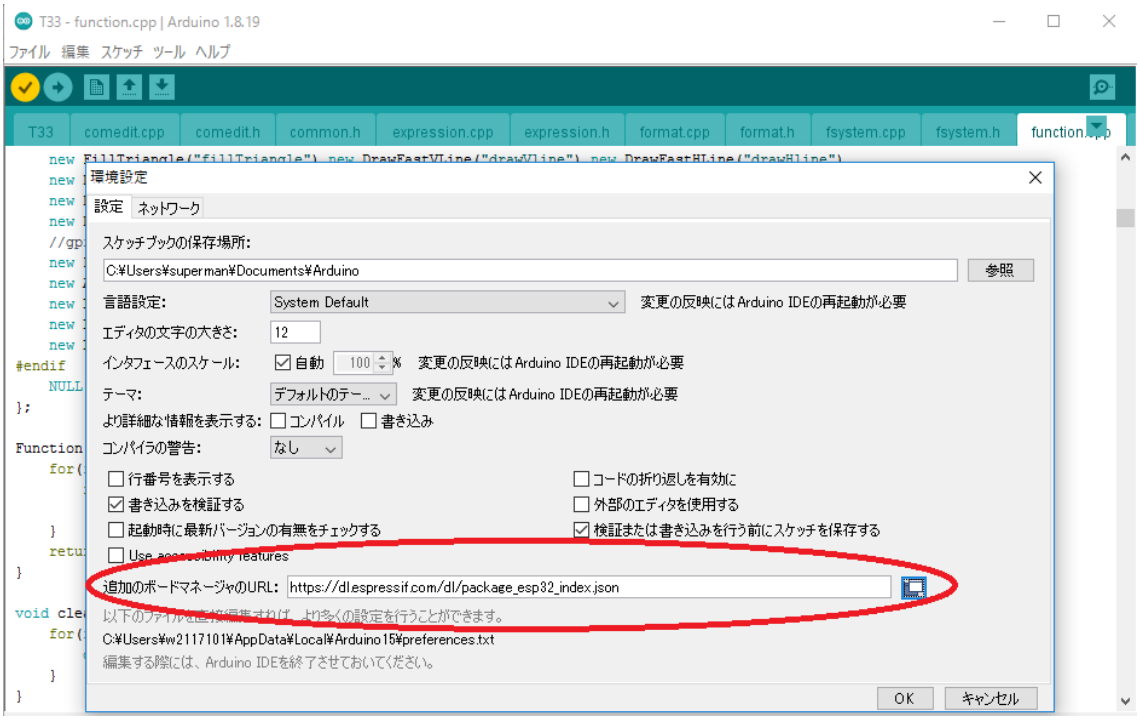
1.3 Arduino IDE の設定

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

・上の URL から、GitHub にある Arduino core for the ESP32 のページを開く。

・"README" → "Documentation" → "Installing(Windows, Linax and macOS)"を開き、Stable release link: の下の URL をコピーする。

Arduino の"ファイル"から、"基本設定(環境設定)"を開き、
"追加のボードマネージャーの URL"の右端のボタンをクリックし、
"追加の URL を一行ずつ入力"の下にコピーした URL を貼り、OK を選択する。



