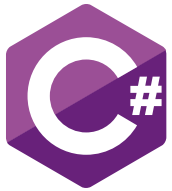
Built-in types and namespaces

Agenda

- The bool type
- Integers
- Floating-point types
- Binary floating-point types
- Decimal floating-point types
- The char type
- The string type
- default operator and literal
- Namespaces

boolean



true/false

bool

True or false

Logical operators can be combined with boolean variables to create Boolean expressions

Boolean expression

```
bool condition = (condition1 || condition2) && condition3;
```

OR conditional logical operator

```
bool condition = condition1 || condition2;
// If condition1 evaluates to true, condition2 is not evaluated
```

AND conditional logical operator

```
bool condition = condition1 && condition2;
// If condition1 evaluates to false, condition2 is not evaluated
```

About bool type

Boolean type is a value type (structure)

`bool` is the same as `Boolean` (structure)

Important methods : `Equals, Parse, ToString, TryParse`

Conditional ?: operator

Evaluates a boolean expression

Returns a result that depends on this evaluation

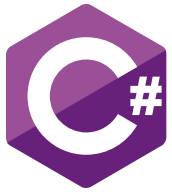
Conditional ?: operator

```
bool condition = 8 > 15;
string message = condition ? "Condition is true" : "Condition is false";
Console.WriteLine(message); // result : Condition is false
```

