C++ Basic Syntax

The C++ Language, Data Types, Expressions, Console I/O

SoftUni Foundation

Georgi Georgiev

A guy that knows C++

```
3   using namespace std;
4
5   int main() {
6       cout << "Hello World" << endl;
7       return 0;
```

Table of Contents

SoftUni Foundation

Questions

SoftUni Foundation

sli.do

#cpp-softuni

SoftUni Foundation

What is C++?

Fast, Mid-level, Multi-Platform

What is C++

SoftUni Foundation

- General purpose programming language
- Compiles to binary – i.e. multi-platform
- Statically typed – data types, classes, etc.
- Multi-paradigm
- Fast

C++ Philosophy

SoftUni Foundation

- Features immediately useful in the real world
- Programmers free to pick their own style
- Useful features more important than preventing misuse
- Features you do not use, you do not pay for
- Programmer can specify undefined behavior

- More: en.wikipedia.org/wiki/C++#Philosophy

No, not this guy, sorry

C++ HISTORY

Created 1979 – 1983 by Bjarne…

This one! See, much better!

C++ HISTORY

Created 1979 – 1983 by Bjarne…

… Stroustrup

Originally “C with Classes”

C++ Standards

SoftUni Foundation

- C++ 98 – first standardized C++ version, C++ 03 – minor revision
- C++ 11 – major revision, C++14 – revision with fixes
  - Many new features and improvements
  - Initializer lists, Rvalue references, lambdas, range-based loops, etc.
- C++ 17 – latest official revision

C++ Compilers & IDEs

SoftUni Foundation

- A C++ compiler turns C++ code to assembly
- An IDE is software assisting programming
  - Has a Compiler, Linker, Debugger, Code Editor
  - Code organization, Tools, Diagnostics
- There are lots of C++ compilers and IDEs
  - Free, open-source, proprietary

Code::Blocks

SoftUni Foundation

- Free C & C++ IDE
- Comes with MinGW GCC compiler
  - C++11 support needs to be enabled from settings
  - C++14 support needs more complicated setting to enable
- Lightweight
  - Can compile single .cpp file
  - Can be used for bigger projects with many files, references, etc.

SoftUni Foundation

Program Structure, Running C++

Entry Point, Building and Running

Hello World

- Here's a classic C++ "Hello World" example

Include the input-output library

Say we're working with the `std` namespace

```
1  #include <iostream>
2  using namespace std;
3
4  int main(int argc, char * argv[]) {
5      cout << "Hello World!" << endl;
6      return 0;
7  }
```

These are optional

Print to the console

"main" function – our entry point

For `main`, `0` means everything went ok

C++ Entry Point & Termination

SoftUni Foundation

- The `main` function – entry point of the program
  - No other function can be named "`main`"
  - C++ needs specific function to start from
  - Everything else is free-form – code ordering, namings, etc.
  - Can receive command line parameters
- Termination – `main` finishes (returns), the program stops
  - The return value of main is the "exit code“
  - `0` means no errors – informative, not obligatory

Program Structure: Including Libraries

SoftUni Foundation

- C++ has a lot of functionality in its standard code libraries
- C++ can also use functionality from user-built code libraries
- Say what libraries to use with the **`#include`** syntax
- For now, for standard libraries: put the library name in **`<>`**

```
1  #include <iostream>
2  using namespace std;
3
4  int main(int argc, char * argv[]) {
```

`iostream` contains console I/O functionality

Program Structure: Blocks

SoftUni Foundation

- Basic building block (pun intended) of a program
- Most actual program code is in blocks, aka “bodies”
- Start with `{` and end with `}`, can be nested
- Functions’ (`main()`), loops’ & conditionals’ code is in blocks

```
4  int main(int argc, char * argv[])
5  {
5      cout << "Hello World!" << endl;
6      return 0;
7  }
```

`main()` code block

Program Structure: Statements & Comments

SoftUni Foundation

- Statement: a piece of code to be executed
  - Blocks consist of statements
- Statements contain C++ code and end with a **;**

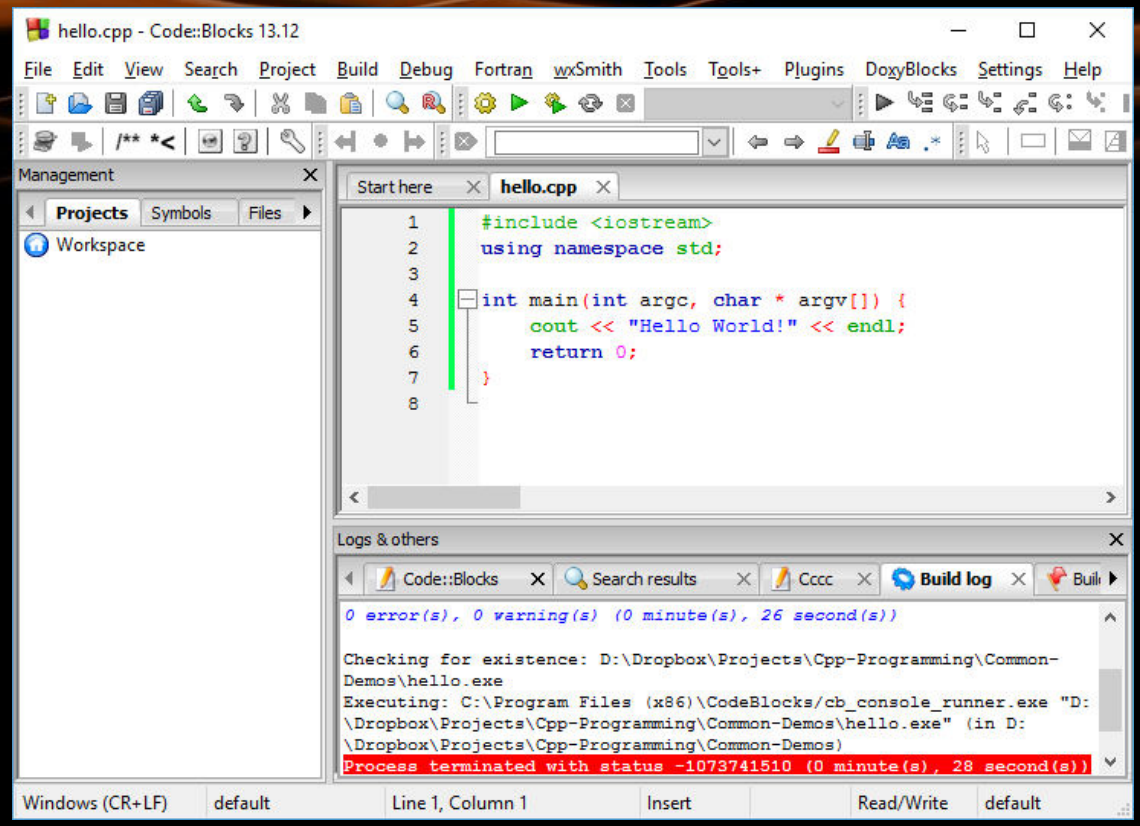
```
4   int main(int argc, char * argv[])
5   {
5       cout << "Hello World!" << endl;
6       return 0;
7   }
```

A statement

Another statement
(Note: usually 1 per line)

- C++ has comments (parts of the code ignored by compiler)
  - **//** comments a line, **/*** starts a multi-line comment, ***/** ends it

SoftUni Foundation



C++ Hello World

Live Demo

SoftUni Foundation

Variables & Primitive Types

Types, Declaration, Initialization, Scope

Fast Intro: Declaring and Initializing Variables

SoftUni Foundation

- `<data_type> <identifier> [= <initialization>];`
- Declaring: `int num;`
- Initializing: `num = 5;`
- Combined: `int num = 5`,
  and additionally `int num(5);` or `int num{5};` (C++11)
- Can declare multiple of same type by separating with comma (`,`)
  - `int trappist1BMassPct=85, trappist1CMassPct=80;`
- NOTE: `int num()` is not a default initialization – it's a function declaration.


