

別冊『システム変数 と システム関数』

10 .	基本データ.....	3
(a)	データ型の種別と判別.....	3
(b)	定義済みの判定と消去.....	3
11 .	コマンドライン引数.....	3
(a)	コマンドライン引数の取得と返却.....	3
12 .	ファイル入出力.....	4
(a)	オープンとクローズ.....	4
(b)	テキストデータの入出力.....	4
(c)	バイナリデータの入出力.....	4
(d)	ファイル位置とバッファ.....	4
13 .	データ表示 (印刷出力).....	5
(a)	色 (ANSI Color).....	5
(b)	文字列の変書式.....	5
(c)	簡易表示と詳細表示.....	5
(d)	書式付き出力.....	5
14 .	文字列操作.....	5
(a)	文字列の比較.....	5
(b)	文字列の検索.....	6
(c)	文字列の置換.....	6
(d)	改行等の削除.....	6
(e)	部分文字列等.....	6
(f)	文字列の配列化.....	6
15 .	データ操作.....	7
(a)	データの変換.....	7
(b)	データの判定.....	7
(c)	データの消去 (未定義化).....	7
(d)	データの複製.....	7
(e)	サイズの計測.....	7
(f)	ソートの実行.....	7
(g)	配列化と配列値.....	8
16 .	ネットワーク.....	9
(a)	典型的な処理フロー (UDPとTCP).....	9
(b)	ソケットの作成.....	9
(c)	回線の接続と切断.....	10
(d)	データの送信と受信.....	10
(e)	IPアドレスとポート番号.....	10
17 .	日付(Date)と時刻(Time).....	11
(a)	現在時刻の取得.....	11
(b)	UNIX時刻 ⇔ 数 値 (年月日 時分秒).....	11
(c)	UNIX時刻 ⇔ 文字列 (年月日 時分秒).....	11

(d)	うるう年の判定.....	12
(e)	時刻構造体.....	12
18 .	数学関数と整数 & 実数.....	13
(a)	三角関数と双曲関数.....	13
(b)	角度変換と座標変換.....	13
(c)	指数関数と対数関数.....	13
(d)	ベッセル関数.....	14
(e)	統計関数と乱数発生.....	14
(f)	整数値と実数値.....	14
(g)	無限大と非数値.....	14
19 .	プロセスと割り込み.....	15
(a)	プロセス関数.....	15
(b)	割り込み制御.....	15
20 .	メモリ管理.....	15
(a)	取得と解放.....	15
(b)	G C 実行.....	15

10 . 基本データ

基本の定数

関連定数 : **TRUE**<I> | **FALS**<I>
関連定数 : **NULL**<P>
関連定数 : **VER** <I>

真の値 (整数の 1) | 偽の値 (整数の 0)
ポインタ **NULL** (= **システム関数の共通エラー戻り値**)
バージョン番号 (=インタプリタのバージョン番号)

(a) データ型の種別と判別

名前	説明
I = type(*)	引数型の判定。戻り値={ 'U' 'S' 'I' 'D' 'P' 'X' 'A' }
I = is { null true fals str int dbl ptr func array } (*)	引数型が { NULL TRUE FALS 文字列 整数 実数 ポインタ 関数 配列 or 構造体 } かの判定。戻り値= { TRUE FALS }

(b) 定義済みの判定と消去

名前	説明
I = isdef(*)	引数「識別子」の定義済み判定。戻り値= { TRUE FALS }
void = erase(*)	引数「識別子」の消去 (未定義化) 。戻り値=なし

11 . コマンドライン引数

基本の定数

関連定数 : **argc** <I> | **argv** <A>
関連定数 : **argr** <I>
関連定数 : **cmd** <S> | **args** <S>
関連定数 : **env** <A>

コマンドライン引数の個数と配列 (C 言語互換)
shift() 可能なコマンドライン引数の残数 (自動更新)
argv[0] | argv[1] ~ argv[argc-1] を連結した文字列
環境変数の配列※

※ 例えば、環境変数 PATH=/usr/bin であった場合 **env**["PATH"]=" /usr/bin" となる。

(a) コマンドライン引数の取得と返却

名前	説明
S = shift ([*]) S = opt shift ([*])	呼び出し毎に argv[1] から順に取得 shift() に同じ。ただし、オプションの場合のみ取得
I = un shift ()	次の取得位置を 1 つ前に戻す (戻り値=次の取得位置)
※ shift() と opt shift() は、取得するデータが無い場合はパラメータ * (省略時 NULL) を戻す。 ※ オプションとは '-' 文字で始まるコマンドライン引数。	

12 . ファイル入出力

基本の定数

関連定数 : <code>stdin</code> <code>stdout</code> <code>stderr</code> <P>	標準入力 標準出力 標準エラー出力
関連定数 : <code>:code</code> <code>:data</code> <P>	コード領域 データ領域 (通常ファイル時に有効)