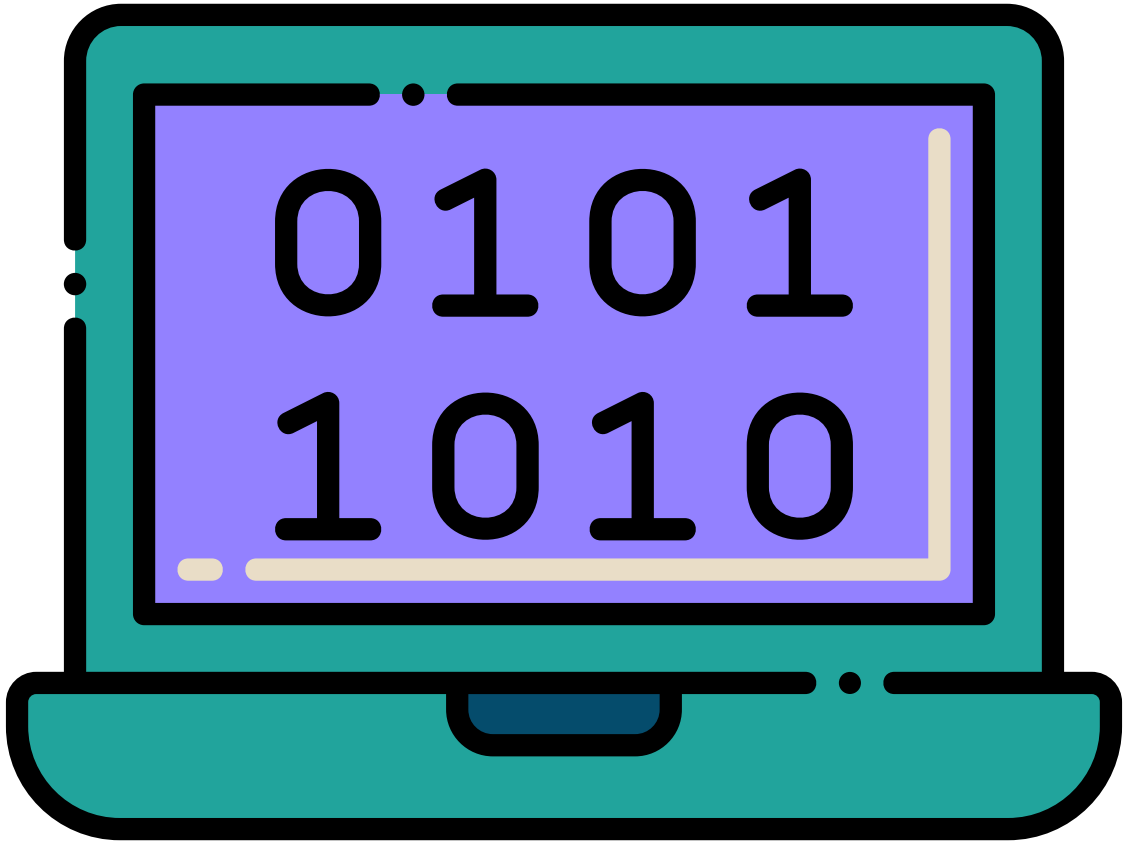
Demo

Create some booleans, use conditional operator

Integers



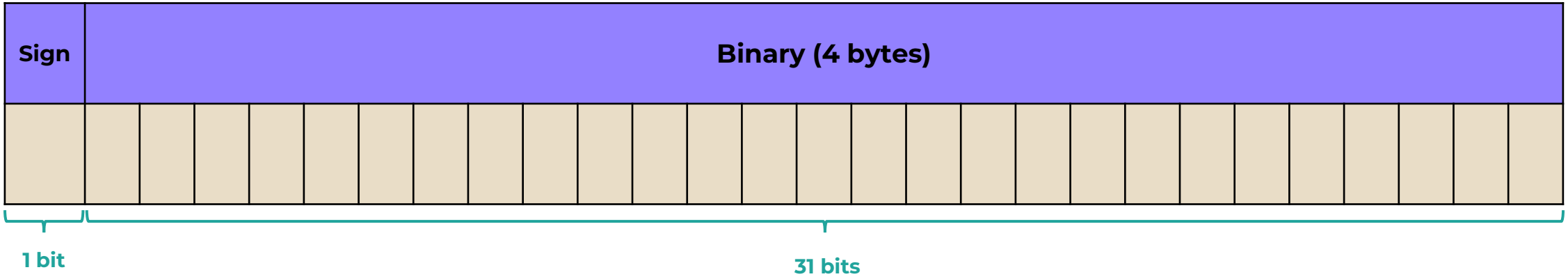

0123456789


0101
1010

