目次へ

シフトJIS、文字コード、云々

ここでは、今までちゃんと説明しなかった「文字エンコーディング」についてちゃんと理解をしてみよう、という感じの話をします。こいつを理解すれば、今まで「シフトJIS」「coding:cp932」とかオマジナイのように紹介していた概念がわかるようになります。わかるといいな。

まず、今まで紹介してきた「文字列」ってのは、pythonの内部では数字のリストみたいなものとして扱われているということについて。（もっと正確にはタプルみたいなもの、というべきだけど、今はそこは問題でない）

イメージがつくりやすいように、対話シェルからord関数とchr関数というものを試してみましょう。

```
>>> ord("a")
97
>>> chr(97)
'a'
```

一文字を与えるとそれに対応する数字を教えてくれるのがordで、数字に対応する一文字を教えてくれるのがchrです。ordはorder(順序)という意味で、chrはcharacter(文字)という意味を縮めた関数名なんでしょうね。文字列どうしの大小関係を比べることができるのは、この仕組みのおかげですね。

で、次は、一文字を超える長さの文字列。文字列は数字のリストだ、というところをざっと実感するのが目的なので、知らない命令が出てきてもあまり深く考えなくてよいです。

```
>>> s = "Hello, World!"
>>> b = list(s)
>>> b
['H', 'e', 'l', 'l', 'o', ',', ' ', 'W', 'o', 'r', 'l', 'd', '!']
>>> bb = map(ord, b)
>>> bb
[72, 101, 108, 108, 111, 44, 32, 87, 111, 114, 108, 100, 33]
```

Hello, World! という文字列が、72, 101, 108… という数字の列になっちゃいました。

通常、この数字ひとつは、0から255の範囲におさまります。0から255までの数（つまり256通り）からひとつ取ったとき、こいつを「1バイト」の情報量である、と呼ぶことがあります。

だから上の例は、文字列をバイトのリストに直したともいえます。このことを、文字列を「エンコード」したんだ、と呼びましょう。英語で言えばencode。コードに直した、ということですね。コードって何だといわれてもアレですが、まあ、人間よりの情報を、よりコンピュータよりの情報に直した、とでも理解したらいいのかなあ。

（ちなみに、エンコードの反対はデコードね。この例なら、バイトのリストを文字列に直すのはデコード。）

何のためにエンコードするのかというと、端的に言えば「ファイルに格納するため」「通信に乗せるため」です。ちょっと不正確なのを覚悟の上でいうと、ハードディスクの中は全部バイト情報のカタマリとして成り立っていますし、インターネット回線の中もバイト情報の連なりが行ったり来たりしています。ディスク容量何キロバイトとか、毎秒何メガバイトの通信速度とか言うでしょ。（キロバイトは1024バイト、メガバイトは1024キロバイトのことね）

日本語をエンコード

で・・・

英語圏だけでコンピュータを使っているときは、256種類あれば普段使っている文字はぜんぶ素直にエンコードできたわけですよ。256種類どころか、それの半分の128種類でもあり余っていますね。AからZ、その小文字、数字、記号いくつか…　余裕です。

でも日本語はひらがなもカタカナも漢字も入れたいから、とても256種類には収めきれない。簡単に「バイトの集合」に変換できない。これが問題の根本です。どうやってこいつを工夫するか、というのが、「日本語エンコード」が存在する理由です。で、不幸なことに、この工夫は一種類じゃないんですよね。

どんな「工夫」が歴史的に存在したか。（とはいっても筆者はぜんぜん専門知識に自信はありません）

まず、英語圏の文字が実際は256種類どころか128種類で足りてしまっている点に目をつけて、残りの128個を日本語っぽい情報にあてはめて使おうとした時代がありました。カタカナくらいならそこに詰め込めました。そういうことで、これをもって「日本語対応したぜ」と済ましていたわけです。英語とカタカナだけのレシートとか宛名シールとか、見たことありますよね。その

時代に作ったシステムがまだ色々なところで動いてるんでしょうね。~~時代遅れだから直せばいいのにね。~~

別の「工夫」の例もあります。日本語をあらわすために、「2バイトで一文字」というルールにするんです。これなら256の二乗で、65536種類の文字をあらわすことができますからね。