(a) オープンとクローズ

名前	説明
P = open (<code>FILE</code> , Mode) I = close (<code>FILE</code>)	ファイルの明示的オープン (通常不要) 戻り値 = <code>FILE</code> ※1 ファイルの明示的クローズ (通常不要)
※1 Mode<S> は、C 言語 fopen(3) 互換のモード文字列。	

(b) テキストデータの入出力

名前	説明
I = getc ([<code>FILE</code> ,]) S = gets ([<code>FILE</code> ,]) S = getn ([<code>FILE</code> ,] Max)	ファイルから1字 1行の読み込み (省略時 <code>stdin</code>)
I = putc ([<code>FILE</code> ,] Chr) I = puts ([<code>FILE</code> ,] Str) I = putn ([<code>FILE</code> ,] Str, Max)	ファイルへ 1字 1行の書き込み (省略時 <code>stdout</code>)
I = ungetc ([<code>FILE</code> ,] Chr) I = ungets ([<code>FILE</code> ,] Str) I = ungetn ([<code>FILE</code> ,] Str, Max)	ファイルへ 1字 1行を元に戻す (省略時 <code>stdin</code>)
※ 文字はChr<I>、文字列はStr<S>、最大文字数はMax<I>で指定。	

(c) バイナリデータの入出力

名前	説明
(Ptr, Cnt) = read ([<code>FILE</code> ,] Cnt)	ファイルから指定バイト数の読み込み (省略時 <code>stdin</code>)
Cnt = write ([<code>FILE</code> ,] Ptr, Cnt)	ファイルへ 指定バイト数の書き込み (省略時 <code>stdout</code>)
※ Cnt<I>は、要求バイト数と成功バイト数。Ptr<P>は、対象データの保存場所アドレス。	

(d) ファイル位置とバッファ

名前	説明
I = getpos (<code>FILE</code>)	ファイル位置の取得 (先頭 = 0) ※1
I = setpos (<code>FILE</code> , I)	ファイル位置の設定 (絶対値指定 : 先頭 = 0 / 末尾 = -1) ※1
I = movpos (<code>FILE</code> , I)	ファイル位置の移動 (相対値指定 : 進む = 正 / 戻る = 負) ※1
I = rewind (<code>FILE</code>)	ファイル先頭位置へ移動 [= setpos (<code>FILE</code> , 0)] ※1
I = flush (<code>FILE</code>)	ファイルバッファのフラッシュ (<code>NULL</code> 指定 → 全ファイル)
※1 戻り値は、関数実行後のファイル位置。	

13 . データ表示 (印刷出力)

(a) 色 (ANSI Color)

`{C | R} - {BLA | RED | GRE | YEL | BLU | MAG | CYA | WHI}` (S) ANSI Color 文字列 (強調色) ※
`{c | r} - {bla | red | gre | yel | blu | mag | cya | whi}` (S) ANSI Color 文字列 (通常色) ※
`C_DEF | c_def` (S) デフォルトに戻す [`C_DEF == c_def`]
※ 色の場所 (2種) ⇒ 文字カラーの指定(C) | 反転カラーの指定(R)
色の種別 (8種) ⇒ 黒(BLA) | 赤(RED) | 緑(GRE) | 黄(YEL) | 青(BLU) | 紫(MAG) | 水(CYA) | 白(WHI)
※ 出力先に応じて、カラー又はモノクロが自動選択される。【例】画面表示⇒カラー/その他⇒モノクロ

(b) 文字列の変書式

`FMT_S2S` (S) 文字列 → 文字列 への変換書式 【初期値 = "%s"】 ※
`FMT_I2S` (S) 整数型 → 文字列 への変換書式 【初期値 = "%d"】 ※
`FMT_D2S` (S) 実数型 → 文字列 への変換書式 【初期値 = "%+f"】 ※
`FMT_P2S` (S) ポインタ (NULL以外) → 文字列 への変換書式 【初期値 = "%p"】 ※
`FMT_N2S` (S) NULL (NULLのみ) → 文字列 への変換書式 【初期値 = "NULL"】 ※
※ デフォルトの変換書式。 `p()` 関数や、"`${変数}`" 展開時などに利用される。

(c) 簡易表示と詳細表示

システム関数	説明
<code>void = p(*, ...)</code> <code>void = d(*, ...)</code>	値の簡易表示 値の詳細表示※ ← 内部データのダンプ表示
<code>x ← 変数名のみを記述</code>	変数 <code>x</code> 値の簡易表示 ← <code>p(x)</code> に同じ
※ 詳細表示: 識別ID {スコープ種別(G=Global S=Static L=Local)+識別番号}、名前、データ型、データ属性、データ値を表示する。	

(d) 書式付き出力

システム関数	説明
<code>I = print (FMT,...)</code> <code>I = eprint(FMT,...)</code> <code>I = fprintf(FILE,FMT,...)</code> <code>S = sprintf(FMT,...)</code>	標準 出力用の <code>print</code> 文 標準エラー出力用の <code>print</code> 文 = <code>fprintf(stderr,FMT,...)</code> ファイル 出力用の <code>print</code> 文 文字列 出力用の <code>print</code> 文
<code>void = dying(FMT,...)</code>	{ <code>eprint(FMT,...); exit(1);</code> } 等価
※ <code>FMT(S)</code> は、C言語 <code>printf()</code> 互換の書式文字列。2進数出力 <code>%b %B</code> 、及び、数値出力 <code>%v %V</code> が指定可能。(整数又は実数を、型に応じて <code>%d %e %f</code> を自動選択)	

