DNCL学習環境「どんくり」

- どんくりは大学入試センター試験のプログラミングの出題に使われているDNCLの学習環境です。
- アルゴリズムの記述に適しています。
- オンラインで利用できる他、ダウンロードしてローカルで利用できます。

使い方

- オンライン版はインストールなしでブラウザで動作します。
  - サンプルプログラムが用意されています。
  - Google Chromeで動作を確認しています。
- インストール版は、ダウンロードしてファイルを展開してください。管理者権限は不要です。
  - Windows用（約100MB）
  - Mac用（約100MB）

履歴

- 2018/10/10 開発版V0.2を公開しました。
- 2018/8/11 開発版V0.1を公開しました。（オンライン版、Windows用（約100MB）、Mac用（約100MB））

言語と命令の説明

- 以下は言語の説明と使用できる命令です。公式の仕様書に準拠しているほか、独自の「拡張機能」を用意しています。
- プログラムの中で、「数字、英字、記号、空白」は全角半角（日本語文字と英語文字）を区別せずに使えます。
  - 日本語の長音「ー」とマイナス「－」は区別されます。
  - 掛け算は「*」「＊」「×」を使います。
  - 割り算は「/」「／」を使います。（整数の割り算は「÷」、整数の余りは「%」「％」を使います）
  - 大小の比較は「>」「＞」、「>=」「≧」、「<」「＜」、「<=」「≦」、「=」「＝」、「!=」「≠」を使います。
  - 論理演算は「かつ」「または」「でない」を使います。
  - 代入は「←」を使います。
- プログラムの英語表示と編集が可能です。
  - 「DNCL」と「英語表示」を切り替えることで、日本語のDNCL表記とC言語風の表記が変換されます。
    - ※ インデントが自動調整されます。改行位置が変わったり、空白行が無くなったりする可能性があります。
  - 掛け算は「*」を使います。
  - 割り算は「/」を使います。（整数の割り算は「÷」、整数の余りは「%」を使います）
  - 大小の比較は「>」、「>=」、「<」、「<=」、「==」、「!=」を使います。
  - 論理演算は「&&」「||」「!」を使います。

- 代入は「=」を使います。

変数

変数名は英字[a-z,A-Z]で始まり、その後に英字、数字、「_」が続きます。 変数には「←」（英語表示では「=」）で初期値を代入してから使います。 配列の場合は後述の「～のすべての値を～にする(allinit)」で初期値を設定してから使うこともできます。複数の代入文を「,」で区切って並べられます。

```
x ← 3
Arr ← {1, 2, 3}
moji_moji ← "文字"
x ← 3, y ← 4
```

```
// 英語表示
x = 3
Arr = {1, 2, 3}
moji_moji = "文字"
x = 3, y = 4
```

増やす/減らす

変数の値を指定した数だけ増減します。 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。

```
xを1増やす
yを50減らす
```

```
// 英語表示
x += 1;
y -= 50;
```

表示文

式や変数の値を表示します。 複数の値を「と」で区切って指定できます。 改行の有無を指定できます。

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

```
// 英語表示
println(x);
println(123+456,"aiueo");
print("こんにちは");
```

数値

小数点付きの数値を使えます。 先頭にマイナス（-）符号を付けられます。

```
123
123.456
-123
```

四則演算

+, -, ×, - は小数点を考慮した計算を行います。÷, % は整数の計算を行います（商と余り）。英語表示の乗算は*を使います。

```
1+2
1-x*50%4
```

比較演算

＞, ＜, ≧, ≦, ＝, ≠ を使えます。英語表示では >, <, >=, ⇐, ==, != を使います。

```
1>2
2≠1
```

論理演算

「かつ」「または」「でない」を使えます。英語表示では &&, ¦¦, ! を使います。

```
1>2 または 2>1
1>0 かつ 5>3 でない
```

```
// 英語表示
1>2 || 2>1
!(1>0 && 5>3)
```

文字列

“　”または「 」で囲って文字列を記述します。

```
"こんにちは"
「こんばんは」
```

両辺のどちらかが文字列の場合は、「＋」は「数の足し算」ではなく「文字の連結」の意味になります。

```
3+"こんにちは"  // 結果は"3こんにちは"
「こんばんは」+3 // 結果は"こんばんは3"
3+"4"      // 結果は"34"
```

配列参照

配列の要素は、配列名の後に[]で囲み添え字を書きます。要素は1から始まります。多次元配列は、「,」で区切って要素を指定します。英語表示では[ ]を並べて指定します。

```
x ← Arr[1]
y ← Arr[1,2]
```

```
// 英語表示
x = Arr[1];
y = Arr[1][2];
```

