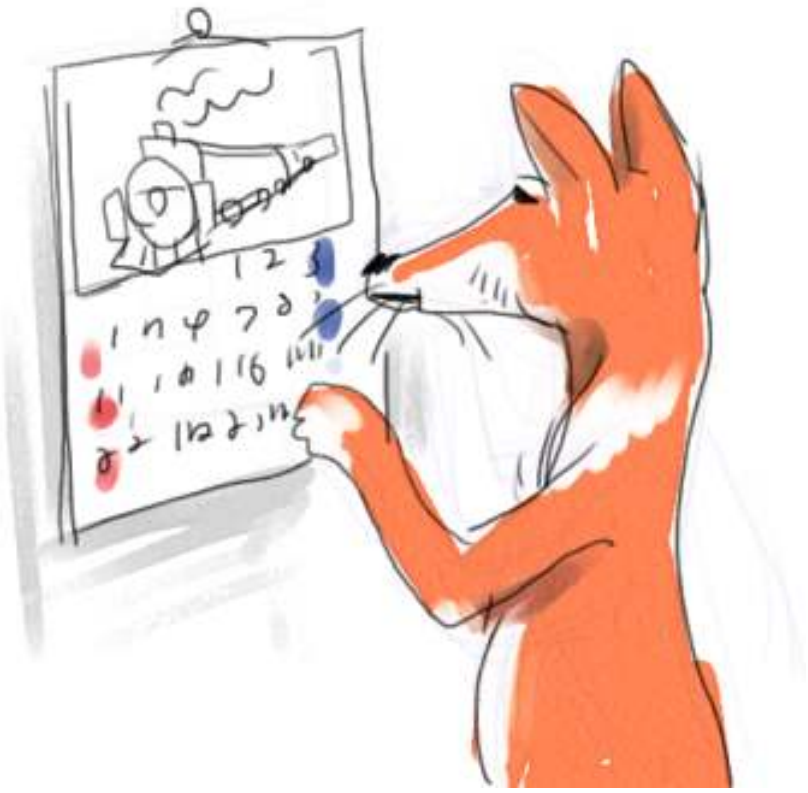
目次へ

# 「n営業日後」を計算してみる



さて、日付を扱う方法についてちょっと分かったので、早速これを応用した練習問題をやってみようかと思います。

よく、「翌営業日」とか「何々営業日後」とか言いますね。土日とか祝日とかの休日を「除いた」日数を足したあとの日付のことです。

そんなのを人間がいちいちカレンダーを見ながら計算するのはアレなので、コンピュータがそいつを計算してくれるといろいろ嬉しいかも知れません。練習問題としても手ごろかなと思うわけです。

突然ですが、下のコードをコピペでもダウンロードでもいいので手元において、実行してみてください。アイコンのあたりをクリックするとダウンロードできると思います。途中、なんとも異様な部分がありますが気にせずに。説明はあとでします。

