1.0.0

H10: Gevorderde tekstverwerking

Voorstelling van tekst

Wat is Unicode?

Alle gegevens in het geheugen van je computer worden voorgesteld als een opeenvolging van binaire cijfers. Dat is gewoon veruit de handigste manier om computers te bouwen. Helaas is het niet de handigste voorstelling. Als je een bestand opent in een text editor, zie je bijvoorbeeld niet `001000111000110101010101110` (enzovoort), maar wel `hallo wereld`, of . Dat komt omdat er afspraken gemaakt zijn rond welke reeksen bits welke karakters voorstellen. Die afspraken zijn doorheen de tijd ontstaan. Een van de simpelste is 7-bit ASCII, waarbij één byte gebruikt werd om een karakter voor te stellen. 7 bits bepaalden welk karakter werd voorgesteld (er waren er maximum 128), de achtste diende om te controleren dat er geen datacorruptie was opgetreden.

Naarmate computers in meer regio's gebruikt werden, werd duidelijk dat 128 karakters niet volstonden om elke taal van de wereld digitaal voor te stellen. Laat staan de emoji die we vandaag hebben. Als oplossing voor dat probleem is de **Unicode standaard** vastgelegd. Deze koppelt, eenvoudig gesteld, aan een hele grote reeks getallen een overeenkomstig symbool.

Het is duidelijker als je dit in de praktijk ziet. Volgende voorbeelden komen van unicode-table.com:

f Latin Small Letter F

f

U+0066

Click to copy and paste symbol

f Copy

Technical information

Name	Latin Small Letter F
Unicode number	U+0066
HTML-code	f
CSS-code	\0066
Block	Basic Latin
Uppercase	F
Unicode version:	1.1 (1993)
Alt code:	Alt 102

De letter f is symbool nummer 102. Dit is omdat het "Unicode number" hexadecimaal wordt uitgedrukt. De HTML-code is decimaal.

♻ Black Universal Recycling Symbol Emoji

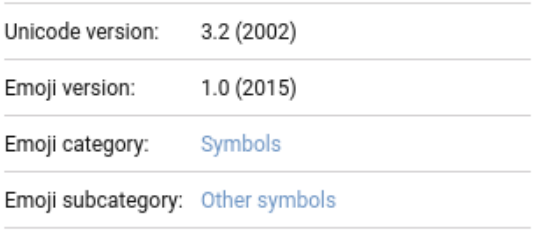
U+267B

Click to copy and paste symbol

♻ Copy

Technical information

Name	Black Universal Recycling Symbol
Unicode number	U+267B
HTML-code	♻
CSS-code	\267B
Block	Miscellaneous Symbols



Unicode version:	3.2 (2002)
Emoji version:	1.0 (2015)
Emoji category:	Symbols
Emoji subcategory:	Other symbols

Het symbool voor recyclage is symbool nummer 9851. Zelfde uitleg als voor de letter f.

Wat is een karakterset?

De Unicode standaard is niet genoeg om tekst voor te stellen in de praktijk. Je moet immers ook weten hoe je bits moet samen nemen. Anders gezegd: maak je groepjes van één byte, van twee bytes, van drie bytes,... om zo de getallen te vinden die symbolen voorstellen? Er zijn meerdere antwoorden op die vraag en in de praktijk gaat het eerder om combinaties. Zo kan je zeggen dat je méestal 1 byte gebruikt, maar soms meer als je aan één byte niet genoeg hebt (omdat je een symboolnummer groter dan 255 hebt). Hoe dat precies werkt, laten we hier achterwege. Maar we onthouden wel dat we hier per bestand een afspraak rond moeten maken. Deze afspraak noemen we de **karakterset** of **encodering** van ons bestand. C#-programma's veronderstellen intern bijvoorbeeld (net als JavaScript programma's) dat je **gewoonlijk** twee bytes gebruikt om een karakter voor te stellen, maar ze hebben een achterpoortje waardoor je meer dan twee bytes kan gebruiken. Deze encodering heet **UTF-16**. Dit betekent niet dat je in C# niet met bestanden in een andere karakterset kan werken of zelfs dat je .cs-files UTF-16 moeten gebruiken. Maar het bepaalt wel hoe we speciale karakters kunnen noteren via escape symbolen.

