

---

---

豊四季タイニーBASIC for micro:bit V0.06

リファレンスマニュアル RV01

---

---

作成日 2018年1月26日

作成者 たま吉さん



---

## 目次

---

### 豊四季タイニーBASIC for micro:bit V0.06 リファレンスマニュアル RV01

目次 .....	i
1. 仕様や構成など .....	1
1.1 はじめに .....	1
1.2 関連情報 .....	1
1.3 著作権・利用条件について .....	2
1.4 システム構成 .....	2
1.5 仕様 .....	3
1.6 ボードピン構成 .....	4
1.7 ボード上のピン一覧 .....	5
1.8 メモリーマップ .....	6
1.9 文字(フォント)コード .....	7
1.10 編集操作操作キー .....	8
1.11 キー入力コード .....	8
2. 利用環境設定 .....	11
2.1 ファームウェアの書き込み .....	11
2.2 ターミナルソフトの通信条件の設定 .....	12
2.3 接続とターミナルソフトの表示設定 .....	13
2.4 コンソール画面の表示設定 .....	13
2.5 プログラムの保存と読み込み .....	14
2.6 プログラム保存領域の初期化 .....	14
2.7 パソコンからのプログラム読み込み .....	15
2.8 フラッシュメモリのエクスポート .....	16
2.9 保存したプログラムをマイコンボードに読み込むには「2.6 プログラム保存領域の初期化」 .....	16
2.10 プログラムの自動起動方法 .....	18
2.11 RTC の時刻設定 .....	18
3. 使ってみよう！ .....	19
3.1 入力可能状態は OK■（カーソル） .....	19
3.2 Hello,world を表示してみる .....	19
3.3 Hello,world を Hello,Tiny BASIC に変更してみる .....	19
3.4 PRINT の前に 10 を追加してプログラムにしよう .....	20
3.5 プログラムを実行してみよう .....	20
3.6 プログラムに追加する .....	20
3.7 行番号を並びなおす .....	21
3.8 やっぱり 10 行は不要、削除したい .....	21

3.9	プログラムを保存したい.....	22
3.10	保存されたプログラムを読み込み.....	22
4.	プログラム構成要素の説明（コマンド・関数等）.....	23
4.1	プログラムの構造と形式.....	23
4.2	行と命令文.....	24
4.3	制御文.....	25
4.4	コマンド・関数.....	25
4.5	コマンド・関数一覧.....	26
4.6	数値.....	29
4.7	文字・文字列.....	29
4.8	文字列利用における制約.....	29
4.9	変数・配列変数.....	30
4.10	演算子.....	31
4.11	式.....	32
4.12	定数.....	32
5.	各制御文の詳細.....	35
5.1	GOTO 指定行へのジャンプ（制御命令）.....	35
5.2	GOSUB サブルーチンの呼び出し（制御命令）.....	36
5.3	RETURN GOSUB 呼び出し元への復帰（制御命令）.....	37
5.4	IF 条件判定（制御命令）.....	38
5.5	FOR TO ～ NEXT 繰り返し実行（制御命令）.....	39
5.6	END プログラムの終了（制御命令）.....	40
6.	各コマンド・関数の詳細.....	41
6.1	RUN プログラムの実行（システムコマンド）.....	41
6.2	RENUM 行番号の振り直し（システムコマンド）.....	42
6.3	DELETE プログラムの指定行の削除（システムコマンド）.....	43
6.4	WIDTH シリアルターミナルの画面サイズの設定.....	44
6.5	LIST プログラムリストの表示.....	45
6.6	NEW プログラムの消去.....	46
6.7	LOAD 内部フラッシュメモリからプログラムを読み.....	47
6.8	SAVE 内部フラッシュメモリへのプログラム保存.....	48
6.9	FILES 内部フラッシュメモリ内保存プログラムの一覧表示.....	49
6.10	REM コメント.....	51
6.11	LET 変数に値を代入.....	52
6.12	CLV 変数領域の初期化.....	53
6.13	LRUN 指定プログラム番号の実行.....	54
6.14	EXPORT 内部フラッシュメモリの内容のエクスポート.....	55
6.15	ABS 絶対値の取得（数値関数）.....	57

6.16	ASC 文字から文字コードへの変換（数値関数） .....	58
6.17	FREE プログラム領域の残りバイト数の取得（数値関数） .....	60
6.18	INKEY キー入力の読み取り（数値関数） .....	61
6.19	RND 乱数の発生（数値関数） .....	62
6.20	LEN 文字列の長さの取得（数値関数） .....	63
6.21	MAP 数値のスケール変換（数値関数） .....	64
6.22	GRADE 等級判定（数値関数） .....	65
6.23	CHR\$ 文字コードから文字への変換（文字列関数） .....	67
6.24	BIN\$ 数値から2進数文字列への変換（文字列関数） .....	68
6.25	HEX\$ 数値から16進数文字列への変換（文字列関数） .....	69
6.26	DMP\$ 整数の小数付き数値文字列への変換（文字列関数） .....	70
6.27	STR\$ 変数が参照する文字列の取得・文字列の切り出し .....	71
6.28	WAIT 時間待ち .....	72
6.29	TICK 経過時間取得（数値関数） .....	73
6.30	POKE 指定アドレスへのデータ書き込み .....	74
6.31	PEEK 指定アドレスの値参照（数値関数） .....	75
6.32	PRINT 画面への文字表示 .....	76
6.33	INPUT 数値の入力 .....	78
6.34	CLS 画面表示内容の全消去 .....	79
6.35	COLOR 文字色の設定 .....	80
6.36	ATTR 文字表示属性の設定 .....	81
6.37	LOCATE カーソルの移動 .....	82
6.38	REDRAW 画面表示の再表示 .....	83
6.39	VPEEK 画面指定位置の文字コード参照（数値関数） .....	84
6.40	MATRIX LEDマトリックス利用の有効/無効設定 .....	85
6.41	PSET LEDマトリックス 点の描画 .....	86
6.42	LINE LEDマトリックス 直線の描画 .....	87
6.43	RECT LEDマトリックス 矩形の描画 .....	88
6.44	CIRCLE LEDマトリックス 円の描画 .....	89
6.45	BITMAP LEDマトリックス ビットマップ画像の描画 .....	90
6.46	GPRINT LEDマトリックス 文字列の描画 .....	91
6.47	GSCROLL LEDマトリックス グラフィックススクロール .....	92
6.48	GPEEK LEDマトリックス上の指定位置ピクセルの参照（数値関数） .....	93
6.49	GINP LEDマトリックス 指定領域のピクセルの有無判定（数値関数） .....	94
6.50	MSG LEDマトリックス メッセージ表示 .....	95
6.51	SETFONT LEDマトリックス フォント設定 .....	96
6.52	CLP フォント設定の初期化 .....	97
6.53	TONE 単音出力 .....	98

6.54	NOTONE 単音出力停止 .....	99
6.55	SETTONE 単音出力ポートの設定 .....	100
6.56	PLAY 音楽演奏(MML 文) .....	101
6.57	TEMPO 音楽演奏のテンポの設定 .....	103
6.58	DATE 現在時刻の表示 .....	104
6.59	GETDATE 日付の取得 .....	105
6.60	GETTIME 時刻の取得 .....	106
6.61	SETDATE 時刻の設定 .....	107
6.62	GPIO GPIO 機能設定 .....	108
6.63	OUT デジタル出力 .....	109
6.64	POUT PWM パルス出力 .....	110
6.65	SHIFTOUT デジタルシフトアウト出力 .....	111
6.66	IN デジタル入力 (数値関数) .....	112
6.67	ANA アナログ入力 (数値関数) .....	113
6.68	SHIFTIN デジタルシフト入力 (数値関数) .....	114
6.69	PULSEIN 入力パルス幅の計測力 (数値関数) .....	115
6.70	I2CR I2C スレーブデバイスからのデータ受信 (数値関数) .....	116
6.71	I2CW I2C スレーブデバイスへのデータ送信 (数値関数) .....	118
6.72	NPBEGIN NeoPixel の利用開始 .....	121
6.73	NPEND NeoPixel の利用終了 .....	122
6.74	NPCLS NeoPixel 表示クリア .....	123
6.75	NPRGB NeoPixel 指定ピクセル RGB 表示指定 .....	124
6.76	NPPSET NeoPixel 指定ピクセルの色コード指定 .....	125
6.77	NPPUT NeoPixel ピクセルデータの転送 .....	126
6.78	NPSHOW NeoPixel 設定を表示に反映 .....	128
6.79	NPSHIFT NeoPixel 表示を1つシフト .....	129
6.80	NPLEVEL NeoPixel 輝度補正值設定 .....	130
6.81	RGB 15ビットカラーコードの取得 .....	131
6.82	RGB8 8ビットカラーコードの取得 .....	132
7.	応用プログラム .....	133
7.1	フォントエディタ .....	133

## 1. 仕様や構成など

### 1.1 はじめに

「豊四季タイニーBASIC for micro:bit」は整数型 BASIC インタープリタ実行環境です。  
Tetsuya Suzuki 氏が開発・公開している「TOYOSHIMI Tiny BASIC for Arduino」（以降オリジナル版と呼称）をベースに、micro:bit 向けに移植し、機能拡張を行いました。「豊四季タイニーBASIC for micro:bit」版のプログラムソースはオリジナル版と同様に公開しており、自由に改造等を行うことができます。

オリジナル版からの機能拡張としては、プログラム編集機能の強化（スクリーンエディット機能、プログラム保存機能、電子工作のための制御機能（デジタル入出力、PWM 出力、アナログ入力、I2C バス通信、シリアル通信）を追加しました。

本リファレンスマニュアルの構成は、前半に概要、システム構成、仕様、レガシーBASIC 環境を始めて利用する方向けの解説、中盤にリファレンス、後半に応用の内容となります。オリジナル版のタイニーBASIC の文法を含む、新しく追加した機能、コマンドについて解説・説明します。

### 1.2 関連情報

「豊四季タイニーBASIC（オリジナル版）」に関する情報

#### □ 開発者公開ホームページ

電脳伝説 Vintagechips 豊四季タイニーBASIC 確定版

<https://vintagechips.wordpress.com/2015/12/06/豊四季タイニーbasic 確定版/>

#### □ 開発者公開リソース

[https://github.com/vintagechips/ttbasic\\_arduino/](https://github.com/vintagechips/ttbasic_arduino/)

#### □ 開発者執筆書籍

タイニーBASIC を C で書く

<http://www.socym.co.jp/book/1020>



原本のご紹介

タイニー BASIC を C で書く  
パソコンでもマイコンでも走り、機能拡張し放題。  
著者一鈴木哲哉  
定価—3,200円+税  
判型/ページ数—B5変形/288ページ  
ISBN978-4-8026-1020-9

「豊四季タイニーBASIC for micro:bit」に関する情報

#### □ 公開サイト

豊四季タイニーBASIC for micro:bit (Tamakichi/ttbasic\_microbit)

[https://github.com/Tamakichi/ttbasic\\_microbit](https://github.com/Tamakichi/ttbasic_microbit)

#### □ micro:bit に関する情報

- micro:bit 公式 HP (日本語)

<http://microbit.org/ja/>

### 1.3 著作権・利用条件について

「豊四季タイニーBASIC formicro:bit」の著作権は移植・機能拡張者の「たま吉」および、オリジナル版開発者の「Tetsuya Suzuki 氏」にあります。本著作物の利用条件は、オリジナル版「豊四季タイニーBASIC」に従うものとします。

オリジナル版は GPL (General Public License) でライセンスされた著作物であり、「豊四季タイニーBASIC for micro:bit」もそれに GPL ライセンスされた著作物とします。本著作物もこれに従うものとします。

また、下記の利用条件も守って下さい。

- 1) 2次再配布においてはバイナリーファイルのみの配布を禁止、必ずプロジェクト式で配布すること。
- 2) 上記配布において「豊四季タイニーBASIC for micro:bit」の明記、配布元の URL を明記する。
- 3) 営利目的の NAVER 等のまとめサイト(まとめ者に利益分配のあるもの)へのいかなる引用の禁止

#### 補足事項

オリジナル版において開発者 Tetsuya Suzuki 氏は次の利用条件を提示しています。

- 1) TOYOSHIKI Tiny BASIC for Arduino

[https://github.com/vintagechips/ttbasic\\_arduino](https://github.com/vintagechips/ttbasic_arduino)

の記載内容/

*(C)2012 Tetsuya Suzuki*

*GNU General Public License*

- 2) 豊四季タイニーBASIC のリファレンスを公開

<https://vintagechips.wordpress.com/2012/06/14/豊四季タイニーbasic リファレンス公開/>

の記載内容/

*豊四季タイニーBASIC がもたらすいかなる結果にも責任を負いません。*

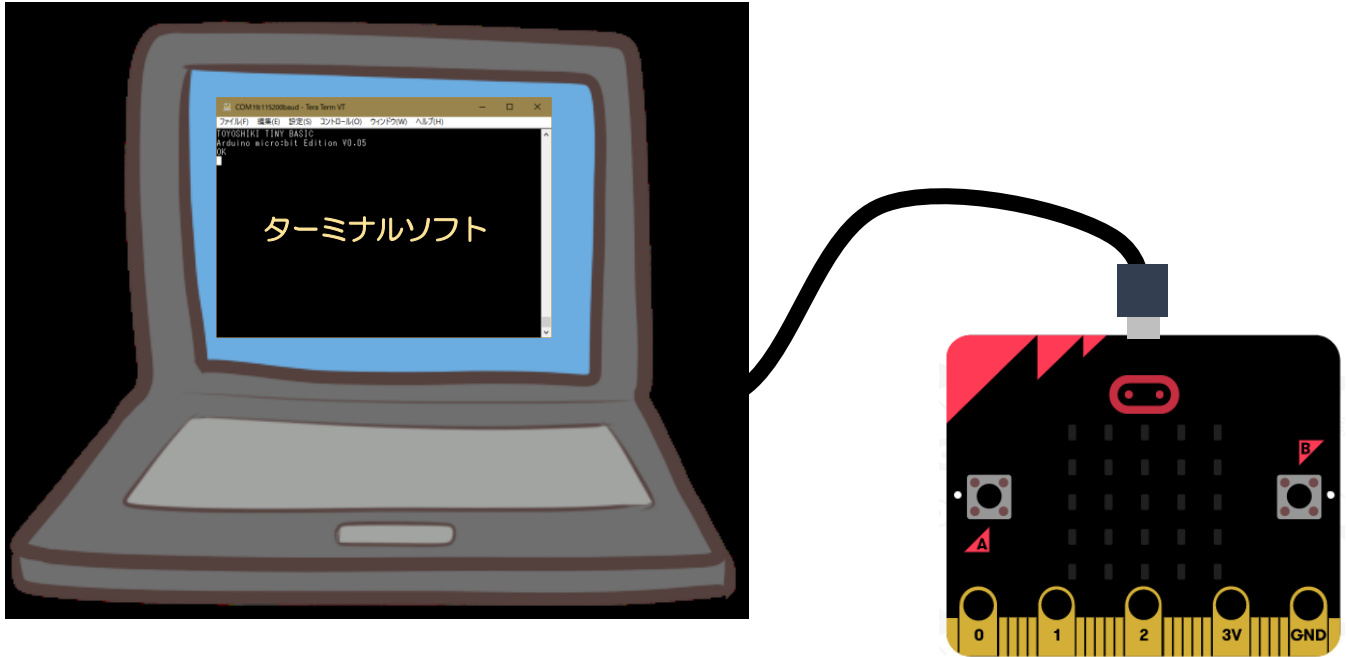
*著作権者の同意なしに経済的な利益を得てはいけません。*

*この条件のもとで、利用、複写、改編、再配布を認めます。*



## 1.4 システム構成

「豊四季 Tiny BASIC for micro:bit」は、次の構成で利用可能です。



## 必要なもの

- パーソナルコンピューター：Windows 10 パソコン ※USB ポートが利用出来ること
- Micro:bit 本体
- Micro USB ケーブル
- シリアルコンソール対応ターミナルソフト：TeraTerm (V4.89 以降)

豊四季 Tiny BASIC インタープリター言語を利用したプログラム作成は、パソコン上のターミナルソフト上にてスクリーンエディタとコマンドを使って対話形式で行います。

```

COM19:115200baud - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
list
10 'Neopixel(1)
20 NPBEGIN 0,16
30 NPCLS
40 FOR I=0 TO 7
50 NPRGB I,0,0,(255-I)-1
60 NEXT I
70 NPSHIFT 1
80 WAIT 50
90 GOTO 70
OK
  
```

Windows 以外の OS 環境でも同様の環境が構築できれば、利用出来ると思います。

## 1.5 仕様

## (1) ソフトウェア仕様

項目		概要
プログラミング環境		ターミナル上での CUI (キャラクユーザーインタフェース) USB ケーブル経由シリアル接続のスクリーンエディタ+実行環境
ターミナルスクリーン画面		16×10 文字 ~ 128 文字×30 文字の範囲で任意指定
B A S I C 言 語 仕 様	形式	整数型 BASIC (符号付 16 ビット整数 -32768~32767)
	プログラム領域	4096 バイト
	利用文字コード	1 バイト ASCII コード (半角記号・英数字、半角カタカナ) + 表示可能文字はターミナル環境のフォント設定に依存 LED マトリックス用文字は IchigoJam 互換
	利用可能変数	変数名: 208 個 A~Z の英字 1 文字 26 個, A0..A6 ~ Z0..Z6 の英字 1 文字+数字 1 文字 182 個 配列変数: 100 個 @(0) ~ @(99)
	ユーザー開放 作業用メモリ	SRAM 1024 バイト (先頭アドレス定数 MEM~)
	プログラム保存方式	内部フラッシュメモリ 8 本 (32,768 バイト)

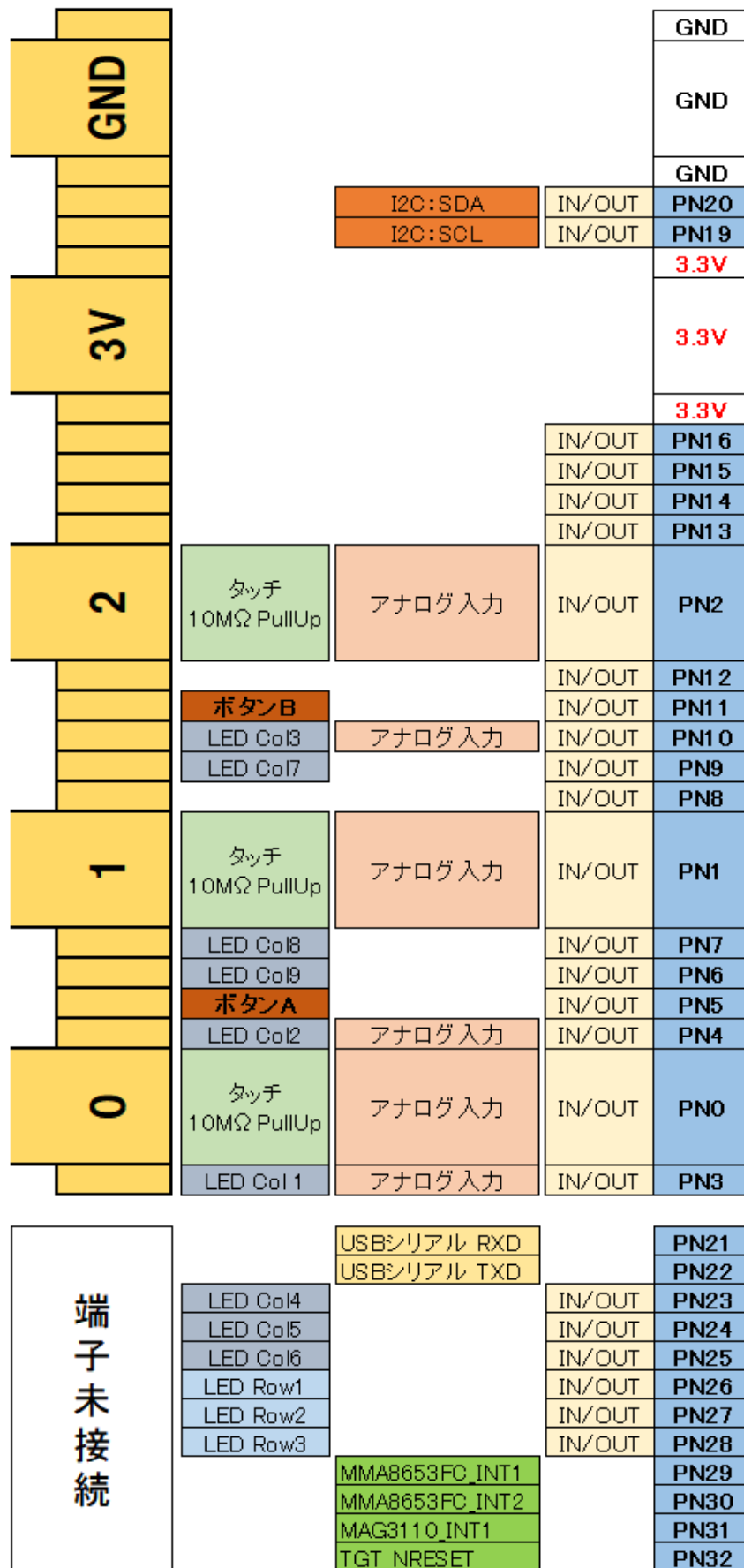
## (2) ハードウェア仕様

項目	概要
動作クロック	16MHz
LED マトリックス	5×5 ドットマトリックス (3×9 ダイナミック駆動方式)
加速度センサー	MMA8653 (3 方向)
時計機能	RTC 利用による実装 精度 (推定): ±10 分/日
サウンド	未対応 (今後対応予定)
USB I/F	マスストレージ・DFU 利用ファームウェア書込み、 USB-シリアル通信
シリアル通信	1 チャンネル: 115200、データ長 8 ビット、 ストップビット 1 パリティ None
I2C インタフェース	1 チャンネル (マスター利用のみ) アドレス 7 ビット、400kbps
GPIO	・デジタル入力 1 ビット× 20 ポート ・デジタル出力 1 ビット× 20 ポート ・PWM 出力 3 チャンネル (周波数固定、デューティ比 1/255 刻み)
アナログ入力	6 チャンネル (分解能 10 ビット)
単音出力	デジタル矩形波出力による簡易単音再生

## (3) サポートする周辺機器

- NeoPixel (WS2812)  
専用コマンドによる表示制御をサポートします。
- 圧電スピーカー  
TONE、NOTONE コマンドによる単音出力をサポートします。

## 1.6 ボードピン構成



## 1.7 ポート上のピン一覧

ピン名称	ピン番号	用途	説明
PN0	0	アナログ入力、デジタル IN/OUT、10MΩPullUp	汎用、タッチセンサー
PN1	1	アナログ入力、デジタル IN/OUT、10MΩPullUp	汎用、タッチセンサー
PN2	2	アナログ入力、デジタル IN/OUT、10MΩPullUp	汎用、タッチセンサー
PN3	3	アナログ入力、デジタル IN/OUT、LED Col1	汎用、LED マトリックス(デジタル OUT)
PN4	4	アナログ入力、デジタル IN/OUT、LED Col2	汎用、LED マトリックス(デジタル OUT)
PN5	5	デジタル IN/OUT、ボタン A (10kΩPullUp)	ボタン A (デジタル IN)
PN6	6	デジタル IN/OUT、LED Col9	汎用、LED マトリックス(デジタル OUT)
PN7	7	デジタル IN/OUT、LED Col8	汎用、LED マトリックス(デジタル OUT)
PN8	8	デジタル IN/OUT	汎用
PN9	9	デジタル IN/OUT、LED Col7	汎用、LED マトリックス(デジタル OUT)
PN10	10	アナログ入力、デジタル IN/OUT、LED Col3	汎用、LED マトリックス(デジタル OUT)
PN11	11	デジタル IN/OUT、ボタン B (10kΩPullUp)	ボタン B(デジタル IN)
PN12	12	デジタル IN/OUT	汎用
PN13	13	デジタル IN/OUT	汎用
PN14	14	デジタル IN/OUT	汎用
PN15	15	デジタル IN/OUT	汎用
PN16	16	デジタル IN/OUT	汎用
PN17	17		利用禁止
PN18	18		利用禁止
PN19	19	デジタル IN/OUT、I2C : SDA	I2C 専用
PN20	20	デジタル IN/OUT、I2C : SCL	I2C 専用
PN21	21	USB シリアル RXD	USB シリアル 専用
PN22	22	USB シリアル TXD	USB シリアル 専用
PN23	23	デジタル IN/OUT、LED Col4	LED マトリックス(デジタル OUT) 専用
PN24	24	デジタル IN/OUT、LED Col5	LED マトリックス(デジタル OUT) 専用
PN25	25	デジタル IN/OUT、LED Col6	LED マトリックス(デジタル OUT) 専用
PN26	26	デジタル IN/OUT、LED Row1	LED マトリックス(デジタル OUT) 専用
PN27	27	デジタル IN/OUT、LED Row2	LED マトリックス(デジタル OUT) 専用
PN28	28	デジタル IN/OUT、LED Row3	LED マトリックス(デジタル OUT) 専用
PN29	29	MMA8653FC INT1	利用禁止
PN30	30	MMA8653FC INT2	利用禁止
PN31	31	MG3110 INT1	利用禁止
PN32	32	TGT NRESET	利用禁止

- **色塗り部**は利用禁止
- 各ピン 5mA（ソース、シンク利用）まで電流を流すことが可能
- 全ピン合計では 90mA まで利用可能

## 1.8 メモリーマップ

「豊四季タイニーBASIC for micro"bit」ではフラッシュメモリ領域 256k バイト、SRAM は 16k バイトの領域を利用しています。ここでは、これらの領域の内訳を示します。

## フラッシュメモリ

256k バイトの領域はページ単位(1024 バイト)で領域が管理されています。

領域の利用は下記の通りです。

アドレス	ページ番号	ページ数	用途
0x00000000-0x0000BFFF	0	48	Tiny BASIC インタープリタ
0x0000C000-0x0001FFFF	49	48	未使用
0x00018000-0x0003FFFF	96	32	プログラム保存用 (4k バイト×8)

## 仮想アドレス

タイニーBASIC では、仮想アドレス利用することで SRAM 上のデータの参照・書き込みを行うことができます。

仮想アドレスは次の構成となります。

仮想アドレス	定数名	領域サイズ(バイト)	用途
\$0000	VRAM	最大 3,840	キャラクタスクリーン表示用メモリ (CW×CH) 最大 128×30 = 3,840 バイト
\$1900	VAR	420	変数領域 (A~Z, A0:A6~Z0:Z6) + 4 バイト
\$1AA0	ARRAY	200	配列変数領域 (@ (0) ~ @ (99) )
\$1BA0	PRG	4,096	プログラム領域
\$2BA0	MEM	1,024	ユーザーワーク領域
\$2FA0	FNT	1,283	フォント 256 文字
\$37A0	GRAM	27	LED マトリックス用メモリ 3×9=27 バイト

各領域のアドレス先頭は定数 VRAM, VAR, ARRAY, PRG, MEM, FNT, GRAM に定義されています。

## 1.9 文字(フォント)コード

LEDマトリックスのフォントのキャラクターを下記に示します。

フォントはサイズは5x5ドットです。IchigoJamと互換性があります。

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
\$00																
\$10																
\$20																
\$30																
\$40																
\$50																
\$60																
\$70																
\$80																
\$90																
\$A0																
\$B0																
\$C0																
\$D0																
\$E0																
\$F0																

