基本的なオブジェクト

数値

- 数値を表すオブジェクトです。
- 代入式の右辺や、括弧で囲まれた部分に数式を**中置記法**で記述できます。
- 数式で扱えるデータは、数値と、数値に変換できる値を持つ文字列です。
- 2進(0b)と16進(0x)で0b1100□0xFFのように定数を記述できます。
- 角度は1周を360度とする角度で表します。
- 計算はJavaの倍精度実数□double□で行われます。ただし、一部の関数演算は単精度実数□float□で行われることがあります。
- 定数として、**円周率**を表す**π□PI**が用意されています。
- 数値演算子「+, -, ＊, /, %」は、内部的にそれぞれ□add, sub, mul, div, mod□という命令に変換されて扱われます。論理演算子「==, !=, >, >=, <, ⇐」も、内部的にそれぞれ□eq, ne, gt, ge, lt, le□という命令に変換されて扱われます。

- **+ , - , □, ×, , ÷ : 四則演算。 * （例）「3 ＊ 40」を計算し「120」を表示します□ <code> ラベル！（3 ＊ 40）作る。 </code> * 足す□add□, 引く□sub□, 掛ける□mul□, 割る□div）: 四則演算。命令として使います。 * （例）「3 ＊ 40」を計算し「120」を表示します□ <code> ラベル！（3！ 40 掛ける）作る。 </code> * %: 余り * （例）8を3で割った余りを計算し「2」を表示します□ <code> ラベル！（8 % 3）作る。 </code> * 余り□mod）: 余り。命令として使います。 * （例）8を3で割った余りを計算し「2」を表示します□ <code> ラベル！（8！ 3 余り）作る。 </code> * == , != , ≠, > , >= , ≧, < , ⇐, ≦ : 比較演算。両辺が数値または数値に変換できる文字列の場合は、数値として比較されます。 * （例）「4 > 3」を計算し「[true]□を表示します□ <code> ラベル！（4 > 3）作る。 </code> * sqrt: ルート（√） * （例）「1 + √4」を計算し「3」を表示します□ <code> ラベル□□1□+□sqrt□4□□作る。 </code> * sin , cos , tan: 三角関数。 *（例□□sin(30)□を計算し「0.5」を表示します□ <code> ラベル□□sin□30□□作る。 </code> * asin , acos , atan , atan2: 三角関数の逆関数。 *（例□□arcsin(0.5)□を計算し「30」を表示します□ <code> ラベル□□asin□0.5□□作る。 </code> atan2は、X座標の値にY座標の値をパラメータとして実行します□atanは-90〜90の値を返しますが□atan2は-180〜180の値を返します。 * （例）「(-10, 10)」の座標から正接の逆関数を計算し「135」を表示します□ <code> ラベル□□-10□10□atan2□作る。 </code> * round , ceil , floor: 丸め、切り上げ、切り捨て。 * （例）「0.7」を四捨五入し「1」を表示します□ <code> ラベル□□round□0.7□□作る。 </code> * exp: 指数関数。 *（例□□e^{0.5□を計算し「1.6487212」を表示します□ <code> ラベル□□exp□0.5□□作る。 </code> * log: 底が10の対数。 *（例□□log 100□を計算し「2」を表示します□ <code> ラベル□□log□100□□作る。 </code> * ln: 底がeの対数。 *（例□□ln 100□を計算し「4.6051702」を表示します□ <code> ラベル□□ln□100□□作る。 </code> * pow: べき乗。「2の3乗」は「pow□2,3□□ではなく□□2□3□pow□と書くことに注意してください。 *（例）「2^3」を計算し「8」を表示します□ <code> ラベル□□2□3□pow□作る。 </code> * abs: 絶対値。 *（例）「｜-3|」を計算し「3」を表示します□ <code> ラベル□□abs□-3□□作る。 </code> * 乱数, random : 正の整数を与えると、実行するたびに値が異なる1□nの整数を返します。 *（例）1から10までの整数をランダムに表示します□ <code> ラベル□□random□10□□作る。 </code> 0か負の数を与えた場合には、実行するたびに値が異なる0〜1の実数を返します。 *（例）0から1までの実数をランダムに表示します□ <code> ラベル□□random□0□□作る。 </code> * 乱数初期化 : 0以外の整数を与えると、それ以降に生成される乱数が毎回同じ順序で生成されるようになります。0を与えると、そのときどきのランダムな順序に戻ります。 *（例）整数「5」に対応した乱数系列を表示します□ <code> 乱数初期化（5）。 ラベル□□random□10□□作る。 </code> * コード文字: 指定された文字コード□UTF-16□の文字を返します。 *（例）文字列□“A”□を表示します□ <code> ラベル□□0x41□コード文字）作る。 </code> * □例）文字列**

「“あ”」を表示します <code> ラベル 0x3042 コード文字）作る。 </code> * 以下はJava版のみ * 進数: n進数に変換します nは2〜16の整数です。 *（例）「10」の2進表現である「1010」を表示します <code> ラベル！（10！2 進数）作る。 </code> * 大きい整数にする: 多倍長整数オブジェクトを作ります。10桁以上の値を生成する場合は、数値でなく文字列から生成してください。 *（例）値が2の多倍長整数オブジェクトを作り、「2^70」を計算します <code> x 2 大きい整数にする。 ラベル x 70 pow 作る。 </code> ## 文字列 * 文字の並びを扱うオブジェクトです。 * 数値を表す文字列（例：“123.45”）は数値として扱うことができます。 * 任意の文字は数値の「コード文字」命令を使って文字コードから文字を生成できるほか、文字列を囲む「“」などの引用符を表す文字の定数が用意されています。 * ”: dq, ダブルクオート, ダブルクォーテーション * “: ldq, 左ダブルクオート, 左ダブルクォーテーション * ”: rdq, 右ダブルクオート, 右ダブルクォーテーション * : ldb, 左二重かぎ括弧 * : rdb, 右二重かぎ括弧 *「含む？」「分割」「置き換える」「全部置き換える」では、正規表現で文字列のパターンを指定できます。使用できる主な表現を示します。[1] * . : 任意の1文字。 * ? : 直前の文字の0回か1回の繰り返し。 * * : 直前の文字の0回以上の繰り返し。 * + : 直前の文字の1回以上の繰り返し。 * \^ : 先頭。 * $: 末尾。 * | : 左右のパタンのいずれか OR * ( ) : 範囲指定。 * [] : 列挙された文字のいずれか。「-」は文字範囲。先頭の「^」は続く文字を含まないという指定。 * == , != , ≠, > , >= , ≧, < , ⇐, ≦ : 比較演算。両辺が数値または数値に変換できる文字列の場合は、数値として比較されます。それ以外は文字列として比較されます。 *（例） “b” > “a” を計算し「[true] を表示します <code> ラベル “b” > “a” 作る。 </code> * + : 文字列を連結します。 *（例）2つの文字列「“私は”」と「“かめたです”」を連結し「“私はかめたです”」を表示します <code> ラベル！（“私は” + “かめたです”）作る。 </code> * 連結: 文字列を連結します。複数の文字列を連結できます。 *（例）3つの文字列「“私は”」と「“かめた”」と「“です”」を連結し「“私はかめたです”」を表示します <code> ラベル！