

Learning the C# Syntax



Gill Cleeren

CTO Xebia Microsoft Services Belgium

@gillcleeren

Overview



Understanding the essential C# building blocks

Working with built-in types

C# operators

Using date and time

Converting between types

Implicit typing

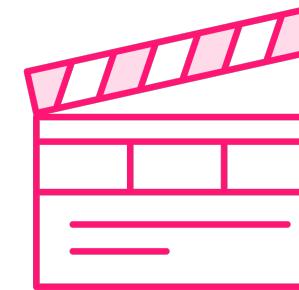


Understanding the Essential C# Building Blocks

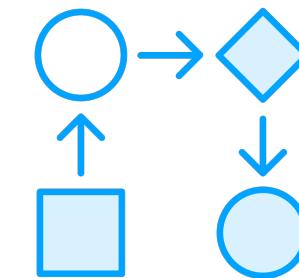


C# Statements

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



Actions



Flow of the program



End with semicolon



C# Statements

```
Console.WriteLine("Hello, World!")
```

```
;
```



C# Identifiers

```
string input = Console.ReadLine();  
string 2_input = Console.ReadLine();
```

Identifiers start with a letter or underscore and can contain letters, digits and underscores



C# Comments

Single line comments

Program.cs

```
//The next line will read a value from the console  
string input = Console.ReadLine();
```



C# Comments

Multiline comments

Program.cs

```
/*
    In the next block of code,
    we will read a value from the console
*/
string input = Console.ReadLine();
```

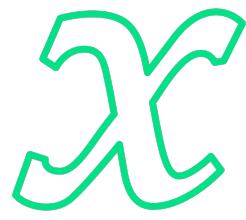


C# Keywords

int	ref
in	return
class	lock
using	long
while	string
new	struct
null	const
if	enum
case	void



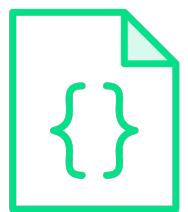
C# Variables



A variable holds a value



Integer, string, date...

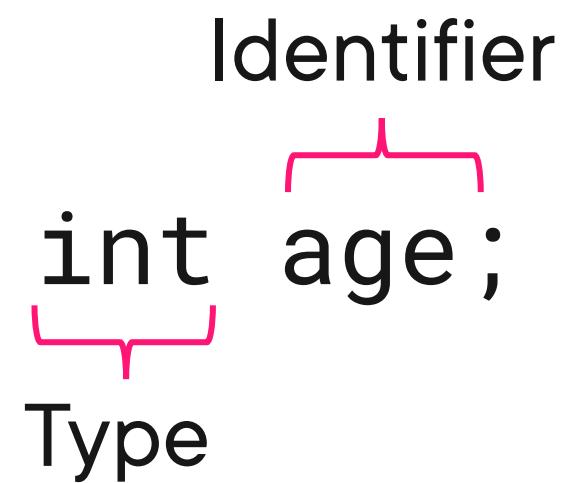


Created in a declaration statement



Creating an Integer Variable

Identifier
`int age;`
Type



Creating an Integer Variable

```
int age;  
int Age;
```



Creating an Integer Variable

```
int ageOfEmployee;
```



Creating an Integer Variable

```
int age;
```

Assignment operator

```
age = 25;
```



Value



Using the Variable

```
Console.WriteLine(age);
```



Demo



Using the essential C# building blocks



Working with Built-in Types





C# is a strongly typed language

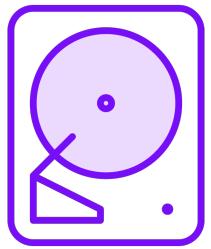
Every variable has a type

Used to store information

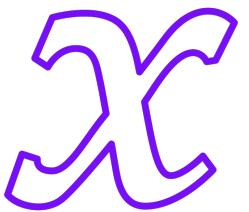
**Expressions will return a value of a
specified type**



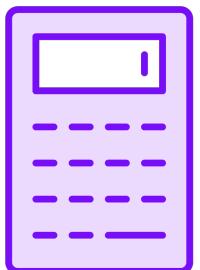
Using Data Types in C#



Size and location in memory



Data range



Supported operations



Data Types in C#

Predefined types

User-defined types



Predefined Data Types in C#

bool

int

float

double

decimal

char



More Predefined Data Types

byte (sbyte)

short (ushort)