SoftUni
Foundation
Declaring & Initializing Variables
LIVE DEMO

Quick Quiz

SoftUni Foundation

- Tony gives George 5 apples
- Later, Angus gives George 3 apples
- How many apples does George have?

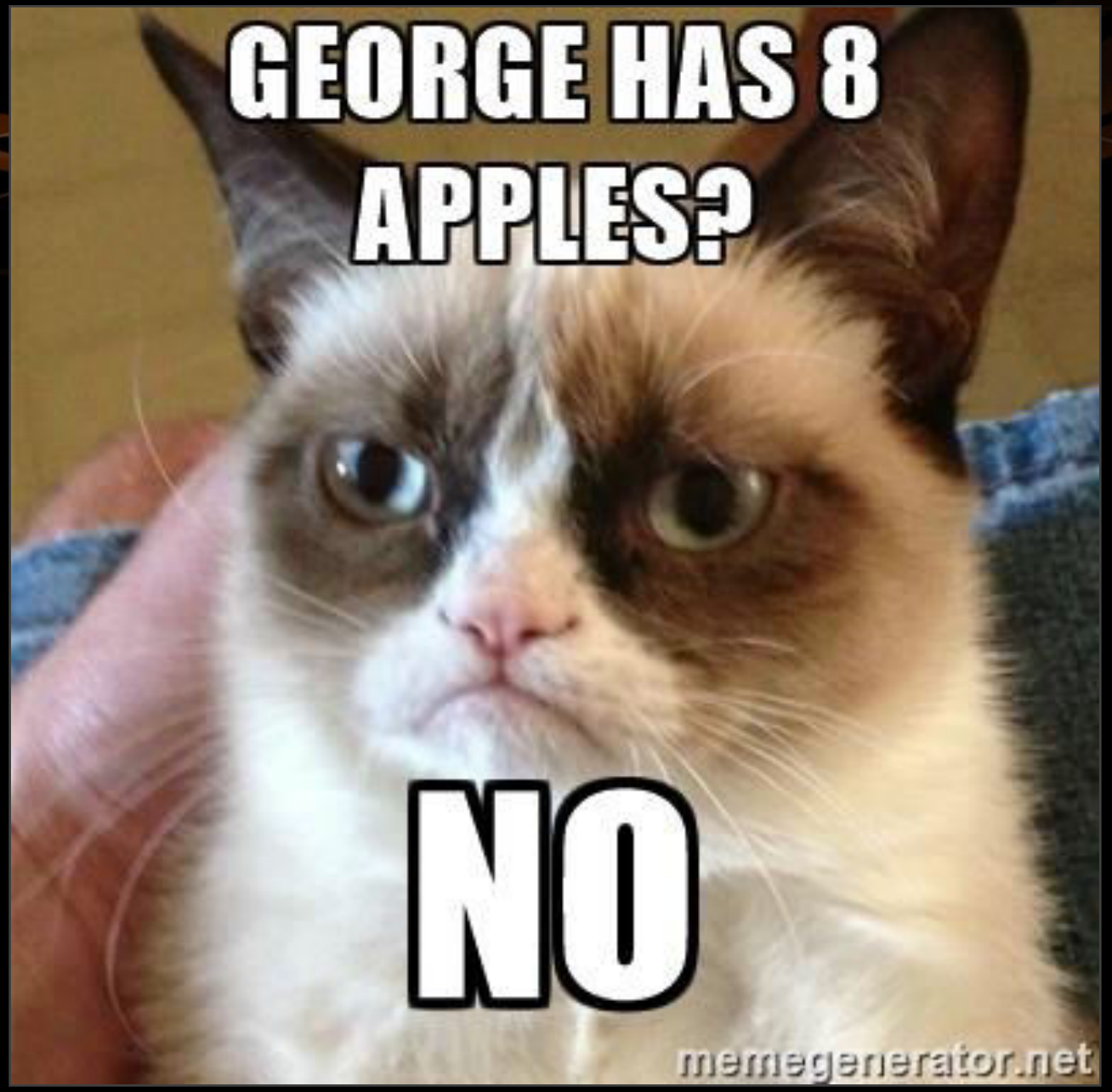
TIME:

C++ PITFALL: UNINITIALIZED LOCALS

George has 13837 apples.

Why?

Nobody said George had 0 apples to begin with.




SoftUni
Foundation
Uninitialized Locals
LIVE DEMO

Local & Global Variables

SoftUni Foundation

- Global: defined outside blocks, usable from all code
- Local: defined inside blocks, usable only from code in their block
- Locals DO NOT get initialized automatically
- Globals get initialized to their "default" value (**0** for numerics)

```
int secondsInMinute = 60;
int minutesInHour = 60;
int hoursInDay = 24;
int secondsInHour = secondsInMinute * minutesInHour;
int main() {
    int days = 3;
    int totalSeconds = days * hoursInDay * secondsInHour;
```


SoftUni
Foundation
Global & Local Variables
LIVE DEMO

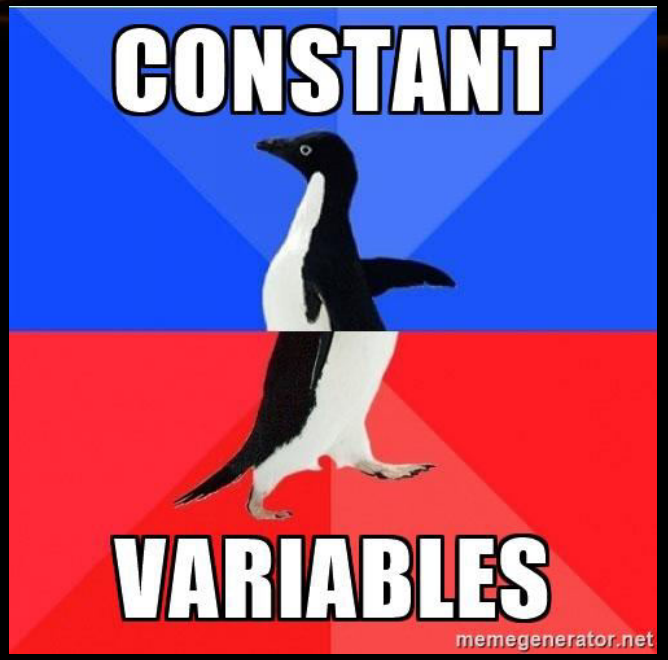
const Variables

SoftUni Foundation

- C++ supports constants – "variables" that can't change value
- Can and MUST receive a value at initialization, nowhere else
- Can be local, can be global
- **secondsInMinute**, **minutesInHour**, etc., aren't things that normally change in the real world – the following won't compile:

```
const int secondsInMinute = 60;
int main() {
    secondsInMinute = 13; //compilation error
```

- NOTE: the const keyword has other uses which we'll discuss later on



const Variables

LIVE DEMO

Other variable modifiers

SoftUni Foundation

- `static` variables initialize once and exist throughout program
  - Can be used to make a local variable that acts like a global one
  - Can be used on a global variable, but has no real effect
- `extern` tells the compiler a variable exists somewhere in a multi-file project (to avoid multi-declaration)

SoftUni Foundation

Primitive Data Types

Integer, Floating-point and Symbolic values

Integer Types – `int`

SoftUni Foundation

- C++ has "only one" integer type – **`int`**
- "Width" modifiers control the type's size and sign
  - **`short`** – at least 16 bits; **`long`** – at least 32 bits
  - **`long long`** – 64 bits (C++11, Windows supports it on C++03 too)
- **`signed`** & **`unsigned`** – use or not use memory for sign data
- Modifiers can be written in any order
- **`int`** can be omitted if any modifier is present
- Defaults: **`int`** "usually" means **`signed long int`**