businessday.py

```
# -*- coding: cp932 -*-
import datetime

a = [(2010,10,11), (2010,11,3), (2010,11,23), (2010,12,23),
     (2010,12,28), (2010,12,29), (2010,12,30), (2010,12,31),
     (2011,1,3), (2011,1,4), (2011,1,10), (2011,2,11), (2011,3,21)]
holidays = []
for i in a:
  holidays.append(datetime.date(i[0], i[1], i[2]))
```

```
#
# MAKING THIS FUNCTION IS LEFT FOR YOU!
#
def get_n_business_days_forward(d, n):
  return d


import base64
import zlib
import pickle

a = """eJxtlO2S4zAIhfc5Sa+mBAj9HKGPMftZZJH71wgkOzyShStl99eA3kPI/vn3LI+fXy5UHr921
fXzCPf6eD3p8TchEpG+EP5AygPuXO9BhHIYmguqnxADNBakCWqpZocaQiNns5L6ZyAoez14PUeArBbaf
1tPyPN6pImReKWbdyTya+lYkJISC/doC4pq7wwSGTeEkZG7pOufzBMSpDRFWr8Lqgj1r5AiNJOLkAlOL
RW+rgaUGUw9UeKFZWokSt3B9wM2Z2gmKhfPbK1ZEjWjy1dGTrIHm4/xlpGT8CyZ8s2QhGdNPnv1nJTnH
tfIdrOgpDyD9BfUsEUpBVoaL6gjJFGrK1Dqdc2r877iiVSPqzuRpCAzE+NtJZTq/tCcLZ9wwqBlfDwYJ
YnSODqu2qUmqicZvGVEEzW/Uu1zMtQwGGynSlLdGgZXaN5InjGaKY+VdLeGOTiIrKiahH83TJhWNQkvH
zPNZ/Gl+37G1Og7iNlcBZD6LVcFpGEUt64qIONa/HW7iAYE3YPz3NvgrBOZvvyPxQxkamI81OSm4aLcC
i3I3KfLYaweJZQv1byZJDFfI/MMfmdQ4zU2zu7enlj/KGq85sG7EXOeLAQ1tgOKPjKhynY8tRDG9o8Gk
fn24n1vgOOHMBK69Nr5OgHRYMMZIKOAOUNPnFIaATExhtvdGKslXJBbOAQZwUw+O1tFRlMcZxSZHqWR7
VNryMzEeM2oMBPKxy4OKswS2u9mUOL3GRSYjiLDQDnl9KCyeIhYsAnYGQgJPsh2qgsQUTO5TvUKSA97T
k5bdQVkhmOgZyL3hsVSYjxTROZwzT4C+kBGw366cO1kehhscjbUKMjMwNSt3kB9OZW8w6DCYLccKwdqD
HbfDIrM/RuDKnMu2etBmSVuq7q37wgq12vfhzA2RMcApobWqrv5xgSiBavq2ZqzADKwFHdzEiD7kyXc2
1k3GRkONtTj+BRkKpbjO2RWZFqwqp7jbioyI3TXnauhfqlm9pqTxpzi+NpRYzs8bjt1v6JMFJnjeXcQK
qiynR4cGHYGZbbTQwJTnQky63U6pFQCSAO9eiMVkNgWetqCigIzvjINGGuMKA5vqCPECdoVDYQq6rM/x
8pEqIVe1TOdiApCA93wF2IiQhlL6Nb2OJ+RSWpOlu1sKDbXuE/bnt5EKDfHV7i2tzsRyv3+/qXzymcMy
iOFrZVddpB7f9f3uDL/GqUBTJzgb2YCo2G4tOsjekn95z8GrlZ+"""
questions = pickle.loads(zlib.decompress(base64.b64decode(a)))

num = 1
for i in questions:
  orig = datetime.date(i[0], i[1], i[2])
  distance = i[3]
  answer = datetime.date(i[4], i[5], i[6])
  cand = get_n_business_days_forward(orig, distance)
  print "(Q%d) %s + %d business day(s)" % (num, orig, distance),
  if cand == answer:
    print "is %s, CORRECT!" % (answer)
  else:
    print "should be %s, not %s..." % (answer, cand)
    print "YOU FAILED."
    break
  num += 1
else:
  print "YOU PASSED ALL QUESTIONS!!"
```

実行してみて下のような出力が得られるようなら、とりあえず成功です。

```
(Q1) 2010-10-04 + 1 business day(s) should be 2010-10-05, not 2010-10-04...
YOU FAILED.
```

このスクリプトが何なのかを説明します。実はこれ、まだ未完成です。もっと詳しく言うと、get_n_business_days_forward関数が極めて手抜きな形でしかありません。

get_n_business_days_forward関数は、引数dには日付オブジェクト（もうポケモンとか言わないよ）、引数nには数字を渡されて、日付dのn営業日後の日付をreturnするというつもりで作り出しました。こいつがちゃんと正しく書ければ、YOU FAILED なんていうメッセージで終わらずに、ちゃんと合格のメッセージが表示されるはずなのです。

import base64…とかそこらへんから下のスクリプトは、特に理解する必要はありません。筆者があらかじめ作った、「get_n_business_days_forward関数が正しくできたか試してやろう君・一号」というもので、色々なパターンでこの関数を呼び出して、期待通りの結果が得られるかをテストしてくれるという~~イヤな~~スゴい奴です。問題と解答集が簡単に見られないように、グチャグチャな感じの文字列にいわば暗号化されているというわけですね。全部で何問入っているのかは秘密ですが、この「試してやろう君」が繰り出すすべての問題に、あなたのget_n_business_days_forward関数が正しい回答を返すことができれば、この練習問題はクリアですよ。

