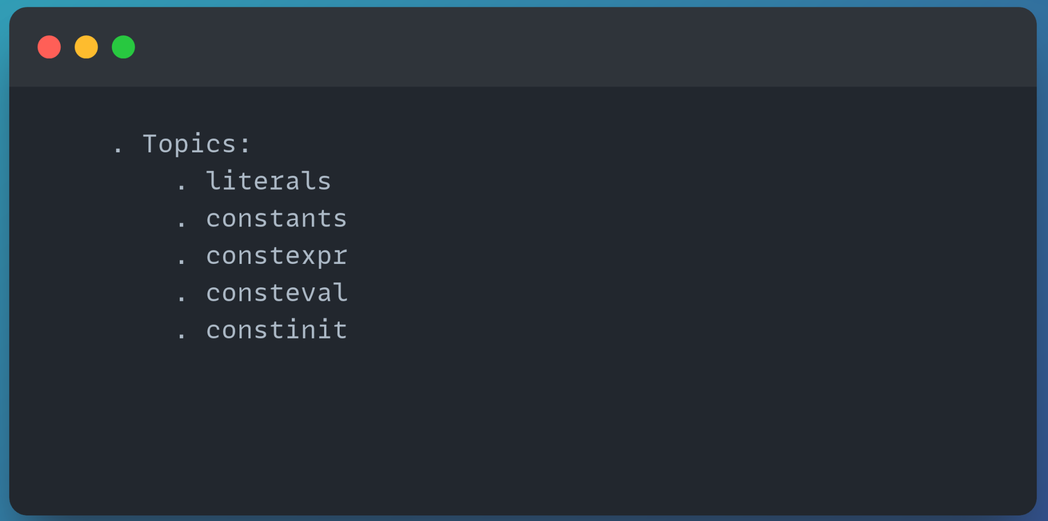
# Literals and constants



# Literals and constants

A literal is a fixed value directly written in the code, representing a constant of a basic type. Examples include integers (42), floating-point numbers (3.14), characters ('A'), strings ("Hello"), and boolean values (true, false). **Literals are not variables** and cannot be modified.

# Literals

```
// 2 Bytes
 short short_var{ -32768 };//  No special literal type for short
 short int short_int{ 455 };//  No special literal type for short
 signed short signed_short{ 122 };// No special literal type for short
 signed short int signed_short_int{ -456 };// No special literal type for short
 unsigned short int unsigned_short_int{ 5678U };

 // 4 Bytes
 const int int_var{ 55 };//
 signed signed_var{ 66 };//
 signed int signed_int{ 77 };//
 unsigned int unsigned_int{ 555U };//

 // 4 or 8 Bytes
 long long_var{ 88L };// 4 OR 8 Bytes
 long int long_int{ 33L };
 signed long signed_long{ 44l };
 signed long int signed_long_int{ 44l };
 unsigned long int unsigned_long_int{ 555ul };

 long long long_long{ 888ll };// 8 Bytes
 long long int long_long_int{ 999ll };
 signed long long signed_long_long{ 444ll };
 signed long long int signed_long_long_int{ 1234ll };
 // Grouping Numbers : C++14 and onwards
 unsigned int prize{ 1'500'00'0u };
 fmt::println("The prize is : {}", prize);
 fmt::println(" signed_long_long_int : {}", signed_long_long_int);
```

# Literals

```
// Narrowing errors
// Possible narrowing errors are cought by the braced initializer method.
// Assignment and functional don't catch that.

unsigned char distance {555u}; //Error [0~255]
unsigned int game_score {-20}; //Error
fmt::println("game_score : {}", game_score);
```

# Literals

```cpp
// With number systems - Hex : prefix with 0x
unsigned int hex_number{ 0x22BU };// Dec 555
int hex_number2{ 0x400 };// Dec 1024
fmt::println("The hex number is : {:x}", hex_number);
fmt::println("The hex number2 is : {}", hex_number2);

// Representing colors with hex
int black_color{ 0xffffff };
fmt::println("Black color is : {}", black_color);


// Octal literals : prefix with 0
int octal_number{ 0777u };// 511 Dec
fmt::println("The octal number is : {}", octal_number);
//!!BE CAREFUL NOT TO PREFIX YOUR INTEGERS WITH 0 IF YOU MEAN DEC
int error_octal{ 055 };// This is not 55 in memory , it is 45 dec
fmt::println("The erronous octal number is : {}", error_octal);
// Binary literals
unsigned int binary_literal{ 0b11111111u };// 255 dec
fmt::println("The binary literal is : {}", binary_literal);
```

