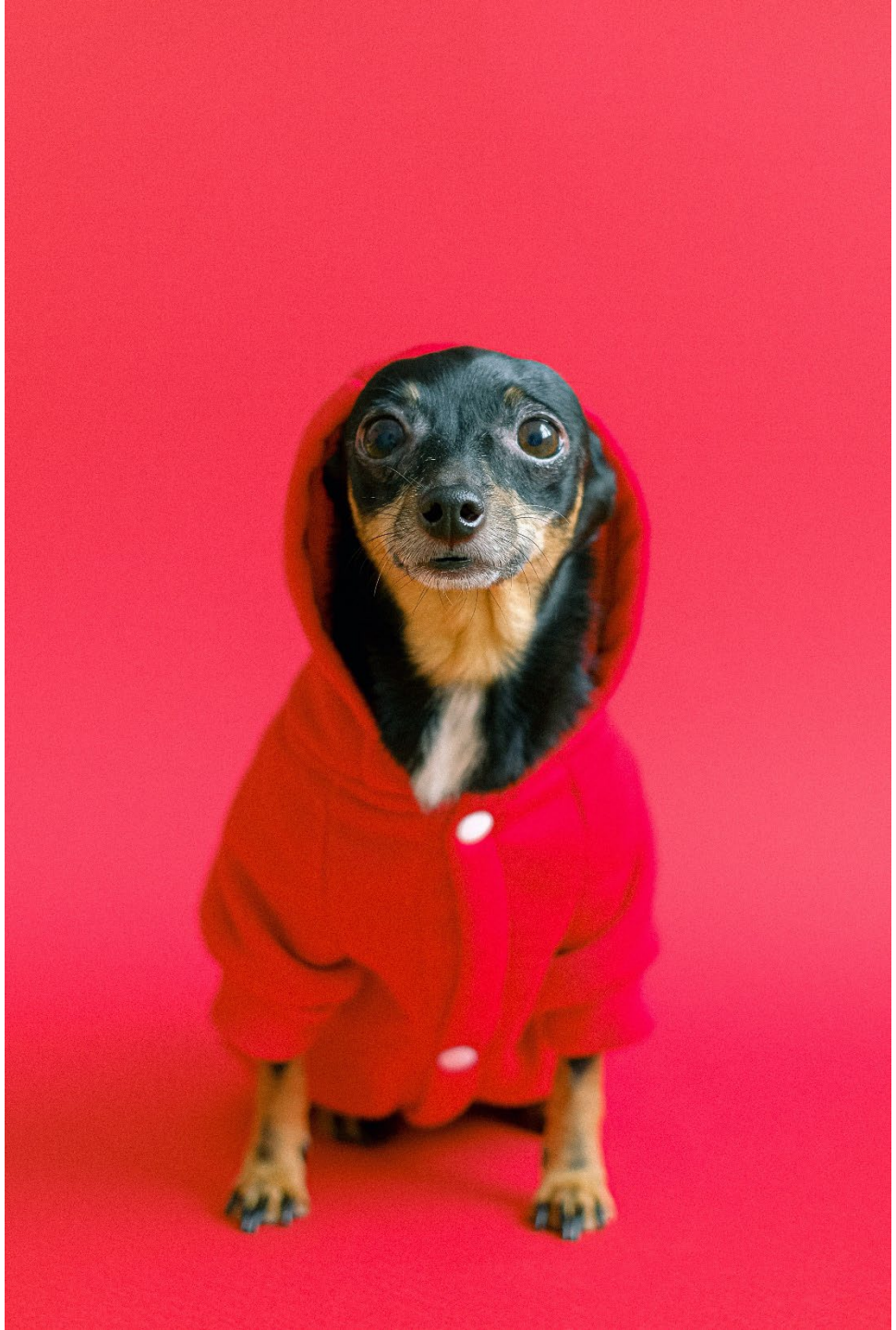
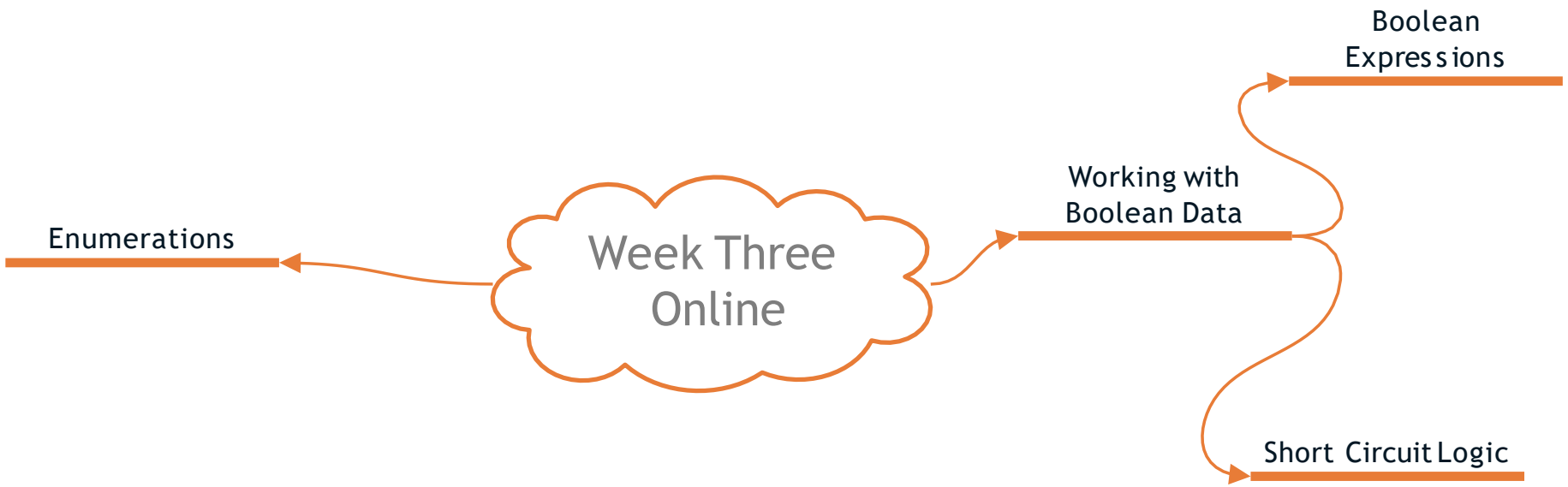
COMP 3602