(備考)

キャラクター表は下記のプログラムで作成しています。

```

10 'font map
20 WIDTH 128,30
30 N=16
40 GOSUB "@hr"
50 FOR C=0 TO 255 STEP N
60 GOSUB "@line(C,N)"
70 GOSUB "@hr"
80 NEXT C
90 END
100 "@line(C,N)"
110 FOR L0=0 TO 4
120 ?" |";
130 FOR I0=C TO C+N-1
140 D0=PEEK(FNT+5*I0+L0)
150 FOR J0=0 TO 4
160 IF D0&($80>>J0) ?"#"; ELSE ?" ";
170 NEXT J0
180 ?" |";
190 NEXT I0
200 ?
210 NEXT L0
220 RETURN
230 "@hr"
240 ?" ";
250 FOR I0=0 TO 95
260 ?"-";
270 NEXT I0
280 ?
290 RETURN

```

ライセンスに関する表記



CC BY IchigoJam

フォントデータは IchigoJam 1.2.1 のフォントを参考にして作成しています。

(絵文字部分は IchigoJam の8x8 フォントを見ながら自作しました)

## 1.10 編集操作操作キー

「豊四季タイニーBASIC for micro:bit」では、シリアルコンソール画面でのフルスクリーン編集をサポートしています。

## 編集キー一覧

編集キー	機能
[ESC]	プログラム中断、シリアルコンソールでは要2回押し
[F1]	画面の全消去
[F2]	カーソル位置の行消去
[F3]	カーソルの次行に空行挿入
[F5]	画面の再表示
[BackSpace]	カーソル前の文字の削除
[Insert]	上書きモード、挿入モードも切り替え
[Home]	カーソルを行の先頭に移動
[END]	カーソルを行の末尾に移動
[PageUP]	カーソルを画面右上に移動、画面のスクロールドウン
[PageDown]	カーソルを表示している最終行に移動、スクロールアップ
[Delete]	カーソル位置の文字の削除
[←]	カーソルを左に移動
[→]	カーソルを右に移動
[↑]	カーソルを上を移動
[↓]	カーソルを下に移動
[Enter]	行入力の確定、改行
[NumLock]	テンキーのロック、ロック解除
[カタカナ/ひらがな/ローマ字]	カタカナ入力のON、OFF
[Ctrl] + C	プログラム中断
[Ctrl] + D	カーソル位置の行削除
[Ctrl] + K	カタカナ入力のON、OFF
[Ctrl] + L	画面の全消去
[Ctrl] + N	カーソルの次行に空白挿入
[Ctrl] + R	画面の再表示
[Ctrl] + X	カーソル位置の文字の削除

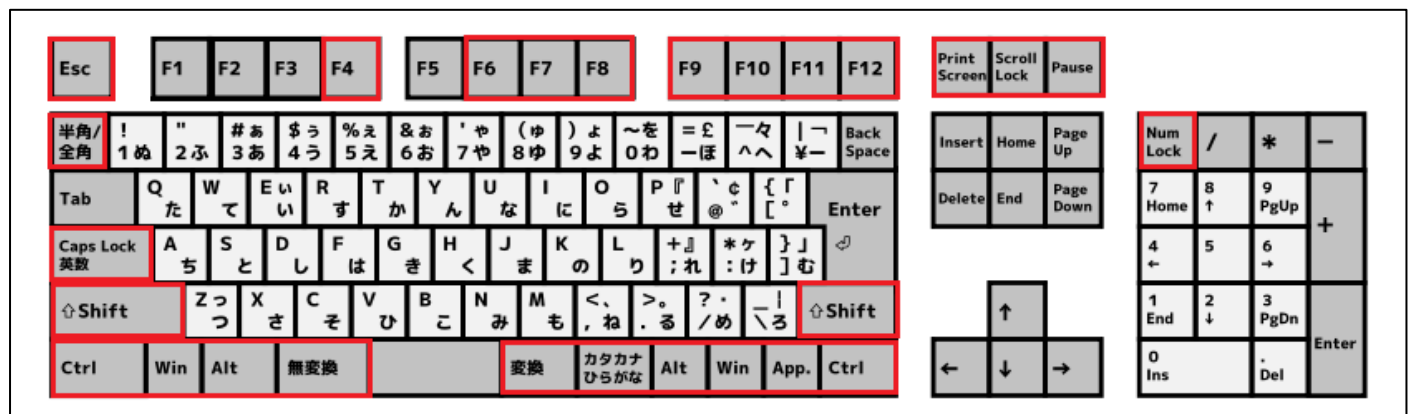
## 1.11 キー入力コード

INKEY()関数等で入力したキーを判定する際は、キー入力コードを用います。

大部分のキー入力コードは入力する文字コード、ASCIIコードと一致します。カーソルキー等の入力制御用のキーは、「豊四季タイニーBASIC for micro:bit」固有のコードとなります。

## □ キー入力コードが取得出来ないキー

赤枠のキーは、制御用または利用不可キーとなっています。



106/109 キーボード (※画像は ウィキペディア <https://ja.wikipedia.org/wiki/キー配列> より拝借)

## □ キー入力コード一覧

キーボード入力にて取得出来るキー入力コードを以下に示します。

コードに数値の記載のあるキーに限りコードの取得が可能です。

※現バージョンでは半角カタカナのローマ字入力はサポートしていません。IME にて入力して下さい。

キー	併用なし		Shift		CTRL		カタカナ		SHIFT+カタカナ	
	文字	コード	文字	コード	文字	コード	文字	コード	文字	コード
[Backspace]		8						8		8
[Enter]		13						13		13
[Tab]		9						9		9
[CapsLock]										
[Shift]										
[Ctrl]										
[Win]										
[Alt]										
[無変換]										
[空白 (Space)]	空白	32					空白	32	空白	32
[変換]										
[カカひらがなローマ字]										
[メニュー]										
[Insert]		134						134		134
[Home]		132						132		132
[PageUp]		136						136		136
[Delete]		133						133		133
[End]		137						137		137
[PageDown]		135						135		135
[↑]		129						129		129
[↓]		128						128		128
[←]		130						130		130
[→]		131						131		131
[PrintScreen]										
[ScrollLock]										
[Pause/Break]										
[! め]	1	49	!	33			1	49	!	33
[" ふ]	2	50	"	34			2	50	"	34
[# ああ]	3	51	#	35			3	51	#	35
[\$ うう]	4	52	\$	36			4	52	\$	36
[% ええ]	5	53	%	37			5	53	%	37
[& おお]	6	54	&	38			6	54	&	38
[' 7 やや]	7	55	'	39			7	55	'	39
[( ゆゆ]	8	56	(	40			8	56	(	40
[) 9 よよ]	9	57	)	41			9	57	)	41
[ 0 をわ]	0	48					0	48	0	48
[ - ほ]	-	45	=	61	CTRL -	29	-	176	=	61
[ ~ へ]	^	94	~	126			^	94	~	126
[   ￥_]	¥	92		124			¥	92		124

■はサポートしていないキーです。



キー	併用なし		Shift		CTRL		カタカナ		SHIFT+カタカナ	
	文字	コード	文字	コード	文字	コード	文字	コード	文字	コード
[Q た]	q	113	Q	81	CTRL Q	17	q	113	Q	81
[W て]	w	119	W	87	CTRL W	23	w	119	W	87
[E いい]	e	101	E	69	CTRL E	5	エ	180	エ	180
[R す]	r	114	R	82	CTRL R	18	r	114	R	82
[T か]	t	116	T	84	CTRL T	20	t	116	T	84
[Y ん]	y	121	Y	89	CTRL Y	25	y	121	Y	89
[U な]	u	117	U	85	CTRL U	21	ウ	179	ウ	179
[I に]	i	105	I	73	CTRL I	9	イ	178	イ	178
[O ら]	o	111	O	79	CTRL O	15	オ	181	オ	181
[P せ]	p	112	P	80	CTRL P	16	p	112	P	80
[`@` ]	@	64	`	96			`	222	`	96
[ [ [ ` ] ] ]	[	91	{	123			「	162	°	223
[A ち]	a	97	A	65	CTRL A	1	ア	177	ア	177
[S と]	s	115	S	83	CTRL S	19	s	115	S	83
[D し]	d	100	D	68	CTRL D	4	d	100	D	68
[F は]	f	102	F	70	CTRL F	6	f	102	F	70
[G き]	g	103	G	71	CTRL G	7	g	103	G	71
[H く]	h	104	H	72	CTRL H	8	h	104	H	72
[J ま]	j	107	J	74	CTRL J	10	j	107	J	74
[K の]	k	107	K	75	CTRL K		k	107	K	75
[L り]	l	108	L	76	CTRL L	12	l	108	L	76
[+; れ]	;	59	+	43			;	59	+	43
[*: け]	:	58	*	42	CTRL *	28	:	58	*	42
[ ] り む]	]	93	}	125			」	163	}	125
[Z っつ]	z	122	Z	90	CTRL Z	26	z	122	Z	90
[X さ]	x	120	X	88	CTRL X	24	x	120	X	88
[C そ]	c	99	C	67	CTRL C		c	99	C	67
[V ひ]	v	118	V	86	CTRL V	22	v	118	V	86
[B こ]	b	98	B	66	CTRL B	2	b	98	B	66
[N み]	n	110	N	78	CTRL N	14	ン	221	ン	221
[M も]	m	109	M	77	CTRL M	13	m	109	M	77
[<, ね]	,	44	<	60			,	164	<	60
[>. る]	.	46	>	62	CTRL >	30	.	161	>	62
[?/. め]	/	47	?	63	CTRL ?	31	/	47	?	63
[¥_ ろ]	¥	92	_	95			¥	92	_	95

■はカタカナローマ字入の子音のため、2 文字入力にてコードが確定します。

## 2. 利用環境設定

本章では、「豊四季タイニーBASIC for micro:bit」を使ってプログラムの開発を行うための環境及びその設定について、解説します。

### 2.1 ファームウェアの書き込み

「豊四季タイニーBASIC for micro:bit」を利用するには、お手持ちの micro:bit に「豊四季タイニーBASIC for micro:bit」用ファームウェアを書き込む必要があります。このファームウェアは、JavaScript ブロックエディタ作成しダウンロードして書き込むプログラムと同レベルのプログラムであり、micro:bit の動作を変更するものではありません。いつでも別のプログラムに変更して利用することが出来ます。

#### 「豊四季タイニーBASIC for micro:bit」用ファームウェア書き込み手順

- ① 「豊四季タイニーBASIC for micro:bit」プロジェクトを下記のリンクから  
ファイル `ttbasic_microbit-master.zip` をダウンロードします。  
[https://github.com/Tamakichi/ttbasic\\_microbit/archive/master.zip](https://github.com/Tamakichi/ttbasic_microbit/archive/master.zip)  
`ttbasic_microbit-master.zip`
- ② ダウンロードしたファイルを解凍します。  
エクスプローラーにて `ttbasic_microbit-master.zip` を選択し、  
マウス右クリックにて開いたメニューから「すべて展開」を選択することで解凍することが出来ます。  
解凍したフォルダ内の `bin\ttbasic_microbit.ino.hex` がファームウェアです。
- ③ micro:bit をパソコンに接続します。
- ④ エクスプローラーで micro:bit プログラムアップロード用ドライブ MICROBIT を開きます。
- ⑤ `ttbasic_microbit.ino.hex` を micro:bit プログラムアップロード用ドライブ MICROBIT に  
ドラック&ドロップ操作で書き込みます。  
書き込みが完了すると、micro:bit プログラムアップロード用ドライブ MICROBIT が再接続されます。

以上の操作で書き込みが完了します。

次のステップとしては、ターミナルソフトを使って micro:bit に接続します。

## 2.2 ターミナルソフトの通信条件の設定

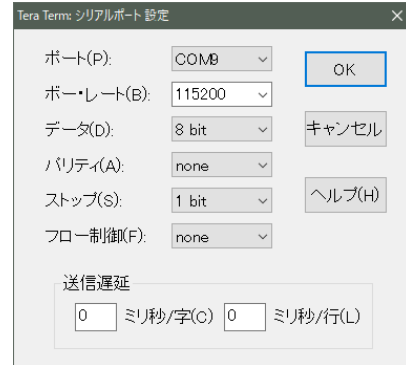
ここでは、Windows 10 パソコン上でターミナルソフト TeraTerm を利用する場合の設定について解説します。  
TeraTerm をお使いの場合は、下記のサイトからダウンロード・インストールして下さい。

### 通信条件の設定

ファームウェアを書き込んだ後、micro:bit を USB ケーブルに接続した状態にします。

#### ①通信条件の設定

ポート	設定値
ポート	各自の環境に合わせて設定
ボー・レート	115200(任意)
データ長	8 bit
パリティビット	無し
ストップビット	1bit
フロー制御	無し



USB 接続の場合、ボー・レートの設定任意です。

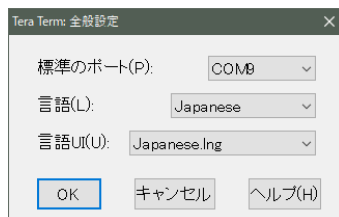
ボー・レートの設定は無視され USB 通信の最適な速度で通信を行うようです（推測）。

TeraTerm の設定は、メニュー[設定] - [シリアルポート]から[シリアルポート設定]画面を開いて行います。

#### ②言語等に関する設定

「豊四季タイニー BASIC for micro:bit」には 1 バイトコードの半角カタカナの利用をサポートします。  
半角カタカナ利用のためには、利用する文字列コードとしてシフト JIS を指定します。

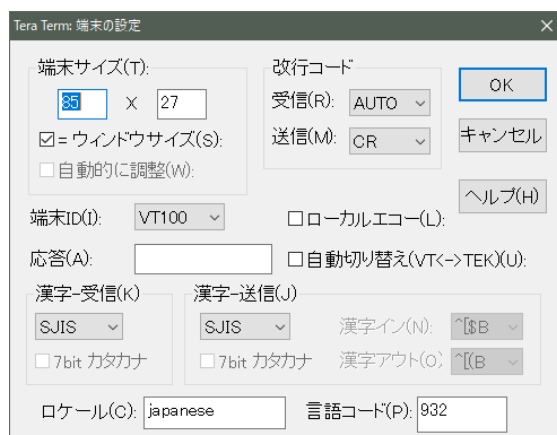
[設定] - [全般設定] - [全般設定]画面 の設定



言語 : Japanese

言語 UI : Japanese.lng

[設定] - [端末の設定] - [端末の設定]画面 の設定



端末サイズ : 80x24 以上に設定

漢字-受信 : SJIS

漢字-送信 : SJIS

改行コード受信 : AUTO、送信 CR

ウィンドウサイズ : ☒チェックを入れる

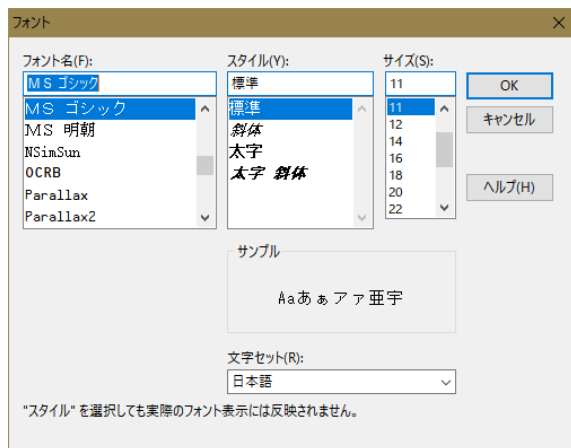
## 2.3 接続とターミナルソフトの表示設定

ターミナルソフトの設定を行ったら、micro:bit のリセットボタンを押して micro:bit を再起動します。  
再起動すると次のターミナル上に起動メッセージが表示されます。

```
TOYOSHIKI TINY BASIC
Arduino micro:bit Edition V0.05
OK
```

表示されない場合は、ケーブル及びターミナルソフトの通信条件の設定を見直して下さい。  
ターミナルソフトのウィンドウサイズをマウス操作等で変更すると表示内容が消えてしまいます。  
この場合は[F5]キーを押して、再表示して下さい。

表示された文字が小さい場合はフォントを調整して下さい。  
フォントは、文字セットに日本語が設定可能なフォントを利用して下さい。



## 2.4 コンソール画面の表示設定

「豊四季タイニーBASIC for micro:bit」のコマンドにて、スクリーンエディタの画面サイズ、フォントの色、背景色の設定を行うことができます。

### コマンド

コマンド	説明
WIDTH n, m	テキスト表示の縦・横文字数を設定します。 n: 横文字数 16~128 (デフォルト 80) m: 縦文字数 10~30 (デフォルト 24)
COLOR n [, m]	テキスト文字色、背景色を設定します。 n: 文字色 0~8 m: 背景色 0~8  COLOR コマンドの色指定については、「6.35 COLOR 文字色の設定」を参照して下さい。 COLOR コマンド実行後、CLS コマンドを実行するか、[F1]キーを押すことで画面全体に色設定が反映されます。
ATTR n	文字表示属性を設定します。 n: 0 ノーマル、 1 下線、 2 反転、 3、ブリンク、 4 ボールド
CLS	画面表示のクリア キーボードの [F1]キー、CTRL+L キーでもクリアできます。

(補足) ターミナルソフト TeraTerm の設定により、利用するフォントの種類・サイズ等の指定を行うことができます。  
TeraTerm の設定については、TeraTerm のマニュアルを参照して下さい。

## 2.5 プログラムの保存と読み込み

作成したプログラムは内部フラッシュメモリに保存可能です。

フラッシュメモリへの保存 [SAVE](#) プログラム番号(0~7)

フラッシュメモリからの読み込み [LOAD](#) プログラム番号(0~7)

フラッシュメモリに保存しているプログラムのリスト表示: [FILES](#)

```
files
0:CLS
1:'oscilloscope
2:'ジ コレヨカジ
3:(none)
4:(none)
5:(none)
6:(none)
7:(none)
OK
```

FILES コマンドはプログラムの先頭行を表示します。

プログラム番号の格納領域にプログラムが保存されていない場合は、(none)が表示されます。

フラッシュメモリ上のプログラムは ESASE コマンドにて削除することが出来ます。

[LRUN](#) コマンドを用いると、指定したプログラムをロードして実行することが出来ます。

```
LRUN 1
```

マイコンボードの起動時に予め指定したプログラムを起動することも可能です。

詳細は、「2.10 プログラムの自動起動」を参照して下さい。

## 2.6 プログラム保存領域の初期化

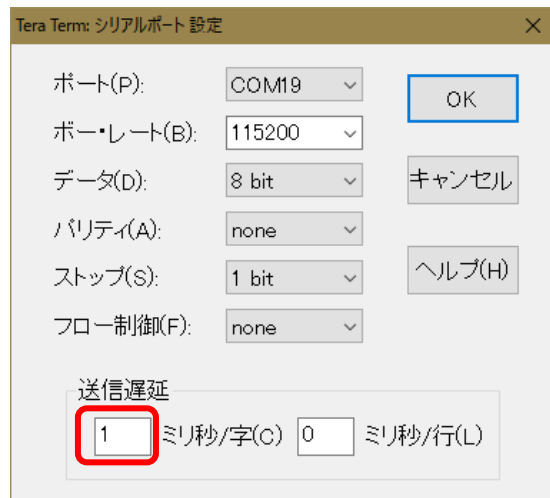
内部フラッシュメモリの初期化は、[ERASE](#) コマンドを利用します。

全領域を初期化するには次のコマンドを実行します。

```
ERASE 0,7
OK
```

## 2.7 パソコンからのプログラム読み込み

パソコンからマイコンボードにプログラムを転送する方法としては、  
ターミナルソフトの画面にプログラムソースコピー＆ペーストする方法が簡単でおすすめです。  
この際、シリアルポートの設定にて[送信遅延]を下記のように設定して下さい。



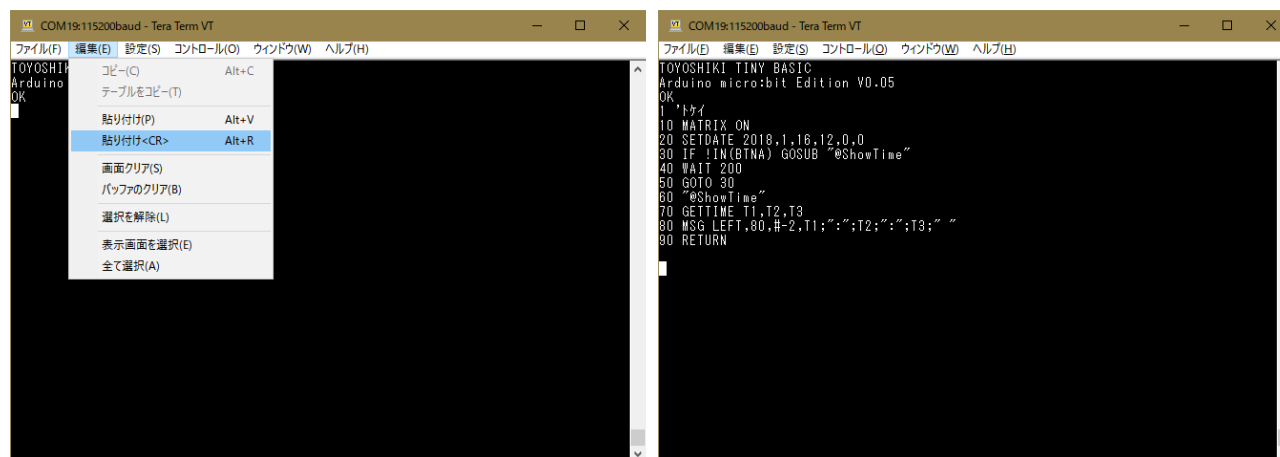
試しに、次のプログラムをコピーしてパソコンから micro:bit にプログラムを転送してみましょう。

```

1 'トイ
10 MATRIX ON
20 SETDATE 2018,1,16,12,0,0
30 IF !IN(BTNA) GOSUB "@ShowTime"
40 WAIT 200
50 GOTO 30
60 "@ShowTime"
70 GETTIME T1,T2,T3
80 MSG LEFT,80,#-2,T1;";";T2;";";T3;" "
90 RETURN

```

TeraTerm では、メニュー [編集] - [貼り付け<CR>] か ALT+R キーにて貼り付けることができます。



(注意) プログラムの読み込みが終わったら、送信遅延の設定は元に戻して下さい。プログラム実行時にコンソール画面からの文字入力を行うコマンド INPUT、INKEY()にて遅延が発生し、意図しない動作をする場合があります。

## 2.8 フラッシュメモリのエクスポート

内部フラッシュメモリに保存されている、プログラムをパソコンにバックアップする機能として、[EXPORT](#) コマンドが用意されています。

パソコン上のターミナルソフトから接続した状態で、EXPORT コマンドを実行します。

ターミナルソフト上に保存されているプログラムが全て出力されますので、その内容をコピー＆ペーストにてメモ帳等に張り付けて保存します。

```
export
NEW
10 CLS
20 GETTIME H,M,S
30 @(0)=H/10: @(1)=H%10
40 @(2)=10
50 @(3)=M/10: @(4)=M%10
60 @(5)=10
70 @(6)=S/10: @(7)=S%10
80 FOR I=0 TO 7
90 BITMAP I*24+10,20,FNT,@(I)+ASC("0"),6,8,4
100 NEXT I
110 GOTO 20
SAVE 0

NEW
10 'oscilloscope
20 CLS
30 GPIO PB1,ANALOG
40 "LOOP"
50 R=ANA(PB1)
60 Q=R/20
70 LOCATE 0,0: ?R;" "
80 PSET GW-1,GH-Q-4,1
90 WAIT 50
100 GSCROLL 8,208,3
110 GOTO "LOOP"
SAVE 1

NEW
1 'シリアル'
5000 CLS
5010 GETTIME H,M,S
5020 @(0)=H/10: @(1)=H%10
5030 @(2)=10
5040 @(3)=M/10: @(4)=M%10
5050 @(5)=10
5060 @(6)=S/10: @(7)=S%10
5070 FOR I=0 TO 7
5080 BITMAP I*24+10,20,FNT,@(I)+ASC("0"),6,8,4
5090 NEXT I
5100 GOTO 20
SAVE 2

OK
```

## 2.9 保存したプログラムをマイコンボードに読み込むには「2.6 プログラム保存領域の初期化

内部フラッシュメモリの初期化は、ERASE コマンドを利用します。

全領域を初期化するには次のコマンドを実行します。

```
ERASE 0,7
OK
```

パソコンからのプログラム読込」の内容を参考にして、  
保存していたテキストの「NEW～SAVE X」の単位でコピー＆ペーストしてプログラム番号単位で保存していきます。  
作業については、「6.14EXPORT 内部フラッシュメモリの内容のエクスポート」の内容も参照して下さい。



## 2.10 プログラムの自動起動方法

Micro:bit の起動時またはリセット時にフラッシュメモリー上のプログラム番号 No.0 のプログラムを起動することが出来ます。その方法について説明します。

### 方法1

- ・電源 ON またはリセット時に **B ボタン** を押す

**B ボタン** は PN11 に接続しているため、PN11 を GND に接地することで **B ボタン** を押していることになります。

### 方法2

- ・**ボタン A、ボタン B** を5秒間押し続け、**A ボタン** のみ離す

**B ボタン** は PN11 に接続しているため、PN11 を GND に接地することで **B ボタン** を押していることになります。

micro:bit をバッテリー用コネクタにて電源供給（電池で稼働されている）している場合、**リセットボタン** によるリセットを行うことが出来ません。この場合は方法2により、プログラムの自動起動を行って下さい。

また、リセットを行う場合は **A ボタン** のみを5秒間押し続けた後、離すことでも行うことが出来ます。

## 2.11 RTC の時刻設定

RTC の時刻設定は次のコマンドで行います。

SETDATE 年,月,日,時,分,秒

例：

```
SETDATE 2018,1,18,12,0,0
OK
```

設定した時刻は DATE コマンドで確認できます。

例：

```
DATE
2017/08/10 [Thr] 12:32:45
OK
```

設定した時刻は、電源を切るか、リセットすると初期化されます。

時刻の初期値は 1970/1/1 00:00:00 です。

### 3. 使ってみよう！

この章では、「レガシーBASIC（古き良き時代の BASIC 言語）の雰囲気を知らない方々にその雰囲気をつかんでもらいましょう」ということで、軽いフットワークで解説進めます。

準備は OK？ 前章の実行環境の設定は完璧？ それでは電源を入れて使ってみよう！

#### 3.1 入力可能状態は OK■（カーソル）

```
TOYOSHIKI TINY BASIC
Arduino micro:bit Edition V0.05
OK
■
```

この状態は、「いつでもこいや〜」状態。  
きみのコマンド入力を待っている状態。  
何か打ち込んでみよう！

#### 3.2 Hello,world を表示してみる

直接、次の文字を打ってみよう。

```
print "Hello,world" Enter
```

入力ミスの修正は[BS]キー、[DEL]キー、カーソルキーなどいつものキーが使えるよ。  
入力出来たら、最後に[Enter]キーを押してみよう。

```
print "Hello,world" Enter
Hello,world
OK
■
```