・メニューバーの　ツール → ボード → ボードマネージャー を開き、
検索欄に ESP32 と検索する。

ESP32 by Espressif System をインストールする。

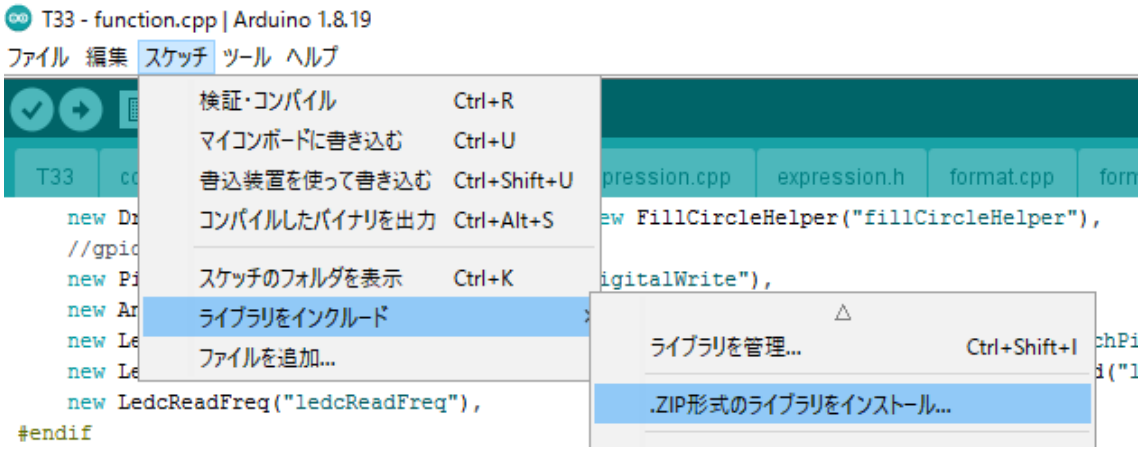
・Arduino IDE の"ファイル" → "開く"から、ダウンロードした MicomScript の src ファイルを選択する。

]

・ツール から、画像のように設定する。ボード、Partition Scheme、シリアルポートはデバイスに合わせて選択する。

・Arduino IDE の スケッチ → ライブラリをインクルード →zip 形式のライブラリをインストール

を選択し、MicomScript の lib から TFT_eSPI-master.zip を選択する。(このファイルは解凍しない。)



1.4 Arduino IDE でマイコンボードに書き込む

画像の〇部分をクリックすることでコンパイルされ、マイコンボードに書き込まれる。

1.5 TeraTerm の設定

・"シリアル"を選択する。
・設定 → シリアルポート を選択し、スピード:115200 に設定する。
・設定 → 端末 より、受信:LF に設定する。
・編集 → 画面クリア をクリックし、画面をクリアする。

1.6 TeraTerm でプログラム作業をする。

1.7 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、入力促進記号をあらわします。

●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。
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。
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。
変数名は半角のみ使用可能。
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出来ません。
(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。)

●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 型のことを指す。
数字…0～9 までの値を含む文字列のことを指す。
返される値が、先頭の値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、演算処理がされる。
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する。
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る。
その際、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る。

3.1 加算 （＋）

数値が演算子の左側で、文字列(string)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)として演算されます。
文字列が演算子の左側の時、文字列結合されます。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)が結合されます。

```
$ 1 + 2.1 + "string"
3
↑1 + 2 + 0 として演算。

$ 1.5 + 1 + "string"
2.5
↑1.5 + 1 + 0 として演算。

$ "string" + 1 + 2.1
string12.1
↑"string" + "1" + "2.1" として演算。

$ "Micom" + "Script"
MicomScript
```

$ 1.1 + "1.1apple"
2.2
↑文字列の先頭が数字である場合、その値の数字の部分の数値として扱われ、1.1 + 1.1 として演算。

$ "apple" + [1,2,3]
"appleList"

3.2 減算 （ - ）

数値が演算子の左側で、文字列(string)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)として演算されます。
string 型の値が先頭であるときの演算は不可。

$ 2 - 1.1
1

$ 1.5 - 1
0.5

$ 2.3 - "0.3"
2

$ 12 - "11apple"
1

$ 12 - "apple"
12

3.3 乗算 （*）

string 型が先頭の場合、後ろのデータ型は integer 型に統一され、文字列の重複回数になります。
後ろのデータ型が string 型の場合、文字の直前までの数字が読み取られます。
"12aaaaaaa"のようなときは 12(integer)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)となります。

```
$ 1 * 1.5
1

$ 1.5 * 1
1.5

$ 1.1 * "2.0"
2.2

$ 1 * "string"
0

$ "string" * 2.3
stringstring
↑後ろのデータ型は integer 型に統一され、文字列の重複回数になります。

$ "hello" * "world"
hello
↑"world" は 1 として演算。
```

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 型のデータ型の値が先頭であるとき、演算される string 型の値は、戦闘が数字の場合、その数字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不可。
後ろが string 型の場合、文字列の先頭が数字の場合文字の直前まで数値として読み取られます。

```
$ 4.4 / 2
2.2

$ 2.0 / 3
0.666667

$ 2.2 / "2.2"
1
```
↑string 型の値を数値として扱える場合のみ演算可能。

