OO APPLICATION DEVELOPMENT– LES02

ROGIER VAN DER LINDE

LES 01 terugblik

Topics les01

Check of je volgende onderwerpen onder de knie hebt:

- werken met projecten, XAML, design view, codebehind… (herhaling AD1)
- events koppelen aan WPF controls (herhaling AD2)
- werken met eenvoudige vormen: **`Ellipse`**, **`Rectangle`**
- de klasse **`Random`** gebruiken
- de klassen **`DispatcherTimer`** en **`TimeSpan`** gebruiken
- de **`String`** klasse: interpolatie & verbatim, **`Length`** property, methodes **`ToLower()`**, **`Trim()`**, **`Substring()`**, **`IndexOf()`**, **`Split()`**…

DispatcherTimer, TimeSpan

Fragment:

```
private int ticks;
private DispatcherTimer timer;

public MainWindow() {
    InitializeComponent();
    ticks = 0;
    timer = new DispatcherTimer();
    timer.Interval = TimeSpan.FromMilliseconds(200);
    timer.Tick += DoSomething;
}

private void DoSomething(object sender, EventArgs e) {
    // elke 200 milliseconden uitgevoerd
    ticks++;
}
```

Drawing

Fragment voor het tekenen van een ellips met gebruik van brushes and thickness:

```
public MainWindow() {
    // ...
    ellipse = new Ellipse();
    ellipse.Width = 400;
    ellipse.Height = 200;
    ellipse.Stroke = new SolidColorBrush(Colors.Orange);
    ellipse.Fill = new SolidColorBrush(Colors.Black);
    ellipse.Margin = new Thickness(0, 0, 0, 0);
    canvas.Children.Add(ellipse);
}
```

- je kan de ellips en de meeste eigenschappen ook gewoon in XAML of in Design View aanmaken en instellen
- om inconsistenties te vermijden (bv. label waarde klopt niet met slider waarde), stel je waarden best zoveel mogelijk in via MainWindow() en niet via XAML of Design View

Les02 topics

Topics les02

System.IO namespace:

- bestanden schrijven met StreamWriter; methode WriteLine()
- bestanden lezen met StreamReader; methodes ReadLine() en ReadToEnd()
- paden opbouwen met System.IO.Path.Combine()
- directories lezen met Directory.GetFiles en Directory.GetDirectories
- file informatie met FileInfo

System.Environment klasse:

- speciale folders gebruiken met GetFolderPath() methode
- NewLine property
- Exit() methode

Topics les02

Microsoft.Win32 namespace:

- gebruik van de OpenFileDialog en SaveFileDialog klasse

Andere:

- try-catch-finally, Exceptions
- MessageBox
- C# collections: array, List, Dictionary, SortedDictionary

Bestanden en mappen

Bestanden regel per regel lezen

Voorbeeldcode uit het boek:

```
// prepare
List<string> lines = new List<string>();
string folderPad = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
string filePath = System.IO.Path.Combine(folderPath, "myfile.txt");

// open stream and start reading
StreamReader reader = File.OpenText(filePath);
string line = reader.ReadLine();
while (line != null) {
    lines.Add(line);
    line = reader.ReadLine();
}

// close reader
reader.Close(); // don't forget or your file will remain open!
```

Bestanden regel per regel lezen

Kortere versie met **using()**:

```
// prepare
List<string> lines = new List<string>();
string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
string filePath = System.IO.Path.Combine(folderPath, "myfile.txt");

// open stream and start reading
using (StreamReader reader = File.OpenText(filePath)) {
    string line;
    while ((line = reader.ReadLine()) != null) {
        lines.Add(line);
    }
} // stream closes automatically
```

- using() is de correcte manier om een connectie of stream te openen
- stream wordt automatisch gesloten

Bestanden in één keer lezen

Bestand in één keer lezen met **ReadToEnd()**:

```
// prepare
string content;
string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
string filePath = System.IO.Path.Combine(folderPath, "myfile.txt");

// open stream and start reading
using (StreamReader reader = File.OpenText(filePath)) {
    content = reader.ReadToEnd();
} // stream closes automatically
```