で、現時点で実装されているこの関数の定義を見てみましょう。

```
def get_n_business_days_forward(d, n):
  return d
```

日付dを受け取って、そして、あとは引数nとか知ったことでなしに、そのままその値を返すというだけのつくりとなっております。「何日後」とかそういうレベルではありません。そりゃうまくいくわけないよ。

だから、「試してやろう君」の出題する第一問にさえマトモに答えられませんでした。

```
(Q1) 2010-10-04 + 1 business day(s) should be 2010-10-05, not 2010-10-04...
YOU FAILED.
```

「2010年10月4日の『1営業日後』はいつか」という出題に対して、この関数は「はい、それは2010年10月4日です!!!!」と威勢よく答えて、「違うよ、10月5日だろ…」といわれてテスト失格、というのがこのメッセージの意味です。

じゃあ、覚えたばかりのtimedeltaオブジェクトを使って、単純な日付の足し算でもしてみましょうか。下のようになるはず、と解るでしょうか。確認してみてね。

```
(略...)

def get_n_business_days_forward(d, n):
  return d + datetime.timedelta(n)

(略...)
```

n日後、というのをあらわすtimedeltaオブジェクトを作って、それをdに足して返す。これで、一番単純な形の実装はできます。これで「試してやろう君」にチャレンジするとどうなるか。下、実行結果。

```
(Q1) 2010-10-04 + 1 business day(s) is 2010-10-05, CORRECT!
(Q2) 2010-10-04 + 3 business day(s) is 2010-10-07, CORRECT!
(Q3) 2010-10-04 + 0 business day(s) is 2010-10-04, CORRECT!
(Q4) 2010-10-14 + 3 business day(s) should be 2010-10-19, not 2010-10-17...
YOU FAILED.
```

お、3問はクリアできるみたいだね。でも4問目であえなくアウト。

これは、2010年10月16日、2010年10月17日が土日ですから、そいつを飛ばしながら3日進めなければいけなかったということですね。10月14日の3営業日後といえば、10月17日じゃなくて、10月19日が期待された答えだったと。

日付オブジェクトから曜日を導き出す方法は前に説明しましたので、ここらへんをうまく使うと第4問も乗り切れるでしょう。

で、もうひとひねり。

土日以外の祝日は、pythonは素では判断できません。「試してやろう君」は祝日がからんできたときの問題も準備していますから、これもちゃんと加味した実装ができないと、やっぱりいつか失格になってしまいますよ。

祝日のリストはあらかじめ準備しましたので、こいつを活用してくれてよいです。スクリプトのはじめのほうに書いた部分で、holidaysという名前のリストを作っておきました。リストの中には祝日を示すdateオブジェクトがひとそろい入っています。「試してやろう君」が出してくる問題は、せいぜいこのあたりの日付の範囲だけですから、充分足りるでしょう。

というわけで、まずは出題してしまいましょう。

## 練習問題06

**【練習問題：06】「試してやろう君・一号」のテストを乗り切るよう、get_n_business_days_forward関数を完成させよ。**

この関数の仕様をもうちょっと詳細に述べておきますね。

- dにdate形オブジェクト、nに整数を受け取る関数である。
- dateオブジェクトを返す関数である。
- 土曜、日曜、祝日を除いて、指定日数だけ先の日付を計算すること。祝日は、holidaysリストに含まれるものがすべてと前提してよい。
- 0日後、というパターンがありうる。このときは、受け取った日付をそのまま返せばよい。もし引数dが土日祝日だった場合もそのまま返してよい。

少なくともこのテキストでしかpythonを習ってない方は、たぶん非常に難しいと感じるでしょう。いくつかの新しい構文をお教えするべきときがきたようです。while, continue, break, あとはリストに使う in という構文です。（知ってる人には、ええ今さら！？と驚かれそうですね。でも、基本的に、必要になったタイミングごとに新しいことを紹介する、というパターンでやってみたいのです。）

