CSSE1

Kristof Raes, 2020-2021

Besturingssystemen

Hardware

Computers

Supercomputers

- Enkele processoren (’60 - ’70).
- Honderden/duizenden processoren.
- Quadriljoenen berekeningen in nanoseconden.
- Honderden vierkante meters ruimte.
- Gebouwd voor specifieke doeleinden.

Voorbeelden:

- Weather and climate operational supercomputing system (foto).
    - National Oceanic and Atmospheric Administration.
    - Weer en klimaat voorspellingen.
- IBM’s Sequoia machine
    - Beveiliging van nucleaire wapens.
    - Moleculaire dynamische berekeningen.

Mainframe computers

- Kleinere versie van supercomputers.
- Grote organisaties en bedrijven.
- Virtuele machines.
- Marktleider IBM.
- Kostprijs: 75.000 tot 10 miljoen euro.

Minicomputer

- Jaren ’60 & ’70.
- Kleinere mainframe.
- Ondersteund tot 200 gebruikers.
- Netwerk
- Servers en microcomputer.

Server & datacenter

- Diensten verlenen.

- Netwerk
- Klein in omvang.
- Datacenter
    - Grote hoeveelheid servers.
    - Optimale + beveiligde omgeving.
    - Racks
- Voorbeelden:
    - Webserver
    - Fileserver
    - Gameserver
    - Cloud, filmpje: wat is de cloud?

Microcomputer

- Personal computers.
- Desktop computers, laptops, video game consoles,…
- Microprocessor

Samenvatting

- Supercomputers
    - Specifiek doeleind.
- Mainframe computers.
    - Grote bedrijven
- Minicomputer
    - Vroeger -> kleinere bedrijven
- Server & datacenter.
    - Diensten verlenen.
- Microcomputer
    - Personal computer.

Computeronderdelen

Inputtoestellen

- Communiceren met de computer.
- Toetsenbord, muis, joystick, controller,…
- Keyboards
    - Elke vorm van input door toetsen.
- Aanwijzende toestellen.
    - Elke manier om een punt aan te wijzen.

Outputtoestellen

- De computer communiceert met ons.
- Weergeven van informatie.
- Hoofdtelefoon, monitor, printers,…

Verwerkingstoestellen

- Informatie te verwerken.

- Moederbord
    - Centraal
    - Laat componenten communiceren.
    - Chipset: onderverdeeld in 2 delen namelijk Northbridge en Southbridge.
    - Northbridge
        - Snelle componenten.
            - Processor socket.
                - CVE/CPU: Centrale verwerkingseenheid of Central Processing Unit, informatiefilmpje Dell, Computer Processors Explained.
                    - De CPU bestaat uit 3 componenten.
                        - Memory of storage unit.
                            - Bewaart instructies, data en resultaten.
                        - Control unit.
                            - Controleert bewerkingen van andere computeronderdelen.
                        - ALU (Arithmetic Logic Unit).
                            - Arithmetic section (rekenkundige operaties).
                            - Logic section (logische bewerkingen).
                - Hersenen
                - Ophalen, decoderen en uitvoeren van instructies.
                - Uitvoeren van mathematische en logische calculaties.
                - Input data -> output data.
                - Snelheid in Gigahertz (GHz).
                - Klok die aantal keer per seconde slaat.
                    - 3.1 GHz - 3.1 biljoen per seconde.
                - Elke slag -> berekening.
                - Beïnvloed door meerdere factoren.
                - 32-bit vs 64-bit CPU
                    - 64-bit de meest moderne.
                    - Verschil in het aantal berekeningen.
                    - Kan meer data uit het geheugen behandelen.
                    - Software programmeurs.
                    - Software in 32 of 64 bit.
                    - Mac OS X heeft geen 32 bit versie meer.
            - Aansluiting geheugenmodules.
                - RAM: random access memory
                    - Tijdelijk geheugen.
                    - Elke geheugenplaats even snel toegankelijk.
                    - Lezen en schrijven kan in willekeurige volgorde gebeuren.
                - ROM: read only memory
                    - Geheugen die men gebruikt wanneer er data moet bewaard worden als het toestel uitstaat.
            - Videokaart
                - GPU: graphics processing unit, deze neemt de videotaken over van de CPU.
                - BIOS: bewaren van de settings van de grafische kaart en voert testen uit op het geheugen van de kaart bij het opstarten.

            - RAM: wanneer de GPU afbeeldingen maakt, dient hij deze samen met andere informatie te kunnen bewaren.
                - Informatie over pixels.
                - Kleuren
                - Locatie ervan op het scherm.
            - Slots:
                - PCI: Peripheral component interconnect
                - AGP: Advanced graphics port
                - PCIe: PCI Express, snellere communicatie tussen grafische kaart en het moederbord. Mogelijkheid tot het aansluiten vn meerdere grafische kaarten.
        - Uitbreidingskaarten
    - Southbridge
        - Trage componenten.
        - BIOS (Basic input/output system, EFI voor Mac)
            - Is een bibliotheek met basisinstructies.
            - Is de brug tussen opstarten van hardware en het opstarten van het besturingssysteem.
            - Weet waar het besturingssysteem staat.
            - Toegankelijk via bepaalde toets of toetsencombinatie.
            - Aanpassingen maken.
            - Wijzigingen opgeslagen in CMOS (Complementary Metal-Oxide-Semiconductor), grootte 256 bytes.
        - Aansluiting opslagtoestellen.
            - HDD (hard disk drive) of harde schijf is een extern geheugen. Je moet het zien als een elektromechanisch computeronderdeel waar er gegevens op kunnen bewaard worden. De harde schijf is de schijf die ronddraait en waar er een lees- en schrijfoperaties worden op uitgevoerd.
            - SSD (solid state drive) is een opslagmedium zoals een harde schijf met dat verschil dat het geen schijf is maar een flashgeheugen. Het voordeel van SSD's zijn de kortere zoek- en toegangstijd voor bestanden in vergelijking met harde schijven.
        - Aansluitingen voor toetsenbord, muis, netwerk,…

Data

Het is nuttig te weten hoe hoeveelheden data uitgedrukt worden in termen van computers, hieronder een overzicht.

Naam	Symbool	Verklaring
bit	b	bit (samentrekking van binary en digit), 0 of 1 als waarde.
Byte	B	Byte is een binaire eenheid van data, 1 Byte = 8 bits.
Kilobyte	kB	10^3 bytes (ofwel 1000 bytes)
Megabyte	MB	10^6 bytes (ofwel 1.000.000 bytes)

Naam	Symbool	Verklaring
Gigabyte	GB	10^9 bytes (ofwel 1.000.000.000 bytes)
Terabyte	TB	10^12 bytes (ofwel 1.000.000.000.000 bytes)
Petabyte	PB	10^15 bytes (ofwel 1.000.000.000.000.000 bytes)
Exabyte	EB	10^18 bytes (ofwel 1.000.000.000.000.000.000 bytes)

Naam Symbool Verklaring

Computer Science

Waarvoor staat Computer Science?

Computer Science is de studie van hoe een computersysteem werkt, meestal vanuit het wiskundige en theoretische perspectief.

Hierbij ga je de processen bekijken/onderzoeken/bestuderen die omgaan met data en die eventueel kunnen worden weergegeven als gegevens in de vorm van software.
Meestal gaat men hierbij gebruikmaken van algoritmen om digitale informatie te manipuleren of aan te passen, op te slaan en te verspreiden/communiceren.

Software Engineering

Waarvoor staat Software Engineering?

Software Engineering is de studie van hoe een softwaresysteem werkt.

Hierbij kijk je van naderbij uit welke delen een programma bestaan en hoe je van input naar output gaat met daarbij de verschillende stappen die je gaat inbouwen (sequentie, iteratie, selectie, OOP, variabele, datacollectie,...).

Als software engineer ga je een vraag of een probleem omzetten in een programma/applicatie.

