D++のはまり日誌

最近はRoguelike関係の話題ばっかです。
[NetHack] [EM] [FHS] [Spork] [UNH]
[Xang] [変愚] [Tiny]

antenna side: Dempa++
Twitter side: @dplusplus
<[Web][つれづれ]WIDEのIRCサービ...

2010-01-26

■[Python]Python基礎文法最速マスター ★★15★

↓に便乗してPython版も書いてみました。

- Perl基礎文法最速マスター - Perl入門～サンプルコードによるPerl入門～
- Ruby基礎文法最速マスター - Route 477
- PHP基礎文法最速マスター - Shin x blog

ほとんど上記の記事と同じような内容で書いたのでPython入門記事としては色々抜けていたりしますがご了承ください。

Pythonは現在3.x系がリリースされていますが本記事では基本的にPython2.6について書きます。

参考文献:

- 初めてのPython (asin:4873113938)
- Python Documentation Index http://www.python.org/doc/
- Python 和訳Document http://www.python.jp/doc/ (Python2.5)

0. 対話環境として使う

対話環境

pythonはそのまま実行すると対話環境として働きます。

```
% python
```

```
(中略)
Type "help", "copyright", "credits" or "license" for more information.
>>> print 'hello'
hello
>>> 1 + 2
3
```

help()に関数やその他のオブジェクトを渡すことでヘルプが(あれば)見られます。qをタイプするとヘルプを抜けます。

```
>>> help(1)
Help on int object:

class int(object)
 |  int(x[, base]) -> integer
(以下略)
```

dir()にオブジェクトを渡すことでオブジェクトの属性名一覧が返されます。

```
>>> dir(1)
['__abs__', '__add__', '__and__', '__class__', '__cmp__', '__coerce__', '__delattr__',
```

どんな関数があるかを調べるのに使ったりしますが、標準のままではだいぶ見辛いので外部モジュールのseeの導入をおすすめします。

- Coffee and Ashtray: pythonにsee()を入れる

type()でオブジェクトの型を調べます。

```
>>> type('hoge')
<type 'str'>
```

exit()を呼び出すと終了します。

1. 基礎

print文

print文です。自動的に末尾に改行が付加されます。

カンマで区切って複数いっぺんに出力もできます。この場合、間に半角空白が挿入されます。

```
print 'Hello world'
print 'hoge', 'moge', 'fuga'
```

入力

```
raw_input()
input()
```

raw_input()は入力された文字列をそのまま返します。

input()はeval(raw_input())と等価であり、入力文字列をPythonの式として解釈した結果を返すため、入力文字列が信頼できない状況では使ってはいけません。(参考→Wikipedia:ja:eval)

コメント

一行コメントです。#から行末までがコメントアウトされます。

```
# コメント
```

変数の宣言

Pythonには変数の宣言はありません。変数に代入する型も制限されません。

```
a = 1
a = 'hoge' # aを上書き
```

スクリプトの実行

```
% python hoge.py
```

2 数値

整数や小数、複素数が使えます。末尾にjを付与すると虚数を表します。

```
num = 1
num = 3.14
num = 1 + 1j
```

各種演算

```
num = 1 + 1     # 2
num = 1 - 1     # 0
num = 1 * 2     # 2
num = 5 / 2     # 2 (整数どうしの除算は整数に切り捨てられる)
num = 5.0 / 2   # 2.5 (どちらかが小数なら結果も小数)
num = 5 % 2     # 1 (剰余をとる)
num = 2 ** 3    # 8 (累乗)
```

インクリメントとデクリメント

Pythonには++演算子がないので、+=や-=を使います。

```
i += 1
i -= 1
```

論理演算

Pythonの論理演算は以下のとおりです。orやandはTrue/Falseを返すわけではないことに注意。

```
x or y  # x が偽なら y 、そうでなければ x
x and y # x が偽なら x 、そうでなければ y
not x   # x が偽なら True 、そうでなければ False
```

3. 文字列

文字列の表現

通常の文字列はシングルクォートかダブルクォートで囲みます。

また、'''～'''、"""～"""で複数行にわたるブロック文字列を構成できます。クォートに囲まれている範囲では改行がそのまま扱われます。

```
str1 = 'abc'
str2 = "def"
str3 = '''ghi
jkl
mno'''
```

