and *a* to force representational form but only using ASCII characters. Here is an example:

```
>>> "{0}  {0!s}  {0!r}  {0!a}".format(decimal.Decimal("93.4"))
"93.4  93.4  Decimal('93.4')  Decimal('93.4')"
```

In this case, `decimal.Decimal`'s string form produces the same string as the string it provides for `str.format()` which is what commonly happens. Also, in this particular example, there is no difference between the representational and ASCII representational forms since both use only ASCII characters.

Here is another example, this time concerning a string that contains the title of a movie, "翻訳で失われる", held in the variable `movie`. If we print the string using `"{0}".format(movie)` the string will be output unchanged, but if we want to avoid non-ASCII characters we can use either `ascii(movie)` or `"{0!a}".format(movie)`, both of which will produce the string `'\u7ffb\u8a33\u3067\u5931\u308f\u308c\u308b'`.

So far we have seen how to put the values of variables into a format string, and how to force string or representational forms to be used. Now we are ready to consider the formatting of the values themselves.

Format Specifications

The default formatting of integers, floating-point numbers, and strings is often perfectly satisfactory. But if we want to exercise fine control, we can easily do so using format specifications. We will deal separately with formatting strings, integers, and floating-point numbers, to make learning the details easier. The the general syntax that covers all of them is shown in Figure 2.6.

For strings, the things that we can control are the fill character, the alignment within the field, and the minimum and maximum field widths.

A string format specification is introduced with a colon (`:`) and this is followed by an optional pair of characters—a fill character (which may not be `}`) and an alignment character (`<` for left align, `^` for center, `>` for right align). Then comes an optional minimum width integer, and if we want to specify a maximum width, this comes last as a period followed by an integer.

Note that if we specify a fill character we must also specify an alignment. We omit the sign and type parts of the format specification because they have no effect on strings. It is harmless (but pointless) to have a colon without any of the optional elements.

Let's see some examples:

```
>>> s = "The sword of truth"
>>> "{0}".format(s)      # default formatting
'The sword of truth'
```