Python では、OS 上で扱われている具体的な文字コードの文字列を、Python 内部形式のユニコード文字に変換することを、「エンコード」という。
エンコードされた文字を、具体的な文字コードの文字列に変換することを、「デコード」という。

具体的な文字コードとは、ASCII とか、Shift JIS とか EUC とか UTF-8 とかのこと。

以下、Python 3 ではまったく変わってしまったので書き直す。
Python 3 の文字列はすべて、エンコードされた Unicode 文字列として、Python インタープリターが保持している。

なので、そもそもエンコード処理が成功していなければ、そのデータを文字列として Python プログラムが保持することさえ、できなくなっている。
エンコードするには、Python インタープリターが、そのデータの文字コードを正確に知っていなければならない、ということになる。

そうだとすると、文字列リテラルとか、実行中に渡される引数とか標準入出力とかにおいて、正しい文字列がわかっていないとエラーになってしまう、ということだ。

Python 3 は、デフォルトでソースコード中の文字列リテラルは UTF-8 であるとみなす。そうでないと Syntax Error を吐く。
恐ろしいことだ。

```
# encoding: utf-8
```

じゃあ標準入出力はどうしているか？
お そ ら く だ が、 sys.stdin.encoding、 sys.stdout.encoding、 sys.getfilesystemencoding、 sys.getdefaultencoding などを使っていると思われる。
その前提が違っている文字が渡されると、UnicoeDecodeError が吐かれるはずだ。

```
# リテラル文字列からバイト型データを作る。
# encoding を指定しないと、デフォルトの encoding が使われる、とのこと。
# Python のユニコード文字列は、Python が扱える任意の型のバイト型データに変換可能。
b1 = bytes("ほげほげ", encoding='cp932')
b2 = "ほげほげ".encode('utf-8')

# バイト型データは、特定のエンコード型を持つことができる。すなわちそれは、Python 2 までの文字列。またはいわゆる普通のコンピューターのエンコードされた文字列。
# Python で文字列データを扱うには、エンコードされたバイト型データを Python 3 文字列（Unicode 型）にしないとならない。これがデコード処理。
# デコード処理を行うためには、元のバイト型データがどんな encoding なのか正しく指定しないと、UnicodeDecodeError が発生する。
d1 = b1.decode('cp932')
d2 = str(b2, encoding='utf-8')
```

----- 旧 -----
ユニコード文字列にデコードするメリットとしては、Python がマルチバイト文字を扱うことができるようになる、ということ。
プログラマが文字コードを意識した処理を書くことで、意図しない文字コードの欠損や変換（い

わゆる文字化け）が生じないプログラムを書く、といったことになる。

Python ユニコード文字列にデコードを行うとき、プログラマは元の文字列の文字コードが何であるのか、正しく知っていなければならない。
しかしこれは、しばしば問題を伴う。
日本語ではメジャーな文字コードが Shift JIS、EUC、UTF-8 と複数あるため、ユーザからの入力文字列を受け取るプログラムは、容易に正しい文字コードを特定できない状況で使用される場合があるということ。

Perl の Encode.pm のような標準モジュールを持たない Python では、根本的な解決策はない。

コンソール上でのみ使用されるプログラムの場合、入力文字の文字コードを推測する方法として、

```
import sys
decoded_string1 = unicode(input_string1, sys.getfilesystemencoding())
decoded_string2 = unicode(input_string2, sys.stdin.encoding)
```

というのがある。後者はコンソールから入力された文字の文字コードとしてはかなり信憑性がある。
しかし OS のファイルシステム上に存在しているファイルの中身が、すべて sys.getfilesystemencoding() で解決できるかと言うと、そうは言えない場合もある。

デコード時の文字コード指定が間違っていると、Python ではほぼ 100% 例外が発生する。
効率は悪いが、デフォルトの文字コードでデコードできない場合は、try: except: で可能性のある文字コードを全部試して行く、という方法もある。

一番困るのは CGI のようなもので、Perl だと Encode.pm で文字コードを guess できるが、Python には標準ではそのようなものはない（というか、標準でマルチバイト文字コードの guess ができるなんて、Perl くらいのもんだ）。

自分で実装する、どっかから拾ってくる、nkf のような OS 上の仕組みを使う、といった苦しい逃げを打たなければならない。

Python ユニコード文字列（デコードされた文字列）の標準出力への出力は単純で、print 文は sys.stdout.encoding にエンコードして出力を行う。
sys.stdout.encoding 以外の文字コードで標準出力に書き出したら、文字化けするので、これは当然の動作と言えば当然。

ファイルに書き出す場合は、プログラマが好きな文字コードにエンコードすればよい。