```
0 00e9 Ll LATIN SMALL LETTER E WITH ACUTE
1 0bf2 No TAMIL NUMBER ONE THOUSAND
2 0f84 Mn TIBETAN MARK HALANTA
3 1770 Lo TAGBANWA LETTER SA
4 33af So SQUARE RAD OVER S SQUARED
1000.0
```

The category codes are abbreviations describing the nature of the character. These are grouped into categories such as "Letter", "Number", "Punctuation", or "Symbol", which in turn are broken up into subcategories. To take the codes from the above output, `'Ll'` means 'Letter, lowercase', `'No'` means "Number, other", `'Mn'` is "Mark, nonspacing", and `'So'` is "Symbol, other". See the General Category Values section of the Unicode Character Database documentation for a list of category codes.

2.5 Comparing Strings

Unicode adds some complication to comparing strings, because the same set of characters can be represented by different sequences of code points. For example, a letter like 'ê' can be represented as a single code point U+00EA, or as U+0065 U+0302, which is the code point for 'e' followed by a code point for 'COMBINING CIRCUMFLEX ACCENT'. These will produce the same output when printed, but one is a string of length 1 and the other is of length 2.

One tool for a case-insensitive comparison is the `casefold()` string method that converts a string to a case-insensitive form following an algorithm described by the Unicode Standard. This algorithm has special handling for characters such as the German letter 'ß' (code point U+00DF), which becomes the pair of lowercase letters 'ss'.

```
>>> street = 'Gürzenichstraße'
>>> street.casefold()
'gürzenichstrasse'
```

A second tool is the `unicodedata` module's `normalize()` function that converts strings to one of several normal forms, where letters followed by a combining character are replaced with single characters. `normalize()` can be used to perform string comparisons that won't falsely report inequality if two strings use combining characters differently:

```
import unicodedata

def compare_strs(s1, s2):
    def NFD(s):
        return unicodedata.normalize('NFD', s)

    return NFD(s1) == NFD(s2)

single_char = 'ê'
multiple_chars = '\N{LATIN SMALL LETTER E}\N{COMBINING CIRCUMFLEX ACCENT}'
print('length of first string=', len(single_char))
print('length of second string=', len(multiple_chars))
print(compare_strs(single_char, multiple_chars))
```

When run, this outputs:

```
$ python3 compare-strs.py
length of first string= 1
length of second string= 2
True
```

The first argument to the `normalize()` function is a string giving the desired normalization form, which can be one of 'NFC', 'NFKC', 'NFD', and 'NFKD'.

The Unicode Standard also specifies how to do caseless comparisons: