目次へ

datetimeモジュール

datetimeモジュールは使い方をおぼえると極めて便利なので、ぜひ知っておきたいものです。名前から想像できるとおり、dateとかtimeを扱う、つまり日付や時間を扱うことができるモジュールです。

使うためにはインポートが必要です。下のよう。対話シェルでも試せますよ。

```
>>> import datetime
```

まずは使い方の例をいくつか下に示してから、あとでちゃんとした説明を加えましょう。

```
>>> datetime.date.today()  <--- これは「今日」を知りたいときの書き方
datetime.date(2010, 10, 26)
>>> datetime.datetime.now()  <--- これは「現在時刻」までを含めて知る書き方
datetime.datetime(2010, 10, 26, 22, 41, 38, 70000)
>>> a = datetime.date(2011, 3, 1)  <--- 年、月、日を指定して「日付」を構成する
>>> a
datetime.date(2011, 3, 1)
>>> a.year  <--- 変数に入れておいて、年だけを見る
2011
>>> a.month  <--- 同様、月だけ見る
3
>>> a.day  <--- 日だけ見る
1
>>> a.weekday()  <--- その日の「曜日」がわかる。1は火曜日のこと
1
>>> a = datetime.date(2011, 3, 2)
```

```
>>> a.weekday()
2  <--- 2011年3月2日は水曜日らしい。2は水曜日のこと
```

変数aに何かを入れることができたところにまず注目します。datetime.date という関数で(2011, 3, 1)という数を処理したあとの「何ものか」が変数aに入りました。これは何でしょう。単なる数でもありませんし、文字列でもありません。リストでも辞書でもありません。ましてや「ファイルを開いているという目印」とかいったものでもありません。

じゃあ何だ。まさか、また新しい種類の何かか。

すみませんがそのとおりで、このとき、変数aには「日付オブジェクト」とでも呼ぶべきものが入ります。格好つけて「オブジェクト」なんて言いましたが、訳せば「モノ」ですね。

また新顔なのか。ええい、一体、結局のところpythonでは変数に何種類の「モノ」が入るんだ。

そろそろ、こんなに色々な種類の何かが存在して、なんでも変数に入れることができるという点について、腰を据えたまじめな説明がいるかもしれません。

データ型

整数とか小数とか、文字列とか、リストとか、辞書とか、また今回出てきた「日付」とか、そういうデータの種類のことを「データ型」といいます。「素」のpythonが扱えるデータ型はたくさんありました。リストとか辞書とか、今まで出てきたそういうやつです。今回、datetimeモジュールを使うことで、「日付」という新しいデータ型を使えるようになりました。その他のモジュールを使うようになると、もっといろいろなデータ型が新たに出てきて、それぞれ異なる使い方に従うことになります。

実は、その上、新しいデータ型を作ってしまうことさえできます。

つまり、pythonにおいては、事実上無限の種類のデータ型があらわれるということです。

聞くだにガッカリする話ですが、まあ気を取り直しましょうよ。pythonの全モジュールをマスターしようとしているわけでもないですから、その意味では、おぼえるべき「データ型」は実際は限られています。データ型の種類がいろいろあるとはいっても、ある程度お決まりの扱い方だったりして、そのうち別に戸惑わなくなっていきますし。

せっかくなので、データ型がいっぱいあるから楽しいな、くらいに考えましょう。今回出てくる日付型を使いこなせば、カレンダー的な計算や、さっきの例でも見た曜日の計算なんかが簡単にできる機能が手に入るわけです。いわば、新しいデータ型をおぼえることは、新しい機能をひとつ身につけることなんですよ。

この際、データ型をポケモンなんかに例えてしまってもいいんじゃないかな、とたった今思いつきました。(ちょっとググってみたら、同じようなアイデアを持った人もいるらしい。よし、そんなにハズしてなさそうだ)

好きなポケモンを、なんでもいいのでひとつ思い浮かべてください。そいつをイシツブテとかフシギダネとか呼ぶ代わりに、date型と呼びましょう。同じ種類のポケモンが現れたら、そいつも

date型ね。その種類のポケモンのワザは、「年、月、日を数字で覚えていて、カレンダー計算ができる」というものです。バトルには役に立ちませんが、でもイメージはできそうですね。ポケモンごとに別の年月日をおぼえていることが可能なのも、納得しやすいですね。a変数には、このポケモンが一匹くっついているというわけ。まるでモンスターボール…いや、そこまで言うのはやめましょう。別に不都合はないんですが、これ読む人がみんなポケモンに詳しいわけでもないからね。

ついでに言っておくと、今までに出てきたリストとか辞書とかもデータ型です。リストはlist型、辞書はdict型って言います。種類ごとにいろいろな「機能」があることも見てきましたね。知ってるポケモンを適当にあてはめておいてもいいですよ。やりたけりゃ。