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。
integer 型または float 型のデータ型の値が先頭であるとき、演算される string 型の値は、数字のみの場合、その数字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不可。
後ろが string 型の場合文字列の先頭が数字の場合文字の直前まで数値として読み取られます。
list 型の値が先頭であるとき、先頭より後ろの値に数字以外が含まれる string 型の値が含まれているとエラーとなります。
list 型どうしの剰余演算は不可。

```
$ 6 % 2.5
0

$ 6.0 % 2.5
1

$ "18 % "8"
2

$ 18.0 % "8.5"
1
```

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 型のデータ型の値が先頭であるとき、演算される string 型の値は、数字のみの場合、その数字が数値として扱われ、それ以外の string 型の値は数値の 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不可。
後ろが string 型の場合文字列の先頭が数字の場合文字の直前まで数値として読み取られます。

$ 2 ** 2.5
4

$ 2.0 ** 2.5
5.65685

$ 2 ** "2.0"
4

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。

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。
" ++a "のように記述した場合、a の値をインクリメントしたあとの値を返します。
逆に" a++ "のように記述した場合、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、デクリメントは不可。

```
$ 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。

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。

4.1 if(引数){}

条件分岐を実行する関数。
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。
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。
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 の処理を終了する。
③.{ }内の処理を実行後、引数 3 の処理を一度実行し、②に返る。

引数 1 では var 変数名 ... と記述可能。
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する。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。

```
$ 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)を記述するものとする。
数字のみの string 型の値は、数値として扱われる。

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。
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。
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。
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。
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。
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

$ 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

$ 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。

$ 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。

```
$ pow(2,3)
8
```

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。

```
$ 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。

```
$ 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。

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

7.11 平方根 sqrt()

