

Afbeelding met tekst, Graphics, tekenfilm, schermopname

Door AI gegenereerde inhoud is mogelijk onjuist.

Dhr. Renaud Leroy

Campus impuls | VISO Mariakerke

Informaticawetenschappen

Inhoudsopgave

[Introductie 5](#_Toc210139674)

[Essentials: datatypes en variabelen 6](#_Toc210139675)

[Wat is C# 6](#_Toc210139676)

[Variabelen 6](#_Toc210139677)

[var 7](#_Toc210139678)

[Uitvoer (in console) 7](#_Toc210139679)

[Invoer (in console) 8](#_Toc210139680)

[Data-types 8](#_Toc210139681)

[String (basis) 8](#_Toc210139682)

[Oefeningen: string 9](#_Toc210139683)

[Int 10](#_Toc210139684)

[Float 11](#_Toc210139685)

[Converties 11](#_Toc210139686)

[Oefeningen 12](#_Toc210139687)

[Collecties 13](#_Toc210139688)

[Arrays 13](#_Toc210139689)

[List 13](#_Toc210139690)

[ForEach 14](#_Toc210139691)

[For 14](#_Toc210139692)

[Oefeningen 16](#_Toc210139693)

[String (uitgebreid) 18](#_Toc210139694)

[Contains 18](#_Toc210139695)

[StartsWith / EndsWith 18](#_Toc210139696)

[Equals 18](#_Toc210139697)

[IndexOf/lastIndexof 19](#_Toc210139698)

[Insert/Remove 19](#_Toc210139699)

[Join 19](#_Toc210139700)

[Length 19](#_Toc210139701)

[Replace 19](#_Toc210139702)

[Split 20](#_Toc210139703)

[Substring 20](#_Toc210139704)

[ToLower/ToUpper 20](#_Toc210139705)

[Trim 20](#_Toc210139706)

[Oefeningen 20](#_Toc210139707)

[Controlestructuren 22](#_Toc210139708)

[Conditionele structuren (if/else) 22](#_Toc210139709)

[Operatoren 22](#_Toc210139710)

[If / else if / else 22](#_Toc210139711)

[Herhaaldelijke uitvoer 24](#_Toc210139712)

[Begrensde herhaling 24](#_Toc210139713)

[Onbegrensde herhaling 24](#_Toc210139714)

[Oefeningen 25](#_Toc210139715)

[Object-oriënterend programmeren (OOP) 26](#_Toc210139716)

[Klassen: inleiding 26](#_Toc210139717)

[Bibliografie 29](#_Toc210139718)

# Introductie

Digitale systemen en algoritmes zijn in onze maatschappij nauwelijks nog weg te denken. Van het eten dat op jouw bord komt, tot de games die je speelt, in elk aspect van ons dagelijks leven is informatica aanwezig. Soms op de achtergrond, zonder dat je erbij stil staat.

In deze cursus zal je een uitgebreide kijk krijgen tot programmeren, maken van applicaties en oplossen van algoritmes. Bij het programmeren kiezen we voor C#, aangezien dit één van de meest gebruikte programmeertalen is in applicatie-ontwikkeling. Algoritmes zijn (vaste) stappenplannen die een computer kan uitvoeren om een taak uit te voeren. Dit kan héél simpel het berekenen van een getal zijn, tot het sorteren van data of het vinden van routes in een doolhof.

Voor het programmeren, gaan we gebruik maken van Visual Studio (hierna: VS). Dit is een populaire IDE (ontwikkelomgeving), waarin je eenvoudig extra stukjes software kan toevoegen. Ook gaan we werken met GitHub. Dit is een platform waar je code kan opladen en delen met anderen. Dit is geïntegreerd met VS, zodat je héél snel en zonder al te veel tussenstappen je code kan krijgen. O.a. de oplossingen op de oefeningen in deze cursus zullen via GitHub worden aangeleverd. Meer info hierover verder in de cursus.

# Essentials: datatypes en variabelen

## Wat is C#

C# is een programmeertaal, net als bijvoorbeeld C, C++, Python, Java en JavaScript. Met een programmeertaal kun je **instructies** geven aan een computer. Deze instructies worden uiteindelijk omgezet in machinetaal, zodat de computer ze kan uitvoeren.

Één groot voordeel van C# is dat het een statische programmeertaal is. Dit wil o.a. zeggen dat als een variabele een int (geheel getal) is, dit ook altijd zo blijft. Dit zorgt er dan ook weer voor dat het onterecht gebruik van deze variabele, bijvoorbeeld bij een functie dit een string (tekst) vereist, dus al direct bij het compileren (in VS tijdens het typen) zal worden gedetecteerd, in plaats van bij het uitvoeren (runtime).

Een ander voordeel is de uitgebreide codebase binnen C# en Microsoft. Er zijn platformen voor web-applicaties (ASP.NET Core), geïntegreerde ORM's (Entity Framework Core), mobile applicaties (MAUI), games (Unity), …

In de beginjaren van de digitale revolutie zijn er veel programmeertalen ontstaan (en verdwenen), maar sinds enkele jaren lijkt de strijd gestreden te zijn en is C# sowieso een blijver.

**Belangrijk: elke codelijn (niet hetzelfde als lijn in de editor) moet worden afgesloten met een ; !**

## Variabelen

Elke programmeertaal, en dus ook C#, werkt met variabelen. Dit zijn stukjes computergeheugen die worden vrijgehouden om data naartoe te kunnen schrijven. Bij het programmeren onthoud je dan de locatie (adres) van het dit deeltje computergeheugen, zodat je dit kan uitlezen of iets naartoe kan schrijven.

Een voorbeeld van een geheugenadres is bijvoorbeeld *0x7ffe5367e044*. Good luck! Dit is natuurlijk niet te onthouden, en dus ook niet werkbaar. Daarom kan je met variabelen werken. Dit zijn verwijzingen naar een geheugenlocatie, maar je kan die een eigen naam geven. Als we het bovenstaande voorbeeld nemen, is *leeftijd* hier een variabele. Dit verwijst naar een geheugenlocatie waarin het getal 15 zit opgeslagen.

Bij de naamgeving van een variabele, zijn er wel enkele voorwaarden / informele regels:

* Begint met een letter of underscore ( \_ )
* Bevat enkel letters, cijfers en underscores
* Zijn hoofdlettergevoelig

In het onderstaande voorbeeld zie je een paar voorbeelden van correcte en foute variabelen.

1. ~~string 1enaam = “Marcel”;~~ begint niet met letter of underscore
2. string voor.naam = “Marcel”; bevat een speciaal teken
3. int leeftijd = 15;
4. int Leeftijd = 18;
5. string voornaam = "Marcel";

Opgelet: leeftijd en Leeftijd zijn aparte variabelen

Tenslotte heb je ook nog verschillende manieren om namen van variabelen die uit meerdere woorden bestaan, leesbaar te kunnen opstellen. Bij *camelCase* en *PascalCase* schrijf je alle woorden aan elkaar, maar laat je ieder woord met een hoofdletter beginnen (*camelCase* begint met kleine letter). Bij *snake\_case* schrijf je de woorden van elkaar, maar gebruik je een underscore om ze aan elkaar te koppelen. Je bent vrij om eender welke stijl te gebruiken, maar gelieve deze niet te combineren.