でも安易にこんなことをしては、今まで英語圏で作られていたデータファイルが読み込めません。逆に、日本で英語だけのデータファイルを作ったとしても、英語圏で読み込めません。なので最初は今までどおりの「1バイトで一文字」からファイルを始めて、必要なときに「今から2バイトで一文字だぞ」という合図をあらわす特殊なデータの並び（エスケープシーケンス）を入れて、日本語用のデータを続ける。で、それが終わったら「2バイトで一文字の用事は終わったよ」という合図で元に戻す、という方式が案出されました。この方法は今でも使われる場面はあって、「7ビットJIS」とか「ISO-2022-JP」とか呼び名がついています。

この7ビットJISを使って「ABC漢字」という日本語をエンコードすると、下のようになりますよ。（打ち込む命令の意味はまだわかんなくていいです。でも、iso-2022-jpという何かでencodeしている、という気配は感じられますね）

```
>>> map(ord, u'ABC漢字'.encode('iso-2022-jp'))
[65, 66, 67, 27, 36, 66, 52, 65, 59, 122, 27, 40, 66]
```

(65, 66, 67) は、英語圏と同じでA, B, Cです。次の(27, 36, 66)が、2バイトモードが始まるよ、という目印。(52, 65)が「漢」、(59,122)が「字」、最後の(27, 40, 66)が、2バイトモードが終わったよ、という目印。

次にいよいよ、おなじみの「シフトJIS」という呼び名がついた「工夫」について。こいつは、さっきの7ビットJISみたいに、「今から2バイトモード」という目印をわざわざ使いません。「0から127の間は一文字分の英語圏テキストだけど、128以上が来たら、次の一バイトとあわせて日本語一文字分ね」という方法を使います。ちょっとした付加的ルールはありますが、ザックリした理解としてはこんな感じ。

```
>>> map(ord, u'ABC漢字'.encode('shift-jis'))
[65, 66, 67, 138, 191, 142, 154]
```

(65, 66, 67)は今までと同じ。次の138を見たときに、シフトJIS的には「128以上だ。次の191とあわせて一文字」ということになりますから、(138, 191)で「漢」一文字。つぎも(142, 154)で「字」一文字。ちょっと短めに済みますね。

シフトJISと同じ発想で、「日本語EUC」というのもあります。

```
>>> map(ord, u'ABC漢字'.encode('euc-jp'))
[65, 66, 67, 180, 193, 187, 250]
```

(65, 66, 67)までは今までと同じですが、「漢」をあらわすデータが(180, 193)、「字」をあらわすデータが(187, 250)という決まりになっています。アイデアはシフトJISも日本語EUCも同じなのに、結果として違う方式が存在するんですよ。ちなみにどっちもよく使われています。不幸でしょ。なんでこんなことになってるんだ、と言われると困りますが、話せば長いことです。得てして、こういうことって起こるものなんです。

この機会だから、「UTF-8」というエンコード方式もあることを言っておきましょうか。

```
>>> map(ord, u'ABC漢字'.encode('utf8'))
[65, 66, 67, 230, 188, 162, 229, 173, 151]
```

こいつは、日本語一文字が3バイトになる特徴があります。230が来たら、「次と、次」をあわせてはじめて一文字、とかそんなルールがあるもので。

なんせ、日本語をバイトのリストに直す方法は、是非はともかく、たくさんあるということがお分かりいただけましたでしょうか。シフトJISでエンコードしてファイルに格納したテキストファイルは、シフトJISでデコードすれば正しく復元できます。日本語EUCでエンコードしたものは、日本語EUCで正しくデコードできます。でも、たとえば日本語EUCでエンコードしたものをシフトJISでデコードしたら？　すごく無意味な文字がズラズラ出てくること請け合いですね。こいつがいわゆる「文字バケ」というやつですよ。文字バケの例なんて示すまでもないですよね。いろんなところで散々見てるでしょうから。

coding:cp932

pythonで日本語を含むスクリプトを作るとき、必ず一行目に下のおまじないを書きましょう、といいました。

```
# coding: cp932
```

スクリプトは通常ファイルに格納されているんですから、このスクリプト自身もバイトのリストにエンコードされているというわけですね。だから自分がどんなエンコードされているのかをここでpythonに知らせてあげているのです。

cp932ってのは、ほとんどシフトJISと同じようなものです。（厳密には、シフトJIS系列の中にも細かい違いを持った「流派」みたいなのがあって、またこれが悩ましいんだけど…）