.NET

Wat is .NET?

.NET is een gratis en cross-platform opensource ontwikkelaarsplatform (zie ook opensourcesoftware) voor het maken van verschillende types van applicaties.

Een ontwikkelaarsplatform op zich bestaat uit de talen waarin je programmeert en de bibliotheken die je hierbij kan gebruiken.

.NET = languages + libraries

Met .NET kan je meerdere talen, editors en bibliotheken gebruiken om zo applicaties te ontwikkelen voor web, mobile, IoT, desktop or gaming. Maakt niet uit of dit in C#, F# of Visual Basic is geschreven, de geschreven code zal zoals oorspronkelijk draaien op elk compatibel besturingssysteem.

De verschillende .NET implementaties nemen het zware werk van je over en maken het werk lichter.

.NET documentatie

De talen.

- C# (C Sharp), programmeertaal, ontwikkeling in 2000 door Microsoft als onderdeel van .NET initiatief, introductie samen het .NET-framework, objectgeoriënteerd, qua syntaxis/semantiek sterk gelijkend op Java; documentatie.
- Visual Basic, programmeertaal, verschenen in 1992, strong typing, de variant onder .NET is Visual Basic .NET en die is verschenen in 2002 ; documentatie.
- F# (F Sharp), scripttaal, eerste versie uit 2005, strong typing, .NET Framework; documentatie; meer info: fsharp.org.

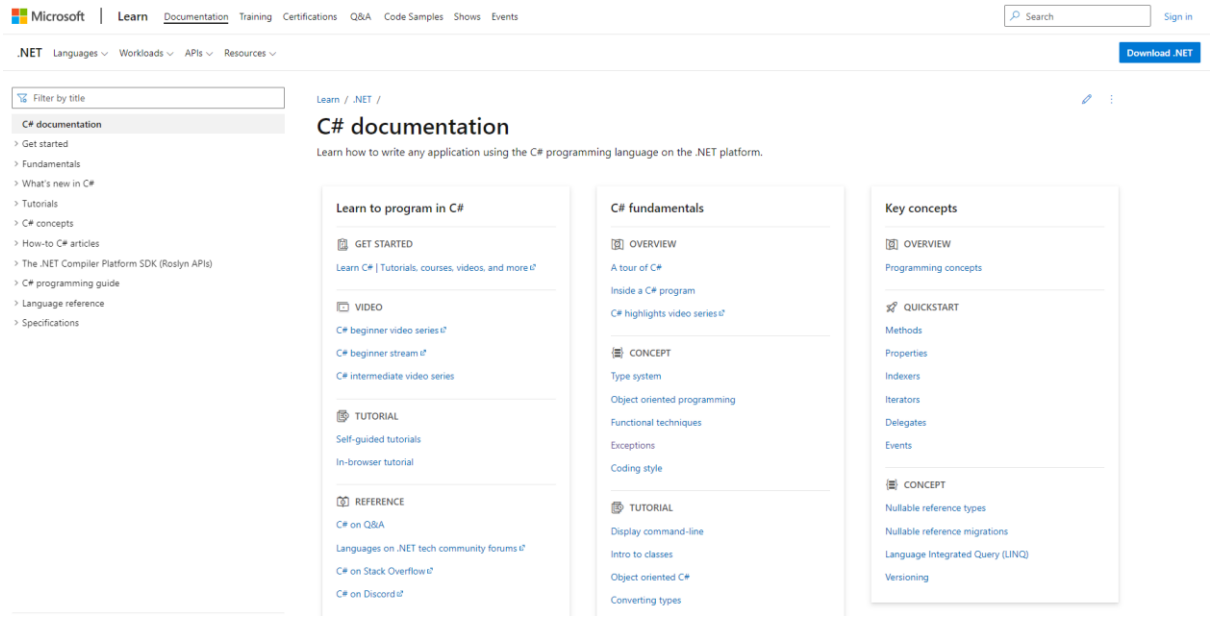
De programmeerplatformen.

- .NET Core: applicaties voor Windows, Lunix en macOS (kort en bondig, draait op alles); documentatie.
    - .NET API Browser
- .NET Framework: websites, services en applicaties op Windows; documentatie.
    - .NET API browser
- Xamarin/Mono: .NET for mobile; documentatie.
- .NET standard: de basisbibliotheek waarover de drie bovenstaande programmeerplatformen over beschikken.

Waarom .NET?

- Meer dan 60000 ontwikkelaars en 3700 bedrijven dragen bij tot .NET.

C# documentatie



Meer informatie omtrent C#, kan je steeds vinden op: https://learn.microsoft.com/en-us/dotnet/csharp/

Installatie

Microsoft Visual Studio Code

Microsoft Visual Studio Code (korte notatie: VS Code) is klein en licht, maar bevat een sterke source code editor die draait op je desktop en is beschikbaar voor Windows, Mac en Linux. Het bevat ondersteuning voor JavaScript, TypeScript en Node.js en heeft een rijk ecosystem van extensies voor andere talen, zoals C++, C#, Python, PHP, Go...

Voorlopig worden reeds meer dan 30 programmeertalen ondersteund, zoals: JavaScript, C#, C++, PHP, Java, HTML, CSS, SQL,...
Het editeren is gefocused op het schrijven van code (multiple cursors, autosave, …). Deze editor bevat verschillende features:

- snel zoeken via RegEx;
- definities van klassen e.d. bekijken;
- code outline;
- intellisense;
- debugging;
- git version control;
- vorm aanpasbaar door de gebruiker;
- uitbreidbaar via extensions…

Installatie

macOS

1. Download de zip met Visual Studio Code voor Mac;
2. Dubbel-klik op het gedownloade archiveerbestand om te inhoud uit te pakken;
3. Sleep de Visual Studio Code.app naar de Applicatiefolder (Eng.: Applications folder), zodat deze beschikbaar is in het **Launchpad**.
4. Voeg VS Code toe aan de Dock door rechts te klinnen op het icoon en vervolgens te kiezen voor **Options, Keep in Dock**.

Windows

Download de 64-bit versie om Microsoft Visual Studio Code ten volle te gebruiken!

1. Download het Visual Studio Code installatieprogramma voor Windows.
2. Na de download, voer het installatieprogramma uit (VSCodeUserSetup-{bits}-{version}.exe).
3. De standaardlocatie van het VS Code programma bevindt zich onder C:\Users\{username}\AppData\Local\Programs\Microsoft VS Code.

Configuratie

CLI

We kunnen VS Code ook uitvoeren via de terminal door het commando code te typen, nadat we dit commando toegevoegd hebben aan de $PATH waarde uit het systeem. Om dit te verwezenlijken lanceren we VS Code, vervolgens openen we het **Command Palette** via ⇧⌘P (macOS) of cntrl+p (Windows) en typen hier > Shell command: install 'code' command in PATH. Herstart de terminal zodat de nieuwe waarde voor $PATH van kracht is. Standaard is het pad al aangepast na het uitvoeren van de Windows installatie.

Extensies

Een extensie (Eng.: extension) is een uitbreiding van een toepassing die extra functionaliteit toevoegt.

Zie ook

Extension Marketplace

- https://code.visualstudio.com/docs/editor/extension-gallery
- https://marketplace.visualstudio.com/vscode

Installeer een extensie door de Extensions tool te openen met: ⇧ Shift+Cmd+X (macOS) en ⇧ Shift+Ctrl+X (Windows) of via het icoon .

De volgende extensies zijn aanbevolen voor het opleidingsonderdeel Computer Science & Software Engineering I:

- Programming
  - **C#**

Verdere informatie kan je vinden op de Visual Studio Code website omtrent C#.

Voorkeuren

Voorkeuren (Eng.: settings) voor de VS Code editor kunnen we instellen via het menuitem File > Preferences > Settings of via de shotcut Cmd+, (macOS) en Cntrl+, (Windows).