Integer Sizes and Ranges

SoftUni Foundation

- The C++ standard doesn't have very strict size guarantees
  - `int` is 32-bits on most mainstream PCs
  - Use `sizeof(int)` to get the size (in bytes) on your system
- Ranges depend on size, a 32-bit integer has about 4 billion values
  - So, a signed 32-bit integer has the range of about (-2 billion, +2 billion)
  - Average human lifespan = 2 billion seconds. `int` is older than you!
- Sizes: http://en.cppreference.com/w/cpp/language/types#Properties
- Ranges: http://en.cppreference.com/w/cpp/language/types#Range_of_values

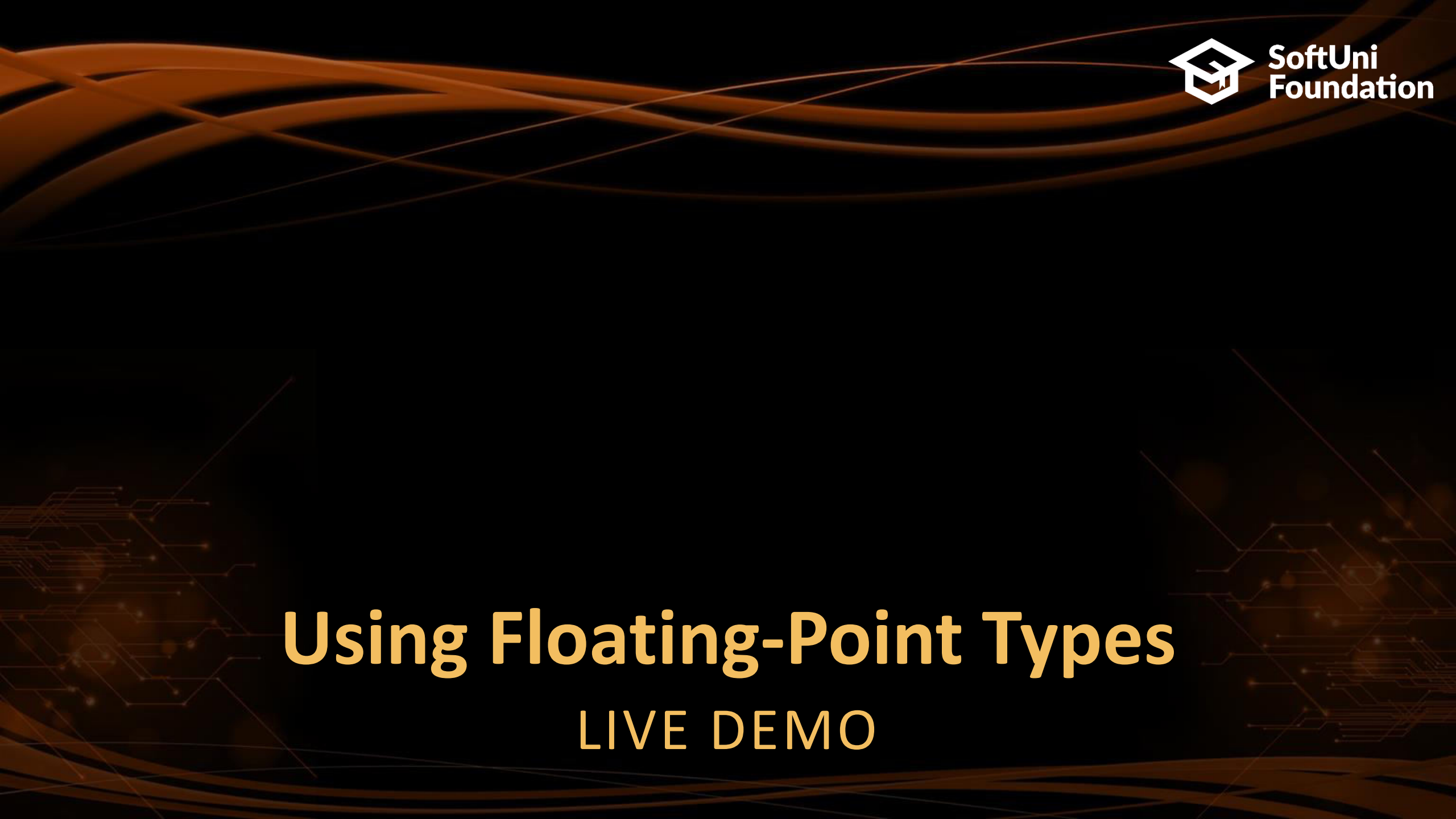

SoftUni
Foundation
Integer Types
LIVE DEMO

Floating-Point Types

SoftUni Foundation

- Represent real numbers (approximations)
  - 2.3, 0.7, -Infinity, -1452342.2313, NaN, etc.
- `float`: single-precision floating point, usually IEEE-754 32-bit
- `double`: double-precision, usually IEEE-754 64-bit

Name	Description	Size*	Range*
float	Floating point number.	4bytes	$\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$ (~7 digits)
double	Double precision floating point number.	8bytes	$\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$ (~15 digits)
long double	Long double precision floating point number.	8bytes	$\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$ (~15 digits)


SoftUni
Foundation
Using Floating-Point Types
LIVE DEMO

Guaranteeing Type Sizes

SoftUni Foundation

- C++ has some cross-platform support for fixed-size types
- There is the C++11 **`bitset`**, the **`cstdint`** library, etc.
- Code running on most systems has access to system macros
  - E.g. on Windows: **`INT32`**, **`INT8`**, **`FLOAT`**, **`INT_PTR`**, **`LONG_PTR`**
- Universally-portable code with fixed sizes is not really possible
  - Do you really expect a toaster to guarantee a 64-bit integer?
- In most cases, type-size guarantees are not necessary

SoftUni Foundation

char

No, not this char

The C++ char

```
char a = 'a';
```

Character Types – char

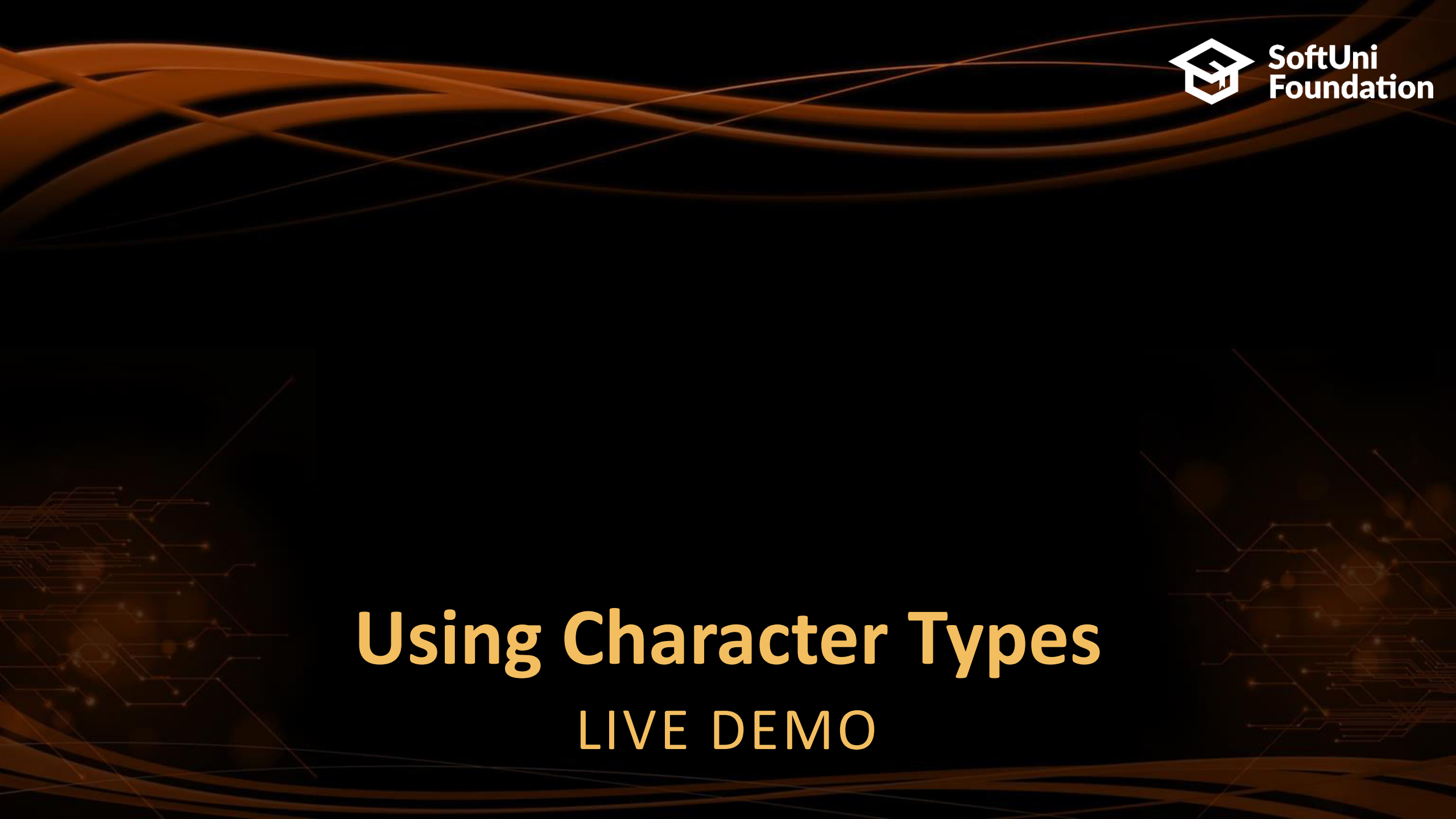
- **char** is the basic character type in C++
- Basically an integer interpreted as a symbol from ASCII
- Guaranteed to be 1 byte – a range of 256 values
- Initialized by either a character literal or a number (ASCII code)

```
int main() {
    char letter = 'a';
    char sameLetter = 97;
    char sameLetterAgain = 'b' - 1;
    cout << letter << sameLetter << sameLetterAgain << endl;
    return 0;
}
```