## var

We gaan straks zien welke datatypes er bestaan in C#, maar aangezien C# een statische taal is moet VS weten van welk datatype een variabele is. Daarom zie je in het bovenstaande voorbeeld dat hier int (geheel getal) en string (tekst) voor staat. Dit kan echter achterwege worden gelaten, als uit de definitie van een variabele het datatype blijkt.

Afbeelding met tekst, schermopname, Lettertype, Webpagina

Door AI gegenereerde inhoud is mogelijk onjuist.In onderstaand voorbeeld gaan we 3 keer *var* gebruiken in plaats van een datatype, waarbij we dus eigenlijk het datatype laten bepalen. De eerste keer zeggen we ook gelijk dat de variabele (leeftijd) gelijk is aan 15. Dit is een *int*, dus de variabele leeftijd is dus van het type *int*. De tweede keer zeggen we dat de variabele *naam* gelijk is aan de uitvoer van de functie *Console.ReadLine* (zie hieronder). Deze functie geeft een *string* (tekst) terug, dus *naam* is van het type *string*. De derde keer zeggen we niets meer dan enkel de naam van de variabele (*hobby*). Omdat dit onvoldoende is om het datatype te bepalen, krijgen we dan ook een foutmelding (CS0818).

## Uitvoer (in console)

Om variabelen op het scherm af te drukken, gebruik je de functie *Write* en *WriteLine* van de klasse *Console*. Net zoals bij functies in MS Excel gebruik je haakjes bij functies om *parameters* mee te geven. Parameters zijn gegevens die een functie nodig heeft om te kunnen werken. De print functie schrijft iets op het scherm. Die heeft dus één parameter, nl. dat iets dat op het scherm moet komen.

Het verschil tussen *Write* en *WriteLine*, is dat er bij *WriteLine* een regeleinde wordt toegevoegd, en de volgende uitvoer dus op een nieuwe lijn zal verschijnen.

1. leeftijd = 15;
2. Console.WriteLine(leeftijd);
3. Console.Write(3);

Hint: je kan snel *Console.WriteLine() bekomen door   
cw + Tab te typen.*

## Invoer (in console)

Hierboven hebben we gezien hoe je variabelen kan aanmaken en kan afdrukken op het scherm. We kunnen echter ook tekst inlezen. Dit kan met de functie *ReadLine* van de klasse *Console*. Deze functie heeft, net als *Console.WriteLine*, haakjes.

1. var naam = Console.ReadLine();

## Data-types

Bij het programmeren werken we veel met data. Maar er zijn verschillende soorten data. Bekijk hieronder de voorstelling van een persoon:

Hallo, ik ben jullie leerkracht, *Renaud Leroy*. Ik ben *32* jaar, ben *1,70* meter groot en heb *3* kinderen. Mijn hobby’s zijn: *lezen, voetbal, gamen, fietsen en muziekspelen*. Ik speel *wel* dwarsfluit maar *geen* piano. Mijn lievelingsfilm is *The Lord of the Rings: The Fellowship of the Ring*.

In *cursief* stonden alle stukjes data die interessant zouden kunnen zijn voor een algoritme. Maar niet alle data is dus van hetzelfde type.

|  |  |  |
| --- | --- | --- |
| Data | Soort data | Datatype |
| *Renaud Leroy, LOTR: The Fellowship of the Ring* | Tekst | string |
| *32, 3* | Gehele getallen | int(eger) |
| 1,70 | Kommagetallen | float |
| *Lezen, voetbal, games, fietsen, muziekspelen* | Lijst | array |
| *wel, geen* | Waar/Vals | boolean |

Zo heb je bijvoorbeeld stukjes tekst, zoals een naam of de titel van een film. Je hebt ook getallen, zoals een leeftijd, een aantal kinderen of een grootte, maar dit kunnen dus gehele getallen of kommagetallen zijn. Je kan ook wel of niet iets hebben, en al deze soort data kan je ook nog eens in lijsten steken. Hieronder overlopen we per datatype wat je ermee kan doen.

## String (basis)

Een string is een stuk tekst. Eigenlijk is dit een lijst (array) van tekens (char (van character)), maar omdat tekst zoveel voorkomt bij het programmeren beschouwen we dit als een geheel.

Belangrijk is dat tekst steeds tussen dubbele aanhalingstekens moet worden geplaatst.

1. var eerste\_tekst = "Hallo, ik ben tekst";

Als je dubbele quotes wil gebruiken in jouw tekst, kan je gebruik maken van een \. Deze plaats je voor het aanhalingsteken, zodat dit niet als het einde van de string wordt beschouwd maar als een teken van de tekst.

1. var zin = "Ik zei: \"Ik ga nu mijn huiswerk maken.\"";

Je kan ook teksten met elkaar combineren met een +-teken.

1. var naam = "Obama";
2. var voornaam = "Barack";
3. var president = voornaam + naam; ("BarackObama")

Het +-teken heeft echter het nadeel dat dit bij andere datatypes niet compatibel is. Beter is om een *format* te gebruiken. Hiervoor typ je een $-teken voor de string en gebruik je accolades als placeholder om variabelen in de string in te voegen. Dit wordt ook wel *stringinterpolatie* genoemd.

1. var leeftijd = 64;
2. var voornaam = "Barack";
3. var naam = "Obama";
4. var president = $"{voornaam} {naam} is {leeftijd} jaar.";

## Oefeningen: string

1. Schrijf de tekst "Hello World!" op het scherm.
2. Maak een variabele (*naam)* en schrijf hier jouw naam in weg. Schrijf vervolgens op het scherm: "Hallo *naam*!"
3. Maak 2 variabelen voor jouw naam (*naam*) en de titel van jouw lievelingsgame (*game*). Schrijf vervolgens op het scherm: *naam* speelt graag *titel*!
4. Kies een favoriete quote van een film, steek die in een variabele (*quote*) en schrijf op het scherm: Mijn favoriete quote: "*qoute*"
5. Schrijf de volgende tekst op het scherm, zonder gebruik te maken van variabelen: *Ik zei tegen hem: "Ik bel je 's avonds"*
6. Vraag aan de gebruiker de naam van diens favoriete huisdier ("Wat is jouw favoriete huisdier? ") en schrijf:Ik hou zo van *huisdier.*
7. Vraag aan de gebruikers eerst diens naam ("Wat is jouw naam? "), daarna diens favoriete film ("Wat is jouw favoriete film? ") en toon het volgende op het scherm:Hallo *naam*, ik kijk ook graag naar *film*.