# Literals

```
// Other literals. This is just an example and we will learn
// more about  strings as we progress in the course.
char char_literal{ 'c' };
int number_literal{ 15 };
float fractional_literal{ 1.5f };
std::string string_literal{ "Hit the road" };

 //Print the literals with fmt
 fmt::println("The char literal is: {}", char_literal);
 fmt::println("The number literal is: {}", number_literal);
 fmt::println("The fractional literal is: {}", fractional_literal);
 fmt::println("The string literal is: {}", string_literal);
```

# Constants

```
//2.Constants
const int age {34};
const float height {1.67f};

//age = 55; // Can't modify
//height = 1.8f;

int years { 3 * age};
fmt::println("Age: {}, Height: {}, Years: {}", age, height, years);
```

* It is a good practice to declare variables constant by default in your mind and only make them modifiable if you need to.

# constexpr variables

```cpp
//constexpr variables
//constexpr variables are always evaluated at compile time. constexpr implies const
constexpr int SOME_LIB_MAJOR_VERSION {1237};
constexpr int eye_count {2};
constexpr double PI {3.14};
//eye_count = 4;  //Error. constexpr variables are const
fmt::println("eye count: {}", eye_count);
fmt::println("PI: {}", PI);

//int leg_count {2}; // Non constexpr. leg_count is not known at compile time
//constexpr int arm_count{leg_count}; // Error

constexpr int room_count{ 10 };
constexpr int door_count{ room_count + 2};// OK. Can use constexpr to initialize another constexpr

const int table_count{ 5 };
constexpr int chair_count{ table_count * 5 };// Works. Can use const to initialize constexpr

//static_assert( SOME_LIB_MAJOR_VERSION == 123);

//int age = 5;
//static_assert( age == 5); // Error. age is not a constant expression. Can't be used in static_assert.
```

# constexpr functions

```cpp
//constexpr functions
constexpr int add(int a, int b){
  return a + b;
}

int main(){

  auto value1{add(3, 4)}; // Evaluated at compile time. 3 and 4 are literals.

  int factor{2}; // Run time value. Not known at compile time.
  auto value2{add(3, factor)}; // Evaluated at run time

}
```

# consteval functions

```
//consteval functions (Guaranties that the function is evaluated at compile time)
consteval int multiply(int a, int b){
  return a * b;
}

int main(){
  auto value3{multiply(3, 4)}; // Evaluated at compile time. 3 and 4 are literals.
  int factor1{2}; // Run time value. Not known at compile time.
  auto value4{multiply(3, factor1)}; // Error. factor1 is not known at compile time.

}
```

# constinit variables

```cpp
//constinit : C++20. Guarantees that the variable is initialized at compile time.
//Initialization with a run-time value will lead to a compiler error.

//global scope


int main(){
  //Declaring constinit variables somewhere inside (directly or indirectly) the main function scope will lead to a compiler error.

  constinit double height{1.72}; //Error: 'height': 'constinit' only allowed on a variable declaration with static or thread storage duration
  constinit int age{ add(3,4) }; // Error: 'age': 'constinit' only allowed on a variable declaration with static or thread storage duration
                  // These errors mean that constinit variables are only allowed in the global scope.

}
```

# constinit variables

```cpp
//constinit : C++20. Guarantees that the variable is initialized at compile time.
//Initialization with a run-time value will lead to a compiler error.

//global scope
constinit int age{ add(3,4) }; // Ok. age is in the global scope and is initiallized with a compile time value
//Can change the value in a constinit variable. See body of do_work().
//We just can't do it here because this is a declaration and not a definition.
const constinit double height{1.72}; // const and constinit can be combined

int randomizer = 5;
//constinit int  car_count{add(3,randomizer)}; // Error. randomizer is not a compile time value


int main(){

  age = 30; // Can change a constinit variable
  //height = 1.8; // Error. height is now const.

  fmt::println("Age: {}, Height: {}", age, height);
}
```