14 . 文字列操作

(a) 文字列の比較

システム関数	説明
<code>I = strcmp (S1,S2)</code> <code>I = strcmpi (S1,S2)</code>	文字列 <code>S1</code> と <code>S2</code> の大小比較 ※
<code>I = strncmp (S1,S2,I)</code> <code>I = strncmpi (S1,S2,I)</code>	文字列 <code>S1</code> と <code>S2</code> の大小比較 (最大 <code>I</code> 文字まで) ※
<code>I = startwith(S1,S2)</code> <code>I = endwith (S1,S2)</code>	文字列 <code>S1</code> が <code>S2</code> で開始するか判定。戻り値= {TRUE FALSE} 文字列 <code>S1</code> が <code>S2</code> で終了するか判定。戻り値= {TRUE FALSE}
※ 戻り値は、C言語 <code>strcmp()</code> 互換。 <code>i</code> 付き関数は Ignore Case 版。(大小文字の区別なし)	

(b) 文字列の検索

システム関数	説明
I = strchr(S,Chr) str r chr(S,Chr)	文字 Chr の {順方向 逆 方向} 検索
I = strstr(S,Str) str r str(S,Str)	文字列 Str の {順方向 逆 方向} 検索
(Is,Ie) = strreg(S,/RE/)	正規表現 /RE/ のマッチ位置検索 {Is=始点、Ie=終点}
※ いずれの関数も、文字列 S 内の位置が戻り値。(先頭=0)	

(c) 文字列の置換

システム関数	説明
S = ssub(S,V1,V2) gsup(S,V1,V2)	文字列 S 内の V1 を V2 に { 1回 全部 } 置換
S = evale _{sc} (S) S = rva _{le} sc(S)	ESCシーケンスの順方向変換 (例: "\n" → 0x0a) ESCシーケンスの逆方向変換 (例: 0x0a → "\n")
S I = upper(S I) lower(S I)	引数の {大文字 小文字} 変換
※ パラメータ V1 は、文字 (列)、又は、正規表現。パラメータ V2 は、文字 (列)。	

(d) 改行等の削除

システム関数	説明
S = chop (S)	{ 行末 } の改行コードを 1 組削除します。
S = h strip(S) t strip(S) strip(S)	{行頭 行末 両方} の空白コードを全部削除します。
※ 改行コードは、Win/Mac/Unix 形式に全対応。空白コードは、isspace() で判定。	

(e) 部分文字列等

システム関数	説明
I = subchr(S,Ic)	文字列 S の位置 Ic の文字を戻します。(= S[Ic])
S = substr(S,Is,Ie)	文字列 S の位置 Is~Ie の部分文字列を戻します。
※ 位置の指定方法 (先頭から指定する場合) : 先頭 = 0, 1, 2, ...	
※ 位置の指定方法 (末尾から指定する場合) : 末尾 = -1, -2, -3, ...	

(f) 文字列の配列化

システム関数	説明
A = split (S,V [,I]) A = split 0 (S,V [,I])	文字列 S を V で区切った各要素で配列化 (//) ただし、空要素はスキップ。(P acked形式)
A = scan (S,V [,I]) A = p scan (S,V [,I])	文字列 S を V でマッチ各要素で配列化します。 (//) ただし、空要素はスキップ。(P acked形式)
※ いずれの関数も、パラメータ V には、文字、文字列、又は、正規表現が指定できます。又、戻り値 A は、新規に生成された 1 次元配列です。(添字は 0 ~)	
第 3 パラメータ (I) 指定時は、その添字値に対応する要素 (S) のみを戻します。	

15 . データ操作

(a) データの変換

システム関数	説明
I = int (*) D = dbl (*) I D = atov (*) I = bin(S) oct(S) dec(S) hex(S)	パラメータの整数化 【 別名 atoi() 】 パラメータの実数化 【 別名 atof() 】 パラメータの数値化 (型を自動判定し整数又は実数を返す) 文字列 S を {2 8 10 16} 進数と見なして整数化
S = str (*) P = ptr (*) /RE/ = regexp(S)	パラメータの文字列化 パラメータのポインタ化 正規表現 S のコンパイル

(b) データの判定

システム関数	説明
I = is {alnum alpha ascii blank cntrl digit graph lower print punct space upper xdigit} (I S)	{文字 文字列} 種別判定。戻り値= {TRUE FALS}

(c) データの消去 (未定義化)

システム関数	説明
U = erase (*)	定義済み {変数 関数} の未定義化 (消去)

(d) データの複製

システム関数	説明
* = dup (*)	データの複製 (主に、配列データ用)

