Formatted Output

This lesson discusses features of the std.format module, which is a module inside Phobos, a standard library of D.

WE'LL COVER THE FOLLOWING

Formatted output

`std.format` module is not about the core features of the D language. Like all modules that have the prefix `std`, `std.format` is a module inside **Phobos**, the standard library of D.

D's input and output format specifiers are similar to the ones in the C language. The format specifiers and flags are summarized as follows:

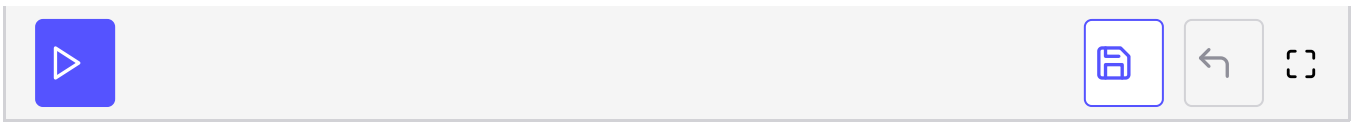
```
Flags (can be used together)
     -      flush left
     +      print the sign
     #      print in the alternative way
     0      print zero-filled
   space    print space-filled

Format Specifiers
     s      default
     b      binary
     d      decimal
     o      octal
    x,X     hexadecimal
    f,F     floating point in the standard decimal notation
    e,E     floating point in scientific notation
    a,A     floating point in hexadecimal notation
    g,G     as e or f

     ,      digit separators

     (      element format start
     )      element format end
     |      element delimiter
```

Format specifiers and flags

We have been using functions like `writeln` with multiple parameters as necessary to print the desired output. The parameters would be converted to their `string` representations and then sent to the output.

Sometimes this is not sufficient. The output may have to be in a very specific format. Let's look at the following code that is used to display items of an invoice:

```
import std.stdio;

void main() {
    double [] items;
    items ~= 1.23;
    items ~= 45.6;

    for (int i = 0; i != items.length; ++i) {

        writeln("Item ", i + 1, ": ", items[i]);

    }

}
```



The output:

```
Item 1: 1.23
Item 2: 45.6
```

Despite the information being correct, we may be required to display it in a different format. For example, maybe the decimal marks (the dots in this case) must line up, and we must ensure that there always are two digits after the decimal mark, as in the following output:

```
Item 1:  1.23
Item 2: 45.60
```

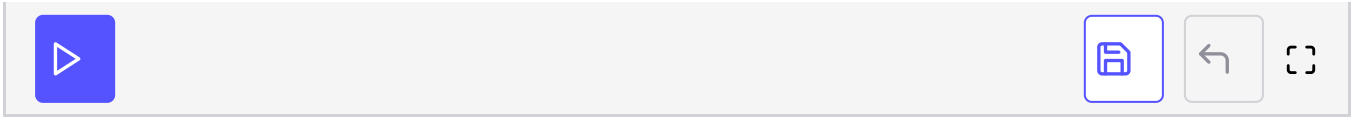
Formatted output is useful in such cases. The output functions `write()` and `writeln()`, which we have been using so far have counterparts that contain the letter `f` in their names: `writef()` and `writefln()`. The letter `f` is short for

formatted. The first parameter of these functions is a format `string` that describes how the other parameters should be printed.

For example, `writefln()` can produce the desired output above with the following format `string`:

```
writefln("Item %d:%9.02f", i + 1, items[i]);
```

```
import std.stdio;

void main() {
    double [] items;
    items ~= 1.23;
    items ~= 45.6;

    for (int i = 0; i != items.length; ++i) {

        writefln("Item %d:%9.02f", i + 1, items[i]);

    }

}
```



Format string

The format string contains regular characters that are passed to the output as-is and special *format specifiers* that correspond to each parameter that is to be printed. Format specifiers start with the `%` character and end with a format character. The format string above has two format specifiers: `%d` and `%9.02f`. Every specifier is associated with a respective parameter in the `writefln()` function, usually in the order of appearance.For example, `%d` is associated with `i + 1` and `%9.02f` is associated with `items[i]`. Every specifier specifies the format of the parameter that it corresponds to. Format specifiers may have parameter numbers as well.

Other than the format specifiers, the characters of the format string are displayed as-is. Such regular characters of the format specifier above are highlighted in `"Item %d:%9.02f"`.

Parts of format specifier

Format specifiers consist of several parts, most of which are optional. These are listed below:

are listed below:

```
%  flags  width  separator  precision  format_character
```

Parts of format specifier

> **Note:** The spaces between these parts are inserted here to help with readability; they are not part of the specifiers.

The `%` character at the beginning and the format character at the end are required; the others are optional.
Because `%` has a special meaning in format strings, when we need to print a `%` as a regular character, we must type it as `%%`.

In the next lesson, we will see a part of formatted output i.e. the format character.