Hoe stellen we een individueel karakter voor?

Om één karakter voor te stellen, gebruiken we het type `char`. Je kan hier een waarde aan toekennen door ze tussen enkele quotes te plaatsen, als volgt:

```
char letterF = 'f';
char recycle = ' ';
```

Tegelijkertijd kan je er mee rekenen. Dan ga je eigenlijk met het Unicode nummer van dat symbool werken. Om terug toe te kennen aan een karakter, moet je wel weer casten. Bijvoorbeeld:

```
char letterG = (char) ('f' + 1);
```

Dit werkt, want in de Unicode standaard komt de g na de f.

Je kan karakters ook noteren via hun Unicode nummer, door een "Unicode escape" te gebruiken. Hierbij noteer je eerst een backslash, dan een `u` en dan het Unicode nummer in **hexadecimale** notatie. Als je dit print op je scherm, zie je net hetzelfde als met de notatie hierboven. Het voordeel is dat je het symbool niet op je toetsenbord moet hebben staan.

```
char recycle = '\u267B';
```

Er zijn ook "gewone" escapes voor tekens die je vaak nodig hebt. Deze zijn onder andere `'\n'` (newline) en `'\t'` (tab). `'\u000A'` en `'\u0009'` werken ook, maar zijn wat lastiger te onthouden. Let wel op: als je een backslash wil gebruiken zonder dat deze een escape sequentie activeert, moet je de backslash zelf escapen. Dus `\\` stelt eigenlijk het karakter `\` voor.

Wat hier verder opvalt: we hebben maximum 4 hexadecimale cijfers, dus in theorie 65536 karakters. Het is waar dat we maar 65536 waarden van het type `char` hebben. Maar binnenin een `string` kan je ook karakters noteren en in deze situatie zijn er combinaties van karakters die als één geheel gezien worden. We noemen deze **surrogate pairs** en ze geven weer als één karakter. Bijvoorbeeld:

```
Console.WriteLine("Dit is één symbool: \ud835\udcb3");
```

Je hoeft de werking van surrogate pairs voor deze cursus niet in detail te kennen, maar als je ooit een applicatie schrijft die emoji,... bevat, vind je hier de details.

> Je kan emoji gebruiken in je programmatekst, maar je moet de terminal ook verwittigen dat je dit wil doen. Hiervoor zet je de encodering **van de terminal** via `Console.OutputEncoding = Encoding.UTF8`.

Verbatim strings

Door een `@` (verbatim character) voor een string te plaatsen zeggen we concreet: "de hele string die nu volgt moet je beschouwen zoals hij er staat. Je mag alle escape karakters negeren en er mogen line breaks voorkomen in deze string."

Dit wordt vaak gebruikt om een filepath iets leesbaarder te maken.

- Zonder verbatim: `string path= "c:\\Temp\\myfile.txt";`
- Met verbatim: `string path= @"c:\Temp\myfile.txt";`

> Bovenstaande strings zijn **identiek** voor C#! De @ geeft geen enkele extra functionaliteit, maar zorgt gewoon dat we dezelfde tekst (meerbepaald: backslashes en line breaks) wat makkelijker kunnen intypen.

Volgende stukken code doen dus hetzelfde:

```
string tekst = @"Back

\

Slash";
Console.WriteLine(tekst);
```

```
string tekst = "Back\n\n\\\n\nSlash";
Console.WriteLine(tekst);
```

Intern is er géén verschil. Voor de `WriteLine` wordt uitgevoerd, worden beide waarden in dezelfde voorstelling gegoten. Maar in dit geval is de eerste versie wel veel makkelijker te lezen voor de programmeur.

Interpolatie met formattering

Via stringinterpolatie schrijf je je string min of meer zoals hij er uiteindelijk moet uitzien, maar vervang je de niet-letterlijke delen door geformatteerde waarden. Dit levert een goed leesbaar resultaat.