object

string



Creating an Integer Value

```
int a = 2;  
int b = a + 3;
```

Expression



Creating a Boolean Value

```
bool c = true;
```



C# Types Lead to Type Safety

```
int c = 3;  
c = true;
```



Demo



Working with primitive types



Using a const Value

```
const decimal interestRate = 0.07m;
```



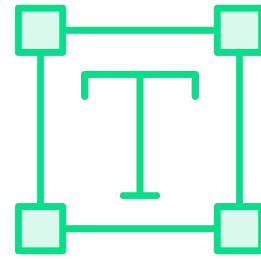
Demo



Using constant values



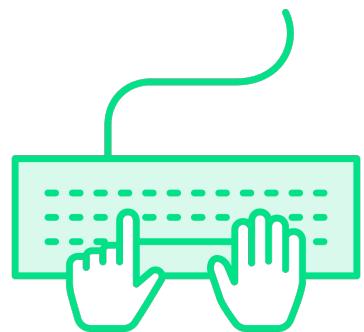
Understanding Strings



Contains text

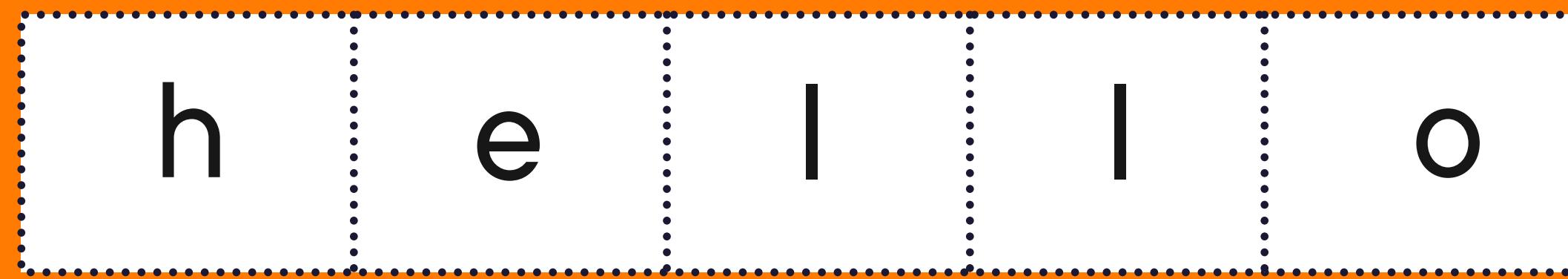


Stored as list of char objects



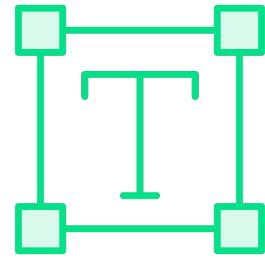
string type





h e l l o

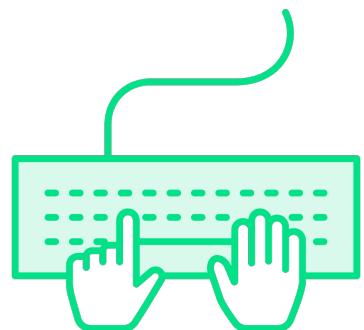
Understanding Strings



Contains text



Stored as list of char objects



string type



```
string s1 = "Hello world";
string s2 = string.Empty;
```

Creating Basic Strings



Demo



Creating strings





We'll learn
a lot more about strings
in an upcoming module.



C# Operators



```
int a, b, c;
```

```
a = 3;
```

```
b = 10;
```

```
c = a++;
```

```
b = a + b * c;
```

Expressions in C#

Arithmetic expressions



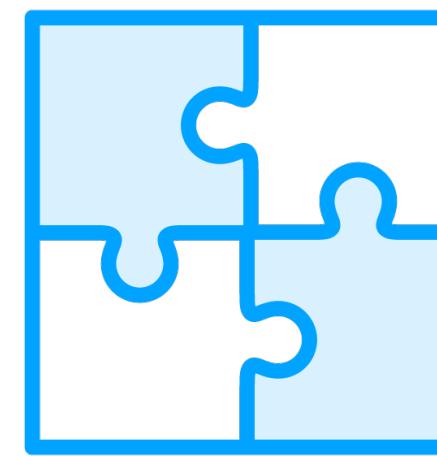
Operators in C#



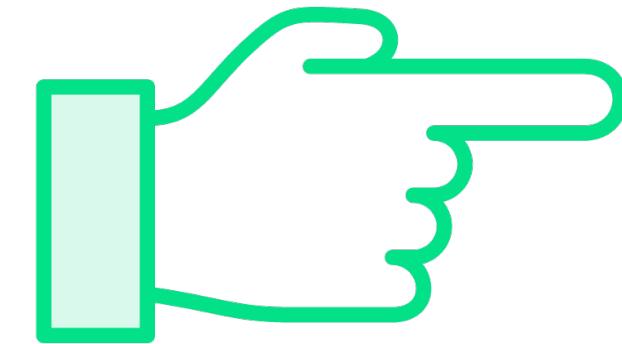
Arithmetic
operators



Equality
operators



Logical
operators



Assignment
operators



Using Arithmetic Operators

Operator	Example
+	$a + b$
-	$a - 3$
*	$a * b * c$
/	$a / 10$
++	$a++$
--	$b--$



Compound Assignment Operators

```
int month = 3;
```

```
month = month + 1;
```

```
month += 1;
```



