Chapter 6

Prettier Printing

We have been printing out information using the built-in `print` function. Sometimes, however, we have had to concatenate many little strings with + to insert into sentences the values we want to print, or use inconvenient extra parameters like `end=''` to prevent default behaviour giving an undesirable result. In this chapter, we will review the `print` function, and then explore a better method of printing with Python.

Recalling the print function

The `print` function takes a value. If the value is not a string, it converts it to a string with `str`. Then, it prints it to the screen and moves one line down by printing a newline character:

```
Python
>>> print('entrance')
entrance
>>> print(1)
1
>>> print([1, 2, 3])
[1, 2, 3]
```

We have sometimes suppressed the newline by using an `end` argument:

```
Python
>>> print('entrance', end='')
entrance>>>
```

Printing with separators

We can supply more or fewer arguments to the `print` function:

```
Python
>>> print()

>>> print('one', 'two', 'three')
one two three
```

We see that `print` with no arguments just prints a newline. Supplying multiple arguments will print them all out, separated by spaces. We can change the separator:

```
Python
>>> print('one', 'two', 'three', sep='-')
one-two-three
```

Easier printing with format strings

The `print` function is useful, but becomes rather clumsy when we are doing more complicated formatting. Python provides more advanced printing through what are called *format strings*. Here is a function to print the minimum and maximum items in a list of numbers as we might write it traditionally:

```
def print_stats(l):
    print(str(min(l)) + ' up to ' + str(max(l)))
```

(We wrote our own minimum and maximum functions earlier, but they are in fact built in to Python). Here it is in use:

```
Python
>>> print_stats([2, 3, 5, 7, 11, 13, 17, 19, 23, 29])
2 up to 29
```

Now, the same function using a format string:

```
def print_stats(l):
    minimum = min(l)
    maximum = max(l)
    print(f'{minimum} up to {maximum}')
```

There are two things to notice. First, the use of `f'` to begin a string instead of just `'`. This denotes a format string. Second, the sections inside the format string which are demarcated with curly braces `{...}`. The variable names in these will be substituted for the values of those variables. In fact, we can put whole expressions in the curly braces, simplifying further:

```
def print_stats(l):
    print(f'{min(l)} up to {max(l)}')
```

Even in this simple example, we can see that it is rather easier to read our program when written with format strings, when compared with the repeated concatenation in the original. Consider a function to print out a table of powers (the ** operator raises a number to a power):

```
def print_powers(n):
    for x in range(1, n):
        print(f'{x} {x ** 2} {x ** 3} {x ** 4} {x ** 5}')
```

Much like our times table in chapter 3, the columns are not lined up:

```
Python
>>> print_powers()
1 1 1 1 1
2 4 8 16 32
3 9 27 81 243
4 16 64 256 1024
5 25 125 625 3125
6 36 216 1296 7776
7 49 343 2401 16807
8 64 512 4096 32768
9 81 729 6561 59049
```

Format strings can do this for us automatically, with the addition of a *format specifier* within the curly braces. We add `:5d` at the end of each one. The `5` is for the column width, and `d` for decimal integer – the number will be right-justified in the column.

```
def print_powers(n):
    for x in range(1, n):
        print(f'{x:5d} {x ** 2:5d} {x ** 3:5d} {x ** 4:5d} {x ** 5:5d}')
```

Here is the result:

```
Python
>>> print_powers()
1     1     1     1     1
2     4     8    16    32
3     9    27    81   243
4    16    64   256  1024
5    25   125   625  3125
6    36   216  1296  7776
7    49   343  2401 16807
8    64   512  4096 32768
9    81   729  6561 59049
```

Printing to a file

Instead of printing to the screen, we can print to a file by adding a `file` argument to the `print` function:

```
def print_powers(n):
    f = open('powers.txt', 'w')
    for x in range(1, n):
        print(f'{x:5d} {x ** 2:5d} {x ** 3:5d} {x ** 4:5d} {x ** 5:5d}', file=f)
    f.close()
```

The function open here opens the new file `'powers.txt'` for writing (hence `'w'`). We then supply the `file` argument to the `print` function. Afterward, we must be sure to close the file using the `close` method on the file `f`. A cleaner method is to use the **`with`** ... **`as`** structure:

```
def print_powers(n):
    with open('powers.txt', 'w') as f:
        for x in range(1, n):
            print(f'{x:5d} {x ** 2:5d} {x ** 3:5d} {x ** 4:5d} {x ** 5:5d}',
                  file=f)
```

The file will be closed automatically once the part of the program indented further to the right than the **`with`** is complete, so there is no need for us to close it explicitly. In the questions, we will use format strings to create some files of our own.

Common problems

We must remember to use the `f` prefix to our strings when using format strings, or we get the wrong result:

```
Python
>>> p = 15
>>> q = 12
>>> print('Total is {p + q}')
Total is {p + q}
```

Here is what it should look like:

```
Python
>>> print(f'Total is {p + q}')
Total is 27
```

Quotation marks can end a format string, even when they are with the `{}` braces:

```
Python
>>> def two(x): return x + x
...
>>> print(f'Twice is {two('twice')}')
  File "<stdin>", line 1
    print(f'Twice is {two('twice')}')
                           ^
SyntaxError: invalid syntax
```

The solution is to use double quotation marks instead:

```
Python
>>> print(f"Twice is {two('twice')}")
Twice is twicetwice
```

Comments cannot appear inside braces:

```
Python
>>> print(f'This is the result: {result #update later}')
  File "<stdin>", line 1
SyntaxError: f-string expression part cannot include '#'
```

Finally, when opening a new file for output with the **`with`** ... **`as`** ... construct, remember that we must specify `'w'`.

```
Python
>>> open('output.txt')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'output.txt'
>>> open('output.txt', 'w')
```

Summary

We have expanded our knowledge of the built-in `print` function beyond the simple uses we encountered before. We learned about the powerful notion of format strings, and how to use them to shorten and simplify our code. Finally we printed to files, so our programs can now have effects persist even when we close Python.

Questions

1. We can print a list like `[1, 2, 3]` easily using the `print` function. Imagine, though, that the print function could not work on lists. Write your own `print_list` function which uses simple `print` calls to print the individual elements, but adds the square brackets, commas and spaces itself. Do so without using format strings.
2. Now rewrite your function using format strings. Which is easier to read and write?
3. The method `rjust` on strings will right-justify them to the given width.

   ```
   Python
   >>> '2'.rjust(5)
   '    2'
   ```

   Use this method to rewrite our `print_powers` function without format strings, but still with properly lined-up columns.
4. The method `zfill` on a string, given a number, will pad the string with zeroes to that width. For example, `'435'.zfill(8)` will produce `00000435`. Modify your previous answer to use this function to print our table of powers with uniform column widths padded by zeroes.
5. Write a program which asks the user to type in a list of names, one per line, like `Mr James Smith`, and writes them to a given file, again one per line, in the form `Smith, John, Mr`.
6. Rewrite the function from the previous question using format strings, if you did not use them the first time.
7. Use the `find` function introduced the previous chapter to write a program which prints the positions at which a given word is found in each of given list of sentences. For example, consider this list:

   ```
   ['Three pounds of self-raising flour',
    'Two pounds of plain flour',
    'Six ounces of butter']
   ```

   Your function, given this list and the string `'pound'`, should print:

   ```
   pound found at position 6 in sentence 1
   pound found at position 4 in sentence 2
   pound not found in sentence 3
   ```

8. Modify your answer to question 7 to print the information to a file with a given name.