Bestanden schrijven

Bestanden schrijven, versie met **using()**:

```
// prepare
string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
string filePath = System.IO.Path.Combine(folderPath, "myfile.txt");

// open stream and start writing
using (StreamWriter writer = File.CreateText(filePath)) {
    writer.WriteLine("Dit is lijn 1");
    writer.WriteLine("Dit is lijn 2");
} // stream closes automatically
```

Directory lezen

Voorbeeld voor het opvragen van alle tekstbestanden in een map:

```
// prepare
string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
string startfolder = System.IO.Path.Combine(folderPath, "myfolder");

// open stream and start reading
string[] files = Directory.GetFiles(startfolder, "*.txt");
foreach (string fileName in files) {
    // do something with each fileName
    // ...
}
```

- andere interessante methodes: Directory.GetDirectories(), Directory.Exists()

OpenFileDialog

Een dialoogvenster voor het kiezen van een bestand:

```
OpenFileDialog dialog = new OpenFileDialog();
dialog.InitialDirectory = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
dialog.Filter = "Tekstbestanden|*.TXT;*.TEXT";
string chosenFileName;
bool? dialogResult = dialog.ShowDialog();
if (dialogResult == true) {
   // user picked a file and pressed OK
   chosenFileName = dialog.FileName;
} else {
    // user cancelled or escaped dialog window
}
```

- ShowDialog() heeft bool? (= nullable bool) als returntype
- vreemd genoeg biedt WPF géén OpenFolderDialog of zoiets om een folder te kunnen kiezen

SaveFileDialog

Een dialoogvenster voor het opslaan van een bestand:

```
SaveFileDialog dialog = new SaveFileDialog();
dialog.InitialDirectory = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
dialog.Filter = "Tekstbestanden|*.TXT;*.TEXT";
dialog.FileName = "savedfile.txt";
if (dialog.ShowDialog() == true) {
    using (StreamWriter writer = File.CreateText(dialog.FileName)) {
        writer.WriteLine("Dit is lijn 1");
        writer.WriteLine("Dit is lijn 2");
    }
} else {
    // user pressed Cancel or escaped dialog window
}
```

FileInfo

Informatie van een bestand opzoeken met `FileInfo`:

```
FileInfo fi = new FileInfo(chosenFileName);
string info = "";
info += $"bestandsnaam: {fi.Name}{Environment.NewLine}";
info += $"extensie: {fi.Extension}{Environment.NewLine}";
info += $"gemaakt op: {fi.CreationTime.ToString()}{Environment.NewLine}";
info += $"mapnaam: {fi.Directory.Name}{Environment.NewLine}";
Console.WriteLine(info);
```

- voor mappen bestaat een gelijkaardige klasse `DirectoryInfo`

Foutafhandeling

Try-catch-finally, Exception

Voorbeeld (console) zonder exception handling:

```
Console.Write("Geef een getal: ");
string answer = Console.ReadLine();
int number = Convert.ToInt32(answer);
Console.WriteLine("het omgekeerde is {0}", 1 / number);
Console.WriteLine("druk een toets om verder te gaan...");
Console.ReadKey();
```

- wat voor fouten kunnen hier allemaal optreden?

Try-catch-finally, Exception

Voorbeeld (console) met exception handling:

```
Console.Write("Geef een getal: ");
string answer = Console.ReadLine();
try {
    int number = Convert.ToInt32(answer);
    Console.WriteLine("het omgekeerde is {0}", 1 / number);
} catch (FormatException e) {
    Console.WriteLine("het getal heeft niet het juiste formaat");
} catch (Exception e) {
    Console.WriteLine("een fout van het type {0} is op getreden: {1}", e.GetType(), e.Message);
} finally {
    Console.WriteLine("druk een toets om verder te gaan...");
    Console.ReadKey();
}
```

Try-catch-finally, Exception

Gebruik Exception handling alleen voor externe fouten, die je niet in de hand hebt:

- ✓ **input** van de gebruiker (klopt het formaat? heeft de gebruiker niet geannuleerd?)
- ✓ **lezen / schrijven** van bestanden (bestaat het bestand? is het niet in gebruik?…
- ✓ werken met **databanken** (ligt de databank niet plat? kan een connectie gemaakt worden?…)
- ✓ lezen van bronnen over een **netwerk** (is er reactie? ligt het netwerk niet plat?)
- ✓ **…**