Using `char` as a Number

SoftUni Foundation

- `char` is essentially a 1-byte integer, it can be used as such
- `char` is also useful when processing data byte-by-byte
- Use `signed` or `unsigned` when using `char` as a number
  - Guarantees range of `char` as `[-128, 127]` or `[0, 255]`
  - Otherwise the system decides what's best for characters
  - … you don’t want the system pretending to be smart
- Avoid using `char` as a number unless you have a good reason to

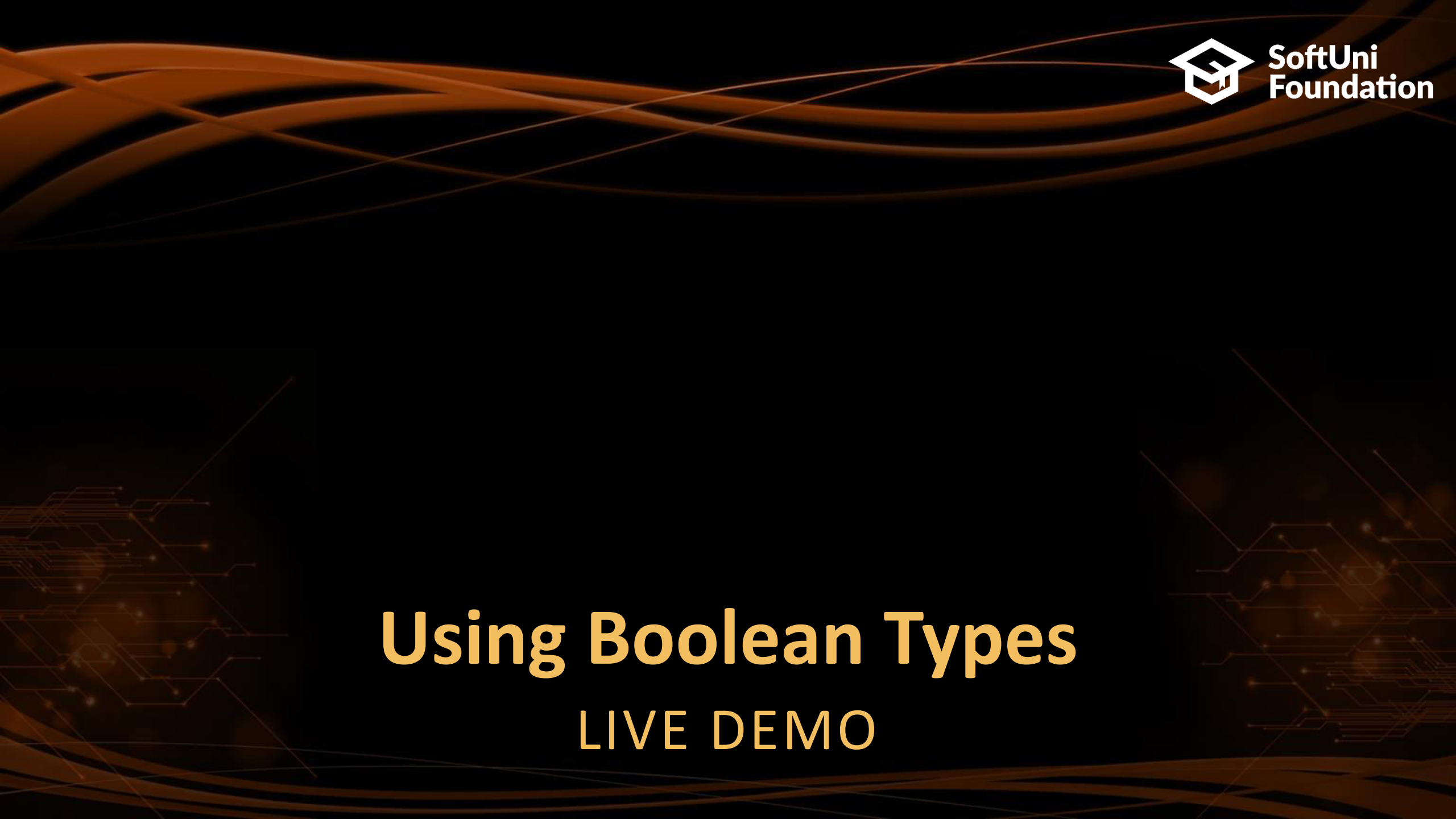

SoftUni
Foundation
Using Character Types
LIVE DEMO

Boolean Type – bool

SoftUni Foundation

- **bool** – a value which is either **true** or **false**, takes up 1 byte
- Takes **true**, **false**, or numeric values
  - Any non-zero numeric value is interpreted as **true**
  - Zero is interpreted as **false**

```
int main() {
    bool initializedWithKeyword = true;
    bool initializedWithKeywordCtor(false);
    bool initializedWithZero = 0;
    bool initializedWithNegativeNumber(-13);
```

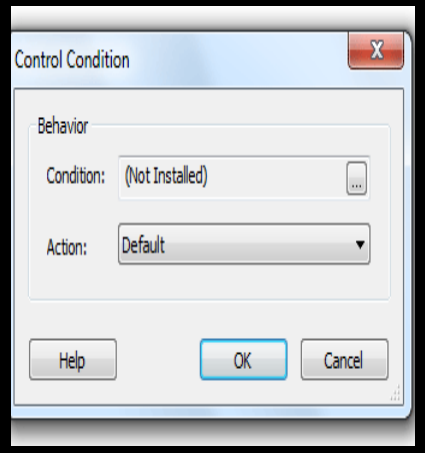

SoftUni
Foundation
Using Boolean Types
LIVE DEMO

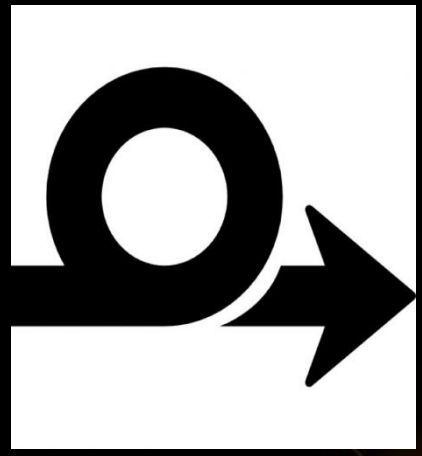
Implicit & Explicit Casting

SoftUni Foundation

- Types which "fit" into others can be assigned to them implicitly
- For integer types, "fit" usually means requiring less bytes
  - Valid: `char a = 'a'; int i = a;`
  - NOT VALID: `int i = 97; char a = i;`
  - For floating point, `float` fits into `double`
- If you really want to store a "bigger" type in a "smaller" type:
  - Explicitly cast the "bigger" type to the "smaller" type:
    `smallType smallVar = (smallType) bigVar;`
- Can lose accuracy if value can't be represented in "smaller" type

