# Format Specifiers: Width, Separator, Precision and Flags

This lesson explains four more parts of the format specifier: width, separator, precision and flags.

WE'LL COVER THE FOLLOWING



## Width

This part determines the *width in characters that the argument is displayed in*. If the width is specified as the character `*`, then the actual width value is read from the next argument (that argument must be an `int`). If width is a negative value, then the `-` flag is assumed.

```
import std.stdio;

void main() {

    int value = 100;
    writefln("In a field of 10 characters:%10s", value);
    writefln("In a field of 5 characters :%5s", value);

}
```

## Separator

*The comma character is used to separate digits of a number in groups*. The default number of digits in a group is 3, but it can be specified after the

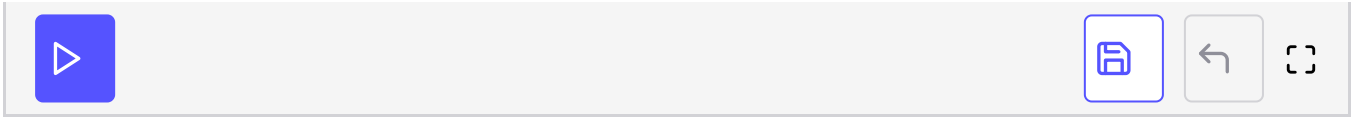
comma:

```
import std.stdio;

void main() {

    writefln("%,f", 1234.5678); // Groups of 3
    writefln("%,s", 1000000);   // Groups of 3
    writefln("%,2s", 1000000);  // Groups of 2

}
```



Separator

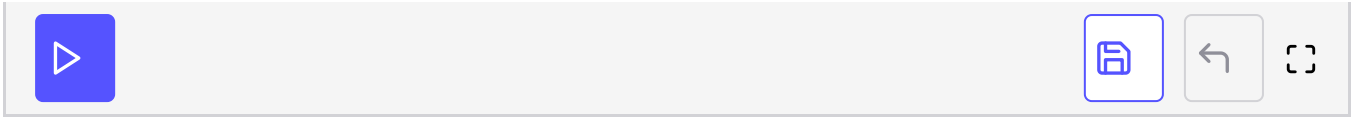
If the number of digits is specified as the character `*`, then the actual number of digits is read from the next argument (that argument must be an `int`).

```
import std.stdio;

void main() {

    writefln("%,*s", 1, 1000000); // Groups of 1

}
```



Groups of 1

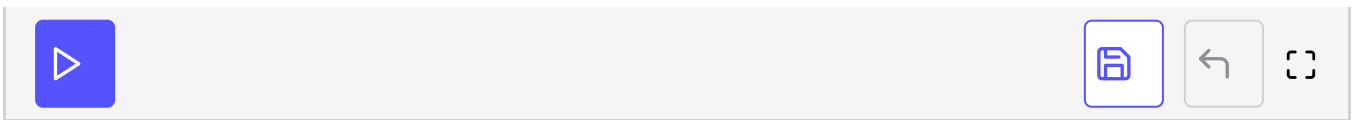
Similarly, it is possible to specify the separator character by using a question mark after the comma and providing the character as an additional argument before the number:

```
import std.stdio;

void main() {

    writefln("%,?s", '.', 1000000); // The separator is '.'

}
```



Using . for separation

## Precision #

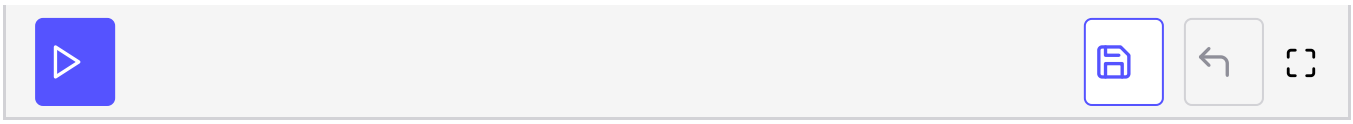
*Precision* is specified after a dot in the format specifier. For floating point types, it determines the precision of the printed representation of the values. If the precision is specified as the character `*`, then the actual precision is read from the next argument (that argument must be an `int`). Negative precision values are ignored.

```
import std.stdio;

void main() {
    double value = 1234.56789;

    writefln("%.8g", value);
    writefln("%.3g", value);
    writefln("%.8f", value);
    writefln("%.3f", value);

    auto number = 0.123456789;
    writefln("Number: %.*g", 4, number);
}
```



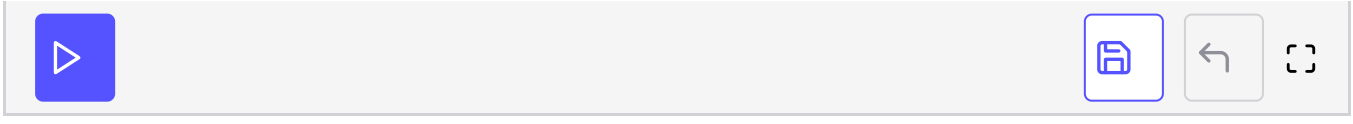
Precision

## Flags #

More than one flag can be specified.
`-`: the value is printed left-aligned in its field; this flag cancels the 0 flag

```
import std.stdio;

void main() {
    int value = 123;
    writefln("Normally right-aligned:|%10d|", value);
    writefln("Left-aligned :|%-10d|", value);
}
```



Flags

`+`: if the value is positive, it is prepended with the `+` character; this flag

`+`: if the value is positive, it is prepended with the `+` character; this flag cancels the space flag.

```
import std.stdio;

void main() {

    writefln("No effect for negative values : %+d", -50);
    writefln("Positive value with the + flag : %+d", 50);
    writefln("Positive value without the + flag: %d", 50);

}
```

`#`: prints the value in an alternate form depending on the format_character

- `o`: the first character of the octal value is always printed as 0
- `x` and `X`: if the value is not zero, it is prepended with `0x` or `0X`
- floating points: a decimal mark is printed even if there are no significant digits after the decimal mark
- `g` and `G`: even the insignificant zero digits after the decimal mark are printed

```
import std.stdio;

void main() {

    writefln("Octal starts with 0 : %#o", 1000);
    writefln("Hexadecimal starts with 0x : %#x", 1000);
    writefln("Contains decimal mark even when unnecessary: %#g", 1f);
    writefln("Rightmost zeros are printed : %#g", 1.2);

}
```

`0`: the field is padded with zeros (unless the value is nan or infinity); if precision is also specified, this flag is ignored.

```
import std.stdio;

void main() {
```

```
    writefln("In a field of 8 characters: %08d", 42);

}
```

**space character:** if the value is positive, a space character is prepended to align the negative and positive values.

```
import std.stdio;

void main() {

    writefln("No effect for negative values: % d", -34);
    writefln("Positive value with space : % d", 56);
    writefln("Positive value without space : %d", 56);

}
```

Space character

---

In the next lesson, we will see the formatted element output.