CO453 Application Programming

Week 1 - C# Part 4

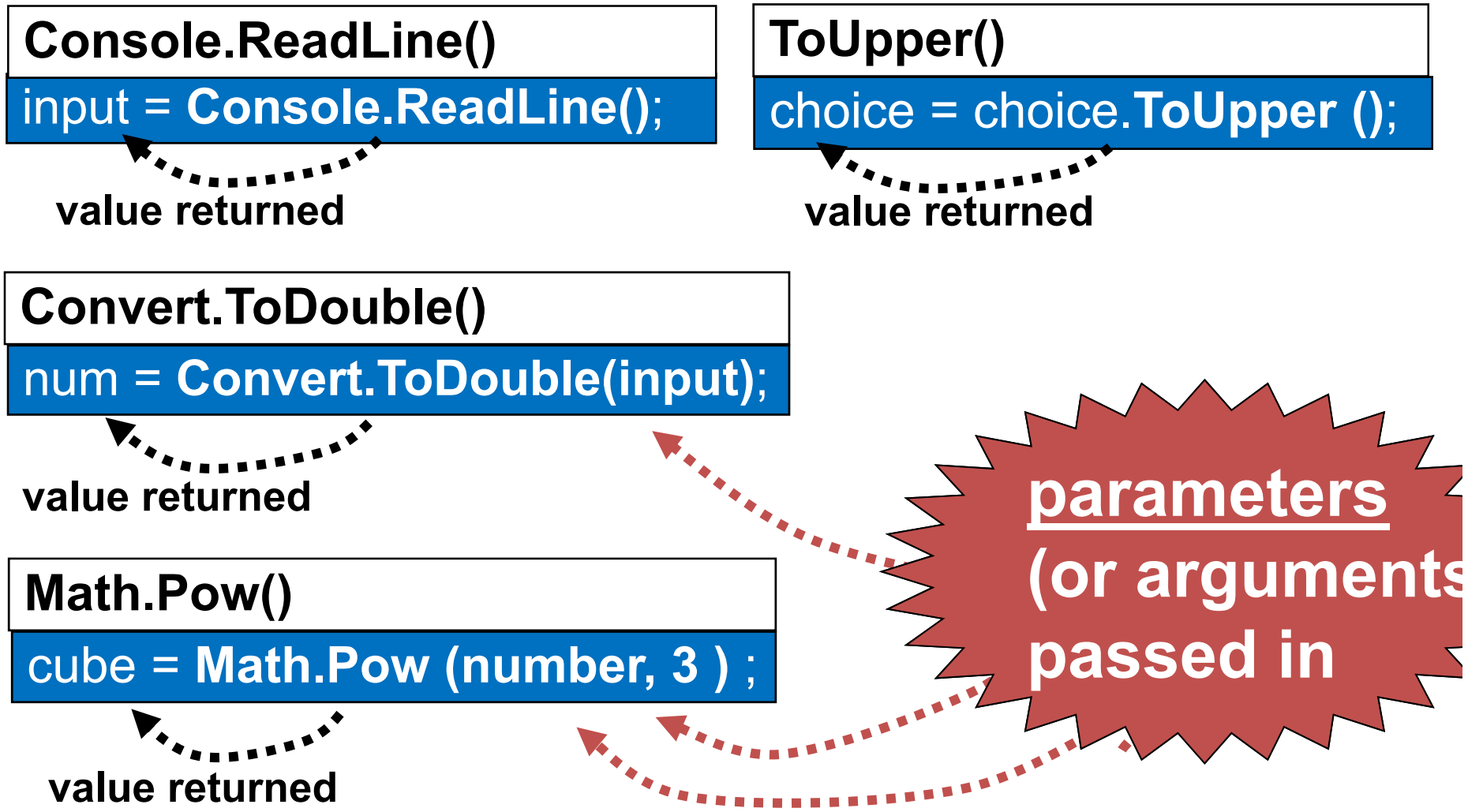
Methods and parameter passing

Some familiar C# methods (functions)