int type variable creation

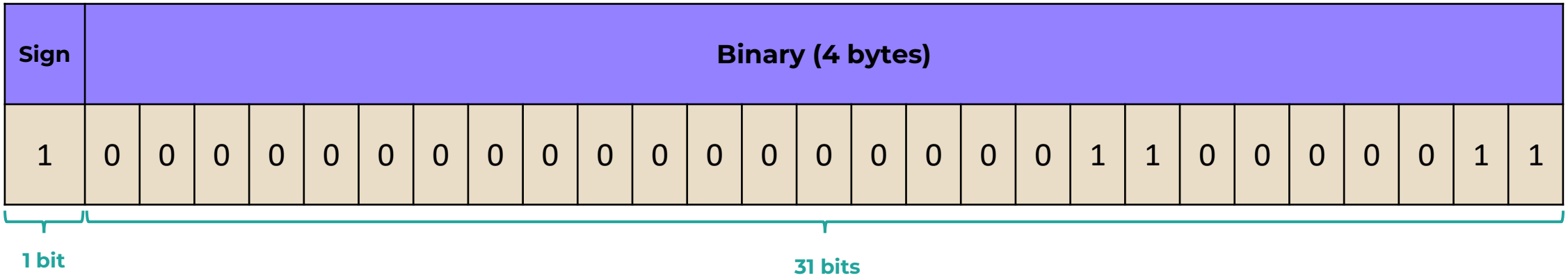
```
int myInteger = -131;
```

int (32 bits)

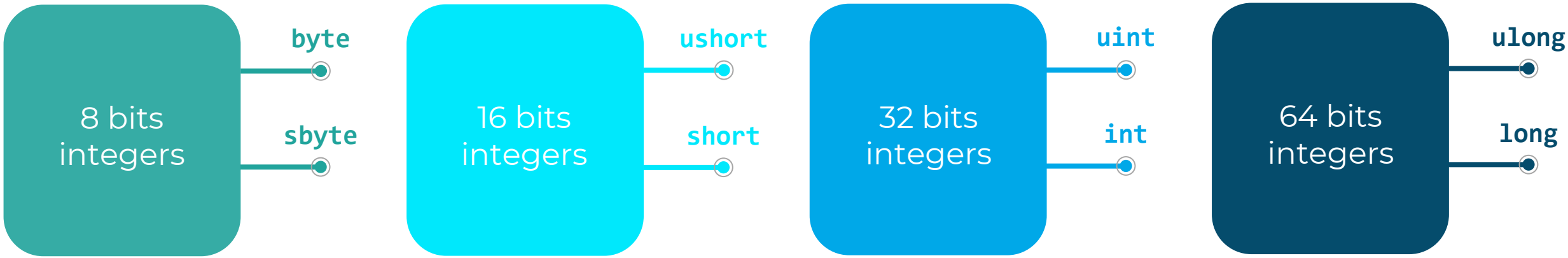


int (32 bits)