SoftUni Foundation





Expressions, Operators, Conditionals, Loops,

Literals, `if`, `switch`, `else`, `for`, `while`

C++ Numeric Literals

SoftUni Foundation

- Represent values in code, match the primitive data types
- Integer literals – value in a numeral system

```
unsigned long long num;
num = 5; num = -5; num = 5L; num = 5ULL; num = 0xF;
```

- Floating-point literals – decimal OR exponential notation
  - Suffix to describe precision (single or double-precision)

```
double num;
num = .42; num = 0.42; num = 42e-2;
float floatNum;
floatNum = .42f; floatNum = 0.42f; floatNum = 42e-2f;
```

Non-Numeric Literals

SoftUni Foundation

- Character literals – letters surrounded by apostrophe (')

```
char letter = 'a';
```

- String literals – a sequence of letters surrounded by quotes (")

```
cout << "Hello World!" << endl;
```

- Boolean literals – `true` and `false`

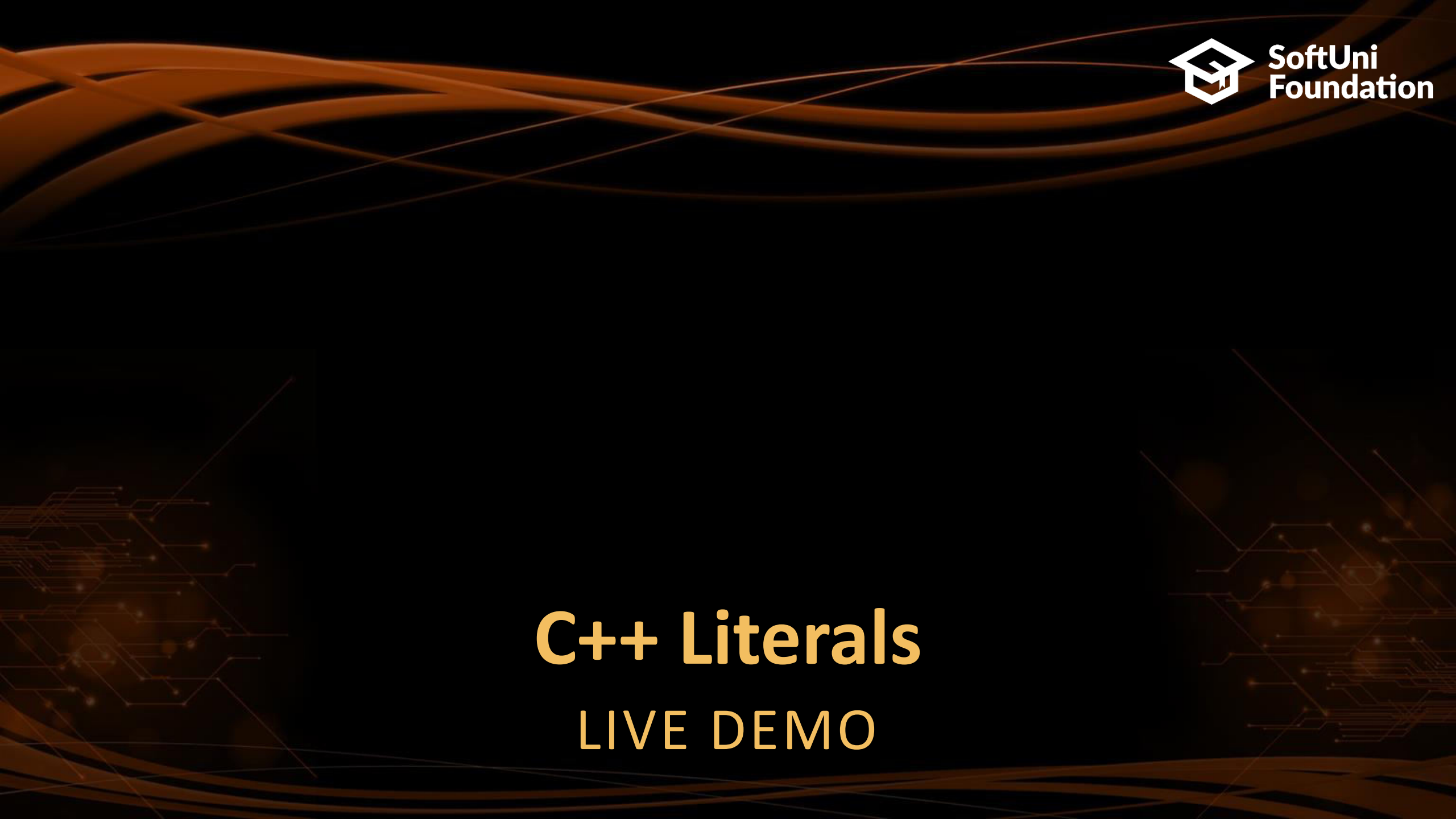
```
bool cppIsCool = true;
```

C++ Literals – Things to Keep in Mind

SoftUni Foundation

- Integers are positive – writing a `-` does a minus operation
- Integers should be suffixed if they are too large for a simple `int`
  - Compiler might try to extend automatically, but don't rely on it
- Floating-points are double by default, suffix with `f` for `float`

```
#include<iostream>
using namespace std;
int main() {
    cout << -42 << " " << 052 << " " << 0x2a << " " << 0x2A << endl
        << 0.42 << " " << .42f << " " << 42e-2 << endl;
}
```