## while構文と continue, break

whileってのは、繰り返し処理を書くやりかたの一つです。for ってのはもう知っているはずですね。これに似たものです。whileの書き方をひと言でいえば下の通り。

```
while（真偽値）:
    なんかの処理...
```

真偽値を調べる部分がwhileの直後に書くもので、これがTrueになる間だけ、下に続く「なんかの処理」を続けます。

簡単な例を挙げればもうちょっと解るでしょうか。

```
c = 0
while c < 10:
  print c
  c = c + 1
```

cには最初0を入れておきます。で、c < 10 という条件を満たす間だけ、cの内容をprintして、ひとつ足す、という動作を繰り返します。下のような表示になるだろうな、ということが簡単に想像できるでしょう。

```
0
1
2
3
4
5
6
7
8
9
```

c = c + 1 の結果 c が10になったら、次のwhileは成り立ちません。だから繰り返し処理は終わりです。

for で同じようなループを作ろうと思ったら、あらかじめ 0 から 9 が入ったリストを作って、それでforを作ることになりますね。

```
a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
for i in a:
  print i
```

こんな感じで、while と for の似ているところと違うところを確認しておいてください。

ついでに紹介しますが、a = [0, 1, 2, 3, …] なんてものをもっと簡単に作る方法がありますので、覚えておくとよいです。range といいます。

```
>>> a = range(10)
>>> a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> a = range(1000)
(結果省略)
```

こんな感じで、range は、0から始まってnの直前で終わる、連続した数字でできたリストを簡単に作ってくれます。range(10)だと結果として10個の値の入ったリストを作ってくれます。だから10回の繰り返しは下のように書いてもいいですね。

```
for i in range(10):
  (何かする)
```

繰り返しのためにrangeを使う場合は、rangeのかわりにxrangeを使うほうがいいんだけど、まあ、ループ回数が少ないうちはどっちでもいいです。（ループが1000回とか10000回とかになったら、rangeのかわりにxrangeと書いたほうがいいです。省メモリになるから…と、今はこの程度の説明しかできませんが。）

で、continueとかbreakは、このループの流れをちょっと変更したいときに使う文句です。あ、別にrangeを使うループに限らず、どんな形のループでもOKです。

break から行きましょう。これは、今動いているループを強制的に抜け出す命令です。

```
for i in range(10):
  if i > 2:
    break
  print i
```

この例を見ましょう。10回繰り返すつもりで書いたfor命令ですが、実行結果は下のようになります。

```
0
1
2
```

iが2より大きくなったら、breakが実行されます。これでループは終わり。そういうこと。

使い道としては、例えば、リストの中に10より大きい数が少なくとも一つあるかどうか、を調べるという仕事があったときなんかに有用でしょう。下のような例を想像しましょう。

```
a = [7, 5, 9, 2, 12, 3, 0, 3, 2, 1, 11]
for i in a:
  if i > 10:
    print "ATTAYO!"
    break
```

ひとつづつリストの中を調べていますが、最初に10より大きい数が見つかったら、もう続きをやる必要がありません。こんなときに、breakを使ってループをやめてしまえばいいというわけですね。

で、次は continue です。こっちの機能は、ループの次の回に進むときの命令です。「今回はもういいから、次行こう、次」って感じ。例を見るのが早いでしょう。

```
a = [7, 5, 9, 2, 12, 3, 0, 3, 2, 1, 11]
for i in a:
  if a > 10:
    continue
  print i
```

リストの中をひとつづつ調べていって、10より大きい数字があったらcontinue、と書いています。continueすると、ループの中のそれより下の記述（ここではprint）をすべて飛ばして、forの直後から「次の回」をはじめるのです。実行結果は下のようになります。

```
7
5
9
2
3
0
3
2
1
```