(e) サイズの計測

システム関数	説明
I = size (*)	パラメータのサイズを求める ※
I = strlen(S)	文字列 S のサイズ (長さ) を求める
※ 関数 size() の戻り値は、文字列→長さ (バイト数)、整数型&実数型&ポインタ→表現ビット数、関数型→演算ノード数、配列型→要素数となる。	

(f) ソートの実行

システム関数	説明
A = sort([X,] V,...)	パラメータ V を昇順にソートします (戻り値=ソート済み配列)
A = rsort([X,] V,...) A = argsort(V) argrsort(V)	パラメータ V を降順にソートします (戻り値=ソート済み配列) {昇順ソート 降順ソート} のインデックス値 ※1
※ 比較関数 X を指定しない場合は、パラメータの単純ソートとなります。この場合、パラメータ V は、{文字列 整数 実数} 又はそれらを含む配列が指定できます。比較関数 X を指定した場合は、比較関数による一般ソートとなります。この場合、パラメータ V は自由に設定出来ます。なお、比較関数 X は、2つの引数を取り比較結果を数値の符号で戻す関数となります。(C言語 qsort(3) の比較関数互換。)	

(g) 配列化と配列値

システム関数	説明
Ao = vals(Ai) rvals(Ai)	配列 Ai の各要素値を要素とする、1次配列 Ao を生成します
Ao = keys(Ai) rkeys(Ai)	配列 Ai の各添字値を要素とする、1次配列 Ao を生成します
Ao = tags(Ai) rtags(Ai)	配列 Ai の添字+要素を要素とする、1次配列 Ao を生成します
A = append(A,V)	配列 A の最後に値 V を追加します。
I = ndim (Ai)	(ハッシュ要素を含まない) 配列 Ai の次元数を求めます
Ao = shape(Ai)	(ハッシュ要素を含まない) 配列 Ai の形 状を求めます
関数 vals(),keys(),tags() は入力配列 Ai を昇順にサーチして出力配列 Ao の生成処理を行います。 一方、関数 rvals(),rkeys(),rtags() は降順にサーチして生成処理を行います。	

16 . ネットワーク

(a) 典型的な処理フロー (UDPとTCP)

参考：UDPの処理フロー

処理フロー	クライアント側	サーバー側
0. 準備	不要 (自動)	<code>sock_wait = udp_socket(IP, PORT)</code>
2. 送信 2. 受信	<code>tx_udp(IP, PORT, DATA)</code> <code>(IP, PORT, DATA) = rx_udp()</code>	

【例】UDP echo クライアント (ソケットは**自動**で作成され、その値が **UDP_SOCKET** に設定&利用される。)

```
tx_udp( 127.0.0.1, "echo", "Hello!!" )    # UDPパケットの送信 (localhost)
( ip, port, buf ) = rx_udp()               # UDPパケットの受信
print("%s\n",buf)                          # 受信メッセージの表示
```

【例】UDP echo サーバー (ソケットは**手動**で作成され、その値が **UDP_SOCKET** に設定&利用される。)

```
sock_wait = udp_socket( 0, "echo" )        # UDPソケットの作成 (0→自動)
while( TRUE ){
    ( ip, port, buf ) = rx_udp()           # UDPパケットの受信
    tx_udp( ip, port, buf )               # UDPパケットの送信 (返答)
}
```

参考：TCPの処理フロー

処理フロー	クライアント側	サーバー側
0. 準備	不要 (自動)	<code>sock_wait = tcp_socket(IP, PORT)</code>
1. 接続	<code>connect(IP, PORT)</code>	<code>listen(sock_wait)</code>
2. 送信 2. 受信	<code>tx_tcp(DATA)</code> <code>DATA = rx_tcp()</code>	
3. 切断	<code>disconn()</code>	

【例】TCP echo クライアント (ソケットは**自動**で作成され、その値が **TCP_SOCKET** に設定&利用される。)

```
connect( 127.0.0.1, "echo" )              # TCPサーバーへ接続 (localhost)
tx_tcp( "Hello!!" )                      # TCPデータの送信
buf = rx_tcp()                          # TCPデータの受信
print("%s\n",buf)                        # 受信メッセージの表示
```

【例】TCP echo サーバー (接続の待ち受けは**手動**で作成の `sock_wait` の値を利用する。接続が確立すると、接続済みの新しいソケットが**自動**で作成され、**TCP_SOCKET** に設定&利用される。)

```
sock_wait = tcp_socket( 0, "echo" )       # TCPソケットの作成 (0→自動)
while( listen(sock_wait) ){               # 接続の待ち受け&接続
    buf = rx_tcp()                       # TCPデータの受信
    tx_tcp(buf)                         # TCPデータの送信 (返答)
    disconn()                           # TCP回線の切断
}
```

(b) ソケットの作成

システム関数	説明
<code>sock = udp_socket(ip,port)</code> <code>sock = tcp_socket(ip,port)</code> <code>sock = raw_socket(ip)</code>	{UDP TCP RAW} ソケットを作成し、{ip port} を設定する。なお、数値0を指定すると自動選択 (お任せ) となる。 戻り値=ソケット番号
※ ip は整数、又は、ホスト名を表す文字列。 port は整数、又は、サービス名を表す文字列。 ※ソケット sock の値は、{ UDP_SOCKET TCP_SOCKET RAW_SOCKET } にも自動設定される。	