Operators Depend on the Type

```
string result1 = "a" + "b";
```

```
string result2 = "a" * "b";
```



Demo



Using operators in C#

Default values for types in C#



```
int int.MaxValue = int.MaxValue;  
int int.MinValue = int.MinValue;  
double double.MaxValue = double.MaxValue;
```

Members on Primitive Types



```
char myChar = 'a';
bool isWhiteSpace = char.IsWhiteSpace(myChar);
bool isDigit = char.IsDigit(myChar);
bool isPunctuation = char.IsPunctuation(myChar);
```

Members of char Type



Demo



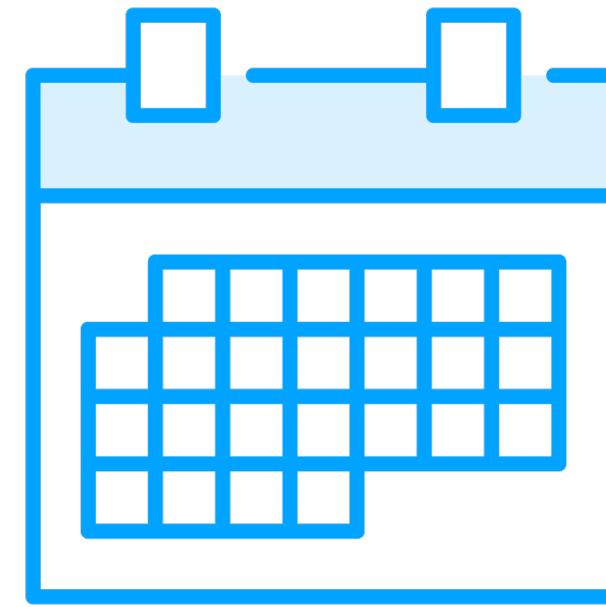
Working with members of int and char



Using Date and Time in C#



Working with Dates



DateTime



TimeSpan



```
DateTime employeeStartDate = new DateTime(2025, 03, 28);  
DateTime today = DateTime.Today;  
DateTime twoDaysLater = someDateTime.AddDays(2);  
DayOfWeek day = someDateTime.DayOfWeek;  
bool isDST = someDateTime.IsDaylightSavingTime();
```

Working with **DateTime** and **DateOnly**



Demo



Working with Date**T**ime



Converting Between Types



This Doesn't Work...

```
int a = 3;  
a = "Hello world";
```



Changing between Types

Implicit conversion

Casting
Explicit conversion

Helpers



```
int a = 123456789;  
long l = a;
```

Using an Implicit Cast



```
double d = 123456789.0;  
int a = (int) d;
```

Performing an Explicit Cast



Demo



Converting between types



We'll take a look at parsing
in a later module.



Implicit Typing



So Far, We've Used Explicit Typing

Explicit typing

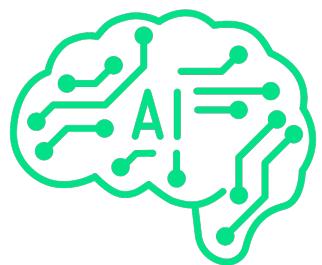
```
int a = 123;  
bool b = true;  
double d = 11.0;
```

Implicit typing

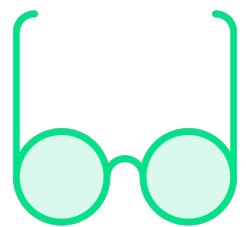
```
var a = 123; //a will be an integer  
var b = true; //b will be a boolean  
var d = 11.0; //d will be a double
```



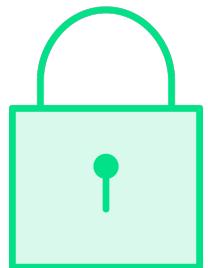
Understanding Implicit Typing



Type is inferred



Not always as readable



Sometimes required (using LINQ)



This Won't Work...

```
var employeeAge;
```



Demo



Using var



Summary



C# is a strongly typed language

Contains built-in data types

Conversion between types is supported



Up Next:

Using decisions and iterations in C#