## Int

Net als bij Python is er een verschil tussen een kommagetal en een geheel getal. Een *int* is een geheel getal. Hiermee kan je de standaardbewerkingen +, -, \* (maal) en / doen. Let wel op bij delen! Als je een *int* gedeeld door een andere *int* doet, dan krijg je ook een *int*. Dus 5 / 3 wordt dan niet 1,666 maar wel 1. Dit komt uit Python overeen met //. Je kan dit voorkomen door één van de twee getallen een *float* (kommagetal) te laten zijn. Dit kan door het getal te laten volgen door een *f*.

// wordt in C# gebruikt om commentaar te typen.

Je kan uiteraard ook haakjes gebruiken om voorrang te geven aan bepaalde bewerkingen.

Mogelijks leerde je in Python ook al wat de modulo is. Dit is de restberekening van een deling tussen twee getallen. Zo zie je in het onderstaande voorbeeld dat 5 % 3 de restwaarde is van 5 te delen door 3. 3 past 1 keer in 5, dan hou je nog 2 over, dus 5 % 3 is dus gelijk aan 2.

1. var restwaarde = 5 % 3;

Veelgebruikt bij *int* in C# is ++ en --. Dit verhoogt (++) of verlaagt (--) het getal met 1. Je kan dan ook nog kiezen of je dit voor of achter de variabele zet. Dit is vooral relevant als je gelijktijdig ook extra bewerkingen wil doen, bv. afdrukken op het scherm. Zet je deze ervoor, dan wordt de wijziging gedaan voor de andere bewerking. Zet je ze erna, dan wordt dit pas na deze bewerking gedaan.

Zo geeft de volgende code

1. var getal = 5;
2. Console.WriteLine(--getal);
3. getal = 5;
4. Console.WriteLine(getal--);
5. Console.WriteLine(getal);
6. getal = 5;
7. Console.WriteLine(++getal);
8. getal = 5;
9. Console.WriteLine(getal++);
10. Console.WriteLine(getal);

De volgende uitvoer:

4

5

4

6

5

6

## Float

*Float* is het representeren van een rationaal getal. Let op: zelfs ook bijvoorbeeld 3,0 is een kommagetal. De reden waarom *float* en *int* dan nog apart bestaan, heeft te maken met de geheugenallocatie van deze variabelen. Beiden gebruiken 32 bits. Dit wil zeggen dat wanneer je een variabele aanmaakt van het type *int*, deze 32 bits computergeheugen reserveren. Ook al gebruiken ze niet alle bits om hun getal te onthouden, toch worden deze 32 bits vastgelegd en kunnen die niet door een andere variabele worden gebruikt. Zo bestaat het getal 5 uit 3 bits (101), maar toch zullen de andere 29 bits (allemaal 0), toch niet beschikbaar blijven voor andere variabelen.

De 64-bit varianten van *int* en *float* zijn *long* en *double.*

Zo heeft een *int* met de 32 beschikbare bits een bereik van -2 147 483 648 tot 2 147 483 647, en heeft *float* 'maar' een bereik van ±16 777 216. Uiteraard kan deze wel meer significant worden op het gedeelte na de komma, afhankelijk van hoeveel bits daarvoor resteren.

Hieronder een voorbeeld van de situatie die hierboven werd geschetst, waarbij er een deling wordt uitgevoerd. Als je twee *integers* door elkaar deelt, dan krijg je als resultaat ook een *int*. Als je echter een *float* gebruikt, dan krijg je als resultaat ook een *float*:

1. int teller = 5
2. float deler = 2;
3. float quotient = teller / noemer;

## Converties

Afbeelding met tekst, Lettertype, Webpagina, lijn

Door AI gegenereerde inhoud is mogelijk onjuist.Misschien had je er al op gelet, maar *Console.ReadLine()* geeft een *string* terug (tekst). Dit is echter een probleem voor getallen, want de *string* (lees: tekst) 5 – 3 (lees: getal 3), heeft geen enkele betekenis. Daarom krijg je ook een foutmelding als je dit probeert.

Je moet dus de tekst (bv. "5") omzetten naar het getal (bv. 5). Dit kan op verschillende manieren. Één daarvan is *Convert.To….()*. De puntjes staan er omdat er verschillende conversies bestaan van één type naar een ander. Hier kiezen we dan voor *Convert.ToInt32()* voor *int* en *Convert.ToSingle()* voor *float.* We geven hetgeen we willen converteren mee als parameter. Wat er gebeurt wanneer we een ongeldige waarde ingeven, bv. "abc", zien we later.

1. var teller = Convert.ToInt32(Console.ReadLine());
2. var noemer = Convert.ToSingle(Console.ReadLine());
3. var quotient = teller / noemer;

## Oefeningen

1. Maak een programma dat 2 getallen aan de gebruiker vraagt en van deze getallen het volgende toont: som, verschil, product, quotiënt.
2. Maak een programma dat een invoer 'x' bevraagt en vervolgens de functiewaarde van de volgende functies berekent:
3. Maak een programma dat de lengte ("Lengte (m): ") en het gewicht ("Gewicht (kg): ") van iemand bevraagd. Bereken vervolgens het BMI van deze persoon ("BMI = "). De berekening hiervoor is .
4. Maak een programma dat aan de gebruiker een aantal pizza's vraag die hij wenst te bestellen ("Aantal pizza's: "). Een pizza kost € 14,50 euro, en je hebt een speciale deal: 4 + 1 gratis! Laat de gebruiker het aantal pizza's ingeven en bereken de totale prijs.

# Collecties

## Arrays

Een variabele kan ook een *array* zijn. Een array is een datatype dat een lijst van verschillende waarden uitmaakt, die van hetzelfde datatype zijn. Een array gebruikt vierkante haakjes [ ] en wordt als volgt opgesteld:

1. int[] nummers = new int[3];
2. nummers[0] = 1;
3. nummers[1] = 2;
4. nummers[2] = 3;

Hier maak je een lijst aan van 3 plaatsen groot (*new int [3]*) en vul je de 3 plaatsen op.

Let op: arrays zijn **zero-based**,   
dat wil zeggen dat ze vanaf 0 beginnen te tellen.   
nummers[0] wijst dus naar de eerste plaats in de lijst.

Aangezien een array een **vaste grootte** heeft, is het niet mogelijk om buiten deze grootte waarden in te stellen. In de onderstaande code wordt er getracht de bovenstaande array van lengte 3 uit te breiden met een vierde waarde (*[3]*). Dit geeft dan ook een foutmelding.

1. nummers[3] = 4;

Met de functie *Resize()* kan je een array uitbreiden. In het voorbeeld hieronder wordt de array van 3 plaatsen uitgebreid tot 5 plaatsen. Let op: dit is echter niet zo performant! In werkelijkheid wordt er een nieuwe array van 5 plaatsen gemaakt, en worden de 3 elementen uit de 'oude' array gekopieerd in de nieuwe.