関連定数 : **UDP_SOCKET(I)** UDPデフォルトソケット値 (自動設定) ※
関連定数 : **TCP_SOCKET(I)** TCPデフォルトソケット値 (自動設定) ※
関連定数 : **RAW_SOCKET(I)** RAWデフォルトソケット値 (自動設定) ※
 ※ {UDP | TCP | RAW} 関連関数が、ソケット指定の省略時に利用するソケット番号。初期値は0 (無効=ソケット値未設定)、この状態で利用されると新しいソケットが自動で作成&設定される。又、自動&手動に関係なく、新しいソケットが作成される毎にソケット値が自動設定 (自動更新) される。

(c) 回 線の接続と切断

システム関数	説明
<code>sock = listen ([sock])</code>	ソケット <code>sock</code> を利用して、TCP回線の接続を待ち受ける。戻り値=新たに作成された接続済みのソケット番号。
<code>sock = connect([sock],ip,port)</code>	ソケット <code>sock</code> を利用して、相手 <code>ip&port</code> にTCP回線の接続を行う。戻り値=接続済みのソケット番号。
<code>disconn([sock])</code>	ソケット <code>sock</code> を利用して、TCP回線の切断を行う。
※ <code>ip</code> は整数、又は、ホスト名を表す文字列。 <code>port</code> は整数、又は、サービス名を表す文字列。 ※ ソケット <code>sock</code> 省略時には、デフォルトソケット TCP_SOCKET が使用される。	

(d) データの送信と受信

システム関数	説明
<code>len = tx_udp([sock],ip,port,buf[,len])</code> <code>len = tx_tcp([sock],buf[,len])</code>	送信先に <code>buf(S P)</code> データを <code>len(I)</code> バイト {UDP TCP} にて送信する。 <code>len</code> 省略時は全データ。戻り値=送信バイト数。
<code>(ip,port,buf,len) = rx_udp([sock])</code> <code>(buf,len) = rx_tcp([sock])</code>	自分宛の {UDP TCP} パケットを受信する。戻り値=送信元 <code>ip&port</code> (UDP)、受信データ&バイト数。
<code>time = ping([sock],ip)</code>	指定 <code>ip</code> に PING パケットを送信&受信します。戻り値=往復時間。[単位=秒]
※ <code>ip</code> は整数、又は、ホスト名を表す文字列。 <code>port</code> は整数、又は、サービス名を表す文字列。 ※ソケット <code>sock</code> 省略時は、{ UDP_SOCKET TCP_SOCKET RAW_SOCKET } が使用される。又、バイト数 <code>len</code> は、ペイロードデータ部のバイト数。	

タイムアウト値

関連定数 : **UDP_TIMEOUT(D)** UDP関連関数のタイムアウト値 (単位 [秒]) ※
関連定数 : **TCP_TIMEOUT(D)** TCP関連関数のタイムアウト値 (単位 [秒]) ※
関連定数 : **RAW_TIMEOUT(D)** RAW関連関数のタイムアウト値 (単位 [秒]) ※
 ※初期値は INF (無限=ブロッキング動作)。0 を設定すると待ち時間 0 秒 (ノンブロッキング動作) となり、3.5 を設定すると 3.5 秒後にタイムアウトする。

(e) I Pアドレスとポート番号

システム関数	説明
<code>(ip,port,type) = lsock(sock)</code> <code>(ip,port,type) = rsock(sock)</code>	ソケット <code>sock</code> の {ローカル側 リモート側} の IPアドレス、 <code>port</code> 番号、 <code>type</code> (プロトコル種別を表す文字列) を戻す。

システム関数	説明
<code>S = ip2host(I)</code> <code>I = host2ip(S)</code>	IPアドレス ⇒ ホスト名 ホスト名 ⇒ IPアドレス
<code>S = ip2str(I)</code> <code>I = str2ip(S)</code>	IPアドレス ⇒ 文字列 ※1 文字列 ⇒ IPアドレス
<code>S = port2serv(I,type)</code> <code>I = serv2port(S,type)</code>	ポート番号 ⇒ サービス名 ※2 サービス名 ⇒ ポート番号 ※2
※1 変換書式は、システム変数 FMT_IP2STR に従う。 ※2 <code>type</code> は、プロトコル種別を表す文字列 {"udp" "tcp"}、又は、NULL (指定無し)。	

関連定数: [FMT_IP2STR\(S\)](#)

IPアドレスの文字列化用書式 (初期値 = "%3d.%3d.%3d.%3d")

17 . 日付(Date)と時刻(Time)

【 参 考 】

- ・ **UNIX時刻** とは、1970-01-01 00:00:00 からの経過時間 [秒] ⇒ スクリプトの計算処理用
- ・ **UTC** は、**世界標準時 (協定世界時)** ⇒ 世界時での入出力用
- ・ **LTZ** は、**Local Time Zone (地方標準時)** (例：日本標準時) ⇒ 地方時での入出力用