C++ Literals

LIVE DEMO

Expressions and Operators

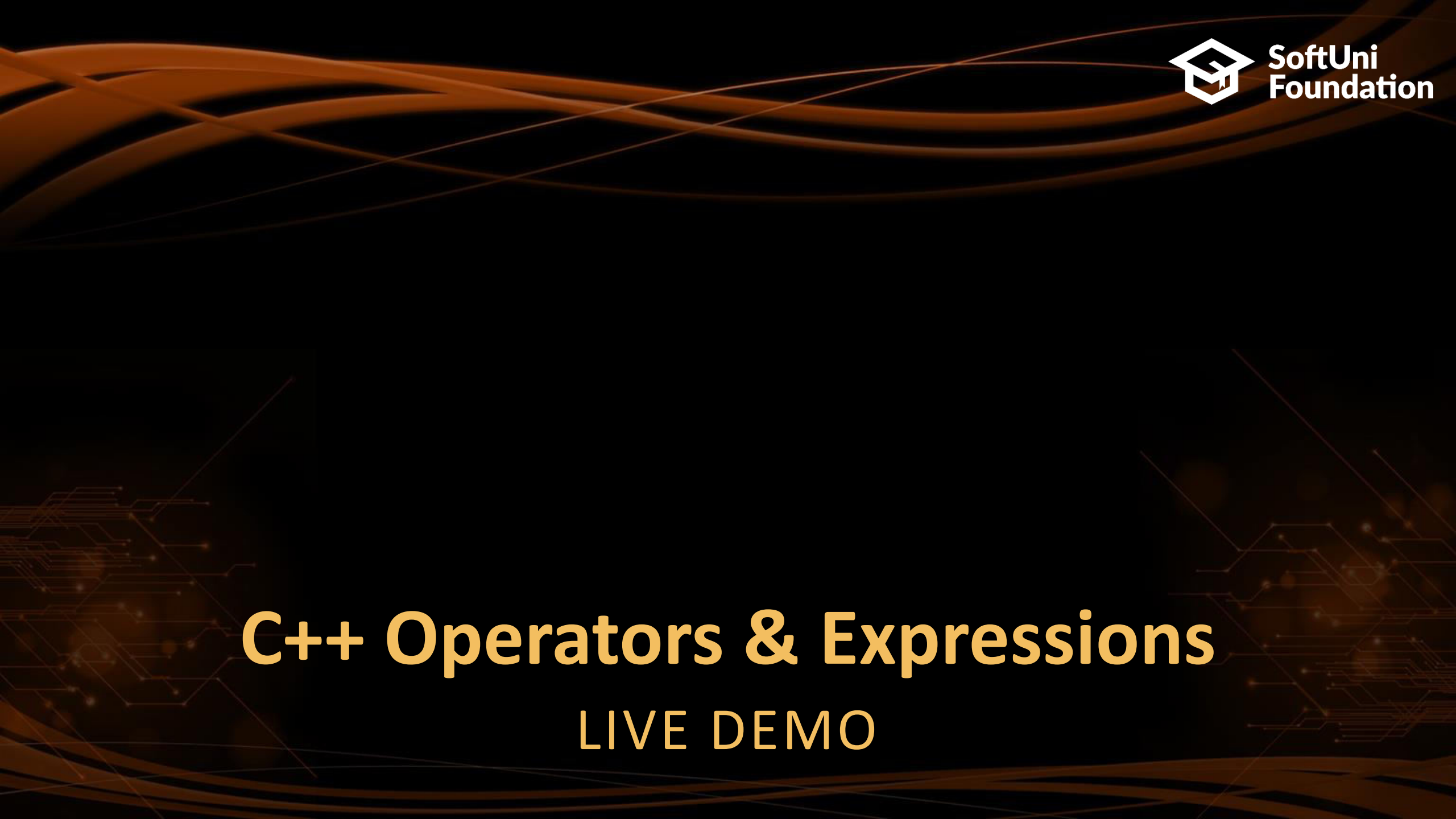
SoftUni Foundation

- Operators in C++ are similar to those in other languages
- Operators perform actions on one or more variables/literals
  - Can be customized for different behavior based on data type
- C++ operator precedence and associativity table: http://en.cppreference.com/w/cpp/language/operator_precedence
  - Don't memorize. Use brackets or check precedence when needed
- Expressions: literals/variables combined with operators/functions

Commonly Used C++ Operators

SoftUni Foundation

Category	Operators											
Arithmetic	+	-	*	/	%	++	--					
Logical	&&	||	^	!								
Binary	&	|	^	~	<<	>>						
Comparison	==	!=	<	>	<=	>=						
Assignment	=	+=	-=	*=	/=	%=	&=	|=	^=	<<=	>>=	
String concatenation	+											
Other	.	[]	()	a?b:c	new	delete	*	->	::	(*type*)	<<	>>



C++ Operators & Expressions

LIVE DEMO

Conditionals

SoftUni Foundation

- The **`if-else`** statement takes in a Boolean expression
  - If the expression evaluates to **`true`**, the **`if`** block is executed
  - If the expression evaluates to **`false`**, the **`else`** block is executed

```
double value1 = 5 * 5 / 2.f, value2 = 5 * 5 / 2;
if (value1 > value2) {
    cout << "value1 is larger" << endl;
} else {
    cout << "value2 is larger" << endl;
}
```

  - The **`else`** block is optional
  - Block **`{ }`** brackets can be omitted if only 1 statement

“Chaining” if-else

CAN “CHAIN” SEVERAL CHECKS ONE AFTER THE OTHER

```
if (value1 > value2) {
    cout << "value1 is larger";
} else if (value1 == value2) {
    cout << "values are equal";
} else {
    cout << "value2 is larger";
}
```

THE CODE BELOW IS EQUIVALENT. EACH ELSE BLOCK CONTAINS 1 “IF” STATEMENT, SO THEY DON’T NEED BRACKETS. THE LEFT VARIANT SKIPS THE BRACKETS

```
if (value1 > value2) {
    cout << "value1 is larger";
} else {
    if (value1 == value2) {
        cout << "values are equal";
    } else {
        cout << "value2 is larger";
    }
}
```

Quick Quiz TIME:

SoftUni Foundation

- What will this code print out to the console?

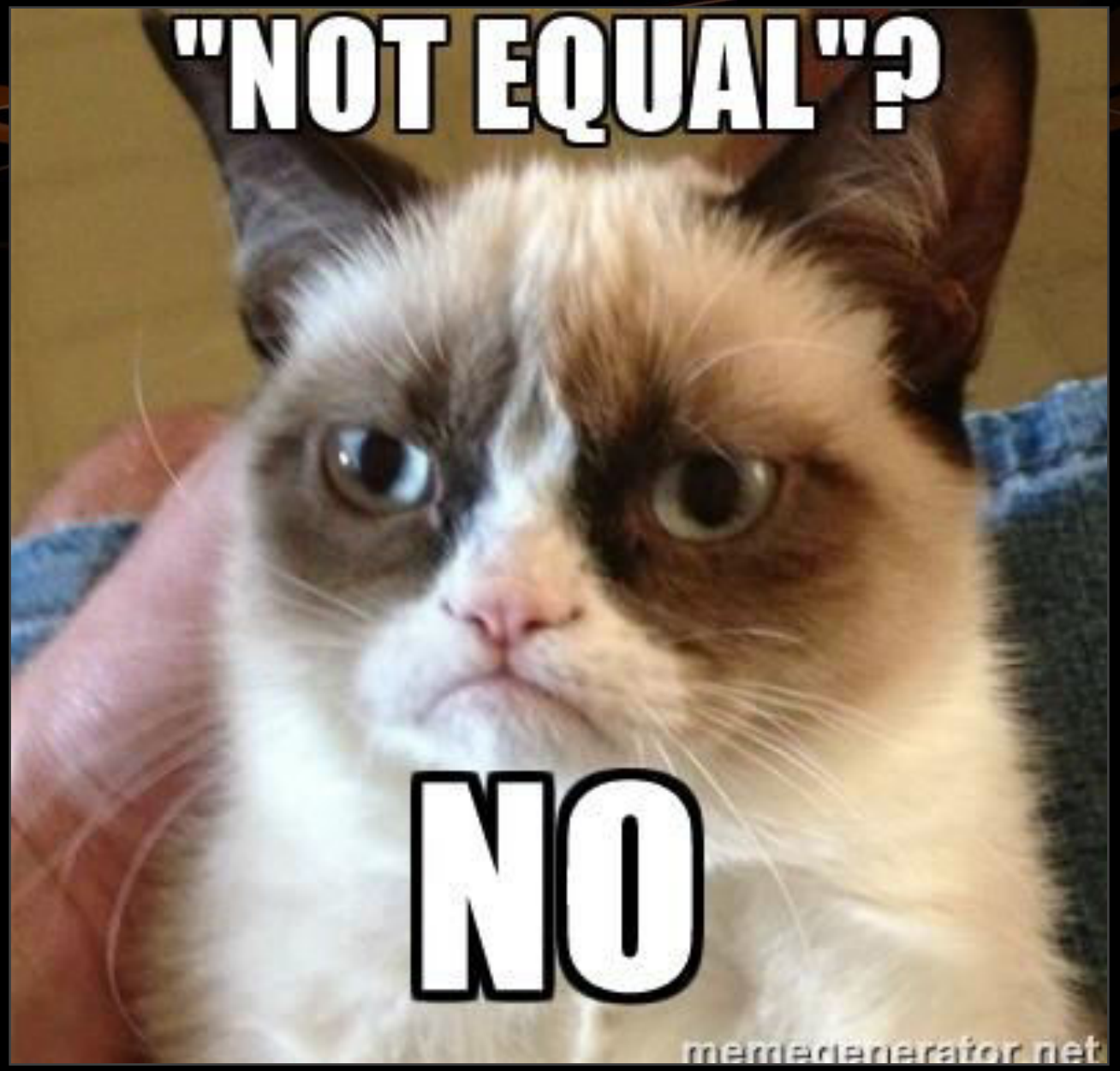
```
#include<iostream>
using namespace std;

int main() {
    int a = 5;
    int aMod3 = a % 3;

    if (a = aMod3) {
        cout << "equal" << endl;
    } else {
        cout << "not equal" << endl;
    }

    return 0;
}
```

C++ PITFALL: ASSIGNMENT INSTEAD OF COMPARISON IN CONDITIONAL

Check the code again.