-131 —Binary→ 10000011 (absolute value)



Integers


8 bits integers
byte
sbyte
16 bits integers
ushort
short
32 bits integers
uint
int
64 bits integers
ulong
long

sbyte

Signed 8-bit integer

From -128 to 127

byte

Unsigned 8-bit integer

From 0 to 255

short

Signed 16-bit integer

From -32 768 to 32 767

ushort

Unsigned 16-bit integer

From 0 to 65 535

int

Signed 32-bit integer

From -2 147 483 648 to 2 147 483 647

uint

Unsigned 32-bit integer

From 0 to 4 294 967 295

long

Signed 64-bit integer

From -9 223 372 036 854 775 808 to
9 223 372 036 854 775 807

ulong

Unsigned 64-bit integer

From 0 to 18 446 744 073 709 551 615

MinValue and MaxValue

Represents the smallest and the largest value possible for an integer type

About integer types

Integer types are reference types (structure)

`int` is the same as `Int32` (structure)

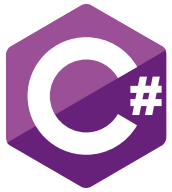
Important fields : `MinValue, MaxValue`

Important methods : `CompareTo, Parse, ToString, TryParse`

Demo

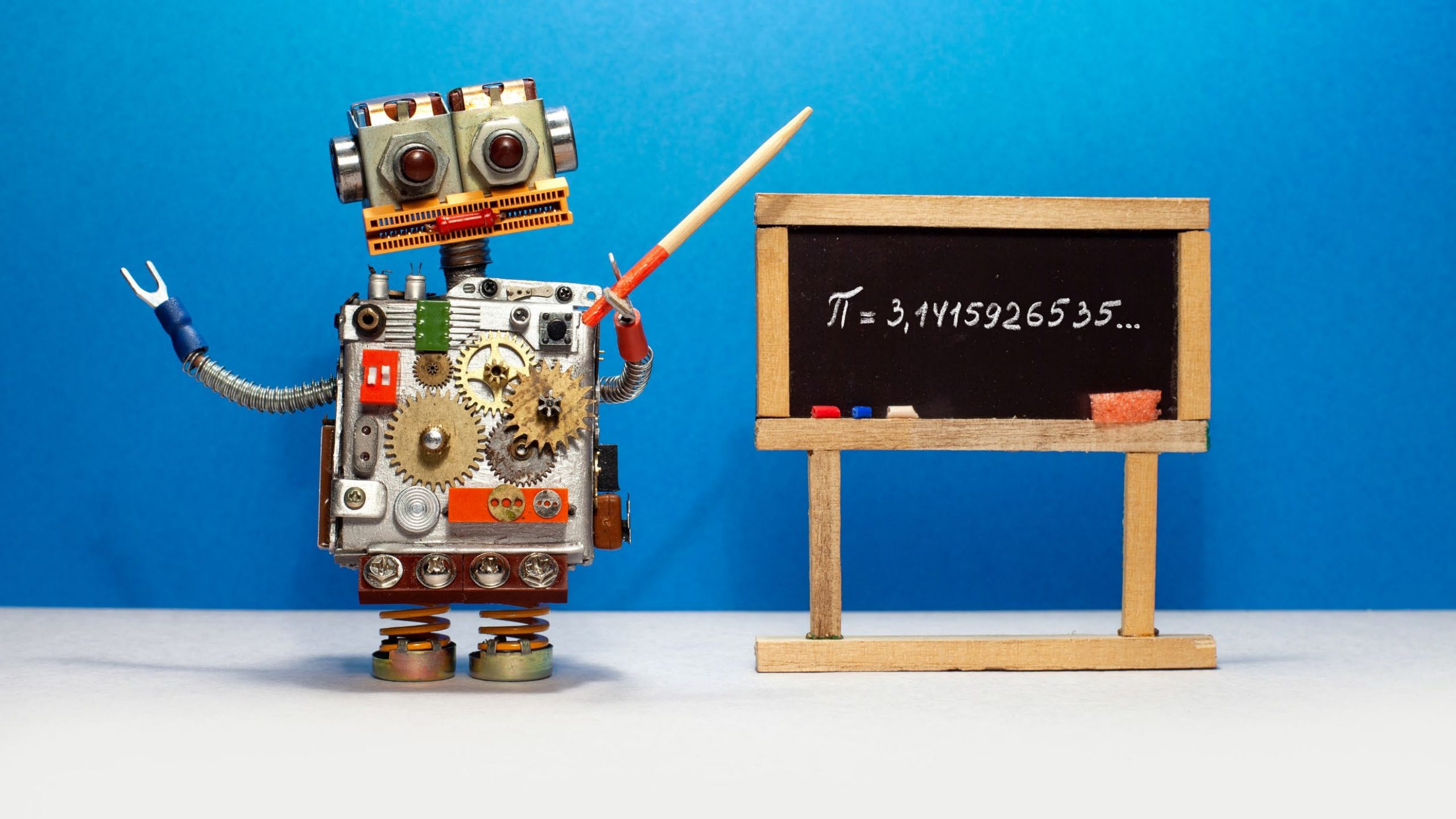
Integers types

Floating-point types



1/4

π


π = 3,1415926535...

3.14159265358979323846264338327950288419716939937
51058209749445923078164062862089986280348253421170
67982148086513282306647093844609550582231725359408
12848111745028410270193852110555964462294895493038
19644288109756659334461284756482337867831652712019
09145648566923460348610454326648213393607260249141
27372458700660631558817488152092096282925409171536
43678925903600113305305488204665213841469519

