

Starter



1. What is the range for $2 * \text{random.Next}(1, 7) + 1$?
 -
2. Write the code to generate random integers from -5 to 0 inclusive
 -
3. Derive a general command for generating random positive integers between 1 and n inclusive
 -
4. What about for positive random integers which start at a and end at b ?
5. What about for positive integers which start at a , end at b and go up in steps of c ?

Starter



1. What is the range for $2 * \text{random.Next}(1, 7) + 1$?
 - Odd integers, 3 to 13
2. Write the code to generate random integers from -5 to 0 inclusive
 -
3. Derive a general command for generating random positive integers between 1 and n inclusive
 -
4. What about for positive random integers which start at a and end at b ?
5. What about for positive integers which start at a , end at b and go up in steps of c ?

Starter



1. What is the range for $2 * \text{random.Next}(1, 7) + 1$?
 - Odd integers, 3 to 13
2. Write the code to generate random integers from -5 to 0 inclusive
 - `random.Next(-5, 1)`
3. Derive a general command for generating random positive integers between 1 and n inclusive
 -
4. What about for positive random integers which start at a and end at b ?
5. What about for positive integers which start at a , end at b and go up in steps of c ?

Starter



1. What is the range for `2 * random.Next(1, 7) + 1`?
 - Odd integers, 3 to 13
2. Write the code to generate random integers from -5 to 0 inclusive
 - `random.Next(-5, 1)`
3. Derive a general command for generating random positive integers between 1 and n inclusive
 - `random.Next(1, n + 1)`
4. What about for positive random integers which start at a and end at b ?
5. What about for positive integers which start at a , end at b and go up in steps of c ?

Starter



1. What is the range for $2 * \text{random.Next}(1, 7) + 1$?
 - Odd integers, 3 to 13
2. Write the code to generate random integers from -5 to 0 inclusive
 - `random.Next(-5, 1)`
3. Derive a general command for generating random positive integers between 1 and n inclusive
 - `random.Next(1, n + 1)`
4. What about for positive random integers which start at a and end at b ?
5. What about for positive integers which start at a , end at b and go up in steps of c ?

Starter

4. What about for positive random integers which start at a and end at b ?
 -
5. What about for positive integers which start at a , end at b and go up in steps of c ?
 -

Starter

4. What about for positive random integers which start at a and end at b ?
 - `random.Next(a, b + 1)`
5. What about for positive integers which start at a , end at b and go up in steps of c ?
 -

Starter

4. What about for positive random integers which start at a and end at b ?
 - `random.Next(a, b + 1)`
5. What about for positive integers which start at a , end at b and go up in steps of c ?
 - `c * random.Next(0, (int) ((b - a) / (c + 1))) + a`

Approaching weekly assignments

Discuss: Which of these behaviours help learning? Which hinder learning? Why?

1. Use multiple different colours when marking your work rather than just green or red
2. Create summary notes as a separate document
3. When you're not sure on a question, paste in the correct answer and mark it as correct
4. Reading through your notes
5. When marking, write a comment to explain why you got something incorrect
6. Look at the answers after attempting each question
7. Look at the answers after attempting all questions
8. Look at your notes while answering the questions

9. Complete the assignment across multiple days

Subroutines

Subroutines

- Self-contained block of code that itself has an **identifier**
- There are other names
 - Function, procedure, routine, method
- Can be **called** from other parts of the program using its identifier
- May or may not have inputs/arguments/parameters
- May or may not return a value

Subroutines

- Self-contained block of code that itself has an **identifier**
- There are other names
 - Function, procedure, routine, method
- Can be **called** from other parts of the program using its identifier
- May or may not have inputs/arguments/parameters
- May or may not return a value
- Today, we are focusing on **procedures**
 - A **procedure** is a subroutine that does not return a value
 - They are sometimes known as **void functions**

A brief detour

- A **class** is a collection of properties and subroutines defined for a particular type of object
 - e.g. a person
- We can have multiple **instances** of a class
 - Each instance has its own properties
 - person1 has name "Alex"
 - person2 has name "Tristan"

A brief detour

- A **class** is a collection of properties and subroutines defined for a particular type of object
 - e.g. a person
- We can have multiple **instances** of a class
 - Each instance has its own properties
 - person1 has name "Alex"
 - person2 has name "Tristan"
- Sometimes, we want properties or subroutines to belong to the class itself, rather than each instance
 - e.g. an automatically incrementing ID for each person
- We call these **static**

A brief detour

- A **class** is a collection of properties and subroutines defined for a particular type of object
 - e.g. a person
- We can have multiple **instances** of a class
 - Each instance has its own properties
 - person1 has name "Alex"
 - person2 has name "Tristan"
- Sometimes, we want properties or subroutines to belong to the class itself, rather than each instance
 - e.g. an automatically incrementing ID for each person
- We call these **static**

- Our code currently is wrapped in a Program class that represents our whole program, so anything we want to exist once for the whole program is marked as **static**

Writing procedures

```
static void GreetUser()  
{  
    Console.Write("Name: ");  
    string name = Console.ReadLine();  
  
    Console.WriteLine("Hello, " + name);  
}  
  
static void Main(string[] args)  
{  
    GreetUser(); // This calls the GreetUser subroutine  
}
```

Example

```
static void Answer()  
{  
    Console.WriteLine("Fine, thank you");  
}  
  
static void Question()  
{  
    Console.WriteLine("How are you?");  
}  
  
static void Main(string[] args)  
{  
    Console.WriteLine("Here is a question...");  
    Question();  
}
```