下のように、

```
# coding: shift-jis
```

でもほとんどの場合は大丈夫ですし、サンプルスクリプトをこうやって紹介する人もいるでしょう。

まあ、とにかく、この行は、pythonに「このスクリプトはこういうエンコードですよ」と知らせるための特別なルールです。これを書きそびれると、pythonは今から読み込むスクリプトを「英数字オンリー」のものとみなしますから、あとでその仮定にそぐわないデータが見つかるとエラーを報告して実行してくれないのです。

もしスクリプトをEUC-JPエンコードで格納して実行する場合（Linuxとかを使うと、こっちを使う機会のほうが多いかな）は、この行はこう変えるべきですし、

```
# coding: euc-jp
```

スクリプトをUTF-8エンコードで格納するときは

```
# coding: utf-8
```

です。

ついでに：HTMLでも自分自身のエンコードを宣言する場所があります

HTMLも、色々なエンコードで日本語とか他の言語を表してよいテキストファイルです。で、自分自身がどういうエンコードをしているか、下の部分が宣言しています。

```
<meta http-equiv="Content-Type" content="text/html; charset=shift_jis" />
```

metaタグの使い道のひとつですね。ここでは、shift_jis、つまりシフトJISで書かれてます、と宣言しています。もちろん日本語EUCで作ったHTMLは、ここをeuc-jpを書いておくべきです。

文字セット

文字セット（文字集合）というものは、エンコーディングとは違うということをここで明らかにしておきましょう。

今までに見てきたエンコードのルールは、文字をどうやってバイトのリストに直すか、というものでした。

文字セットってのは、「ぜんぶでどれほどの範囲の文字を使うか」ということです。

英語圏なら、AからZ, aからz、0から9、あと記号をいくつかと、改行とかタブとかの特殊文字…　というのが「使いたい全部の文字」でした。これが「ASCII（アスキー）文字セット」。

で、ASCII文字セットを含んで、あと日本語特有のひらがな、カタカナ、一般に使われる漢字、あと記号類（罫線、ギリシャ文字、キリル文字等）色々… というのを全部挙げたのが「日本語文字セット」。

で、それらを全部含みながら、まれにしか使わない珍しい漢字や各国の文字をめいっぱい（ハングル、アラビア文字、タイの文字等）集めてまとめたのが「多言語文字セット」。

シフトJISエンコードとか日本語EUCエンコードは、これらのうち「日本語文字セット」が表現できます。この文字セットの名前は「JISなんとか」って感じの規格に正確に定義されていますが、あんまり筆者は詳しくないです。微妙に違うバリエーションがいくつも存在します。

UTF-8エンコードは、ユニコードという名前の多言語文字セットが扱えます。ユニコードにもいくつかバージョンがあるらしいんだけど、筆者はやっぱりそんなに詳しくないです。

ってことで、使いたい文字セットに応じて、使うべきエンコードのルールも選ぶべきというわけですね。

この考え方を呑み込んでおけば、ある日、こんなスレたセリフが吐けるようになるでしょう。

「このテキストファイル、メモ帳で読み込んだらバケてらあ。この独特のバケかたから見るに、たぶんEUCだったんだね。○○エディタならEUCが扱えるから、そっちで開きなおそう。メモ帳はシフトJISしか使えないんだよ、まったく。　...あっと、この文字だけは、打ち込んでもまともに保存できないようだ。これだけ「?」なんて記号に置き換えられちゃう。そうか、この文字はJIS(日本語)の文字セットに入ってなかったのか。ユニコードが扱えるエディタを探してこなくちゃいけないな」

参考までに、このページは多言語対応のユニコードで管理されています。 안녕하세요. **สวัสดี**. اَلسَّلامُ عَلَيْكُمْ. ほらね。(ブラウザによっては、ちゃんと見えないかも)

まだ説明すべきことは色々あるなあ

pythonで、日本語を含む文字列の「何番目の文字」というのをちゃんと取得するのは、ちょっとコツがいるのです。この方法とその理由を説明するのは面倒そうだなー。

具体的には、

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

となってしまって、たとえば「に」だけを抜き出すのは直観的な方法ではうまいくいきません。これをどうすればいいんだい、という話。

どう説明しようかなー。

いつか必要が生じたときにちゃんと説明したいと思いますが、ここではあえて中途半端なナゾにして残しておこうかな、とも思います。

ヒントとしては、直前の例で a という変数に入ったのは、「エンコードされたあとのもの」なのか、「エンコードされる前の何か」なのか、という点ですが...

ここまでまずはOKという方は、続きとして、こっちへどうぞ。