Character and string Literals

This lesson explains character and string literals, how they are used and what their types are.

WE'LL COVER THE FOLLOWING

Character literals

Character literals are specified within single quotes as in 'a', '\n', '\x21', etc.

As the character itself

The character may be typed directly using the keyboard or copied from a separate text: 'a', 'ş'.

As the character specifier

The character literal may be specified using a backslash character followed by a special character. For example, the backslash character itself can be

specified by `\\`. The following character specifiers are accepted:

Syntax	Definition
\'	single quote
\"	double quote
\?	question mark
\\	backslash
\a	alert (bell sound on some terminals)
\b	delete character
\f	new page
\n	new-line
\r	carriage return
\t	tab
\v	vertical tab

As the extended ASCII character code

Character literals can also be specified by their codes. The codes can be specified either in the hexadecimal system or in the octal system. When using the hexadecimal system, the literal must start with `\x` and must use two digits for the code. Additionally, when using the octal system the literal must start with `\` and have up to three digits. For example, the literals '\x21' and '\41' are both exclamation points in hexadecimal and octal notation respectively.

As the Unicode character code

When the literal is specified with `u` followed by 4 hexadecimal digits, then its type is `wchar`. When it is specified with `U` followed by 8 hexadecimal digits, then its type is `dchar`. For example, '\u011e' and '\U0000011e' are both the Ğ character, having the type `wchar` and `dchar`, respectively.

As named character entity

Characters that have *entity names* can be specified by that name using the HTML character entity syntax '&name;'. D supports all character entities from HTML 5. For example, '€' is €, '♥' is ♥, and '©' is ©.

String literals

String literals are a combination of character literals and can be specified in a variety of ways.

Double-quoted string literals

The most common way of specifying string literals is by typing their characters within double quotes as in “hello.” Individual characters of string literals follow the rules of character literals. For example, the literal “A4 ka\u011fit: 3½TL” is the same as “A4 kağıt: 3½TL.”

WYSIWYG string literals

When `string` literals are specified using back-quotes, the individual characters of the `string` do not obey the special syntax rules of character literals. For example, the literal `c:\nurten` can be a directory name on the Windows operating system. If it were written using double quotes, the `\n` part would mean the new- line character:

```
writeln(`c:\nurten`);
writeln("c:\nurten");
```

The output:

```
c:\nurten ← wysiwyg (what you see is what you get)
c:        ← the character literal is taken as new-line
urten
```

Wysiwyg `string` literals can alternatively be specified using double quotes but prepended with the `r` character: `r"c:\nurten"` is also a wysiwyg `string` literal.

Delimited `string` literals

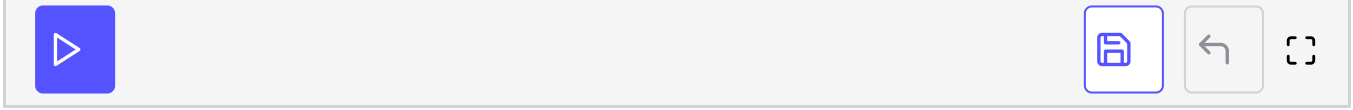
The `string` literal may contain delimiters that are typed right inside the `double` quotes. These delimiters are not considered to be parts of the value of the literal. Delimited `string` literals start with a `q` before the opening double quote. For example, the value of `q".hello."` is `"hello"`; the dots are not part of the value. As long as it ends with a new-line, the delimiter can have more than one character:

```
import std.stdio;

void main() {

writeln(q"MY_DELIMITER
first line
second line
MY_DELIMITER");

}
```



Such a multi-line `string` literal including indentation is called a **heredoc**.

Token `string` literals

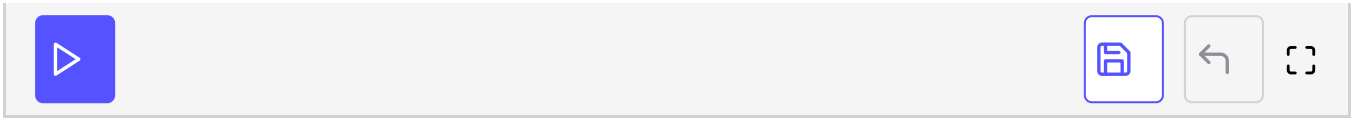
String literals that start with `q` and that use `{` and `}` as delimiters can contain only legal D source code:

```
import std.stdio;

void main() {

    auto str = q{int number = 42; ++number;};
    writeln(str);

}
```



This feature is particularly useful to help text editors display the contents of the `string` as syntax-highlighted D code.

Types of string literals

By default the type of a `string` literal is `immutable(char)[]`. An appended `c`, `w` or `d` character specifies the type of the string explicitly as `immutable(char)[]`, `immutable(wchar)[]` or `immutable(dchar)[]`, respectively. For example, the characters of “hello” are of type `immutable(dchar)`.
We have seen in the strings lesson that these three string types are aliased as `string`, `wstring` and `dstring`, respectively.

24.5 Literals are calculated at compile time

It is possible to specify literals as expressions. For example, instead of writing the value of the total number of seconds in January as 2678400 or 2_678_400, it is possible to specify the second as a product of numbers that make up that value, namely 60 * 60 * 24 * 31. The multiplication operations in that expression do not affect the run-time speed of the program; the program is compiled as if 2678400 was written instead.
The same applies to `string` literals. For example, the concatenation operation in "hello " `~` “world” is executed at compile time, not at run time. The program is compiled as if the code contained the single `string` literal “hello world.”

In the next lesson, you will learn about `enum`.