Literals

Now, we'll learn about literals.

WE'LL COVER THE FOLLOWING

- Raw string literals
  - Example

> Literals are explicit values in the program.

A literal can be a value of any basic data type such as `int`, `bool`, etc. The value of a literal is fixed. Hence, literals are referred to as **constants**.

C++ has several built-in literals:

```
#include <iostream>

int main() {
  int unsigned_int = u'U'; // Unsigned integer
  int hex = 0x2a; // Hexadecimal
  std::string raw = R"(Raw String)"; // Raw string

  std::cout << unsigned_int << ", " << hex << ", " << raw << std::endl;
}
```

Literals are not restricted to simple values. Even a lambda function like

```
lambda function [](int a, int b){ return a+b; }
```

is a **function literal**.

Raw string literals

Raw string literals suppress the interpretation of a string. We can see an example on **line 6** in the code above.

Raw string literals are written in the following format:

```
R"(Raw String)"
```

They also support extended syntax:

```
R"Sep(Raw String)Sep" // Sep (maximal 16 characters long, no spaces, backslashes or colons)
```

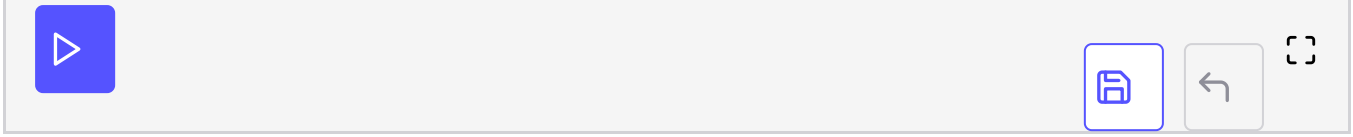
Raw string literals act as import helpers for path expressions and regular expressions:

```
R"(C:\temp\newFile.txt)" // Path expression
R"(c\+\+)" // Regular expression
```

Example

Here's an example of raw string literals in action:

```
#include <iostream>
#include <string>

int main(){

  std::cout << std::endl;

  std::string nat = "C:\temp\newFile.txt";
  std::cout << nat << std::endl;

  // including \t \n
  std::string raw1 = std::string(R"(C:\temp\newFile.txt)");
  std::cout << "\n" << raw1 << std::endl;

  // including \t \n and using delimiter
  std::string raw2 = std::string(R"TRENNER(C:\temp\newFile.txt)TRENNER");
  std::cout << "\n" << raw2 << std::endl;

  // raw string including "
  std::string raw3 = std::string(R"(a raw string including ")");
  std::cout << "\n" << raw3 << std::endl;

  std::cout << std::endl;

}
```



In the next lesson, we'll discuss **user-defined** literals.