C# Application Development

Week Three - Online



This Week's Learning Outcomes



Working with Boolean Data

Boolean Values
Can be represented by a literal value, variable, expression or method return.

```
70      while (count < MAX_COUNT)
71      {
72          // code here
73      }
```

If Statement and Loop Tests
Are boolean expressions

```
76      if (count == MAX_COUNT)
77      {
78          // code here
79      }
```

Truth Table for the AND (&&) Operator		
Left Operand	Right Operand	Result
true	true	true
true	false	false
false	true	false
false	false	false

Truth Table for the OR (||) Operator		
Left Operand	Right Operand	Result
true	true	true
true	false	true
false	true	true
false	false	false

Working with Boolean Data

Second Expression Might Not Be Evaluated

Often the result of an AND/OR operation can be determined solely from the first expression.

If the second expression is the result of a method return, the method might not get executed. This may create side effects.

Methods should always be limited to performing a single task and here is an example of why this is sound programming practice.

```
15
16      Console.WriteLine("true && true");
17      result = returnTrue() && returnTrue();
18      Console.WriteLine($"result = {result}\n");
19
20      Console.WriteLine("false && true");
21      result = returnFalse() && returnTrue();
22      Console.WriteLine($"result = {result}\n");
23
```

Boolean Short Circuit Demo

```
true && true
returnTrue method called
returnTrue method called
result = True

false && true
returnFalse method called
result = False
```

The result of this AND operation will be false because the first expression is false.
The second expression need not be evaluated because its value will not alter the result.

Enumerations

```
3 references
public enum DayOfWeek
{
    Sunday
  , Monday
  , Tuesday
  , Wednesday
  , Thursday
  , Friday
  , Saturday
}

0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        DayOfWeek day = DayOfWeek.Tuesday;
        DayOfWeek nextDay = day + 1;

        Console.WriteLine($"Today is:    {day}");
        Console.WriteLine($"Tomorrow is: { nextDay}");
    }
}
```

Elements of an enum. These are labels, not strings as they are not double quoted.

An enumeration (enum) is a user-defined data type that defines a set of constant values, all rolled into a single type.

Enum Facts:

- Value type
- Inherits from System.Enum
- Underlying type is int (by default)
- All fields are constants
- More than one field may have the same underlying value

Enumerations

```
7   namespace EnumerationDayOfWeek
8   {
        3 references
9       public enum DayOfWeek
10      {
11          Sunday
12        , Monday
13        , Tuesday
14        , W   DayOfWeek.Tuesday = 2
15        , Thursday
16        , Friday
17        , Saturday
18      }
19
        0 references
20      class Program
21      {
            0 references
```

By default, the underlying value starts at 0 and increases by 1 for each label that is added.

Enumerations

```
onDayOfWeek                                  EnumerationDayOfWeek.Program
namespace EnumerationDayOfWeek
{
    3 references
    public enum DayOfWeek
    {
        Sunday = 1
      , Monday = 2
      , Tuesday = 3
      , Wednesday = 4
      , Thursday = 5
      , Friday = 6
      , Saturday = 7
    }
```

Can also optionally explicitly set values for each label

Enumerations

```
0 references
public enum DayOfWeekAsLong : long
{
    Sunday
  , Monday
  , Tuesday
  , Wednesday
  , Thursday
  , Friday
  , Saturday
}
```

Values can be of any **numeric** type (int, long, etc) but cannot be strings