CSC101

Introduction to C#

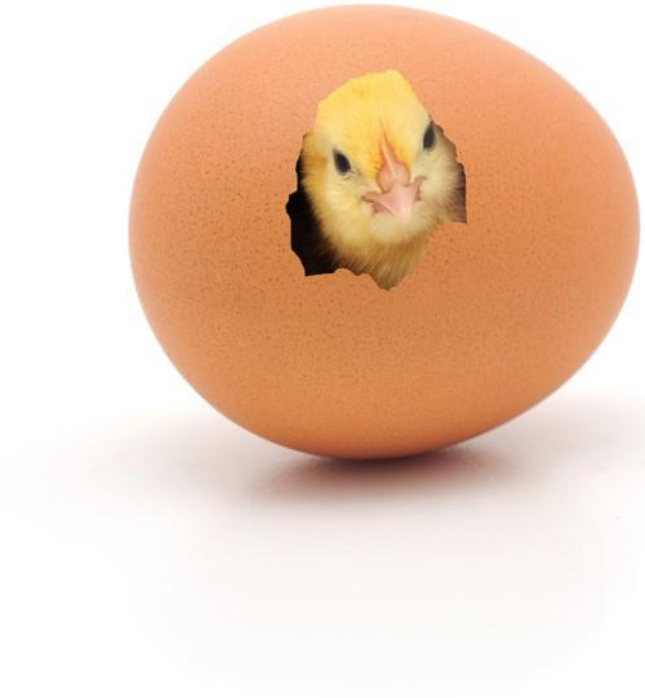
Download class materials from
university.xamarin.com

Microsoft

Xamarin University

Objectives

1. Know the role of C# and Xamarin
2. Set up and run your first program
3. Learn the fundamentals of C#

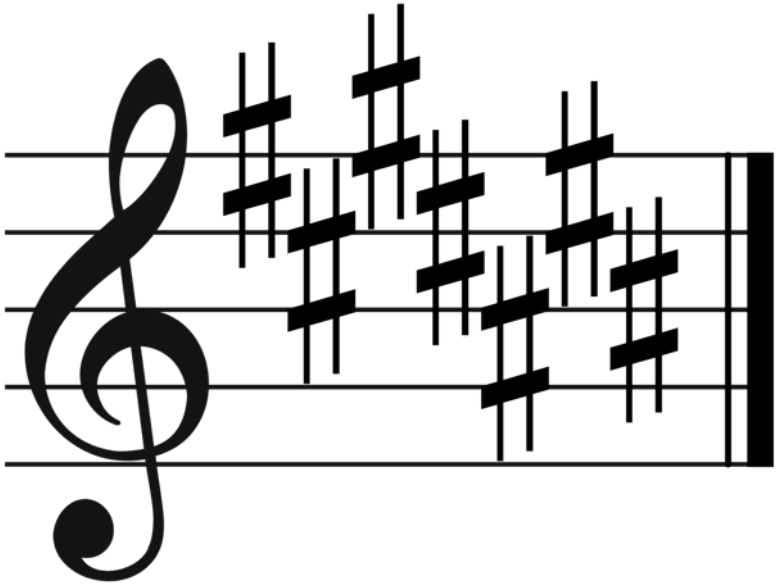


Know the role of C# and Xamarin

Xamarin University

Tasks

1. Define some common terms
2. Review the history of C#
3. Learn how C# fits into mobile programming



Programmers

- People who write software call themselves *programmers* or *developers*
  - (…or software engineers, or sometimes hackers)

What is a programming language?

- A *programming language* is used to give precise instructions, called a program, to a computer

This is a valid C# program.

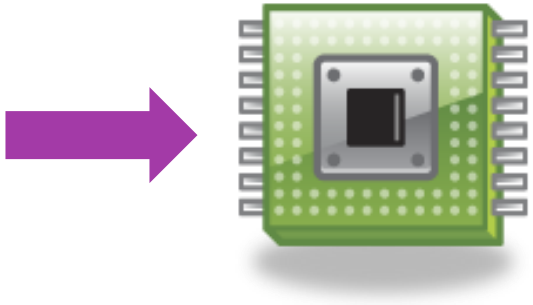
Can you guess what this does?

```
int x = 3;
int y = 5;
int max;

if (x > y) {
  max = x;
}
else {
  max = y;
}
```

What is a CPU?