1. Array.Resize(ref nummers, 5);
2. nummers[3] = 4;
3. nummers[4] = 5;

## List

Een array heeft een specifieke grootte, waar je niet boven kan gaan na initialisatie. Wil je toch een collectie de **dynamisch groter en kleiner** wordt, gebruik dan *List<>*. Enkele interessante functie om te beginnen met *List* zijn:

* *Add(element)*: voegt een nieuw element toe aan het einde van de lijst.
* *Insert(positie,element)*: voegt een nieuw element toe op een specifieke positie. Ook hier verwijst de positie 0 naar de voorste (eerste) positie in de lijst.
* *Remove(element):* verwijdert een bepaalde waarde uit de lijst. Let op: *Remove* gaat de lijst van voor naar achter af en verwijdert het eerste element dat hij tegen komt. Komt het meegegeven element meerdere keren voor, dan wordt het slecht een eerste keer verwijderd.
* *RemoveAt(positie):* net zoals bij *Insert()* zal deze functie een waarde op een specifieke plaats verwijderen.

Hieronder vind je een voorbeeld van een lijst met getallen. Er worden 3 getallen (nl. 1, 2, 3) toegevoegd en vervolgens het eerste getal (1) getoond op het scherm en dan het tweede getal (2) verwijderd.

1. List<int> lijst = new List<int>();
2. lijst.Add(2);
3. lijst.Add(3);
4. lijst.Insert(0, 1);
5. Console.WriteLine(lijst[1]);
6. Lijst.Remove(2);
7. lijst.RemoveAt(1);

## ForEach

Om een collectie te overlopen, kan je een for each loop gebruiken. Deze gaat een bepaald stuk code voor elk element in de collectie uitvoeren. Zie onderstaand voorbeeld waarbij voor elk getal in de collectie de deling door 2 wordt getoond op het scherm. De i die hier wordt gebruikt is een placeholder (variabele) voor de huidige waarde die wordt overlopen, en is wat betreft de naam vrij te kiezen (mag niet eerder gebruikt zijn als variabele naam).

1. List<int> lijst = new List<int>();
2. lijst.Add(1);
3. lijst.Add(2);
4. lijst.Add(3);
6. foreach(int i in lijst){
7. Console.WriteLine(i / 2.0);
8. }

## For

Een *for-lus* wordt gebruikt om een bepaalde *int* (hieronder bv. *i*) vanaf een bepaalde startwaarde naar een eindwaarde te laten gaan, met een bepaalde verhoging/verlaging per keer. Concreet komt dit overeen met de *for i in range(…)* lus die je kent uit Python. Een *for-lus* bestaat uit 3 delen:

1. Je stelt een variabele in met een startwaarde
2. Je bepaalt tot wanneer de herhaling wordt uitgevoerd
3. Je verhoogt/verlaagt de variabele met een bepaalde waarde

In het onderstaande voorbeeld, start de variabele *i*  met de waarde 1, blijft de herhaling duren zolang *i* kleiner of gelijk is aan 5 en verhoogt *i* per herhaling met *1* (*i++*).

1. for(int i = 1; i <= 5; i++){
2. }

De start en eindpositie kan je vrij kiezen, al moet je natuurlijk ervoor zorgen dat de eindpositie ooit kan worden bereikt. In het onderstaande scenario verhoogt de variabele steeds meer, terwijl de eindpositie lager ligt (0) dan de startpositie (3).

1. for(int i = 3; i > 0; i++)
2. {
3. Console.WriteLine(i);
4. }

Een *for-lus* wordt vaak gebruikt in combinatie met de lengte van een array/list. Het nadeel van *foreach* is dat je, zolang je de *array/list* overloopt, geen wijzigingen hierop kan uitvoeren. Jouw programma zou anders niet meer weten waar te moeten hervatten bij het toevoegen of verwijderen van een element.

Neem voor de onderstaande voorbeelden de volgende *list*:

1. List<int> nummersList = new List<int>();
2. nummersList.Add(5);
3. nummersList.Add(6);
4. Afbeelding met tekst, schermopname, lijn, Lettertype

   Door AI gegenereerde inhoud is mogelijk onjuist.nummersList.Add(5);

Afbeelding met tekst, schermopname, Lettertype

Door AI gegenereerde inhoud is mogelijk onjuist.Bovendien heb je ook geen mogelijkheid om het element zelf te wijzigen.

Echter, m.b.v. een *for-lus* die de lengte van de *array/list* gebruikt, kan je dit wel doen. Met de vierkante haken (*[ ]*) kan je een element op een specifieke plaats in een *array/list* aanspreken. Als je dan de indexen (posities) in een *array/list* gebruikt, dan kan je gerust in de lijst wijzigingen aanbrengen. Let er dan wel op, als je bijvoorbeeld elementen verwijderd of toevoegt, je hier ook rekening mee houdt in de teller.

1. for(int i = 0; i < nummersList.Count; i++)
2. {
3. nummersList[i] = 0;
4. }

of

1. for(int i = 0; i < nummersList.Count; i++)
2. {
3. nummersList.RemoveAt(i);
4. i--;
5. }

## Oefeningen

1. Maak een lijst en steek hier 5 getallen in. Overloop deze lijst en toon de getallen op het scherm.
2. Maak een lijst en steek hier 3 getallen in. Overloop deze lijst en tel alle getallen bij elkaar op.
3. Maak een lijst en steek hier 3 namen in van huisdieren ("Geef een naam van jouw huisdier op: "). Overloop deze lijst en toon deze strings op het scherm.
4. Maak een lijst van getallen. Laat automatisch hier alle getallen van 278 t.e.m. 300 in steken. Overloop vervolgens deze lijst en tel deze getallen bij elkaar op.
5. Laat de gebruiker een lijst van 3 ingrediënten opgeven. Overloop vervolgens deze lijst en plaats hier overal het nummer van de lijst voor. Bijvoorbeeld: 1. Eieren 2. Melk 3. Bloem
6. Laat de gebruiker een getal ingeven waarvan hij de tafel van 10 wil hebben ("Geef een getal in: "). Steek vervolgens in een lijst alle vermenigvuldigingen van 1 t.e.m. 10 in. Toon deze getallen op het scherm en vervang vervolgens deze getallen door de helft van deze getallen. Bijvoorbeeld: bij het ingeven van het getal 6, vul je eerst de lijst op met 6, 12, 18, … Vervolgens toon je 6, 12, 18, … op het scherm en vervang je de getallen in deze lijst met de getallen 3, 6, 9, …
7. Vraag de gebruiker hoeveel titels van films hij wil ingeven ("Aantal films: "). Vervolgens vraag je het aantal keer dat werd opgegeven naar een filmtitel, van slechtste naar beste ("Titel film (van slecht naar goed): "). Vervolgens toon je de films van goed naar slecht, met ervoor een nummer (1, 2, 3, …).

Bijvoorbeeld: Films: "Super slechte film", "Oké film", "Best film ooit".

Uitvoer:   
1. Beste film ooit   
2. Oké film  
3. Super slechte film

# String (uitgebreid)

We hebben eerder gezien dat een string als een pseudo primitief datatype wordt beschouwd. In feite is het geen primitief datatype, maar een array van *char*-elementen. We gaan in dit hoofdstuk een aantal functies bekijken van String. Bij sommige van deze functies zal je het gegeven van een array herkennen in de werking ervan.

