Image Understanding and Pattern Recognition
Prof. Dr. Thomas M. Breuel

18.06.2009

Dr. Faisal Shafait

Document and Content Analysis (SS 2009)

Exercise Sheet 3

to be submitted by E-Mail to faisal.shafait@dfki.de by: 29.06.2009

Use Python 2.x for the following exercises. Python 2.x has two kinds of strings:

- regular strings (byte sequences), written as in `"abc"`
- unicode strings, written as in `u"abc"`

Furthermore, the default encoding for strings in Pyton 2.x is set to ”ascii”, which means that you will get an error if you try to print anything that's not an ASCII character:

```
>>> print u"\u1000"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  UnicodeEncodeError: 'ascii' codec can't encode character u'\u1000'
  in position 0: ordinal not in range(128)
```

If you want to see interesting Unicode characters, you need to explicitly encode your strings in your terminal's encoding. The default terminal emulator under Gnome uses a ”utf-8” encoding. If you just encode, the weird characters show up escaped:

```
>>> u"\u1000".encode('utf-8')
'\xe1\x80\x80'
>>>
```

If you actually want this 8-bit string sent to the console literally, you need to print them explicitly:

```
>>> print u"\u1000".encode("utf-8")
...
>>>
```

However, you don't actually need to display any of these strings in order to do the exercises (although you may find it useful).

Many Unicode fonts do not have glyphs for all Unicode characters; one font that tries for complete coverage of the Basic Multilingual Plane is the “unifont” font. On Debian and Ubuntu, you can `apt-get install ttf-unifont` (for other platforms, search for Unifont on Google and follow the instructions). To make a Gnome terminal that uses Unifont, create a new profile and select “unifont” as the font.

You can put extended Unicode characters into strings using quoting:

- `u"\u1234"` yields a character that is representable as a 16 bit codepoint. This always takes four hex digits.
- `u"\U00010000` yields a character from an "astral plane". This always takes 8 hex digits.

Exercise 3.1 (Python Unicode)
Write a pair of functions that convert a Python unicode string to a list of integer codepoints and back.

Exercise 3.2 (UTF-16)
Explain:

```
>>> s = u"\ud800\udc00"
>>> len(s)
2
>>> len(s.encode("utf-16").decode("utf-16"))
1
>>>
```

What happens if you use utf-8 instead of utf-16?
Explain:

```
>>> len("x".encode("utf-16"))
4
>>>
```

Exercise 3.3 (UTF-8)
Write a pair of functions, `utf8_to_codepoints` and `codepoints_to_utf8` for decoding and encoding UTF-8.

Exercise 3.4 (Unicode Leet Speak)
Write a function `unileet(s)` that takes any string of English letter, digits, spaces, hyphens, periods, and commas, and returns a readable Unicode string that uses no ASCII characters whatsoever. The closer your resulting string looks to the original, the better.