表示出来たね。PRINT は文字列を表示するコマンドなのです！  
入力は大文字、小文字どちらでも OK！

#### 3.3 Hello,world を Hello,Tiny BASIC に変更してみる

カーソルキー[↑][↓][←][→]でカーソルを動かして、world の w に移動してみよう。

```
print "Hello,world"
Hello,world
```

[DEL]キーで world を削除して、代わりに Tiny BASIC と入力しよう。  
入力後に、[Enter]キーを押すのを忘れずに！

```
print "Hello,Tiny BASIC" Enter
Hello,Tiny BASIC
OK
■
```

こんな感じで[Enter]が入力確定とコマンド実行の合図だ。  
[Enter]を入力する際、カーソルは行のどこにあっても構わない。

### 3.4 PRINT の前に 10 を追加してプログラムにしよう

再びカーソル操作で print の前に 10 を挿入して[Enter]キーを押してみよう。

```
10print "Hello,Tiny BASIC" Enter
Hello,Tiny BASIC
OK
```

これでプログラムとして登録された。LIST コマンドで確認してみよう。  
カーソルを OK の下の空き行に移動して、list と入力して[Enter]キーを入力

```
10 print "Hello,Tiny BASIC" Enter
Hello,Tiny BASIC
OK
list Enter
10 PRINT "Hello,Tiny BASIC"
OK
■
```

登録したプログラムの内容が表示されたね。

まだ、行番号 10 しか登録していないので、その 1 行だけの表示だ。

このように、先頭に数字(=行番号)をつけると行番号の後ろのコマンドがプログラムとして登録されるのだ。

### 3.5 プログラムを実行してみよう

う〜ん、気分的に、画面を綺麗にしてから表示したくなった。

画面をきれいにしてから、3回実行してみよう。

```
cls Enter
run Enter
Hello,Tiny BASIC
OK
run Enter
Hello,Tiny BASIC
OK
run Enter
Hello,Tiny BASIC
OK
```

3回同じ処理が実行出来たね。

CLS は画面を消すコマンド、RUN はプログラムを実行。よく使うコマンドだよ。

### 3.6 プログラムに追加する

今の“画面きれいにしてから、3回表示”をプログラムにやらせてみよう！

次のように入力して、プログラムとして登録するよ。

```
5 cls Enter
7 for i=1 to 3 Enter
20 next i Enter
```

入力したプログラムを確認！

```
List Enter
5 CLS
7 FOR I=1 TO 3
10 PRINT "Hello,Tiny BASIC"
20 NEXT I
OK
```

今入力した 5,7,10 行が追加されたね。プログラムは行番号順に表示されるよ。

それでは実行！ run コマンドで実行するよ。

```
run Enter
Hello,Tiny BASIC
Hello,Tiny BASIC
Hello,Tiny BASIC
OK
```

3 行表示出来た。

プログラムの実行順番は行番号順。CLS が最初に実行されるよ。

FOR 文は繰り返しを行う命令。3 回 10 行の表示を行ったよ。

## 3.7 行番号を並びなおす

もう一回、プログラムを確認しよう。

```
List Enter
5 CLS
7 FOR I=1 TO 3
10 PRINT "Hello,Tiny BASIC"
20 NEXT I
OK
```

行番号の 5,7,10,20 と並びがちょっと美しくないですね。

直してみよう。

RENUM コマンドを使ってプログラムの行番号を 10 ずつに整えるよ。

```
renum Enter
OK
List Enter
10 CLS
20 FOR I=1 TO 3
30 PRINT "Hello,Tiny BASIC"
40 NEXT I
OK
```

きれいに並んだね。

## 3.8 やっぱり 10 行は不要、削除したい

やっぱり、直前の表示内容は消したくない。

10 行はいらない。削除したい。行番号だけ入力して[Enter]を押してみよう。

```
10 Enter
```

これで 10 行が削除されるよ。確認してみよう。

```
List Enter
20 FOR I=1 TO 3
30 PRINT "Hello,Tiny BASIC"
40 NEXT I
OK
```

10 行が無くなったね。

こんな感じ行番号だけの入力で、行削除。よく使う操作だよ。

DELETE コマンドを使っても行の削除が出来るよ。

```
DELETE 10 Enter
```

20 行から始まるのは美しくない。行番号をまた振り直そう。

```
renum Enter
OK
List Enter
10 FOR I=1 TO 3
20 PRINT "Hello,Tiny BASIC"
30 NEXT I
OK
```

### 3.9 プログラムを保存したい

今作成したプログラムは SRAM 上に保存されているのだ。

マイコンの電源を切ると消えてしまう・・・

消えないで欲しい・・・。そんな時、SAVE コマンドを使うよ。

SAVE コマンドはプログラムをフラッシュメモリに保存するよ。

フラッシュメモリに保存されたプログラムは電源を切っても消えないよ。

次のコマンドを打ってみましょう。

```
save Enter
OK
```

これで保存できたはずだ。

### 3.10 保存されたプログラムを読み込み

本当に保存されているか確認してみましょう。

SRAM 上のプログラムを消してフラッシュメモリか読み込みと試してみよう。

NEW コマンドを打ってプログラムを初期化するよ。

```
new Enter
OK
```

これで消えたはず。

LIST コマンドで確認。

```
list Enter
OK
```

何も出てこない。本当に消えた。

次に LOAD コマンドでフラッシュメモリからプログラム読み込むよ。

```
load Enter
OK
list Enter
10 FOR I=1 TO 3
20 PRINT "Hello,Tiny BASIC"
30 NEXT I
OK
```

成功！ 復活出来た！・・・

保存しているプログラムは FILES コマンドで確認できるよ。

0 番に保存されているよ。

```
files Enter
0:FOR I=1 TO 3
1:(none)
2:(none)
3:(none)
4:(none)
5:(none)
6:(none)
7:(none)
OK
```

これで基本的な操作は終了だ。

基本はバッチリ、準備万端！ Tiny BASIC を使った更なるプログラミングをエンジョイしよう！

## 4. プログラム構成要素の説明（コマンド・関数等）

本章では、「豊四季タイニーBASIC for micro"bit」のプログラム言語について解説します。

### 4.1 プログラムの構造と形式

次のプログラムリストは、LED マトリックスに時刻を表示するプログラムです。

実行し、A ボタンを押すと時刻がスクロール表示されます。

```

1 'トイ
10 MATRIX ON
20 SETDATE 2018,1,16,12,0,0
30 IF !IN(BTNA) GOSUB "@ShowTime"
40 WAIT 200
50 GOTO 30
60 "@ShowTime"
70 GETTIME T1,T2,T3
80 MSG LEFT,80,#-2,T1;":";T2;":";T3;" "
90 RETURN

```

プログラムの構造的は・・・

プログラムは複数の行（行番号＋命令文）で構成されます。

例）1 'トイ ：プログラム内容を説明するコメントです。

例）10 MATRIX ON ：LED マトリックスを有効にしています。

命令文は1 つまたは複数のコマンドや式で構成されます。

例）30 IF !IN(BTNA) GOSUB "@ShowTime" ：A ボタンが押されたらサブルーチンを呼び出しています。

コマンドはコマンド名と引数で構成されます。

例）20 SETDATE 2018,1,16,12,0,0 ：時刻を設定しています。

引数は式、変数、定数にて構成されます。

例）80 MSG LEFT,80,#-2,T1;":";T2;":";T3;" " ：変数に格納された時刻を LED マトリックスに表示しています。

基本的に、プログラムは行番号順にシーケンシャルに実行されます。

例）1 行から 30 行は順番に実行されます。

制御命令により、分岐や繰り返しを行うことができます。

例）30 行は A ボタンが押された場合、60 行から 90 行のサブルーチンに分岐実行後、復帰します。

例）50 行は GOTO 文で無条件で 30 行にジャンプします。

このようにプログラムは様々な要素で構成れます。

次節からは、これらの各構成要素について解説します。

## 4.2 行と命令文

### □ 行

行とは利用者がコンピュータに対して指示を行う単位です。行は命令文で構成されます。  
「豊四季タイニーBASIC for micro"bit」では、行の単位で利用者からの指示を処理します。

スクリーンエディタ、およびコンソールから、

```
PRINT "Hello" Enter
Hello
OK
```

のように **Enter** キーの入力により、行の単位で指示を受け取りその処理を行います。  
また、行の先頭に行番号を付けた場合、

```
10 PRINT "Hello" Enter
```

#### 「指定した行番号で行（命令文）をプログラムに登録せよ」

との指示であると判断し、命令文は実行せずに、行（命令文）をプログラムとしてメモリーに登録します。

```
10 Enter
```

のように行番号だけを指定した場合は、

#### 「指定した行番号で行（命令文）を空にせよ（削除せよ）」

との指示であると判断し、該当する行をプログラムから削除します。

### □ 命令文

命令文とはコマンド、式、関数、コメント、ラベルを組み合わせて記述した命令です。

コマンドはコロン:を使って1行に複数記述することができます。

```
10 I=I+1:LOCATE 0,I:COLOR 7,0:PRINT "Hello":GOTO 50:'サンプル
```

### コメント文

文において、REM、および省略形のシングルクォーテーション'を使った以降の文はコメントとなります。

例：

```
10 'Smple program
20 REM Sample program
30 PRINT "Hello":'print hello
```

### ラベル

行の先頭のダブルクォーテーションで囲った文字列はラベルとして動作します。

ラベル自体は実行時に何も行いません。GOTO 文、GOSSUB 文のジャンプ先の指定で利用します。

ラベルの後ろにはコマンドを続けて記述することができます。

例：

```
10 "LOOP":PRINT "Hello"
20 GOTO "LOOP"
```

## 4.3 制御文

制御文は制御命令を使ったプログラムの記述文です。複数のキーワード、式の組み合わせで構成されます。プログラムは RUN コマンドを実行することにより、通常は先頭行から行番号順に逐次実行されます。この逐次実行は制御文を用いることで条件分岐、繰り返しを行うことが出来ます。

FOR 文の例：繰り返し処理を行う

```
10 FOR A=0 TO 10 STEP 5
20 PRINT A
30 NEXT A
```

以下に制御命令の一覧を示します。

制御命令	説明
<a href="#">GOTO</a> 行番号 <a href="#">GOTO</a> ラベル	指定行へのジャンプ
<a href="#">GOSUB</a> 行番号 <a href="#">GOSUB</a> ラベル	サブルーチンの呼び出し
<a href="#">RETURN</a>	GOSUB 文サブルーチン呼び出しからの復帰
<a href="#">IF</a> 条件式 実行文 1 <a href="#">IF</a> 条件式 実行文 1 ELSE 実行文 2	条件判定
<a href="#">FOR</a> 変数=初期値 TO 最終値 実行文(複数行可能) NEXT 変数 <a href="#">FOR</a> 変数=初期値 TO 最終値 STEP 増分 繰り返し実行文 NEXT 変数	繰り返し
<a href="#">END</a>	プログラムを終了する

各制御命令の詳細については、「5 各制御文の詳細」を参照して下さい。

## 4.4 コマンド・関数

### コマンド

コマンドとは何らかの処理を行うプログラム要素です。コマンドには一般コマンドとシステムコマンドの2種類があります。一般コマンドはコマンドラインからの直接利用とプログラム内での利用が可能です。

システムコマンドは、プログラムの管理を行うコマンドです。プログラム中で利用した場合、不整合が生じるため、コマンドラインでのみ利用可能としています。

システムコマンドをプログラム中で利用した場合、エラー("Cannot use system command")となります。

例：プログラム内にシステムコマンド [DELETE](#) を記述

```
10 ? "SAMPLE"
20 DELETE 10
30 END
RUN
Cannot use system command in 20
20 DELETE 10
OK
```

### 関数

関数とは何らかの値を返す命令文です。関数には数値関数と文字列関数の2種類があります。

数値関数は数値を返す関数です。文字列関数は文字列を返す関数です。

数値関数はコマンドの引数や式のなどの数値として利用します。

文字列関数は PRINT,SPRINT,GPRINT の引数、ファイル名の引数にて利用できます。

例：関数の利用

```
10 A=ASC("A")*5+FNT
20 FOR I=0 TO 4
30 PRINT BIN$(PEEK(A+I)>>3,5)
40 NEXT I
50 END
```



```
run
01100
10010
11110
10010
10010
OK
```

上記のプログラムは文字“A”のフォントパターンを2進数で表示します。

10行にて数値関数 [ASC\(\)](#) を利用しています。30行で文字列関数 [BIN\\$\( \)](#) を利用しています。

関数単独での利用は出来ません。コマンドや関数の引数、式中でのみ利用可能です。

例：関数の利用は文法エラーとなる

```
ASC("A")
Syntax error
OK
```

## 4.5 コマンド・関数一覧

機能別に分類したコマンド及び関数の一覧を示します。

各コマンド及び関数の利用方法の詳細については、「7.各コマンド・関数の詳細」を参照して下さい。

### □ システムコマンド

<a href="#">RUN</a>	プログラムの実行
<a href="#">RENUM</a>	行番号の振り直し
<a href="#">RENUM</a> 先頭行番号	先頭行番号：振り直し開始先頭行
<a href="#">RENUM</a> 先頭行番号,間隔	間隔：行間の増分
<a href="#">DELETE</a> 行番号	プログラムの指定行の削除
<a href="#">DELETE</a> 先頭行番号,末尾行番号	

### □ プログラム関連

#### コマンド

<a href="#">WIDTH</a> 横文字数,縦文字数	シリアルターミナルの画面サイズの設定
<a href="#">LIST</a>	プログラムリストの表示
<a href="#">LIST</a> 開始行番号	開始行番号：表示を開始する行
<a href="#">LIST</a> 開始行番号,終了行番号	終了行番号：表示を終了する行
<a href="#">NEW</a>	プログラムの消去
<a href="#">LOAD</a>	内部フラッシュメモリからプログラム読み込み
<a href="#">LOAD</a> プログラム番号	プログラム番号：読み出し元番号 (0~7)
<a href="#">SAVE</a>	内部フラッシュメモリ/SDカードへのプログラム保存
<a href="#">SAVE</a> [プログラム番号]	プログラム番号：保存先番号 (0~9)
<a href="#">SAVE</a> “ファイル名”	“ファイル名”：SDカード上のファイル名
<a href="#">FILES</a>	内部フラッシュメモリのプログラム一覧表示
<a href="#">FILES</a> 開始[,終了]	開始,終了:プログラム番号
<a href="#">REM</a> [コメント文] or `[コメント文]	コメント 省略形の` (シングルクォーテーション)も可能
<a href="#">LET</a> 変数=式	変数・配列変数に値を代入(LETは省略可能)
<a href="#">LET</a> @(添え字)=n1[,n2,n3,n4...]	配列変数では連続代入指定が可能
<a href="#">CLV</a>	変数領域の初期化
<a href="#">LRUN</a> プログラム番号	指定したプログラムを内部フラッシュメモリから読んで実行する
<a href="#">LRUN</a> プログラム番号,行番号	
<a href="#">LRUN</a> プログラム番号,“ラベル”	
<a href="#">EXPORT</a>	内部フラッシュメモリの内容をエクスポートする
<a href="#">EXPORT</a> 対象プログラム番号	対象番号 エクスポートする番号
<a href="#">EXPORT</a> 開始プログラム番号,終了プログラム番号	開始番号, 終了番号 エクスポートする範囲
<a href="#">ERASE</a> プログラム番号	内部フラッシュメモリ上のプログラム削除
<a href="#">ERASE</a> 開始プログラム番号,終了プログラム番号	

## 数値関数

<a href="#">ABS</a> (整数)	絶対値の取得
<a href="#">ASC</a> (変数)	変数が参照する文字列の取得/文字列の切り出し 例 A=ASC("ABCD",2) S="ABC":B=ASC(S,2)
<a href="#">ASC</a> (変数,文字位置)	
<a href="#">ASC</a> (文字列)	
<a href="#">ASC</a> (文字列,文字位置)	
<a href="#">FREE</a> ()	プログラム領域の残りバイト数の取得
<a href="#">INKEY</a> ()	キー入力の読み取り
<a href="#">RND</a> ()	乱数の発生
<a href="#">LEN</a> (変数)	文字列長の取得
<a href="#">LEN</a> (文字列)	
<a href="#">MAP</a> (値,開始範囲 1,終了範囲 1, 開始範囲 2,終了範囲 2)	数値のスケール変換
<a href="#">GRADE</a> (値, 配列 No,個数)	指定した値の等級を求める

## 文字列関数

<a href="#">CHR\$</a> (文字コード)	画文字コードから文字への変換
<a href="#">BIN\$</a> (数値[,桁指定])	数値から 2 進数文字列への変換
<a href="#">HEX\$</a> (数値[,桁指定])	数値から 16 進数文字列への変換
<a href="#">DMP\$</a> (数値)	整数の小数付き数値文字列への変換 小数桁数 0~4 整数部桁数 0~8
<a href="#">DMP\$</a> (数値,小数桁数)	
<a href="#">DMP\$</a> (数値,小数桁数,整数部桁数)	
<a href="#">STR\$</a> (変数)	変数が参照する文字列の取得
<a href="#">STR\$</a> (変数,先頭,長さ)	
<a href="#">STR\$</a> (文字列)	
<a href="#">STR\$</a> (文字列,先頭,長さ)	

## □ 時間待ち・時間計測関連

## コマンド

<a href="#">WAIT</a> ミリ秒	時間待ち
--------------------------	------

## 数値関数

<a href="#">TICK</a> ()	起動からの経過時間取得（ミリ秒単位）
-------------------------	--------------------

## □ 記憶領域操作関連

## コマンド

<a href="#">POKE</a> アドレス,データ[,データ...データ]	指定アドレスへのデータ書き込み
---	-----------------

## 数値関数

<a href="#">PEEK</a> (アドレス)	指定アドレスの値参照
-----------------------------	------------

## □ キャラクタ表示関連

## コマンド

<a href="#">PRINT</a> [#n.] 数値・文字列[: 数値・文字列...][:]	画面への文字表示 文末:で改行の抑制可能
<a href="#">INPUT</a> 変数	数値の入力 例: INPUT "A=",A,-1
<a href="#">INPUT</a> 変数,オーバーフロー時の既定値	
<a href="#">INPUT</a> プロンプト, 変数	
<a href="#">INPUT</a> プロンプト, 変数, オーバーフロー時の既定値	
<a href="#">CLS</a>	画面表示内容の全消去
<a href="#">COLOR</a> 文字色	文字色の設定 ※ターミナル版でのみ利用可能
<a href="#">COLOR</a> 文字色,背景色	
<a href="#">ATTR</a> 属性	文字表示属性設定 ※ターミナル版でのみ利用可能
<a href="#">LOCATE</a> 横位置,縦位置	カーソルの移動
<a href="#">REDRAW</a>	画面表示の再表示

## 数値関数

<a href="#">VPEEK</a> (横位置,縦位置)	画面指定位置の文字コード参照
---------------------------------	----------------

## □ LED マトリックス表示関連

## コマンド

<a href="#">MATRIX</a> ON/OFF	LED マトリックス利用の有効/無効設定
<a href="#">PSET</a> x, y, 色	点の描画
<a href="#">LINE</a> x1,y1,x2,y2,色	直線の描画
<a href="#">RECT</a> x1,y1,x2,y2,色,モード	矩形の描画
<a href="#">CIRCLE</a> x, y,半径,色,モード	円の描画
<a href="#">BITMAP</a> x, y, アドレス, インデックス, 幅, 高さ [,倍率]	ビットマップ画像の表示
<a href="#">GPRINT</a> X,Y,[#n.] 数値・文字列[:数値・文字列...][:]	指定位置に文字列描画
<a href="#">GSCROLL</a> x1,y1,x2,y2,方向	指定領域内でのグラフィックススクロール
<a href="#">MSG</a> スクロール方向, ウェイト時間, メッセージ文	メッセージ文の表示
<a href="#">SETFONT</a> 文字コード,d1,d2,d3,d4,d5	フォントの定義
<a href="#">CLP</a>	フォント定義を初期状態に戻す

## 数値関数

<a href="#">GPEEK</a> (横位置,縦位置)	画面上の指定位置ピクセルの参照
<a href="#">GINP</a> (横位置,縦位置,高さ,幅,色)	指定領域のピクセルの有無判定

## □ NeoPixel 制御関連

## コマンド

<a href="#">NPBEGIN</a> ピン番号, LED 数	NeoPixel の利用を開始する
<a href="#">NPEND</a>	NeoPixel の利用を終了する
<a href="#">NPCLS</a>	LED の表示クリア
<a href="#">NPRGB</a> LEDNo, R, G, B [表示指定]	指定した LED の表示色 RGB 指定
<a href="#">NPPSET</a> LEDNo, 色 [表示指定]	指定した LED の表示色指定
<a href="#">NPPUT</a> LEDNo, 仮想アドレス, 転送数, バイト長 [表示指定]	ピクセルデータの転送
<a href="#">NPSHOW</a>	設定を表示に反映する
<a href="#">NPSHIFT</a> UP/DOWN [, ループ指定 [表示指定]]	表示を1つシフトする
<a href="#">NPLEVEL</a> 輝度	NPPSET, NPPUT で指定した色の輝度補正値を設定する

## 数値関数

<a href="#">RGB</a> (R,G,B)	R, G, B から 2 バイト色コード (0~32767) を求める
<a href="#">RGB8</a> (R,G,B)	R(0-7), G(0-7), B(0-3) から色コード (0~255) を求める

## □ サウンド関連

## コマンド

<a href="#">TONE</a> 周波数,出力期間	指定周波数のパルス出力
<a href="#">NOTONE</a>	パルス出力の停止
<a href="#">SETTONE</a> ピン番号	単音出力ポートの設定
<a href="#">PLAY</a> "MML 文"	MML による単音音楽演奏
<a href="#">TEMPO</a> テンポ	MML による単音音楽演奏のテンポの設定 (デフォルト 120)

## □ RTC (時刻) 関連

## コマンド

<a href="#">DATE</a>	現在時刻の表示
<a href="#">GETDATE</a> 変数 1,変数 2,変数 3,変数 4	日付の取得 変数 1 から順に年,月,日,曜日コードを格納
<a href="#">GETTIME</a> 変数 1,変数 2,変数 3	時刻の取得 変数 1 から順に時,分,秒を格納
<a href="#">SETDATE</a> 年,月,日,時,分,秒	時刻の設定

## □ GPIO・入出力関連

## コマンド

<a href="#">GPIO</a> ピン名 ピン番号,機能名	GPIO 機能設定 ピン名利用時は GPIO 記述省略可能
<a href="#">OUT</a> ピン番号,出力値	デジタル出力
<a href="#">POUT</a> ピン番号, デューティ値	PWM パルス出力
<a href="#">POUT</a> ピン番号, デューティ値,周波数	
<a href="#">SHIFTOUT</a> データピン,クロックピン, 出力形式,出力データ	デジタルシフトアウト出力

## 数値関数

<a href="#">IN</a> (ピン番号)	デジタル入力
<a href="#">ANA</a> (ピン番号)	アナログ入力
<a href="#">SHIFTIN</a> (データピン番号, クロックピン番号, 入力形式)	デジタルシフトイン入力
<a href="#">SHIFTIN</a> (データピン番号, クロックピン番号, 入力形式,条件)	
<a href="#">PULSEIN</a> (パルス入力ピン番号,検出信号,タイムアウト)	入力パルス幅の測定
<a href="#">PULSEIN</a> (パルス入力ピン番号,検出信号,タイムアウト, スケール)	
<a href="#">I2CR</a> (デバイスアドレス,コマンドアドレス,コマンド長, データアドレス,データ長)	I2C スレーブデバイスからのデータ受信
<a href="#">I2CW</a> (デバイスアドレス,コマンドアドレス,コマンド長, データアドレス,データ長)	I2C スレーブデバイスへのデータ送信

システムコマンド、一般コマンドの詳細については「7 各コマンド・関数」を参照して下さい。

## 4.6 数値

### □ 数値

「豊四季タイニーBASIC for micro:bit」で使える数値は整数型のみとなります。

整数型は 16 ビット幅、有効範囲は-32768～32767 となります。式、数値定数、数値関数、変数、配列変数はすべてこれに従います。

数値の表記は 10 進数、16 進数、2 進数形式の表記可能です。

10 進数表記 (-32768～32767)   : -1, -32767, 100  
 16 進数表記 (1 桁～4 桁)       : \$1345, \$abcd, \$Abcd  
 2 進数表記 (1 桁～16 桁)       : `1, `10101, `1111111111111111

#### (注意)

数値を中間コードに変換（文字列数値を 2 バイト型数値に変換）する際、オーバーフローが発生した場合は overflow エラーとなります。

例①：

```
?32768
Overflow
```

評価・演算においてオーバーフローが発生した場合はエラーとはなりません。

例②：

```
?32767+1
-32768
```

例①、例②は一見、同じような記述ですが、結果に差異が発生することをご理解下さい。

## 4.7 文字・文字列

### □ 文字・文字列

「豊四季タイニーBASIC for micro:bit」では半角英数字の文字列を扱うことが出来ます。

文字列の表記は次の通りです。

“文字列”       :     ダブルクォーテーションで囲みます。  
                   文字列は 0～255 文字まで指定可能です。

例:

```
?PRINT "Hello"
Hello
OK
```

## 4.8 文字列利用における制約

「豊四季タイニーBASIC for micro:bit」では制約付きで文字列の利用をサポートしています。

変数への代入も可能です。

例:

```
10 A="ABCDE"
20 PRINT A
RUN
ABCDE
OK
```

変数への文字列代入は、文字列格納アドレスを変数に代入することにより実現しています。

C 言語のポインタによる参照方式と同等です。

ただし、文字列情報を[長さ+文字列]の形式で管理しているため、文字列の参照を代入した変数の値を変更して利用することは行えません。

下記のような利用は出来ません。行った場合、意図しない動作となります。

例：

```
10 A="ABCDE"
20 A=A+1
30 PRINT A
```

また、コマンドラインでの利用は行えません。文字列参照が正しく行うことが出来ません。

例：

```
A="ABCDE"
OK
?A
```

変数に代入した文字列の操作を行う次の関数を用意しています。

- LEN() 文字列の長さの取得

```
10 A="ABCDE"
20 L=LEN(A)
RUN
OK
```

- STR\$() 変数で参照している文字列の取得、部分切り出し

```
10 A="ABCDE"
20 PRINT STR$(A)
30 PRINT STR$(A,4,1)
RUN
ABCDE
D
OK
```