C言語でいうところのsprintf的な操作は%演算子、あるいはformat()関数で行います。

```
str4 = '%s def' % str1              # 'abc def'
str5 = '%s %s' % (str1, str2)       # 'abc def'
str6 = '{1} {0}'.format(str1, str2) # 'def abc'
```

文字列操作

各種文字操作です。

```
# 結合
join1 = 'hoge' + 'moge' # 'hogemoge'
join2 = ','.join(['aaa', 'bbb', 'ccc']) # 'aaa,bbb,ccc'

# 分割
record1 = 'aaa bbb  ccc'.split() # ['aaa', 'bbb', 'ccc'] デフォルトでは空白で分割
record2 = 'aaa,bbb,ccc'.split(',') # ['aaa', 'bbb', 'ccc']

# 長さ(文字数)
length = len('Supercalifragilisticexpialidocious') # 34

# 切り出し
# [start:stop]の形で指定して切り出す
substr1 = '0123456789'[:3]  # '012'  0～stop-1を切り出す
substr2 = '0123456789'[3:6] # '345'  start～stop-1を切り出す
substr3 = '0123456789'[6:]  # '6789' start～末尾を切り出す

# 検索
result = 'abcd'.find('cd') # 見つかった場合はその位置、見つからなかった場合は-1が返る
```

4. リスト、タプル

リスト

他言語でいう配列のような型です。要素としてどんな型のものでも含むことができます。

タプル

「オブジェクトの組」という点ではリストに似ていますが、要素の追加や削除はできません。関数から複数の値を返したいとき等に多用されます。

```
# 空リストの作成
list1 = []
list2 = list()
# 複数要素からなるリストを作成
list3 = [0, 1, 2]
# 他のリストの(浅い)コピーを作成
list4 = list(list3)
# 連番のリストを作成
list5 = range(3) # [0, 1, 2] range(stop)は[0, 1, ..., stop-1]を返す
list6 = range(3, 6) # [3, 4, 5] range(start, stop)は[start, start+1, ..., stop-1]を返す
list7 = range(1, 8, 2) # [1, 3, 5, 7] 3番目の引数で値の増減量を指定できる

# 空タプルの作成
tuple1 = ()
tuple2 = tuple()
# 要素が1つのタプルの作成
tuple3 = (1,)
# 複数要素からなるタプルを作成
tuple4 = (0, '1', 2, [3], -4)
```

要素の参照と代入

リスト、タプルは0から始まる添字で要素を参照できます。また、タプルは要素を新たに代入できません。

```
# 要素の参照
list2[1] # 1
list2[-1] # 2 インデックスに負数を指定することで後ろから辿れる
tuple4[2] # 2

# 要素の代入
list2[0] = 100
```

タプル/リストの要素数

```
len(list3) # 5
len(tuple4) # 5
```

リストの操作

```
array = [1, 2, 3, 4]
```

```
# 任意の要素を取り出す
first = array.pop(0) # first == 1, array == [2, 3, 4]
# 任意の要素を追加する
array.insert(0, 5) # array == [5, 2, 3]
# 末尾を取り出す
last = array.pop() # last == 3, array = [5, 2]
# 末尾に追加
array.append(9) # array == [5, 2, 9]
# 末尾にリストを追加
array.extend([0, 1]) # array == [5, 2, 9, 0, 1]
```

5. ディクショナリ

他の言語でハッシュや連想配列、マップ等と呼ばれているものと似たようなものです。

ディクショナリのキーには、数値、文字列、タプルをはじめとした不変オブジェクトが指定できます。

```
dic = {} # 空ディクショナリ
dic = {'a': 1, 'b': 2}
```

ディクショナリの要素の参照と代入

```
# 要素の参照
dic['a'] # 1

# 要素の代入
dic['a'] = 100
dic['c'] = 42
```

ディクショナリに関する関数

```
# キーの取得
dic.keys() # ['a', 'c', 'b'] (辞書式順や追加順に並ぶとは限らない)
# 値の取得
dic.values()
# キーの存在確認
'a' in dic # True
# ハッシュのキーの削除
del dic['a']
```