Door het `$`-teken **VOOR** de string te plaatsen geef je aan dat alle delen in de string die tussen accolades staan als code mogen beschouwd worden. Een voorbeeld maakt dit duidelijk:

```
string name = "Finkelstein";
int age = 17;
string result = $"Ik ben {name} en ik ben {age} jaar oud.";
```

In dit geval zal dus de inhoud van de variabele `name` tussen de string op de plek waar nu `{name}` staat geplaatst worden. Idem voor `age`. Dit mag, zelfs al is `age` geen string: hetgeen tussen de accolades staat, wordt altijd intern omgezet naar een string voor het in het resultaat wordt geplaatst. Dit gebeurt via de methode `ToString`, die voor elk soort data bestaat. Dus eigenlijk wordt het achter de schermen:

```
string result = $"Ik ben {name.ToString()} en ik ben {age.ToString()} jaar oud.";
```

Berekeningen doen bij string interpolatie

Je mag eender welke *expressie* tussen de accolades zetten bij string interpolation. Een expressie is iets dat je kan uitrekenen en dat je een resultaat oplevert. Bijvoorbeeld:

```
string result = $"Ik ben {name} en ik ben {age+4} jaar oud.";
```

Alle expressies tussen de accolades zullen eerst uitgevoerd worden voor ze tussen de string worden geplaatst. De uitvoer wordt nu dus: `Ik ben Finkelstein en ik ben 17 jaar oud.`

Eender welke expressie is toegelaten, dus je kan ook complexe berekeningen of zelfs andere methoden aanroepen:

```
string result = $"Ik ben {age*age+(3%2)} jaar oud.";
```

Mooier formatteren

Bij stringinterpolatie kan je ook bepalen hoe de te tonen variabelen en expressies juist weergegeven moeten worden. Je geeft dit aan door na de expressie, binnen de accolades, een dubbelpunt te plaatsen gevolgd door de manier waarop moet geformatteerd worden:

Wil je bijvoorbeeld een kommagetal tonen met maar 2 cijfers na de komma dan schrijf je:

```
double number = 12.345;
Console.WriteLine($"{number:F2}");
```

Hier stelt `2` het aantal cijfers na de komma voor. Je kan zelf kiezen om meer of minder cijfers te tonen door het getal aan te passen. Dit voorbeeld is gelijkaardig aan `Console.WriteLine(Math.Round(number,2))`, maar niet identiek. Dat komt omdat de geformatteerde string **verplicht** twee cijfers na de komma toont, zelfs als die niets veranderen aan de waarde van het getal. Als je een double zonder formattering weergeeft, worden nullen achteraan niet getoond. Dus voor getallen als `12` of `15.1` of `4.999` (met minder dan twee cijfers na de komma **na afronding**) maakt het een verschil.

Nog enkele nuttige vormen:

- D5: geheel getal bestaande uit 5 cijfers (`123` wordt `00123`) (werkt enkel op gehele getallen en je kan het getal 5 vervangen door een positieve waarde naar keuze)
- C: geldbedrag (`12,34` wordt € 12,34: teken van valuta afhankelijk van instellingen pc)

Alle format specifiers staan hier opgelijst. Voor deze cursus volstaat het dat je deze specifiers kent voor de verschillende soorten data.

Je kan je data ook een vaste breedte en uitlijning geven. Dit doe je door meteen na de waarde in kwestie een komma te plaatsen, met daarna de gewenste breedte. De tekst wordt dan rechts gealigneerd. Je kan ook een minteken gebruiken en dan wordt de tekst links gealigneerd. Bijvoorbeeld:

```
Console.WriteLine($"{"hallo",20} wereld");
Console.WriteLine($"{"hallo",-20} wereld");
```

Dit levert:

```
               hallo wereld
hallo                wereld
```

Dus de tekst `"hallo"` wordt opgevuld totdat hij in totaal 20 karakters telt. Elke regel telt 27 karakters, want `" wereld"` is niet mee geformatteerd, maar is gewoon achter de geformatteerde tekst gezet.

Manueel formatteren

Formattering is een handig hulpmiddel, maar het zal niet altijd alle scenario's dekken. In het voorbeeld hierboven is het bijvoorbeeld **verplicht** een constante waarde te gebruiken. Je kan geen variabele invullen waar `20` staat. Het is bijvoorbeeld ook niet mogelijk tekst te centeren (in plaats van links of rechts uit te lijnen). Dat hoeft geen probleem te zijn: je kan altijd een methode schrijven die tekst op de gewenste manier omzet.