Some methods you may have used already

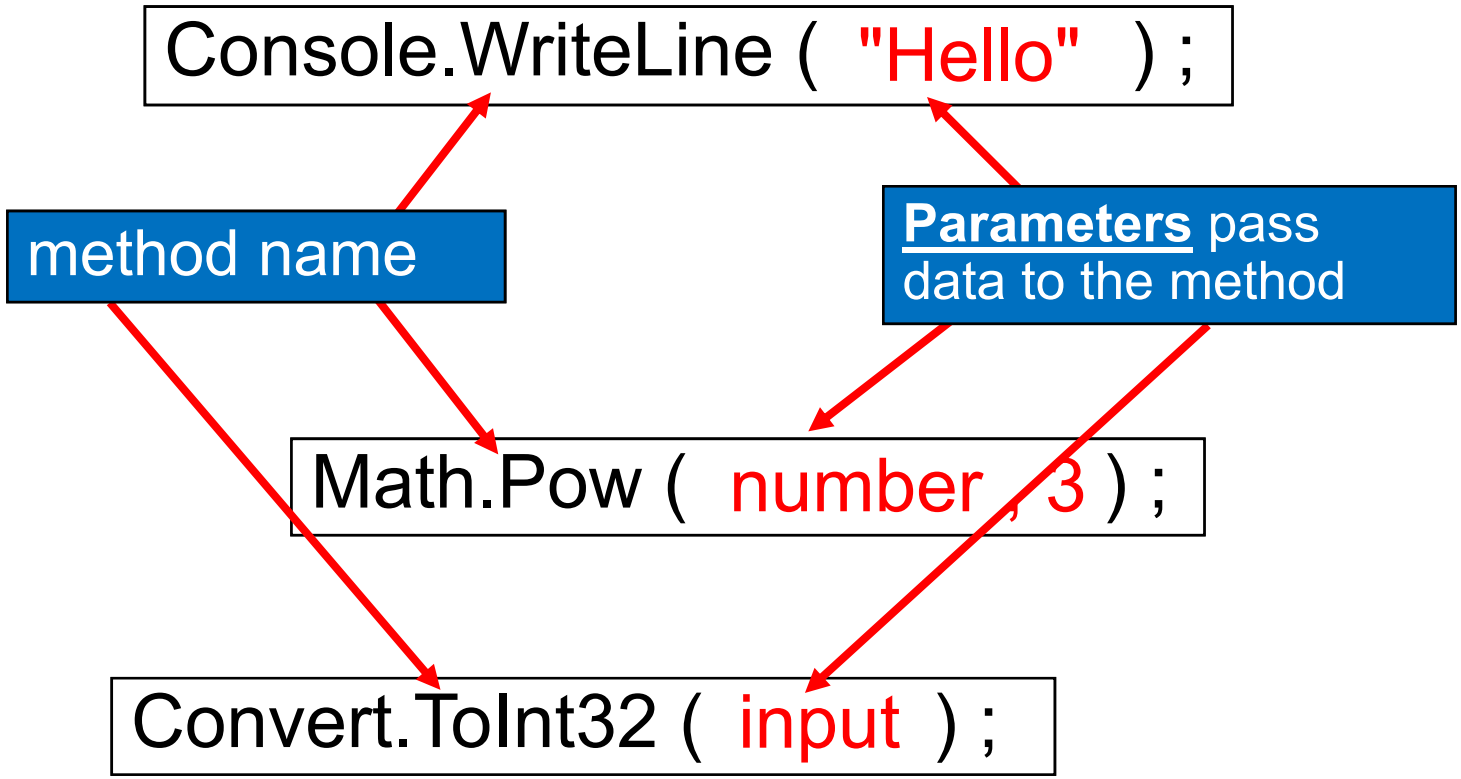
1. **Main()** .. The method that belongs in every C# program. It is always executed first.
2. **Console.Clear()** .. This method is used to clear the console screen
3. **.ToUpper()** .. Converts a string variable into upper case
4. **.ToLower()** .. Converts a string variable into lower case
5. **Convert.ToDouble(…)** .. takes a string **parameter** and returns a double number value
6. **Convert.ToInt32(…)** .. takes a string **parameter** and returns an integer number value