Let op: deze functies geven een **nieuwe *string*** terug! De oude *string* blijft dus onveranderd.

## Contains

Deze functie bekijkt om een gegeven string in een andere voorkomt. In dit geval wordt er gekeken of "C" voorkomt in de tekst "Abcdef". Let op, dit is hoofdlettergevoelig dus in dit geval zal dit niet waar zijn. Wil je dit niet hoofdlettergevoelig maken, kan je als tweede parameter *StringComparison.CurrentCultureIgnoreCase* toevoegen.

1. string tekst = "Abcdef";
2. bool bevat\_C = tekst.Contains("C"); // false
3. bool bevat\_C\_of\_c = tekst.Contains("C", StringComparison.CurrentCultureIgnoreCase); // true

## StartsWith / EndsWith

Deze twee functies controleren of een gegeven string begint- of eindigt met een specifieke tekenreeks. Ook hier kan je *StringComparison.CurrentCultureIgnoreCase* gebruiken om dit niet hoofdlettergevoelig te maken.

1. string tekst = "Abcdef";
2. tekst.StartsWith("A");
3. tekst.StartsWith("a", StringComparison.CurrentCultureIgnoreCase);
4. tekst.EndsWith("f");
5. tekst.EndsWith("F", StringComparison.CurrentCultureIgnoreCase);

## Equals

Met deze functies kan je nagaan of twee strings aan elkaar gelijk zijn. Opnieuw kan dit wel of niet hoofdlettergevoelig.

1. string tekst = "Abcdef";
2. tekst.Equals("Abcdef");
3. tekst.Equals("abcdef", StringComparison.CurrentCultureIgnoreCase);

## IndexOf/lastIndexof

Hiermee kan je de eerste/laatste positie van een teken(reeks) opzoeken in een tekst. Het eerste teken staat plaats 0. Hier zie je dus een schijn van dat een *string* eigenlijk een *char[]* is. Ook dit kan weer wel of niet hoofdlettergevoelig.

1. string tekst = "Abcdef";
2. int positie\_b = tekst.IndexOf("b");
3. int positie\_B = tekst.IndexOf("B", StringComparison.CurrentCultureIgnoreCase);
4. int laatste\_positie\_d = tekst.LastIndexOf("d");
5. int laatste\_positie\_D = tekst.LastIndexOf("D", StringComparison.CurrentCultureIgnoreCase);

## Insert/Remove

Dit zijn eigenlijk opnieuw de functies die we bij *Array* en *List* hebben gezien, maar hier dan toegepast op *string*. Dit kan ook meerdere tekens uitmaken.

1. string tekst = "Abcdef";
2. tekst.Insert(0, "123");
3. tekst.Remove(1);
4. tekst.Remove(1, 2); // verwijder elementen vanaf positie 1, 2 posities ver

## Join

Met deze functie kan je een *array/list* van *strings* aan elkaar koppelen tot één grote *string*. Hiervoor geef je dan ook mee welke "tekst" tussen de verschillende *strings* moet worden gehangen.

1. string[] teksten = new string[] { "abc", "def", "ghi" };
2. string aan\_elkaar = string.Join(" - ", teksten);

## Length

Dit geeft de lengte van de *string* weer, nog zo'n eigenschap van een *array* die naar voren komt. Let op: dit is een eigenschap, geen functie, dus geen haakjes.

1. string tekst = "Abcdef";
2. int lengte = tekst.Length;

## Replace

Vervangt alle gezochte *strings* in een *string* door een andere *string.* Ook hier terug kan je bij het zoeken wel of niet hoofdlettergevoelig werken.

1. string tekst = "Abcdef";
2. string zonder\_b = tekst.Replace("b", "x");
3. string zonder\_B = tekst.Replace("B", "x", StringComparison.CurrentCultureIgnoreCase);

## Split

Met split kan je een *string* opdelen in aparte delen, op basis van een bepaald(e) teken(reeks). Dit levert dan een *string[]* (*array*) op.

1. string tekst = "Dit is een zin";
2. string[] woorden = tekst.Split(" ");

## Substring

Hiermee kan je een deel uit een *string* nemen. Je moet sowieso het eerste de eerste positie opgeven vanaf waar je wil nemen. De tweede parameter, de lengte bij het nemen van de *substring*, is optioneel. Vul je deze niet in, dan wordt tot het einde van de *string* genomen.

1. string tekst = "Abcdef";
2. string cd = tekst.Substring(2; 2);
3. string def = tekst.Substring(3);

## ToLower/ToUpper

Hiermee kan je tekst in hoofd- of kleine letters plaatsen.

1. string Tekst = "Abcdef";
2. string tekst = Tekst.ToLower();
3. string TEKST = Tekst.ToUpper();

## Trim

Hiermee kan je de witruimte, zoals spaties, lijneindes, tabs, …, voor en na een *string* verwijderen.

1. string tekst = " Abcdef ";
2. string zonder\_witruimte = tekst.Trim();

## Oefeningen

1. Geef jouw voornaam ("Voornaam: ") en achternaam ("Naam: ") in, en toon jouw initialen op het scherm. Bijvoorbeeld bij Renaud Leroy wordt dit R. L.
2. Laat de gebruiker een zin intypen, waarbij je de eerste zin in hoofdletters plaats en de rest in kleine letters.
3. De gebruikers geeft een videogame in, met tussen haakjes de console, gevolgd door een streepje ( - ), waarna de prijs komt (al dan niet met euroteken). Haal hieruit de titel van het spel, de console en de prijs (float).
4. Laat de gebruiker een woord ingeven en vervang alle middelste letters (dus behalve de eerste de laatste) door \*-tjes.
5. Verwijder uit een tekst alle dubbele tekens. De tekst "Ik eet banaan in een motorboot" wordt dan ook "Ik et banan in en motobot".
6. Laat de gebruiker een woord ingeven ("Geef een woord in:"). Vervolgens maak je een string waarin welke letter van elkaar staat met een spatie, en in hoofdletters staat. Bijvoorbeeld: voor "banaan" wordt dit "B A N A A N".
7. Maak een programma dat een lijst woorden vraagt aan de gebruiker. Per woord wordt er dan getoond of dit wel of geen palindroom is. Een palindroom is een woord dat hetzelfde is als je het omdraait (bv. lepel). Let erop dat er geen rekening wordt gehouden met hoofd- en kleine letters. Bv. Woorden: appel lepel negen acht wordt

appel: false

lepel: true

negen: true

acht: false

1. Vraag een zin en maak een nieuwe string waarin de volgorde van de woorden omgekeerd is.
2. Lees een zin in en toon het langste woord in hoofdletters.
3. Vraag een zin en verdubbel alle spaties. Let op: er kunnen ook 3 x een spatie na elkaar in staan, of 4, 5, 6, ... keer

# Controlestructuren

Traditioneel loopt een code van boven naar onder. Er zijn echter manieren om code te herhalen (zie hoofdstuk *List/array*) of om code over te slaan.