タイムゾーン&時差

関連定数 : **LTZ** <S>

関連定数 : **LTZ_OFFSET** <I>

ローカルタイムゾーンを表す文字列

LTZ時刻 - **UTC時刻**の値【単位=秒】

日本の例 : "JST"

日本の例 : +32400

※ **LTZ** と **LTZ_OFFSET** の初期値は、インタプリタ起動時に OS より取得&設定されます。**LTZ_OFFSET** の値は、正の値=進み時差/負の値=遅れ時差となります。この値を変更することで、スクリプト内部の時差を任意の値に設定可能です。(なお、この値を変更しても OS には影響を与えません。)

デフォルト変換書式

関連定数 : **FMT_SDATE**(S)

関連定数 : **FMT_STIME**(S)

年月日 ⇔ 文字列の変換書式 (初期値="%Y-%m-%d")

時分秒 ⇔ 文字列の変換書式 (初期値="%H:%M:%S")

※

※

※ C言語 {strftime(3) | strptime(3)} 互換の変換書式 (文字列) です。任意の値に設定可能です。

(a) 現在時刻の取得

システム関数	説明
D = time()	現在の UNIX時刻 を取得します。【単位=実数秒】 ※1
A = ptime()	3種類のプロセス時刻を取得します。 ※2
※1 UNIX時刻 は、単一の数値で表現できる世界共通の時刻。(時差、夏時間、うるう秒全てなし。)	
※2 戻り値 A は構造体。メンバー名と値の意味は以下の通り。【単位=実数秒】	
A.ptime = スクリプト開始 ~ 関数 ptime() 実行までの経過時間 (Process時間)	
A.utime = Process時間のうちユーザーモード実行時間 (UsrCPU 時間)	
A.stime = Process時間のうちカーネルモード実行時間 (SysCPU 時間)	

(b) **UNIX時刻** ⇔ 数 値 (年月日 | 時分秒)

システム関数	説明
(y,m,d[,h,m,s]) = t2\$ymd(D)	年,月,日[,時,分,秒] ← UNIX時刻 ※1
(h,m,s) = t2\$hms(D)	時,分,秒 ← UNIX時刻 ※1
D = \$ymd2t(y,m,d[,h,m,s] STR)	UNIX時刻 ← 年,月,日[,時,分,秒] ※1※2
D = \$hms2t(h,m,s STR)	UNIX時刻 ← 時,分,秒 ※1※2
※\$ ⇒ u 又は "" の時 UTC 変換。\$ ⇒ l の時 LTZ 変換。(LTZ_OFFSET調整有り)	
※1 {y m d h m} は、年,月,日,時,分を表す整数。{s} は、秒を表す実数。	
※2 1つの文字列 STR<S> が与えられると、各々システム変数 {FMT_SDATE FMT_STIME} に従って変換	

(c) **UNIX時刻** ⇔ 文字列 (年月日 | 時分秒)

システム関数	説明
S = \$sdate(D)	"YYYY-MM-DD" ← UNIX時刻 ※1
S = \$stime(D)	"HH:MM:SS" ← UNIX時刻 ※1
S = str\$fptime(FMT,D)	"任意文字列" ← UNIX時刻 ※2
D = str\$ptime(FMT,S)	UNIX時刻 ← "任意文字列" ※2
※\$ ⇒ u 又は "" の時 UTC 変換。\$ ⇒ l の時 LTZ 変換。(LTZ_OFFSET調整有り)	
※1 変換書式は、システム変数 {FMT_SDATE FMT_STIME} に従う。	
※2 FMT<S> は、各々C言語 {strftime(3) strptime(3)} 互換の変換書式。	

(d) うるう年の判定

システム関数	説明
I = isleap(I)	西暦年(I)のうるう年判定 (戻り値=TRUE/FALS)

(e) 時刻構造体

システム関数	説明
A = t2\$tm(D) D = \$tm2t(A)	時刻構造体 tm ← UNIX時刻 UNIX時刻 ← 時刻構造体 tm
A = tm_norm(A)	時刻構造体 tm の正規化 (例: 3時65分→4時05分)
※\$ ⇒ u 又は "" の時 UTC 変換。\$ ⇒ l の時 LTZ 変換。 (LTZ_OFFSET調整有り)	

時刻構造体 tm メンバー (9種)

tm.year = 年 (西暦4桁)
tm.mon = 月 (01~12)
tm.day = 日 (01~31)
tm.hour = 時 (00~23)
tm.min = 分 (00~59)
tm.sec = 秒 (00.0~60.0)

tm.wday = 曜日 (00~06) 整数 → 日曜0~土曜6
tm.yday = 年日 (0~365) 整数 → 正月0~年末364 | 365
tm.isdst = 夏時間の有効性 (TRUE/FALS)