- ASC() 指定位置の文字コードの取得

```
10 A="ABCDE"
20 C=ASC(A,4,1)
RUN
68
OK
```

## 4.9 変数・配列変数

### □ 変数

変数とは数値を格納する保存領域です。数値と同様に式に利用できます。

変数には、通常の変数の他に**配列変数**があります

変数に格納する数値は 16 ビット幅、有効範囲は-32768～32767 となります。

**変数**は数値型のみ利用可能ですが、文字列の格納アドレスを代入することにより、文字列の参照を行うことが出来ます。

変数の表記

通常の変数：変数名は英字 A～Z の 1 文字、または英字+数字(0～6)の 2 文字の利用が可能です。

例:

```
10 A0=200
20 A1=300
30 A=A0+A1
40 S="Hello"
```

**配列変数**：@(添え字)の形式で添え字は数値で 0～99(@ (0)～@ (99)) まで利用可能

添え字の数値には式、変数等の利用が可能

例:

```

10 A=5
20 @(0)=30/5,
30 @(A+1)=5

```

代入式において、指定した添え字を起点として連続代入が可能

例:

```

10 @(10)=100,200,300,400,500

```

## 4.10 演算子

### □ 演算子

数値演算で利用できる演算子を示します。

記述例の A,B には数値、変数、配列変数、関数、カッコで囲んだ式が利用できます。

#### 算術演算子

演算子	説明	記述例
+	足し算	A+B    A と B を足す
-	引き算	A-B    A から B を引く
*	掛け算	A*B    A と B の積
/	割り算	A/B    A を B で割る
%	剰余算	A%B    A を B で割った余り

#### ビット演算子

演算子	説明	記述例
&	ビット毎の AND 演算	A&B    A と B のビット毎の AND
	ビット毎の OR 演算	A B    A と B のビット毎の OR
>>	ビット右シフト演算	A>>B    A を右に B ビットシフト
<<	ビット左シフト演算	A<<B    A を左に B ビットシフト
~	ビット毎の反転	~A    A の各ビットを反転
^	ビット毎の XOR	A^B    A と B の XOR

比較演算、論理演算の論理反転は、0 が偽、0 以外が真となります。

0 以外が真となりますので、1、-1 はとも真となります。

#### 比較演算子

演算子	説明	記述例
=	等しいかを判定	A=B    A と B が等しければ真, 異なれば偽
!=	異なるかを判定	A!=B    A と B が異なれば真, 等しければ偽
<>		
<	小さいかを判定	A<B    A が B 未満であれば真、そうでなければ偽
<=	小さいまたは等しいかを判定	A<=B    A が B 以下であれば真、そうでなければ偽
>	大きいかを判定	A>B    A が B より大きければ真、そうでなければ偽
>=	大きいまたは等しいかを判定	A>=B    A が B 以上であれば真、そうでなければ偽

#### 論理演算子

演算子	説明	記述例
AND	論理積	A AND B    A,B が真なら真、でなければ偽
OR	論理和	A OR B    A,B どちらかが真なら真、でなければ偽
!	否定	!A    A が真なら偽、偽なら真

### □ 演算子の優先順序

演算子の優先度を下記に示します。優先度の数値が小さいほど優先度が高くなります。

計算結果が意図した結果にならない場合は、優先度の確認、括弧をつけて優先度を上げる等の対応を行って下さい。

## 演算子の優先度

優先度	演算子
1	括弧で囲った式
2	!, ~
3	*, / , % , & ,   , << , >>, ^
4	+, - ,
5	=, <> , != , > , >= , < , <= , AND , OR

比較演算子と論理演算は優先度が同レベルです。比較演算子の演算を先に行う場合は、括弧を使って優先度を上げて下さい。

例：

```
?1<5 and 2>3
0
?(1<5) and (2>3)
1
```

## 4.11 式

## □ 式

式とは 1、1+1、A、A+1、ABS(-1) などの演算・値の評価を伴う記述をいいます。

式はプログラム実行にて計算・評価が行われ、1つの整数値の値として振る舞います。

変数への値の代入、コマンド、条件判定(IF 文)、関数に引数に式が用いられた場合は、評価後の値がコマンドおよび関数に渡されます。

式の利用：

変数への値の代入(代入命令 LET は省略可能)

```
LET A=(1+10)*10/2
A=5*5+3*2
@(I+J) = I*J
```

コマンド・関数の引数

```
LOCATE X+I*2, J:PRINT "*"
OUT PC13, I>J
A=EEPREAD(I+J+1)/2
```

## 4.12 定数

「豊四季タイニーBASIC for micro:bit」では次の定数が利用出来ます。

定数はコマンドの引数や式の中で数値関数と同等に利用出来ます。

## □ 1 ビット入出力値

HIGH, LOW

HIGH は 1, LOW は 0 が割り当てられています。

各コマンドの引数、IF 文、式にて利用出来ます。

## □ 仮想メモリ領域先頭アドレス定数

VRAM,VAR,ARRAY,PRG,MEM,FNT,GRAM

データ領域を参照するための定数です。各定数の詳細は次の通りです。

定数名	値	領域サイズ	用途
VRAM	\$0000	3,840	スクリーン表示用メモリ (CW×CH)
VAR	\$1000	440	変数領域 (A~Z, A0~Z6) + 4 バイト
ARRAY	\$1240	200	配列変数領域(@ (0) ~ @ (99) )
PRG	\$1340	4,096	プログラム領域
MEM	\$2340	1,024	ユーザーワーク領域
FNT	\$2740	1,283	フォント 256 文字
GRAM	\$2F40	27	LED マトリックス表示用メモリ

定数値のアドレスは仮想的なアドレスです。実アドレスとは異なります。

PEEK、POKE、I2CW、I2CR、BITMAP、LDBMP のメモリ領域を利用するコマンドで利用できます。

定数の利用例：仮想メモリ領域先頭アドレス定数を表示する

```
10 '仮想メモリ' リス
20 PRINT "仮想メモリ' リス"
30 PRINT "VRAM ",HEX$(VRAM,4)
40 PRINT "VAR ",HEX$(VAR,4)
50 PRINT "ARRAY:",HEX$(ARRAY,4)
60 PRINT "PRG :",HEX$(PRG,4)
70 PRINT "MEM :",HEX$(MEM,4)
80 PRINT "FNT :",HEX$(FNT,4)
90 PRINT "GRAM :",HEX$(GRAM,4)
```

実行結果

```
run
仮想メモリ' リス
VRAM 0000
VAR 1900
ARRAY:1AA0
PRG :1BA0
MEM :2BA0
FNT :2FA0
GRAM :37A0
OK
```

## □ 画面表示の定数

画面サイズを参照する定数です。

CW : キャラクタ画面の横文字数 (デフォルト 80)  
 CH : キャラクタ画面の縦行数 (デフォルト 24)  
 GW : LED マトリックスの横ドット数 (5)  
 GH : LED マトリックスの縦ドット数 (5)

## □ 方向を示す定数

GSCROLL、MSG コマンドでスクロール方向を指定に利用出来ます。

UP : 値 0 上方向  
 DOWN: 値 1 下方向  
 RIGHT: 値 2 右方向  
 LEFT : 値 3 左方向  
 TOP : 値 4 上面(スクロールなし)

## □ ピン番号定数

GPIO、OUT、IN、ANA、SHIFTOUT、SHIFTIN コマンド・関数のピン番号の指定に利用します。

各ピン番号定数に実際のピン番号 0 ~ 32 が割り当てられています。

PN0 ~ PN32

## □ GPIO モード設定定数

GPIO コマンドのモード設定を行うための定数です。

OUTPUT, INPUT\_PU, INPUT\_PD, INPUT\_FL



#### □ ビット方向定数

LSB, MSB

ビット送信等で上位、下位を指定するための定数です。

SHIFTIN,SHFITOUT コマンドで利用します。

#### □ OFF/ON 定数

OFF : 値 0

ON : 値 1

## 5. 各制御文の詳細

### 5.1 GOTO 指定行へのジャンプ（制御命令）

#### □ 書式

GOTO 行番号

GOTO "ラベル"

#### □ 説明

プログラムの実行を行番号、"ラベル"で指定した行にジャンプします。

行番号には数値、式（変数、関数を含む）が利用可能です。

ラベルを指定した場合、行先頭に同じラベルがある行にジャンプします。

ラベルはダブルクォテーション( ")で囲って指定します。

```
10 I=0
20 "LOOP"
30 PRINT "@"
40 I=I+1:IF I=5 GOTO 60
50 GOTO "LOOP"
60 END
```

上記の例では、40 行の GOTO で 60 行にジャンプ、50 行のラベル指定で 20 行にジャンプしています。

ラベルを使うとプログラムの流れがわかりやすくなります。

行の先頭に無いラベルは、ジャンプ先としての利用は出来ません。

```
10 GOTO "hoge"
20 END
30 ? "Hello": "hoge": ? "test"
run
Undefined line number or label in 10
10 GOTO "hoge"
OK
```

行番号には式や変数の指定が可能ですが、RENUM コマンドによる行番号付け替えに対応出来ない場合があります。

①GOTO N\*100+500

②GOTO 500+N\*100+500

①では RENUM コマンドは何もしません。②は 500 を番号とみなして更新します。

ただし①②とも計算結果が正しい行番号となることは保証出来ません。ご注意下さい。

#### □ エラーメッセージ

Undefined line number or label	: 指定した行、ラベルが存在しない
Overflow	: 指定した数値が-32768 ~ 32767 を超えている
Syntax error	: 文法エラー、書式と異なる利用を行った

#### □ 利用例

ラベルを使ったループ処理

```
100 GOSUB "SUB01"
110 GOSUB "SUB02"
120 N=0
130 "LOOP"
140 PRINT "N=";N
150 N=N+1:IF N<5 GOTO "LOOP"
160 END
170 "SUB01"
180 PRINT "SUB01"
190 RETURN
200 "SUB02"
210 PRINT "SUB02"
220 RETURN
```

## 5.2 GOSUB サブルーチンの呼び出し（制御命令）

## □ 書式

GOSUB 行番号  
GOSUB "ラベル"

## □ 説明

プログラムの実行を一旦、行番号またはラベルで指定した行に移ります。  
移った先で RETURN 命令により、呼び出した GOSUB 命令の次の位置に戻ります。  
行番号には数値、式（変数、関数を含む）が利用できます。  
ラベルを指定した場合は、行先頭に同じラベルがある行に移ります。  
ラベルはダブルクォーテーション(“)で囲って指定します。

```
10 GOSUB "PRN_LOGO"
20 GOSUB "PRN_DATE"
30 GOSUB 200
40 END
50 "PRN_LOGO"
60 PRINT "Tiny BASIC for microbit"
70 RETURN
80 "PRN_DATE"
90 PRINT "Edition V0.05 2018/01/16"
100 RETURN
200 PRINT "Ready"
210 RETURN
```

上記の例では、10 行、20 行でラベルを指定してサブルーチンを呼び出しています。  
30 行では、行番号を指定してサブルーチンを呼び出しています。  
ラベルを使うことで、プログラムの実行がわかりやすくなります。  
行の先頭に無いラベルは、GOSUB 文の呼び出し先としての参照はされません。

```
10 GOSUB "hoge"
20 END
30 ? "Hello":"hoge":?"test":RETURN
run
Undefined line number or label in 10
10 GOSUB "hoge"
OK
```

行番号には式や変数の指定が可能です。RENUM コマンドによる行番号付け替えに対応出来ない場合があります。

①GOSUB N\*100+500

②GOSUB 500+N\*100+500

①では RENUM コマンドは何もしません。②は 500 を行番号とみなして更新します。  
ただし①②とも計算結果が正しい行番号となることは保証出来ません。ご注意ください。

## □ エラーメッセージ

Undefined line number or label	: 指定した行、ラベルが存在しない
Overflow	: 指定した数値が-32768 ~ 32767 を超えている
GOSUB too many nested	: GOSUB のネスト数が規定(=10)を超えた
Syntax error	: 文法エラー、書式と異なる利用を行った

## □ 利用例

ラベルを使ったサブルーチンの呼び出し例

```
100 GOSUB "SUB01"
110 GOSUB "SUB02"
120 N=0
130 "LOOP"
140 PRINT "N=";N
150 N=N+1:IF N<5 GOTO "LOOP"
160 END
170 "SUB01":PRINT "SUB01":RETURN
200 "SUB02":PRINT "SUB02":RETURN
```

## 5.3 RETURN GOSUB 呼び出し元への復帰（制御命令）

## □ 書式

RETURN

## □ 説明

直前に呼び出された GOSUB の次の処理に復帰します。

詳細は「6.2 GOSUB サブルーチンの呼び出し（制御命令）」を参照して下さい。

## □ エラーメッセージ

Undefined line number or label	: 指定した行、ラベルが存在しない
Overflow	: 指定した数値が-32768 ~ 32767 を超えている
RETURN stack underflow	: GOSUB の呼び出しがないのに RETURN を実行
Syntax error	: 文法エラー、書式と異なる利用を行った

## □ 利用例

ラベルを使ったサブルーチンの呼び出し例

```

100 GOSUB "SUB01"
110 GOSUB "SUB02"
120 N=0
130 "LOOP"
140 PRINT "N=";N
150 N=N+1:IF N<5 GOTO "LOOP"
160 END
170 "SUB01"
180 PRINT "SUB01"
190 RETURN
200 "SUB02"
210 PRINT "SUB02"
220 RETURN

```

## 5.4 IF 条件判定（制御命令）

## □ 書式

IF 条件式 実行文 1

IF 条件式 実行文 1 ELSE 実行文 2

## □ 説明

条件式の真偽判定を行い、真の場合は実行文 1 を実行します。

真の場合に ELSE がある場合、ELSE 文以降の命令文は実行しません。

偽の場合、ELSE がある場合、実行文 2 を実行します。なければ次の行にスキップします。

真偽は次の条件となります。

真：評価した値が 0 以外である

偽：評価した値が 0 である

真偽判定を行う条件式には定数、数値、関数を含む計算式が記述できます。

式の例：

```
IF A > 5 B=B-1
IF A+1 PRINT A
IF A & 1 TONE 440,200
IF INKEY() = ASC("A") GOTO 100
```

定数の例

```
IF HIGH GOTO 100
```

数値の例

```
IF 123 PRINT "123"
```

関数の例

```
IF RND(1) X=X+1
```

実行文には IF、GOTO、GOSUB 等の制御命令を使うことも可能です。

**（注意）ELSE 利用の制約**(ぶら下がり ELSE に関する補足)

IF 文をネストして利用した場合、ELSE 文は直前の IF 文に対応します。

例：IF A=1 IF B=1 ? "A,B=1" ELSE ? "A=1,B<>1"

上記の ELSE は 2 番目の IF 文に対応します。

この「ELSE 文は直前の IF 文に対応」の条件で、ELSE 文に IF 文をネスト出来ます。

例: IF A=1 IF B=1 ? "A,B=1" ELSE IF B=2 ? "A=1,B=2" ELSE IF B=3 ? "A=1,B=3"

上記では、A=1,B=1,2,3 の条件に対して対応する表示メッセージを表示する例です。

## □ エラーメッセージ

IF without condition

: 条件式が定義されていない

Overflow

: 指定した数値が -32768 ~ 32767 を超えている

## □ 利用例

y を押したら、“Y key” を表示して終了、その他のキーなら“End” を表示して終了する

何も押されていないならば 10 行から 30 行をループする。

```
10 A=INKEY()
20 IF A=ASC("y") PRINT "Y key":END ELSE IF A<>0 PRINT "End":END
30 GOTO 10
```

## 5.5 FOR TO ~ NEXT 繰り返し実行（制御命令）

## □ 書式

FOR 変数=初期値 TO 最終値 実行文(複数行可能) NEXT 変数

FOR 変数=初期値 TO 最終値 STEP 増分 繰り返し実行文(複数行可能) NEXT 変数

## □ 説明

変数を初期値から最終値まで増やし、FOR から NEXT の間の実行文を繰り返し実行します。

変数が最終値に達した時点で繰り返しを止め、NEXT の次の命令の実行を行います。

STEP にて増分を指定しない場合、増分は 1 となります。

STEP を用いた場合は、マイナス値を含め任意の増分の指定が可能です。

例：

```
10 PRINT "start."
20 FOR I=0 TO 5 STEP 2
30 PRINT I
40 NEXT I
50 PRINT "done."
```

実行結果

```
start
0
2
4
done.
ok
```

上記の例では I を 0 から 5 まで、2 ずつ増加して 30 行の命令を繰り返し実行します。

I が 4 の時、増分 2 を足すと終了値 5 を超えた 6 となるので繰り返しを終了し、50 行を実行します。

FOR 文はネストも可能です(利用例を参照)。

FOR~NEXT の繰り返しは、利用している変数が繰り返し条件を満たさなくなった時点で繰り返しを終了します。

最初の例に“35 I=6”を追加し、ループ内で条件を満たさなくすることでループを抜けることが出来ます。

```
10 PRINT "start."
20 FOR I=0 TO 5 STEP 2
30 PRINT I
35 I=6
40 NEXT I
50 PRINT "done."
```

実行結果

```
start
0
done.
ok
```

## □ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
FOR without variable	: FOR 文で変数を指定していない
FOR without TO	: FOR 文で TO を指定していない
NEXT without counter	: NEXT に対応する FOR 文が無い
FOR too many nested	: FOR 文のネスト数が規定数(=10)を超えた
Overflow	: 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

画面の指定位置に\*を表示する（FOR 文のネストの例）

```
10 CLS
20 FOR Y=5 TO 10
30 FOR X=5 TO 10
40 LOCATE X,Y:PRINT "*"
50 NEXT X
60 NEXT Y
```

## 5.6 END プログラムの終了（制御命令）

### □ 書式

END

### □ 説明

プログラムの実行を終了します。

### □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

### □ 利用例

特定の条件でプログラムを終了する

```
10 I=0
20 PRINT I
30 IF I=5 END
40 GOTO 20
```

## 6. 各コマンド・関数の詳細

### 6.1 RUN プログラムの実行（システムコマンド）

#### □ 書式

RUN

#### □ 引数

なし

#### □ 説明

プログラムの実行を行います。

実行中のプログラムは [ESC]キーまたは、[CTRL+C]キーにて強制終了することが出来ます。

シリアルコンソール上で[ESC]キーにて中断する場合は、[ESC]キーを2回押す必要があります。

プログラムの実行中は、カーソルが非表示となります。

プログラムの実行中は実行すべき次の行が無い場合、実行を終了します。

また END コマンドにより実行を終了します。

プログラムの実行中にエラーが発生した場合は実行を終了します。

**（注意）** WAIT 命令で長い時間待ちを行っている場合は、時間待ちが終了するまで、[ESC]キー、[CTRL+C]キーによる強制終了を行うことが出来ません。

#### □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った、引数に数値以外を指定した

#### □ 利用例

##### プログラムを実行する

```
LIST
10 I=0
20 PRINT "*";
30 I=I+1:IF I<10 GOTO 20
40 PRINT
50 END
OK

RUN
*****
OK
```



## 6.2 RENUM 行番号の振り直し（システムコマンド）

## □ 書式

RENUM

RENUM 開始行番号

RENUM 開始行番号,増分

## □ 引数

開始番号 : 振り直しをする新しい行番号の開始番号 (0 ~ 32767)

増分 : 行番号の増分 (1 ~ 32767)

## □ 説明

プログラム全体の行番号を指定した条件にて振り直します。

引数を省略した場合は、行番号を 10 行から 10 間隔で振り直します。

開始番号のみ指定した場合、指定した開始番号から 10 間隔で振り直します。

開始番号と増分を指定した場合、指定した開始番号から指定した増分で振り直します。

振り直しにおいて、GOTO 文、GOSUB 文のとび先の行番号も更新されます。

(注意) GOTO,GOSUB に存在しない行番号を指定している場合、更新は行われません。  
また、行番号に計算式を利用している場合、正しい更新が行われない場合があります。  
GOTO 100+N\*10  
の記載の場合、先頭の数値 100 を行番号とみなして更新します。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った、引数に数値以外を指定した  
Illegal value : 振り直しをする新しい行番号が有効範囲を超えている  
Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

行番号を 100 から 10 間隔で振り直す

```
LIST
10 I=0
20 PRINT "*";
30 I=I+1:IF I<10 GOTO 20
40 PRINT
50 END
OK
RENUM 100,10
OK
LIST
100 I=0
110 PRINT "*";
120 I=I+1:IF I<10 GOTO 110
130 PRINT
140 END
```

## 6.3 DELETE プログラムの指定行の削除（システムコマンド）

## □ 書式

DELETE 行番号

DELETE 先頭行番号,末尾行番号

## □ 引数

行番号 : 削除対象の行番号 1～32767

先頭番号 : 削除対象範囲の先頭番号 1～32767

末尾番号 : 削除対象範囲の末尾番号 1～32767

## □ 説明

プログラム内の指定した行、指定した範囲（先頭行番号、末尾行番号）の行を削除します。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った、引数に数値以外を指定した

Overflow : 指定した数値が-32768 ～ 32767 を超えている

Illegal value : 先頭番号と末尾番号の指定に矛盾がある

## □ 利用例

プログラム内の 20 行から 50 行の範囲を削除する

```

10 ? "AAAAAA"
20 ? "BBBBBB"
30 ? "CCCCCC"
40 ? "DDDDDD"
50 ? "EEEEEE"
60 ? "FFFFFF"
70 ? "GGGGGG"

```

DELETE 20,50

OK

LIST

```

10 ? "AAAAAA"
60 ? "FFFFFF"
70 ? "GGGGGG"

```

OK

## 6.4 WIDTH シリアルターミナルの画面サイズの設定

## □ 書式

WIDTH 横文字数, 縦文字数

## □ 引数

横文字数：ターミナル画面上のスクリーンの横文字数 16 ～ 128（デフォルト 80）

縦文字数：ターミナル画面上のスクリーンの縦文字数 10 ～ 30（デフォルト 24）

## □ 説明

シリアルターミナル画面上のスクリーンサイズの設定を行います。

この設定にてスクリーンエディタの表示文字数、スクロール範囲の変更を行うことができます。

スクリーンサイズは、最小サイズ 16 桁×10 行 ～ 最大 128 桁×45 行 となります。

## □ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った、引数に数値以外を指定した
Illegal value	: 指定した引数の値が範囲外である
Overflow	: 指定した数値が-32768 ～ 32767 を超えている

## □ 利用例

スクリーンサイズを 80 桁×30 行に設定する

```
WIDTH 80,30
```

## 6.5 LIST プログラムリストの表示

## □ 書式

LIST

LIST 表示開始行番号

LIST 表示開始行番号, 表示終了行番号

## □ 引数

表示開始行番号：表示を開始する行番号（0 ～ 32767）

表示終了行番号：表示を終了する行番号（0 ～ 32767）

## □ 説明

プログラムリストの表示を行います。

引数を指定しない場合は、全てのプログラムを表示します。

表示開始番号を指定した場合は、その番号以降のプログラムリストを表示します。

表示開始番号、表示終了番号を指定した場合は、その範囲のプログラムリストを表示します。

表示したプログラムリストは、カーソルを移動して編集することが出来ます。

編集後は必ず[ENTER]キーを押して入力確定を行って下さい。

## □ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った、引数に数値以外を指定した
Overflow	: 指定した数値が-32768 ～ 32767 を超えている

## □ 利用例

```
LIST
10 I=0
20 PRINT "*";
30 I=I+1:IF I<10 GOTO 20
40 PRINT
50 END
OK

list 20,30
20 PRINT "*";
30 I=I+1:IF I<10 GOTO 20
OK
```

## 6.6 NEW プログラムの消去

## □ 書式

NEW

## □ 引数

なし

## □ 説明

プログラム領域のプログラム、変数、配列変数を消去します。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った、引数に数値以外を指定した  
 Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

プログラムを消去する

```
LIST
10 I=0
20 PRINT "*";
30 I=I+1:IF I<10 GOTO 20
40 PRINT
50 END
OK

run
*****
OK
?I
10
OK
NEW
OK
LIST
OK
?I
0
OK
```

## 6.7 LOAD 内部フラッシュメモリからプログラムを読み

## □ 書式

LOAD

LOAD プログラム番号

## □ 引数

プログラム番号 : 0 ~ 7 内部フラッシュメモリのプログラム番号

## □ 説明

内部フラッシュメモリからプログラムを読み込みます。

引数を省略した場合は、内部フラッシュメモリのプログラム番号を読み込みます。

引数にプログラム番号を指定した場合は、指定したプログラム番号を読み込みます。

プログラム番号 0~7 で指定します。読入の際、変数領域は初期化されます。

## □ エラーメッセージ

Syntax error : 書式と異なる利用を行った、プログラム番号に変数、式を指定した  
 Illegal value : プログラム番号の指定が 0~9 の範囲外である  
 Program not found : 指定したプログラム番号にプログラムが保存されていない  
 Overflow : 指定した数値が-32767 ~ 32767 を超えている

## □ 利用例

プログラム番号 1 を読み込む

LOAD 1
OK

## 6.8 SAVE 内部フラッシュメモリへのプログラム保存

## □ 書式

SAVE

SAVE プログラム番号

## □ 引数

プログラム番号 : 0 ~ 7

## □ 説明

プログラムをマイコン内のフラッシュメモリに保存します。

引数の省略、数値を指定した場合は内部フラッシュメモリに保存します。

プログラムは最大で 8 つ保存可能です。保存先はプログラム番号 0~7 で指定します。

引数の省略した場合、プログラム番号 0 に保存します。

## □ エラーメッセージ

Syntax error	: 書式と異なる利用を行った、プログラム番号に変数、式を指定した
Illegal value	: プログラム番号の指定が 0~7 の範囲外である
Overflow	: 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

プログラム番号 1 にプログラムを保存する

SAVE 1
OK

## 6.9 FILES 内部フラッシュメモリ内保存プログラムの一覧表示

## □ 書式

FILES

FILES 開始プログラム番号

FILES 開始プログラム番号,終了プログラム番号

## □ 引数

開始プログラム番号 : 表示開始プログラム番号 0～7

終了プログラム番号 : 表示終了プログラム番号 0～7

## □ 説明

マイコン内のフラッシュメモリに保存されているプログラムの一覧を表示します。

引数を指定しない場合は、内部フラッシュメモリ内のプログラム番号 0～7 の先頭行をリスト表示します。

開始プログラム番号,終了プログラム番号を指定した場合、その範囲のリストを表示します。

プログラム番号にプログラムが保存されていない場合は(none)と表示されます。

プログラム先頭行にコメントをつけると、一覧表示でのプログラムの内容が分かり易くなります。

例:

```
files
0:'Edit bitmap
1:'RTC TEST
2:(none)
3:(none)
4:(none)
5:(none)
6:(none)
7:1'LED Blink
OK
```

(注意) 豊四季タイニーBASIC (ファームウェア) の新規利用または更新を行った直後は、フラッシュメモリ上の既存データの内容の不整合により正しく表示できない場合があります。その場合は、ERASE コマンドにてフラッシュメモリ上のプログラムの消去を行って下さい。

## □ エラーメッセージ

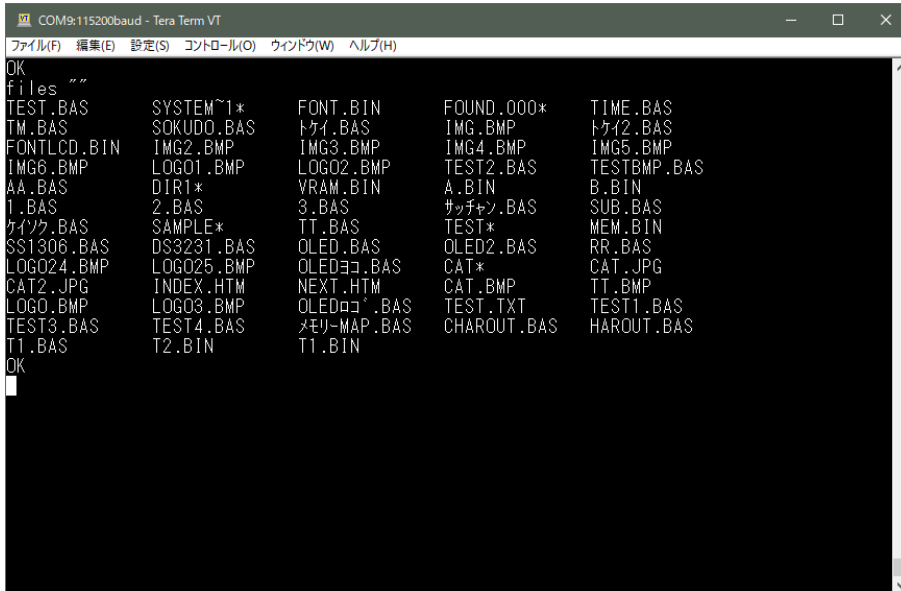
Syntax error : 書式と異なる利用を行った、プログラム番号に変数、式を指定した  
 Illegal value : 指定した開始プログラム番号,終了プログラム番号の値が正しくない。  
 Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

シリアルコンソール画面でSDカード内のファイル一覧を表示する。

(補足) 表示する横の列数は、スクリーンの横のサイズに応じて調整されます。





## 6.10 REM コメント

## □ 書式

REM コメント

‘ コメント

## □ 引数

コメント : 任意の文字列

## □ 説明

プログラムに説明等の記載を行います。

‘(シングルクォート)は REM の省略形です。

REM および以降の文字以降はすべてコメントとみなし、プログラムとして実行されません。

プログラムの先頭行にコメントを付けた場合は、FILES コマンドで保存プログラム一覧を表示時に、各プログラムの見出しとなります。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

## □ 利用例

コメントの記述例

```
LIST
1 REM Print starts
10 I=0:'initialize value
20 PRINT "*";
30 I=I+1:IF I<10 GOTO 20
40 PRINT
50 END
```

## 6.11 LET 変数に値を代入

## □ 書式

LET 変数=値

LET 配列変数=値,値,値,値,...