指定した値の平方根を返す関数。

```
$ 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。

$ 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。

$ 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。
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。

８．時間関数

8.1 time()

起動してからの秒数を返す関数。

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。

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。

8.4 delay()

プログラムを指定した時間だけ止める関数。
単位はミリ秒(ms)。

９．画面表示関数

9.1 setColor()

setColor(引数 1,引数 2)
文字色、背景色を変更する関数。図形は文字色で描画される。
引数 1 に文字色、引数 2 に文字の背景色を指定。初期値が文字色は黒、背景色は白。変更するまでその色で固定される。背景色は指定しなくても良い。
色は 65k カラーで指定する。また以下の色は文字で指定可能。

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

setColor(RED)
setColor(RED,BLUE)

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。初期値は 1。

textSize(10)

9.3 clearScreen()

画面全体を白く塗りつぶす関数。

drawRect(10,10,10,10)
clearScreen()

9.4 fillScreen()

画面全体を塗りつぶす関数。

setColor(RED)
fillScreen()

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。

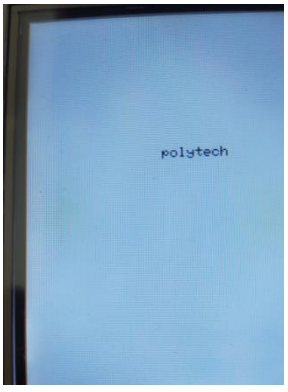
drawLine(10,10,200,200)

9.6 drawPixel()

drawPixel(x,y)
座標(x,y)に点を描画する関数。

drawPixel(100,100)

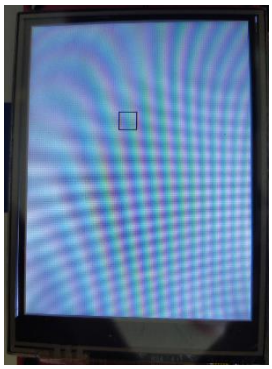
9.7 drawString()



drawString(str,x,y)
座標(x,y)に文字列 str を描画する関数。文字の背景は背景色で塗りつぶされる。

drawString("polytech",100,100)

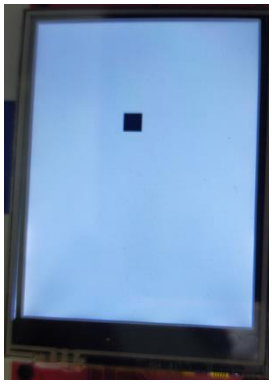
9.8 drawRect()



drawRect(x,y,w,h)
四角形の左上座標(x,y)、幅 w、高さ w を指定し四角形を描画する関数。

drawRect(100,100,20,20)

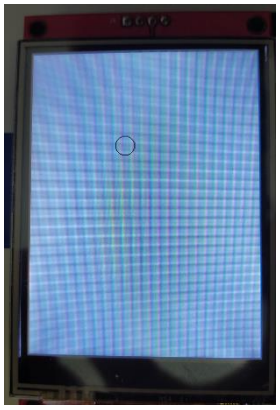
9.9 fillRect()



fillRect(x,y,w,h)
四角形の左上座標(x,y)、幅 w、高さ w を指定し塗つぶされた四角形を描画する関数。

fillRect(100,100,20,20)

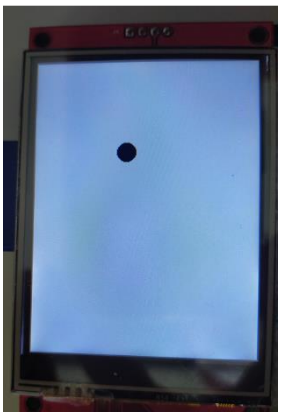
9.10 drawCircle()



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。

drawCircle(100,100,10)

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。

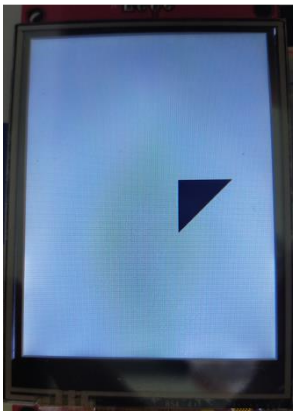
fillCircle(100,100,10)

9.12 drawTriangle()

drawTriangle(x1,y1,x2,y2,x3,y3)
座標(x1,y)、座標(x2,y2)、座標(x3,y3)を指定し、角形を描画する関数。

drawTriangle(150,150,200,150,150,200)

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。

fillTriangle(150,150,200,150,150,200)

9.14 drawVline()

drawVline(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。

drawVline(10,10,10)

9.15 drawHline()

drawHline(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。

drawHline(10,10,20)

9.16 drawEllipse()

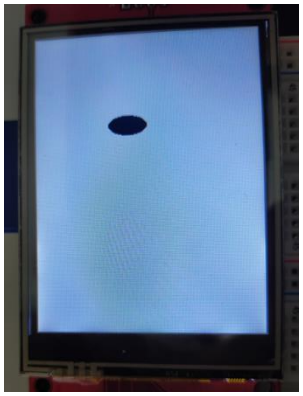
drawEllipse(x,y,rx,ry)
中心座標(x,y)、x 軸の半径 rx,y 軸の半径 ry を指定し、楕円の線を描画する関数。

drawEllipse(100,100,20,10)

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。

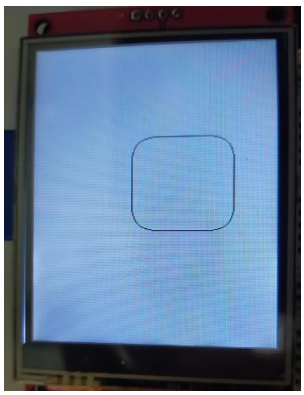
fillEllipse(100,100,20,10)



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、矩形の線を引く関数。(角の丸い四角を描画)

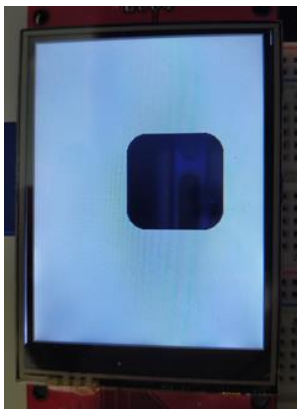
drawRoundRect(100,100,10,10,5)



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。

fillRoundRect(100,100,10,10,5)



9.20 drawCircleHelper()

drawCircleHelper(x,y,r,c)
座標(x,y)、円の半径 r、描画したい円弧のエリアビット c(1:左上, 2:右上, 4:右下, 8:左下)を指定し、円弧を描画する関数。

drawCircleHelper(100,100,10,1)

9.21 fillCircleHelper()

fillCircleHelper(x,y,r,c,d)
座標(x,y)、円の半径 r、描画したい円弧のエリアビット c(1:左半分, 2:右半分)、横幅 d を指定し、描画する関数。

fillCircleHelper(100,100,10,1,10)

１０．GPIO 関数

pin　　→ ピン番号
io　　→ INPUT / OUTPUT / INPUT_PULLUP
HL　　→ HIGH / LOW
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する。
ピンの番号とピンの動作を入力か出力に設定する関数。