6. 制御文

Pythonは一般的なプログラミング言語と異なり、タブやスペースによるインデント(字下げ)が意味を持ちます。

同じ深さのインデントは同じブロックに属するものとみなされます。

if文

```
if 条件:
```

```
    print 'true'
```

if～else文

```
if 条件:
    print 'true'
else:
    print 'false'
```

if～elif文

他の言語のようにelse ifではなくelifです。

```
if 条件1:
    print 'if1'
elif 条件2:
    print 'if2'
```

while文

```
i = 0
while i < 5:
    # 処理
    i += 1
```

いわゆる無限ループには while 1:が主に使われます。

for文

```
for i in xrange(5):
    # inの右に渡されたリスト等の要素がiに順次代入されて処理される
    print i
```

複数回繰り返し

```
for i in xrange(1000000):
    print 'SPAM ' # 1000000回繰り返し
```

xrange()はrange()と似ていますが、最初からリスト全体を生成するわけではないのでこういったループ目的に多用されます。

7. 関数

関数は以下のようにして記述します。返り値はreturn文で返します。(return文がない場合はNoneが返ります)

```
def add1(num1, num2):
    return num1 + num2
```

lambda式で無名関数を作成できます。lambda (引数): (python式)の形で表現します。

```
add2 = lambda x, y: x + y
add2(1, 2) # 3
```

lambdaによる無名関数は、小規模な関数を作成して使用したり他の関数の引数とするのに向いています。

無理にlambdaで関数を記述しようとすると可読性も落ちるので、ある程度以上規模の大きい関数は通常の形式で記述したほうがよいです。

8. ファイル入出力

ファイルコピーの例を示します。

```
# ファイルを読み込みモードでオープン
orig = open('orig.txt')
# ファイルを書き込みモードでオープン
copy = open('copy.txt', 'w')
for line in orig:
    # lineは改行文字を含む
    copy.write(line)

orig.close()
copy.close()
```

9. 雑多なtips

Pythonの真偽値

pythonにおいて、None、False、ゼロとして解釈できる数値(0, 0.0, 0j)、空文字列(''')、空リスト、空タプル、空ディクショナリは偽と評価されます*1。

その他は真と評価されます。

切り捨て

```
num = int(1.5) # 1
```

小数点以下n桁目で丸める

```
num = round(3.14159, 3) # 3.142
num = round(1250, -2)   # 1300.0  桁数に負数も指定できる
```

文字→数値

```
# 10進表記で変換
int('100') # 100
# 基数を指定して変換
```

```
int('FFFF', 16) # 65535
# 小数
float('1.0') # 1.0
```

数値→文字

```
str(1234) # '1234'
```

文字列→配列

```
list('hoge') # ['h', 'o', 'g', 'e']
```

文字列をそのまま出力したい

print文を使うと空白や改行が付加されて出力されますが、それを望まない場合。

```
sys.stdout.write('hogehoge')
```

コマンドライン引数

sys.argvにリスト形式で格納されています。sys.argv[0]はスクリプト自身を指します。

```
import sys
print sys.argv
```

後置のif～else

以下のように3項演算子のような書き方ができます。

```
result = 'true_value' if 条件 else 'false_value'
```

クラス定義

クラスはclass 名前:で定義します。

```
class Person():
    def __init__(self, name, age):
        self.name, self.age = name, age
        self.address = None

jhon = Person('Jhon', 15)
print jhon.name
jhon.address = 'USA, Earth'
```

map

mapを使ってリスト等の各要素を変換できます。リスト内包表記も使えます。

```
map1 = map(str, range(5)) # ['0', '1', '2', '3', '4']
map2 = [str(x) for x in range(5)] # ['0', '1', '2', '3', '4']
```

filter

filterを使ってリスト等の各要素から条件に合うものを抽出したリストを生成できます。リスト内包表記も使えます。

```
filter1 = filter(lambda x: 'cat' in x, ['cat', 'dog', 'catalog']) # ['cat', 'catalog']
filter2 = [x for x in ['cat', 'dog', 'catalog'] if 'cat' in x]
```

タプル代入、リスト代入

タプル代入、リスト代入と呼ばれる代入方法があります。

```
(a, b) = (1, 2) # a=1, b=2
[c, d] = [a, b] # c=1, d=2
a, b = b, a # a, bの値を交換
```

タプルやリストとして評価されるような式を右辺にして用いると便利です。

```
year, month, day = '2010/01/26'.split('/')
```

例外処理

raiseで例外を投げ、try～except(～finally)文で捕捉します。

```
try:
    #何かの処理
    if 条件:
        raise Exception();
except Exception, e:
    # 捕捉したException型の例外はeに代入される
    (略)
finally:
    # 例外の有無に関わらず実行されるブロック(必須ではない)
    (略)
```