変数=値

配列変数=値,値,値,値,...

## □ 引数

変数 : 変数、または配列変数

配列変数 : 配列変数 @(値)

配列の値(添え字)には式、数値、変数、配列変数、数値定数の利用が可能

値 : 式、数値、変数、配列変数、数値定数

## □ 説明

変数に値を代入します。

値は式、数値、定数、変数、配列変数等を評価した整数値です。

配列変数への代入は、複数の値を指定した連続代入が可能です。指定した添え字を起点に順番に代入します。

LET @(3)=1,2,3,4,5

上記の記述は、

LET @(3)=1: LET @(4)=2: LET @(5)=3: LET @(5)=4: LET @(5)=5

と同じです。

LET は省略可能です。次の2は同じ結果となります。

LET A=A+1

A=A+1

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

```

LIST
1 'Print starts
10 LET I=0:'initialize value
20 PRINT "*";
30 LET I=I+1:IF I<10 GOTO 20
40 PRINT
50 END

```

## 6.12 CLV 変数領域の初期化

## □ 書式

CLV

## □ 引数

なし

## □ 説明

変数領域（変数、配列）の初期化を行います。

変数 A～Z、配列@(0)～@(99)はすべて 0 に初期化されます。

LRUN コマンドにてプログラムをロードして実行する場合、変数領域は初期化されません。

これにより呼び出し元のプログラムの変数をひ引き継ぐことが出来ます。

呼び出されたプログラムにて明示的に変数領域の初期化を行う場合は、CLV コマンドを使って下さい。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Overflow : 指定した数値が-32768 ～ 32767 を超えている

## □ 利用例

```
a=123
OK
?a
123
OK
CLV
OK
?a
OK
```

## 6.13 LRUN 指定プログラム番号の実行

## □ 書式

LRUN プログラム番号

LRUN プログラム番号,行番号

LRUN プログラム番号,"ラベル"

## □ 引数

プログラム番号 : 実行するプログラムの番号 0 ~ 7

行番号 : プログラムの実行開始行番号 1~32767

"ラベル" : プログラムの実行開始行のラベル指定

## □ 説明

プログラム番号またはファイル名で指定したプログラムを実行します。

第 1 引数に数値を指定した場合は、内部フラッシュメモリから指定したプログラム番号のプログラムを読み込んで実行します。第 2 引数に行番号またはラベルの指定がある場合は、その行からプログラムの実行を開始します。

(注意) LRUN にてプログラムを実行した場合、変数領域の初期化は行われません。直前のプログラムが設定した変数の値を引き継ぎます。初期化が必要な場合は呼び出されたプログラム中で CLV コマンドを実行して下さい。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った  
 Illegal value : プログラム番号の指定が 0~7 の範囲外である  
 Program not found : 指定したプログラム番号にプログラムが保存されていない  
 Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

入力したプログラム番号を実行する。

```
10 INPUT N
20 LRUN N
```

## 6.14 EXPORT 内部フラッシュメモリの内容のエクスポート

## □ 書式

EXPORT

EXPORT 対象番号

EXPORT 開始番号,終了番号

## □ 引数

対象番号 : エクスポートするプログラム保存番号 0 ~ 7

開始番号,終了番号 : エクスポートするプログラム保存番号の範囲 0 ~ 7

## □ 説明

マイコン内のフラッシュメモリに保存されているプログラム(プログラム番号 0~7)を出力表示します。

ターミナルソフト等を利用することにより、出力表示されたプログラムリストをコピーすることで、内部保存されたプログラムのバックアップを行うことができます。

引数に何も指定しない場合は、0 ~ 7 までの全てのプログラムを画面に出力します。

対象番号を指定した場合は、該当するプログラム番号の内容のみ出力します。

開始番号と終了番号を指定した場合はその範囲のプログラムを出力します。

出力形式は次の形式となります。

```
NEW
1 'トイ
10 MATRIX ON
20 SETDATE 2018,1,16,12,0,0
30 IF !IN(BTNA) GOSUB "@ShowTime"
40 WAIT 200
50 GOTO 30
60 "@ShowTime"
70 GETTIME T1,T2,T3
80 MSG LEFT,80,#-2,T1;":";T2;":";T3;" "
90 RETURN
SAVE 1
```

プログラム番号毎のプログラムリストの先頭に“NEW”コマンド、末尾に“SAVE プログラム番号”が付加されます。

ターミナルソフトにペーストすることにより、フラッシュメモリ内に再登録することができます。

## □ エラーメッセージ

Syntax error : 書式と異なる利用を行った、プログラム番号に変数、式を指定した

Illegal value : 指定した引数が有効範囲を超えている

Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

プログラム番号 1,2 に保存されているプログラムのエクスポート

```
EXPORT 1,2
NEW
10 'oscilloscope
20 CLS
30 GPIO PB01,ANALOG
40 "LOOP"
50 R=ANA(PB01)
60 Q=R/20
70 LOCATE 0,0:?" "
80 PSET GW-1,GH-Q-4,1
90 WAIT 50
100 GSCROLL 8,208,3
110 GOTO "LOOP"
SAVE 1

NEW
10 CLS
20 RECT 0,0,GW,GH,1,1
30 RESETTICK
40 FOR I=0 TO 215
50 FOR X=0 TO 200 STEP 16
60 GSCROLL X,0,8,GH,0
70 GSCROLL X+8,0,8,GH,1
80 NEXT X
100 NEXT I
110 ?TICK()
120 GOTO 120
SAVE 2

OK
```

## 6.15 ABS 絶対値の取得（数値関数）

## □ 書式

ABS(値)

## □ 引数

値 : -32767 ~ 32767

式、変数、配列変数、数値、数値定数の指定が可能

## □ 戻り値

指定した値の絶対値

## □ 説明

指定した値の絶対値を返します。

（注意）-32768 の絶対値 32768 はオーバーフローとなるため、-32768 を指定した場合はオーバーフローエラーとなります。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った  
 Overflow : 指定した数値が-32767 ~ 32767 を超えている  
 '(' or ')' expected : '(' または ')' が無い

## □ 利用例

変数の値の絶対値を表示する

```
10 FOR I=-3 TO 3
20 PRINT ABS(I)
30 NEXT I
```

RUN

```
3
2
1
0
2
3
OK
```



## 6.16 ASC 文字から文字コードへの変換（数値関数）

## □ 書式

ASC(文字列)

ASC(文字列,文字位置)

ASC(変数)

ASC(変数,文字位置)

## □ 引数

文字列： “文字列”  
 “ABC” の形式とし、ダブルクォーテーション文字を囲みます

文字位置： 1～32767  
 変換対象となる左からの文字位置を指定します。

変数： 文字列参照している変数または配列変数

## □ 戻り値

指定文字に対応する文字コード（0 ～ 255）

## □ 説明

指定した文字に対応する文字コードを返します。

文字列のみを指定した場合、先頭の文字コードを返します。

```
10 ?ASC("ABCD")
RUN
65
OK
```

文字位置を指定した場合、指定した位置の文字コードを返します。

```
10 ?ASC("ABCD",3)
RUN
67
OK
```

変数を指定した場合、変数が参照している文字列のコードを返します。

```
10 A="ABCDEF"
20 ?ASC(A)
30 ?ASC(A,3)
RUN
65
67
OK
```

**（注意）** 変数の文字列参照はプログラム中にのみ有効です。コマンドラインでは文字列参照を正しく行うことが出来ません。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Illegal value : 文字指定位置が不当

'(' or ')' expected : '(' または ')'がない

Overflow : 指定した数値が-32768 ～ 32767 を超えている

## □ 利用例

アルファベット A～Z を表示する

```
10 C=ASC("A")
20 FOR I=0 TO 25
30 PRINT CHR$(C+I);
40 NEXT I
50 PRINT
```

```
run
ABCDEFGHIJKLMNOPQRSTUVWXYZ
OK
```

## 6.17 FREE プログラム領域の残りバイト数の取得（数値関数）

## □ 書式

FREE()

## □ 引数

なし

## □ 戻り値

プログラム領域の残りバイト数 0 ～ 4095

## □ 説明

プログラム領域の残りバイト数を返します。

作成したプログラムサイズを確認する場合は、下記の記述にて行うことが可能です。

```
PRINT 4095-FREE()
```

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

'(' or ')' expected : 引数部の記述が正しくない

## □ 利用例

プログラムサイズを調べる

```
PRINT 4095-FREE()
287
OK
```

## 6.18 INKEY キー入力の読み取り（数値関数）

## □ 書式

INKEY()

## □ 引数

なし

## □ 戻り値

押したキーの文字コード 0 ～ 255

キーが押されていない場合は 0 を返します

## □ 説明

キーボード上の押しているキーのコードを取得します。キーが押されていない場合は、0 を返します。

押したキーが返すコードについては「エラー! 参照元が見つかりません。エラー! 参照元が見つかりません。」を参照下さい。

**（注意）** [ESC]、[CTRL-C]はプログラム中断用のため、キー入力の読み取りは出来ません。  
 キーボードにおいて利用出来ないキーはコードの取得が出来ません。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

'(' or ')' expected : '(' または ')'がない

## □ 利用例

入力したキーのコードを調べる

```
10 CLS
20 I= INKEY()
30 IF I>0 LOCATE 0,0:?"#3,I
40 GOTO 20
```

## 6.19 RND 乱数の発生（数値関数）

## □ 書式

RND(値)

## □ 引数

値 : 0 ~ 32767

式、変数、配列変数、数値、数値定数の指定が可能

## □ 戻り値

0 から指定した値未満の乱数

## □ 説明

0 から指定した値-1 の範囲の乱数を発生させ、その値を返します。

R=RND(10)

の場合、変数 R には 0~9 範囲の値が代入されます。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った  
 Overflow : 指定した数値が-32768 ~ 32767 を超えている  
 '(' or ')' expected : '(' または ')'が無い

## □ 利用例

ジャンケンの表示

```

10 @(0)="グー","チョキ","ハー"
20 FOR I=1 TO 5
30 R=RND(3)
40 PRINT I,":";STR$(@(R))
50 NEXT I
run
1:チョキ
2:ハー
3:チョキ
4:グー
5:ハー
OK

```

## 6.20 LEN 文字列の長さの取得（数値関数）

## □ 書式

LEN(変数)

LEN(文字列)

## □ 引数

変数 : 変数、または配列変数

文字列 : “文字列”の形式（ダブルクォーテーションで囲み）

## □ 戻り値

文字列の長さ 0 ～ 32767

## □ 説明

文字列定数、変数で参照してる文字列の文字数をカウントし、その値を返します。

```

10 ?LEN("12345678")
20 A="ABCDEF"
30 @(0)="abcdef"
40 ?LEN(A)
40?LEN(@(0))
RUN
8
6
6
OK

```

**（注意）** 文字列のサポートは限定的です。  
 他の文字列関数と組み合わせた利用はサポートしていません。  
 例) 下記のような記述は出来ません。  
 ?LEN(BIN\$(100))

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った  
 Overflow : 指定した数値が-32768 ～ 32767 を超えている  
 '(' or ')' expected : '(' または ')'が無い

## □ 利用例

文字を拡大表示する

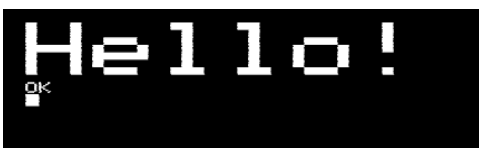
変数 A が参照している文字列の長さ LEN()関数で取得し、その長の分、文字を拡大表示します。

```

10 CLS
20 A="Hello!"
30 FOR I=1 TO LEN(A)
40 BITMAP (I-1)*30,0,FNT,ASC(A,I),6,8,5
50 NEXT I
60 LOCATE 0,5

```

実行結果



## 6.21 MAP 数値のスケール変換（数値関数）

## □ 書式

MAP(値,範囲下限, 範囲上限, 新範囲下限, 新範囲上限)

## □ 引数

値 : 変換対象の数値 -32768 ~ 32767  
 範囲下限 : 対象数値の数値の下限 -32768 ~ 32767  
 範囲上限 : 対象数値の数値の上限 -32768 ~ 32767  
 新範囲下限 : 新しい対象数値の数値の下限 -32768 ~ 32767  
 新範囲上限 : 新しい新しい対象数値の数値の下限 -32768 ~ 32767

## □ 戻り値

スケール変換後の値 -32767 ~ 32767

## □ 説明

指定した値のスケール変換を行い、その値を返します。

例：

```
A=MAP(ANA(PN0),0,1023,0,99)
```

上記の例ではアナログ入力値 0~1023 を 0 ~ 99 のレンジに変換し、その値を返します。

アナログ入力や画面座標指定、PWM パルス出力等にて MAP 関数を使うことで、簡単にレンジ変換を行うことができます。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った  
 Overflow : 指定した数値が-32768 ~ 32767 を超えている  
 '(' or ')' expected : '(' または ')'が無い

## □ 利用例

アナログ入力値 0~1023 を 0~3.3V で表示する

```
10 CLS
20 V=MAP(ANA(PN0),0,1023,0,3300)
30 LOCATE 0,0:PRINT DMP$(V,3)
40 WAIT 200
50 GOTO 20
```

## 6.22 GRADE 等級判定（数値関数）

## □ 書式

GRADE(判定対象値, 配列番号,個数)

## □ 引数

判定対象値 : 変換対象の数値 -32768 ~ 32767

配列番号 : 0 ~ 99

個数 : 1 ~ 99

## □ 戻り値

等級 0 ~ 99（配列番号=0）、範囲外 -1

## □ 説明

判定対象値を配列に格納された値を閾値として等級判定し、その値を返します。

配列に格納されている閾値は大きい順にソートされている必要があります。

判定対象値と配列データを先頭から順番に比較し、配列データよりも大きい場合はその配列データを該当閾値をして等級を返します。等級は閾値が格納されている先頭の配列を 0 とし、範囲外の場合は -1 を返します。

例：

```

10 @(0)=5120,2560,1280,640,320,160,80,40,20,10
20 INPUT "V=",V
30 G=GRADE(V,0,10)
40 ?G
50 GOTO 20

```

実行結果

```

run
V=12
9
V=10000
0
V=1000
3
V=200
5
V=500
4
V=3
-1

```

上記の例では入力値に該当する等級を返します。

10000 を入力した場合、5120 以上であるため 0 を返します。1000 を入力した場合、640 以上であるため 3 を返します。4 を入力した場合は配列に格納されている最小値が 10 であるため、範囲外と判定し-1 を返します。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Illegal value : 指定した値が有効範囲外である

Overflow : 指定した数値が-32768 ~ 32767 を超えている

'(' or ')' expected : '(' または ')'が無い

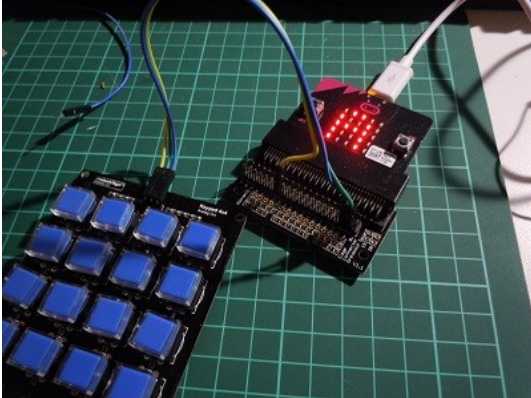


## □ 利用例

アナログ入力 4x4 キーパッドのキー入力判定を行う。

```
10 'Keypad 4x4
20 G0=-1
30 @(10)=1013,920,840,780,670,630,590,560,502,477,455,435,400,320,267,228
40 G0=G
50 G=GRADE(ANA(1),10,16)
60 IF G<>G0 WAIT 1 GOTO 40
70 IF G>=0 ?"KEY=[";G+1;"]":MSG TOP,0,CHR$(65+G)
80 GOTO 40
```

実行結果



## 6.23 CHR\$ 文字コードから文字への変換（文字列関数）

## □ 書式

CHR\$(文字コード)

CHR\$(文字コード, 文字コード, ... 文字コード)

## □ 引数

文字コード： 0～255

## □ 戻り値

指定した文字コードに対する文字

## □ 説明

指定した文字コードに対応する文字を返します。

文字コードは複数指定することが可能です。

PRINT、MSG、GPRINT の引数にて利用可能です。

範囲外の値を指定した場合は空白文字(" ")を返します。

## □ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
Illegal value	: 文字コードに 0～255 以外の数値を指定した
'(' or ')' expected	: '(' または ')' が無い
Overflow	: 指定した数値が -32768 ～ 32767 を超えている

## □ 利用例

アルファベット A～Z を表示する

```
10 C=ASC("A")
20 FOR I=0 TO 25
30 PRINT CHR$(C+I);
40 NEXT I
50 PRINT
```

```
run
ABCDEFGHIJKLMNOPQRSTUVWXYZ
OK
```

## 6.24 BIN\$ 数値から 2 進数文字列への変換（文字列関数）

## □ 書式

BIN\$(数値)

BIN\$(数値 , 桁数)

## □ 引数

数値： 変換対象の整数値( -32768 ~ 32767 )

桁数： 出力桁数( 0 ~ 16 )

## □ 戻り値

2 進数文字列(1 桁~16 桁)

## □ 説明

指定した数値を 2 進数文字列に変換します。

数値には式、変数、定数等の指定が可能です。

PRINT、MSG、GPRINT の引数にて利用可能です。

桁数を指定した場合は数値が指定した桁数に満たない場合は、0 で桁を補います。

指定した桁数を超える場合は数値の桁数を優先します。

桁数を指定しない場合は、先頭の 0 は付加されません。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

'(' or ')' expected : '(' または ')'がない

Illegal value : 桁数の値が正しくない

Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

2 進数で変数の内容を表示する

```

10 A=1234:B=1
20 PRINT BIN$(A)
30 PRINT BIN$(B)
40 PRINT BIN$(A,4)
50 PRINT BIN$(B,4)

```

```

run
10011010010
1
10011010010
0001
OK

```

## 6.25 HEX\$ 数値から 16 進数文字列への変換（文字列関数）

## □ 書式

HEX\$(数値)

HEX\$(数値 , 桁数)

## □ 引数

数値： 変換対象の整数値( -32767 ~ 32767 )

桁数： 出力桁数( 0 ~ 4 )

## □ 戻り値

16 進数文字列(1 桁~4 桁)

## □ 説明

指定した数値を 16 進数文字列に変換します。

数値には式、変数、定数等の指定が可能です。

PRINT、MSG、GPRINT の引数にて利用可能です。

桁数を指定した場合は数値が指定した桁数に満たない場合は、0 で桁を補います。

指定した桁数を超える場合は数値の桁数を優先します。

桁数をしない場合は、先頭の 0 は付加されません。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

'(' or ')' expected : '(' または ')'がない

Illegal value : 桁数の値が正しくない

Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

入力した整数値を 16 進数表示する

```
10 INPUT "value=",V
20 PRINT HEX$(V,4)
30 GOTO 10
```

## 6.26 DMP\$ 整数の小数付き数値文字列への変換（文字列関数）

## □ 書式

DMP\$(数値)

DMP\$(数値,小数点桁数)

DMP\$(数値,小数点桁数,整数部桁数)

## □ 引数

数値 : 変換対象整数 -32767 ~ 32767

小数点桁数 : 小数点以下の桁数を指定 0 ~ 4（省略時は2）

整数部桁数 : 小数点以上の桁数を指定 0 ~ 8（省略時は0）

## □ 戻り値

小数点を付加した数値文字列

## □ 説明

指定した数値を小数点付きの文字列数値に変換します。

引数の小数点以下の桁数を  $n$  とした場合、数値  $\div 10^n$  の計算を行い、小数点以下の数値を含めて表示します。

```
PRINT DMP$(3141,3)
3.141
```

小数点以下の桁数を指定しない場合、小数点桁数は2となります。

```
PRINT DMP$(3141)
3.14
```

整数部桁数を指定した場合、桁数に満たない場合は空白で補完します。

本関数は PRINT、MSG、GPRINT の引数にて利用可能です。

## □ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
Illegal value	: 指定した引数の値が有効範囲を超えている
Overflow	: 指定した数値が -32768 ~ 32767 を超えている
Illegal value	: 桁数の値が正しくない
'(' or ')' expected	: '(' または ')' が無い

## □ 利用例

アナログ入力値 0~4095 を 0~3.3V で表示する

```
10 CLS
20 V=MAP(ANA(PN0),0,1023,0,3300)
30 LOCATE 0,0: ? DMP$(V,3)
40 WAIT 200
50 GOTO 20
```

実行結果

```
run
2.011
2.215
2.294
2.330
```

アナログ入力値 0~1023 を MAP 関数にて 0~3300 に変換し、

0~3300 の数値を DMP\$( ) 関数にて "0.000" ~ "3.300" の文字列数値に変換して表示しています。

## 6.27 STR\$ 変数が参照する文字列の取得・文字列の切り出し

## □ 書式

STR\$(変数)

STR\$(変数,先頭位置,長さ)

STR\$(文字列)

STR\$(文字列,先頭位置,長さ)

## □ 引数

変数 : 文字列を参照している変数または配列変数

先頭位置 : 切り出す文字位置先頭 1~32767

長さ : 切り出す文字の長さ 1~32767

文字列 : ダブルクォーテーションで囲った文字列定数("文字列")

## □ 戻り値

指定した文字列および変数が参照している文字列を全部または一部を切り出して返します。

## □ 説明

指定した文字列および変数が参照している文字列を全部または一部を切り出して返します。

引数に先頭位置、長さを指定した場合は、その条件にて文字列を切り出して返します。

先頭位置は 1 からの指定となります。

本関数は PRINT、MSG、GPRINT の引数にて利用可能です。

例:

```
10 S="123456789"
20 ?STR$(S)
30 ?STR$(S,5,2)
40 ?STR$("ABCDE",5,1)
```

実行結果

```
run
123456789
56
E
OK
```

上記の例では、変数 S が参照している文字列"123456789"に対して、  
 20 行は全て出力、30 行は 5 番目からの文字から 2 文字 "56"を出力、  
 30 行は直接指定した文字列"ABCDE"の 5 番目の文字から 1 文字を出力しています。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Illegal value : 指定した文字位置、長さの値が不当

'(' or ')' expected : '(' または ')'が無い

Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

変数が参照している文字列を切り出して表示する

```
10 'モリ'ツツサ サンプル
20 S="Hello,Tiny BASIC"
30 L=LEN(S)
40 PRINT STR$(S);" LEN=";L
50 PRINT STR$(S,1,5)
60 C=ASC(S,12)
RUN
Hello,Tiny BASIC LEN=16
Hello
```

## 6.28 WAIT 時間待ち

## □ 書式

WAIT 待ち時間(ミリ秒)

## □ 引数

待ち時間： 0 ～ 32767 （単位 ミリ秒）

## □ 説明

引数で指定した時間(ミリ秒単位)、時間待ち(ウェイト)を行います。

最大で 32767 ミリ秒（32.8 秒）の時間待ちが可能です。

長い時間待ちを行う必要がある場合は、TICK()や GETTIME を使った方法を検討してください。

**（注意）** 時間待ち中はキー操作によるプログラム中断を行うことは出来ません。

短い時間の指定にてループ処理を行う等の対策を行ってください。

## □ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
Illegal value	: 待ち時間に範囲外の値を指定した
Overflow	: 指定した数値が-32768 ～ 32767 を超えている

## □ 利用例

画面上の指定位置に時刻を 1 秒間隔で更新表示する

```
10 SETDATE 2017,4,1,12,0,0
20 CLS
30 LOCATE 5,5
40 DATE
50 WAIT 1000
60 GOTO 30
```

## 6.29 TICK 経過時間取得（数値関数）

経過時間を取得する。

## □ 書式

TICK()

TICK(モード)

## □ 引数

モード 0：経過時間をミリ秒単位で取得する（デフォルト）

1：経過時間を秒単位で取得する

## □ 戻り値

モード指定に従った経過時間を返す。

モード 0： 0 ～ 32767 ミリ秒の経過時間を返す(約 32.767 秒でオーバーフロー)

モード 1： 0 ～ 32767 秒の経過時間を返す(約 9.1 時間でオーバーフロー)

## □ 説明

起動からの経過時間を返します。経過時間は CLT コマンドにて初期化が可能です。

CLT コマンドと組み合わせて使うことで、処理時間の測定を行うことができます。

モード指定無し、または 0 を指定の場合、ミリ秒単位の経過時間を返します。

1 を指定した場合は、秒単位の経過時間を返します。

（注意）オーバーフローが発生しますので、長時間の測定には利用することが出来ません。

## □ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
'(' or ')' expected	: '(' または ')' がない、引数の指定が正しくない
Illegal value	: モードの指定値が正しくない
Overflow	: モードの指定値が -32767 ～ 32767 を超えている

## □ 利用例

1 万回ループの実行時間を測定する（プログラム：TICK.BAS）

```

10 A=0
20 CLT
30 FOR I=1 TO 10000
40 A=A+1
50 NEXT I
60 PRINT A
70 PRINT TICK();"msec"

RUN
10000
225msec

```



## 6.30 POKE 指定アドレスへのデータ書き込み

## □ 書式

POKE 仮想アドレス,データ

POKE 仮想アドレス,データ,データ, ... データ (可変個数指定)

## □ 引数

仮想アドレス : 仮想アドレス(16ビット) \$0000 ~

データ : 書き込むデータ (下位 8ビットのみ有効)

## □ 説明

指定した仮想アドレスに指定したデータを書き込みます。

仮想アドレスの指定には次の定数を利用することで有用な領域への書き込みが簡単に行えます。

VRAM	: 画面表示用メモリ (CW×CH)	WIDTH コマンドの設定により領域サイズは可変
VAR	: 変数領域	サイズ 416 バイト
ARRAY	: 配列変数領域(@ (0) ~ @ (99) )	サイズ 200 バイト
PRG	: プログラム領域	サイズ 4096 バイト
MEM	: ユーザーワーク領域	サイズ 1024 バイト
FNT	: フォント領域	サイズ 1283 バイト
GRAM	: LED マトリックス表示用メモリ	サイズ 9 バイト

仮想アドレスの詳細については、「1.8 メモリーマップ」の「仮想アドレス」を参照下さい。

ユーザーワーク領域は利用者が自由に利用出来る領域です。

それ以外の領域は BASIC の実行にて利用する領域です。

指定した仮想アドレスに上記以外の領域を指定した場合はエラーとなります。

例として画面上のカーソル位置 X,Y への文字の書き込みは次のようになります。

VRAM に書き込んだ内容を画面の表示に反映するには REDRAW コマンドを実行します。

```
10 POKE VRAM+Y*80+X,ASC("A")
20 REDRAW
```

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Out of range value : S 領域外のアドレスを指定した

Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

変数 A から Z の内容をユーザーワーク領域に保存する

```
10 FOR I=0 TO 51
20 POKE MEM+I,PEEK(VAR+I)
30 NEXT I
```

## 6.31 PEEK 指定アドレスの値参照（数値関数）

## □ 書式

PEEK(仮想アドレス)

## □ 引数

仮想アドレス： 参照を行う SRAM 領域先頭からの相対バイトアドレス(16 ビット) \$0000 ～

## □ 戻り値

指定した仮想アドレスに格納されている 1 バイトデータ（0～255）

## □ 説明

指定した仮想アドレスに格納されている値（1 バイト）を返します。

仮想アドレスの指定には次の定数を利用することで有用な領域への参照が簡単に行えます。

VRAM	： 画面表示用メモリ（CW×CH）	WIDTH コマンドの設定により領域サイズは可変
VAR	： 変数領域	サイズ 416 バイト
ARRAY	： 配列変数領域(@ (0) ～ @ (99) )	サイズ 200 バイト
PRG	： プログラム領域	サイズ 4096 バイト
MEM	： ユーザーワーク領域	サイズ 1024 バイト
FNT	： フォント領域	サイズ 1283 バイト
GRAM	： LED マトリックス表示用メモリ	サイズ 9 バイト

仮想アドレスの詳細については、「1.8 メモリーマップ」の「仮想アドレス」を参照下さい。

ユーザーワーク領域は利用者が自由に利用出来る領域です。

それ以外の領域は BASIC の実行にて利用する領域です。

例として画面上のカーソル位置 X,Y に格納されている文字の参照は次のようになります。

C = PEEK(VRAM+Y\*CW+X)

## エラーメッセージ

Syntax error           ： 文法エラー、書式と異なる利用を行った  
 Out of range value   ： 領域外のアドレスを指定した  
 '(' or ')' expected   ： 括弧の指定が正しくない  
 Overflow             ： 指定した数値が-32768 ～ 32767 を超えている

## □ 利用例

文字“A”の LED マトリックス用フォントデータを参照し、2 進数で表示する

```
10 FOR I=0 TO 4
20 D=PEEK(FNT+ASC("A")*5+I)
30 PRINT BIN$(D>>3,5)
40 NEXT I
```

実行結果

```
run
01100
10010
11110
10010
10010
OK
```

## 6.32 PRINT 画面への文字表示

## □ 書式

PRINT

PRINT 文字列|数値

PRINT 文字列|数値;

PRINT 文字列|数値; 文字列|数値;...