注意 1) 型は、sec は実数。他は全て整数 (TRUE/FALSを含む)。

注意 2) wday と yday は、システム関数出力時に有効となる。(システム関数入力時は設定不要。)

注意 3) 黄色の数値は、うるう秒又はうるう年の範囲。

18 . 数学関数と整数&実数

基本定数

関連定数 : SYS_ANGLE <D>	システム関数の角度単位 (RAD DEG)
関連定数 : M_PI <D>	円周率 (M_PI = 3.141593...)
関連定数 : M_E <D>	自然対数の底 (M_E = 2.718282...)
関連定数 : INF <D>	無限大 (INFINITY)
関連定数 : NAN <D>	非 数 (Not A Number)
	【参考：実数型】
関連定数 : {INT DBL PTR}_SIZE <I>	{整数 実数 ポインタ} のビット数 例 : 64 [bit]
関連定数 : DBL_{MANT EXPO}_SIZE <I>	実数の {仮数部 指数部} のビット数 例 : 52,11 [bit]
関連定数 : INT_{MAX MIN} <I>	整数の {最大値 (正) 最小値 (負) }
関連定数 : DBL_{MAX MIN} <D>	実数の {最大値 (正) 最小値 (負) }
関連定数 : INT_{QUANT EPSILON} <I>	整数の {正の最小値 イプシロン値}
関連定数 : DBL_{QUANT EPSILON} <D>	実数の {正の最小値 イプシロン値}

(a) 三角関数と双曲関数

システム関数	説明
D = sin (D) cos (D) tan (D)	{正弦 余弦 正接} 関数 ※
D = asin (D) acos (D) atan (D)	{正弦 余弦 正接} 逆関数 ※
D = atan2(Dy,Dx)	{正接} 逆関数 (2変数指定) ※
D = sinh(D) cosh(D) tanh(D)	双曲線 {正弦 余弦 正接} 関数 ※
D = asinh(D) acosh(D) atanh(D)	双曲線 {正弦 余弦 正接} 逆関数 ※

※ 角度単位は RAD です。 (**SYS_ANGLE** = RAD | DEG で設定可能です。)

(b) 角度変換と座標変換

システム関数	説明
D = deg(D) rad2deg(D)	Deg ← Rad
D = rad(D) deg2rad(D)	Rad ← Deg
(Deg,Min,Sec) = rad2dms(D)	度分秒 ← Rad (Deg と Min は整数 Sec は実数)
(Deg,Min,Sec) = deg2dms(D)	度分秒 ← Deg (//)
D = dms2rad(Deg,Min,Sec)	Rad ← 度分秒 (Deg と Min は整数 Sec は実数)
D = dms2deg(Deg,Min,Sec)	Deg ← 度分秒 (//)
(X,Y,Z) = xrot(X,Y,Z,θ)	X 座標軸を +θ 回転 ※
(X,Y,Z) = yrot(X,Y,Z,θ)	Y 座標軸を +θ 回転 ※
(X,Y,Z) = zrot(X,Y,Z,θ)	Z 座標軸を +θ 回転 ※

※ 角度単位は RAD です。 (**SYS_ANGLE** = RAD | DEG で設定可能です。)

(c) 指数関数と対数関数

システム関数	説明
D = exp(D) exp2(D) exp10(D)	{ e 2 10 } の累乗 【 exp() 別名 = expe() 】
D = log(D) log2(D) log10(D)	{ 底 e 底 2 底 10 } の対数 【 log() 別名 = loge() 】
D = pow(Dx,Dy) mod(Dx,Dy)	Dx の Dy 乗 Dx/Dy の余剰 (= 演算子%)
D = lgamma(D) tgamma(D)	{ loge True } ガンマ関数
D = sqrt(D) cbrt(D)	平方根 立方根
D = lin (D) dbi (D)	リニア値 デシベル値

(d) ベッセル関数

システム関数	説明
$D = j_0(D) \mid j_1(D) \mid j_n(In, Dx)$ $D = y_0(D) \mid y_1(D) \mid y_n(In, Dx)$	第一種ベッセル関数 第二種ベッセル関数

(e) 統計関数と乱数発生

システム関数	説明
$D = \text{dnorm}(X [, AVE, SIG])$ $P = \text{pnorm}(X [, AVE, SIG])$ $X = \text{qnorm}(P [, AVE, SIG])$	正規分布の確率密度の値 (X における確率密度の値) ※3 正規分布の累積確率の値 (X 以下の範囲の確率の値) ※3 $\text{pnorm}()$ の逆関数 (確率が P となる X の値) ※3
$D = \text{max}(V) \mid \text{med}(V) \mid \text{min}(V)$ $I = \text{argmax}(V) \mid \text{argmed}(V) \mid \text{argmin}(V)$	{ 最大値 中央値 最小値 } ※1 { 最大値 中央値 最小値 } のインデックス値 ※1
$D = \text{sum}(V) \mid \text{ave}(V)$ $D = \text{svar}(V) \mid \text{uvar}(V)$ $D = \text{sdev}(V) \mid \text{udev}(V)$ $D = \text{erf}(D) \mid \text{erfc}(D)$	{ 合計値 平均値 } ※1 { 標本 不偏 } 分散 ※1※2 { 標本 不偏 } 標準偏差 ※1※2 { 誤差 余誤差 } 関数
$I = \text{rand}(MAX)$ $D = \text{urand}()$ $D = \text{nrnd}([AVE, SIG])$ $U = \text{seed}(I)$	整数乱数の発生 (戻り値 = 0 以上 MAX 未満) 一様乱数の発生 (戻り値 = 0.0 以上 1.0 未満) 正規乱数の発生 (平均値 = AVE / 標準偏差 = SIG) ※3 乱数シード値の設定
※1 パラメーター V は { 整数、実数、又は、それらを要素とする配列 } の並びです。 ※2 標本値は $1/N$ の式、不偏値は $1/(N-1)$ の式です。 ※3 パラメーター省略時は、標準正規分布 (AVE=0.0 / SIG=1.0) となります。	