10より大きい数は、continueによって処理を抜かされたというわけですね。

break と continue の機能は、ざっとこんなところです。今は for の中でそれぞれを使いましたけど、while でつくったループ内でも同様に動作しますよ。

while と break を使った、とてもよくある書き方のことも紹介しておきます。危険も伴いますけど。

```
while True:
  何か...
  何か...
  if 何か...:
```

```
    break
  何か...
```

whileの次の真偽値は、Trueに決めうちされています。どれだけループを繰り返しても、Trueは Trueで変わりません。つまり、これはは「永久ループ」を作る書き方です。永久です。コンピュータの配線が焼き切れるか、電源が切断されるかしない限り、世の終わりまででも回り続けます。（まあ、[Ctrl] + [C] で実行中断すれば終わりますけど。）

で、この永久ループを抜け出すときに、breakを使うというわけです。何か一定の条件を満たしたらbreak、とかそういう書き方をしておけば、このループも有限で安心なものになりますね。

この書き方は、たとえば「土日祝日が続く限り、日付を永久に一日ずつ進めるんだけど、それ以外の平日につきあたったらbreak」とかそんな用途に使えるかな...

## inでリスト内を調べる

あと、これも今まで紹介しなかったのが遅すぎるくらいの話ですが、リストに対する in について。

これはまあ簡単な話で、リストの中にある値が含まれているかどうかを調べるための簡単な書き方です。

```
>>> a = ['apple', 'orange', 'banana']
>>> 'orange' in a
True
>>> not 'orange' in a
False
>>> 'lettuce' in a
False
>>> not 'lettuce' in a
True
```

「このリストにこの値は入ってますか」という質問についての真偽値を得ることができます。notをつければ、「このリストにこの値は入っていないですか」という質問についての真偽値です。まあ、これ以上の説明がいらないくらい簡単な話ですね。

祝日リストに「この日付」は入ってるかな、ということを知りたいときに使って下さい。dateオブジェクトの有無でも、もちろん調べられますよ。

## 真偽値に and や or をつける

そういえば、これの説明もまだしたことがなかったので、この機会に紹介しておかないと。

if とか while でつかう、真偽値の問い合わせのことです。TrueやFalseで値が帰ってくるアレですね。

それらを複数組み合わせて新たに真偽値を計算するというやりかたです。もっと簡単に言うと、「何々、または何々」とか「何々、かつ何々」という表現をする方法です。ブール演算とかいう用語を使う人もいますが、まあ、知らなくてもいいと思います。

でも、実際は何というほどのことでもないので、例だけを見て、直観的に理解しちゃってください。

```
if (score >= 60) and (score < 80):
  print "あなたはB判定"
```

上の例が、「何々かつ何々」の例。scoreという変数に、きっとテストの点数か何かが入ってるんでしょう。その点数が60点以上で「かつ」80点未満のときはB判定、という判断を書きました。この and という書き方がソレです。

80点以上のときはA判定なのかな。まあ、これは今適当に考えた例なので、それ以上の設定は特にありません。

で、下が「何々または何々」の例。

```
if (tenki == 'rain') or (tenki == 'wind'):
  print "仕事休む"
```

天気が雨か、または天気が風の場合、仕事休みたい人の頭の中はこんなふうですね。or でふたつの真偽値問い合わせをつなぐのがミソです。

この and と or をさらに組み合わせて、複雑な条件を組み立てていくこともできます。

```
if (今日は平日) and (今は8時過ぎ) and (今は20時前):
  print "xx号車は女性専用車両"
```

こんな風に三つ以上条件を連ねてもいいし、

```
if (千葉で2連敗) or ((千葉で1勝1敗) and (ドームで1敗)):
  print "日本シリーズ準優勝おめでとう"
```

こんな風にカッコを組み合わせて難しい条件にしてもいいです。

あと、not で条件を逆転させることもできて、

```
if (not テレビがある) and (not ラジオがある):
  オラこんな村いやだ
```

こんな感じの記述ができます。（この場合、テレビとラジオのどちらもなければ、吉幾三は東京に出ます）

## あとは組み立てるのみ

今回の練習問題のために新たに必要になるかも知れないプログラム要素は、こんなものかなあ。

あとはもう、こいつらをどう組み立てたらやりたい仕事になるか、というパズルみたいなもんです。チャレンジ乞う。