Examples of Methods with and without parameters



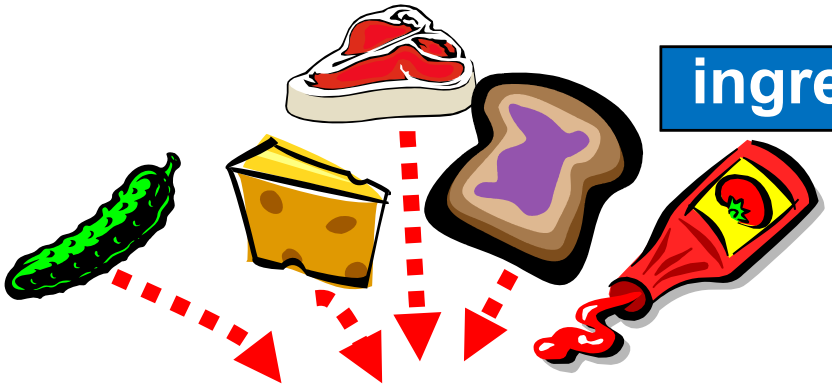
What are Parameters?

Parameters are used to pass information into methods, using the brackets as a kind of 'doorway' e.g.



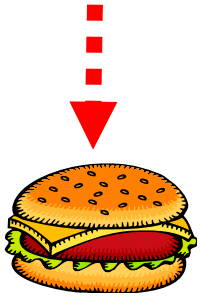
The method uses the parameter data and does something with it when the method is called

A method or function is like a burger maker



ingredients (parameters)

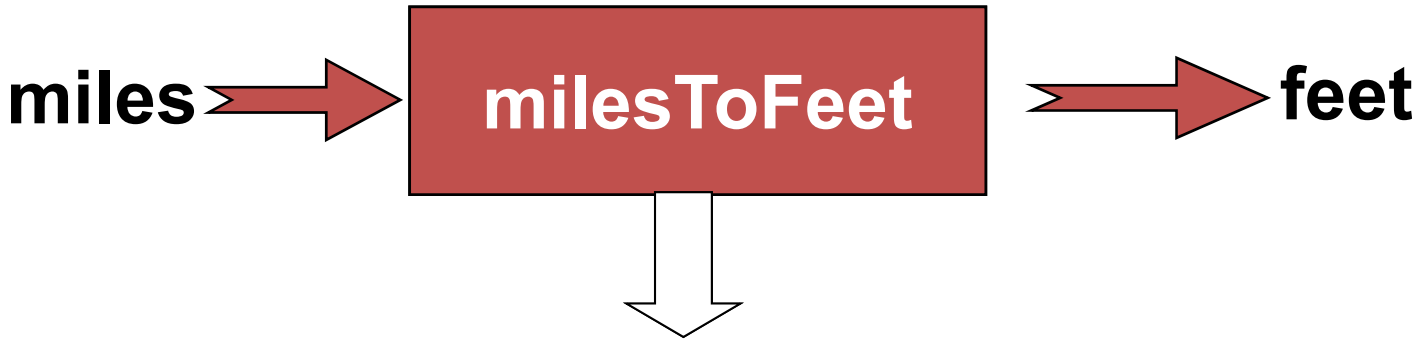
burger maker



burger (result returned)

Designing a method to convert miles into feet

milesToFeet() method




```
public double milesToFeet( double miles )
{
    double feet;          // local variable
    feet = miles * 1760 * 3 ;
    return feet;
}
```