（“私は”！“かめた” “です” 連結）作る。 </code> * 長さ？ : 文字数を返します。 *（例）文字列「“はろー”」の長さを計算し「3」を表示します <code> ラベル！（“はろー”！長さ？）作る。 </code> * 何文字目？ : 文字列が何文字目に含まれるかを調べます。含まれない場合は0を返します。 *（例）文字列「“かめた”」が「“私はかめたです”」の何文字目に含まれているかを調べ「3」を表示します <code> ラベル！（“私はかめたです”！“かめた”何文字目？）作る。 </code> * 含む？ : 文字列を含むかを判定します。真偽値が返ります。文字列には正規表現を使えます。 *（例）フィールドに文字列を入力して を押すと、文字列に「山」という文字が含まれる場合はフィールドに「はい」が表示されます <code> f=フィールド！作る。 f 動作 s| s “山” 含む？」！なら f “はい” 書く」実行 </code> * 置き換える : 文字列の一部を置き換えた文字列を返します。元の文字列は変更されません。置き換えは1回だけ行われます。置き換える文字列の指定には正規表現を使えます。 *（例）「はい」を「いいえ」に置き換えて表示します。「はい、はい」が「いいえ、はい」と表示されます <code> s=“はい、はい”。 ラベル s “はい” “いいえ” 置き換える）作る。 </code> * 全部置き換える : 文字列の一部を置き換えた文字列を返します。元の文字列は変更されません。置き換えは複数回行われます。置き換える文字列の指定には正規表現を使えます。 *（例）「はい」を「いいえ」に置き換えて表示します。「はい、はい」が「いいえ、いいえ」と表示されます <code> s=“はい、はい”。 ラベル s “はい” “いいえ” 全部置き換える）作る。 </code> * 以下はJava版のみ * 部分: 文字列を切り出します m n 部分」でm文字目からn文字を取り出します。 *（例）文字列「“私はかめたです”」の3文字目から5文字を切り出し「“かめたです”」を表示します <code> ラベル！（“私はかめたです”！3 5 部分）作る。 </code> * 分割: 区切り文字列を指定すると、分割した文字列が入った配列を返します。区切り文字列には正規表現を使えます。 *（例）文字列 “I/am/kameta” を区切り文字「“/“」で分割し、 ”I” “am” “kameta” を要素とする配列「結果」を作り表示します <code> 結果 “I/am/kameta” “/“ 分割。 ラベル！（結果）作る。 </code> * 文字コード: 文字列の先頭文字の文字コード UTF-16 を返します。 *（例）文字列「”あ”」の文字コードを16進数で表示します。「3042」が表示されます <code> ラベル！（“あ”！文字コード 16 進数）作る。 </code> * 実行: 文字列をドリトルのプログラムとみなして実行します。 *（例）文字列「“かめた＝タートル！作る 100歩 歩く。”」をプログラムとして実行します <code> “かめた＝タートル！作る 100歩 歩く。”！ 実行 </code> * 大きい整数にする: 多倍長整数オブジェクトを作ります。10桁以上の値を生成する場合は、数値でなく文字列から生成してください。 *

（例）値が2の多倍長整数オブジェクトを作り、「2^70」を計算します <code> x “2”大きい整数にする。 ラベル x 70 pow 作る。 </code> ## 真偽値 * あらかじめ「真 はい 偽 いいえ）」という2個のオブジェクトが用意されています。 * 論理積 AND）や論理和 OR）を求めるときは、それぞれ「ぜんぶ どれか」オブジェクトに、真偽値をパラーメータとして「本当」を送ります。パラメータがブロックの場合には、その値が必要になるまでブロックは実行されません[2] * 論理否定 NOT）は、真偽値に「反対」を送ります。 * 本当: 「ぜんぶ」「どれか」と組み合わせて、論理積 AND と論理和 OR を求める。