```
pinMode(25,OUTPUT)
```

10.2 digitalWrite()

digitalWrite(pin , HL)
上のように引数を記述する。
HIGH または LOW を、指定したピンに出力する関数。

```
digitalWrite(25,HIGH)
```

10.3 analogRead()

analogRead(pin)
上のように引数を記述する。
指定したアナログピンから値を読み取る関数。

```
var a = analogRead(25)
```

10.4 digitalRead()

digitalRead(pin)
上のように引数を記述する。
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。

var a = digitalRead(25)

10.5 ledcSetup()

ledcSetup(ch , freq , bit)
上のように引数を記述する。
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。
チャンネルは 0〜15 まで指定できる。

ledcSetup(1,12000,16)

10.6 ledcAttachPin()

ledcAttachPin(pin , ch)
上のように引数を記述する。
LED PWM で利用するピンとチャネルを結びつける関数。

ledcAttachPin(25,1)

10.7 ledcDetachPin()

ledcDetachPin(pin)
上のように引数を記述する。
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。

ledcDetachPin(25)

10.8 ledcWrite()

ledcWrite(ch , tones)
上のように引数を記述する。
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。

ledcWrite(1,500)

10.9 ledcWriteTone()

ledcWrite(ch , freq)
上のように引数を記述する。
LED PWM を利用して、指定した周波数を PWM 出力する関数。

ledcWriteTone(1,6000)

10.10 ledcRead()

ledcRead(ch)
上のように引数を記述する。
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。

var a = ledcRead(ch)

10.11 ledcReadFreq()

ledcReadFreq(ch)
上のように引数を記述する。
LED PWM のチャネルに指定した周波数を取得する関数。

var a = ledcReadFreq

１１．その他のライブラリ関数

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

11.1 abs()

　引数の絶対値を integer 型の値として返す関数。
float 型の値を引数として指定した場合、値の小数点以下を切り捨て、integer 型の絶対値を返します。
string 型の値を引数として指定した場合、値は integer 型の 0 を返す。
また、string 型の値の先頭が数字の場合、文字の直前まで数値として読み取られます。
list 型の値を引数として指定した場合、配列内の値を合計した値を integer 型の絶対値として返します。
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。

```
$ abs(-2)
2

$ abs(2.1)
2

$ abs("hello")
0

$ var a = [1,2,3]
$ abs(a)
6
```
↑配列 a の要素を integer 型として扱い、その合計値が返される。

```
$ var b = [3,"hello"]
$ abs(b)
3
```
↑配列内の string 型の値は integer 型の 0 として扱われる。

$ var e = [1,2,3,[1,2,3]]
$ abs(e)
12
↑配列内のさらに配列の値も同様に、配列内の要素を integer 型として扱ったときの要素の合計値が返される。

11.2 bool()

引数に指定した値の真理値を返す関数。

$ bool(3)
true

$ bool(0)
false

$ bool(3 < 4)
true

$ bool(3 > 4)
false

11.3 fabs()

　引数の絶対値を float 型の値として返す関数。
string 型の値を引数として指定した場合、値は float 型の 0 として値を返します。
list 型の値を引数として指定した場合、配列内の値を合計した値を絶対値にして返します。
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。