さて、これを踏まえて、また最初の例を見直しましょう。

```
>>> datetime.date.today()
datetime.date(2010, 10, 26)
```

datetime.date.todayという（長い）関数は、ポケモン(date型)を一匹どこかから連れてきます。僕らは、そいつがどこから来たのかを気にする必要はありません。とにかく、何もない所にポケモンが一匹あらわれました。そのポケモンは、today関数の仕様により、最初から「今日」を覚えこまされています。date型を表示するとこういう風になるのは、決まりです。print で表示すると、

```
>>> print datetime.date.today()
2010-10-26
```

となります。まあ、date型ポケモンの決まりですから、こんなものと覚えましょう。（気になるという人には、__str__メソッドと__repr__メソッドの違いなんです、と手掛かりだけ出しておきます。もちろん忘れてもいいよ）

```
>>> datetime.datetime.now()
datetime.datetime(2010, 10, 26, 22, 41, 38, 70000)
>>> print datetime.datetime.now()
2010-10-26 22:41:38.70000
```

datetime.datetime.now という（これまた長い）関数は、日付と時間を込みにしたデータを返します。実は、date型に似てますが、datetime型という、異なるタイプのポケモンです。こいつは「年月日、時分秒」までを覚えているという、高機能ポケモンです。（進化…いや何でもない。）nowっていうくらいだから、ここで派遣されたポケモンは、初めから現在の日付と時間までを覚えこんだ形で出現します。秒単位どころか、実はこのデータ型は、そのあとのマイクロ秒の単位まで持っています。ここでは0.07秒ってところか。なんという精密さだ、datetime型ポケモン。

次の例。

```
>>> a = datetime.date(2011, 3, 1)
>>> a
datetime.date(2011, 3, 1)
>>> a.year
2011
>>> a.month
3
>>> a.day
1
```

```
>>> a.weekday()
1
```

別にdate型が最初から「今日」を教え込まれている必要はなくて、好きな日付を持ったポケモンを出現させてもよいです。datetime.date関数は、それぞれ（年, 月, 日）を引数に受け取って、そいつを持ったやつを派遣してくれます。例え話に引きずられて、自分の懐からポケモンを呼びだしたように思いこまないように。あくまで、pythonが、どこからともなく連れてきてくれるポケモンです。

a.yearで、ポケモンは、自分が覚えているところから「年」だけを叫びます。month と day も同様。weekday を呼び出すには () を一緒に書く必要がありますが、これを発行すると、ポケモンは覚えている年月日から自力で曜日を計算し、そいつを叫びます。うるう年とかで計算を狂わせることもありません。0が月曜、1が火曜、あと続いて、6が日曜です。

datetime型ポケモンも、任意の値を持った状態のものを出現させることができます。そんなに用途はないかな、と思うので、今は省略しますけど。

date型は、カレンダー的に妥当な日付しか持つことはできません。下、例。

```
>>> datetime.date(2010, 4, 31)  <--- 4月は小の月だから、31日はないよ
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: day is out of range for month

>>> datetime.date(2040, 2, 29)  <--- 2040は2月29日があるけど…
datetime.date(2040, 2, 29)
>>> datetime.date(2041, 2, 29)  <--- 2041年だとダメだよ
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: day is out of range for month
```

この特徴を使えば、正しい日付と間違った日付を見分けるという仕事もできそうですね。いや、間違ったデータが来たらスクリプトがエラーで落ちてしまうじゃないか、と言われそうですが、そのうち説明する予定の「例外」という仕組みを導入すれば、スクリプトを中断させずにエラーを扱うことができるようになりますよ。

「例外」の扱い方は、次の練習問題編あたりで紹介すると思います。

余談：ガベージ行きのオブジェクト

あ、ところで、

```
>>> datetime.date.today()
datetime.date(2010, 10, 26)
```

って、特に変数を持ちだしてきて、today関数が生成した「ポケモン」を入れたりしてませんね。このとき、ここで発生したはずのポケモンに、yearとかmonthとか、そんな命令を与えたいときにどうすればいいのかな、と思ったりしませんか。

答えは、このポケモンは「虚無行き」です。技術的には「ガベージ(garbage)」とも言い、それこそメモリ上のゴミとなります。変数にしばりつけない限り、ここで生まれたポケモンと全く同一のものには、未来永劫、二度とめぐり合うことはできません。この例のように使い捨てで済ま

すならもちろん構いませんが、あとで何らかの操作をする予定のものなら、必ず変数に入れる形で受け止めておかなくてはだめですよ。

日付差分型

さて、ある日付の「翌日」を計算するという機能を、date型は持っています。どうやるでしょう。

日付型のデータは day というものを持っていますので、こいつを足し算で1増やしてみましょうか？

```
>>> a = datetime.date(2010, 1, 1)
>>> a.day = a.day + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: attribute 'day' of 'datetime.date' objects is not writable
```

dayは、not writable だそうです。新しい値を入れることはできないようですね。（そうしないと、32日とか、変な値を突っ込まれたらいやでしょうしね）

じゃああれかな、直接数字を足し算したらどうかな。

```
>>> a = datetime.date(2010, 1, 1)
>>> a + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'datetime.date' and 'int'
```

date型とint型（数字のデータ型のこと）の足し算はサポートしてないよ、と明示的に怒られました。

このモジュールが提供する日付足し算は、あらかじめ「日付差分型」というデータ型のものを作っておいて、そいつを足すことと決められています。ちょっと面倒な感じかな。「日付差分型」は、また新しい種類のポケモンです。ポケモンなんだから我慢してください。新種ゲットじゃないですか。

```
>>> d = datetime.timedelta(1)
```

