24.13. Unicode for non-English characters

Sometimes, you may need to deal with text that includes characters that are not part of the standard English alphabet, such as é, ö, Φ, or ¥. This is especially likely if you use REST APIs to fetch user-contributed content from social media sites like Twitter, Facebook, or flickr.

Python's strings are in **unicode**, which allows for characters to be from a much larger alphabet, including more than 75,000 ideographic characters used in Chinese, Japanese, and Korean alphabets. Everything works fine inside Python, for operations like slicing and appending and concatenating strings and using .find() or the `in` operator.

Things only get tricky when you want to input strings into Python, or print them to an output window or write them to a file.

For output, your terminal (output) window will typically be set up to display characters only from a restricted set of languages (perhaps just English). If you issue a print statement on a string containing other characters, it may not display correctly in your terminal window. Indeed, you may get an error message. We will offer a workaround later on this page.

If you want to store unicode text in a file, you have to choose an "encoding". This is analogous to the encoding of special characters in a URL string, but not the same. In a file, each unicode character has to be encoded as one or more "bytes" for storage in a file. We have avoided low-level details about data encodings until now, but understanding a little about bits and bytes will help make sense of this.

A **bit** is a BInary digiT. It is a single value restricted to two (binary) possibilities, which we conventionally write as 0 or 1. Computers store bits as electrical charges (high or low voltage) or as magnetic polarities, or some other way that we need not be concerned about. A sequence of eight 0-1 bits is called a byte. For example: 01001000.

There are 2^^8=256 distinct eight-bit bytes. If we only had 256 possible letters in our alphabet, we could simply encode each letter as one of the available bytes. When we restrict ourselves to regular python strings, using only the ASCII alphabet (English, plus a few special characters), the encoding is that simple, so simple that we haven't had to think about it before.

When there are 75,000 possible characters, they can't all be encoded with a single byte, because there are only 256 distinct bytes (eight-bit sequences). There are many possible encodings. The one you will be most likely to encounter, using REST APIs, is called UTF-8. A single unicode character is mapped to a sequence of up to four bytes.

If you read in a UTF-8 encoded text, and get the contents using `.read()` or `.readlines()`, you will need to "decode" the contents in order to turn it into a proper unicode string that you can read and use.

Fortunately, the requests module will normally handle this for us automatically. When we fetch a webpage that is in json format, the webpage will have a header called 'content-type' that will say something like `application/json; charset=utf8`. If it specifies the utf8 character set in that way, the requests module will automatically decode the contents into unicode: `requests.get('that web page').text` will yield a string, with each of those sequences of one to four bytes coverted into a single character.

(flickr.html) (chapterProject.html)

If, for some reason, you get json-formatted text that is utf-encoded but the requests module hasn't magically decoded it for you, the `json.loads()` function call can take care of the decoding for you. `loads()` takes an optional parameter, encoding. Its default value is 'utf-8', so you don't need to specify it unless you think the text you have received was in some other encoding than 'utf-8'.

Note

You may see a `u` before the string in a Python 2.7 program indicating that it's a unicode string. In Python 3, *all* strings are unicode strings, so you shouldn't encounter any of those strange u characters in this textbook.

Once you have python strings, everything will work fine until you try to print or write the contents to a file. If you print, and your terminal window is not set up to display that language, you may get a strange output, or an error message.

If you try to write to a file with unicode strings, you may get an error. When you write a unicode string to a file, Python tries to encode it in ASCII. If there is a non-ASCII character, the execution fails and raises an error that looks like this: `UnicodeEncodeError: 'ascii' codec can't encode character u'\xea' in position 1: ordinal not in range(128)`.

One solution is to use the Python method to encode the string, using a format such as utf-8. For example, `s.encode('utf-8')` will encode string `s` as `utf-8`. That will encode non-ASCII characters with multiple character sequences that are difficult for people to read but can decoded back into single Unicode characters. This is often the best way.

Another quick-and-dirty option, if you just have a few stray characters that are getting in your way, is to replace any non-ASCII characters with question marks. For example, `s.encode('ascii', 'replace')`. Of course, replacing characters with question marks destroys some of the information, but it may be helpful in some circumstances.

Note

In the Runestone online environment, the `.encode()` and `.decode()` methods are not available for strings. You can only use them in a full python environment. In all of the cached data that we provide from REST APIs in this online book, we have tried to avoid Unicode encoding issues.

Check your understanding

When you have non-ASCII characters in a string and you get an error trying to write them, which method should you invoke on the string?

- (●) A. ``.encode()``
- ( ) B. ``.decode()``
- ( ) C. ``.json()``

Check Me Compare me

(flickr.html) (chapterProject.html)

✔ It's a little counter-intuitive, since it seems like you already have an encoding of the information, but you have to re-encode it by substituting sequences of ASCII characters for some of the Unicode characters.

Activity: 24.13.1 Multiple Choice (24_unicode_1)

In the textbook environment, what should you do if you are unable to write data to a file because of a Unicode encoding error?

- A. Call ``.encode()``
- B. Call ``.decode()``
- C. Call ``.json()``
- D. Get different data, because we don't have ``.encode()`` available in the Runestone environment.

Check Me Compare me

✔ Unfortunately, this is what you have to do.

Activity: 24.13.2 Multiple Choice (24_unicode_2)

You have attempted 3 of 3 activities on this page

✔ **Completed. Well Done!**

 | Back to top

(flickr.html) (chapterProject.html)