# Introduction

### Numbers

There are two different types of numbers in C#:

* Integers: numbers with no digits behind the decimal separator (whole numbers). Examples are -6, 0, 1, 25, 976 and 500000.
* Floating-point numbers: numbers with zero or more digits behind the decimal separator. Examples are -2.4, 0.1, 3.14, 16.984025 and 1024.0.

The two most common numeric types in C# are int and double. An int is a 32-bit integer and a double is a 64-bit floating-point number.

Arithmetic is done using the standard arithmetic operators. Numbers can be compared using the standard numeric comparison operators and the equality (==) and inequality (!=) operators.

C# has two types of numeric conversions:

1. Implicit conversions: no data will be lost and no additional syntax is required.
2. Explicit conversions: data could be lost and additional syntax in the form of a cast is required.

As an int has less precision than a double, converting from an int to a double is safe and is thus an implicit conversion. However, converting from a double to an int could mean losing data, so that requires an explicit conversion.

### If Statements

In this exercise you must conditionally execute logic. The most common way to do this in C# is by using an if/else statement:

int x = 6;

if (x == 5)

{

// Execute logic if x equals 5

}else if (x > 7)

{

// Execute logic if x greater than 7

}else

{

// Execute logic in all other cases

}

The condition of an if statement must be of type bool. C# has no concept of *truthy* values.

# Instructions

In this exercise you'll be writing code to analyze the production of an assembly line in a car factory. The assembly line's speed can range from 0 (off) to 10 (maximum).

At its lowest speed (1), 221 cars are produced each hour. The production increases linearly with the speed. So with the speed set to 4, it should produce 4 \* 221 = 884 cars per hour. However, higher speeds increase the likelihood that faulty cars are produced, which then have to be discarded.

You have three tasks.

#### Task 1: Calculate the success rate:

Implement the (*static*) AssemblyLine.SuccessRate() method to calculate the ratio of an item being created without error for a given speed. The following table shows how speed influences the success rate:

* 0: 0% success rate.
* 1 to 4: 100% success rate.
* 5 to 8: 90% success rate.
* 9: 80% success rate.
* 10: 77% success rate.

#### **Task 2: Calculate the production rate per hour:**

Implement the (*static*) AssemblyLine.ProductionRatePerHour() method to calculate the assembly line's production rate per hour, taking into account its success rate:

AssemblyLine.ProductionRatePerHour(6)*// => 1193.4*

Note that the value returned is a double.

#### **Task 3: Calculate the number of working items produced per minute:**

Implement the (*static*) AssemblyLine.WorkingItemsPerMinute() method to calculate how many working cars are produced per minute:

AssemblyLine.WorkingItemsPerMinute(6)*// => 19*