Using the milesToFeet() method

```
public double milesToFeet(double miles )
{
    double feet;
    feet = miles * 1760 * 3 ;
    return feet;
}
```

```
double numMiles = 10;
double numFeet;
numFeet = milesToFeet(numMiles);

Console.WriteLine(numMiles + " miles is "
                  + numFeet + " feet" );
```

The output is : 10 miles is 52800 feet

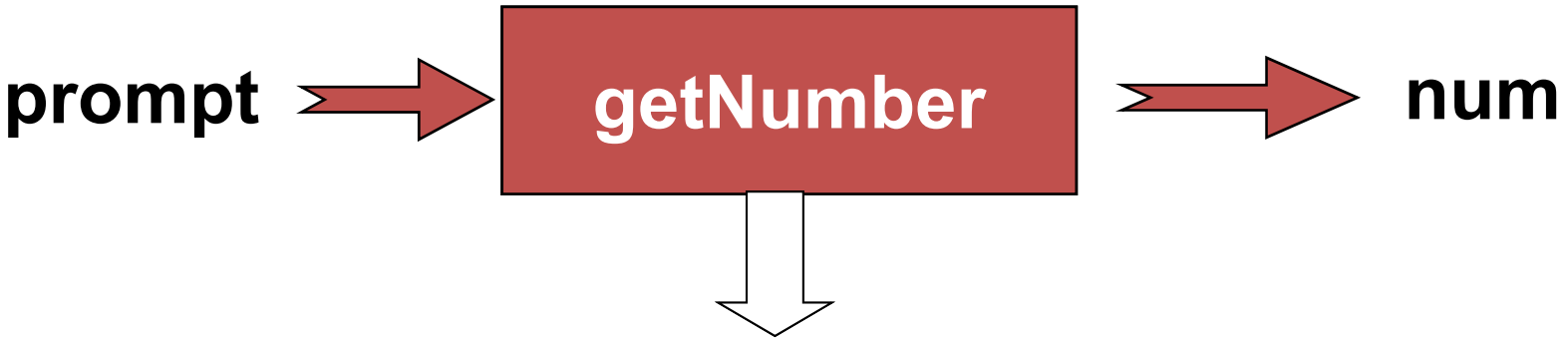
Formal and Actual Parameters

You should have noticed that there are two sets of parameters being used

- The ones that are used to pass values to a method or function are called the **actual parameters**
- The parameters that collect these values in the method or function are called **formal parameters** and are defined in the function itself.

Designing a method for general input of numbers

getNumber() method




```
public double getNumber( string prompt )
{
    double num;     // local variables
    string input;
    Console.Write("Please enter the
                  number of " + prompt);
    input = Console.ReadLine();
    num = Convert.ToDouble(input);
    return num;
}
```

Using the getNumber() method

```
public double getNumber( string prompt )
{
    double num;
    string input;
    Console.Write("Enter the number of " + prompt);
    input = Console.ReadLine();
    num = Convert.ToDouble(input);
    return num;
}
```

```
double numMiles;
numMiles = getNumber("miles");
Console.WriteLine("You entered " + numMiles);
```

The Converter Class

a class that uses the **getNumber()** and **milesToFeet()** methods

Converter class

```
class Converter
{
```

```
private double numMiles, numFeet;
```

```
public static void Main()
{
    Converter myConverter = new Converter();
    myConverter.test();
}
```

```
public void test()
{
    numMiles = getNumber("miles");
    numFeet = milesToFeet(numMiles);
    Console.WriteLine(numMiles + " miles is "
                      + numFeet + " feet");
}
```

Converter class (contd.)