Ingebouwde methodes voor formattering

Via `PadLeft` en `PadRight` kan je tekst opvullen tot de gewenste lengte met andere karakters dan een spatie. Bijvoorbeeld, voor hetzelfde effect als eerder:

```
Console.WriteLine($"{"hallo".PadLeft(20)} wereld");
Console.WriteLine($"{"hallo".PadRight(20)} wereld");
```

Zoals vaak lever je hiermee wat gemak in voor extra flexibiliteit. Je kan namelijk opvullen met een karakter naar keuze, tot een variabele lengte:

```
Console.WriteLine("Hoe breed wil je de string?");
int breedte = Convert.ToInt32(Console.ReadLine());
Console.WriteLine($"{"hallo".PadLeft(breedte,'!')} wereld");
Console.WriteLine($"{"hallo".PadRight(breedte,'!')} wereld");
```

Eigen methodes voor formattering

Als er geen mehode is om een string juist te tonen, kan je er altijd zelf een maken. Je zorgt dat de string en extra opties zoals de gewenste breedte parameters zijn en je geeft de geformatteerde string terug als resultaat. Er is bijvoorbeeld geen methode om tekst in te korten tot een maximale lengte, maar je kan ze zelf schrijven als volgt:

```
static string Trunkeer(string input, int lengte) {
    return input.Substring(0, Math.Min(input.Length, lengte));
}
```

Je zou bijvoorbeeld ook een methode kunnen schrijven om tekst centraal te aligneren.

Werken met arrays van strings

Split

De `Split` methode laat toe een string te splitsen op een bepaald teken. Het resultaat is steeds een **array van strings**.

```
string data= "12,13,20";
string[] gesplitst= data.Split(',');

for(int i=0; i<gesplitst.Length; i++)
{
    Console.WriteLine(gesplitst[i]);
}
```

Uiteraard kan je dit dus gebruiken om op eender welk `char` te splitsen en dus niet enkel op een `','` (komma).

Dit is bijzonder nuttig voor het verwerken van bestanden met gestructureerde data, bijvoorbeeld CSV-bestanden. Dit zijn bestanden waarin elke regel een groepje verwante gegevens voorstelt, gescheiden via een afgesproken symbool zoals `,`. Deze files worden vaak gebruikt als exportformaat voor spreadsheets en databasetabellen.

Join

Via `Join` kunnen we array van `string` terug samenvoegen. Het resultaat is een nieuwe `string`.

Volgende voorbeeld zal de eerder array van het vorige voorbeeld opnieuw samenvoegen maar nu met telkens een `;` tussen iedere string:

```
string joined = String.Join(";", gesplitst);
```

Input en output van tekstbestanden

Text files lezen algemeen

De `System.IO` namespace bevat tal van nuttige methoden en klassen om met bestanden te werken. Om gebruik hiervan te maken plaats je bovenaan je file:

```
using System.IO;
```

Via `System.File.ReadAllLines()` kunnen we een tekstbestand uitlezen. De methode geeft een array van string terug. Per lijn die eindigt met een newline (onder Windows `\r`) zal een nieuwe string aan de array toegevoegd worden.

```
// wat dus ook kan: File.ReadAllLines("C:\\mypoem.txt")
string[] lines = File.ReadAllLines(@"C:\mypoem.txt");
for (int i = 0; i < lines.Length; i++)
{
    Console.WriteLine(lines[i]);
    Console.WriteLine("----------");
}
```

> Deze methode probeert automatisch de juiste encodering van het bestand te detecteren. Dit zal niet voor elke encodering werken. Als je vreemde karakters ziet, zal je moeten uitzoeken wat de encodering is en uitdrukkelijk aangeven dat je deze wenst te gebruiken.

CSV uitlezen

Dankzij `ReadAllLines` en `Split` hebben we nu alle bouwstenen om eenvoudig een csv-bestand te verwerken.

