数値

小数点付きの数値が使えます。 先頭に-符号をつけることができます。 全角半角は問いません。

```
123
123.456
-123
```

四則演算

+,-,×,÷,%が使用できます。 全角半角は問いません。

```
1+2
1-val2*50%arr[1]+"個"
```

比較演算

＞,＜,≧,≦,＝,≠が使用できます。 全角半角は問いません。

```
1>2
2≠1
```

論理演算

「かつ」,「または」,「でない」が使用できます。

```
1>2 または 2>1
1>0 かつ 5>3 でない
```

文字列

“ “ または「」で囲って文字列を記述することができます。

```
"こんにちは"
「こんばんは」
```

配列

{}で囲い、その中で「,」区切りで値を列挙します。

```
{1,2,3}
{1,2,{3,4,5},6}
```

変数名

ローマ字（a-z,A-Z□で始まり、その後にローマ字、数字、_が続く名前が使用できます。

```
num
num□num
num123
```

配列参照

配列の要素には、配列名の後に[]で囲い、その中に添え字を書きます。 要素は1から始まります。 多次元配列の場合は、「,」で区切って要素を指定することができます。 未定義の変数や配列要素に対して配列参照した場合は、その変数に空の配列を代入してから参照します。

```
arr[1]
arr[1,2]
```

関数呼び出し

関数名の後に（）をで囲い、その中に引数を「,」区切りで列挙します。

```
かんすう(100)
かんすう(100,200)
```

代入

←の左辺に代入したい変数、右辺に代入したい値や式などを記述します。

```
num←123
arr←{1,2,3}
arr[1]←5
```

増やす/減らす

値を任意の数だけ増やすことができます。 未定義の変数を対象に実行した場合には、0が代入されてから実行されます。

```
val1を1増やす
val2を50減らす
```

表示文

式や変数などを表示できます。「と」で区切って複数が渡せます。「改行」を渡すことで任意の位置で改行することができます。「改行なしで」をつけることで、行の最後で改行しないようにすることができます。

```
var1を表示する
123+456と"aiueo"と改行を表示する
"こんにちは"と改行なしで表示する
```

配列の初期値設定

配列の要素の値を設定します。 まだ値が定義されていない要素を参照した際に、この値が参照されるようになります。

```
    arr[1] // -> 未定義
    arrのすべての値を0にする
    arr[1] // -> 0
```

繰り返し

for文は次のように記述します。

```
    iを0から10まで1ずつ増やしながら、
        iを表示する
  を繰り返す
    iを10から0まで1ずつ減らしながら、
        iを表示する
  を繰り返す
```

また、while文は次のように記述します。

```
    i←0
    i<10の間、
        iを表示する
        i←i+1
  を繰り返す
```

条件分岐

if文は次のように記述します。

```
  もし1≠1ならば
    1を表示する
  を実行し、そうでなくもし2≠2ならば
    2を表示する
  を実行し、そうでなければ
    3を表示する
  を実行する
```

実行したい文が1文の場合に限り、次のように書くこともできます。

```
もし1=1ならば、"Hello"を表示する
```

配列の要素を入れ替える

「入れ替える」関数により、配列の要素を入れ替えることができます。

```
   arr←{"a","b","c"}
入れ替える(arr,1,3) // <- {"c","b","a"}
```

配列の要素を削除する

「削除」関数により、番号を指定して配列の要素を削除することができます。 削除された段階で、それ以降の要素は切り詰められます。

```
   arr←{"a","b","c"}
削除(arr,2) // <- {"a","c"}
```

配列に要素を挿入する

「挿入」関数により、番号を指定して要素を挿入することができます。

```
   arr←{"a","b","c"}
挿入(arr,"d",2) // {"a","d","b","c"}
```

配列の要素数を取得する

「要素数」関数により、配列の要素数を取得することができます。

```
   arr←{"a","b","c"}
要素数(arr) // <- 3
```

変数の確認

現在定義されている変数の中身を確認したい際には、「確認()」関数を使用できます。

```
   arr ←{1,2,3,4,5}
   var1 ← あいうえお」
確認()
(出力例)
確認------------------
   arr => { 1, 2, 3, 4, 5 }
   var1 => あいうえお
   ------------------------
```

関数の定義

関数は次のように定義することができます。 関数名には全角文字を使用することができます。 （）の中に引数を記述することも可能です。

```
かんすう（）は
  「こんにちは」を表示する
を実行する
かんすう□□str□は
      strを表示する
を実行する
```

また、次のように定義することで、関数に戻り値を設定できます。

```
かんすう□□num□は
      num×□を返す
を実行する
かんすう3（5）を表示する // -> 10
```

性能の確認

指定した関数の性能を測定することができます。 以下の内容を測定しています。

- 実行時間
- 各for文のループ回数
- 各if文の「条件判定を行った回数」「真が評価された回数」「偽が評価された回数」

```
かんすう（）は
      var1を1から10まで1ずつ増やしながら、
          var1を改行なしで表示する
    もしvar1%3=0ならば
      「<-3の倍数！」を表示する
    を実行し、そうでなければ
    改行を表示する
    を実行する
  を繰り返す
を実行する
かんすう（）の性能を確認する
（出力例）
  1
  2
3<-3の倍数！
  4
  5
6<-3の倍数！
  7
  8
9<-3の倍数！
  10
性能テスト----------
実行時間 => 0.003秒
  for1 ： 実行回数 => 10
  if1 ： 比較回数 => 10, 真の回数 => 3, 偽の回数 => 7
```

```
    --------------------------
```

また、次のように記述することで、プログラム全体の性能を確認することもできます。

```
 かんすう（）は
        var1を1から10まで1ずつ増やしながら、
            var1を改行なしで表示する
      もしvar1%3=0ならば
         「<-3の倍数！」を表示する
      を実行し、そうでなければ
      改行を表示する
      を実行する
    を繰り返す
 を実行する
 かんすう（）
 かんすう（）
 性能を確認する
（出力例）
    1
    2
 3<-3の倍数！
    4
    5
 6<-3の倍数！
    7
    8
 9<-3の倍数！
    10
    1
    2
 3<-3の倍数！
    4
    5
 6<-3の倍数！
    7
    8
 9<-3の倍数！
    10
 性能テスト----------
 実行時間 => 0.011秒
    for1 ： 実行回数 => 20
    if1 ： 比較回数 => 20, 真の回数 => 6, 偽の回数 => 14
    --------------------------
```