Lettertype

Het standaard lettertype kan in VS Code gewijzigd worden. Om de leesbaarheid van code te verhogen kunnen we het lettertype FiraCode installeren en instellen als het standaard lettertype binnen VS Code.

1. Download via FiraCode GitHub Repository
2. Pak het zip-bestand uit en navigeer vervolgens, in de uitgepakte folder, naar de ttf folder.
3. Dubbelklik op elk bestand in deze folder en klink **Install font**.
4. Open het voorkeursvenster en navigeer naar het font menuitem.
5. Ken vervolgens de waarde 'Fira Code', Menlo, Monaco, 'Courier New', monospace aan het invulveld onder **Font Family**
6. Bewaar de instellingen.

Meer info

- FiraCode GitHub Repository
- Tahoe Ninjas: Setting Fira Code as your default Visual Studio Code font

Kleurthema instellen

Kies via het menu het volgende item Code > Preferences > Color Theme of Cmd+K Cmd+T (macOS) of Cntrl+K Cntrl+T (Windows) en kies vervolgens één van de beschikbare kleurenthema's (Eng. color themes).

Bestandsiconenthema instellen

Bestandsiconenthema (Eng.: file icon theme) kan ingesteld worden door het menuitem Code > Preferences > File Icon Theme te selecteren en vervolgens één van de beschikbare thema's te kiezen.

Git

Git staat voor "Global Information Tracker". Het is een revisie beheersysteem (Eng.: revision control) en een broncode (Eng.: source code) management systeem (SCM) vergelijkbaar met het alomgekende SVN-systeem (2001 e.v.).

Het Git-systeem, initieel ontwikkeld door Linus Torvalds voor de Linux Kernel Development in 2005, voldoet aan een aantal vereisten (requirements):

- gratis;
- eenvoudig, snel en efficiënt;
- betrouwbaar;
- schaalbaar (scalable), voorbeeld: met honderden teamleden kunnen samenwerken aan hetzelfde project;
- geschiedenis, weten wie wat gedaan heeft en wanneer...;
- transacties, zoals meerdere acties bundelen;

- ondersteuning voor branches, zoals afsplitsing van het hoofdproject die later terug kan samengevoegd worden met het hoofdproject;
- …

Iedere Git-werkmap bevat een volledige repository met een overzicht van de geschiedenis en bevat ook tracking capaciteiten.
Git is niet afhankelijk van een centrale opslagplaats!

Nieuwe versies van een app worden eerst lokaal bewaard in een lokale copy van de centrale opslagplaats (server).
Deze lokale opslagplaats kan later gesynchroniseerd worden met de centrale opslagplaats.
Conflicten in versies worden aangeduid door de Git-software, zodat een teamlid deze kan oplossen!

Installatie

macOS

Open **Terminal** en voer git --version uit om te testen of **git** geïnstalleerd is.

```
git --version
```

Indien git nog niet geïnstalleerd is, zal een popupvenster verschijnen. Klik op de knop "**Installeer**" om de **Command Line Developer Tools** te installeren.

Indien het popupvenster niet verschijnt, kan je het manueel starten met:

```
xcode-select --install
```

Git kan op macOS ook geïnstalleerd worden via Homebrew:

```
brew install git
```

Het is aan te raden om bij de installatie van git via Homebrew eerst de commando's brew update en brew upgrade uit te voeren.

Windows

1. Download git via: https://github.com/git-for-windows/git/releases/.
    1. 32bit versie: https://github.com/git-for-windows/git/releases/download/v2.33.0.windows.2/Git-2.33.0.2-32-bit.exe
    2. 64bit versie: https://github.com/git-for-windows/git/releases/download/v2.33.0.windows.2/Git-2.33.0.2-64-bit.exe
2. Open het gedownload bestand.
3. Selecteer tijdens de "Git Setup Installatie Wizard" de optie om git toe te voegen aan de omgevingsvariabelen, zodat we het git commando overal kunnen aanspreken!

Configuratie

Identiteit

Het eerste dat we moeten doen nadat we Git hebben geïnstalleerd is om onze **gebruikersnaam** en **e-mailadres** in te stellen. Dit is belangrijk omdat elk **Git commit commando** deze informatie gebruikt om het commit object te ondertekenen. Deze informatie zit vervat als meta-informatie in een commit beschrijving.

```
git config --global user.name "John Doe"
git config --global user.email johndoe@example.com
```

Bijvoorbeeld:

```
git config --global user.name "Kristof Raes"
git config --global user.email "kristof.raes@arteveldehs.be"
```

Dit hoeft enkel maar eenmaal uitgevoerd te worden omdat we de `--global` optie vermelden. Git zal steeds deze instellingen gebruiken voor alles wat je gerelateerd doet op het systeem. Indien we deze instellingen willen overschrijven voor een project, dan voeren we deze commando's uit zonder de `--global` optie.

Forceren van https i.p.v. git

Binnen het netwerk van de Arteveldehogeschool wordt het **git-protocol** geblokkeerd door de firewall. We kunnen dit protocol vervangen door `https` via de volgende configuratie:

```
git config --global url."https://".insteadOf git://
```

Op deze manier gebruiken we altijd **https-protocol** i.p.v. git-protocol. De overdracht van data zal dus voortaan geschieden via https.

Editor

Via configuratie kunnen we de standaard editor instellen, die Git zal gebruiken wanneer we een boodschap (Eng.: message) moeten typen. Standaard wordt de "default" editor van het besturingssysteem gebruikt: notepad (Windows), vi of vim (macOS), … .

VS Code kunnen we instellen als default editor:

```
git config --global core.editor "code --wait"
```

Merge en Diff tool

Specifiëren van de standaard “merge tool” om “merge conflicts” op te lossen.

VS Code kunnen we instellen als default “merge tool”:

```
git config --global merge.tool vscodemerge
git config --global mergetool.vscodemerge.cmd 'code --wait --diff $LOCAL $REMOTE'
git config --global mergetool.vscodemerge.trustExitCode false
git config --global diff.external code
```

GitHub

GitHub is een webgebaseerde (web-based) service om software development projecten te herbergen (hosten) gebruik makend van het Git revisie (revision control) beheersysteem. Het wordt veel gebruikt voor open-source software development projecten.

GitHub bevat de volgende plannen:

- Private repositories (betalend);
- Public repositories (gratis en open-source);
- GitHub Enterprise (GitHub systeem opzetten op eigen servers).

Via GitHub (of BitBucket, GitLab,…) kunnen meerdere mensen samenwerken aan 1 of meerdere repositories d.m.v. een remote server.
Elk teamlid van een repository heeft meestal een lokale kopie van deze repository staan, en kan zijn/haar lokale wijzigingen doorvoeren naar de remote server.
De andere teamleden kunnen dan hun lokale kopie synchroniseren met de remote server.

GitHub account

We zullen voor het academiejaar 2021-22 éénmalig een GitHub-account aanmaken, deze bevat de volgende samenstelling: gdmgent-{your AHS-loginname}, bv.: gdmgent-krisraes (allemaal kleine letters). Vermeld ook jouw AHS-mailadres en jouw echte naam. Vergeet ook niet een duidelijke foto te plaatsen bij jouw profiel. Heb je al een account op GitHub die jouw AHS-mailadres gebruikt, dan volstaat het enkel jouw gebruikersnaam te veranderen via Account. Deze wijziging duurt een paar minuten en kan ervoor zorgen dat o.a. links naar persoonlijke Gists niet meer zullen werken.

GitHub biedt ook gratis een packs for education aan. Dit is o.a. handig om **private repositories** aan te maken.

Open vervolgens de terminal (macOS) of command prompt (Windows) waarin we onze **GitHub-loginnaam** en **GitHub e-mailadres** zullen linken aan onze computer:

```
git config --global user.name "gdmgent-{your GitHub-username}"
git config --global user.email "{your GitHub-email address}"
```

Bijvoorbeeld:

```
git config --global user.name "gdmgent-krisraes"
git config --global user.email "kristof.raes@student.arteveldehs.be"
```

.NET Core

Voor het programmeren in C# met Visual Studio Code gebruiken we .NET Core welke een cross-platform versie is van .NET voor het maken van websites, services en console applicaties.

