目次へ

文字コードのことについて、ちょっと補足

直前のテキストで、'こんにちわ' という文字列の三番目の文字として'に'を取り出すのがちょっと難しい、という話をしかけて終わりました。これの説明をしてみます。

何はともあれ、'こんにちわ'という文字列がそもそも何文字だと見なされているのか、まずは対話シェルから調べてみましょう。

```
s = 'こんにちわ'
>>> len(s)
10
```

10文字だって言ってますね。ふーん…（Windowsでコマンドプロンプトという環境以外では、違うことがあります）

もうお分かりの方もいることでしょう。文字列って、python的には、エンコードされたあとの状態で入っているんです。この場合では、シフトJISです。シフトJISでは、'こんにちわ'という文字列は、[130, 177, 130, 241, 130, 201, 130, 191, 130, 237] というバイト情報の並びとなります。

だから、'こんにちわ'の左から三番目の文字がほしいです、とバカ正直に指定すると下のようになるわけ。

```
>>> s = 'こんにちわ'
>>> s[2]
'¥x82'
```

'\x82'って何じゃらほい。あとで説明しますが、これは'a'とか'b'とか、または'\n'とかいったのと同じで、一文字分の文字です。ord関数を覚えていますね。一文字に対応する数字を出してくれるやつです。

```
>>> ord('¥x82')
130
```

出たでしょ。

で、この130っていう数字は、さっきの10個並んだ数字の「左から三番目」の情報です。pythonは、文字列の「実際の文字の単位」じゃなくて、「エンコード後の単位」をもとに区切ろうとしたわけです。文字列って、人間にとって偶然文字と解釈できるだけであって、内部的にはバイト情報のカタマリに過ぎません。

で、これを踏まえて、「こんにちわ」から「に」を取り出す方法を考えます。

最初の方法

まずは、安直だけど、気をつけて使えばちゃんと有効な方法。日本語一文字が2バイトになるんなら、文字の指定も全部二倍したらいいという発想です。「こんにちわ」が10文字だというなら、「5文字目から6文字目」を抜き出せば「に」ということになるんじゃないかな。[:]の記法を思い出しながら。

```
>>> s = 'こんにちわ'
>>> s[4:6]
'¥x82¥xc9'
```

あれえ。変なのが出た。だめなのかな。

いや、やり方は合っています。明示的にprintをつけくわえてみて下さい。ちゃんと期待どおりの表示になりますよ。

```
>>> s = 'こんにちわ'
>>> print s[4:6]
に
```

おお、ちゃんと出た。

ちゃんと出たけど、\x82 とか \xc9 とか、それって結局のところ何なの、という声が聞こえてきそうです。そろそろちゃんと説明したほうがいいかな。

文字列で何でも表現する

以前、改行文字は '\n' だってどこかで説明しました。タブ文字は '\t' だってことも説明しました。'\n'は数字に直すと12、'\t'は9です。改行とかタブはよく使われるものなので、こういったニックネームみたいな書き方が準備されているわけです。

（あ、念のため繰り返しておくと、日本語Windows上では「\」のかわりに「¥」で代用してよいですよ）

'A'は65、'B'は66、…といった感じに、通常表示できるような文字も数字に対応しています。これも多分どこかですでに述べました。

1バイトは0から255までの256種類。通常の文字にも割り当てられていないし、改行とかタブみたいにニックネームをもった文字でもない、そういうものがあります。たとえば0に対応する具体的な文字はありません。pythonはそういう情報もちゃんと表現する方法を準備してくれていて、これはたとえば'\x00'となります。'\x□□'って感じで全部で4字分の表現になりますが、これが一文字として扱われます。

255に対応する文字の書き方は、'\xff'です。

で、このテキストは、たとえば16進数も知らない人が読んでくれると想定していますから、何なら今から16進数のことを説明しなくてはいけないのでしょうが…　今は割愛しましょう。\x□□という表現が出てきたら、何か特殊な一文字なんだ、と理解してくれれば今はよいことにします。□の中には、0から9、またはabcdefのうちどれかが入ります。

なんせ、やろうとしている仕事の途中経過を対話シェルなどで確認してみたときに'\x□□'みたいな表現が出てきたときに「わあ、文字化けだあ」といって驚く必要はないですからね。printで明示的に表示させてみればよいです。

printを使ったのにあいかわらず'\x□□'という表現が出てきてしまう、ということも実はあります。リストとか辞書を直接全部printしてしまおうとする場合などですね。

```
>>> b = ['こんにちわ', '世界']
>>> print b
['¥x82¥xb1¥x82¥xf1¥x82¥xc9¥x82¥xbf¥x82¥xed', '¥x90¥xa2¥x8aE']
>>>
```

これもまた、pythonの仕様だと思ってもらうしかないです。表示されるのがたとえば「世界」とかだと、これがシフトJISでエンコードされた文字列なのか、日本語EUCでエンコードされた文字列なのか、それがアイマイになってしまうではないですか。そういう情報を落とさないように、バイト情報までを正確に表現してくれているんだなと思えば、まあ許せるんじゃないですか。許せないですか、そうですか。

こういうときは、一要素づつprintで表示すればいいです。ちょっと面倒ですけどね。

```
>>> for i in b:
...   print i
...
こんにちわ
世界
```

もうひとつの方法

次に説明するのは、「ユニコード文字列」です。今までの文字列にソックリなものなんですが、使おうとするなら違いを正確に知っておく必要があります。

なんせ、いろいろサンプルを見てみましょう。最初はsって変数にユニコード文字列を入れる例。

```
>>> s = u'こんにちわ'
>>> print s
こんにちわ
```

クォーテーションで囲むときに、こんな風に「u」をくっつけると、今から扱うのはユニコード文字列ですよという表現になります。別に何の変哲も感じられませんが...

```
>>> len(s)
5
```

お、この文字列は5文字だって認識されてるぞ。

```
>>> print s[2]
に
```

で、三文字目はちゃんと期待通りに「に」だって。こんな風に、ユニコード文字列は、人間にとっての見た目上の一文字と、python上での一文字がちゃんと対応する仕組みになってます。

日本語以外が混じってても、正確に文字数を数えてくれますよ。下の通り。

```
>>> s2 = u'毎日がeveryday'
>>> print s2[2:4]
がe
```

これはいいですね。なーんだ、最初からこいつを教えてくれればいいじゃないか、って感じですね。でも、文字列のエンコードとデコードを理解していないとこいつはちゃんと使いこなせないので、今まで説明をあとまわしにしていたのでした。例えば、ユニコード文字列について、下のような言い方で説明したときに、「あ、なるほどね」と言ってもらえるのならばありがたいのですが、いかがでしょう。

- 通常の文字列をデコードすればユニコード文字列になる。
- ユニコード文字列をエンコードすれば通常の文字列になる。
- ファイルに書き込まれている日本語はエンコードされている。
- ファイルに書き出すときもエンコードしなくてはいけない。

どうもスッキリ説明できる自信がまだないんですよ。

今のところは、ユニコード文字列を使うと、部分文字列の取り出しなどに便利なときがある、とは理解しておいてください。でも詳しい使い方の注意点は、必要になるたびに、小分けに説明していきます。歯切れ悪いよね。