- A Central Processing Unit (CPU) executes your program's instructions

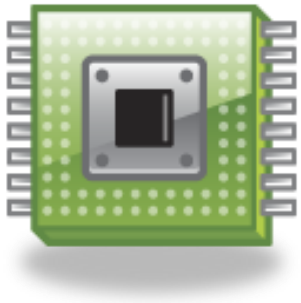
"multiply these two numbers"
"tell me which is larger"
"move this value over there"
"do this 10 times"



It's All A Big Analogy

- CPUs understand high and low voltage, represented as values of 0s and 1s

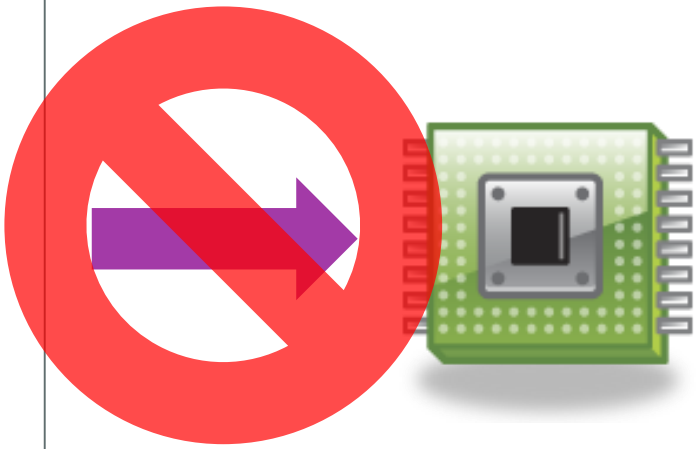
```
1011010011010101
1111001001101000
1010101011000100
1100001011111000
1010100011011111
1101000000101101
```



CPU instruction format

- CPUs only understand 0s and 1s, not typical programming languages

```
int x = 3;
int y = 5;
int max;

if (x > y) {
  max = x;
}
else {
  max = y;
}
```

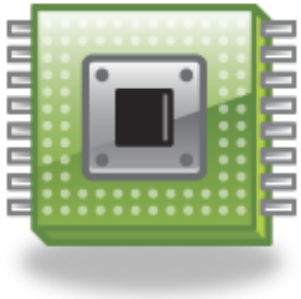


What is a compiler?

- A *compiler* converts your program to a format your CPU can execute

```
int x = 3;
int y = 5;
int max;

if (x > y) {
  max = x;
}
else {
  max = y;
}
```


```
10110100
11010101
11110010
01101000
10101010
11000100
11000010
11111000
10101000
11011111
```



Creating programs

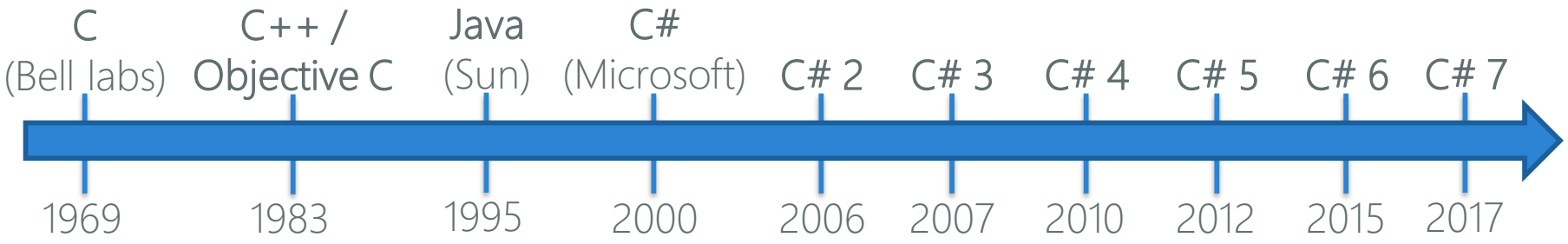
- You type your program into an Integrated Development Environment (IDE)

Visual Studio for Windows

Visual Studio for Mac

A very quick history of C#

- C# evolved out of work done on earlier languages such as C, C++, and Java



Mobile programming

- Mobile Devices are programmed using different programming languages

Android uses Java

iOS uses Objective-C or Swift

Windows uses C#

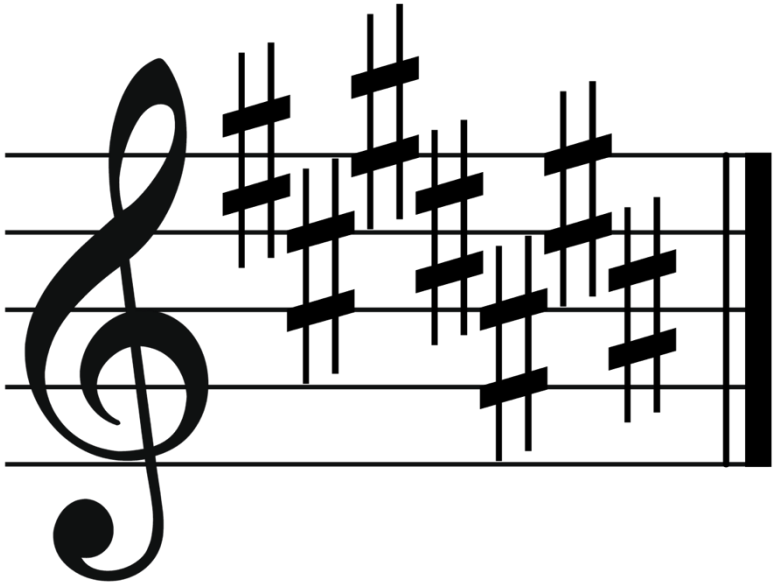
To create the same application on all three platforms, requires writing the same program *three times* in *three different* languages!

Xamarin and mobile programming

- Xamarin tools allow all four platforms to be programmed using C#

Summary

1. Define some common terms
2. Review the history of C#
3. Learn how C# fits into mobile programming



Set up and run your first program

Xamarin University

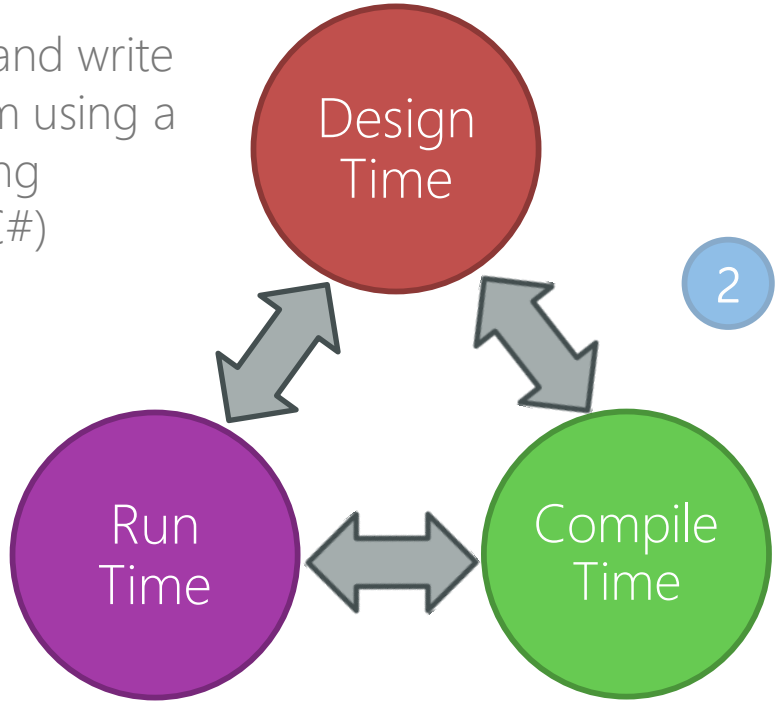
Tasks

1. Set up your programming environment
2. Create "Hello World" program and run it

Cycles of developing a program

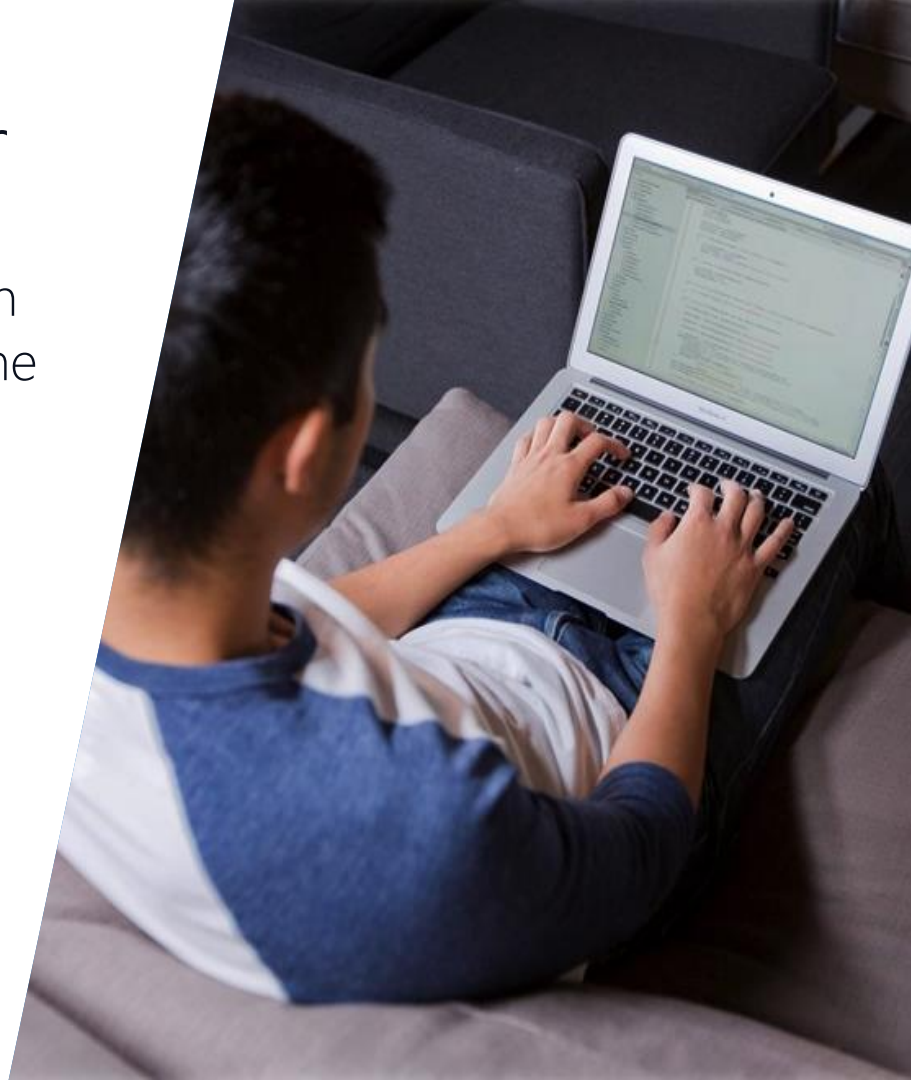
- To create programs we continually move between three different phases

1. We design and write our program using a programming language (C#)
2. We compile our program in the IDE to something the computer understands
3. We run our program on a device to test it and make sure it does what it should



Keyboard and monitor

- Your program can display information on the monitor and get input from the keyboard
- Both of these operations are done through the console