PRINT 文字列|数値; 文字列|数値;... ;

PRINT #桁数,文字列|数値; 文字列|数値;... ;

※ ...は可変指定を示す

※ ; は連結指定、カンマ','も可能

## □ 引数

文字列 : 文字列定数または文字列関数

数値 : 数値定数、変数、配列変数、または数値関数、式

連結指定 : セミコロン';' または カンマ','  
文末に付けると改行抑制される桁数 : #数値 または #-数値 の形式で指定する  
例:#3 、 #-3

## □ 説明

指定した文字列、式、関数、変数、数値をスクリーン画面のカーソル位置に表示します。

```
PRINT "Hello,World"
Hello,World
OK
PRINT 123*3
369
OK
PRINT HEX$(123*3)
171
OK
```

文字列、式、関数、変数、数値は連結指定のセミコロン';'、またはカンマ','にて連結して表示することが出来ます。  
また最後に';'または','が付加されている場合は改行しません。

```
10 N=5:C=3:K=10
20 PRINT "N=";N;" C=";C;
30 PRINT " K=";
40 PRINT K
RUN
N=5 C=3 K=10
OK
```

## 数値の整形表示

#数値にて桁数を指定することで、任意の桁数（指定桁に満たない場合は空白を入れる）にて等間隔で表示します。

数値の前に-（マイナス）を付加した場合、空白をではなく、0(零)で不足桁を補います。

#数値は任意の位置、任意の回数指定できます。

## 桁数指定なし

```
PRINT 1;" ":"2;" ":"3
1:2:3
```

## 桁数指定あり(空白文字で補間)

```
PRINT #2,1;" ":"2;" ":"3
1: 2: 3
```

## 桁数指定あり(0で補間)

```
PRINT #-2,1;" ":"2;" ":"3
01:02:03
```

### 表示位置の指定

LOCATE コマンドを併用することで、スクリーン画面の任意の位置に文字を表示することが出来ます。

```

10 LOCATE 10,10
20 PRINT "Hello!"

```

### 色、属性の指定

スクリーン画面が SCREEN 0 のターミナルコンソールの場合、

COLOR コマンド、ATTR コマンドを併用することで、文字の前景色、背景色の指定、点滅、アンダーライン等の属性を付加することが出来ます。

```

10 COLOR 4,3
20 PRINT "Hello,";
40 ATTR 2
50 PRINT "World".
60 COLOR 7,0:ATTR 0

```

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った、引数に数値以外を指定した  
 Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

時刻を表示する

```

10 GETTIME A,B,C
20 LOCATE 0,0:PRINT #-2,A;":";B;":";C
30 WAIT 1000
40 GOTO 20

```

実行結果

```

10:52:00

```

## 6.33 INPUT 数値の入力

## □ 書式

INPUT 変数

INPUT 変数,オーバーフロー時の既定値

INPUT プロンプト, 変数

INPUT プロンプト, 変数, オーバーフロー時の既定値

## □ 引数

変数 : 入力した値を格納する変数または配列変数

プロンプト : 文字列定数 “プロンプト”

オーバーフロー時の既定値 : -32768 ~ 32767

## □ 説明

現在のカーソル位置にて数値の入力をし、指定した変数にその値を格納します。

キーボードから入力できる文字は符号-, +、数値 0~9、入力訂正の[BS]、[DEL]、入力確定の[Enter]です。

それ以外の入力はできません。

~~数値を入力せずに、[Enter]にて入力確定を行った場合、変数には0が格納されます。~~

引数に変数のみを指定した場合、“変数名:”を表示しその後ろの位置から数値を入力します。

```
INPUT A
A:
```

プロンプトを指定した場合は、そのプロンプトを表示しその後ろ位置から数値を入力します。

```
INPUT "Value=",A
Value=
```

オーバーフロー時の既定値を指定した場合、入力値した数値がオーバーフローを発生した場合は、オーバーフロー時の既定値を変数に設定します。オーバーフロー時の既定値を設定していない場合は、Overflow エラーとなります。

オーバーフロー時の既定値なし

```
INPUT A
A:111111
Overflow
OK
```

オーバーフロー時の既定値あり

```
INPUT A, -1
A:111111
OK
?A
-1
OK
```

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った、引数に数値以外を指定した

Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

```
INPUT A
A=1234
OK

INPUT "Value=" A
Value=1234
OK
```

## 6.34 CLS 画面表示内容の全消去

## □ 書式

CLS

CLS 消去対象画面

## □ 引数

消去対象画面 :

0 メインコンソール画面の全消去 (デフォルト)

1 LED マトリックスの消去

## □ 説明

画面上に表示している内容を全て消します。

引数に消去対象画面を指定した場合、指定した画面の消去を行います。

## □ エラーメッセージ

Illegal value : 指定した引数が有効範囲でない

Overflow : 指定した数値が-32768 ~ 32767 を超えている

Syntax error : 文法エラー、書式と異なる利用を行った

## □ 利用例

画面の表示内容を消去する

CLS
OK

## 6.35 COLOR 文字色の設定

## □ 書式

COLOR 文字色

COLOR 文字色, 背景色

## □ 引数

文字色: 色コード 0～8

背景色: 色コード 0～8

## □ 説明

シリアルコンソールの文字色の設定を行います。指定した色は以降の文字表示に反映されます。

文字色、背景色で指定する色コードに対する色は次の表の通りです。

表 1 色コード

色コード	色
0	黒
1	赤
2	緑
3	茶
4	青
5	マゼンタ
6	シアン
7	白(デフォルト)
8	黄

(注意) 利用するターミナルソフトにより色が正しく表示されない場合があります。  
属性指定との併用では正しく表示されない場合があります。  
画面を[CTRL-R]、[Page UP]、[Page Down]キーにて再表示した場合、色情報は欠落します。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った  
Illegal value : 色コードに範囲外の値を指定した  
Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

ランダムな色で\*を表示する

```
10 FOR I=0 TO 10
20 FOR J=0 TO 10
30 COLOR RND(8): ? "*";
35 WAIT 100
40 NEXT J
50 ?
60 NEXT I
```

## 6.36 ATTR 文字表示属性の設定

## □ 書式

ATTR 属性

## □ 引数

属性：属性コード 0 ～ 4

## □ 説明

シリアルコンソールの文字の表示属性を設定します。指定した表示属性は以降の文字表示に反映されます。

属性に指定する属性コードは次の表の通りです。

表 2 属性コード

属性コード	機能
0	標準(デフォルト)
1	下線
2	反転
3	ブリンク
4	ボールド

(注意) 利用するターミナルソフトにより色が正しく表示されない場合があります。  
属性指定との併用では正しく表示されない場合があります。  
画面を[CTRL-R]、[Page UP]、[Page Down]キーにて再表示した場合、色情報は欠落します。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った  
Illegal value : 属性コードに範囲外の値を指定した  
Overflow : 指定した数値が-32768 ～ 32767 を超えている

## □ 利用例

Hello,world を反転表示する

```
10 CLS
20 LOCATE 5,5
30 ATTR 2:?"Hello,world"
40 ATTR 0
```



## 6.37 LOCATE カーソルの移動

## □ 書式

LOCATE 横位置, 縦位置

## □ 引数

横位置：画面上の横位置 0 ～ CW-1

縦位置：画面上の縦位置 0 ～ CH-1

CW、CH は画面の横桁数、縦行数の示す定数です。

利用するフォント等により異なります。

## □ 説明

カーソルを指定した位置に移動します。

縦横位置それぞれの指定に 0 以下の数値を指定した場合、それぞれの位置は 0 となります。

横位置に CW-1 を超える数値を指定した場合、横位置は CW-1 となります。縦位置に CH-1 を超える数値を指定した場合、縦位置は CH-1 となります。

定数 CW、CH に設定されている値は次のようすることで確認することができます。

```
?CW
37
OK
?CH
27
OK
```

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Overflow : 指定した数値が-32768 ～ 32767 を超えている

## □ 利用例

画面の指定位置にメッセージを表示する

```
10 CLS
20 LOCATE 5,5:PRINT "Hello,world."
30 LOCATE 6,6:PRINT "TinyBASIC"
40 LOCATE 7,7:PRINT "Thank you."
```

実行結果

```
Hello,world.
TinyBASIC
Thank you.
```

## 6.38 REDRAW 画面表示の再表示

## □ 書式

REDRAW

## □ 引数

なし

## □ 説明

表示用メモリ（VRAM）の内容を画面に再表示を行います。

[CTRL-R]、[Page UP]、[Page Down]キーによる再表示をコマンドにて行います。

再表示においては、カーソルの移動は行いません。

再表示においては、個々に色付けした文字の色、背景色、属性は欠落します。

再表示後は最後に指定した文字色、背景色、属性に統一されます。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

表示用メモリ（VRAM）に直接書き内容を画面に反映させる

```
10 POKE VRAM+80*5+50,ASC("b")
20 REDRAW
```

## 6.39 VPEEK 画面指定位置の文字コード参照（数値関数）

## □ 書式

VPEEK(横位置, 縦位置)

## □ 引数

横位置: 0 ~ CW - 1

縦位置: 0 ~ CH - 1

CW, CH は画面の横桁数、縦行数の示す定数です。

利用するフォント等により異なります。

## □ 戻り値

指定位置に表示されている文字の文字コード (0 ~ 255)

## □ 説明

画面上の指定位置に表示されている文字の文字コードを取得します。

引数の指定位置が範囲外の場合は 0 を返します。

## □ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
Illegal value	: 指定した引数の値が有効範囲を超えている
Overflow	: 指定した数値が -32768 ~ 32767 を超えている
'(' or ')' expected	: '(' または ')' が無い

## □ 利用例

画面最上位の行に表示されている文字を表示する

```
10 FOR I=0 TO CW-1
20 C=VPEEK(I,0)
30 PRINT CHR$(C);
40 NEXT I
50 PRINT
```

## 6.40 MATRIX LED マトリックス利用の有効/無効設定

## □ 書式

MATRIX 設定

## □ 引数

設定 : ON 利用有効、OFF 利用無効 または 1 利用有効、0 利用無効

## □ 説明

ボード上のLED マトリックス利用の有効、無効設定を行います。  
 起動直後は有効となっています。

LED マトリックスは、表示制御のためにタイマー割り込みとLED マトリックス用のポートを利用します。

LED マトリックス用のポートを他の用途に利用する場合は、本コマンドを使ってLED マトリックスの利用を無効にする必要があります。

また、LED マトリックスの制御を行っているタイマー割り込み処理は、POUT、SHIFTOUT、PULSEIN 等のパルス入出力のパルス幅に誤差を与える可能性があります。状況の応じて本コマンドを利用して、タイマー割り込みを停止して下さい。

## □ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
Illegal value	: 指定した引数の値が有効範囲を超えている
Overflow	: 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

LED マトリックスを停止する

MATRIX OFF OK
------------------

## 6.41 PSET LED マトリックス 点の描画

## □ 書式

PSET 横座標, 縦座標, 色

## □ 引数

横座標 : -32768 ~ 32767 (表示有効範囲は 0 ~ GW-1 : GW は 5)

縦座標 : -32768 ~ 32767 (表示有効範囲は 0 ~ GH-1 : GH は 5)

色 : 0 消灯、1 点灯、2 反転、それ以外 点灯

※GW、GHはグラフィック画面の横ドット数、縦ドット数を示す定数です。

この定数は、LED マトリックスでは GW、GH とも5となります。

## □ 説明

指定した座標に指定した色の点を描画します。

指定した座標がLED マトリックスの有効範囲外の場合、LED は表示されません。

LED マトリックスの利用を停止している場合は、MATRIX コマンドで利用を有効にしてください。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

LED マトリックを順番に点灯させる

```
10 CLS 1
20 D=1
30 FOR Y=0 TO GH-1
40 FOR X=0 TO GW-1
50 PSET X,Y,D
60 WAIT 100
70 NEXT X
80 NEXT Y
90 IF D D=0 ELSE D=1
100 GOTO 30
```

実行結果



## 6.42 LINE LED マトリックス 直線の描画

## □ 書式

LINE 横座標 1, 縦座標 1, 横座標 2, 縦座標 2, 色

## □ 引数

横座標 1 : -32768 ~ 32767 (表示有効範囲は 0 ~ GW-1 : GW は 5)  
 縦座標 1 : -32768 ~ 32767 (表示有効範囲は 0 ~ GH-1 : GH は 5)  
 横座標 2 : -32768 ~ 32767 (表示有効範囲は 0 ~ GW-1 : GW は 5)  
 縦座標 2 : -32768 ~ 32767 (表示有効範囲は 0 ~ GH-1 : GH は 5)  
 色 : 0 消灯、1 点灯、2 反転、それ以外 点灯

※GW、GHはグラフィック画面の横ドット数、縦ドット数を示す定数です。

この定数は、LED マトリックスでは GW、GH とともに 5 となります。

## □ 説明

指定したグラフィック座標 (横座標 1, 縦座標 1) と (横座標 2, 縦座標 2) 間を結ぶ直線を指定した色で描画します。

指定した座標が LED マトリックスの有効範囲外の場合、LED は表示されません。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った  
 Overflow : 指定した数値が -32768 ~ 32767 を超えている

## □ 利用例

直線を描画する

```
10 CLS 1
20 LINE 0,0,4,4,1
30 LINE 0,4,4,0,1
```

実行結果



## 6.43 RECT LED マトリックス 矩形の描画

## □ 書式

RECT x1, y1, x2, y2, 色, モード

## □ 引数

x1 : 左上横座標 -32768 ~ 32767 (表示有効範囲は 0 ~ GW-1 : GW は 5)  
 y1 : 左上縦座標 -32768 ~ 32767 (表示有効範囲は 0 ~ GH-1 : GH は 5)  
 x2 : 右下横座標 -32768 ~ 32767 (表示有効範囲は 0 ~ GW-1 : GW は 5)  
 y2 : 右下縦座標 -32768 ~ 32767 (表示有効範囲は 0 ~ GH-1 : GH は 5)  
 色 : 0 消灯、1 点灯、2 反転、それ以外 点灯  
 モード : 0 塗りつぶしなし、0 以外 塗りつぶしあり

※GW、GH はグラフィック画面の横ドット数、縦ドット数を示す定数です。

この定数は、LED マトリックスでは GW、GH とも 5 となります。

## □ 説明

指定した位置に指定した色で矩形を描画します。

指定した座標が LED マトリックスの有効範囲外の場合、LED は表示されません。

モードに 0 を指定した場合は線のみを描画します。0 以外を指定した場合は指定色で塗りつぶします。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った  
 Overflow : 指定した数値が -32768 ~ 32767 を超えている  
 Illegal value : 指定した数値が不当である

## □ 利用例

矩形を描画し、反転表示を繰り返す。

```

10 CLS 1
20 RECT 0,0,4,4,1,0
30 WAIT 300
40 RECT 0,0,4,4,2,1
50 GOTO 30
  
```

実行結果



## 6.44 CIRCLE LED マトリックス 円の描画

## □ 書式

CIRCLE 横中心座標, 縦中心座標, 半径, 色, モード

## □ 引数

横中心座標 : -32768 ~ 32767 (表示有効範囲は 0 ~ GW-1 : GW は 5)

縦中心座標 : -32768 ~ 32767 (表示有効範囲は 0 ~ GH-1 : GH は 5)

半径 : 1 ~ 32767

色 : 0 消灯、1 点灯、2 反転、それ以外 点灯

モード : 0 塗りつぶしなし、0 以外 塗りつぶしあり

※GW、GHはグラフィック画面の横ドット数、縦ドット数を示す定数です。

この定数は、LED マトリックスでは GW、GH とも5となります。

## □ 説明

指定した中心座標(横中心座標, 縦中心座標)を起点に指定した半径の円を色で描画します。

指定した座標がLED マトリックスの有効範囲外の場合、LED は表示されません。

モードに 0 を指定した場合は線のみを描画します。0 以外を指定した場合は指定した色で塗りつぶします。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

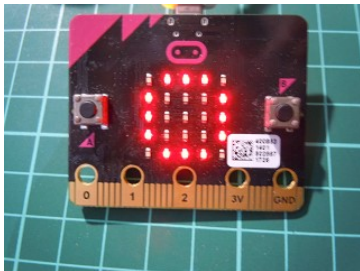
Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

円を描画する

```
10 CLS 1
20 CIRCLE 2,2,2,1,0
```

実行結果





## 6.45 BITMAP LED マトリックス ビットマップ画像の描画

## □ 書式

BITMAP 横座標, 縦座標, 仮想アドレス, インデックス, 幅, 高さ [, 倍率]

## □ 引数

横座標 : -32768 ~ 32767 (表示有効範囲は 0 ~ GW-1 : GW は 5)  
 縦座標 : -32768 ~ 32767 (表示有効範囲は 0 ~ GH-1 : GH は 5)  
 仮想アドレス : 数値(任意の仮想アドレス)  
 インデックス : 0 ~ 32767  
 幅 : 1 ~ 32767  
 高さ : 1 ~ 43767  
 倍率 : 1 ~ 8

※GW、GH はグラフィック画面の横ドット数、縦ドット数を示す定数です。  
 この定数は、LED マトリックスでは GW、GH とも 5 となります。

## □ 説明

指定した座標にビットマップ画像を描画します。

仮想アドレスに描画対象の画像データの先頭格納アドレスを指定します。

インデックスには画像データの先頭格納の格納位置を指定します。

(幅 + 7) / 8 + 高さ \* インデックス の計算にて参照する画像格納位置を移動します。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った  
 Illegal value : 指定した数値が不当である  
 Overflow : 指定した数値が -32768 ~ 32767 を超えている

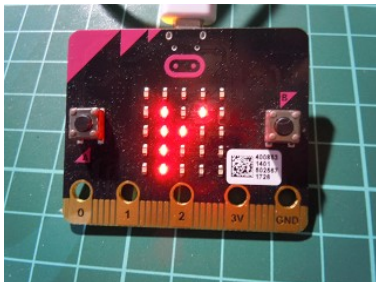
## □ 利用例

次の 56x5 ドットのビットマップを定義し、スクロール表示する



```
10 POKE MEM+0,$7D,$F0,$05,$48,$00,$02,$24,$82,$08,$02
20 POKE MEM+10,$A1,$D4,$C2,$0E,$AA,$04,$12,$AA,$19,$2B
30 POKE MEM+20,$24,$82,$07,$E2,$AA,$11,$22,$A4,$7D,$F8
40 POKE MEM+30,$02,$A9,$D0,$CB,$A6
45 FOR X=0 TO 56
50 BITMAP 0-X,0,MEM,0,56,5
55 WAIT 100
60 NEXT X
```

実行結果



## 6.46 GPRINT LED マトリックス 文字列の描画

## □ 書式

GPRINT 横座標,縦座標

GPRINT 横座標,縦座標, 文字列|数値

GPRINT 横座標,縦座標, 文字列|数値;

GPRINT 横座標,縦座標, 文字列|数値;文字列|数値;...

GPRINT 横座標,縦座標, 文字列|数値;文字列|数値;... ;

GPRINT 横座標,縦座標, #桁数,文字列|数値; 文字列|数値;... ;

※ | はいずれのうち1つを示す

※ ...は可変指定を示す

※ ; は連結指定、カンマ','も可能

## □ 引数

横座標 : 横描画位置 -32768 ~ 32767

縦座標 : 縦描画位置 -32768 ~ 32767

文字列 : 文字列定数または文字列関数

数値 : 数値定数、変数、配列変数、または数値関数、式

連結指定 : セミコロン';' または カンマ','  
文末に付けると改行抑制される桁数 : #数値 または #-数値 の形式で指定する  
例:#3 、 #-3

## □ 説明

指定したグラフィック座標にグラフィックとして文字列を描画します。

MSG コマンドに比べ、任意の座標に文字列を表示することが可能です。

LED マトリックスの領域外の座標の指定も可能です。これにより部分表示を行うことが可能です。

横座標、縦座標以降の記述及び出力結果は PRINT 文と同等です。

出力する文字列の表記方法については、「6.32 PRINT 画面への文字表示」を参照下さい。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った、引数に数値以外を指定した

Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

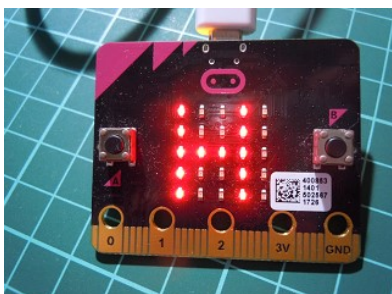
"Hello"のメッセージを[←] [→][↑][↓]キーでスクロール操作する。

```

10 X=0:Y=0
20 K= INKEY()
30 IF K=130 X=X-1
40 IF K=131 X=X+1
50 IF K=129 Y=Y-1
60 IF K=128 Y=Y+1
70 CLS 1
80 GPRINT X,Y,"Hello"
90 WAIT 50
100 GOTO 20

```

実行結果



## 6.47 GSCROLL LED マトリックス グラフィックスクロール

## □ 書式

GSCROLL x1, y1, x2, y2, 方向

## □ 引数

x1 : 左上横座標 0 ~ GW-1 (最大値は環境により可変)  
 y1 : 左上縦座標 0 ~ GH-1 (最大値は環境により可変)  
 x2 : 右下横座標 0 ~ GW-1 (最大値は環境により可変)  
 y2 : 右下縦座標 0 ~ GH-1 (最大値は環境により可変)  
 方向 : UP(0): 上、DOWN(1): 下、RIGHT(2): 右、LEFT(3): 左  
 ※方向は定数またはカッコ内の数値の指定が可能

※GW、GH はグラフィック画面の横ドット数、縦ドット数を示す定数です。  
 この定数は、LED マトリックスでは GW、GH とも5となります。

## □ 説明

画面上の指定範囲表示内容を 1 ドット単位でスクロールします。

画面全体を左スクロールしたい場合は、

GSCROLL 0, 0, GW-1, GH-1, LEFT

のように、画面全体の領域を指定します。

指定した範囲をスクロールしたい場合は、

GSCROLL 0, 0, 4, 3, LEFT

のように範囲を指定します。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った  
 Overflow : 指定した数値が-32768 ~ 32767 を超えている  
 Illegal value : 指定した数値が不当である

## □ 利用例

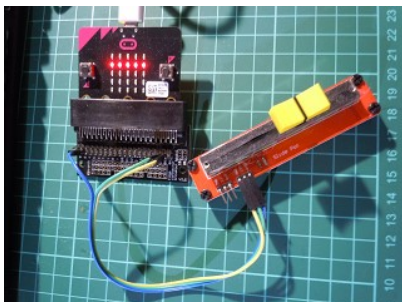
簡易オシロスコープ

スライド抵抗による電圧変化を測定し波形をスクロール表示します。

```

10 'oscilloscope
20 CLS 1
30 "LOOP"
40 R=MAP(ANA(PN0),0,1023,0,4)
50 PSET 4,4-R,1
60 WAIT 100
70 GSCROLL 0,0,4,4,LEFT
80 GOTO "LOOP"
  
```

実行結果



## 6.48 GPEEK LED マトリックス上の指定位置ピクセルの参照（数値関数）

## □ 書式

GPEEK(横位置, 縦位置)

## □ 引数

横位置: 0 ~ GW - 1

縦位置: 0 ~ GH - 1

※GW、GH はグラフィック画面の横ドット数、縦ドット数を示す定数です。

この定数は、LED マトリックスでは GW、GH とも 5 となります。

## □ 戻り値

指定位置に表示されているピクセルの色 0 または 1

## □ 説明

画面上の指定位置に表示されているピクセルを参照します。

引数の指定位置が範囲外の場合は 0 を返します。

## □ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
Overflow	: 指定した数値が -32768 ~ 32767 を超えている
'(' or ')' expected	: '(' または ')' が無い