*1:__len__()が0を返すようなオブジェクト、__nonzero__()がFalseを返すようなオブジェクトも偽と評価される

コメントを書く

 dplusplus 2010/01/29 04:17

入力について少々追加。

 aroma_black 2010/01/29 09:47

改行しない出力は
sys.stdout.write('hogehoge')
だと面倒だと思うので、
print 'hogehoge',
とやると簡単にできます。

 dplusplus 2010/01/29 13:43

タプル/リスト代入について補足。

>aroma_blackさん
コメントありがとうございます。
カンマ付きのprint文は場合によって空白を付与するため書きませんでした。
文が紛らわしかったので直しておきます。

odakin 2010/01/29 17:14

typo発見。
『num = 5 % 2 # 3 (剰余をとる)』
↓
『num = 5 % 2 # 1 (剰余をとる)』

dplusplus 2010/01/29 18:53

>odakinさん
指摘ありがとうございます。修正しました。

ekaf 2010/01/29 21:25

How To Become A Hackerを読んでいたのですが、、、とってもちょうどいい！ありがとう！

dplusplus 2010/02/01 03:26

ブコメつっこみを参考に追加修正
・タプルの説明追加
・lambdaによる無名関数の説明
・サンプルコード増量
・ほか雑多な修正

>ekafさん
お役に立てたならうれしいです。

トラックバック - http://d.hatena.ne.jp/dplusplus/20100126/p1

- [Python][Mercurial]巡回
- tosh-tの日記
- 燈明日記 - 基礎文法最速マスター
- /Users/mizchi/workplace/ - Python+iPython+Emacsの設定 (Mac)
- 404 I've not found - Fri, Jan 29
- たぶん日刊・夫婦でわしだんご - 本日の twitter
- 何かしらの言語による記述を解析する日記 - VBA基礎文法最速マスター
- 何かしらの言語による記述を解析する日記 - Java基礎文法最速マスタ...
- 医者を志す妻を応援する夫の日記 - Brainf*ck基礎文法最速マスター
- think and error - Haskell基礎文法最速マスター
- CX's VBScript Diary - その他の基礎文法マスター
- CX's VBScript Diary - VBScript 基礎文法最速マスター
- 何かしらの言語による記述を解析する日記 - Bash基礎文法最速マスタ...
- なんとなく日記 - JavaScript基礎文法最速マスター
- CX's UWSC Diary - UWSC 基礎文法最速マスター
- 燈明日記 - 基礎文法最速マスターぞくぞくキターーー！
- surume000の日記 - プログラミング言語基礎文法最速マスターまとめ
- はてなかよっ！ - D言語基礎文法最速マスター
- 鼻歌SEOツール - はてなアンテナ ページが更新されました
- きまぐれメモ - 各種言語による基礎文法最速マスターまとめ
- shikaku's memo blog - ○○基礎文法最速マスター
- 永遠に未完成 - Vimスクリプト基礎文法最速マスター
- きまぐれメモ - 各種言語による基礎文法最速マスターまとめ
- [Flash] ActionScript 3.0 基礎文法最速マスター
- (rubikitch loves (Emacs Ruby CUI)) - Emacs Lisp基礎文法最速マス...
- どうでもいい情報置き場 - Whitespace基礎文法最速マスター
- [Diksam]Diksam基礎文法最速マスター
- Life like a clown - はてな的プログラミング言語人気ランキング
- なんとなく日記 - 基礎文法最速マスターシリーズのまとめ
- なんとなく日記 - 汎用スクリプト言語

リンク元

- 588 http://b.hatena.ne.jp/
- 449 http://b.hatena.ne.jp/hotentry
- 330 http://reader.livedoor.com/reader/
- 328 http://b.hatena.ne.jp/hotentry/it
- 313 http://f41.aaa.livedoor.jp/~suraisu/rdl/
- 310 http://pipes.yahoo.com/pipes/pipe.info?_id=faa858a20082ef6d25ad27557e37e011
- 162 http://www.google.co.jp/reader/view/
- 156 http://d.hatena.ne.jp/
- 128 http://www.google.com/reader/view/
- 108 http://twitter.com/

<[Web][つれづれ]WIDEのIRCサービ...