Stel je voor dat een bestand `soccer.csv` volgende tekst bevat:

```
Dams;Tim;1981
Hamdaoui;Mounir;1984
Stoffels;José;1950
```

Volgende code zal dit bestand uitlezen en de individuele data op het scherm tonen:

```
string[] lijnen = File.ReadAllLines(@"C:\soccerstars.csv");
for (int i = 0; i < lijnen.Length; i++)
{
    string[] kolomwaarden = lijnen[i].Split(';');
    Console.WriteLine($"Voornaam speler {i}= {kolomwaarden[1]}" );
    Console.WriteLine($"Achternaam speler {i}= {kolomwaarden[0]}");
    Console.WriteLine($"Geboortejaar speler {i}= {kolomwaarden[2]}");
}
```

Text files schrijven

Je kan tekst uit een bestand lezen, maar uiteraard kan je ook naar een bestand wegschrijven. De 2 eenvoudigste manieren zijn:

- `File.WriteAllText` : deze gebruik je als je 1 enkele string wilt wegschrijven. Typisch bevat deze tekst newline karakters.
- `File.WriteAllLines` : deze is de omgekeerde van `ReadAllLines()` en zal een array van strings wegschrijven. Elke string in de array wordt geschreven als een regel, dus het is alsof je de strings eerst verbindt met een newline via `String.Join` en dan wegschrijft met `File.WriteAllText` .

Een voorbeeld:

```
string[] stringArray = new string[]
    {
        "cat",
        "dog",
        "arrow"
    };


File.WriteAllLines("C:\\file.txt", stringArray);
```

Opgelet met het schrijven naar bestanden: dit zal onherroepelijk het target bestand overschrijven. .Gebruik `if(File.Exists(pathtofile))` om te controleren of een bestand bestaat of niet. Eventueel kan je dan aan de gebruiker bevestiging vragen of je deze effectief wilt overschrijven.

Oefeningen

Inleiding

Al deze oefeningen maak je als statische methoden van een klasse `GevorderdeTekstverwerking`. Je kan elk van deze methoden uitvoeren via een keuzemenu, zoals bij vorige hoofdstukken.

H10-Som-van-getallen

Leerdoelen

- Werken met .Split en .Join
- Itereren over array

Functionele analyse

Je vraagt de gebruiker een aantal getallen gescheiden door ';' in te geven. je laat vervolgens de som van deze getallen zien.

Technische analyse

Schrijf in de klasse `GevorderdeTekstverwerking` een methode `SomVanGetallen`. Deze vraagt de gebruiker een aantal getallen te geven gescheiden door een ';'. Je leest de invoer in, splitst de invoer in getallen, maakt de som en laat die zien. Let op, het aantal getallen mag de gebruiker zelf bepalen.

Voorbeeldinteractie

```
Gelieve getallen gescheiden door ';'  in te geven
12;4;78;100;6;8
12+4+78+100+6+8 = 208
```

H10-Centraal-Aligneren-Tekst

Leerdoelen

- Formateren van tekst
- String interpolatie
- Methode oproep

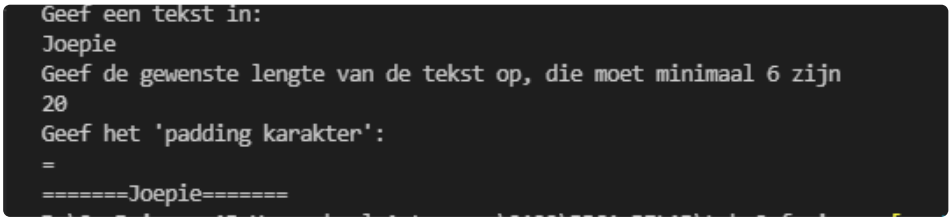
Functionele analyse

Je vraagt de gebruiker een tekst in te geven. Vervolgens vraag je de gewenst breedte van de tekst. Je laat de tekst dan centraal gealigneerd aan de gebruiker zien.

Technische analyse

Schrijf in de klasse `GevorderdeTekstverwerking` een methode `CentraalAlignerenTekst`. Deze vraagt de gebruiker een tekst in te geven en een gewenste lengte (minimaal de lengte van de tekst - controle!). Maak een tweede methode `CentraalAligneren` Vervolgens wordt de tekst centraal gealigneerd getoond aan de gebruiker. De gebruiker kan het padding karakter ook kiezen.

