CSP1150/CSP5110: Programming Principles
Reading 2.2: Formatting Numbers

Python has a built-in function named `format()`, which allows you to format numbers so that they appear how you want. For example, a floating point number may be shown with more decimal places than you would like:

```
number = 99999 / 7
print(number)
14285.57142857143
```
Python

To format a number you pass it to the `format()` function, along with a "format specifier" – a string that uses special characters to specify how to format the number. The Python website has documentation for both the `format()` function and the format specification mini-language.

```
number = 99999 / 7
print( format(number, ',.2f') )
14,285.57
```
Python

The format specification of "`,.2f`" in the example above has resulted in the number being formatted to have commas separating the thousands and has been rounded to two decimal places.

Here are some other examples showing the useful ways in which numbers can be formatted. These examples only show the format function, and the number is entered directly into it.

```
format(0.25, '.0%')
25%
```
Python

"Show the number as a percentage, 0 decimal places"

```
format(201, '06d')
000201
```
Python

"Show the number padded with 0s until it is 6 digits long"

```
format(201, 'b')
11001001
```
Python

"Show the number in binary"

```
format(0.0009863, '.3e')
9.863e-04
```
Python

"Show the number in scientific notation, 3 decimal places"

Some of the formatting options make it easier to display numbers in nicely aligned columns. The example below uses loops to display a list of numbers (loops and lists are covered in Module 3):

```
numbers = [21, 18.548, -34.7, 6, -198.376]

print('Unpadded: ')
for number in numbers:
    print( format(number, '8.2f') )

print('Padded: ')
for number in numbers:
    print( format(number, ' 08.2f') )
```
Python

```
Unpadded:
   21.00
   18.55
  -34.70
    6.00
 -198.38
```

```
Padded:
 0021.00
 0018.55
-0034.70
 0006.00
-0198.38
```

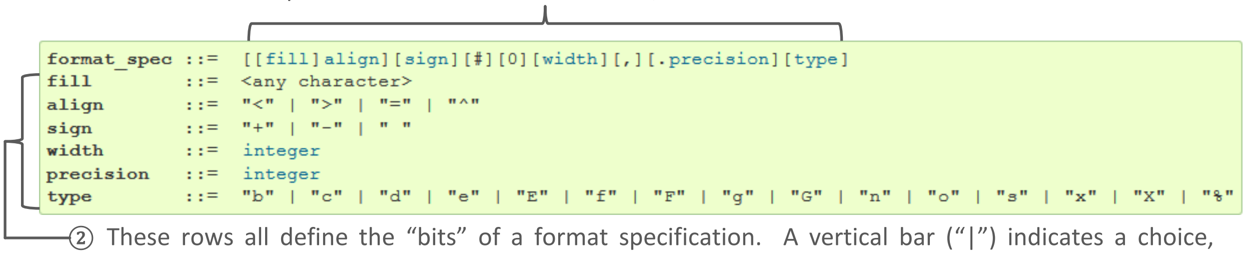
Despite having positive and negative numbers of various lengths and various decimal places, the format specification allows them to all be presented in nicely aligned columns.

So, how does it all work? What do the characters in the format specifications mean?

That's up to **you** to work out. You can no doubt figure out some of it from the examples on the previous page, e.g. "`.2`" rounds a number to 2 decimal places, but trying to memorise all of the possibilities and syntax is pointless – as long as you are *aware* of the general possibilities of formatting, you can *look up the details* when you need them for a specific task.

Doing this will also help to expose you to reading and understanding programming language documentation, which is an important skill for any aspiring programmer to develop. To help get you started on this front, here is a breakdown of the initially-daunting specification for formatting:

① This defines what a format specification can consist of. Square brackets indicate that something is *optional*. Hence, a format specification can be made up of a combination of these "bits", in the listed order.



```
format_spec ::=  [[fill]align][sign][#][0][width][,][.precision][type]
fill        ::=  <any character>
align       ::=  "<" | ">" | "=" | "^"
sign        ::=  "+" | "-" | " "
width       ::=  integer
precision   ::=  integer
type        ::=  "b" | "c" | "d" | "e" | "E" | "f" | "F" | "g" | "G" | "n" | "o" | "s" | "x" | "X" | "%"
```

② These rows all define the "bits" of a format specification. A vertical bar ("|") indicates a choice, e.g. the "align" bit can consist of either <, >, = or ^.

The bits are further described below this table, telling you exactly what they do and how to use them.