```
$ fabs(-2.2)
2.2

$ fabs("hello")
0
```

11.4 float()

引数の数値を浮動小数点数 float 型に変換した値を返す関数。
string 型を引数に指定した場合は 0 が返されます。
list 型の値を引数として指定した場合、配列内の値を合計した値を返します。
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。

```
float()
$ var a = 2
$ type(a)
"integer"
$ a = float(a)
2
$ type(a)
"float"
```

11.5 input()

ユーザーに値を入力させる関数。
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。
入力値はすべて string 型の値になります。

```
$ var a
$ a = input()
"6"

$ println(a)
6

$ type(a)
"string"
```

11.6 int()

引数の数値を integer 型に変換した値を返す関数。
string 型を引数としたとき、integer 型の 0 が返されます。
list 型の値を引数として指定した場合、配列内の値を合計した値を返します。
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。

```
$ var a = 1.1
$ type(a)
"float"
$ a = int(a)
1
$ type(a)
"integer"
```

11.7 max()

引数の中で最大の要素を返す関数。
引数に先頭が数字である string 型を指定した場合、その値は先頭の数字と同じ数値として扱われます。
引数に先頭に数字を含まない string 型を指定した場合、その値は数値 0 として扱われます。
引数に list 型を指定した場合、そ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
```

11.8 min()

引数の中で最小の要素を返す関数。
引数に先頭が数字である string 型を指定した場合、その値は先頭の数字と同じ数値として扱われます。
引数に先頭に数字を含まない string 型を指定した場合､その値は数値 0 として扱われます。
引数に list 型を指定した場合、そ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]
$ min(a)
1

11.9 print()

引数に指定した値を書き出す関数。
println()と相違点は、末尾に改行がない点。
文字列を出力する場合は、" "の内部に表示したい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る。
2 進数、16 進数を引数に指定した場合、10 進数として表示されます。

$ print("print",1)
print1$

$ print(1,2,3)
123$

$ print(0b1100)
12$
↑2 進数→10 進数として表示。

$ print(0xc)
12$
↑16 進数→10 進数として表示。

11.10 printf()

引数内に" "で囲んだ文字列をかき出す関数。
" "内で、¥n を記述した位置で改行されます。
変数を書き出したい場合、下のように、" "内の変数を書き出したい位置に変換指定子を記述し、 ，で区切り書き出したい変数を指定できます。
下のような変換指定子に対応している。

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	実数を最適な形式で出力する。
%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。

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

11.11 println()

引数に指定した値を書き出す関数。
print()との相違点は、末尾に改行がある点。
( )内では改行されません。

$ println("hello",1,2)
hello12

11.12 string()

引数の値を string 型に変換し、値を返す関数。
引数に list 型の値を指定した場合、string 型の文字列”List”が返されます。

```
$ type(string(1.1))
"string"
```

11.13 sum()

引数の合計値を返す関数。
string 型の値は 0 として扱われます。
ただし、string 型の値の先頭が数字の場合、文字の直前まで数値として読み取られます。

```
$ sum(1,2)
3

$ sum(5,"world")
5

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

11.14 type()

引数に指定した値の型名を返す関数。

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

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

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

１２．ユーザ定義関数

ユーザが作成する関数。
ユーザ定義関数名も変数名と同様に半角のみ使用可能。
全角文字は不可。
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る。
予約語を使用することはできない。また、¥ などの特殊文字の使用も不可。

12.1 ユーザ定義関数作成

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

↑のように、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。

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。

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

↑return 変数名
を記述することで、{ }内の指定した変数を、ユーザ定義関数呼び出し時に返すことができます。

sigma 関数を作成し、呼び出すと、初期値 1 から n までを合計した値が返されます。

```
$ 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。

１３．コマンド

13.1 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

13.2 help コマンド名

指定されたコマンドの説明が後述されます。

$ help undef
変数、または関数を削除します。
$ help vlist
定義した変数の変数名と値を表示します。

13.3 vlist

変数リストを表示する。変数に値が入っていない場合は、(null)と表示されます。

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

13.4 undef 変数名

変数定義を削除します。
変数名の部分に all と記述することで、宣言されている全ての変数を削除できます。

```
$ var a,b
$ undef a
$ vlist
b = (null)
```

13.5 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
false : false
FALSE : false
L : false
LOW : false
```

13.6 flist

関数リストを表示します。

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

13.7 undef 関数名()

関数定義を削除します。
同名関数を定義したら、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()
```

13.8 exit

プログラムを終了します。

13.9 reset

リセットを行います。

13.10 clist_ESP

定数リストを表示します。

１４．ファイル操作コマンド

14.1 files

作成した全ファイルの一覧を表示します。

$ files
56 /s01
263 /hanoi

14.2 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。

14.3 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”

14.4 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

14.5 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
```

14.6 show “/ファイル名”

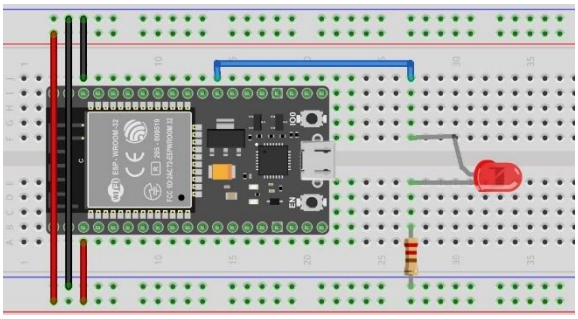
指定したファイルのソースを表示します。

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

１５．サンプルプログラム

LED 点滅

16 pin に接続された LED を 1 秒間隔で点滅させる

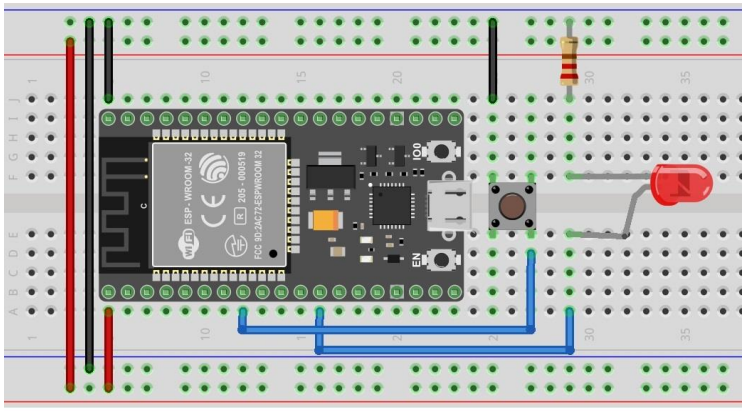


回路図

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

デジタル入出力

スイッチを押すと LED が点灯する。

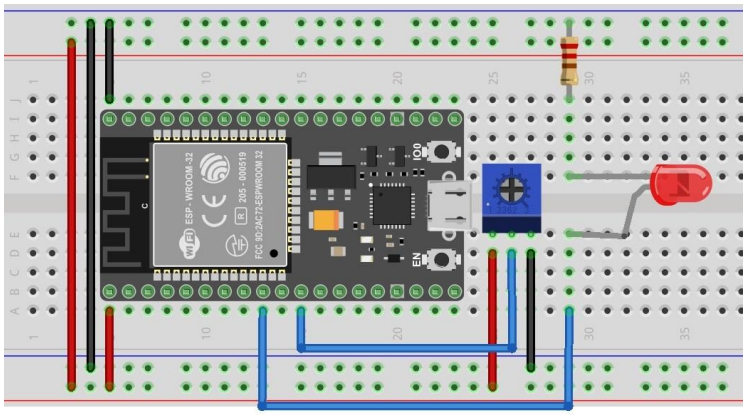


回路図

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

アナログ入出力

可変抵抗を回すと LED の明るさが変わる

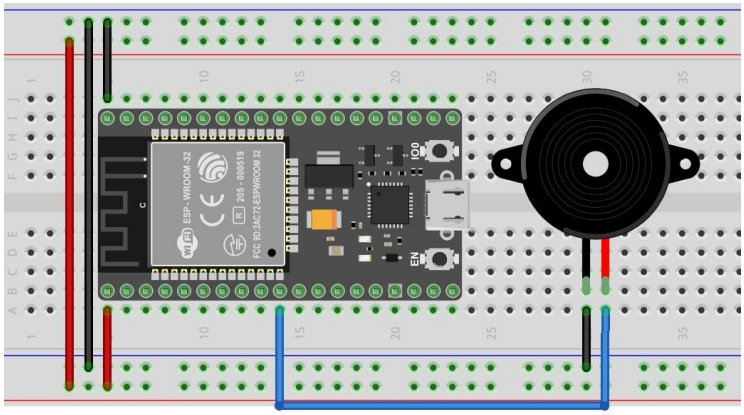


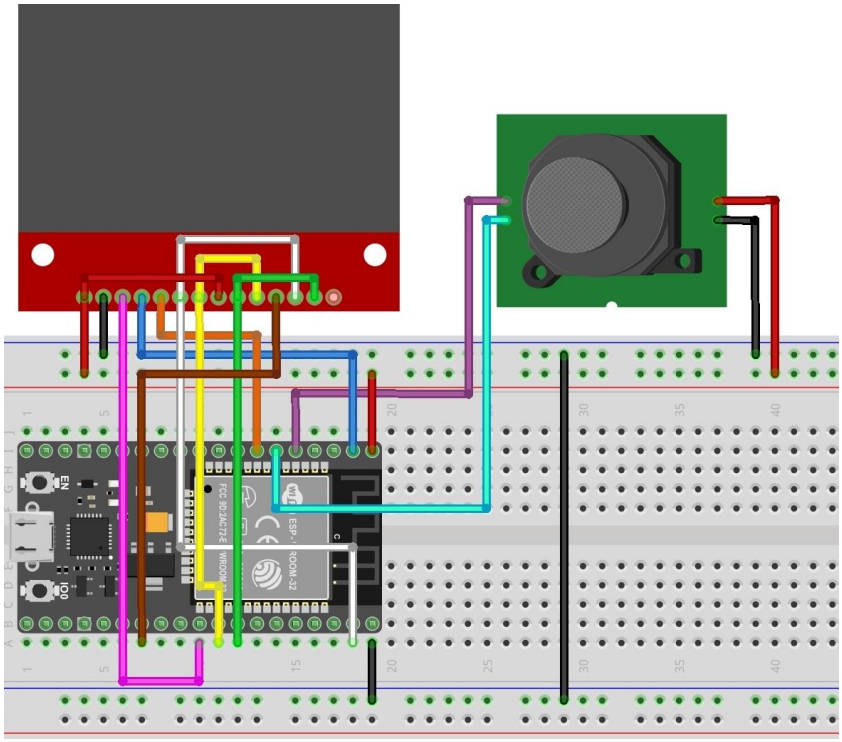
回路図例

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

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