The expression in the `if` is not the `==` check, it is the `=` assignment,

Which here is non-zero, evaluates to `true`, so we go in the `if` block




SoftUni
Foundation
if and if-else
LIVE DEMO

switch-case

- Example of C++ switch-case usage

```
switch (day)
{
    case 1: cout << "Monday"; break;
    case 2: cout << "Tuesday"; break;
    case 3: cout << "Wednesday"; break;
    case 4: cout << "Thursday"; break;
    case 5: cout << "Friday"; break;
    case 6: cout << "Saturday"; break;
    case 7: cout << "Sunday"; break;
    default: cout << "Error!"; break;
}
```

`switch-case` structure

SoftUni Foundation

- The C++ `switch` statement takes in
  - An integer expression OR an enumeration type
  - OR something which converts to an `int` (like `char`)
- The `case` block can contain `case` labels and any other code
- Each label has an expression of the same type as the switch
- The `case` block can also contain the `break` statement
  - If reached, code continues from after the `case` block
- There is a special `default` label (without an expression)

`switch-case` execution

SoftUni Foundation

- `switch` evaluates the expression and finds the matching `case`
- Any code before the matching `case` is skipped
- Any code after the matching `case` is executed
  - Until `break` or the end of the block is reached
  - Without break, labels after the matching one will be executed!
- If there is no matching `case`
  - If the block contains the special `default` label, it is executed
  - Otherwise the case block is skipped


SoftUni
Foundation
switch-case
LIVE DEMO

for Loop

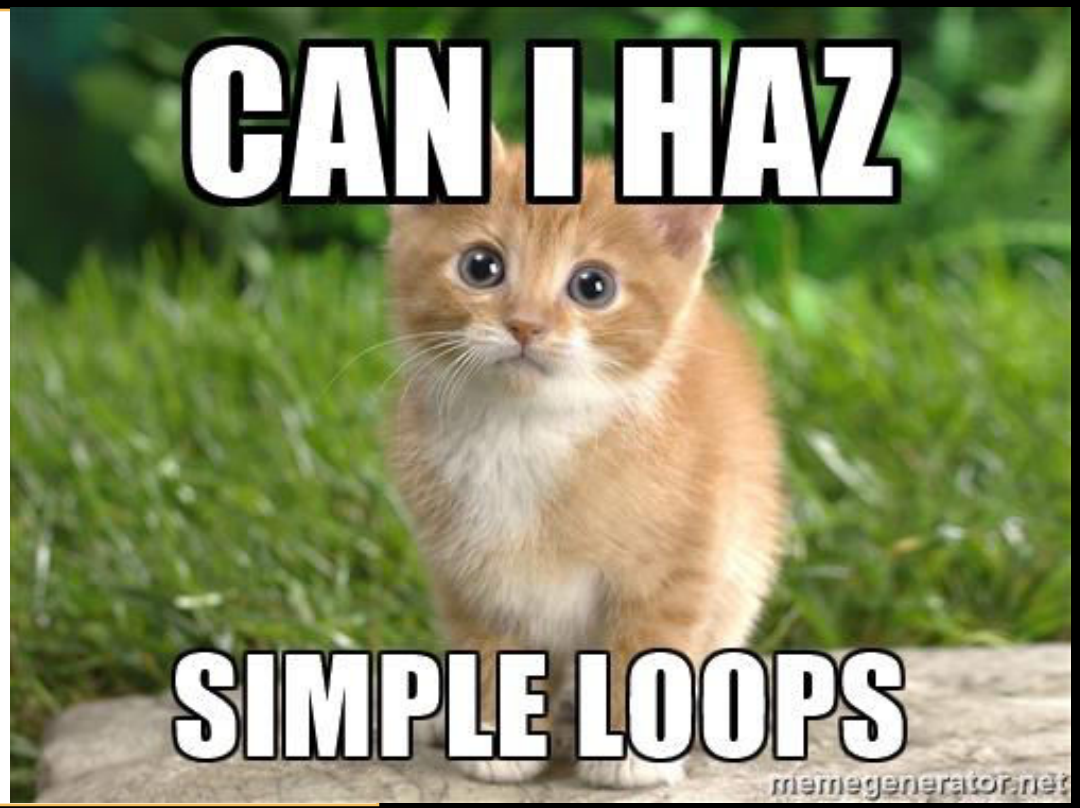
SoftUni Foundation

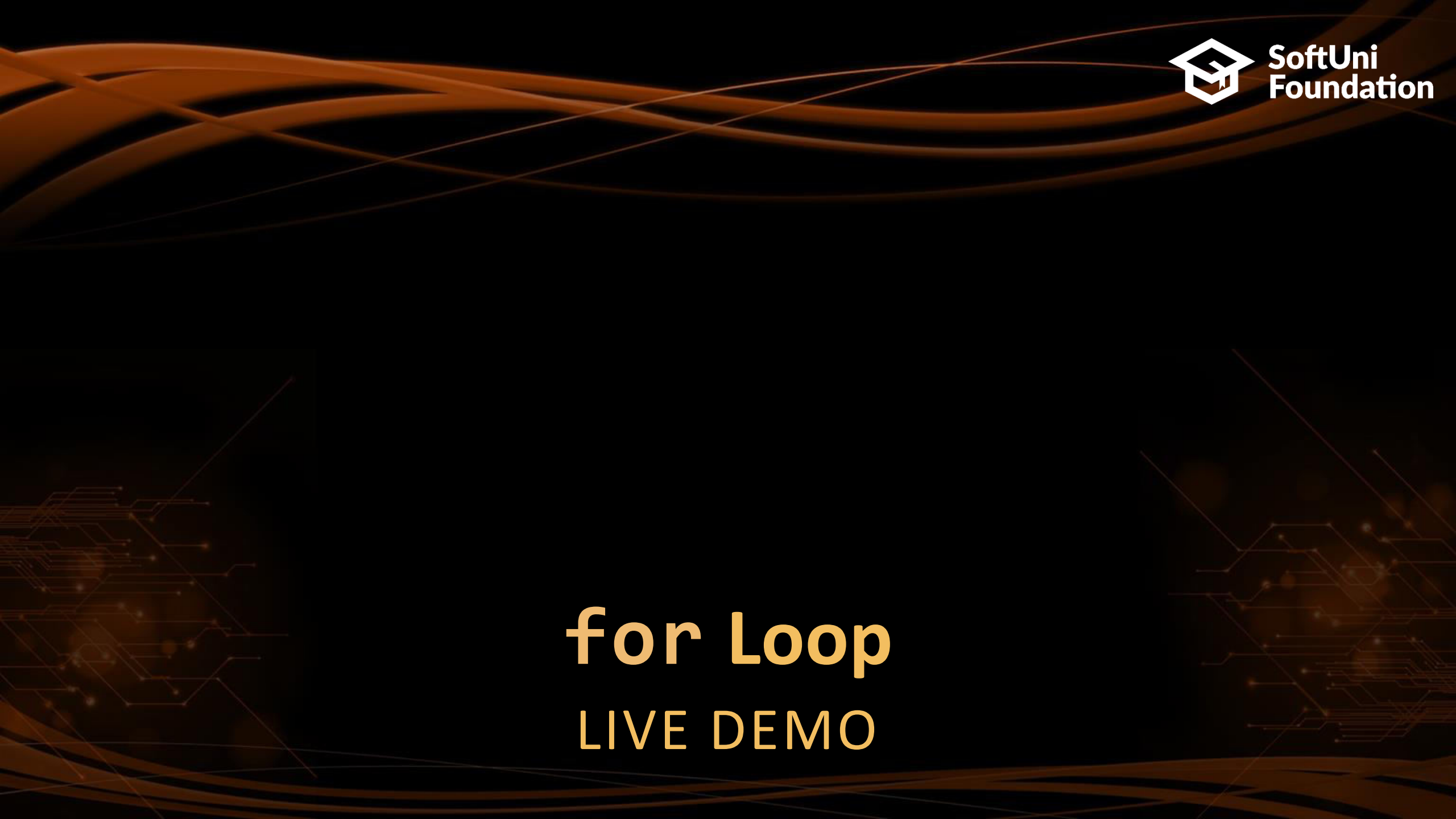
- `for([init]; [condition]; [increment]) {…}`
- The `init` statement can declare and initialize variables
  - Declared variables are usable only IN the `for`'s body
- The loop runs while the `condition` statement is `true`
- `increment` is executed AFTER the `for`'s body
  - Can execute any expression
- Expressions inside `init` and `increment` are separated by comma (`,`)

