Encoding / Decoding

Back in the good ol' days, you quickly learned that you cannot decode a unicode string and you cannot encode a str type either. If you tried to take a unicode string and decode it to ascii (i.e. convert it to a byte string), you would get a UnicodeEncodeError. For example:

```
u"\xa0".decode("ascii")

#Traceback (most recent call last):
#   File "/usercode/__ed_file.py", line 1, in <module>
#    u"\xa0".decode("ascii")
#UnicodeEncodeError: 'ascii' codec can't encode character u'\xa0' in position 0: ordinal not
```

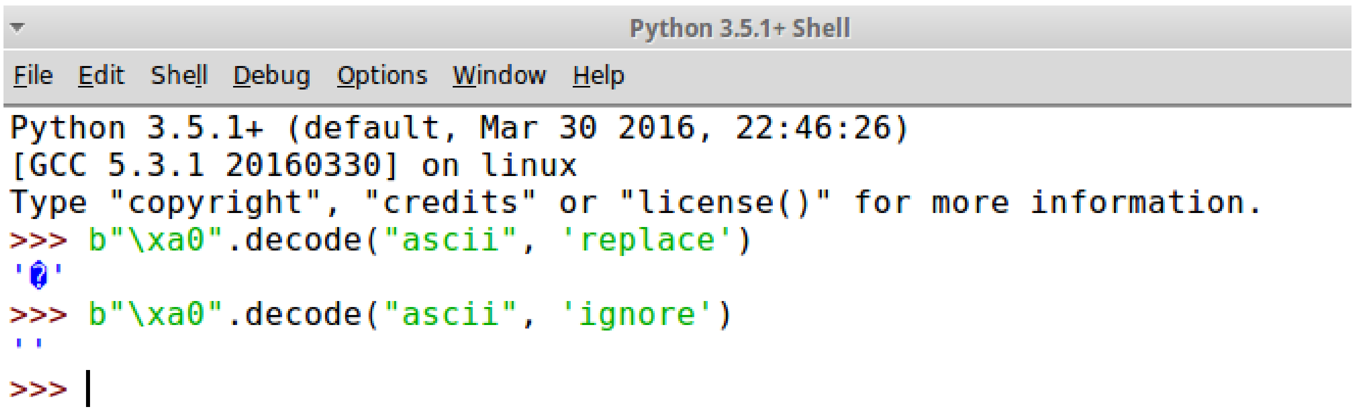
If you tried something like this in Python 3, you would get an **AttributeError**

```
u"\xa0".decode("ascii")
#Traceback (most recent call last):
#  File "/usercode/__ed_file.py", line 1, in <module>
# u"\xa0".decode("ascii")
#AttributeError: 'str' object has no attribute 'decode'
```

The reason is pretty obvious in that strings in Python 3 don't have the decode attribute. However, byte strings do! Let's try a byte string instead:

```
b"\xa0".decode("ascii")
#Traceback (most recent call last):
#  File "/usercode/__ed_file.py", line 1, in <module>
# b"\xa0".decode("ascii")
#UnicodeDecodeError: 'ascii' codec can't decode byte 0xa0 in position
```

That still didn’t quite work the way we wanted as the ASCII codec didn’t know how to handle the character we passed to it. Fortunately you can pass extra parameters to the decode method and make this work:



```
Python 3.5.1+ (default, Mar 30 2016, 22:46:26)
[GCC 5.3.1 20160330] on linux
Type "copyright", "credits" or "license()" for more information.
>>> b"\xa0".decode("ascii", 'replace')
'�'
>>> b"\xa0".decode("ascii", 'ignore')
''
>>> 
```

This example shows what happens if you tell the decode method to replace the character or to just ignore it. You can see the results for yourself.

Now let’s look at an example from the Python documentation to learn how to encode a string.

```
u = chr(40960) + 'abcd' + chr(1972)
print (u.encode('utf-8'))
#b'\xea\x80\x80abcd\xde\xb4'

print (u.encode('ascii'))
#Traceback (most recent call last):
#  File "/usercode/__ed_file.py", line 5, in <module>
# print (u.encode('ascii'))
#UnicodeEncodeError: 'ascii' codec can't encode character '\ua000' in position 0: ordinal not
```

```
u = chr(40960) + 'abcd' + chr(1972)

print (u.encode('ascii', 'ignore'))
#b'abcd'

print (u.encode('ascii', 'replace'))
#b'?abcd?'
```

For this example, we took a string and added a non-ASCII character to the

For this example, we took a string and added a non-ASCII character to the beginning and the end of the string. Then we tried to convert the string to a

bytes representation of the Unicode string using the encode method. The first attempt did not work and gave us an error. The next one used the **ignore** flag, which basically removed the non-ASCII characters from the string entirely. The last example uses the **replace** flag which just puts question marks in place for the unknown Unicode characters.

If you need to work with encodings a lot, Python also has a module called **codecs** that you should check out.

Wrapping Up

At this point you should know enough about Unicode to actually use it. Unicode makes it possible for your application to support other languages both in the code itself and in its output. You have also just scratched the surface on how to encode and decode strings in Python. The documentation on this subject is pretty extensive, so if you need more information, please check it out.