```
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) // 0 -
4095
    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
    }//右に続く
```

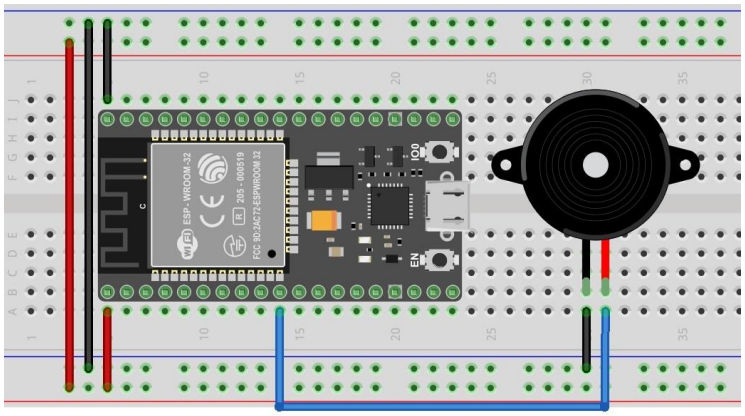
```
 x の値を読み取る関数
function adjustX(x){
    x=x-1892
    if(x/100>=1){
        return 5
    }else if(-1>= x/100){
        return -5
    }
    return 0
}
```

```
 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
}
```

```
setColor(WHITE){
    fillCircle(prex,prey,10)
    setColor(BLACK)
    fillCircle(bx,by,10)
}
```

- 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

```
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)
}
```

Begin license text.

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