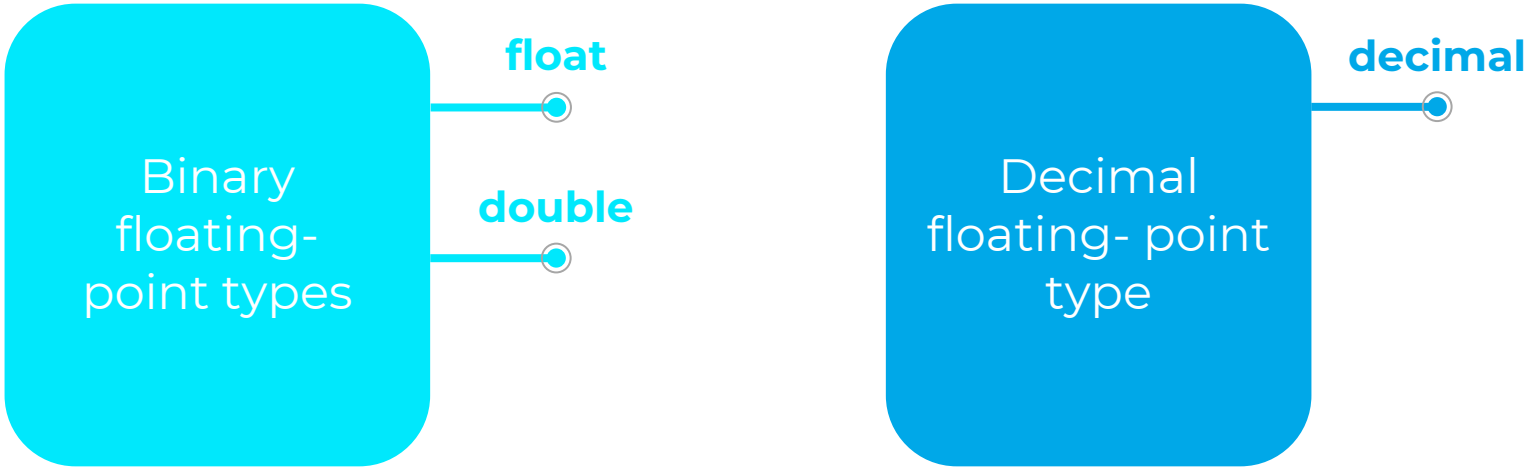
Floating point types

There is not enough RAM to represent irrational numbers

There is not enough RAM to represent some very small or very large numbers

Floating point types can represent these numbers with a lack of precision

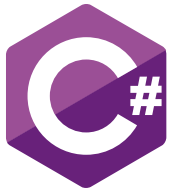
Floating point types



Demo

Binary floating-point types are weird

Binary floating-point types



Floating point numbers

More complex to store in memory than integers

Stored in binary

float

Binary floating point numeric type

32 bits - Range : $\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$

Precision of 6 to 9 digits

double

Binary floating-point numeric type

64 bits - Range : $\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$

Precision of 15 to 17 digits

Suffix

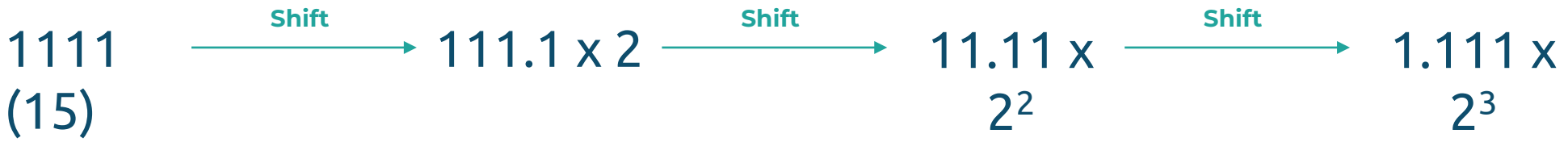
The literal without suffix or with the d or D suffix is of type double
The literal with the f or F suffix is of type float

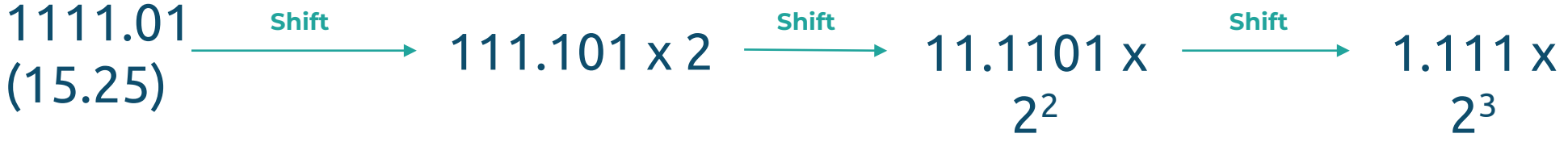
Binary floating-point types

Constituted of 3 parts:

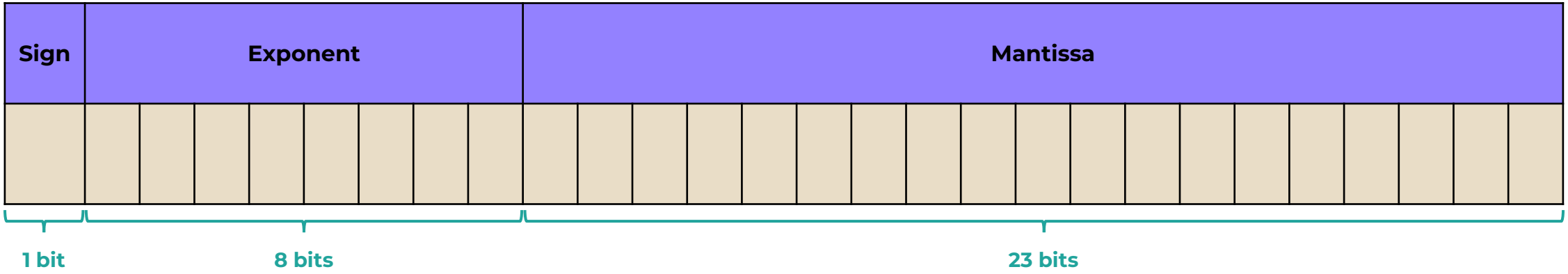
- Sign
- Exponent
- Mantissa

Shifting


1111
(15)
Shift
111.1 x 2
Shift
11.11 x 2^2
Shift
1.111 x 2^3


1111.01
(15.25)
Shift
111.101 x 2
Shift
11.1101 x 2^2
Shift
1.111 x 2^3

float (32 bits)



float (32 bits)