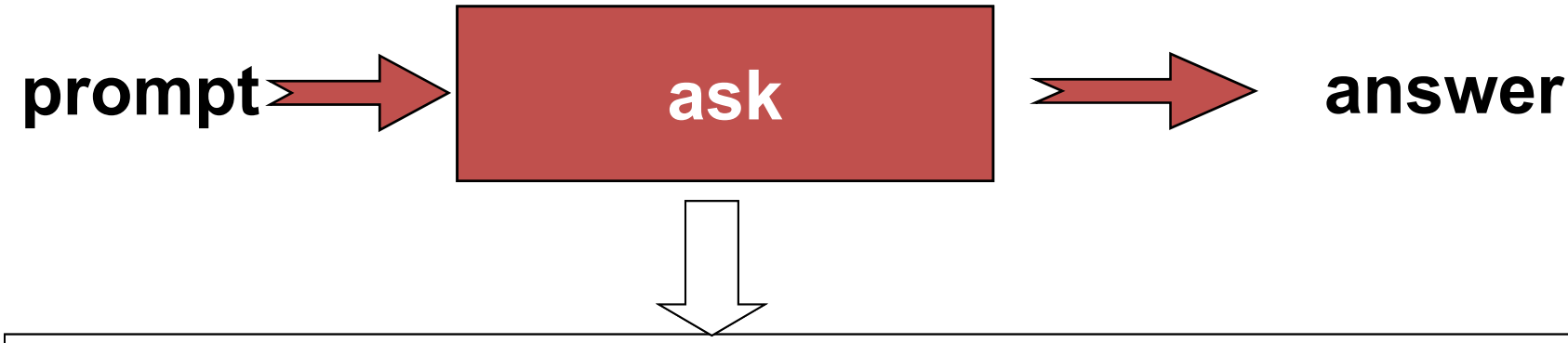
```
public double getNumber(string prompt)
{
   double num; string input;
   Console.Write("Enter the number of " + prompt);
   input = Console.ReadLine();
   num = Convert.ToDouble(input);
   return num;
}
```

```
public double milesToFeet(double miles)
{
   double feet;
   feet = miles * 1760 * 3 ;
   return feet;
}
```

```
} // end of Converter class
```

Designing a method for inputting strings

ask() method




```
public string ask(string prompt )
{
    string answer;          // local variable
    Console.Write(prompt);
    answer = Console.ReadLine();
    return answer;
}
```

The Book Class

an interactive book using the ask() method

Book class

```
class Book
{
```

```
private string author, name, weapon;
```

```
public static void Main()
{
    Book myBook = new Book();
    myBook.getDetails();
    myBook.writeChapter1();
}
```

```
public string ask(string prompt)
{
   string answer;          // local variable
   Console.Write(prompt);
   answer = Console.ReadLine();
   return answer;
}
```