配列の初期値設定

配列の要素の初期値を設定します。

```
Arrのすべての値を0にする
```

```
// 英語表示
allinit(Arr, 0);
```

関数呼び出し

関数名の後に引数を（）で指定します。英語表示では組込関数は英語名になります。

```
書く()
二倍(100)
乗算(100,200)
追加する(Arr,15)
```

```
// 英語表示
書く();
二倍(100);
乗算(100,200);
add(Arr,15);
```

繰り返し

for文に相当する反復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
```

```
// 英語表示
for( i=0 ; i<=10 ; i+=1 ){
□□print(i);
}
for( i=10 ; i>=0 ; i-=1 ){
□□print(i);
```

```
}
```

while文に相当する反復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
```

```
// 英語表示
i=0;
while(i<10){
□□print(i);
□□i+=1;
}
```

回数を指定した繰り返し文は、次のように記述します。

```
i ← 0
ここから5回、
□□iを表示する
□□iを1増やす
を繰り返す
```

```
// 英語表示
i=0;
repeat(5){
□□print(i);
□□i+=1;
}
```

条件分岐

if文に相当する分岐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
```

```
// 英語表示
if(1!=1){
□□print(1);
}else if(2!=2){
□□print(2);
}else{
□□print(3);
```

```
}
```

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
```

```
// 英語表示
if(1==1) print("Hello");
```

以下は独自拡張の機能です。

配列の初期値

値は全体を { } で囲み、「,」で値を区切ります。

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

配列の要素を入れ替える

「入れ替える(swap)」は、配列の要素を入れ替えます。

```
Arr ← {"a","b","c"}
入れ替える(Arr,1,3) // 配列の値は{"c","b","a"}になる
```

配列の要素を削除する

「削除(remove)」関数は、番号を指定して配列の要素を削除します。

```
Arr ← {"a","b","c"}
削除(Arr,2) // 配列の値は{"a","c"}になる
```

配列に要素を挿入する

「挿入(insert)」関数は、番号を指定して要素を挿入します。

```
Arr←{"a","b","c"}
挿入(Arr,"d",2) // 配列の値は{"a","d","b","c"}になる
```

配列の要素数を取得する

「要素数(length)」関数は、配列の要素数を取得します。

```
Arr←{"a","b","c"}
要素数(Arr) // 結果は3が返される
```

変数の確認

「確認(dump)」関数は、プログラムの中で使われている変数の値を確認します。

```
Arr ←{1,2,3,4,5}
x ← 「あいうえお」
確認()

（出力例）
確認------------------
Arr => { 1, 2, 3, 4, 5 }
x => あいうえお
-----------------------
```

関数の定義

関数は次のように定義します。 （）の中に引数を記述できます。

```
あいさつ（）は
  「こんにちは」を表示する
を実行する

書く（str）は
    strを表示する
を実行する
```

```
// 英語表示
function hello(){
  print("hello!");
}
function write(str){
  print(str);
}
```

次の例は、関数に戻り値を設定します。

```
二倍（num）は
    num×2を返す
を実行する

二倍（5）を表示する // 10が表示される
```

```
// 英語表示
function twice(num){
  return num*2;
}
print(twice(5));
```

性能の確認

関数またはプログラムの性能を測定します。 以下の内容を測定しています。

- 実行時間
- 各for文/while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」
- 各関数の呼出回数

次の例は、関数の性能を測定します。

```
倍数判定（）は
    xを1から10まで1ずつ増やしながら、
        xを改行なしで表示する
    もしx%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
倍数判定（）の性能を確認する
```

```
// 英語表示
function is_multiple(){
□□for( x=1 ; x<=10 ; x+=1 ){
□□□□print(x);
□□□□if(x%3==0){
□□□□□□println("<-□の倍数！");
□□□□}else{
□□□□□□println(\n);
□□□□}
□□}
}
profile(is_multiple());
```

```
（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
統計情報----------
(実行時間)
0.007秒
(実行回数)
for1 : 10
if1 : 比較10, 真3, 偽7
```

```
(呼び出し回数)
倍数判定:1
--------------------------
```

次の例は、プログラム全体の性能を確認します。

```
倍数判定（）は
    xを1から10まで1ずつ増やしながら、
        xを改行なしで表示する
    もしx%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
倍数判定（）
倍数判定（）
性能を確認する
```

```
// 英語表示
function is_multiple(){
□□for( x=1 ; x<=10 ; x+=1 ){
□□□□noNL_print(x);
□□□□if(x%3==0){
□□□□□□print("<-□の倍数！");
□□□□}else{
□□□□□□print(\n);
□□□□}
□□}
}
is_multiple();
is_multiple();
performance();
```

```
（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
統計情報----------
(実行時間)
0.018秒
(実行回数)
for1 : 20
if1 : 比較 20, 真 6, 偽 14
(呼び出し回数)
倍数判定 : 2
--------------------------
```

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

Permanent link:
https://dolittle.eplang.jp/dncl?rev=1539122878

Last update: **2018/10/10 07:07**