## Conditionele structuren (if/else)

### Operatoren

Operatoren zijn gereserveerde tekens die we kunnen gebruiken om booleans te verkrijgen uit andere data. Zo kunnen we deze operatoren gebruiken om een voorwaarde van bijvoorbeeld een variabele te testen, en afhankelijk van het resultaat, de boolean een bepaalde waarde te laten aannemen.

De operatoren zijn:

|  |  |  |
| --- | --- | --- |
| Operator | Betekenis | Voorbeeld |
| == | Gelijk aan | 1. int getal1 = 1; 2. bool gelijk\_aan\_3 = getal1 == 3; |
| != | Niet gelijk aan | 1. int getal1 = 1; 2. bool niet\_gelijk\_aan\_3 = getal1 != 3; |
| > | Groter dan | 1. int getal1 = 1; 2. bool groter\_dan\_3 = getal1 > 3; |
| < | Kleiner dan | 1. int getal1 = 1; 2. bool kleiner\_dan\_3 = getal1 < 3; |
| >= | Groter dan of gelijk aan | 1. int getal1 = 1; 2. bool groterofgelijk\_dan\_3 =  getal1 >= 3; |
| <= | Kleiner dan of gelijk aan | 1. int getal1 = 1; 2. bool kleinerofgelijk\_dan\_3 =  getal1 <= 3; |

### If / else if / else

Met de **if**-structuur kan je code enkel uitvoeren als die aan een bepaalde voorwaarde voldoet. Hiervoor schrijf je if, gevolgd door de voorwaarde tussen haakjes en de code die, indien waar, moet worden uitgevoerd, tussen accolades (dit hoeft niet als het maar over één lijn gaat). In onderstaand voorbeeld wordt het getal enkel op het scherm getoond als deze groter is dan 0.

1. var getal = 2;
2. if(getal > 0)
3. Console.WriteLine($"Getal {getal} is positief");
4. if(getal < 0)
5. {
6. Console.WriteLine($"Getal {getal} is negatief");
7. Console.WriteLine("Of gelijk aan 0");
8. }

Let op dat de voorwaarde die je bij een if schrijft, eigenlijk een **boolean** is. Je kan dus ook het volgende doen:

1. var getal1 = 3;
2. var getal2 = 3;
3. bool gelijk\_aan\_elkaar = getal1 == getal2;
4. if(gelijk\_aan\_elkaar)
5. Console.WriteLine("Gelijk aan elkaar");
6. if(getal1 == getal2)
7. Console.WriteLine("Nog steeds gelijk aan elkaar");
8. bool doorgaan = true;
9. if(doorgaan)
10. Console.WriteLine("We gaan door");

De **if-else**-structuur is een uitbreiding van de if-structuur die ook code uitvoert als er aan een bepaalde voorwaarde NIET wordt voldaan.

1. var getal = 2;
2. if(getal > 0)
3. Console.Write("Positief");
4. else
5. Console.Write("Negatief");

Met de **if - else if - else** structuur kan je meerdere voorwaarden nagaan alvorens uiteindelijk ,als alle voorwaarden niet waar waren, in de else terecht te komen.

1. var getal = 2;
2. if(getal > 0)
3. Console.Write("Positief");
4. else if(getal < 0)
5. Console.Write("Negatief");
6. else
7. Console.Write("0");

## Herhaaldelijke uitvoer

Je kan code meerdere keren laten uitvoeren. We spreken dan over begrensde of onbegrensde herhaling. **Begrensde** herhaling wil zeggen dat je een **precies** **aantal** **keer** een herhaling zal uitvoeren. Bijvoorbeeld steeds 3 keer. De eindigheid van de herhaling is dus begrensd tot 3. **Onbegrensde** herhaling wil zeggen dat er **geen** **vaste** **grens** is. Je zal dan herhaling zolang er aan een voorwaarde is voldaan. In theorie kan het zijn dat de voorwaarde steeds is voldaan (zie voorbeeld hieronder), dus let hiervoor op.

### Begrensde herhaling

Voor een begrensde herhaling, gebruiken we een for-lus. Dit werkt met een teller, die steeds met één vermeerderd, tot een bepaalde grens is bereikt. In het onderstaande voorbeeld zal de *variabele teller* de waarden van 1 t.e.m. 5 aannemen.

1. for(int teller = 1; teller <= 5; teller++)
2. {
3. Console.WriteLine(teller);
4. }

Je moet niets met de *variabele teller* doen. Zo kan je bijvoorbeeld het onderstaande doen voor 3 keer "Hallo" op het scherm te schrijven:

1. for(int teller = 1; teller <= 3; teller++)
2. {
3. Console.WriteLine("Hallo");
4. }

Je kan ook de getallen van 10 t.e.m. 1 tonen bijvoorbeeld, door de teller te laten beginnen op 10, zolang de teller groter is dan 0 en telkens met één te verkleinen:

1. for(int teller = 10; teller > 0; teller--)
2. {
3. Console.WriteLine(teller);
4. }

### Onbegrensde herhaling

Bij onbegrensde herhaling werk je met een voorwaarde. Zoals de voorwaarde is voldaan, blijf je doorgaan. Je moet er natuurlijk wel voor zorgen dat de voorwaarde die je stelt, ooit niet waar wordt. Neem de volgende code, waarbij de voorwaarde tussen de haakjes bij *while* bepaald of de code wordt herhaald.

1. int getal = 1;
2. while(getal < 5)
3. Console.WriteLine(getal++);

Deze code gaat continue de huidige waarde van het getal op het scherm afdrukken en vervolgens dit getal met één verhogen. Stel dat je de *++* vergeet, en het volgende uitvoert:

1. int getal = 1;
2. while(getal < 5)
3. Console.WriteLine(getal);

Aangezien de waarde van getal nooit wijzigt, en deze dus 1 blijft, zal de while-lus blijven doorlopen.

Controleer goed de voorwaarde als je een while-lus gebruikt!

Een variant op de *while*-lus is *do-while*. Het verschil ligt in het moment waarop de evolutie van de voorwaarde gebeurd. Bij *while* is dit voor je in de lus gaat, bij *do-while* is dit als je de lus verlaat. Op zich lijkt dit irrelevant, maar dit is vooral belangrijk voor de eerste keer dat je een lus doorloopt. Als jouw lus minstens één keer moet worden uitgevoerd, gebruik je *do-while*. Indien het kan dat de lus nooit moet worden uitgevoerd, kies je *while*.