Installatie

Het installatiebestand kan je vinden op: https://dotnet.microsoft.com/download

macOS

1. Download de PKG file voor de .NET Core 3.1 SDK voor Mac (64 bits);
2. Dubbel-klik op het gedownloade archiveerbestand om te inhoud uit te pakken, en vervolledig de stappen voor het installeren van .NET Core op jouw pc.

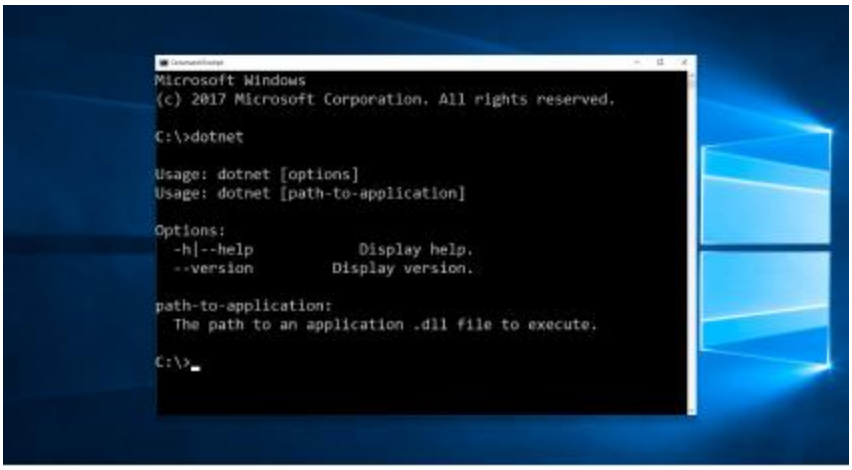
3. Verifieer de installatie door terminal te openen en `dotnet ↲`
   dan zie je de argumenten die je kan meegeven met het commando, een teken dat .NET Core correct geïnstalleerd is (zie screenshot hieronder).

4. Start met het maken van je eerste **Hello World** console applicatie in 5 minuten.

Windows

Download de 64-bit versie om Microsoft Visual Studio Code ten volle te gebruiken!

1. Download het Visual Studio Code installatieprogramma voor Windows.
2. Na de download, voer het installatieprogramma (`dotnet-sdk-{version}-win-x64.exe`) uit en vervolledig de stappen voor het installeren van .NET Core op jouw pc.
3. De standaardlocatie van het VS Code programma bevindt zich onder `C:\Program Files\dotnet\`.
4. Verifieer de installatie door terminal te openen en `dotnet ↲`
   dan zie je de argumenten die je kan meegeven met het commando, een teken dat .NET Core correct geïnstalleerd is (zie screenshot hieronder).



5. Start met het maken van je eerste **Hello World** console applicatie in 5 minuten.

Datatypen

Wat is data?

Data is een gegevensverzameling, ook wel dataset genoemd welke in de meeste gevallen gepresenteerd is in een tabelvorm.

Waarbij de rijen, ook wel records genoemd, in de tabel overeenstemt met een lid uit de gegevensverzameling.

De kolom stemt overeen met een variabele. Elke rij bevat een waarde voor de verschillende variabelen of attributen.

Voorbeelden van variabelen: datum, tijdstip, getal, tekst,…

Een dataset heeft verschillende kenmerken welke de eigenschappen en structuur bepalen zoals onder andere het datatype van de variabelen, het aantal variabelen

Andere vormen van datasets welke geen tabelvorm zijn: XML, JSON,…

De waarden kunnen getallen zijn, maar ook gegevens van nominaal of ordinaal niveau zijn. Voor elke variable zullen normaal gesproken de waarden van hetzelfde niveau zijn, al kunnen er wel gegevens ontbreken, wat op een of andere manier dient te worden aangegeven.

Datatypes

Een datatype of gegevenstype is een specifieke soort van data, het datatype bepaalt welke waarde de variabele kan bevatten.

De datatypes kan je verdelen in primitieve, enkelvoudige en samengestelde datatypes.

Primitieve datatypes.

Een primitief datatype is gedefinieerd door de programmeertaal zelf.

Meest voorkomende primitieve datatypes:

- Boolean
- Char
- Integer
- Double
- Decimal

Overzicht

Verkorte naam	.NET Class en type	Bereik	Grootte	Standaardwaarde
Gehele getallen				
byte	Byte, unsigned integer	0 tot 255	8	0
sbyte	SByte, signed integer	-128 tot 127	8	0
int	Int32, signed integer	-2.147.483.648 tot 2.147.483.647	32	0
uint	UInt32, unsigned integer	0 tot 4.294.967.295	32	0
short	Int16, signed integer	-32.768 tot 32.767	16	0
ushort	UInt16, unsigned integer	0 tot 65535	16	0
long	Int64, signed integer	-9.223.372.036.854.775.808 tot 9.223.372.036.854.775.807	64	0
ulong	UInt64, unsinged integer	0 tot 18.446.744.073.709.551.615	64	0
Reële getallen				
float	Single, single-precision floating point type	-3,402823e38 tot 3,402823e38	32	0.0F
double	Double, double-precision floating point type	-1,79769313486232e308 tot 1,79769313486232e308	64	0.0D
decimal	Decimal, Precise fractional or integral type that can represent decimal numbers with 29 significant digits	±1.0 × 10e−28 tot ±7.9 × 10e28	128	0.0M
Tekst				
string	String, tekenreeks			
Andere types				
char	Char, 1 unicode karakter	U +0000 tot U +ffff	16	‘\0’
bool	Boolean, True (1) of False (0)	True (1) of False (0)	8	False (0)

Verkorte naam	.NET Class en type	Bereik	Grootte	Standaardwaarde
object	Object, basistype van alle andere types			

Datatypes

Enkelvoudige datatypes.

Is een primitief datatype, ook wel simpel datatype genoemd, of een datatype gedefinieerd op basis van een primitief datatype. De voorwaarde is dat een enkelvoudig datatype enkel als geheel kan worden uitgelezen en/of gemanipuleerd.

Numerieke variabelen

Beide soorten variabelen hebben hun beperkingen:

- Integers kunnen enkel gebruikt worden voor gehele getallen.
- Floating point variabelen kunnen problemen veroorzaken met nauwkeurigheid door het afronden van getallen na de komma (getallen vergelijken, niet te gebruiken als teller).
- Bij een float kan 1,5 gelijk zijn aan 1,5000001.

Daarom bestaat er nog een type numerieke variabele: de **decimal**.

- Met een decimal zijn exacte berekeningen met reële getallen mogelijk.
- Een decimal heeft een bereik van 10-28 tot 1028.
- Een decimal is exact: 1,5 is ook echt 1,5
- Berekeningen met decimals zijn echter traag.
- Decimals kunnen ook niet als tellers gebruikt worden.

Fahrenheit met int

```
// Zet de temperatuur 100°F om naar graden Celcius.
int fahrenheit = 100;
int = (fahrenheit – 32) * (5.0 / 9.0);
```

De berekening zal altijd resulteren in 0 omdat 5/9 als resultaat nul heeft.
Het cijfer na de komma wordt immers afgekapt (niet afgerond).

Fahrenheit met double

```
// Zet de temperatuur 100°F om naar graden Celcius.
double fahrenheit = 100;
double celcius = (fahrenheit – 32) * (5.0 / 9.0);
```

De berekening zal resulteren in: 37.7777777777778
Merk op dat er nu afronding plaatsvindt i.p.v. afkapping.

Booleans

Een boolean is een logisch type variabele, welke twee waarden kan hebben namelijk true en false.