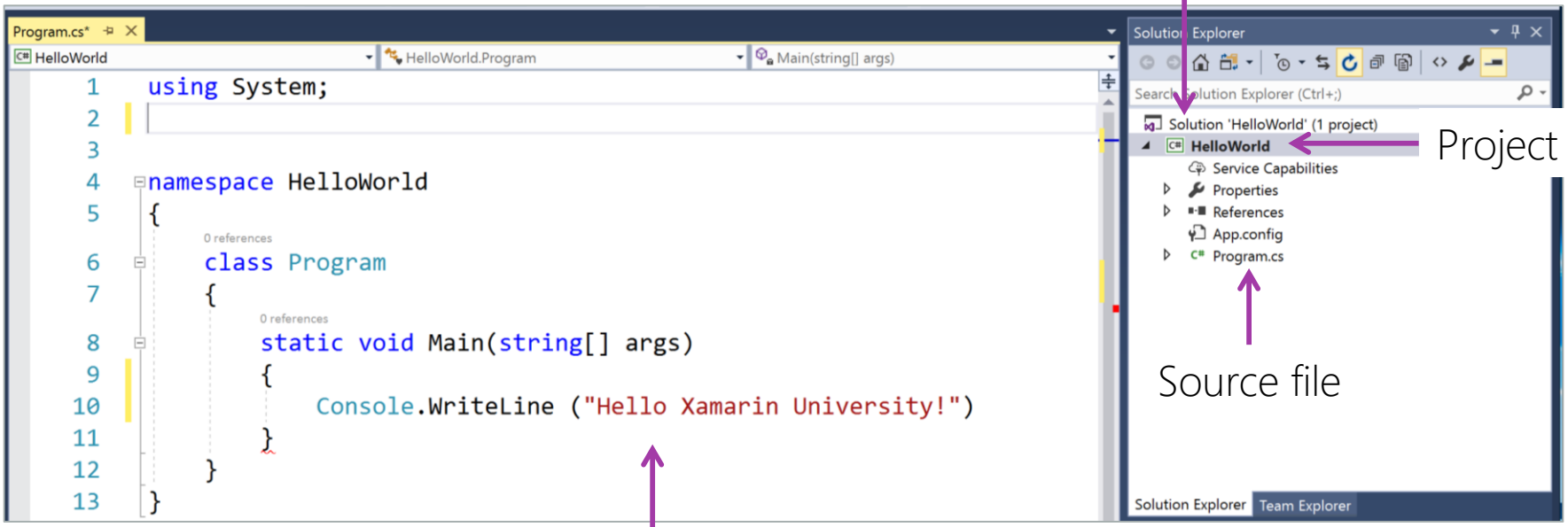
Console.WriteLine

- C# can use **`Console.WriteLine`** to write one line of text to the monitor

```
Console.WriteLine("Hello, C#!");
```

Any text given to **`Console`** will be written to the monitor's screen

Parts of your program in the IDE



Group Exercise

Hello World

Xamarin University

Summary

1. Set up your programming environment
2. Create "Hello World" program and run it

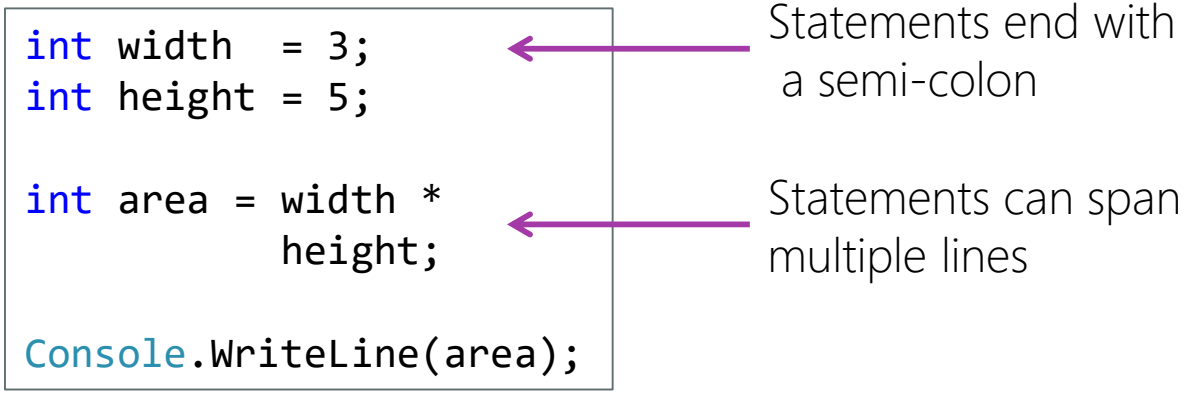
Learn the fundamentals of C#

Tasks

1. What is a C# Statement?
2. Understand types
3. Create variables and constants
4. Strings and white space

Statements

- C# programs are a sequence of *statements* where each statement is a complete programming instruction



Adding notes to your program

- *Comments* are notes that are ignored by the compiler and come in two styles: single line and blocks

single-line comments start with two slashes and are ignored until end of line

block comments start with /* and are ignored until the closing */

```
int foo = 25;    // comments are ignored by C#

/* this will block off the following,
   including the nested comment
int baz = 33;    // the nested comment

which is a nice feature */
```

comments can be nested

What is a type?