1. string wachtwoord = "MyVerySecurePassword123!";
2. string input;
3. do{
4. Console.Write("Geef het wachtwoord in: ");
5. string input = Console.ReadLine();
6. } while(wachtwoord != input);

## Oefeningen

1. Maak een programma dat een e-mailadres aan een gebruiker vraagt. Als het e-mailadres geen @-teken bevat, toon de foutboodschap: "Geen @-teken in het e-mailadres". Een e-mailadres moet ook eindigen met een punt en dan 2 of 3 tekens erna. Als dit niet het geval is, toon je de boodschap: "E-mailadres heeft geen geldig einde". Als alles goed is, toon je: "E-mailadres gevalideerd".
2. De gebruiker geeft een pizzaparty en wil pizza's bestellen. Er is keuze tussen 2 pizzeria's. Pizzeria Don Corleone heeft een standaardprijs, 12,50 euro per pizza. Hij levert wel gratis. Pizzeria Scaletta heeft een deal: per pizza betaal je 13 euro maar vanaf 20 pizza's krijg je 10 % korting. Vanaf 50 pizza's krijg je 20 % korting. Hij levert voor 0,80 euro per kilometer. Maak een programma dat het aantal pizza's bevraagt, alsook de leverafstand. Vervolgens bereken je de goedkoopste optie en geef je die aan de gebruiker. Bv. Aantal pizza: 25. Kilometer: 8 Beste optie: Pizzeria Don Corleone voor 312,5 euro.
3. Maak een programma dat aan een gebruiker een getal vraagt en zegt of dit getal een priemgetal is. Een priemgetal is een getal dat enkel maar deelbaar is voor 1 en zichzelf.
4. Maak een programma dat aan een gebruiker een getal vraat en alle priemgetallen kleiner of gelijk aan dit getal teruggeeft. Een priemgetal is een getal dat enkel maar deelbaar is voor 1 en zichzelf.
5. Maak een programma

# Object-oriënterend programmeren (OOP)

## Klassen: inleiding

In quasi alle programmeertalen kan je gebruik maken van klassen. Klassen zijn beschrijvingen van objecten op basis van hun gemeenschappelijke soort eigenschappen en gedrag.

Bijvoorbeeld: denk aan de volgende dieren: kat, hond, duif, goudvis, koe, kip, vogelbekdier. Al deze dieren zijn compleet verschillend, maar als je ze beschrijft gebruik je dezelfde soort eigenschappen. Een hond is een zoogdier, een kat is een zoogdier, een duif is een vogel, een goudvis is een vis, een koe is een zoogdier, een kip is een vogel, een vogelbekdier is een zoogdier. Of denk aan de kleur, elk van deze dieren heeft een verschillende kleur maar ze hebben we allemaal een kleur, of dit nu wit, bruin, grijs, oranje of eerder wat is.

Op basis van deze zelfde soort eigenschappen, zouden we kunnen zeggen dat elk een dier, ongeacht wat de specifieke waarde is, de volgende kernmerken heeft:

• Soort

• Kleur

• Klasse (zoogdier, reptiel, …)

• Aantal poten

• Gemiddelde hoogte

• Kan wel of niet vliegen

• Legt wel of geen eieren

We kunnen een dier dan omschrijven op basis van de volgende kenmerken:

|  |
| --- |
| Dier |
| * soort: string * kleur: string * klasse: string * aantal\_poten: int * gemiddelde\_hoogte: double * vliegen: bool * eieren: bool |

Waarom doen we dit nu? Wel, bij het programmeren kunnen we zelf een klasse maken, hier eigenschappen aan geven en dan concrete instanties van die klasse maken. De concrete instanties zijn hier dan de concrete dieren die we maken.

Voor een klasse te maken, doe je het volgende:

1. Maak een map ‘Klassen’ in jouw project

2. Rechtermuisklik op deze map > Add > Class. Geef de klasse een naam en klik op Add.

3. In het klasse-bestand dat werd gecreëerd, vervang internal naar public.

4. Maak je eigenschappen aan binnen de accolades van public naam\_klasse { } als volgt:

public datatype\_eigenschap naam\_eigenschap { get; set; }

Bijvoorbeeld:

1. public class Dier
2. {
3. public string Kleur { get; set; }
4. public string Klasse { get; set; }
5. public int Aantal\_poten { get; set; }
6. public double Gemiddelde\_hoogte { get; set; }
7. public bool Vliegen { get; set; }
8. public bool Eieren { get; set; }
9. }

Vervolgens kan je in jouw Program.cs-bestand een concreet dier als volgt aanmaken en eigenschappen invullen:

1. using Dierentuin.Klassen;
2. Dier kat = new Dier();
3. Dier kat = new Dier();
4. kat.Soort = "Kat";
5. kat.Kleur = "Allelei";
6. kat.Klasse = "Zoogdier";
7. kat.Aantal\_poten = 4;
8. kat.Gemiddelde\_hoogte = 0.35;
9. kat.Vliegen = false;
10. kat.Eieren = false;
11. Dier duif = new Dier();
12. duif.Soort = "Duif";
13. duif.Kleur = "Grijs";
14. duif.Klasse = "Vogel";
15. duif.Aantal\_poten = 2;
16. duif.Gemiddelde\_hoogte = 0.2;
17. duif.Vliegen = true;
18. duif.Eieren = true;
19. Dier vogelbekdier = new Dier();
20. vogelbekdier.Soort = "Vogelbekdier";
21. vogelbekdier.Kleur = "Bruin";
22. vogelbekdier.Klasse = "Zoogdier";
23. vogelbekdier.Aantal\_poten = 4;
24. vogelbekdier.Gemiddelde\_hoogte = 0.15;
25. vogelbekdier.Vliegen = false;
26. vogelbekdier.Eieren = true;

Je kan dit ook korter maken. Een klasse kan je een constructor geven. Een constructor is een manier om een object aan te maken en direct op te vullen met een aantal waarden. We gaan hier later meer in detail op in, maar voorlopig kan je een volledige constructor aanmaken als volgt:

1. Ga naar jouw klasse-bestand

2. Klik binnen de accolades van public naam\_klasse { } met de rechtermuisklik > Quick Actions and Refactorings > Generate constructor… > Ok. Je mag hier alles aangevinkt laten staan.

Dan verschijnt er normaal gezien iets als het volgende:

1. public Dier(string soort, string kleur, string klasse, int aantal\_poten, double gemiddelde\_hoogte, bool vliegen, bool eieren)
2. {
3. Soort = soort;
4. Kleur = kleur;
5. Klasse = klasse;
6. Aantal\_poten = aantal\_poten;
7. Gemiddelde\_hoogte = gemiddelde\_hoogte;
8. Vliegen = vliegen;
9. Eieren = eieren;
10. }

Dit kan je dan als volgt gebruiken in jouw Program.cs-bestand:

1. Dier kip = new Dier("Kip", "Allelei", "Vogel", 2, 0.4, false, true);
2. Dier koe = new Dier("Koe", "Allelei", "Zoogdier", 4, 1.6, false, false);

Je kan dan ook een lijst van deze klasse maken:

1. List<Dier> dieren = new List<Dier>();
2. dieren.Add(kat);
3. dieren.Add(duif);
4. dieren.Add(vogelbekdier);
5. dieren.Add(kip);
6. dieren.Add(koe);

## Functies

Klassen kunnen naast een aantal eigenschappen te hebben, ook *gedrag* vertonen. Dit noemen we functies. Functies doen iets. Dit kan een berekening zijn, of het aanpassen van bepaalde eigenschappen of iets op het scherm reproduceren.