Een boolean kan niet gecast worden naar een ander type.

```
bool myBoolean = true;
```

Char

Een char is ene datatype welke karakters kan bewaren, hiermee bedoelen we de eigenlijke letters en cijfers.

- De letter ‘a’ is een karakter.
- De visuele voorstelling van het cijfer ‘1’ is ook een karakter.

Karakters zijn niet beperkt tot het alfabet en de cijfers, ook Chinese, Arabische, etc. tekens zijn karakters.
Karakters worden in C# opgeslagen in een variabele van het type **char**.
Een char variabele kan 1 en slechts 1 karakter bevatten.
Een char variabele gebruikt 2 bytes (16 bits) geheugen.
Een char is een ‘counting’ type. Je kunt een variabele van het type char dus incrementeren.

Naast de letters van het alfabet zijn er ook speciale karakters die we gebruiken om tekst op te maken.
Speciale karakters kun je herkennen aan de backslash.

Karakter	Beschrijving
'\n'	Nieuwe lijn
'\t'	Tab
'\r'	Carriage return (cursor naar begin van de huidige regel brengen)
'\\'	Een backslash

String

Een karakter opslaan is handig, maar hoe sla je een hele zin op in een variabele?
Dat gaan we doen in een string variabele.

Een string is een groep van karakters, genoteerd tussen dubbele quotes: “mijn string”.
Welke men ook wel eens een tekenreeks noemt.
Een string declaratie kun je niet spreiden over twee regels, gebruik speciale karakters in de string om een nieuwe lijn te krijgen.
Een string kun je niet gebruiken als teller en een string heeft geen vaste lengte.

```
string myString = "Hello world!";
```

Je kunt strings samenvoegen door // declareer en initialiseer het gebruik van +
Je kunt een lege string gebruiken met " "

Voor een string bestaan een hele reeks operatoren:
myString.length: geeft de lengte van de string weer.
Compare(): vergelijken van twee strings.
myString.Format(): een string een bepaalde opmaak geven
myString.trim(): een stuk van de string knippen
En nog veel meer...

Samengestelde datatypes.

Samengestelde datatypes of complexe datatypes bestaan uit meerdere simpele types die afzonderlijk kunnen worden gemanipuleerd en uitgelezen.

Deze structuren welke resulteren in arrays, klasse, lijsten,… die samengesteld zijn uit elementen die op zichzelf ook kunnen bestaan uit samengestelde datatypes zoals lijsten van lijsten, geneste structuren,…

String

Is een reeks van karakters.

Conversie

Converteren van datatype wil eigenlijk zeggen dat je het huidige datatype wil wijzigen in een ander datatype.

Convert klasse.

Uitvoeren van een conversie kan je doen via de Convert klasse. Je kan bijvoorbeeld een double waarde in een sbyte datatype steken.

```
1  sbyte csb = Convert.ToSByte(2.41255);
2  Console.WriteLine("Converteren van double naar sbyte: {0}", csb);
```

Referentie Convert klasse Referentie Convert methodes

Impliciete en expliciete conversie.

```
1 int r = 5.0; // Geen impliciete conversie voor double naar int.
2 int s = (int)5.0; // Expliciete conversie.
```

Variabelen

Wat is een variabele?

Een variabele kan je vergelijken met een doos waar je tijdelijk iets in gaat stoppen.

Om daar even op verder te gaan... Je kan verschillende dozen hebben die je gaat stockeren in een rek.
Het rek in dit voorbeeld kan je vergelijken met het geheugen van een device (pc, tablet, smartphone,...), elke variabele neemt een bepaalde hoeveelheid geheugen in van een computer.

Soms dien je als programmeur acties uit te voeren waarbij je iets tijdelijk in een doos wilt stoppen om dan later acties op uit te voeren.

Denk aan een getal 1 en getal 2 waarmee je een som, verschil of een product wil uitvoeren. Hierbij zijn som, verschil en product een bepaalde actie die je gaat uitvoeren op de waarde die in de variabele zit op dat moment.

Zoals je al door hebt zijn variabelen essentieel voor een programmeur.
Een variabele krijgt een naam en een waarde en is van een bepaald type.
Er zijn verschillende types van variabelen.

De variabele kan je gebruiken in een applicatie voor:

- Het ophalen van de waarde.
- Het wijzigen van de waarde.

Een variabele ga je declareren als een bepaald type, in die variabele ga je enkel informatie kunnen opslaan van dat bepaalde type.
Voor de declaratie van een variabele moet je dus een type en naam opgeven voor de variabele. De waarde voor de variabele is optioneel.

Voor een waarde toe te kennen aan een variabele gebruik je het '=' teken.
Het toekennen van een initiële waarde aan een variabele noemt men het initialiseren van de variabele.

Declaratie

- Is het reserveren van een deeltje geheugen van de computer voor een **variabele** met naam **getal** van het type **int**.
- Komt overeen met het nemen van een **doos** van het model **int** en plak daar een label **getal** op.

In C# declareer je een variabele als volgt:

<data type> <name>

Voorbeeld:

```
int getal;
```

Initialisatie

Bij de initialisatie van een variabele ga je over tot het toekennen van een waarde.

```
getal = 0;
```

Voorbeeld van een declaratie en initialisatie van een variabele name van het type string met access specifier private:

```
private string name = "Kristof Raes";
```

Belangrijke regels

- Een variabele begint altijd met een kleine letter.
- Bestaat de naam uit meerdere woorden dan, laten we elk woord beginnen met een hoofdletter maar er komen geen spaties aan bod.
- Dit noemt men camelCasing.
- Je kunt een variabele declareren en initialiseren in 1 lijn code.

- Je kunt een variabele slechts gebruiken na initialisatie.
- Eerst declareren en dan initialiseren.
- Je kunt dezelfde variabele geen twee declareren.

Meer over informatie over types van variabele of datatypes kan je vinden bij **Datatypen**.

Lokale variabelen

Lokale variabele

Een lokale variabele is een variabele die enkel gekend is binnen de scope van een functie of een methode, daarbuiten kan je die niet gebruiken omdat die daar niet gekend is.
Omdat de variabele gedeclareerd is binnen die functie of methode.

Voorbeeld van een lokale variabele type van het type string, je kan de variabele enkel gebruiken binnen de methode GetTypeText:

```
static void GetTypeText(object obj)
{
    string type = obj.GetType().ToString();
    Console.WriteLine(type);
}
```

Globale variabelen

Globale variabele

Een globale variabele is gekend voor het hele programma en ga je bovenaan in je klasse gaan definiëren.

Voorbeeld van een globale variabele counter van het type int, je kan de variabele gebruiken in alle members van de klasse Program (dus in de methode Main en de functie GetText,...:

```
class Program
{
    0 references
    public int counter;
    0 references
    static void Main(string[] args)
    {
        Console.WriteLine("Hello World!");
    }

    0 references
    static string GetText(string input)
    {
        return input;
    }

    0 references
    static void GetType(string input)
```

Casting

Casting

Soms is het nodig om data van het ene type om te zetten naar het andere (bv. faculteitsberekening).
Een long variabele kan perfect de waarde van een int variabele krijgen omdat een long (8 bytes) ‘groter’ is dan een int (4 bytes). Dit noemt men impliciete casting.
Omgekeerd kun je niet zomaar de waarde van een long in een int persen. De C# compiler zal hier een fout geven.
Als je echter zeker bent dat de waarde van de long nooit buiten het bereik van een int valt, dan kun je dit toch doen door middel van expliciete casting.

Een voorbeeld:

```
int a = 6;
int b = 4;
double c = a/b; // c is 1
// Expliciete casting
double c = (double)a/b; // c is 1,5
// Impliciete casting
int a = 6;
double b = 4;
double c = a/b; // c is 1,5
```

Volgorde van bewerkingen

Volgorde van bewerkingen.

Als er geen haakjes gebruikt worden, dan gebeuren de bewerkingen in volgens de voorrangsregels:
1. Increment en decrement.
2. Vermenigvuldigen, delen en modulo.
3. Optellen en aftrekken.
Gebruik dus steeds haakjes als je niet zeker bent!