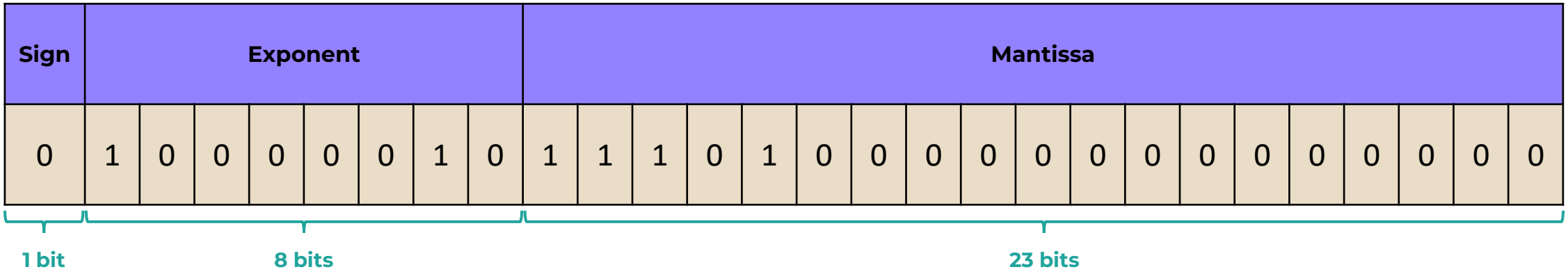
15.25 —Binary ($2^3+2^2+2^1+2^0+2^{-2}$)→ 1111.01 —Shift→ 1.11101×2^3

f

Sign = 0

Exponent = 3 + 127 = 130 —Binary (2^8+2^1)→ 10000010

Mantissa = 11101



Lack of precision of binary floating-point types

```
float f = 0.25f;
Console.WriteLine($"{f:G9}");
// result : 0.25
```

0.25f vs 0.2f

$0.25f \xrightarrow[2^{-2}]{\text{Binary}} 0.01 \xrightarrow{\text{Shift}} 1 \times 2^{-2}$

$0.2f \xrightarrow[2^{-3} + 2^{-4} + 2^{-7} + 2^{-8} + 2^{-11} + 2^{-12} + \ldots]{\text{Binary}} 0.00110011001100 \ldots \xrightarrow{\text{Shift}} 1.10011001100\ldots \times 2^{-3}$

About float types

Float types are reference types (structure)

`float` is the same as `Single`

`double` is the same as `Double`

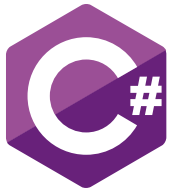
Important fields : `MinValue, MaxValue, Epsilon, NaN, PositiveInfinity, NegativeInfinity`

Important methods : `CompareTo, Parse, ToString, TryParse`

Demo

Use binary floating-point types

decimal



Decimal floating-point type

Decimal floating-point types is stored in decimal (base 10)

decimal

Decimal floating-point numeric type

128 bits - Range : $\pm 1.0 \times 10^{-28}$ to $\pm 7.9228 \times 10^{28}$

Precision of 28 to 29 digits

Decimal floating-point types

Constituted of 3 parts:

- Sign (first bit of 1 x 32 bits integer)
- Exponent (some bits of 1 x 32 bits integer)
- Mantissa (3x32 bits integers)

Floating point types

The literal with the m or M suffix is of type decimal

Precision

- `decimal` has a larger precision than `float` and `double`
- `decimal` has a smaller range than `float` and `double`
- Some operations can be inaccurate

Lack of precision of decimal floating-point type

```
decimal d = 1m/3m;
Console.WriteLine($"{d:G10}");
// result : 0.3333333333
```

About decimal type

Decimal float type is value type (structure)

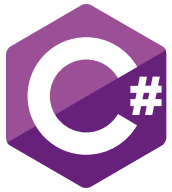
`decimal` is the same as `Decimal`

Important methods : `CompareTo, Parse, TryParse, Add, Substract, Divide, Multiply, Round, Remainder`

Demo

Use decimal floating-point type

char



char

Unicode UTF-16 character

Stored on 16 bits

Is a value type

About char type

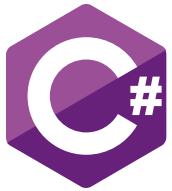
`char` is the same as `Char` (structure)