- A type tells you the *kind* of object you have

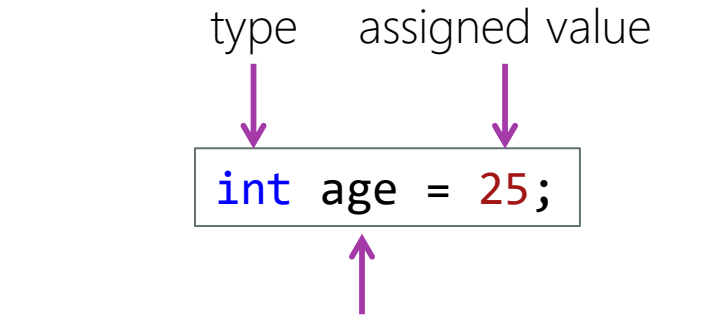
`int` holds numbers → `int myAge = 25;`

`string` holds text → `string myName = "Jesse Liberty";`

`Button` is a UI widget → `Button okButton;`

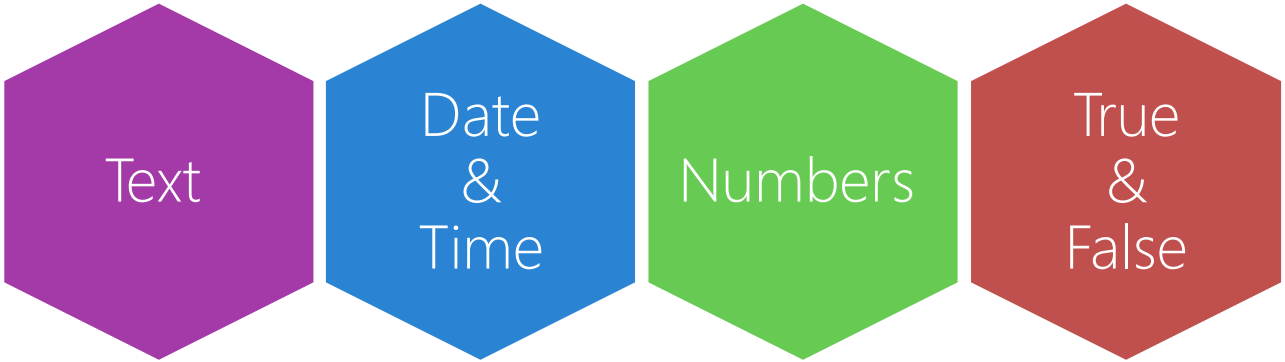
Types and variables

- A variable is a named container for a value with a type



Built-in types

- C# supports a variety of built-in types which you can use to represent data in your program



not a complete list

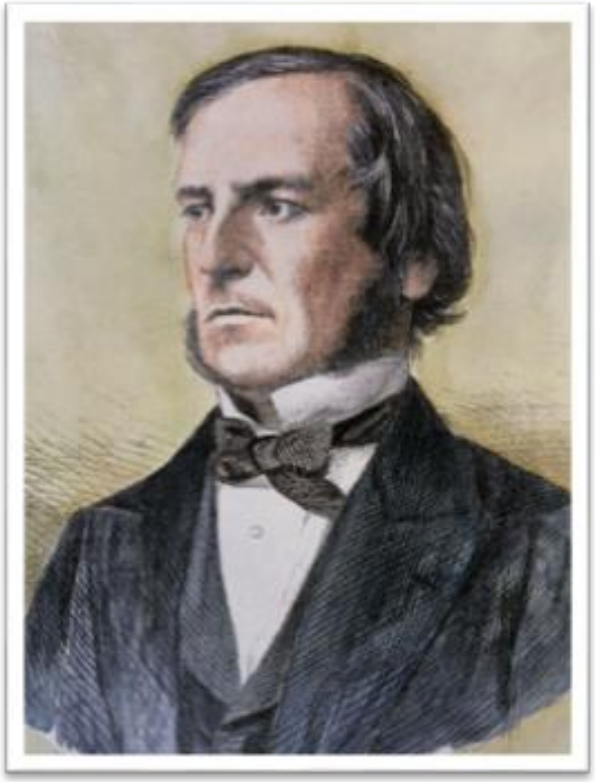
Numeric types

Type	What it holds	For Example
`int`	positive or negative whole numbers	102402
`double`	positive or negative fractional numbers	3.141592653589793
`float`	positive or negative smaller fractional numbers	3.1415926
`short`	positive or negative smaller whole numbers	-4096
`ushort`	positive small whole numbers	40960
`long`	positive or negative large whole numbers	102402454

True / false values

- True / false values can be represented with the **bool** type

```
bool amHappy = true;

bool amSad = false;
```