Operatoren

Operatoren

In code moeten er dikwijls veel berekeningen uitgevoerd worden, elementen met elkaar vergeleken of logische keuzes gemaakt worden. Hiervoor kan je gebruikmaken van operatoren, de verschillende operatoren kan je hieronder vinden in de specifieke groepen.

Overzicht rekenkundige/wiskundige operatoren.

Een wiskundige bewerking voer je uit met één of meerdere operanden en een operator.
De meest gekende operatoren zijn: +, -, *, /
Maar er zijn er ook nog andere…

Operator	**Omschrijving**
++	waarde verhogen met 1
--	waarde verminderen met 1
x + y	som
x - y	verschil
x / y	delen
x * y	vermenigvuldigen
x % y	rest na deling
+x, -x	veranderen van teken

Assignment operatoren

Om aan een variabele een waarde toe te kennen maken we gebruik van een assignment operator.

De meest gekende is het =-teken, maar er bestaan ook nog andere…

Veronderstel: c = 3, d = 5, e = 4, f = 6, g = 12;

Assignment operator	Voorbeeld	Beschrijving	Waarde
+=	c += 7	c = c + 7	10

-=	d -= 4	d = d - 4	1
*=	e *= 5	e = e * 5	20
/=	f /= 3	f = f / 3	2
%=	g %= 9	g = g % 9	3

Overzicht tekenreeks operatoren.

Operator	**Omschrijving**
+	tekenreeksen worden hiermee samengevoegd

Overzicht logische operatoren.

Operator	**Omschrijving**
!	NOT
&&	AND
||	OR

Overzicht relationele operatoren.

Operator	**Omschrijving**
x == y	is gelijk aan
x != y	is niet gelijk aan
x > y	is groter dan
x < y	is kleiner dan
x >= y	is groter dan of gelijk aan
x <= y	is kleiner dan of gelijk aan

Char – speciale karakters

- Naast de letters van het alfabet zijn er ook speciale karakters die we gebruiken om tekst op te maken.
- Speciale karakters kun je herkennen aan de *backslash*.

Karakter	***Beschrijving***
'\n'	Nieuwe lijn
'\t'	Tab
'\r'	Carriage return (cursor naar begin van de huidige regel brengen)
'\\'	Een backslash

Constante

Constanten

Een constante is een variabele met een vaste waarde die bekend zijn tijdens het compileren en niet veranderen gedurende de hele levensduur van je applicatie of programma.
Dus wanneer er een waarde toegekend is aan een constante kan je deze niet meer veranderen.
Constanten zijn handig wanneer je een bepaald getal vaak nodig hebt in een programma.
In plaats van overal 0,80 te gebruiken voor interest maak je een constante met de naam InterestRatio die je in plaats van het getal gebruikt.
Dit maakt het programma leesbaarder. Constanten declareer je met het **const** keyword en begint altijd met een hoofdletter.
Een constante moet in 1 regel tegelijk gedeclareerd en geïnitialiseerd worden.

Voorbeelden van constanten:

- aantal maanden in een jaar
- aantal dagen in week
- aantal uren in een dag

Declaratie van een constante doe je

```
public const int Days = 7;
```

Referentie: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/constants.

Referentie: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/how-to-define-constants.

Sequentie

Sequentie/opeenvolging.

Een sequentie is een opeenvolging van instructies die een voor een uitgevoerd worden in de volgorde waarin ze geschreven zijn.

Hieronder een voorbeeld van een sequentie:

```
getValue();
checkValue();
saveValue();
```

Afbeelding 1:

```
RegistratieStudent();   // Het uitvoeren van de registratie van een student.
CreatieLogin(); // Het aanmaken van een login voor de student.
ActiveerLogin(); // Het activeren van de login op de verschillende platformen (O365, MijnDinar,...).
```

Afbeelding 2:

```
int getal = 0; // Declaratie en initialisatie van de integer getal die we gelijkstellen aan 0.
Console.WriteLine("Geef een getal in tussen 0 en 5."); // Afdrukken van de lijn "Geef een getal in tussen 0 en 5." in de console applicatie.
getal = Convert.ToInt32(Console.ReadLine());
/*
    // We lezen de waarde uit (ingegeven via console applicatie) via de functie ReadLine().
    // De teruggegeven waarde gaan we converteren via de functie ToInt32() van de Convert klasse, zodat we een integer bekomen.
    // De waarde die we hier geconverteerd hebben gaan we in de variabele getal opslagen.
*/

if (getal > 0) // Conditie of voorwaarde; als de variabele getal groter is dan 0, dan voeren we alles uit tussen de accolades.
{
    // Statements; wat we gaan uitvoeren als voldaan is aan de conditie of de voorwaarde.
    Console.WriteLine("Het getal is groter dan 0!"); // Afdrukken van de lijn "Het getal is groter dan 0!"in de console applicatie.
}
```

Selectie

Keuze/selectie.

Intro tot keuze.

Een selectie of keuze wordt meestal uitgevoerd door gebruik te maken van het “if-statement”. Afhankelijk van het resultaat van het “if-statement” zal de handeling tussen de accolades {}, na de if, al dan niet uitgevoerd worden.

We kunnen dus een keuze coderen door middel van een voorwaarde:

if voorwaarde *then* handeling uitvoeren als aan voorwaarde voldaan is *else* else-component *endif*

Als er aan de voorwaarde voldaan is dan zal code uitgevoerd worden dus in het then deel aanwezig is, tussen de accolades {} na de voorwaarde. Indien niet aan de voorwaarde is voldaan dan zullen de handelingen aanwezig in het else deel uitgevoerd worden. De voorwaarde van het “if-statement” kan zowel bestaaan uit een enkelvoudige als uit een samengestelde logische formulering, zie relationele en logische operatoren.

If-statement.

```
int getal = 0;
Console.WriteLine("Geef een getal in tussen 0 en 5");
getal = Convert.ToInt32(Console.ReadLine());

if (getal > 0)// conditie of voorwaarde
{
   // statements
   Console.WriteLine("Het getal is groter dan 0!");
}
```

De conditie of voorwaarde is een logische formulering (true of false als resultaat) waaraan voldaan dient te worden (true als resultaat van de conditie), om de statements tussen accolades uit te voeren, is dat de waarde van de integer variabele getal groter dient te zijn dan 0. Als de conditie een true oplevert dan zal het statement uitgevoerd worden om volgende lijn af te drukken in de console: Het getal is groter dan 0! Als de conditie een false oplevert zal het programma de lijnen code uitvoeren na de accolade van het if-statement.

IfElse-statement.

```
int getal = 0;
Console.WriteLine("Geef een getal in tussen 0 en 5");
getal = Convert.ToInt32(Console.ReadLine());

if (getal > 0)// conditie of voorwaarde
{
   // statements
   Console.WriteLine("Het getal is groter dan 0!");
}
```

```
else
{
   // statements
   Console.WriteLine("Het getal is niet groter dan 0!");
}
```

De conditie of voorwaarde is een logische formulering (true of false als resultaat) waaraan voldaan dient te worden (true als resultaat van de conditie), om de statements tussen accolades uit te voeren, is dat de waarde van de integer variabele getal groter dient te zijn dan 0. Als de conditie een true oplevert dan zal het statement uitgevoerd worden om volgende lijn af te drukken in de console: Het getal is groter dan 0! Als de conditie een false oplevert zal het programma de lijnen code uitvoeren na de accolade van het else-statement.

