Code Reviewing

# Day of Year.txt

Although this code mostly works (December is wrong), it doesn’t cater for leap years. It always assumes that there are only 28 days in February.

The name of the function ‘**dayOfYear’** isn’t descriptive. Does it return a day Monday, Tuesday etc, or the number of days since the start of the year.

The order of the parameters is unfortunate. It would be clearer if it was year, month then day. Note the Year parameter is currently unused as it doesn’t check for leap years.

There is no validation of the parameters, so it would be possible to pass in an invalid day/month. One solution is to pass in a single Date parameter other than separate day, month and years

Instead of the multiple **If** statements, the number of days could be precomputed and held in an array as shown below.

int NumberOfDaysSinceTheStartOfTheYear(DateTime date)

{

var days = new[] { 0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334 };

var result = date.Day + days[date.Month - 1];

var isLeapYear = DateTime.IsLeapYear(date.Year);

if (isLeapYear && date.Month > 2)

result++;

return result;

}

Another solution could be just

int NumberOfDaysSinceTheStartOfTheYear(DateTime date)

{

return date.Subtract(new DateTime(date.Year, 1, 1)).Days + 1;

}

Or in C#, use the built-in method

int NumberOfDaysSinceTheStartOfTheYear(DateTime date)

{

return date.DayOfYear;

}

The source code for this can be found at <https://referencesource.microsoft.com/#mscorlib/system/datetime.cs,ff06f271f088f1a8>

# 2. Is Leap Year

None of the comments add to this code, and only end up hiding the actual logic from the reader.

* The Header copyright block gives no value
* Change history can be viewed from source control
* The old version of the code can be viewed from source control
* The comments for the method header give no value. The convention in C# would be to use /// as that can be used by [intellisense](https://www.google.co.uk/search?safe=active&q=intellisense&spell=1&sa=X&ved=0ahUKEwiumrG5l7_ZAhXGI8AKHQ5zD5IQBQgmKAA).
* The comments in the method give no value

This means that whole file reduces to

bool IsLeapYear(int year)

{

if ((year % 4) != 0) return false;

if ((year % 100) != 0) return true;

return ((year % 400) != 0);

}

The logic in the function is still quite hard to follow. The rules for leap years are

1. The year must be divisible for 4
2. The year must not be divisible by 100 unless it is also divisible by 400

The corrected version of the function would be

bool IsLeapYear(int year)

{

if ((year % 4) != 0) return false;

if ((year % 400) == 0) return true;

return ((year % 100) != 0);

}

This could be re-written to remove the multiple return statements

bool IsLeapYear(int year)

{

var divisbleBy4 = ((year % 4) == 0);

var divisbleBy100 = ((year % 100) == 0);

var divisbleBy400 = ((year % 400) == 0);

return (divisbleBy4 && (!divisbleBy100 || divisbleBy400));

}

Or again, most languages have a built-in method.

# Anagram Checker

Under most conditions this prime number based algorithm works. It will however error when

* Given any character (including whitespace) which isn’t A-Z
* Given a lowercase character.

The algorithm will also have a problem when the generated number exceeds the size of a ‘long’. For example, ZZZZZZZZZZ will overflow a long in C#. In C# one of two things will happen depending on a project level setting

* The program can crash with an overflow error
* Or by default the overflow will be ignored and the value for the long will loop around. This means that it’s possible for two words to generate the same number.

The name of the internal method ‘**AsNumber’** is not very descriptive, a better name might be **ToHash**.

The namespace’s name **IsAnagram** is like the class’s name **AnagramChecker** and so doesn’t provide any additional value.

In terms of performance

* The **IsThisAnAnagram** method could start with a check to ensure that both words are the same length, and return False if not.
* The array of primes is built each time **lookupNumber** is called, this could be changed to a static array.

The method itself is quite complicated and not very efficient, a simpler solution could be

public static bool IsAnagram(string word1, string word2)

{

if (word1.Length != word2.Length) return false;

var lettersFromWord1InOrder = word1.ToArray().OrderBy(letter => letter);

var lettersFromWord2InOrder = word2.ToArray().OrderBy(letter => letter);

return (lettersFromWord1InOrder.SequenceEqual(lettersFromWord2InOrder));

}

# 5. Change Password.jpg

The error was caused by trying to store a password which is too long into the database. Your first thought might be that the frontend should have restricted the maximum length that could be entered in the **new password** field. The bigger issue is however that passwords shouldn’t be stored in plaintext. Instead a hash of the password should be stored. Hashes tend to be the same length regardless of the length of the entered password.

The error message tells you that the application was written in vb.net. The use of vb.net isn’t the issue, but the fact that the application has leaked the information. Raw error messages should not be displayed to end users. The user should see “I’m sorry but an error has occurred” and the full error message should be logged.

The catch block is a waste of time and doesn’t add anything. The use of **Throw ex** rather than just **Throw** damages the stack trace. We can’t see the rest of the code, but it’s likely that the try-catch-finally block could be replaced by a **using** statement.

Finally, the stack trace shows us that a method called **ExecuteSQL** is being called, whilst being passed a string parameter called **SqlStr**. We don’t know for sure, but it’s looks likely that this application might be subject to SQL injection attacks

# 8. Noughts and Crosses (tic.html)

There are many issues with the code, but the main security issue was the following line at the end of the page.

**<script type="text/javascript" src="//www.browsealoud.com/plus/scripts/ba.js"></script>**

This loads in a 3rd party script, which had been compromised with a bitcoin miner

The correct way to include a 3rd party script is to also add a sha hash (SRI) of what you expect the script to contain. For example

<script src="//www.browsealoud.com/plus/scripts/ba.js" integrity="sha256-Abhisa/nS9WMne/YX+dqiFINl+JiE15MCWvASJvVtIk=" crossorigin="anonymous"></script>

If the script then gets changed, the hash will be wrong, and you page won’t load it.

A good resource for this is - https://scotthelme.co.uk/protect-site-from-cryptojacking-csp-sri/

**End of Code**