

Sage Quick Reference: Python Syntax

NUMATA, Yasuhide

Sage Version 4.8

<http://wiki.sagemath.org/quickref>

GNU Free Document License, extend for your own use.

変数への代入

変数 = 代入するもの

変数への代入は =

変数の名前に使える文字は a-z, A-Z, _, 0-9.

ただし, 数字で始まるもの文字列は NG.

システムで使われている文字列も NG.

代入が起こった時点で変数が生成されるので, 変数の宣言をする必要はない.

次のような複合演算子もある:

`a += b` \rightsquigarrow `a = a+b` とほぼ同じ

`a -= b` \rightsquigarrow `a = a-b` とほぼ同じ

`a *= b` \rightsquigarrow `a = a*b` とほぼ同じ

など.

関数の定義

`def` 関数名 (引数):

関数の定義

値を返すときには `return` を使う.

引数は何個でも OK. 無くてもよい.

関数の定義部分はインデントをする. 複数行でも OK.

次のように定義すると, `a_plus_b(2,3)` で 5 が返ってくる.

```
def a_plus_b(a,b):
```

```
    c=a+b
```

```
    return c
```

`if/elif/else`

`if` 条件 1:

条件 1 を満たす時にやること

`elif` 条件 2:

条件 1 を満たさず条件 2 を満たす時にやること

`elif` 条件 3:

条件 1,2 を満たさず条件 3 を満たす時にやること

`else`:

どの条件をも満たさなかった時にやること

`elif` はいくつでも OK. なくとも良い.

`else` は無くとも良い.

処理を書く部分はインデントをする. 複数行でも OK.

`while/else/break/continue`

`while` 条件:

条件を満たす間くり返しやること

`else`:

ループから抜けた時に実行すること

`else` は無くとも良い.

処理を書く部分はインデントをする. 複数行でも OK.

`for/else/break/continue`

`for elm in sequence`:

sequence の各要素 (変数 *elm* に代入される) に対しやること

`else`:

ループから抜けた時に実行すること

`else` は無くとも良い.

処理を書く部分はインデントをする. 複数行でも OK.

0 から 9 までの整数を出力するには

```
for i in range(10):
```

```
    print i
```

`break/continue`

`while` ループまたは `for` ループの中で, `break` を実行すると, 即座にループを抜ける. このとき `else` の部分は実行されない.

`while` ループまたは `for` ループの中で, `continue` を実行すると, 即座に次の繰り返しに移る.

比較演算子

< <= > >= <> != ==

ブール型

`True`, `False`

`not x`, `x and y`, `x or y`

List

`a=[40,31,22,13]`

`len(a)` \rightsquigarrow 4, `a[0]` \rightsquigarrow 40

`a[0]==a[-3]`, `a[1]==a[-2]`, `a[2]==a[-1]` \rightsquigarrow `True`

内包的記法:

[元 `for` くり返し `if` 条件] ('if 条件' は省略可)

[`2*i for i in range(9) if i % 3 == 0`] \rightsquigarrow [0,6,12]

非破壊的な操作 (各操作の前に `a=[0,1]`; `b=[2,3,4,5]`):

`c=a+b` \rightsquigarrow `c==[0,1,2,3,4,5]`

`c=a*2` \rightsquigarrow `c==[0,1,0,1]`

`c=b[:3]` \rightsquigarrow `c==[2,3,4]`

`c=b[1:]` \rightsquigarrow `c==[3,4,5]`

`c=b[1:3]` \rightsquigarrow `c==[3,4]`

`c=b[:]` \rightsquigarrow `c==a`

破壊的な操作 (各操作の前に `a=[0,3,1,2]`):

`a[0]=1` \rightsquigarrow `a==[1,3,1,2]`

`a.append(4)` \rightsquigarrow `a==[0,3,1,2,4]`

`a.insert(1,4)` \rightsquigarrow `a==[0,4,3,1,2]`

`a.extend([10,11])` \rightsquigarrow `a==[0,3,1,2,10,11]`

`b=a.pop()` \rightsquigarrow `a==[0,3,1]`; `b==2`

`b=a.pop(1)` \rightsquigarrow `a==[0,1,2]`; `b==3`

`a.reverse()` \rightsquigarrow `a==[2,1,3,0]`

`a.sort()` \rightsquigarrow `a==[0,1,2,3]`

検索 (各操作の前に `a=[0,2,3,1,2,2]`):

`2 in a` \rightsquigarrow `True` `5 not in a` \rightsquigarrow `True`

`a.index(2)` \rightsquigarrow 1 `a.index(5)` \rightsquigarrow エラー

`a.count(2)` \rightsquigarrow 3

Tuple

`a=(0,1,2,3,4)`

List とほぼ同じに扱える. ただし破壊的な操作はできない.

変数に一度に代入することもできる: `t=(0,1,2)`; `(a,b,c)=t`

List から tuple へ: `a=[30,21,12]`; `tuple(a)`

Tuple から list へ: `a=(30,21,12)`; `list(a)`