task to tackle this early in the book. Instead, we'll just take a brief look at some of their more important capabilities.

In Chapter 2, we saw that a conversion specification can include formatting information. In particular, we used `%.1f` to display a `float` value with one digit after the decimal point. More generally, a conversion specification can have the form `%`*m*`.`*pX* or `%-`*m*`.`*pX*, where *m* and *p* are integer constants and *X* is a letter. Both *m* and *p* are optional; if *p* is omitted, the period that separates *m* and *p* is also dropped. In the conversion specification `%10.2f`, *m* is `10`, *p* is `2`, and *X* is `f`. In the specification `%10f`, *m* is `10` and *p* (along with the period) is missing, but in the specification `%.2f`, *p* is `2` and *m* is missing.

The ***minimum field width***, *m*, specifies the minimum number of characters to print. If the value to be printed requires fewer than *m* characters, the value is right-justified within the field. (In other words, extra spaces precede the value.) For example, the specification `%4d` would display the number 123 as `•123`. (In this chapter, I'll use • to represent the space character.) If the value to be printed requires more than *m* characters, the field width automatically expands to the necessary size. Thus, the specification `%4d` would display the number 12345 as `12345`—no digits are lost. Putting a minus sign in front of *m* causes left justification; the specification `%-4d` would display 123 as `123•`.

The meaning of the ***precision***, *p*, isn't as easily described, since it depends on the choice of *X*, the ***conversion specifier***. *X* indicates which conversion should be applied to the value before it's printed. The most common conversion specifiers for numbers are:

Q&A

- `d` — Displays an integer in decimal (base 10) form. *p* indicates the minimum number of digits to display (extra zeros are added to the beginning of the number if necessary); if *p* is omitted, it is assumed to have the value 1. (In other words, `%d` is the same as `%.1d`.)
- `e` — Displays a floating-point number in exponential format (scientific notation). *p* indicates how many digits should appear after the decimal point (the default is 6). If *p* is `0`, the decimal point is not displayed.
- `f` — Displays a floating-point number in "fixed decimal" format, without an exponent. *p* has the same meaning as for the `e` specifier.
- `g` — Displays a floating-point number in either exponential format or fixed decimal format, depending on the number's size. *p* indicates the maximum number of significant digits (*not* digits after the decimal point) to be displayed. Unlike the `f` conversion, the `g` conversion won't show trailing zeros. Furthermore, if the value to be printed has no digits after the decimal point, `g` doesn't display the decimal point.

The `g` specifier is especially useful for displaying numbers whose size can't be predicted when the program is written or that tend to vary widely in size. When used to print a moderately large or moderately small number, the `g` specifier uses fixed decimal format. But when used to print a very large or very small number, the `g` specifier switches to exponential format so that the number will require fewer characters.