DNCL と Python の比較　　[早見表]

1 変数

DNCL

通常の変数例：kosu, kingaku_kei
　変数名は英字で始まる英数字と「_」の並び

配列変数の例：Tokuten[3], Data[2,4]
　配列名は先頭文字が大文字
　※特に説明がない場合，
　　配列の要素を指定する添字は
　　0から始まる

【Python】

変数名：kosu, Kingaku_kei, _ichikawa3
　変数名は，・英数字と「_」の並び
　　・１文字目は数字以外
　　・大文字と小文字を区別
　　・予約語でないこと
　　・文字数の制限はない
リスト：Tokuten[3], data[2][4]
　リスト(配列)名の規則は変数名と同じ
　※添字(インデックス)は0から始まる
　例；data = [[1,2,3,4,5],[5,6,7,8,9],[10,11,12,13,14]]
　　のとき，data[2][4]=14 である

2 文字列

DNCL

文字列は
ダブルクォーテーション（"）で囲む
　moji = "I'll be back."
　message = "祇園精舎の" + "鐘の声"
　　※＋で連結できる

【Python】

文字列：シングルクォーテーション(')
　　またはダブルクォーテーション(")
　moji2024 = "I'll be back."
　moji2025 = 'I cried "Help me!"'
　message01 = "諸行無常の"
　message02 = '響きあり'
　message = message01 + message02
　　※変数でも＋で文字列の連結ができる

3 代入文

DNCL

kosu = 3 , kingaku = 300
　※複数文を１行で表記できる
kingaku_goukei = kingaku * kosu
namae = "Komaba"
Data = [10,20,30,40,50,60]
Tokuten のすべての値を0にする
nyuryoku =【外部からの入力】

【Python】

kosu = 3 ; kingaku = 300
　※複数文を「;」で区切り１行で表記できる
Tokuten = [0,0,0,0,0,0,0]
Tokuten = [0]*7
Tokuten = [0 for k in range(7)]　など
　※リストは使用前に作成(要素の個数や次数を指定)する必要がある
nyuryoku = input()　　※文字列型
nyuryoku = int(input())　※整数型
nyuryoku = float(input())※浮動小数点型

4 算術演算

DNCL

加減乗除の四則演算は，
『＋』，『－』，『＊』，『／』で表す
整数の除算では，
商(整数)を『÷』で，余りを『%』で表す
べき乗は『＊＊』で表す

【Python】

加減乗除の四則演算は，
『＋』，『－』，『＊』，『／』で表す
整数の除算では，
商(整数)を『//』で，余りを『%』で表す
べき乗は『＊＊』で表す

5 比較演算

DNCL

『==』(等しい)，『!=』(等しくない)，
『＞』，『＜』，『>=』，『<=』

【Python】

『==』(等しい)，『!=』(等しくない)，
『＞』，『＜』，『>=』，『<=』

6 論理演算

DNCL

『 and 』(論理積)，『 or 』(論理和)，
『 not 』(否定)

【Python】

『 and 』(論理積)，『 or 』(論理和)，
『 not 』(否定)
注）not［条件式］　例：not signal=="青"

7 関数

DNCL

```
値を返す関数例：
    kazu = 要素数(Data)
    saikoro = 整数(乱数( )*6)+1
値を返さない関数例：表示する(Data)
表示する(Ka[i],"の得点は",ten[i],"です")
  ※「表示する」関数はカンマ区切りで
    文字列や数値を連結できる
  ※「表示する」関数以外は基本的に
    問題中に説明あり
```

☞ '関数'本体の仕様は特に規定がない

[知識] 乱数( )：0以上1未満の小数の乱数(を発生)
∴ 整数(乱数( )*6)+1 ⇒ 整数乱数 1,2,3,4,5,6

【Python】

```
★ 組み込み/モジュール
  kazu = len(data)
  import random
  saikoro = int(random.random( )*6)+1
★ユーザ定義関数
  定義 ⇒ def 関数名(引数1,引数2,･･･):
              文･処理 ････          ※仮引数
              return 戻り値
  呼出 ⇒ 関数名(引数a,引数b,･･･)  ※実引数
```

[蛇足] 変数名は「1 変数」のような仕様であるが，
Pythonでは全角の日本語文字を用いても
特に問題が起きないようです

8 制御文(条件分岐)

DNCL

```
もし x＜3 ならば：
  ｜ x = x + 1
  └ y = y + 1
もし x＞=3 ならば：
  ｜ x = x - 1
そうでなくもし x＜0 ならば：
  ｜ x = x * 2
そうでなければ：
  └ y = y * 2
```

※｜と└で制御範囲を表し，└は制御文の終わりを示す

[参考] 旧DNCLでは，〈処理〉が１行しかない場合は
全体を１行で書くことが許されていた
例：もし x ＜ 3 ならば x = x + 1 を実行する

【Python】

```
if x ＜ 3:
    x = x + 1
    y = y + 1
if x＞=3:
    x = x - 1
elif x ＜ 0:
    x = x * 2
else:
    y = y * 2
```

```
if x == 3:
    x = x - 1
else:
    y = y * 2
```

9 制御文(繰返し)

DNCL

```
xを0から9まで1ずつ増やしながら繰り返す:
 └ goukei = goukei + Data[x]
                ※「減らしながら」もある
n＜10の間繰り返す：
 ｜ goukei = goukei + n
 └ n = n + 1
```

※｜と└で制御範囲を表し，└は制御文の終わりを示す

☞ ループを抜けた後の各変数の値
[上の手順] x⇒不明 (旧DNCLの仕様では10)
[下の手順] n⇒10

【Python】

```
data = [1,2,3,4,5,6,7,8,9,10]
goukei = 0
for x in range(10):
    goukei = goukei + data[x]
```

※ループを抜けた後：x⇒9 ,goukei⇒55

```
n = 1, goukei = 0
while n ＜ 10:
    goukei = goukei + n
    n = n + 1
```

※ループを抜けた後：n⇒10 ,goukei⇒45

10 コメント

DNCL

```
atai = 乱数( )    #０以上1未満の乱数
※１行内において
  # 以降の記述は処理の対象とならない
```

【Python】

```
atai = 乱数( )    #０以上1未満の乱数
※１行内において
  # 以降の記述は処理の対象とならない
```

[参考] いろいろな言語のコメントアウト
・Pyrhon Ruby (PHP) ⇒ #
・C C++ C# Java JavaScript PHP ⇒ // /* ･･･ */
・BASIC VBA ⇒ '
・HTML ⇒ <!-- ･･･ ---> CSS ⇒ /* ･･･ */