これで、変数dに、「1日足す」っていう意味のデータが入りました。timedelta型っていいます。こいつは、date型に足し算できます。

```
>>> print a + d
2010-01-02

>>> print a - d
2009-12-31

>>> print a + (d*2)
2010-01-03
```

ね。

面白いのは、引き算や、掛け算までもできるようになったという点です。引き算なんて、ちゃんと去年の大晦日が計算できました。チョイ役的なポケモンですが、なかなか使えそうじゃないですか。

なんでdatedelta型じゃなくてtimedelta型なの？と聞かれそうな感じもしますが、timedeltaは時分秒までの細かさの差分もサポートしているからです。まあ、そんな精度の計算も、使うべきときがきたら使ってみましょう。今のところはまあいいか。

timedelta型が出現する第二のパターンは、日付と日付の引き算をするときです。（日付と日付の足し算はできません。）date型からdate型を引き算すると、その差分の値を持ったtimedelta型が発生します。たぶん直感的にOKですよね。例は下。

```
>>> birth = datetime.date(1982, 10, 8)
>>> kyo = datetime.date.today()
>>> sa = kyo - birth
>>> sa
datetime.timedelta(10245)
>>> sa.days
10245
```

1982年10月8日生まれの人は、（これを執筆している時点で）10245日生きているんだな、ということになります。

あとでこの10245という数字を他の用事に使いたいときは、timedelta型のデータに .days という指定をすれば得られます。

datetimeの、大体の使い方はこれで言い切ったかな。表示フォーマットをもうちょっとキレイに直す機能もあったりして、まだ説明したいところも若干残ってます。でも練習問題編のあたりでやることにしましょう。

なんせ、datetimeモジュールの紹介のついでに行った、「データ型」というものの説明が重要なので、なんとか呑みこんでおいてください。ポケモンのたとえでもオッケーです。人前でうっかり「date型ポケモンが…」とか言い出さないように注意する必要はありますけどね。

ついでにポケモンのたとえを借りて用語をもうちょっと紹介させてください。「クラス」と「インスタンス」です。クラスってのは、データ型そのものの呼び名のことです。ピカチュウなら、ピカチュウという「種類」のこと。インスタンスってのは、それの実体のひとつです。「サトシが持ってる、このピカチュウ」がインスタンスです。データ型と言ったときはクラス、変数に実際に入っているときは、それはインスタンスですね。きっといつか役に立つ用語ですので、今のところは「ふーん」くらいでよいので、機会があったら思い出せるくらいにしておいてください。

もういっこ。

モジュールの使い方は、モジュールごとに使い方マニュアルが得られますから、それを適宜見ながらスクリプトを作れればよいです。標準モジュールとはいえ、pythonの「外部」で作った拡張機能みたいなものなんですから、そいつをなんでもかんでも覚えこんでしまうことはありません。むしろ大事なのは、効率的なマニュアル検索の技術かもしれませんよ。まあそれにしても、

このテキストでは、丸暗記しても損のない、「鉄板」な感じに確立したモジュールばかりを扱うつもりですけど。

その意味では、まだ紹介してないけど、dir関数の使い方を覚えておくと、いろいろ度忘れ対策として便利です。

```
>>> a = [1, 2, 3]
>>> dir(a)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__delsli
ce__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getit
em__', '__getslice__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__',
 '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__r
educe__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__'
, '__setitem__', '__setslice__', '__sizeof__', '__str__', '__subclasshook__', 'a
ppend', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort'
]

>>> type(a)
<type 'list'>
```

ここでは a にはリストが入っています。dir( )でそれを囲んで実行してみると、そのデータがどんな「.ナントカ」の機能を持っているか、全部出してくれます。アンダーバー記号とかがくっついたものは特殊なので無視してもいいですが、他は参考にできます。こういうのをたまに見ながら、「ああこの命令が使えたな」とか「この命令の意味が気になるので調べてみよう」とか思うわけです。リストひとつ取っても、まだ紹介してない命令がいくつかあるようですね。でも筆者はこの一覧を見ながら「まあ、必須なのは全部説明したかな...」とか考えてたりしますけど。

ついでのついでに最後に例に挙げたのは、type関数の使い方の例。これは、変数の中身が何タイプのデータ型かを知る手段です。まあ、変数の中身が気になったら、こんな風に書いて問い合わせてみるといいよ。