Voorbeeldinteractie



H10-KerstinkopenNetjes

Leerdoelen

- stringformattering
- string methodes
- gebruik van karakters

Functionele analyse

De eerdere oefening Kerstinkopen kan wat mooier gepresenteerd worden. Ze zou ook ingezet moeten kunnen worden in regio's waar de euro niet gebruikt wordt.

Technische analyse

Maak een kopie van Kerstinkopen in deze klasse.

Pas door middel van stringformattering en string methodes de code aan, zodat:

- Een scheidingslijn getekend wordt onder "Info over je aankopen", die net breed genoeg is. Als we later deze hoofding veranderen (bijvoorbeeld naar "Informatie over je aankopen") moet de lijn mee groeien of krimpen.
- Alle bedragen netjes onder elkaar staan in één kolom. Hiervoor mag je veronderstellen dat de berichten op de linkerkant maximum 25 karakters in beslag zullen nemen.
- Het symbool voor de munteenheid van de gebruiker vanzelf gebruikt wordt. Dit kan dus € zijn maar ook $ of £ of iets anders.

> ⚠ Als je het symbool voor de munteenheid niet te zien krijgt in de terminal van Visual Studio, ligt dat niet aan jouw code. Voer je code dan uit via het commando `dotnet run` in Git bash.

Voorbeeldinteractie

> ⓘ Deze voorbeeldinteractie gebruikt de List versie van Kerstinkopen. Maak de oefening met de array versie. Het gaat hem hier over de weergave onderaan en de aanpassingen daar zijn dezelfde voor beide versies.

```
Wat is het budget voor je kerstinkopen?
100
Wat is de prijs van cadeau 1?
20
Wat is de prijs van cadeau 2?
3
Wat is de prijs van cadeau 3?
7
Wat is de prijs van cadeau 4?
33
Wat is de prijs van cadeau 5?
0
Info over je aankopen:
======================
Totaal bedrag:                $63.00.
Duurste cadeau:               $33.00.
Goedkoopste cadeau:           $3.00.
Gemiddelde prijs:             $15.75.
```

H10-TextCellPersistent

Leerdoelen

- Gebruik van input en output van tekstbestanden

Functionele analyse

We willen onze gemaakte TextCells ook kunnen opslaan en weer openen.

Technische analyse

Maak in je oefeningen map een nieuwe map aan TextCell. In deze map zullen de TextCell bestanden bewaard worden. Je kan in je applicatie ook een relatief pad opgeven (dus niet c:\...). Dit relatief pad gaat vertrekkende van de map waar je nu op werkt een map zoeken (dit is wat kort door de bocht, de werkelijkheid is een stuk ingewikkelder maar voor nu volstaat dit). Het relatieve pad dat wij gebruiken is "./TextCell/filenaam.aptx).

Kopieer het bestand TextCell.cs naar TextCellPersistent.cs. Maak gerbuik van de in de theorie aangehaalde methodes File.ReadAllLines en File.WriteAllLines. Test bij opslaan van het bestand of het bestand reeds bestaat en bij laden van het bestand of het bestand sowieso bestaat.