Subroutines

```
    Answer();  
}
```

Local variables

- Local variables are declared inside a subroutine
 - They are only available in the subroutine they are declared in
- Multiple subroutines can have local variables with the same identifier, but they are not shared across subroutines

Global variables

- Global variables exist at the root level of the program and can be accessed by any subroutine
 - Technically there is no such thing as a global variable in C#, but we can have a **static** property in the **Program** class to achieve the same effect
 - More on this next term with OOP

Example

```
class Program
{
    static int subroutineCalls = 0; // this is a global variable

    static void SayHello()
    {
        subroutineCalls = subroutineCalls + 1;
        Console.WriteLine("Hello!");
    }

    static void Main()
    {
        SayHello();
        SayHello();
    }
}
```

Subroutines

```
    Console.WriteLine("SayHello called " + subroutineCalls + " times");  
}  
}
```

Exercise

You have **10 minutes**:

- Write a program with three procedures
 1. `GetName()` – ask the user to enter their name, which should be stored in a global variable
 2. `GetAge()` – ask the user for their age, and store it in a local variable. Then print the name and age together, in one line
 3. `Main()` – the main entry point for the program. It should call `GetName()`, then `GetAge()`

Extension: Add a procedure called `GetFavouriteColour()`. Then, have `GetAge()` output text in this colour. You'll need another global variable.

Practice



```
class Program {  
    static string aphorism = "";  
  
    static void Quote()  
    {  
        Console.WriteLine("It was the " + aphorism + " of times");  
    }  
  
    static void Main()  
    {  
        aphorism = "best";  
        Quote();  
        aphorism = "worst";  
        Quote();  
    }  
}
```

```
}  
}
```

What does this code print?

Practice



```
class Program {  
    static int numThing = 0;  
    static void Tricksy()  
    {  
        Console.Write(numThing + ", ");  
    }  
    static void Main()  
    {  
        for (int i = 0; i < 5; i += 2)  
        {  
            numThing = i + 1;  
            Tricksy();  
        }  
    }  
}
```

What does this code print?

Practice



```
class Program {  
    static int numThing = 0;  
    static void Tricksy()  
    {  
        Console.Write(numThing + ", ");  
    }  
    static void Main()  
    {  
        for (int i = 0; i < 5; i += 2)  
        {  
            numThing = i + 1;  
            Tricksy();  
        }  
    }  
}
```

What does this code print? 1, 3, 5,

Practice



```
class Program {  
    static int numThing = 0;  
    static void Tricksy()  
    {  
        Console.Write(numThing + ", ");  
    }  
    static void Main()  
    {  
        for (int i = 0; i < 5; i += 2)  
        {  
            int numThing = 0;  
            numThing = i + 1;  
            Tricksy();  
        }  
    }  
}
```

```
}  
}
```

What does this code print?

Practice



```
class Program {  
    static int numThing = 0;  
    static void Tricksy()  
    {  
        Console.Write(numThing + ", ");  
    }  
    static void Main()  
    {  
        for (int i = 0; i < 5; i += 2)  
        {  
            int numThing = 0;  
            numThing = i + 1;  
            Tricksy();  
        }  
    }  
}
```

```
}  
}
```

What does this code print? 0, 0, 0,

Practice



```
class Program {  
    static int numThing = 0;  
    static void Tricksy()  
    {  
        Console.WriteLine(numThing);  
    }  
    static void Main()  
    {  
        int i = 0;  
        while (i < 4)  
        {  
            Tricksy();  
            i++;  
        }  
    }  
}
```

What does this code
print?

Practice



```
class Program {  
    static int numThing = 0;  
    static void Tricksy()  
    {  
        Console.WriteLine(numThing);  
    }  
    static void Main()  
    {  
        int i = 0;  
        while (i < 4)  
        {  
            Tricksy();  
            i++;  
        }  
    }  
}
```

What does this code
print?

0

0

0

0

Practice



```
class Program {  
    static int numThing = 0;  
    static void Tricksy()  
    {  
        Console.WriteLine(numThing);  
        numThing -= 1;  
    }  
    static void Main()  
    {  
        numThing = 0;  
        while (numThing < 4)  
        {  
            Tricksy();  
            numThing += 2;  
        }  
    }  
}
```

What does this code
print?

Practice



```
class Program {  
    static int numThing = 0;  
    static void Tricksy()  
    {  
        Console.WriteLine(numThing);  
        numThing -= 1;  
    }  
    static void Main()  
    {  
        numThing = 0;  
        while (numThing < 4)  
        {  
            Tricksy();  
            numThing += 2;  
        }  
    }  
}
```

What does this code
print?

0

1

2

3

Procedures



- It is considered good programming practice to use a different procedure for each ‘task’ the program has to carry out

In pairs, discuss ‘**why?**’. Write down **two** reasons on a whiteboard.

Statistics program

You have **10 minutes**:

- In `Main()`, get the user to input four numbers and store each in a global variable
- Write a procedure to calculate and print the total
- Write a subroutine to find and print the average (mean)

Extension:

- Find the mode
- Find the median

Menus

- Write a program with a menu using subroutines

```
Choose what you want to do
```

```
1. Print red text  
2. Print two times tables  
3. Print a joke  
4. Quite program
```

- Include a subroutine for the menu itself, and a subroutine to be called for each item in the menu

Hint: `switch` may be useful for this