Important methods : `CompareTo, Equals, IsLetter, IsNumber, IsUpper, IsWhitespace, Parse, ToUpper, TryParse`

Demo

Create and use char variables

string



string

Sequence of Unicode characters

Immutable

It’s a reference type

Empty string

```
string.Empty
```

Format string

The content of the formatted string is determined at runtime

String interpolation vs `String.Format`

Length of a string

String is a sequence

The **Length** property is the number of char elements in the sequence

About string type

`string` is the same as `String` (class)

Important methods : `Compare`, `Concat`, `EndsWith`, `Format`, `IndexOf`, `Split`, `StartsWith`, `Substring`, `Trim`

StringBuilder

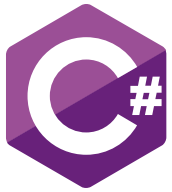
Too many operations on string variables can affect performance

Use `StringBuilder` class for fast string creation

Demo

Create and use string variables

Default operator and literal



Default operator

Produces the default value of a type

Default values of some value types

```
Console.WriteLine($"int default value : {default(int)}");
// output -> int default value : 0
Console.WriteLine($"float default value : {default(float)}");
// output -> float default value : 0
Console.WriteLine($"double default value : {default(double)}");
// output -> double default value : 0
Console.WriteLine($"decimal default value : {default(decimal)}");
// output -> decimal default value : 0
Console.WriteLine($"DateTime default value : {default(DateTime)}");
// output -> DateTime default value: 1 / 1 / 0001 12:00:00 AM
```

Default values of some reference types

```
Console.WriteLine($"string default value is null ? {default(string) == null}");
// output -> string default value is null ? True
Console.WriteLine($"object default value is null : {default(object) == null}");
// output -> object default value is null : True
Console.WriteLine($"List<int> default value is null : {default(List<int>) == null}");
// output -> List<int> default value is null : True
```

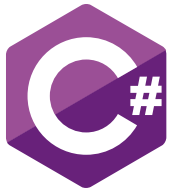
Default literal

To produce the default value when the type can be inferred (assignment, method calls…)

Default literal

```
int i = default;
```

Namespaces



Compilation units

Program = one or more C# source files (compilation units)

Compilation units can depend on each other

Why namespaces ?

Namespace structuring allows to organize types declared in compilation units and to control their scope

.NET types are organized in namespaces

The System namespace

```
System.Console.WriteLine("Hello World!");
```

Namespace declaration

```
namespace MyNamespace
{
    // namespace types
}
```

Namespaces and files

A compilation unit can have a namespace declaration, or belong to the global namespace

The types of a namespace can be defined in different compilation units (files)

Inside Namespaces

It can contain a type or another namespace

Namespaces

Can be a top-level declaration, in the global namespace

Or can be declared inside of another namespace

Within its containing namespace, the name of a namespace must be unique

Namespace nesting

```
namespace Namespace1
{
    namespace Namespace2
    {
        class Type1 { }

        struct Type2 { }
    }
}
```

Namespace nesting

```
namespace Namespace1.Namespace2
{
    class Type1 { }

    struct Type2 { }
}
```

using directive

- Imports the types of a namespace
- Types can be qualified directly rather than through qualified names
- Possible to define a namespace alias and use it in the code
- using directives must appear before any declaration

Using directive

```
using System;
```

Namespaces

System

System.Text/ System.Text.Json

System.IO

System.Net

...

Demo

Import a namespace

Create namespaces

Use namespace aliases

Summary

C# provides many built-in types

The bool type is a value type and represents a Boolean variable

C# provides signed and unsigned integer types. These types are value types that can be 8, 16, 32 or 64 bits long

C# provides binary and decimal floating-point types. These are value types

They can represent irrational numbers with a lack of precision

Binary floating-point types have a larger range of values but less precision. They can be 32 or 64 bits long

Summary

A char stores a 16 bits UTF-16 character. It’s a value type

A string is a sequence of UTF-16 characters. It’s a reference type. A string variable is immutable

Each type variable has a default value (0 for int, false for bool…)

The default operator and literal produces the default value

.NET and custom types are organized in namespaces

To use a type in a C# file you must import its namespace with the using directive