

C# Exceptions - Try.Catch

C# Exceptions

When executing C# code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things.

When an error occurs, C# will normally stop and generate an error message. The technical term for this is: C# will throw an **exception** (throw an error).

C# try and catch

The `try` statement allows you to define a block of code to be tested for errors while it is being executed.

The `catch` statement allows you to define a block of code to be executed, if an error occurs in the try block.

The `try` and `catch` keywords come in pairs:

Syntax

```
try
{
    // Block of code to try
}
catch (Exception e)
{
    // Block of code to handle errors
}
```

Consider the following example, where we create an array of three integers:

This will generate an error, because **myNumbers[10]** does not exist.

```
int[] myNumbers = {1, 2, 3};
```

```
Console.WriteLine(myNumbers[10]); // error!
```

The error message will be something like this:

```
System.IndexOutOfRangeException: 'Index was outside the bounds of the array.'
```

If an error occurs, we can use `try...catch` to catch the error and execute some code to handle it.

In the following example, we use the variable inside the catch block (`e`) together with the built-in `Message` property, which outputs a message that describes the exception:

Example

```
try
{
    int[] myNumbers = {1, 2, 3};
    Console.WriteLine(myNumbers[10]);
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}
```

The output will be:

```
Index was outside the bounds of the array.
```

Exercise (try it)

Program.cs

```
using System;

namespace MyApplication
{
    class Program
    {
        static void Main(string[] args)
        {

```

```
try
{
    int[] myNumbers = {1, 2, 3};
    Console.WriteLine(myNumbers[10]);
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}
}
```

You can also output your own error message:

Example

```
try
{
    int[] myNumbers = {1, 2, 3};
    Console.WriteLine(myNumbers[10]);
}
catch (Exception e)
{
    Console.WriteLine("Something went wrong.");
}
```

The output will be:

Something went wrong.

Exercise (try it)

Program.cs

```
using System;

namespace MyApplication
{
    class Program
    {
```

```
static void Main(string[] args)
{
    try
    {
        int[] myNumbers = {1, 2, 3};
        Console.WriteLine(myNumbers[10]);
    }
    catch (Exception e)
    {
        Console.WriteLine("Something went wrong.");
    }
}
```

Finally

The `finally` statement lets you execute code, after `try...catch`, regardless of the result:

Example

```
try
{
    int[] myNumbers = {1, 2, 3};
    Console.WriteLine(myNumbers[10]);
}
catch (Exception e)
{
    Console.WriteLine("Something went wrong.");
}
finally
{
    Console.WriteLine("The 'try catch' is finished.");
}
```

The output will be:

```
Something went wrong.  
The 'try catch' is finished.
```

Exercise – (try it)

Program.cs

```
using System;  
  
namespace MyApplication  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            try  
            {  
                int[] myNumbers = {1, 2, 3};  
                Console.WriteLine(myNumbers[10]);  
            }  
            catch (Exception e)  
            {  
                Console.WriteLine("Something went wrong.");  
            }  
            finally  
            {  
                Console.WriteLine("The 'try catch' is finished.");  
            }  
        }  
    }  
}
```

The throw keyword

The `throw` statement allows you to create a custom error.

The `throw` statement is used together with an **exception class**. There are many exception classes available in

C#: `ArithmeticException`, `FileNotFoundException`, `IndexOutOfRangeException`, `TimeoutException`, etc:

Example

```
static void checkAge(int age)
```

```
{  
  
    if (age < 18)  
    {  
        throw new ArithmeticException("Access denied - You must be at least  
18 years old.");  
    }  
    else  
    {  
        Console.WriteLine("Access granted - You are old enough!");  
    }  
}  
  
static void Main(string[] args)  
{  
    checkAge(15);  
}
```

The error message displayed in the program will be:

```
System.ArithmeticException: 'Access denied - You must be at least  
18 years old.'
```

If `age` was 20, you would **not** get an exception:

Example

```
checkAge(20);
```

The output will be:

```
Access granted - You are old enough!
```

Program.cs

```
using System;  
  
namespace MyApplication
```

```
{  
    class Program  
    {  
        static void checkAge(int age)  
        {  
            if (age < 18)  
            {  
                throw new ArithmeticException("Access denied - You must be at  
least 18 years old.");  
            }  
            else  
            {  
                Console.WriteLine("Access granted - You are old enough!");  
            }  
        }  
  
        static void Main(string[] args)  
        {  
            checkAge(20);  
        }  
    }  
}
```

Reference

www.w3schools.com