## □ 利用例

画面上に表示しているドットに有無を 2 進数で表示する

```
10 B=0
10 FOR I=0 TO 4
20 B=(B<<1)+GPEEK(I,0)
30 NEXT I
40 PRINT BIN$(B,8)
50 END
```

## 6.49 GINP LED マトリックス 指定領域のピクセルの有無判定（数値関数）

## □ 書式

GINP(横位置,縦位置,高さ,幅,色)

## □ 引数

横位置 : 0～GW - 1  
 縦位置 : 0～GH - 1  
 幅 : 1 ～ GW-1  
 高さ : 1 ～ GH-1  
 色 : 0 黒、1 白

※GW、GH はグラフィック画面の横ドット数、縦ドット数を示す定数です。

この定数は、LED マトリックスでは GW、GH とも5となります。

## □ 戻り値

1 : 指定領域内（境界含む）に指定した色のピクセルが存在する  
 0 : 指定領域内（境界含む）に指定した色のピクセルが存在しない

## □ 説明

指定した領域内の指定した色のピクセルが存在するをチェックし、その結果を返します。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った  
 Overflow : 指定した数値が-32768 ～ 32767 を超えている  
 '(' or ')' expected : '(' または ')'が無い

## □ 利用例

LED マトリックスの指定領域のドットに有無を判定する

```
10 CLS 1
20 PSET 2,2,1
30 ?GINP(0,0,3,3,1)
```

## 6.50 MSG LED マトリックス メッセージ表示

## □ 書式

MSG スクロール方向, ウェイト時間, メッセージ文

## □ 引数

スクロール方向 : 0 ~ 4 または UP(0), DOWN(1), LEFT(2), RIGHT(3), TOP(4)

ウェイト時間 : 0 ~ 5000 (ミリ秒)

メッセージ文 : 文字列、式、数値 (PRINT 文の引数と同じ)

## □ 説明

LED マトリックスにメッセージを表示します。

指定したメッセージは、スクロール方向で指定した方向にスクロールしながら表示されます。

スクロールする速度はウェイト時間にて調整することができます。

メッセージ文の指定方法は、PRINT 文と同等です。

出力する文字列の表記方法については、「6.32 PRINT 画面への文字表示」を参照下さい。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った、引数に数値以外を指定した

Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

ボタン A を押したら時刻を表示する  
(要見直し)

```
10 GETTIME A,B,C
20 IF !IN(BTNA) MSG LEFT,100,#-2,A;":";B;":";C
30 WAIT 200
40 GOTO 10
```

実行結果

## 6.51 SETFONT LED マトリックス フォント設定

## □ 書式

SETFONT 文字コード, d1, d2, d3, d4, d5

## □ 引数

文字コード : 文字コード 0 ~ 255

d1, d2, d3, d4, d5 : フォントデータ 5 バイト 5x5 ドット分

## □ 説明

LED マトリックスに利用するフォントの設定を行います。

LED マトリックス用のフォントは「1.9 文字(フォント)コード」に示すフォントデータがプリセットされていますが、SETFONT コマンドを使うことにより、任意の文字コードのフォントを変更することができます。

CLP コマンドを使って、設定を元に戻すことも可能です。

## 補足

フォントデータ領域は仮想アドレス FNT にて参照することができます。

任意の文字コードのフォントデータ格納アドレスは次の式で求めることができます。

$$A = \text{FNT} + \text{文字コード} * 5$$

したがって、SETFONT 文字コード, d1, d2, d3, d4, d5 は

$$\text{POKE FNT} + \text{文字コード} * 5, \text{d1}, \text{d2}, \text{d3}, \text{d4}, \text{d5}$$

と等価です。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った、引数に数値以外を指定した

Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

文字 A にスマイルを設定する

```
10 SETFONT ASC("A"),$00,$50,$00,$88,$70
20 MSG TOP,0,"A"
```

実行結果



## 6.52 CLP フォント設定の初期化

## □ 書式

CLP

## □ 引数

なし

## □ 説明

SETFONT コマンドにて変更したフォントデータを初期状態に戻します。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った、引数に数値以外を指定した  
Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

フォント設定の初期化

CLP
OK



## 6.53 TONE 単音出力

## □ 書式

TONE 周波数

TONE 周波数,出力期間

## □ 引数

周波数 : 0 ~ 32767 (Hz) 0 の場合は消音

出力期間 : 0 ~ 32767 (ミリ秒) 0 の場合は、継続再生

## □ 説明

指定した周波数の単音出力を行います。

デフォルトの設定では PN8 ピンを単音出力として使用します。

単音出力ピンに圧電スピーカー（圧電サウナ）を接続すること音を出すことが出来ます。

SETTONE コマンドにて利用するピンの変更が可能です。

出力期間の指定がある場合は、その期間パルスを出力します(ミリ秒単位)。

出力期間の指定がある場合、出力完了待ちを行います。

出力期間の指定がない場合は、NOTONE コマンドで停止指示をするまでパルスを出力し続けます。

音階・周波数対応表

	ド	ド#	レ	レ#	ミ	ファ	ファ#	ソ	ソ#	ラ	ラ#	シ
1	33	35	37	39	41	44	46	49	52	55	58	62
2	65	69	73	78	82	87	93	98	104	110	117	123
3	131	139	147	156	165	175	185	196	208	220	233	247
4	262	277	294	311	330	349	370	392	415	440	466	494
5	523	554	587	622	659	698	740	784	831	880	932	988
6	1047	1109	1175	1245	1319	1397	1480	1568	1661	1760	1865	1976
7	2093	2217	2349	2489	2637	2794	2960	3136	3322	3520	3729	3951
8	4186	4435	4699	4978	5274	5588	5920	6272	6643	7040	7459	7902

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

スペースキーを押したら音を鳴らす

```

10 IF INKEY() = 32 TONE 800,50
20 GOTO 10

```

## 6.54 NOTONE 単音出力停止

## □ 書式

NOTONE

## □ 引数

なし

## □ 説明

TONE コマンドによるパルス出力を停止します。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

## □ 利用例

音を停止する

```
10 TONE 400
20 WAIT 200
30 NOTONE
```

## 6.55 SETTONE 単音出力ポートの設定

## □ 書式

SETTONE ピン番号

## □ 引数

ピン番号 : 0 ~ 32 または PNO ~ PN32 (デフォルト PN8)

## □ 説明

TONE、PLAY コマンドにて単音出力に利用するピン番号を設定します。

## □ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
Cannot use GPIO fuinction	: ピン番号に利用出来ないモード設定を行った
Illegal value	: ピン番号、モードに範囲外の値を指定した
Overflow	: 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

PN12 を単音出力に設定し、音を出力します。

```
10 SETTONE 12
20 TONE 400
30 WAIT 200
40 NOTONE
```

## 6.56 PLAY 音楽演奏(MML 文)

## □ 書式

PLAY "MML 文"

## □ 引数

MML 文 : MML を定義した文字列

## □ 説明

MML (Music Macro Language)にて定義した文法を元に、接続している圧電スピーカーを使って単音による、音楽演奏を行います。音の出力はデフォルトではピン番号 8 を利用します。SETTONE コマンドにて変更が可能です。

MML 文には次のコマンドを利用することが出来ます。

- 音階記号 [#|+|-][長さ][.]  
 音階指定 : C、D、E、F、G、A、B ~ または c、d、e、f、g、a、b  
 順番に ド、レ、ミ、ファ、ソ、ラ、シ、ド の音階に対応します。  
 # : 半音上げる (省略可能)  
 + : 半音上げる (省略可能)  
 - : 半音下げる (省略可能)  
 長さ : 1、2、4、8、16、32、64 (省略時は L による長さ、デフォルト値 4)、省略可能  
 1 は全音符、2 は 2 分音符、4 は四分音符、64 は 64 分の一音符  
 . : 長さを半分伸ばす
- R[長さ] : 休符  
 長さ : 1、2、4、8、16、32、64 (省略時は L による長さ、デフォルト値 4)、省略可能  
 1 は全音符、2 は 2 分音符、4 は四分音符、64 は 64 分の一音符  
 . : 長さを半分伸ばす
- L<長さ>  
 音の長さを指定します。省略時の長さの指定。初期値は 4 (四分音符)  
 長さ : 1、2、4、8、16、32、64 (省略時は L による長さ、デフォルト値 4)、省略可能  
 1 は全音符、2 は 2 分音符、4 は四分音符、64 は 64 分の一音符  
 . : 長さを半分伸ばす
- O<オクターブ>  
 音の高さを指定します。  
 オクターブ : 1~8 初期値は 4
- < : 1 オクターブ上げる
- > : 1 オクターブ下げる
- T<テンポ> : テンポを指定する。初期値は 120、32~255
- 空白文字 : スキップします。

演奏の中断は[ESC]キー 2 回押し、または[CTRL]+C キーにて行うことが出来ます。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った  
 Illegal MML : MML 文の文法エラー  
 Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

「ドラミファソラシド」と演奏する。

```
PLAY "CDEFGAB<C"
OK
```

「ねこふんじゃった」を演奏する。

※MML は、下記のサイトに公開されているものを利用しています。

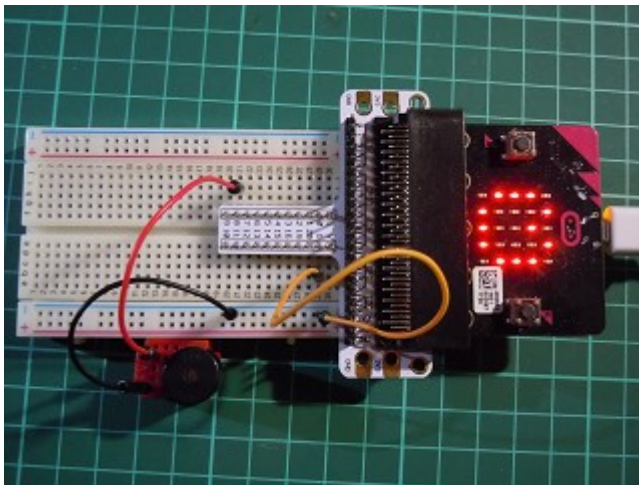
- 主体性の無いページ (<http://astr.me.land.to/>)

MML (<http://astr.me.land.to/tool/mabi/>)

ねこふんじゃった! (<http://astr.me.land.to/tool/mabi/mml/nekof.htm>)

```
10 'ネコフン' ャッタ
20 'リウヨウモト http://astr.me.land.to/tool/mabi/mml/nekof.htm
30 SETFONT 0,$50,$A8,$88,$88,$70
40 MSG TOP,0,CHR$(0)
50 TEMPO 140
60 PLAY "L16D+C+R8F+RF+RD+C+R8F+RF+RD+C+L8RF+RF+RL16FRFRD+C+R8FRFRD+C+R8FRFRD+C+"
70 PLAY "L8RFRFRL16F+RF+RD+C+R8F+RF+RD+C+R8F+RF+RD+C+L8RF+RF+RL16FRFRD+C+R8FRFRD+C+R8FR"
80 PLAY "L16FRD+C+L8RFRFRL16F+RF+RD+C+L8RF+RF+RF+RF+RF+RF+RL16FRFRD+C+L8RFRFRFRFRFR"
90 PLAY "L16F+RF+RD+C+R8F+RF+RD+C+R8F+RF+RD+C+L8RF+RF+RL16FRFRD+C+R8FRFRD+C+R8FRFRD+C+"
100 PLAY "L8RFRFRL16F+RF+R8.F+RC+C+D8C+8.FRF+"

```



## 6.57 TEMPO 音楽演奏のテンポの設定

## □ 書式

TEMPO テンポ

## □ 引数

テンポ : 32 ~ 255 (デフォルト 120)

## □ 説明

PLAY コマンドによる音楽演奏のテンポの設定を行います。

デフォルト値は 120 です。

## □ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
Illegal value	: 引数に範囲外の値を指定した
Overflow	: 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

テンポを 160 に設定します。

```
10 TEMPO 160
20 PLAY "CDEFGAB"
```

## 6.58 DATE 現在時刻の表示

### □ 書式

DATE

### □ 引数

なし

### □ 説明

内蔵 RTC から現在の時刻を読み、その情報を画面に表示します。

**(注意)** 時刻情報は電源 OFF により初期化されます。その場合は SETDATE コマンドにて時刻設定を行って下さい。

### □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った  
 Overflow : 指定した数値が-32768 ~ 32767 を超えている

### □ 利用例

現在の時刻を表示する

```
DATE
2018/01/24 [Wed] 20:55:03
OK
```

## 6.59 GETDATE 日付の取得

## □ 書式

GETDATE 年格納変数,月格納変数,日格納変数,曜日格納変数

## □ 引数

年 格 納 変 数： 取得した西暦年を格納する変数を指定

月 格 納 変 数： 取得した月を格納する変数を指定

日 格 納 変 数： 取得した日を格納する変数を指定

曜日格納変数： 取得した曜日コードを格納する変数を指定

## □ 説明

内蔵 RTC から日付情報を取得し、その値を指定した変数に格納します。

格納される値は次の通りです。

年 格 納 変 数： 西暦年 4 桁整数 1900 ~ 2036

月 格 納 変 数： 1 ~ 12

日 格 納 変 数： 1 ~ 31

曜日格納変数： 曜日コード 0 ~ 6 (0:日 1:月 2:火 3:水 4:木 5:金 6:土)

(注意) 時刻情報は電源 OFF により初期化されます。その場合は SETDATE コマンドにて時刻設定を行って下さい。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

現在の日付を取得する

```
DATE
2018/01/20 [Sat] 12:03:05
OK
GETDATE A,B,C,D
OK
PRINT #-2,a,"/",b,"/",c
2018/01/20
OK
```



## 6.60 GETTIME 時刻の取得

## □ 書式

GETTIME 時格納変数,分格納変数 秒格納変数

## □ 引数

時格納変数： 取得した時を格納する変数を指定

分格納変数： 取得した分を格納する変数を指定

秒格納変数： 取得した秒を格納する変数を指定

## □ 説明

内蔵 RTC から日付情報を取得し、その値を指定した変数に格納します。

格納される値を次の通りです。

時格納変数： 0 ～ 23 整数

分格納変数： 0 ～ 59 整数

秒格納変数： 0 ～ 59 整数

**(注意)** 時刻情報は電源 OFF により初期化されます。その場合は SETDATE コマンドにて時刻設定を行って下さい。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った  
Overflow : 指定した数値が-32768 ～ 32767 を超えている

## □ 利用例

現在の時間を取得する

```
DATE
2018/01/20 [Sat] 12:03:05
OK
GETTIME A,B,C
OK
PRINT #-2,a,": ",b,": ",c
12:03:32
OK
```

(補足) PRINT 文の #-2 は数値を 2 桁(0 付き)で表示する指定です。

## 6.61 SETDATE 時刻の設定

## □ 書式

SETDATE 年,月,日,時,分,秒

## □ 引数

年：1900 ～ 2036      西暦年 4 桁の整数  
 月：1 ～ 12          整数  
 日：1 ～ 31          整数  
 時：0 ～ 23          整数  
 分：0 ～ 59          整数  
 秒：0 ～ 59          整数

## □ 説明

指定した時刻を内蔵 RTC に設定します。

**(注意)** 時刻情報は電源 OFF により初期化されます。その場合は SETDATE コマンドにて時刻設定を行って下さい。

## □ エラーメッセージ

Illegal value           : 指定した引数の数値が有効範囲以外  
 Syntax error           : 文法エラー、書式と異なる利用を行った  
 Overflow               : 指定した数値が-32768 ～ 32767 を超えている

## □ 利用例

時刻の設定と表示を行う

A ボタンを押すと、LED マトリックスに時刻をスクロール表示します。

```

1 'トゲイ
10 MATRIX ON
20 SETDATE 2018,1,16,12,0,0
30 IF !IN(BTNA) GOSUB "@ShowTime"
40 WAIT 200
50 GOTO 30
60 "@ShowTime"
70 GETTIME T1,T2,T3
80 MSG LEFT,80,#-2,T1;":";T2;":";T3;" "
90 RETURN
  
```

## 6.62 GPIO GPIO 機能設定

## □ 書式

GPIO ピン番号, モード

## □ 引数

ピン番号 : 0 ~ 32 または PNO ~ PN32

A ボタン、B ボタンは BTNA, BTNB の定数名による指定が可能です。

モード :

OUTPUT : デジタル出力  
 INPUT\_FL : デジタル入力 (フロート状態 : Arduino の INPUT 指定と同じ)  
 INPUT\_PU : デジタル入力 (内部プルアップ抵抗有効)  
 INPUT\_PD : デジタル (内部プルダウン抵抗有効)

## □ 説明

ボード上の指定したピン番号の入出力機能の設定を行います。

Arduino の pinMode() に相当します。

ピン番号の指定には、0~32 の数値または、定数 PNO ~ PN32 が可能です。

また、GPIO は省略可能です。下記の2つは等価です。

GPIO PA0, OUTPUT  
 PA0, OUTPUT

各モードで利用可能なピン番号

モード	ピン番号
OUTPUT	PN0 ~ PN16, PN19, PN20, PN23 ~ PN28
INPUT_FL	または
INPUT_PU	0 ~ 16, 19, 20, 23 ~ 28
INPUT_PD	

(注意) デフォルトでは、A ボタン、B ボタンは INPUT\_FL が設定されています。また、LED マトリックスで利用するポートは OUTPUT が設定されています。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った  
 Illegal value : ピン番号、モードに範囲外の値を指定した  
 Cannot use GPIO fuinction : ピン番号に利用出来ないモード設定を行った  
 Overflow : 指定した数値が -32768 ~ 32767 を超えている  
 Illegal value : 指定した数値が不当である

## □ 利用例

PN0 ピンからデジタル入力値を読み取り、その値を画面に随時表示します。

```

10 CLS
20 GPIO PN0, INPUT_PU
30 A=IN(PN0)
40 LOCATE 5,5: ? A;
50 GOTO 30
  
```

## 6.63 OUT デジタル出力

## □ 書式

OUT ピン番号, 出力値

## □ 引数

ピン番号 : 0 ~ 32 または PNO ~ PN32

A ボタン、B ボタンは BTNA, BTNB の定数名による指定が可能です。

出力値 :

LOW または 0 : 0V を出力する  
HIGH or 0 以外の値 : 3.3V を出力する

## □ 説明

指定ピンから、指定した出力を行います。

出力を行う場合は事前に GPIO コマンドによる機能設定（出力設定）が必要です。

## □ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
Cannot use GPIO fuinction	: ピン番号に利用出来ないモード設定を行った
Illegal value	: ピン番号、モードに範囲外の値を指定した
Overflow	: 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

LED マトリックスの左上の LED を直接点滅させます。

```
1 'L 効
5 MATRIX OFF
10 GPIO 3,OUTPUT
20 OUT 3,LOW
30 GPIO 26,OUTPUT
35 "@loop"
40 OUT 26,HIGH
50 WAIT 300
60 OUT 26,LOW
70 WAIT 300
80 GOTO "@loop"
```

## 6.64 POUT PWM パルス出力

## □ 書式

POUT ピン番号, デューティー値

## □ 引数

ピン番号 : 0 ~ 32 または PNO ~ PN32

デューティー値 : 0 ~ 255

0 がデューティー比 100%となります。

255 がデューティー比 100%となります。

## □ 説明

指定ピンから、PWM パルス出力を行います。周波数は 490Hz となります(Arduino Uno 互換) (要検証)  
パルス出力を停止する場合は、デューティー値に 0 を指定して下さい。

## □ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
Illegal value	: ピン番号、モードに範囲外の値を指定した
Overflow	: 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

アナログジョイスティックでサーボモーターの制御を行う  
(画像)

```

1 'サーボ モーター制御'
5 CLS
30 P=MAP(ANA(PN0),0,1023,102,491)
40 POUT PN1,P
45 D=MAP(P,102,491,-90,90)
47 LOCATE 0,0:?"#3,D
50 GOTO 30

```

## 6.65 SHIFTOUT デジタルシフトアウト出力

## □ 書式

SHIFTOUT データピン番号, クロックピン番号, 出力形式, 出力データ

## □ 引数

データピン番号: データを出力するピン 0 ~ 32 または PNO ~ PN32

クロックピン番号: クロックを出力するピン 0 ~ 32 または PNO ~ PN32

出力形式: 出力するデータの順番を下記にて指定

LSB または 0: 下位ビットから出力する

MSB または 1: 上位ビットから出力する

出力データ: 出力するデータ (下位 8 ビットのみ有効)

## □ 説明

クロックにて同期を行い、データピンから 1 バイト分のデータを 1 ビットずつ出力します。

Arduino の shiftOut() と同等の動作をします。

データピン、クロックピンは事前に GPIO コマンドによる機能設定 (デジタル出力) が必要です。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った  
 Illegal value : ピン番号、モードに範囲外の値を指定した  
 Overflow : 指定した数値が -32768 ~ 32767 を超えている

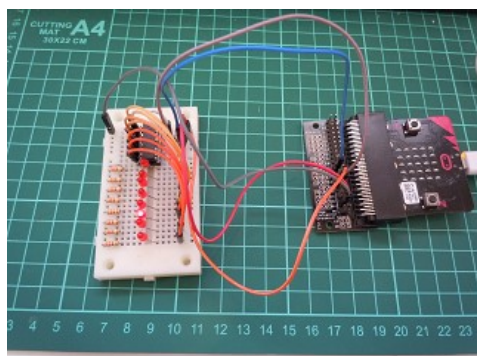
## □ 利用例

シフトレジスタ 74HC595 を使って外付け LED8 個を制御する

```

10 GPIO PN12,OUTPUT:'データ
20 GPIO PN13,OUTPUT:'ラッチ
30 GPIO PN14,OUTPUT:'クロック
40 D=$80
50 FOR I=0 TO 6
60 GOSUB "led"
70 D=D>>1
80 WAIT 200
90 NEXT I
100 FOR I=0 TO 6
110 GOSUB "led"
120 D=D<<1
130 WAIT 200
140 NEXT I
150 GOTO 40
160 "led"
170 OUT PN13,LOW
180 SHIFTOUT PN12,PN14,MSB,D
190 OUT PN13,HIGH
200 RETURN
  
```

実行結果



## 6.66 IN デジタル入力（数値関数）

## □ 書式

IN(ピン番号)

## □ 引数

ピン番号：0 ～ 32 または PNO ～ PN32

A ボタン、B ボタンは BTNA, BTNB の定数名による指定が可能です。

## □ 戻り値

取得した値 0 (LOW) または 1 (HIGH)

## □ 説明

指定ピンの入力値を読み取り、その値を返します。

入力を行う場合は事前に GPIO コマンドによる機能設定（入力設定）が必要です。

## □ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
Illegal value	: ピン番号、モードに範囲外の値を指定した
Cannot use GPIO fuinction	: ピン番号に利用出来ないモード設定を行った
Overflow	: 指定した数値が <sup>a</sup> -32768 ～ 32767 を超えている
'(' or ')' expected	: '(' または ')'がない

## □ 利用例

PN0 ピンからデジタル入力値を読み取り、その値を画面に随時表示します。

```

10 CLS
20 GPIO PN0, INPUT_PU
30 A=IN(PN0)
40 LOCATE 5,5: ? A;
50 GOTO 30

```

## 6.67 ANA アナログ入力（数値関数）

## □ 書式

ANA(ピン番号)

## □ 引数

ピン番号： 0, 1, 2, 3, 4, 10 または PNO, PN1, PN2, PN3, PN4, PN10

## □ 戻り値

取得した値 0～1023(10 ビット)

## □ 説明

指定ピンのアナログ入力値を読み取り、その値を返します。

アナログ入力を行う場合は事前に GPIO コマンドによる機能設定（アナログ入力）が必要です。

## □ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
Illegal value	: ピン番号、モードに範囲外の値を指定した
Overflow	: 指定した数値が-32768 ~ 32767 を超えている
'(' or ')' expected	: '(' または ')'がない

## □ 利用例

PNO ピンからアナログ入力値を読み取り、その値を画面に随時表示します。

```
10 CLS
30 A=ANA(PN0)
40 LOCATE 5,5: ? A; "    "
50 GOTO 30
```



## 6.68 SHIFTIN デジタルシフト入力（数値関数）

## □ 書式

SHIFTIN(データピン番号, クロックピン番号, 入力形式)

SHIFTIN(データピン番号, クロックピン番号, 入力形式, 条件)

## □ 引数

データピン番号 : 0 ~ 32 または PNO ~ PN32

クロックピン番号 : 0 ~ 32 または PNO ~ PN32

入力形式 : 入力するデータの順番を下記にて指定

LSB または 0 下位ビットから入力する

MSB または 1 上位ビットから入力する

条件 : LOW または HIGH

データピンからのデータを読み取るクロックのタイミングを指定します。

LOW : クロックが LOW の場合にデータを読み取る。

HIGH : クロックが HIGH の場合にデータを読み取る。

## □ 戻り値

入力値（1 バイト）

## □ 説明

クロックにて同期を行い、データピンから 1 バイト分のデータを 1 ビットずつ入力します。

Arduino の shiftIn() と同等の動作をします。

引数に条件を指定した場合、クロックが指定した状態の時にデータピンからデータを読み取ります。

条件を指定していない場合は、条件は HIGH（Arduino の shiftIn() と同様の仕様）となります。

データピン、クロックピンは事前に GPIO コマンドによる機能設定（デジタル入出力）が必要です。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Illegal value : ピン番号、モードに範囲外の値を指定した

Overflow : 指定した数値が -32768 ~ 32767 を超えている

'(' or ')' expected : '(' または ')' がない

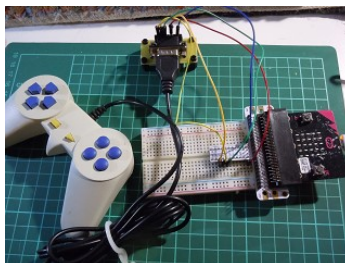
## □ 利用例

ファミコン用互換ゲームパッドからボタン操作情報を取得する

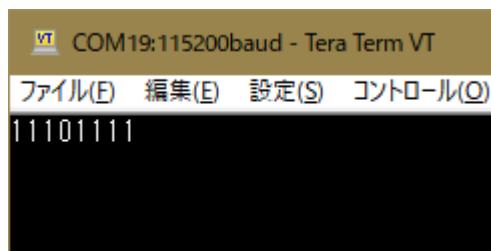
```

10 CLS
20 GPIO PN12, INPUT_FL: 'データ
30 GPIO PN13, OUTPUT: 'ラッチ
40 GPIO PN14, OUTPUT: 'クロック
50 OUT PN13, HIGH
60 OUT PN13, LOW
70 R=SHIFTIN(PN12, PN14, LSB, LOW)
80 LOCATE 0, 0: ?BIN$(R, 8)
90 WAIT 100
100 GOTO 50

```



実行結果



## 6.69 PULSEIN 入力パルス幅の計測力（数値関数）

## □ 書式

PULSEIN(パルス入力ピン番号, 検出信号, タイムアウト)

PULSEIN(パルス入力ピン番号, 検出信号, タイムアウト, スケール値)