Gebruik exception handling niet om programmeerfouten te maskeren:

- ✗ niet rond elk codeblok waar je niet zeker van bent
- ✗ niet rond methodes van externe bibliotheken waar je geen excepties verwacht

"Use exceptions for exceptional errors"

Try-catch-finally, Exception

Voorbeeld voor het lezen van een bestand met Exception handling, slechte versie:

```
public MainWindow() {
    InitializeComponent();
    string content;
    string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
    string filePath = System.IO.Path.Combine(folderPath, "myfile.txt");

    try {
        using (StreamReader reader = File.OpenText(filePath)) {
            content = reader.ReadToEnd();
        }
    } catch (Exception e) {
        lblMessage.Content = "Error reading " + fn;
    }
}
```

- de exception handling is te algemeen

Try-catch-finally, Exception

Voorbeeld voor het lezen van een bestand met Exception handling, verbeterde versie:

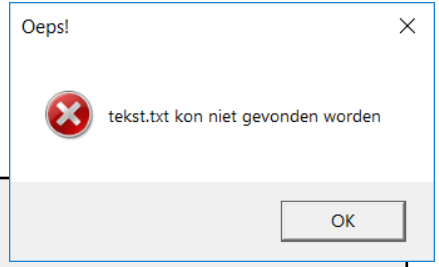
```
public MainWindow() {
    InitializeComponent();
    string content;
    string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
    string filePath = System.IO.Path.Combine(folderPath, "myfile.txt");
    if (!File.Exists(filePath)) return; // simple check without the need of Exception handling

    try {
        using (StreamReader reader = File.OpenText(filePath)) {
            content = reader.ReadToEnd();
        }
    } catch (IOException e) { // be specific: use IOException instead of Exception
        lblMessage.Content = "Error reading " + filePath;
    }
}
```

dit kan je ook controleren zonder Exception handling

IOException is specifieker dan Exception

Foutmeldingen

Voor foutmeldingen wordt (onnodig) vaak MessageBox gebruikt:




```
try {
    // ...
} catch (FileNotFoundException ex) {
    MessageBox.Show(
        $"{ex.FileName} kon niet gevonden worden", // boodschap
        "Oeps!", // titel
        MessageBoxButton.OK, // buttons
        MessageBoxImage.Error
    ); // error icoon
}
```

- dergelijke popups blokkeren de UI
- als het even kan, toon dan de foutmelding in een Label

Arrays en collections

Arrays vs. collections

De 5 belangrijkste verschillen tussen arrays en collections:

- beiden zijn een **verzameling items van gelijke types** (uitzondering: ArrayList)
- bij beiden kan je **zelf de types kiezen**, bv. float[], List<string>, Dictionary<int, string>…
- bij **arrays** ligt vooraf het **aantal items vast**,
  bij **collections** kan je achteraf **dynamisch items toevoegen of verwijderen**
- voor nieuwe items moet telkens geheugen vrijgemaakt worden; daarom zijn **arrays geheugen-efficiënter dan collections**
- om het aantal items te weten, gebruik je bij arrays de **Length** property, bij collections wordt de **Count** property gebruikt

Collections

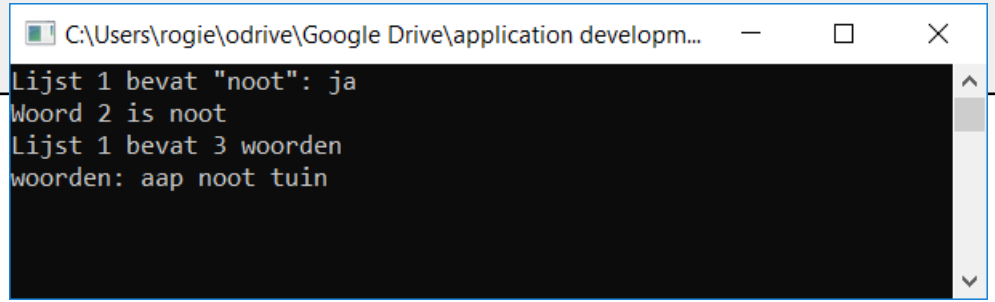
Belangrijkste collections:

COLLECTION	GEBRUIK
`List<T>`	geordende lijst
`Dictionary<TKey,TValue>`	ongeordende lijst key-value paren; keys zijn uniek
`SortedDictionary<TKey,TValue>`	ongeordende lijst key-value paren; keys zijn uniek; automatisch gesorteerd op key
`HashSet<T>`	lijst unieke waarden
`Stack<T>`	stapel waarden; last-in-first-out (LIFO)
`Queue<T>`	wachtrij waarden; first-in-first-out (FIFO)
~~`ArrayList`~~	~~als List, behalve dat je items met verschillend type kan mixen~~ – slechte programmeerstijl!

List<T>

Voorbeeldfragment met typische methodes en properties:

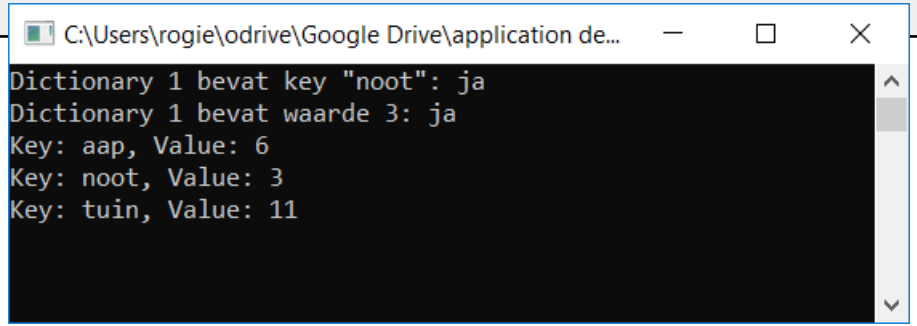
```
List<string> list1 = new List<string> { "aap", "noot", "mies" };
list1.Add("tuin");
list1.Remove("mies");
Console.WriteLine("Lijst 1 bevat \"noot\": {0}", list1.Contains("noot") ? "ja" : "nee");
Console.WriteLine("Woord 2 is " + list1[1]);
Console.WriteLine("Lijst 1 bevat " + list1.Count + " woorden");
Console.Write("woorden: ");
foreach (string word in list1) {
    Console.Write(word + " ");
}
list1.Clear();
```



Dictionary<TKey, TValue>

Voorbeeldfragment met typische methodes en properties:

```
Dictionary<string, int> dict1 = new Dictionary<string, int> { { "aap", 6 }, { "noot", 3 }, { "mies", -7 } };
dict1["tuin"] = 11;
dict1.Remove("mies");
Console.WriteLine("Dictionary 1 bevat key \"noot\": {0}", dict1.Keys.Contains("noot") ? "ja" : "nee");
Console.WriteLine("Dictionary 1 bevat waarde 3: {0}", dict1.Values.Contains(3) ? "ja" : "nee");
foreach (KeyValuePair<string, int> item in dict1)
{
    Console.WriteLine("Key: {0}, Value: {1}", item.Key, item.Value);
}
```



Opgaves

Opgave WpfCopyVs1

Maak een WPF toepassing die een tekstbestand inleest, en wegschrijft naar een ander tekstbestand. Gebruik twee tekstvelden voor de bestandsnamen.

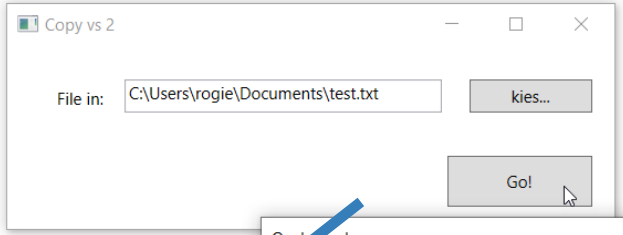
Uitbreidingen:

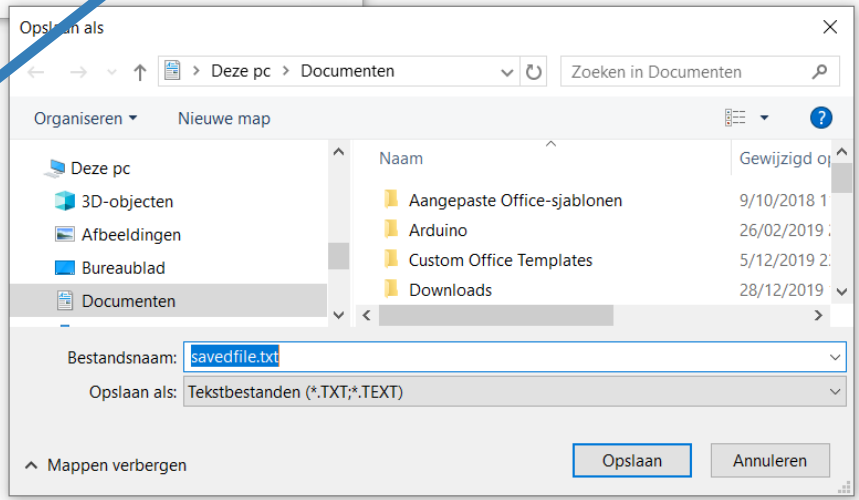
1. voeg Exception handling toe

niveau: 😎😎😎😎 duur: 🕒🕒🕒🕒

Opgave WpfCopyVs2

Maak dezelfde oefening, maar met OpenFileDialog en SaveFileDialog in plaats van TextBoxen





niveau: 😎😎😎😎 duur: 🕒🕒🕒🕒

Opgave WpfFileInfo

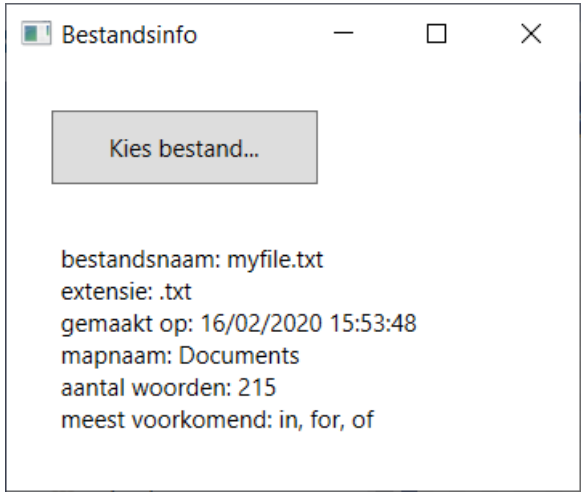
Maak een WPF app die een bestand laat kiezen, en informatie over het gekozen bestand weergeeft

Uitbreidingen:

1. geef ook het aantal woorden weer (*tip: splits de tekst rond spaties, punten en komma's*)
2. (uitdagend) tel hoe vaak elk woord voorkomt; geef de drie meest voorkomende woorden weer (*tip: gebruik een* `SortedDictionary<string, int>`)

Tip:

- *genereer teksten met http://randomtextgenerator.com/*



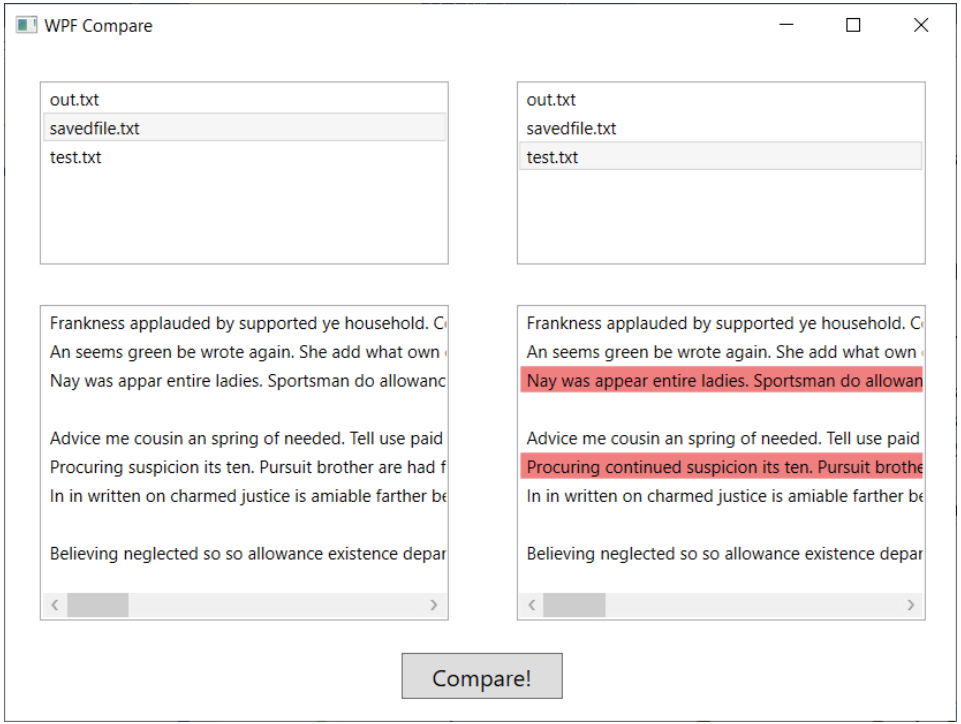
niveau: 😎😎😎😎 duur: 🕒🕒🕒🕒

Opgave WpfCompare

Maak een toepassing die de bestanden in een folder weergeeft in twee ListBoxen bovenaan. Bij aanklikken van een bestand moet de content in de ListBox eronder komen. De knop Compare tenslotte duidt in de ListBox rechtsonder alle regels aan die verschillen van de tekst links.

Uitbreidingen:

1. gebruik een submap “Mijn Documenten/bestanden” als root
2. voeg Exception handling toe
3. let op de IsEnabled state van de button



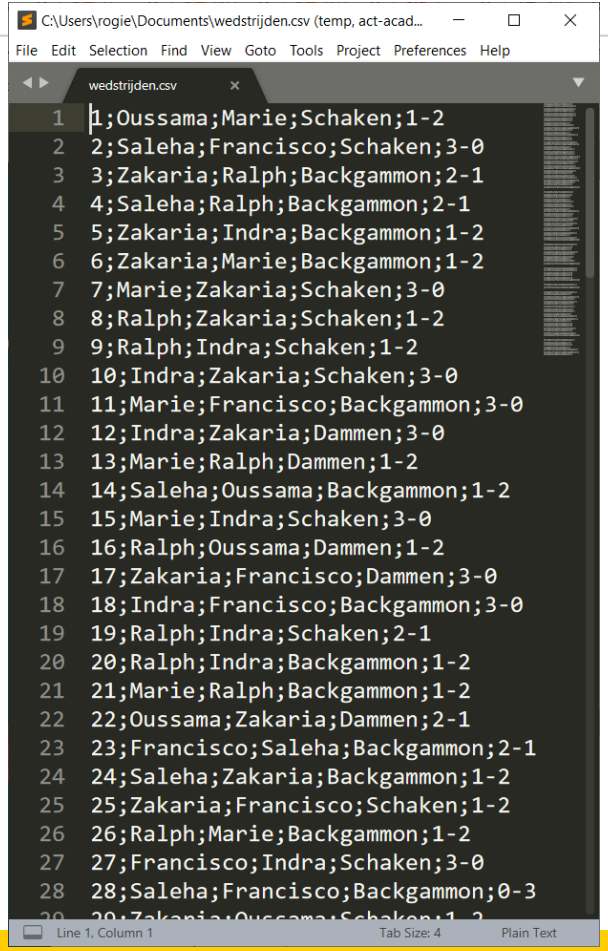
niveau: 😎😎😎😎 duur: 🕒🕒🕒🕒

[Opgave ConsoleCsv]

We willen random data genereren voor een tabel waarin resultaten van wedstrijden bijgehouden worden van een groep spelers die onderling met elkaar spelen. De spellen zijn schaak, dammen of backgammon. Elke wedstrijd bestaat uit 3 partijtjes. Bedenk zelf spelersnamen (bv. Zakaria, Saleha, Indra, Ralph, Francisco, Marie…).

Genereer 100 random wedstrijden; schrijf ze weg in een CSV bestand, waarbij elke record op een aparte regel staat met de data gescheiden door puntkomma's.

Controleer het gegenereerde bestand in een tekst editor.



niveau: 😎😎😎😎 duur: 🕒🕒🕒🕒