George Boole, inventor of Boolean Algebra, mathematician, philosopher. 1815-1864

Working with strings

- Strings are a type which hold a series of characters

```
string name = "Jesse";
string favoriteColor = "blue";

favoriteColor = "red";
```


double-quotes are used to surround string literals

Combining strings

- Strings can be combined together with "+" to create a brand new string

```
string first = "hello";
string second = "world";

string combined = first + " " + second;  // "hello world"
```

new string contains the value "hello world"

Working with single characters

- Can define single character values with `char` type

```
char firstInitial = 'J';

char firstLetter = 'a';
```

values are surrounded in single quotes and can be upper or lower case

Creating special characters

- Use backslash "\" to indicate an escape character – this changes the meaning of the subsequent character and can be used to create non-printable values

```
char tab = '\t';
string quote = "\"";
char newLine = '\n';
string carriageReturn = "\r";
```

Working with variables

- Variables hold values that can be used and updated as the program runs

```
int age = 45;
int thisYear = DateTime.Now.Year;
int yearBorn = thisYear - age;
```

variables provide a "holding" place for values we use in our programs such as calculations

Displaying variables with Console

- **`Console.WriteLine`** can output text to the display and will automatically convert non-text variables into text

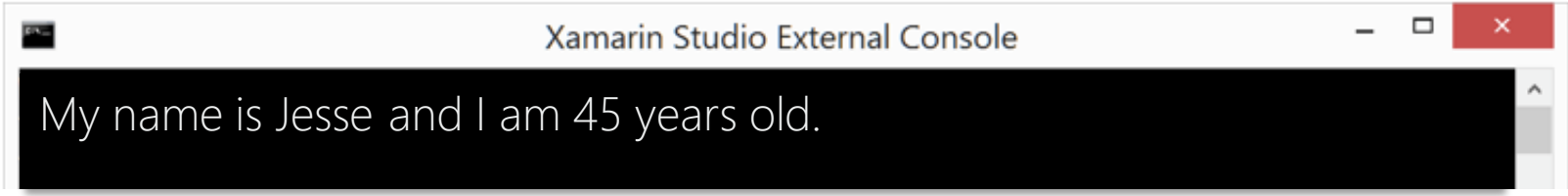
```
int age = 45;
int thisYear = DateTime.Now.Year;
int yearBorn = thisYear - age;

Console.WriteLine("You were born in " + yearBorn);
```

this numeric value is being converted to a text value so it can be combined to the string and then shown on the display

Complex Console output

- **`Console.WriteLine`** also supports *replacement values* which can be added into the output, the values are passed in a comma-separated list along with the text to display

```
string name = "Jesse";
int age = 45;
Console.WriteLine("My name is {0} and I am {1} years old.",
                  name, age);
```



Type safety

- Compiler will generate an error when you attempt to assign a value which is inappropriate for the type – this helps avoid errors/bugs in your programs

```
int age = 45;          // ok

int myAge = "Jesse"; // error!
```

error – attempting to assign a textual value (**string**) to a numeric container (**int**)

This is sometimes referred to as static typing because the compiler checks the values before the program is allowed to execute, vs. checking the value while it executes

Naming your variables

- Identifiers and language keywords in C# are case sensitive

```
int age = 24;
int Age = 35;
```


These are distinct variables in C#,
each capable of holding a different value

Note: the code shown is fine for the C# compiler, but could be confusing to humans who are reading the code, should avoid using the same names for variables if possible!

The value in consistent case conventions

- Two common conventions used for variable identifiers

Camel case: first letter lower case (humps for new words) →

```
int myAge = 24;
```

Pascal case: first letter upper case (also humps for new words) →

```
int MyAge = 35;
```

Pascal and Camel case preferred for readability →

```
int thisisconfusing = 22;

int this_is_yucky = 44;
```

Dealing with whitespace

- Whitespace characters are <space>, <tab>, <new line> and may or may not be significant in your code

Extra whitespace is always okay

Missing whitespace may not be okay

```
int myAge = 25;   // Ok
int myAge =
    25;           // Ok
int myAge=25;     // Ok

intmyAge=25;      // No

string myName = "Thomas Jefferson";
```

whitespace is always preserved in strings

Group Exercise

Write a program that creates a constant, a variable, a string and displays their values to the console

Xamarin University

Flash Quiz

Xamarin University

Flash Quiz

① Can a variable change its value while the program is running

a) Yes

b) No

Flash Quiz

① Can a variable change its value while the program is running

a) Yes

b) No

Flash Quiz

② Does white space matter in a program?