パラメータに複数の真偽値を指定できます。「ぜんぶ」の場合はパラメータのすべてが真のときに真を返します。 * （例）変数 x y は両方とも真（はい）なので、「 全部本当 」」が表示されます <code> x=はい y=はい。 「ぜんぶ x y 本当」！なら「ラベル！ 全部本当 作る」実行 </code> どれか」の場合はパラメータのすべてが偽のときに偽を返します。 * （例）変数 x y のどちらかは真（はい）なので、「 どれか本当 」」が表示されます <code> x=はい y=いいえ。 「どれか x y 本当」！なら「ラベル！ どれか本当 作る」実行 </code> * 反対: 真偽値と反対の値を返します。真偽値が「はい」なら「いいえ」が返り、真偽値が「いいえ」なら「はい」が返ります。 * （例） x の反対は真（はい）なので、「 いいえ 」」を表示します <code> x=いいえ x 反対」！なら「ラベル！ いいえ 作る」実行 </code> ## ブロック * 内部にプログラムコードを持つオブジェクトです。 * メソッド定義に用いられるほか、タイマーや繰り返しなどで利用します。 * 先頭の「 … 」でパラメータを受け取れます。「;」から後はローカル変数です。 * 実行された場合は、最後に実行された値が返ります。 * 繰り返す : ブロックの中をn回繰り返して実行します。 * （例）ブロック「出力！ こんにちは 書く」を3回繰り返して実行し、「こんにちは」が3回表示されます <code> 出力＝リスト！作る。 「出力！ こんにちは 書く」！3回 繰り返す。 </code> 何回目の実行かはパラメータとして渡されます。 * （例）実行回数をパラメータ n で受け取り、実行するたびに表示します <code> 出力＝リスト！作る。 n| 出力 n 書く」！5回 繰り返す。 </code> * なら, そうでなければ: 条件判断を行います。条件が成り立つときは「なら」の後のブロックを、成り立たないときは「そうでなければ」の後のブロックを実行します。「そうでなければ」は省略可能です。 * （例）乱数の値が5より大きい場合に「大吉」を表示します <code> 乱数（10）＞5」！なら「ラベル！ 大吉 作る」実行 </code> * 例）乱数の値が5より大きい場合に「大吉」を、そうでない場合は「小吉」を表示します <code> 乱数（10）＞5」！なら「ラベル！ 大吉 作る」そうでなければ「ラベル！ 小吉 作る」実行 </code> * の間 : 条件が成り立つ間、後のブロックを繰り返し実行します。 * （例）変数 x が10以下の間、 s=s+x x=x+1 を繰り返し実行します <code> x=1 s=0 x ⇐ 10 の間「s=s+x x=x+1 実行。 ラベル s 作る。 </code> * 実行 : ブロックに入っているプログラムを実行します。 * （例）ブロック「ラベル！ こんにちは 作る」を実行します <code> ラベル！ こんにちは 作る」！実行 </code> ## タイマー * 一定時間ごとに、与えられたブロックを繰り返して実行します。 * 標準では、0.1秒間隔で100回繰り返します（約10秒間です）。間隔は「間隔」で変更できます。 * 回数を「回数」で指定した場合には、指定された回数を実行すると終了します。実行のパラメータで回数を指定することもできます。 * 回数の代りに時間を指定した場合は、指定された時間で実行を終了します。 * 間隔の最小時間は1ミリ秒（0.001秒）です。 * 実行される間隔は、あまり正確ではありません。たとえば0.1秒間隔で10回繰り返して実行した場合、正確に1秒間にはなりません。大まかな目安として使ってください。また、指定した間隔より長い時間がかかる命令を実行した場合には、「回数」を指定した実行には時間がかかってもその回数を実行し、「時間」を指定した場合にはその時間が経過した時点で繰り返しを終了します。 * タイマーはプログラムの流れと並行して（スレッドとして非同期に）実行され、プログラムはタイマーの終了を待たずに先に進みます。 * タイマーの終了を待ってから次にタイマーを実行するには「次に実行」を、タイマーの終了を待ってから次に1回だけ命令を実行するには「最後に実行」を使います。 * 実行されるブロックには、何回目の実行かを表す数がパラメータとして渡されます。 * 実行は「中断」で止めることができます。タイマーは次の実行に移ります。 * 「停止」でそのタイマーの実行を完全に止めることができます。 * 「待つ」は廃止されました Bit Arrow版では使用できません * Java版（V3.0以降）では「待つ」はエラーにならずに動きますが、できるだけ「次に実行」と「最後に実行」を利用してください。 * 作る : 新しいタイマーを作ります。 * （例）タイマーを作り「時計」という名前にします <code> 時計＝タイマー！作る。 </code> * 間隔 : n秒間隔で動くようにします。 * （例）繰り返す間隔を「1秒」に設定します <code> 時計＝タイマー！作る。 時計！1秒 間隔 </code> * 回数 : n回動くようにします。 * （例）繰り返す回数を「10回」に設定します

<code> 時計＝タイマー！作る。 時計！10回　回数□ </code> * 時間：n秒間だけ動くようにします。 *（例）繰り返す時間を「5秒」に設定します□ <code> 時計＝タイマー！作る。 時計！5秒　時間□ </code> * 実行：ブロックを実行します。 *（例）タイマーを作り、「かめた！3歩　歩く」を繰り返し実行します□ <code> かめた＝タートル！作る。 時計＝タイマー！作る。 時計！「かめた！3歩　歩く」実行□ </code> 回数を指定すると、その回数だけ実行します。 *（例）5回繰り返して実行します□ <code> かめた＝タートル！作る。 時計＝タイマー！作る。 時計！「かめた！3歩　歩く」5回　実行□ </code> 何回目の実行かはパラメータとして渡されます。 *（例）何回目の繰り返しかを表示しながら実行します□ <code> かめた＝タートル！作る。 カウント＝ラベル！作る。 時計＝タイマー！作る。 時計□□□n|カウント□□n□書く。かめた！3歩　歩く」実行□ </code> * 次に実行：タイマーが終るのを待って次のタイマーを実行します。 *（例）タートルが前進するタイマーの実行が終るのを待ってから、タートルが回転するタイマーを実行します□ <code> かめた＝タートル！作る。 出力＝ラベル！作る。 時計＝タイマー！作る。 時計□□□n|出力□□n□書く。かめた！3歩　歩く」実行。 時計□□□n|出力□□n□書く。かめた！3度　左回り」次に実行□ </code> * 最後に実行：タイマーが終るのを待って命令を1回だけ実行します。 *（例）タイマーの実行が終るのを待ってから「終了！」を表示します□ <code> かめた＝タートル！作る。 出力＝ラベル！作る。 時計＝タイマー！作る。 時計□□□n|出力□□n□書く。かめた！3歩　歩く」実行。 時計！「出力！“終了！”　書く」最後に実行□ </code> * 中断：実行中のタイマーの実行を中断します。 タイマーに待ち行列がある場合は、次の実行に進みます。 *（例）中断ボタンを押すとタイマーの実行を中断します□ <code> かめた＝タートル！作る。 中断ボタン＝ボタン！“中断”　作る。 中断ボタン：動作＝「時計！中断」。 出力＝ラベル！作る。 時計＝タイマー！作る。 時計□□□n|出力□□n□書く。かめた！3歩　歩く」実行。 時計□□□n|出力□□n□書く。かめた！-3歩　歩く」実行□ </code> * 停止：実行中のタイマーの実行を停止します。 タイマーに待ち行列がある場合は、すべての実行を停止します。 *（例）停止ボタンを押すとタイマーの実行を停止します□ <code> かめた＝タートル！作る。 停止ボタン＝ボタン！“停止”　作る。 停止ボタン：動作＝「時計！停止」。 出力＝ラベル！作る。 時計＝タイマー！作る。 時計□□□n|出力□□n□書く。かめた！3歩　歩く」実行。 時計□□□n|出力□□n□書く。かめた！