(f) 整数値と実数値

システム関数	説明
$I = \text{int} (*)$ $D = \text{dbl} (*)$ $I \mid D = \text{atov} (*)$	パラメータの整数化 【 別名 $\text{atoi}()$ 】 パラメータの実数化 【 別名 $\text{atof}()$ 】 パラメータの数値化 (戻り値の型は自動判定)
$D = \text{abs}(D)$	絶対値
$D = \text{ceil}(D) \mid \text{floor}(D) \mid \text{trunc}(D)$ $D = \text{rint}(D) \mid \text{round}(D)$	実数の丸め込み { 切り上げ 切り下げ 0 の方向 } 実数の丸め込み { 偶数丸め 四捨五入 }
$(Dint, Dfra) = \text{modf}(D)$ $D = Dint + Dfra$ $(Dman, Iexp) = \text{frexp}(D)$ $D = \text{ldexp}(Dman, Iexp)$	実数 D から { 整数=Dint & 小数=Dfra } への分解 { 整数=Dint & 小数=Dfra } から実数 D への合成 実数 D から { 仮数=Dman & 指数=Iexp } への分解 { 仮数=Dman & 指数=Iexp } から実数 D への合成

(g) 無限大と非数値

システム関数	説明
$I = \text{isinf}(D)$ $I = \text{isnan}(D)$	無限大の判定 (戻り値 = TRUE/FALS) 非 数の判定 (戻り値 = TRUE/FALS)

19 . プロセスと割り込み

(a) プロセス関数

システム関数	説明
D = sleep(D) U = pause() U = exit (I)	スリープ の実行 (時間 = D [秒] / 戻り値 = 残時間[秒]) ポーズ の実行 (時間 = 無制限 / 戻り値 = 無し) スクリプトの終了 (終了値 I)
I = system(S) A = syscmd(S) S = 'S'	外部コマンド S の実行 (戻り値 = コマンドの終了値) 外部コマンド S の実行 (戻り値※) 外部コマンド S の実行 (戻り値 = 標準出力)
※ 戻り値 A は、A.stat<I>=終了値 A.stdout<S>=標準出力 A.stderr<S>=標準エラー出力	
関連定数 : \$\$ <I> PID <I> 関連定数 : \$? <I>	自己プロセスの I D 値 (\$\$ == PID) 外部コマンドの終了値

(b) 割り込み制御

システム関数	説明
I = tx_sig(Isig, Ipid)	シグナル (Isig番) をプロセス (Ipid番) に送信 戻り値 = 整数値のゼロ (成功) 又は NULL (失敗)
P = rx_sig(Isig, Func)	シグナル (Isig番) 受信時の処理関数 (Func) を登録 ※ 戻り値 = 設定前の Func (成功) 又は NULL (失敗)
※ Func は、1つの引数を取る任意のユーザー関数です。シグナル受信時に割り込み実行されます。この時、受信したシグナル番号がパラメータとしてセットされます。なお、Func に {SIG_DFL SIG_IGN} を指定すると、シグナル受信時の動作が {既定動作 受信無視} となります。	
関連定数 : SIG {DFL IGN ERR} <P> 関連定数 : NSIG <I> 関連定数 : SIG {ABRT ALRM CHLD CONT FPE HUP ILL INT KILL PIPE QUIT SEGV STOP TERM TSTP TTIN TTOU USR1 USR2} <I>	割り込み制御用 {既定動作 受信無視 エラー(NULL)} シグナルの個数 (0 ~ NSIG-1 が有効) POSIX標準シグナル

20 . メモリ管理

(a) 取得と解放

システム関数	説明
P = alloc(I) U = free (P)	I バイトのメモリ領域を確保し、先頭アドレスを戻す。 関数 alloc() で確保したメモリ領域を解放する。

(b) GC実行

システム関数	説明
U = gc_collect()	ガーベージコレクションを明示的に実行する。 ※
※ 極めて大量のメモリを繰り返し使用&解放する場合等においては、本関数を適宜実行することでメモリ使用量を大幅に削減できる場合があります。(通常は、明示的な実行は不要。)	