a) Yes
b) No
c) Sometimes

Flash Quiz

② Does white space matter in a program?

a) Yes

b) No

c) Sometimes

Expressions

- An expression is a statement that "returns" a value
  - The expression's value is often assigned to a variable

```
int sum = 5 + 3;

int x = 5, y = 3;
sum = x + y;

bool isMale = true;

int z = y = 5;
```

can initialize two values using comma

can assign multiple variables to same value

Mathematical calculations

- Math operators are used to perform mathematical operations with constants and variables
  - + for addition
  - – for subtraction
  - * for multiplication
  - / for division
  - % for modulo division

```
int  x = 5, y = 11;

int sum = y + x;        // 16

int diff = y - x;       // 6

int product = y * x;    // 55

int quotient = y / x; // 2

int remainder = y % x;  // 1
```

Integer vs. floating point math

- Division of integers returns a whole number (you lose the "remainder")

```
int  x = 5, y = 11;
double z = y / x;    // 2
```

```
double x = 5, y = 11;
double z = y / x;    // 2.2
```

To get fractions, you must divide `double` or `float` values

```
int  x = 5, y = 11;
double z = y % x;    // 1
```

To get remainder, use modulus (%)

Changing a value

- C# allows you to change a variables value after it has been assigned

```
int x = 5;      // start out as 5

x = 10;         // x is now 10

x = x + 1;      // x is now 11 (10 + 1)

x = x * 10;      // x is now 110 (11 x 10)
```


Note: it's important to keep in mind that we are changing *variables* here. These are *not* mathematical formulas being defined.

Compound assignments

- Shorthand syntax available to perform a math operation and assign result back to a variable

these two statements do the same thing – add "1" to x

```
int x = 5;

x = x + 1;    // x is now 6

x += 1;       // x is now 7

x -= 1;       // x is now 6
```

Prefix operator

- Prefix operator allows you to add or subtract "1" from a value and then assign the result to a second variable

```
int x = 5, y = 0, z = 0;

y = ++x; // x = 6, y = 6 increment and then assign

z = --y; // y = 5, z = 5 decrement and then assign
```


Notation is to put increment (++) or decrement (--) in *front* of the variable you want to change

Postfix operator

- Postfix operator allows you to assign a variables value to a second variable and *then* add or subtract "1" from the original variable, in this case the assignment occurs *before* the increment or decrement

```
int x = 5, y = 0, z = 0;

y = ++x; // x = 6, y = 6 increment and then assign
y = x++; // x = 6, y = 5 assign and then increment

z = --y; // y = 5, z = 5 decrement and then assign
z = y--; // y = 4, z = 5 assign and then decrement
```

Put increment (++) or decrement (--) *after* the variable you want to change

Comparative operators

- Comparison operators allow you to compare one value to another and return a **`true`** or **`false`** (boolean) response
  - `>` greater than
  - `>=` greater than or equal to
  - `<` less than
  - `<=` less than or equal to
  - `==` equal to

```
int x = 5, y = 10, z = 10;

y > x;          // true

y < x;          // false

x >= z;         // false

y == z;         // true
```

Flash Quiz

Xamarin University

Flash Quiz

① In the following code, what is the value of z?

```
int x = 3, y = 10;
int z = y / x;
```

a) 3.3
b) 3 Remainder 1
c) 3
d) 1

Flash Quiz

① In the following code, what is the value of z?

```
int x = 3, y = 10;
int z = y / x;
```

a) 3.3

b) 3 Remainder 1

c) 3

d) 1

Flash Quiz

② In the following code fragment, what is the value of z?

```
int x = 3, y = 10;
int z = y % x;
```

a) 3.3
b) 3 Remainder 1
c) 3
d) 1

Flash Quiz

② In the following code fragment, what is the value of z?

```
int x = 3, y = 10;
int z = y % x;
```

a) 3.3
b) 3 Remainder 1
c) 3
d) 1

Summary

1. What is a C# Statement?
2. Understand types
3. Create variables and constants
4. Strings and white space

Where are we going from here?

- You now know some of the basic fundamental ideas behind programming and the C# programming language

- In the next course, we will examine how to make decisions in our programs through *branching* and *loops*

Thank You!

Please complete the class survey in your profile:
university.xamarin.com/profile