-3歩　歩く」実行□ </code> ## 配列 * 中に複数のデータを入れられるオブジェクトです。 * ひとつの配列の中に異なる種類のオブジェクトを入れることができます。 * 長さをあらかじめ決める必要はありません。 * 要素の番号は1から始まります。最初の要素は1番目です。 * 作る：新しい配列を作ります。パラメータで初期値を指定することもできます。 *（例）配列を作ります。要素はありません□ <code> 配列1＝配列！作る。 ラベル！（配列1）作る。 </code> * □例）文字列□“a”□□□“b”□が要素の配列を作ります□ <code> 配列1＝配列□“a”□“b”□作る。 ラベル！（配列1）作る。 </code> * 書く：配列にオブジェクトを追加します。配列の最後に追加されます。 *（例）文字列□“a”□□□“b”□が要素の配列を作り、文字列□“c”□を追加します□ <code> 配列1＝配列□“a”□“b”□作る。 配列1□“c”□書く。 ラベル！（配列1）作る。 </code> * 挿入：配列にオブジェクトを入れます□ □n obj 挿入」で、n番目の位置にobjが挿入されます。元のn番目以降の要素は後ろにずれます。 *（例）文字列□“a”□□□“b”□が要素の配列を作り、2番目の位置に文字列□“c”□を追加します□□[a c b]□が表示されます□ <code> 配列1＝配列□“a”□“b”□作る。 配列1□2□“c”□挿入。 ラベル！（配列1）作る。 </code> * 上書き：配列のオブジェクトを上書きします□□n obj 上書き」で□n番目の要素がobjで上書きされます□nが配列の要素数より大きいときは、配列の大きさが拡張されて値が書かれます。 *（例）文字列□“a”□□□“b”□□□“c”□が要素の配列を作り、2番目の要素を文字列□“d”□で上書きします□□[a d c]□が表示されます□ <code> 配列1＝配列□“a”□“b”□“c”□作る。 配列1□2□“d”□上書き。 ラベル！（配列1）作る。 </code> * 読む：配列の要素を返します。要素を1からはじまる整数で指定します。 *（例）文字列□“a”□□□“b”□□□“c”□が要素の配列を作り、2番目の要素を表示します□□b□が表示されます□ <code> 配列1＝配列□“a”□“b”□“c”□作る。 ラベル！（配列1！2　読む）作る。 </code> * ランダムに選ぶ：配列の要素をランダムに返します。 *（例）文字列□“a”□□□“b”□□□“c”□が要素の配列を作り、要素をランダムに表示します□ <code> 配列1＝配列□“a”□“b”□“c”□作る。 ラベル！（配列1！ランダムに選ぶ）作る。 </code> * 要素数？：配列の要素数を返します。 *（例）文字列□“a”□□□“b”□□□“c”□が要素の配列を作り、要素数を表示します。「3」が表示されます□ <code> 配列1＝配列□“a”□“b”□“c”□作る。 ラベル！（配列1！要素数？）作る。 </code> * 消す：配列の要素を消します。指定されたオブジェクトを配列からすべて削除します。 *（例）文字列□“a”□□

□“b”□□□“c”□が要素の配列を作り、値が「“b”□の要素を削除します□□[a c]□が表示されます□ <code> 配列1＝配列□“a”□“b”□“c”□作る。 配列1□“b”□消す。 ラベル！（配列1）作る。 </code> * 位置で消す : 配列の要素を消します。要素の位置を指定して削除します。 *（例）文字列□“a”□□□“b”□□□“c”□が要素の配列を作り、1番目の要素を削除します□□[b c]□が表示されます□ <code> 配列1＝配列□“a”□“b”□“c”□作る。 配列1！1 位置で消す。 ラベル！（配列1）作る。 </code> * クリア : 配列の要素をすべて消します。 *（例）文字列□“a”□□□“b”□□□“c”□が要素の配列を作り、すべての要素を削除します。「[ ]」が表示されます□ <code> 配列1＝配列□“a”□“b”□“c”□作る。 配列1！クリア。 ラベル！（配列1）作る。 </code> * それぞれ実行 : 配列の要素の数だけブロックを繰り返して実行します。パラメータには配列の要素が1個ずつ渡されます。 *（例）文字列□“abc”□□□“d”□□□“wxyz”□が要素の配列を作り、それぞれの要素をパラメータとしてブロック□□x| 出力□□x□長さ？）書く」を3回繰り返し実行します。「3」、「1」、「4」が表示されます□ <code> 出力＝リスト！作る。 配列1＝配列□“abc”□“d”□“wxyz”□作る。 配列1□□□x| 出力□□x□長さ？）書く」それぞれ実行□ </code> * 連結: パラメータで指定された要素を追加した配列を返します。パラメータに配列を指定した場合はその要素が追加されます。 *（例）文字列「“ 大阪 ”」が要素の配列1と「“ 東京 ”」、「“ 北海道 ”」が要素の配列2を連結します。「[大阪 東京 北海道]」が表示されます□ <code> 配列1＝配列！“ 大阪 ” 作る。 配列2＝配列！“ 東京 ” “ 北海道 ” 作る。 配列3＝配列！（配列1）（配列2） 連結。 ラベル！（配列3）作る。 </code> * 選ぶ: 配列の各要素に対してブロックを実行し、結果が「真（はい）」の要素からなる配列を返します。 *（例）文字列「“ 東京 ”」、「“ 北海道 ”」、「“ 三重 ”」、「“ 鹿児島 ”」が要素の配列を作り、それぞれの要素をパラメータとしてブロック□□x□□x□長さ？）== 2」の値が真になる要素の配列を表示します。「[東京 三重]」が表示されます□ <code> 配列1＝配列！“ 東京 ” “ 北海道 ” “ 三重 ” “ 鹿児島 ” 作る。 ラベル！（配列1□□□x□□x□長さ？）== 2」選ぶ）作る。 </code> * 加工: 配列の要素に対してブロックを実行し、それらの結果を要素とする配列を返します。 *（例）1, 3, 5が要素の配列1を作り、それぞれの要素の値を2倍するプログラムが定義されたブロック□□n□n * 2□を実行します。「[2 6 10]」が表示されます□ <code> 配列１＝配列！1 3 5 作る。 配列２＝配列□□□□n|□n□*□2□加工。 ラベル！（配列２）作る。 </code> * 最大: 最大の値を持つ要素を返します。値は数値以外の場合は文字列として比較します。 *（例）1, 5, 3が要素の配列1を作り、最大の要素を表示します。「5」が表示されます□ <code> 配列１＝配列！1 5 3 作る。 ラベル！（配列１！最大）作る。 </code> * 最小: 最小の値を持つ要素を返します。値は数値以外の場合は文字列として比較します。 *（例）1, 5, 3が要素の配列1を作り、最小の要素を表示します。「1」が表示されます□ <code> 配列１＝配列！1 5 3 作る。 ラベル！（配列１！最小）作る。 </code> * 結合: 配列の各要素を結合してひとつの文字列を返します。パラメータで要素の区切り文字を指定できます。 *（例）文字列「“ こんにちは、 ”」、「“ かめた ”」、「“ です！ ”」が要素の配列を作り、それらを結合した文字列を返します。「“ こんにちは、かめたです！ ”」が表示されます□ <code> 配列1＝配列！“ こんにちは、 ” “ かめた ” “ です！ ” 作る。 ラベル！（配列1！結合）作る。 </code>

1)
正規表現の詳細は、市販の書籍やWebサイトの解説などを参照してください。マッチした文字列の参照はサポートしていません。
2)
「ぜんぶ」「どれか」の実体は、それぞれ「はい」「いいえ」です。ブロックのパラメータが実行されるタイミングは**sec□common□object□boolean** を参照してください。

From:
https://dolittle.eplang.jp/ - **プログラミング言語「ドリトル」**

Permanent link:
https://dolittle.eplang.jp/ref_basic?rev=1518142106

Last update: **2018/02/09 11:08**