Fibonacci with "Empty" for Loop

- Every time you do something like this, a kitten dies

```
int currentNum = 1;
for(int i = 2, lastNum = 1, newCurrent;

    i < fibonacciToCalculate;

    i++,
    newCurrent = lastNum + currentNum,
    lastNum = currentNum,
    currentNum = newCurrent)
{ }

cout << currentNum << endl;
```




SoftUni
Foundation
for Loop
LIVE DEMO

while and do-while Loops

SoftUni Foundation

- while (condition) { body code; }
  - Executes until condition becomes false, may never execute

```
int age = 0;
while (age < 18) {
    cout << "can't drink at age " << age << endl;
    age++;
}
cout << "age " << age << ", can finally drink!" << endl;
```

- do { body code; } while (condition);
  - First executes body, then checks condition
  - Guaranteed to execute at least once

SoftUni Foundation

while and do-while Loops

LIVE DEMO

Loops

- C++ loop control keywords:
  - `break` – interrupts the loop and continues after its block
  - `continue` – the current iteration skips the remaining part of the loop block
- C++11 also added a range-based for loop
  - We'll discuss it in the lecture on arrays

SoftUni Foundation

Basic Console I/O

Writing to and Reading from the Console

C++ Streams 101

SoftUni Foundation

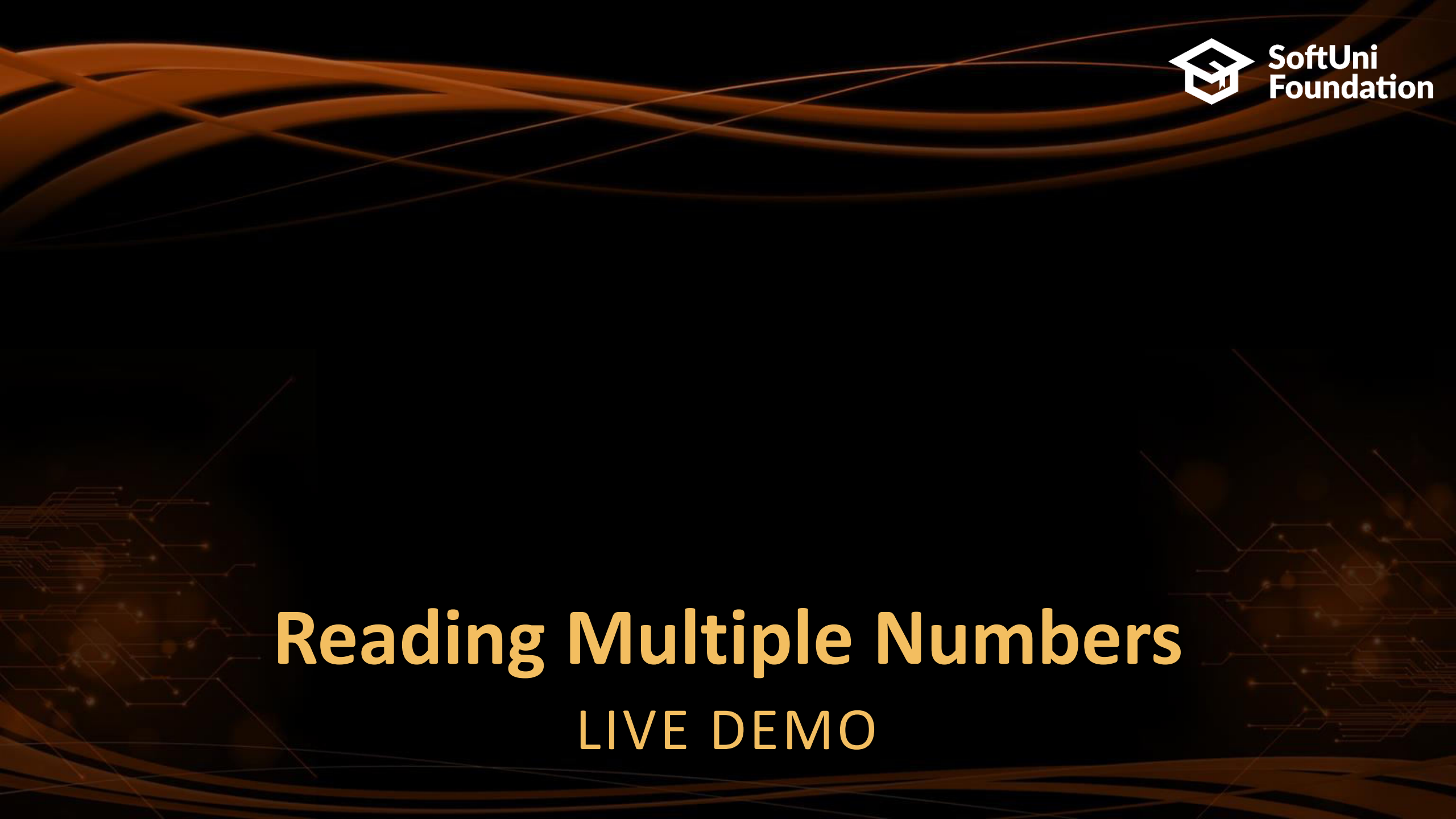
- Things (classes) that either read or write data piece by piece
- Example: we've been using **`cout`** throughout this lecture
  - Writes data to the console (standard output)
- **`cout`** has a counterpart – **`cin`**
  - Reads data from the console (standard input)
  - **`cin`** uses the **`>>`** operator to read
  - **`cout`** uses the **`<<`** operator to write

```
#include<iostream>
using namespace std;
int main() {
    int a, b;
    cin >> a >> b;
    cout << a + b << endl;
    return 0;
}
```

Reading (Multiple) Numeric Values with `cin`

SoftUni Foundation

- Expects console data to mostly match variable type literals
- Expects input data to be separated by one or more spaces/lines
- For integer types, a sequence of digits is expected
  - Can have sign, can NOT have type suffix (i.e. NO `L`, `ULL`, etc.)
- For floating-point types:
  - Can be an integer
  - Can be a sequence of digits separated by "`.`"
  - Can be exponential notation (e.g. `0.42e+2` will be read as `42`)


SoftUni
Foundation
Reading Multiple Numbers
LIVE DEMO

Summary

SoftUni Foundation

- C++: fast, statically-typed, imperative, multi-paradigm language
- Integer types – `int` with modifiers for size and sign
  - Size: `long`, `short`; Sign: `signed`, `unsigned`
  - Size in bytes depends on system
- Floating point types – float and double, following IEEE754
- `if-else` and `switch-case` allow code branching
- `for`, `while`, `do-while` allow repeat-execution of code
- Use `cin >>` / `cout <<` for reading from/writing to console

C++ Basic Syntax

SoftUni Foundation

Questions?

https://softuni.bg/courses/