Je hebt 2 soorten functies. Je hebt functies die enkel iets doen en je hebt functies die ook iets teruggeven. Let op: met teruggeven bedoelen we dat de functie een getal of een string opstelt en je dit dan in een aparte variabele kan opslaan. Een getal op het scherm schrijven is niet iets terug geven, aangezien jouw code achteraf niet weet welk getal dit was.

Een functie heeft een bepaalde opbouw:

*public return\_type naam\_functie(parameters) { … (code) }*

Het return type is het type van wat de functie teruggeeft. Als jouw functie bijvoorbeeld een getal berekend, kan dit int of double zijn. Stelt jouw functie een tekst op, kan dit string zijn. Geeft jouw functie niets terug, dan schrijf je *void*.

De naam van de functie mag je zelf kiezen, al gelden ook hier dezelfde regels als de naamgeving bij variabelen. Kies steeds een zinvolle naam voor jouw functie die héél krachtig en duidelijk omschrijft wat de functie doet! Je gaat ook steeds proberen om functies slechts één iets te laten doen.

Parameters zijn gegevens die een functie nodig heeft om te werken. Stel dat je een functie schrijft die een macht van een bepaald getal berekend. Dan heeft jouw functie 2 zaken nodig, nl. het grondgetal en de exponent. Bij de parameters schrijf je steeds eerst het datatype en vervolgens de naam van de parameter. Dit is dan een variabele waarin een waarde wordt ingevuld als de functie wordt opgeroepen en waarmee je binnen jouw functiecode hiermee kan werken.

Jouw code schrijf je uiteindelijk tussen accolades. Als jouw functie iets teruggeeft, doe je dit door return te schrijven, gevolgd door wat je wil teruggeven. Let op: na een return wordt de functie afgesloten, ook al staat er nog code achter.

Hieronder een paar voorbeelden van hoe functies eruit kunnen zien:

1. public class Leerling
2. {
3. public string Voornaam { get; set; }
4. public string Naam { get; set; }
5. public string Klas { get; set; }
6. public int Leeftijd { get; set; }
7. public List<double> Punten\_wiskunde { get; set; }
8. public List<double> Punten\_fysica { get; set; }
9. public List<double> Punten\_chemie { get; set; }
10. public List<double> Punten\_biologie { get; set; }
11. public string volledige\_naam()
12. {
13. return $"{Voornaam} {Naam}";
14. }
15. public void toon\_volledige\_naam()
16. {
17. System.Console.WriteLine(volledige\_naam());
18. }
20. public void voeg\_punt\_toe(string vak, double punt)
21. {
22. vak = vak.ToUpper();
23. if(vak=="WISKUNDE")
24. Punten\_wiskunde.Add(punt);
25. else if(vak=="FYSICA")
26. Punten\_fysica.Add(punt);
27. else if(vak=="CHEMIE")
28. Punten\_chemie.Add(punt);
29. else if(vak=="BIOLOGIE")
30. Punten\_biologie.Add(punt);
31. }
32. public double bereken\_totaal\_gemiddelde()
33. {
34. return Math.Round(Punten\_wiskunde.Average() + Punten\_fysica.Average() +
35. Punten\_chemie.Average() + Punten\_fysica.Average(), 2);
36. }
37. public bool mag\_school\_verlaten()
38. {
39. return Leeftijd >= 16;
40. }
41. public void verander\_klas\_zelfde\_jaar(string nieuweKlas)
42. {
43. if (Klas == "")
44. {
45. Klas = nieuweKlas;
46. return;
47. }
48. string jaarKlas = Klas.Substring(0, 1);
49. string jaarNieuweKlas = nieuweKlas.Substring(0, 1);
50. if (jaarKlas == jaarNieuweKlas)
51. Klas = nieuweKlas;
52. }
53. }

## Oefeningen

1. Maak een klasse *Cilinder*. Een cilinder heeft 2 eigenschappen: straal basisvlak en hoogte. Laat deze eigenschappen invullen met een constructor. Maak vervolgens functies die de oppervlakte (mantel) en inhoud van de cilinder berekenen en teruggeven. Maak ook een functie die het volgende op het scherm schrijft: “Een cilinder met een straal 3 en een hoogte 10 heeft een mantel van 245 en een volume van 282,7”
2. Maak een klasse *Persoon*. Geef deze klasse de volgende eigenschappen: naam, voornaam, geboortejaar, lengte, gewicht, geslacht. Maak een constructor die al deze eigenschappen instelt. Maak een functie die de initialen van de persoon toont op het scherm. Maak ook een functie die de leeftijd van de persoon berekend. Deze functie krijgt een jaartal mee als parameter (je mag ervan uit gaan dat de persoon reeds is verjaard in dat jaar). Maak ook een functie die het BMI van die persoon bepaald.
3. Maak een klasse *Vergelijking*. Deze klasse heeft 3 eigenschappen: een waarde voor tweedegraadsterm, een waarde voor eerstegraadsterm en een constante. Maak een constructor waarmee je een vergelijking van de eerste- of tweedegraad kan opstellen. Schrijf een functie die de vergelijking toont op het scherm, bv. 2x + 3 of x² + 5x + 9 of 3x² + 8. Let op, die zijn de termen die een 0 hebben niet getoond. Schrijf ook een functie die de nulwaarden voor deze functie berekend. Schrijf tenslotte ook een functie die de eerste en tweede functie combineert en samen op het scherm schrijft, bv. “2x + 3 heeft als nulpunt -1,5” of “x² + “x2 - 5x + 6 heeft als nulpunten 2 en 3”. Je mag niet-gehele getallen schrijven als een breuk, afgerond op 2 cijfers na de komma.
4. Maak een klasse ‘IJskraam’. Een ijskraam heeft verschillende bakken ijs: vanille, chocolade en aardbei. Uit een bak kan je ongeveer 20 bolletjes uit halen. Maak 3 eigenschappen die het aantal bollen bijhoudt per smaak die nog overschieten in de bakken. Maak ook een eigenschap die het bedrag van het ijskraam onthoudt. Schrijf een constructor waarmee je een ijskraam kan maken. Deze constructor krijgt een bedrag mee dat in de kassa zit (soorten munten/biljetten) maakt niet uit. Schrijf een functie die een bestelling doet en het wisselgeld berekend. De functie heeft 3 parameters voor de smaken van de bollen (als je maar 2 bollen wil is de 3e parameter leeg), een indicatie of je wel/geen potje wil en het gegeven bedrag. Een bol kost 2,5 euro. 2 bollen kosten 4 euro en 3 bollen kosten 5,5 euro. Een potje kost 0,5 euro. De functie heeft ook een bedrag dat de klant geeft. Als er niet genoeg wisselgeld in de kassa is, schrijf je ‘Niet genoeg wisselgeld’. Als het aangeboden bedrag door de klant onvoldoende is, schrijf je ‘Je hebt … te weinig betaald’. Anders bereken je het wisselgeld en toon je dit ‘Je krijgt … terug’.

# Bibliografie

**Ongeldige bron opgegeven.**