## □ 引数

パルス入力ピン番号 : 0 ~ 32 または PNO ~ PN32

検出信号 : LOW、HIGH または 0、1 測定対象のパルス

タイムアウト : パルス検出待ちタイムアウト時間 0 ~ 32767 (ミリ秒)

スケール値 : 計測時間のスケール変換 1 ~ 1327671 (デフォルト値 1)

## □ 戻り値

正常時 : 測定したパルス幅 0 ~ 32767 (単位はスケール値 × マイクロ秒)

タイムアウト時 : 0

オーバーフロー時 : -1

## □ 説明

**パルス入力ピン番号**で指定したピンから入力される信号のパルス幅を計測し、その値を返します。

測定完了は、入力ピンの状態が**検出信号**の状態になった時点で計測を開始し、**検出信号**の状態でなくなった時点で計測を終了します。たとえば、測定する**検出信号**が **HIGH** の場合、PULSEIN()はピンが **HIGH** になるのを待ち、**HIGH** になった時点で計測を開始し、ピンが **LOW** になるタイミングで計測を終了し、そのパルスの長さを返します。**タイムアウト**内に完全なパルスが受信されなかった場合は 0 を返します。

測定したパルスが整数値 32767 を超えた場合は、オーバーフローとし-1 を返します。

オーバーフローを回避したい場合は、**スケール値**を適宜調整して下さい。

**スケール値**が 1 の場合、1~31767 マイクロ秒までのパルスの測定を行うことができます。この場合、500kHz~15Hz までのパルスを測定出来ます。15Hz より遅い、信号の測定はオーバーフローが発生します。

**スケール値**が 1000 の場合、1~31767 ミリ秒までのパルスの測定を行うことができます。この場合、500Hz~0.015Hz までのパルスの測定が可能です。

本関数は Arduino の `pulseIn()`関数を利用して計測しています。

測定値はあくまでも目安であり、精度はあまり高くありません。

データピン、クロックピンは事前に GPIO コマンドによる機能設定（デジタル入出力）が必要です。

## □ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
Illegal value	: ピン番号、モードに範囲外の値を指定した
Overflow	: 指定した数値が-32768 ~ 32767 を超えている
'(' or ')' expected	: '(' または ')'がない

## □ 利用例

パルス幅を計測する

```
10 CLS
20 GPIO BTNA, INPUT_FL
30 A=PULSEIN(BTNA, LOW, 200, 1)
40 LOCATE 0,0: ?#8,A
50 GOTO 30
```

## 6.70 I2CR I2C スレーブデバイスからのデータ受信（数値関数）

## □ 書式

I2CR(デバイスアドレス,コマンドアドレス,コマンド長,受信データアドレス,データ長)

## □ 引数

デバイスアドレス : 0 ~ 127 (\$00 ~ \$7F) (7ビット指定)  
I2C スレーブアドレスを7ビット形式で指定  
コマンドアドレス : 0 ~ 32767 (\$0000 ~ \$1FFFF)  
送信するコマンドが格納されている仮想アドレス  
コマンド長 : 0 ~ 32767  
送信するコマンドのバイト数  
受信データアドレス : 0 ~ 32767 (\$0000 ~ \$1FFFF)  
受信データを格納するSRAM内仮想アドレス  
データ長 : 0 ~ 32767  
受信するデータのバイト数

## □ 説明

I2C スレーブデバイスからデータを受信します。

送信先はデバイスアドレスにてI2C スレーブアドレスを指定します。

受信において、コマンド等の制御データの送信が必要な場合は、コマンドアドレス、コマンド長にて送信するデータを指定します。受信のみを行う場合は、コマンドアドレス、コマンド長に0を設定します。送信受信するデータは仮想アドレスにて指定します。

仮想アドレスの指定には次の定数を利用することで有用な領域への参照が簡単に行えます。

VRAM	: 画面表示用メモリ (CW×CH)	WIDTH コマンドの設定により領域サイズは可変
VAR	: 変数領域	サイズ 416 バイト
ARRAY	: 配列変数領域 (@(0)~@(99) )	サイズ 200 バイト
PRG	: プログラム領域	サイズ 4096 バイト
MEM	: ユーザーワーク領域	サイズ 1024 バイト
FNT	: フォント領域	サイズ 1283 バイト
GRAM	: LED マトリックス表示用メモリ	サイズ 9 バイト

I2C 通信には次の接続ピンを利用します。

PN19 : SDA (I2C データ)

PN20 : SCL (I2C クロック)

## □ 戻り値

0 : 正常終了  
1 : 通信バッファに対してデータが長すぎる  
2 : アドレス送信に NACK が返された  
3 : データ送信に NACK が返された  
4 : その他のエラー

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った  
Illegal value : 指定した引数の値が不当である  
Out of range value : 指定した値が有効範囲を超えている  
Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

I2C EEPROM(AT24C256 スレーブアドレス \$50)にデータの読み書きを行う

```
100 POKE MEM+0,$00,$00
110 POKE MEM+2,64,65,66,67
120 R=I2CW($50,MEM,2,MEM+2,4)
130 ? "snd r=";R
140 POKE MEM+6,0,0,0,0
150 WAIT 5
160 R=I2CR($50,MEM,2,MEM+6,4)
170 ? "rcv r=";R
180 ? "rcv data:";
190 FOR I=0 TO 3
200 ? PEEK(MEM+6+I);" ";
210 NEXT I
220 ?
```

## 実行結果

```
run
snd r=0
rcv r=0
rcv data:64 65 66 67
OK
```

## 6.71 I2CW I2C スレーブデバイスへのデータ送信（数値関数）

## □ 書式

I2CW(デバイスアドレス,コマンドアドレス,コマンド長,データアドレス,データ長)

## □ 引数

デバイスアドレス : 0 ~ 127 (\$00 ~ \$7F) (7ビット指定)  
 I2C スレーブアドレスを7ビット形式で指定

コマンドアドレス : 0 ~ 32767 (\$0000 ~ \$1FFFF)  
 送信するコマンドが格納されている SRAM 内仮想アドレス

コマンド長 : 0 ~ 32767  
 送信するコマンドのバイト数

データアドレス : 0 ~ 32767 (\$0000 ~ \$1FFFF)  
 送信するデータが格納されている SRAM 内仮想アドレス

データ長 : 0 ~ 32767  
 送信するデータのバイト数

## □ 説明

I2C スレーブデバイスにデータを送信します。

送信先はデバイスアドレスにて I2C スレーブアドレスを指定します。

送信するデータは SRAM 先頭からの仮想アドレスにて指定します。

送信するデータはあらかじめ、設定しておく必要があります。

コマンドとデータの区別はありません。デバイスに対しては、単純にコマンド、データの順に送信しています。

仮想アドレスの指定には次の定数を利用することで有用な領域への参照が簡単に行えます。

VRAM	: 画面表示用メモリ (CW×CH)	WIDTH コマンドの設定により領域サイズは可変
VAR	: 変数領域	サイズ 416 バイト
ARRAY	: 配列変数領域 (@ (0) ~ @ (99) )	サイズ 200 バイト
PRG	: プログラム領域	サイズ 4096 バイト
MEM	: ユーザーワーク領域	サイズ 1024 バイト
FNT	: フォント領域	サイズ 1283 バイト
GRAM	: LED マトリックス表示用メモリ	サイズ 9 バイト

I2C 通信には次の接続ピンを利用します。

PN19 : SDA (I2C データ)

PN20 : SCL (I2C クロック)

## □ 戻り値

- 0 : 正常終了
- 1 : 通信バッファに対してデータが長すぎる
- 2 : アドレス送信に NACK が返された
- 3 : データ送信に NACK が返された
- 4 : その他のエラー

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Illegal value : 指定した引数の値が不当である

Out of range value : 指定した値が有効範囲を超えている

Overflow : 指定した数値が -32768 ~ 32767 を超えている

## □ 利用例

I2C EEPROM(AT24C256 スレーブアドレス \$50)にデータの読み書きを行う

```

100 POKE MEM+0,$00,$00
110 POKE MEM+2,64,65,66,67
120 R=I2CW($50,MEM,2,MEM+2,4)
130 ? "snd r=";R
140 POKE MEM+6,0,0,0,0
150 WAIT 5
160 R=I2CR($50,MEM,2,MEM+6,4)
170 ? "rcv r=";R
180 ? "rcv data:";
190 FOR I=0 TO 3
200 ? PEEK(MEM+6+I);" ";
210 NEXT I
220 ?

```

## 実行結果

```

run
snd r=0
rcv r=0
rcv data:64 65 66 67
OK

```

接続している I2C スレーブを調べる

(補足) I2C スレーブアドレスのみ送信し、正常終了(ACK を返した)のアドレスを調べています。

10 FOR I=0 TO \$7F

```

20 C=I2CW(I,MEM,0,MEM,0)
30 IF C=0 PRINT HEX$(I,2);" ";
40 NEXT I
50 PRINT :PRINT "done."

```

## 実行結果

```

run
0E 1D
done.
OK

```



## 6.72 NPBEGIN NeoPixel の利用開始

## □ 書式

NPBEGIN ピン番号, LED 数

## □ 引数

ピン番号 : NeoPixel のデータピン 0 ~ 32 または PNO ~ PN32

LED 数 : NeoPixel の LED 構成数 1 ~ 256

## □ 説明

NeoPixel の利用を開始します。

LED 数で指定した表示用バッファを確保し、指定したピン番号を利用して NeoPixel の制御を開始します。

既に NPBEGIN コマンドにて NeoPixel の利用開始している場合は、一旦利用を終了してから利用開始します。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った  
 Illegal value : ピン番号、LED 数に有効範囲外の値を指定した  
 Out of range value : 指定した値が有効範囲を超えている  
 Overflow : 指定した数値が -32768 ~ 32767 を超えている

## □ 利用例

LED16 個搭載 NeoPixel リングを使って青い軌跡を回転表示させる

```
10 'Neopixel(1)
20 NPBEGIN 0,16
30 NPCLS
40 FOR I=0 TO 7
50 NPRGB I,0,0,(2<<I)-1
60 NEXT I
70 NPSHIFT 1
80 WAIT 50
90 GOTO 70
```

実行結果





## 6.73 NPEND NeoPixel の利用終了

## □ 書式

NPEND

## □ 引数

なし

## □ 説明

NeoPixel の利用を終了します。

NeoPixel の表示は消去し、表示用バッファ等の資源は解放されます。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

NeoPixel の利用を終了します。

```
NPEND
OK
```

## 6.74 NPCLS NeoPixel 表示クリア

## □ 書式

NPCLS

## □ 引数

なし

## □ 説明

NeoPixel の点灯中の LED をすべて消灯します。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Neopixel not start : NeoPixel が利用開始されていない

Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

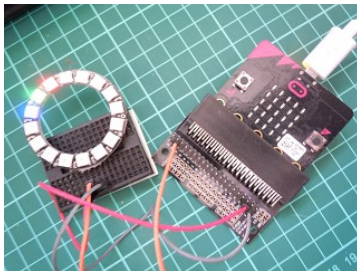
NeoPixel の LED を 3 つ点灯した後、2 秒後にすべての LED を消灯する

```

10 NPBEGIN 0,16
20 NPRGB 0,7,0,0
30 NPRGB 1,0,7,0
40 NPRGB 2,0,0,7
50 WAIT 2000
60 NPCLS

```

## 実行結果



## 6.75 NPRGB NeoPixel 指定ピクセル RGB 表示指定

## □ 書式

NPRGB LED ピクセル番号, R, G, B [,表示指定]

## □ 引数

LED ピクセル番号 : NeoPixel の LED ピクセル番号 0 からの通番  
 R : RGB の R 成分 0 ~ 255  
 G : RGB の G 成分 0 ~ 255  
 B : RGB の B 成分 0 ~ 255  
 表示指定 : 0 表示に反映しない 1 表示に反映する (デフォルト)

## □ 説明

NeoPixel の指定ピクセルの表示指定を行います。

表示指定に 1 を指定した場合、設定は直ちに表示に反映されます。

表示指定に 0 を指定した場合、設定のみで表示には反映されません。この場合は、NPSHOW コマンドにて反映することができます。

## □ エラーメッセージ

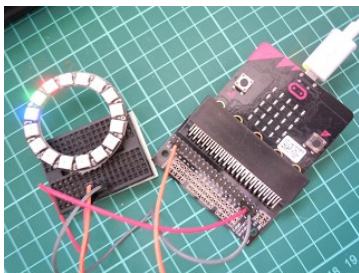
Syntax error : 文法エラー、書式と異なる利用を行った  
 Neopixel not start : NeoPixel が利用開始されていない  
 Illegal value : 指定した引数の値が正しくない  
 Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

NeoPixel の LED 3 点 赤、緑、青を 点灯する

```
10 NPBEGIN 0,16
20 NPRGB 0,7,0,0
30 NPRGB 1,0,7,0
40 NPRGB 2,0,0,7
```

## 実行結果



## 6.76 NPPSET NeoPixel 指定ピクセルの色コード指定

## □ 書式

NPPSET LED ピクセル番号, 色コード [,表示指定]

## □ 引数

LED ピクセル番号 : NeoPixel の LED ピクセル番号 0 からの通番  
 色コード : 0 ~ 32767、(RGB15 ビット、各色 5 ビット)  
 表示指定 : 0 表示に反映しない 1 表示に反映する (デフォルト)

## □ 説明

NeoPixel の指定ピクセルの色コード指定を行います。

表示指定に 1 を指定した場合、直ちに表示に反映されます。

表示指定に 0 を指定した場合、設定のみで表示には反映されません。NPSHOW コマンドにて表示に反映することができます。

色コードは 15 ビットの範囲で指定します。15 ビットの構成は次の通りです。

RRRRRGGGGGBBBBB

この 15 ビット長の色コードは関数 RGB() を使って求めることができます。

RGB 各色は 0 ~ 31 (5 ビット) まで指定可能です。

C=RGB(1,2,3)

これは、下記の計算で求める方法と等価です。

C=(1<<10)+(2<<5)+3

NeoPixel の RGB の各色は 8 ビット長です。本コマンドの色コードによる指定は 5 ビット長です。

残り 3 ビットは NPLEVEL コマンドにて輝度として指定することができます。

NPLEVEL コマンドにて指定する輝度 0~3 を指定した場合、最終的な RGB 各色要素の設定は次の値となります。

R = (色コード>>10) <<輝度

G = ((色コード>>5) & \$1F) <<輝度

B = (色コード & \$1F) <<輝度

## □ エラーメッセージ

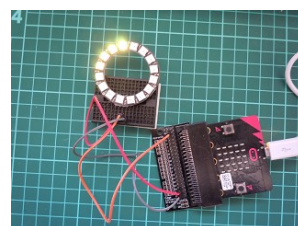
Syntax error : 文法エラー、書式と異なる利用を行った  
 Neopixel not start : NeoPixel が利用開始されていない  
 Illegal value : 指定した引数の値が正しくない  
 Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

NeoPixel の LED16 点のうち 6 点に配列で定義した色設定にする

```
10 @(0)=32736,15840,10560,5280,2112,1056
20 NPBEGIN 0,16
30 NPLEVEL 0
40 FOR I=0 to 5
50 NPPSET I,@(I),0
60 NEXT I
70 NPSHOW
```

実行結果



## 6.77 NPPUT NeoPixel ピクセルデータの転送

## □ 書式

NPPUT LED ピクセル番号, データ格納仮想アドレス, ピクセル数, バイト長 [,表示指定]

## □ 引数

LED ピクセル番号 : NeoPixel の LED ピクセル番号 0 からの通番  
 データ格納仮想アドレス : 0 ~ 32767 、(RGB15 ビット、各色 5 ビット)  
 転送数 : 転送するピクセルデータ数 1 ~  
 バイト長 : 1 ピクセルあたりの色コードバイト数 1 ~ 3  
 表示指定 : 0 表示に反映しない 1 表示に反映する (デフォルト)

## □ 説明

NeoPixel にピクセルデータを転送します。

表示指定に 1 を指定した場合、直ちに表示に反映されます。

表示指定に 0 を指定した場合、設定のみで表示には反映されません。NPSHOW コマンドにて表示に反映することができます。

各ピクセルの色コードはバイト長の指定により異なります。バイト長 1~3 の各色情報は次の通りです。

バイト長が 1 の場合

色コードは下記の 8 ビット構成となります。

**RRRGGGBB**

色コードの範囲は 0~255 となります。

この色コードは関数 RGB8() を使って求めることができます。

`C=RGB8(1,2,3)`

バイト長が 2 の場合

色コードは 15 ビットの範囲で指定します。15 ビットの構成は次の通りです。

**RRRRRGGGGGBBBBB**

この 15 ビット長の色コードは関数 RGB() を使って求めることができます。

バイト長が 3 の場合

色コードは R、G、B 各色要素に 1 バイトを要します。

**RRRRRRRR , GGGGGGGG , BBBBBBBB**

この 15 ビット長の色コードは関数 RGB() を使って求めることができます。

`C=RGB(1,2,3)`

NeoPixel の RGB の各色は 8 ビット長です。

バイト長に 2 または 3 を指定した場合、不足分のビットのうち 3 ビットは NPLEVEL コマンドにて輝度として指定することができます。

## □ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
Neopixel not start	: NeoPixel が利用開始されていない
Illegal value	: 指定した引数の値が正しくない
Overflow	: 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

1 バイト長色コードデータ 16 個を転送し表示する

```

10 POKE MEM,10,20,40,80,160,15,27,56,88,200,100,120,55,177,188,72
20 NPBEGIN 0,16
30 NPCLS:NPLEVEL 0
40 NPPUT 0,MEM,16,1,1
50 NPSHIFT
60 WAIT 100
70 GOTO 50
    
```

実行結果



## 6.78 NPSHOW NeoPixel 設定を表示に反映

## □ 書式

NPSHOW

## □ 引数

なし

## □ 説明

NeoPixel の表示設定を表示に反映します。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Neopixel not start : NeoPixel が利用開始されていない

Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

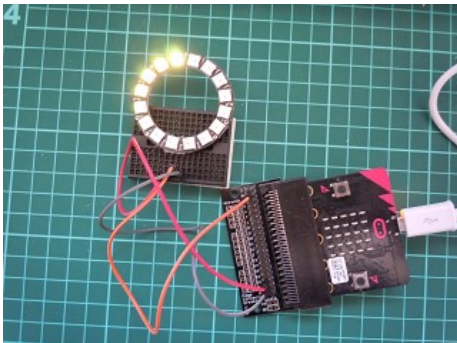
NeoPixel の LED16 点のうち 6 点に配列で定義した色設定にする

```

10 @(0)=32736,15840,10560,5280,2112,1056
20 NPBEGIN 0,16
30 NPLEVEL 0
40 FOR I=0 to 5
50 NPPSET I,@(I),0
60 NEXT I
70 NPSHOW

```

## 実行結果



## 6.79 NPSHIFT NeoPixel 表示を1つシフト

## □ 書式

NPSHIFT 方向 [, ループ指定 [,表示指定]]

## □ 引数

方向 : UP、DOWN または 0、1

ループ指定 : 1 端の表示を先頭に移動する（デフォルト）、0 端の表示を先頭に移動しない

表示指定 : 0 表示に反映しない 1 表示に反映する（デフォルト）

## □ 説明

NeoPixel の表示を指定した方向にシフトします。

表示指定に 1 を指定した場合、直ちに表示に反映されます。

表示指定に 0 を指定した場合、設定のみで表示には反映されません。NPSHOW コマンドにて表示に反映することができます。

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Neopixel not start : NeoPixel が利用開始されていない

Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

要見直し

LED16 個搭載 NeoPixel リングを使って青い軌跡を回転表示させる

```
10 'Neopixel(1)
20 NPBEGIN 0,16
30 NPCLS
40 FOR I=0 TO 7
50 NPRGB I,0,0,(2<<I)-1
60 NEXT I
70 NPSHIFT 1
80 WAIT 50
90 GOTO 70
```

実行結果





## 6.80 NPLEVEL NeoPixel 輝度補正值設定

## □ 書式

NPLEVEL 輝度

## □ 引数

輝度 : 0 ~ 3

## □ 説明

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Neopixel not start : NeoPixel が利用開始されていない

Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

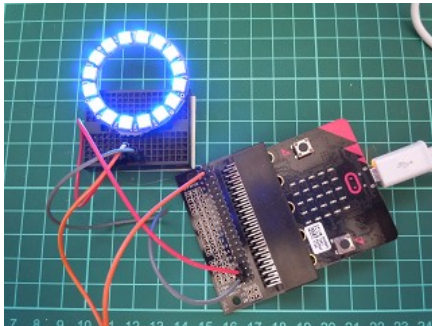
LED を青色で繰り返し強弱表示する

```

10 NPBEGIN 0,16
20 FOR I=0 TO 15
30 POKE MEM+I,RGB8(0,0,3)
40 NEXT I
50 L=0:D=1
60 NPLEVEL L
70 NPPUT 0,MEM,16,1,1
80 WAIT 200
90 L=L+D:IF L=4 D=-1 L=3 ELSE IF L=-1 D=1 L=0
100 GOTO 60

```

実行結果



## 6.81 RGB 15 ビットカラーコードの取得

## □ 書式

RGB(R, G, B)

## □ 引数

R : 0 ~ 31

G : 0 ~ 31

B : 0 ~ 31

## □ 戻り値

15 ビット色コード

## □ 説明

RGB の各色コードから 15 ビット色コードを計算し、その値を返します。

15 ビットの色コード、構成は次の通りです。

RRRRRGGGGGBBBBB

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Illegal value : 引数に範囲外の値を指定した

Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

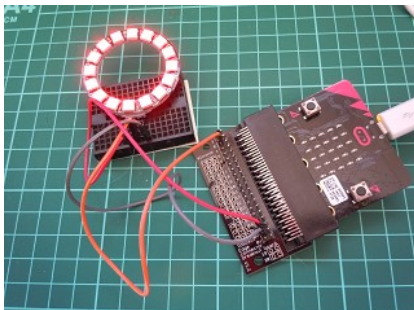
LED を赤色で繰り返し強弱表示する

```

10 NPBEGIN 0,16
20 FOR I=0 TO 15
30 @(I)=RGB(1,0,0)
40 NEXT I
50 L=0:D=1
60 NPLEVEL L
70 NPPUT 0,ARRAY,16,2,1
80 WAIT 200
90 L=L+D:IF L=4 D=-1 L=3 ELSE IF L=-1 D=1 L=0
100 GOTO 60

```

実行結果



## 6.82 RGB 8 ビットカラーコードの取得

## □ 書式

RGB(R, G, B)

## □ 引数

R : 0 ~ 7

G : 0 ~ 7

B : 0 ~ 3

## □ 戻り値

8 ビット色コード

## □ 説明

RGB の各色コードから 8 ビット色コードを計算し、その値を返します。

8 ビットの構成は次の通りです。

RRRGGBBB

## □ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Illegal value : 引数に範囲外の値を指定した

Overflow : 指定した数値が -32768 ~ 32767 を超えている

## □ 利用例

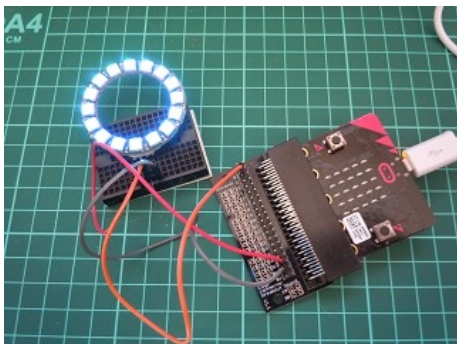
LED をシアン色で繰り返し強弱表示する

```

10 NPBEGIN 0,16
20 FOR I=0 TO 15
30 POKE MEM+I,RGB8(0,1,1)
40 NEXT I
50 L=0:D=1
60 NPLEVEL L
70 NPPUT 0,MEM,16,1,1
80 WAIT 200
90 L=L+D:IF L=4 D=-1 L=3 ELSE IF L=-1 D=1 L=0
100 GOTO 60

```

## 実行結果



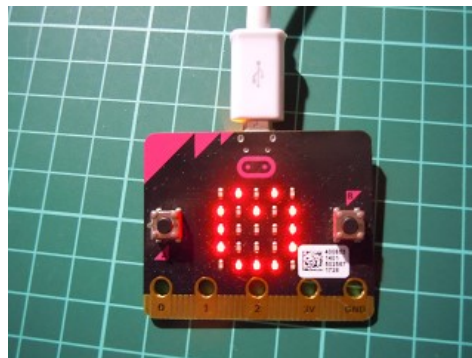
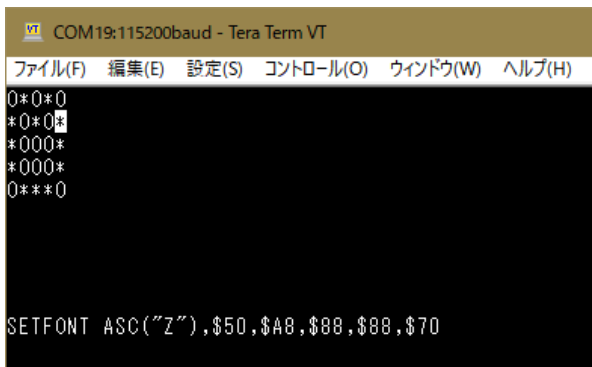
## 7. 応用プログラム

### 7.1 フォントエディタ

後田さんが作成・公開いたしました LED マトリックス用フォントエディタです。  
(5 行、75 行にカーソル表示の追加を行っています)

#### 使い方

- ・カーソルキー [←][→][↑][↓]でカーソルを移動します。
- ・[スペース]キーでカーソル位置のドットをセット・リセットします。  
画面の表示に連動して、LED マトリックスの LED も点灯、消灯します。
- ・画面上に表示されている SETFONT.. をコピーして利用することができます。



#### プログラムソース

```

1 'PCG EDIT
5 ATTR 0:COLOR 7
10 CLS:CLS 1:CLV:LET @(20)=79,42
20 FOR I=0 TO 4:FOR J=0 TO 4
30 LOCATE I,J: ?CHR$(@(20));:NEXT J:NEXT I:GOSUB 500
40 K= INKEY()
50 X=X+(K=131)*(X<4)-(K=130)*(X>0)
60 Y=Y+(K=128)*(Y<4)-(K=129)*(Y>0)
70 LOCATE X,Y:C=VPEEK(X,Y)
75 ATTR 2: ?CHR$(C):LOCATE X,Y:WAIT 100:ATTR 0: ?CHR$(C):LOCATE X,Y
80 IF K=32:IF C= @(20) P= @(21) ELSE IF C= @(21) P= @(20)
90 IF K<>32 GOTO 40
100 ?CHR$(P);
110 FOR I=0 TO 4:D=0
120 FOR J=0 TO 4:C=VPEEK(J,I)
130 IF C= @(21) D=D+(1<<(7-J))
140 NEXT J:@(I)=D
150 NEXT I
160 GOSUB 500:GOTO 40
500 LOCATE 0,10: ?"SETFONT ASC(";CHR$(34,90,34,41);
510 FOR I=0 TO 4: ?", "$";HEX$( @(I),2);:NEXT I
520 SETFONT ASC("Z"),@(0),@(1),@(2),@(3),@(4)
530 MSG TOP,0,"Z"
540 RETURN

```

プログラムを終了した後、画面上の文字表示が反転している場合は、次のコマンドで元に戻すことができます。

```
ATTR 0
OK
```