other methods

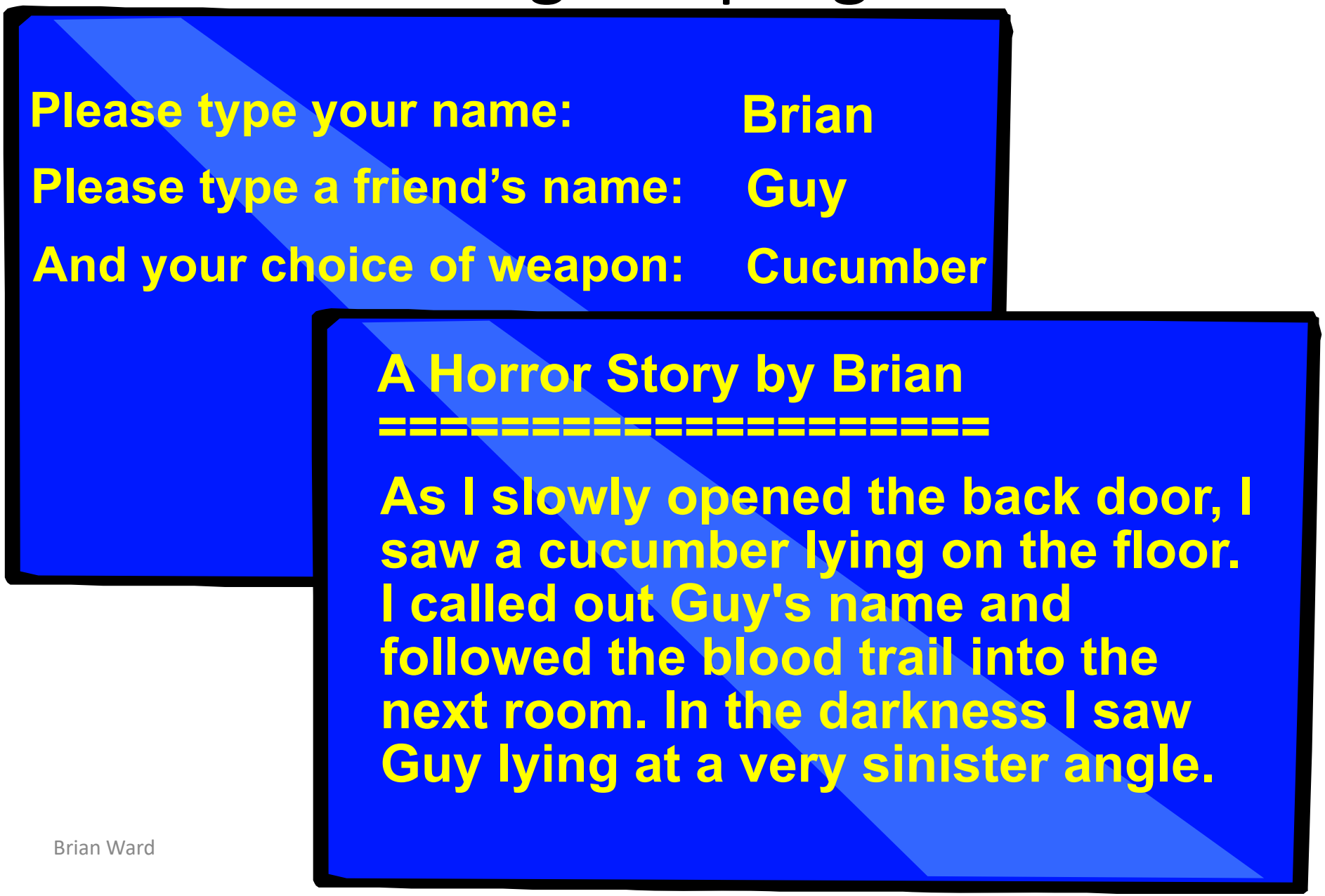
Book class (contd.)

```
public void getDetails()
{
   author = ask("Please type your name:");
   name = ask("Please type a friend's name:");
   weapon = ask("And your choice of weapon:");
}
```

```
public void writeChapter1()
{
  Console.Clear();
  Console.WriteLine("A Horror Story by " + author);
  Console.WriteLine("========================");
  Console.WriteLine("As I slowly opened the back
  door, I saw a " + weapon + " lying on the floor.
  I called out " + name + "'s name and followed the
  blood trail into the next room. In the darkness I
  saw " + name + " lying at a very sinister angle."
}
```

```
} // end of Book class
```

Running the program



How can we return more than 1 result from a method or function?

- We can use parameters to make changes to the original variables.
- To do this we can use reference parameters instead of value parameters.
- Reference parameters are defined using **ref**
  e.g. public void times (ref double n1, ref double n2)
  defines n1 and n2 as reference parameters
- Now any change to n1 or n2 inside the method will also change the value of the parameter passed to it.
- This is because they are essentially the same variable .. using the same memory address
- Note you must also use **ref** when you call the method
  e.g. times (ref number1, ref number2);

The Constructor

- The constructor is a special method in a class
- It always has the same name as the class
- When an object is created from a class, the constructor is automatically executed
- It is used to initialise the new object

```
public Dice()
{
   randy = new Random();
}
```

This constructor creates a new Random object, used to generate random numbers for the Dice

The Last Slide

Extra Reading

Methods and Parameters

- Using Parameters helps to make class methods and functions more useful
- Methods can then work in different situations and different programs
- this increases independence and flexibility