Voorbeeldinteractie

```
Hoe veel cellen telt je spreadsheet "laad" om een bestaande textcel te laden
8
|A         |B         |C         |D         |E         |F         |G         |H         |
|          |          |          |          |          |          |          |          |
Welke cel wil je wijzigen - "bewaar" om je TextCel bestand op ts slaan, "stop" om te stoppen?
D
Wat wil je hier invullen?
5
|A         |B         |C         |D         |E         |F         |G         |H         |
|          |          |          |5         |          |          |          |          |
Welke cel wil je wijzigen - "bewaar" om je TextCel bestand op ts slaan, "stop" om te stoppen?
A
Wat wil je hier invullen?
D+3
|A         |B         |C         |D         |E         |F         |G         |H         |
|D+3       |          |          |5         |          |          |          |          |
Welke cel wil je wijzigen - "bewaar" om je TextCel bestand op ts slaan, "stop" om te stoppen?
A
Wat wil je hier invullen?
=D+3
|A         |B         |C         |D         |E         |F         |G         |H         |
|8         |          |          |5         |          |          |          |          |
Welke cel wil je wijzigen - "bewaar" om je TextCel bestand op ts slaan, "stop" om te stoppen?
B
Wat wil je hier invullen?
=D+A
|A         |B         |C         |D         |E         |F         |G         |H         |
|8         |13        |          |5         |          |          |          |          |
Welke cel wil je wijzigen - "bewaar" om je TextCel bestand op ts slaan, "stop" om te stoppen?
bewaar
Geef de naam van je TextCell bestand: TextCell1
Je TextCell bestand is bewaard.
|A         |B         |C         |D         |E         |F         |G         |H         |
|8         |13        |          |5         |          |          |          |          |
Welke cel wil je wijzigen - "bewaar" om je TextCel bestand op ts slaan, "stop" om te stoppen?
```

```
Hoe veel cellen telt je spreadsheet "laad" om een bestaande textcel te laden
laad
Geef je TextCell bestand (zonder extensie aptx)
aaa
Bestand niet gevonden!
Geef je TextCell bestand (zonder extensie aptx)
TextCell1
|A         |B         |C         |D         |E         |F         |G         |H         |
|8         |13        |          |5         |          |          |          |          |
Welke cel wil je wijzigen - "bewaar" om je TextCel bestand op ts slaan, "stop" om te stoppen?
E
Wat wil je hier invullen?
=A+B+D
|A         |B         |C         |D         |E         |F         |G         |H         |
|8         |13        |          |5         |26        |          |          |          |
Welke cel wil je wijzigen - "bewaar" om je TextCel bestand op ts slaan, "stop" om te stoppen?
bewaar
Geef de naam van je TextCell bestand: TextCell2
Je TextCell bestand is bewaard.
|A         |B         |C         |D         |E         |F         |G         |H         |
|8         |13        |          |5         |26        |          |          |          |
Welke cel wil je wijzigen - "bewaar" om je TextCel bestand op ts slaan, "stop" om te stoppen?
stop
```

H10-Pixels-Persistent

Functionele analyse

We wensen de kunstwerken die we in Pixels tekenen op te slaan en in te laden.

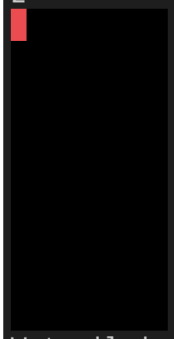
Technische analyse

Kopieer de code voor Pixels naar deze klasse.

Zorg ervoor dat je twee extra opties hebt in je tekenprogramma. Om een tekening op te slaan, moet je elke `ConsoleColor` casten naar een `int`. Dan kan je één rij pixels opslaan als één rij getallen, gescheiden door puntkomma. Om een tekening in te laden, doe je het omgekeerde.

Voorbeeldinteractie

```
Wat wil je doen?
1. Een pixel kleuren?
2. Afbeelding tonen?
3. Een afbeelding exporteren?
4. Een afbeelding inladen?
5. Stoppen?
2
```



```
Wat wil je doen?
1. Een pixel kleuren?
2. Afbeelding tonen?
3. Een afbeelding exporteren?
4. Een afbeelding inladen?
5. Stoppen?
3
Geef het volledige pad naar de locatie waar je wil opslaan, inclusief bestandsnaam.
C:\Users\ikke\mijnafbeelding.csv
Bestand geëxporteerd!
Wat wil je doen?
1. Een pixel kleuren?
2. Afbeelding tonen?
3. Een afbeelding exporteren?
4. Een afbeelding inladen?
5. Stoppen?
4
Geef het volledige pad van het bestand datje wil inladen.
C:\Users\ikke\mijnandereafbeelding.csv
Wat wil je doen?
1. Een pixel kleuren?
2. Afbeelding tonen?
3. Een afbeelding exporteren?
4. Een afbeelding inladen?
5. Stoppen?
2
```

```
Wat wil je doen?
1. Een pixel kleuren?
2. Afbeelding tonen?
3. Een afbeelding exporteren?
4. Een afbeelding inladen?
5. Stoppen?
```