IfElseIfElse-statement.

```
int getal = 0;
Console.WriteLine("Geef een getal in tussen 0 en 5");
getal = Convert.ToInt32(Console.ReadLine());

if (getal > 0)// conditie of voorwaarde
{
   // statements
   Console.WriteLine("Het getal is groter dan 0!");
}
else if (getal == 5)
{
   // statements
   Console.WriteLine("Het getal is gelijk aan 5!");
}
else
{
   // statements
   Console.WriteLine("Het getal is niet groter dan 0!");
}
```

Zoals in bovenstaand voorbeeld is te zien kan men ook nog opteren om de ifelse uit te breiden met een elseif zodat je hierbij een extra keuze kan toevoegen tot je klassieke ifelse-structuur.

Geneste IfElse-structuur.

Bij een geneste IfElse-structuur is het zo dat je meerdere controlestructuren gaat hebben op verschillende niveaus. Hieronder kan je een voorbeeld vinden van een geneste IfElse-structuur.

```
int getal = 0;
Console.WriteLine("Geef een getal in tussen 0 en 5");
getal = Convert.ToInt32(Console.ReadLine());

if (getal > 0)// conditie of voorwaarde
{
   // statements
   if (getal < 5)
```

```
    {
        // statements
        if(getal >= 1 && getal < 4)
        {
            Console.WriteLine("Het getal is kleiner dan 5!");
        }
    }
    else if(getal == 5)
    {
        Console.WriteLine("Het getal is gelijk aan 5!");
    }
    else
    {
        // statements
        Console.WriteLine("Het getal is groter dan 0!");
    }
}
```

Switch-statement.

Vanaf je meer dan 2 keuzemogelijkheden dient in te bouwen is de switch-structuur efficiënter.

De switch-opdracht heeft de algemene structuur:

```
        switch (getal) // integer expressie
        {
            case 1: // constant expressie
                Console.WriteLine("Het getal is 1."); //statement
                break; //jump statement
            case 2:
                Console.WriteLine("Het getal is 2.");
                break;
            case 3:
            case 4:
                Console.WriteLine("Het getal is 3 of 4.");
                break;
            case 5:
                Console.WriteLine("Het getal is 5.");
                break;
            default:
                Console.WriteLine("Het getal is een ander getal... :-(");
                break;

        }
```

De expressie, welke achter switch staat, van de switch-structuur is meestal een variabele. De waarde van de variabele gaat dan vergeleken worden met de constant expressie. Als de waarde overeenstemt met een bepaalde constant expressie, voorbeeld case 5, dan zal het statement na de constant expressie uitgevoerd worden. Elke mogelijkheid, case x dus, eindigt met het jump-statement waar men break; plaatst. break; zal er voor zorgen dat er geen code meer zal uitgevoerd worden van de switch-structuur na de break; indien de case in kwestie uitvoerbaar is.

Goed om weten:

- Bij een een switch kan je geen rekenkundige of logische operatoren gebruiken.
- Bij case kan je enkel waarden meegeven.
- Een case werkt niet met bereiken, voorbeeld: 5-8.
- Je kan wel een case definiëren voor 2 mogelijkheden, zoals bij het voorbeeld hierboven waarbij hetzelfde mag uitgevoerd worden voor case 3 en 4.

Iteratie

Herhaling/iteratie.

Een iteratie zorgt ervoor dat je een opeenvolging van instructies een x aantal keren kunt herhalen. Men spreekt ook wel van een lus of een herhaling.

Enkele voorbeelden:

- Overlopen van een lijst.
- Vermenigvuldigingstafel opstellen.

For loop.

Een **for loop** of ook wel **begrensde herhaling** is er op voorhand vastgelegd hoeveel keren de instructies herhaald moeten worden.

```
for (int i = 0; i < length; i++)
{
   // instructie
}
```

While loop.

Dit is de **voorwaardelijke herhaling** met **aanvangsvoorwaarde**.
Zolang er dus aan een bepaalde voorwaarde voldaan is worden de statements tussen de accolades {} uitgevoerd.
Indien er niet meer voldaan is aan de voorwaarde dan worden de instructies na de loop uitgevoerd.

Bij dit type van iteratie of herhaling is de aanvangsvoorwaarde, namelijk teller = 0, al op voorhand onderzocht voor de loop voor de eerste keer wordt uitgevoerd. Indien er niet voldaan kan worden aan de voorwaarde dan gaat de loop nooit uitgevoerd worden.

```
while (true)
{

}
```

Do While loop.

Dit is de **voorwaardelijke herhaling** met **afbreekvoorwaarde**, bij dit type van herhaling wordt de voorwaarde of conditie pas onderzocht nadat de loop voor de eerste keer werd uitgevoerd.

Bij de **for** en de **while** gaan de statements of instructies pas uitgevoerd worden als aan de voorwaarde is voldaan, bij de **do while** niet.

De herhaling met afbreekvoorwaarde is vooral handig om de input of invoer te controleren. Bij de input moet de gebruiker minstens 1 keer iets in te geven en de applicatie vraagt de input telkens opnieuw tot de input voldoet aan de voorwaarde(n).

```
do
{

 } while (true);
```

ForEach loop.

Door middel van een ForEach loop ga je alle elementen in collectie overlopen.

```
foreach (var item in collection)
{

}
```

Methoden

Wat is een methode?

Een applicatie is een verzameling van kleine, doch simpele modules (kleine applicaties op zich).
Een applicatie is niet één methode met daarin alle functionaliteit. Splits de functionaliteit in verschillende op zich bestaande deeltjes.

Een methode of procedure is een reeks van commando's welke in volgorde uitgevoerd worden. Een methode geeft geen returnwaarde terug. Deze is van het type void.

Elk C# programma bestaat ten minste uit 1 klasse met een methode Main van het type void.

Voorbeeld:

```
static void GetType(string input)
{
    Console.WriteLine(input.GetType());
}
```

De Main-methode van elke console applicatie:

```
namespace oefening1
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Packaging

Gerelateerde klassen (Persoon, Student, Medewerker, …) zitten vervat in dezelfde bibliotheek (library) of namespace.
Klassen die gebruikt kunnen worden in andere applicaties maken we best aan in een apart

project, namelijk een Class Library.
De .NET FrameWork Class Library is zo'n bibliotheek die we kunnen gebruiken in verschillende applicaties. Bevat voorgedefinieerde klassen om:
Wiskundige bewerkingen, manipulaties van strings, input/output, database operaties, …

Modulerisatie van een applicatie door de taken te separeren in op zich bestaande (selfcontained) eenheden (units) --> USER DEFINED METHODS
Bouwstenen die we kunnen hergebruiken (software reuse), herhaalde implementatie van code vermijden --> generalisatie
De applicatie wordt dus beter beheersbaar en kan eenvoudigweg uitgebreid worden.

Methoden worden uitgevoerd (invocatie) door een methode, die toegankelijk is, aan te roepen.
Deze methode zal vervolgens opdrachten uitvoeren en eventueel een waarde teruggeven aan de aanvrager (caller of client).
Een baas vraagt aan zijn medewerkers om taken uit te voeren, wanneer deze taken vervuld zijn door de medewerker, zal deze rapporteren aan zijn baas.

Een methode in C# heeft steeds de volgende opbouw:
[accessModifier] returnType MethodName ([parameterList])
{
statements
}
accessModifier: public of private
returnType: type dat de functie teruggeeft
FunctionName: naam van de functie
parameterList: lijst met parameters

Bepaalde methodes hebben een zuiver proceduraal karakter. Ze geven geen informatie terug en krijgen bij de declaratie vooraan het codewoord void mee.
Bij andere methodes worden bewerkingen uitgevoerd met als gevolg dat een resultaat moet teruggegeven worden. Deze methodes zijn te beschouwen als functies. Het teruggeven van een waarde gebeurt met het sleutelwoord return. Bij de declaratie wordt de methode voorafgegaan door het datatype van de waarde die moet teruggegeven worden.

Methoden die geen waarde teruggeven, maar wel taken uitvoeren --> void datatype
Methoden die een waarde teruggeven en tevens taken uitvoeren --> non-void datatypen (string, int, custom datatypen, …)

Scope van declaraties.

De levensduur van een declaratie, variabele.
Definiëren (declareren) we een nieuwe variabele binnen een methode, dan is deze variabele enkel aanspreekbaar binnen deze methode.

Plaatsen we een variabele of declaratie buiten een methode, dan is deze variabele aanspreekbaar binnen elke methode uit deze klasse.

Parameters (arguments)

Informatie wordt naar methodes doorgegeven via parameters.
Dit kan op twee manieren:

- by value: een kopie van de waarde (kopie van de inhoud) van de variabele wordt doorgegeven Het originele wordt niet aangepast.
- by reference: de referentie (het adres) van de variabele wordt doorgegeven.

Normaal worden parameters altijd doorgegeven bij value.
In het andere geval moet men bij de methodedeclaratie en het aanroepen van de methode de parameter laten voorgaan door het sleutelwoord **ref**

Pass-by-value:
Een kopie van de waarde wordt doorgegeven.
Pass-by-reference:
De referentie naar het adres wordt doorgegeven.

Static methoden

Bepaalde methodes zijn objectgebonden en zijn alleen toe te passen op instanties van de klasse.
Andere zijn objectonafhankelijk en zijn alleen toe te passen op het niveau van de klasse. Het zijn de zogenaamde static methodes.
Meer hierover bij het gedeelte over OOP.

Main methode

De Static Main Methode uit console applicaties.
Tijdens het starten van een applicatie, bestaan er nog geen objecten, vandaar dat we een statische methode, in dit geval Main, moeten aanspreken welke de entry point is van onze applicatie.
Het is mogelijk om command line parameters op te geven (parameters voor een Main methode).
Het is eveneens mogelijke om meerdere klassen met een Main methode te voorzien binnen eenzelfde project, vooral voor testdoeleinden.

Functie

Wat is een functie?

Hetzelfde als een methode met dat verschil dat een functie wel een waarde teruggeeft we spreken in dat geval van een return.

Een returntype kan zijn van het type string, int, double, byte, bool,...

Voorbeeld:

```
static string GetText(string input)
{
    return input;
}
```

Commentaar

Commentaar plaatsen.

Het is aanbevolen je broncode altijd te voorzien van commentaar zodat je later gemakkelijk wijzigingen kunt uitvoeren en de broncode ook leesbaar/begrijpbaar is voor anderen.

Commentaar in C# kunnen we doen door volgende tekencombinatie // voorop onze commentaar te plaatsen.

Je kan kiezen voor 1 regel commentaar of voor meerdere regels commentaar:

```
// 1 regel commentaar

/*
   Meerdere
   regels
   commentaar.
*/
```

Documenteren van code.

Het is aanbevolen je broncode altijd te documenteren, door te voorzien in commentaarlijnen zodat je later gemakkelijk wijzigingen kunt uitvoeren (onderhoudbaarheid) en de broncode ook leesbaar/begrijpbaar is voor anderen.

Ondanks een veelvoorkomend misverstand wordt het documenteren van code niet gebruikt om te beschrijven wat de code doet maar wel hoe dit gebeurt.

Commentaar (single line en multi line) in C#.

Het plaatsen van commentaar in C# kunnen we doen door volgende tekencombinatie // (2 slashes) voor onze commentaartekst te plaatsen.

Je kan kiezen voor 1 regel commentaar of voor meerdere regels commentaar:

```
// 1 regel commentaar

/*
   Meerdere
   regels
   commentaar.
*/
```

Commentaar ter documentatie (XML).

Een stap verder is XML-commentaar ter documentatie en ziet er bijna hetzelfde uit als gewone commentaar, maar dan met embedded XML. Hierbij kan je ook zowel single line als multi line commentaar gebruiken. Je schrijft ze op dezelfde manier als gewone commentaar, maar met een extra karakter. Single line XML commentaar ter documentatie gebruikt 3 slashes (///) in plaats van 2. De multi line variant gebruikt een extra asterisk (*).

```
class Gebruiker
{
   /// <summary>
   /// De Naam van de Gebruiker.
   /// </summary>
   public string Naam { get; set; }

   /**
   * <summary>De Leeftijd van de Gebruiker.</summary>
   */
   public string Leeftijd { get; set; }
}
```

Hierboven zie je beide vormen. De eerste variant met gebruik van 3 slashes (///) wordt meestal gebruikt voor commentaar ter documentatie.

Voorbeeld XML documentatie van broncode.* Voorbeeld IntelliSense.*

Hierbij een overzicht van de best pratices:

- Omwille van de consistentie dien je alle publiek zichtbare types en members te documenteren. Als je het moet doen, doe het dan voor alles.
- Private members kunnen ook gedocumenteerd worden via XML commentaar. Dit zorgt er wel voor dat de innerlijke (mogelijk vertrouwelijke) werking vrijgegeven wordt van uw code.
- Een echt minimum moet zijn dat types en hun members een <summary>-tag hebben omdat die informatie nodig is voor de IntelliSense.
- De tekst in documentatie dient te bestaan uit volledige zinnen en eindigen met een punt.
- Partial klassen worden volledig ondersteund en documentatie wordt samengevoegd tot één item voor dat type.
- De compiler verifieert de syntax van de <exception>, <include>, <param>, <see>, <seealso> en <typeparam> tags.
- De compiler valideert de parameters die bestandspaden en verwijzingen naar andere delen van de code.

Referentie XML documentation comments.

Takenlijst via commentaar.

Je kan een takenlijst samenstellen door in commentaar deze items toe te voegen, je dient deze lijn wel te starten met de single line commentaar als startsequentie direct gevolgd door TODO, FIXME, NOTE, HACK.

Hieronder een voorbeeld:

```
Gebruiker g = new Gebruiker();
//TODO: het instellen van de naam van de gebruiker...
//FIXME: eventueel een constructor voorzien in de klasse Gebruiker...
```

In de Todo view, als je gebruik maakt van de Todo+ extensie in Visual Studio Code, krijg je dan een overzicht van de items volgens categorie en dit per bestand.

Voorbeeld takenlijst in broncode via Todo+.* Voorbeeld takenlijst in broncode via TODO Highlight.*

Referentie extensie TODO Highlight. Referentie extensie Todo+.

Exceptions

Exceptions

Een exception is een fout die gegooid wordt via keyword throw. Het type van exception hangt af van de fout die opgetreden is.
Het afhandelen van exceptions noemt men Exception Handling.

Je kan zelf een exception gooien op volgende manier:

```
throw new NotImplementedException();
throw new IndexOutOfRangeException();
```

Try-catch

Je kan een exception opvangen door rond de code die je wenst uit te voeren een try-catch te zetten.

De code dient dus in de try te komen en het opvangen van eventuele exceptions dien je in het catch blok te behandelen.

```
object o = null;
try
{
   int i = (int)o;   // Error
}
catch (InvalidCastException e)
{
   Console.WriteLine(e.Message);
}
```

Wil je verschillende types van exceptions opvangen dan kan je opteren voor verschillende catches te schrijven ofwel geef je geen argument mee tussen de haakjes en dan vang je alle exceptions op.

```
object o = null;
try
{
   int i = (int)o;   // Error
}
catch (InvalidCastException e)
{
   Console.WriteLine(e.Message);
}
catch (Exception e)
{
   Console.WriteLine(e.Message);
}
try
{
```

```
    throw Exception(); // Error
}
catch
{
    throw;
}
```

Try-finally

Als je bepaalde handelingen wenst uit te voeren die niet geïmpacteerd mogen geraken door een exception, dan opteer je voor een finally blok te plaatsen.

```
int i = 123;
string s = "tekst";
object obj = s;

try
{
    i = (int)obj;
    Console.WriteLine("Einde van try.");
}
finally
{
    Console.WriteLine("Uitvoeren van finally blok na het optreden van een exception.");
    Console.WriteLine("i: {0}", i);
}
```

Try-catch-finally

Als je de combinatie van alle 3 wenst dien je de 3 blokken toe te voegen.

```
int i = 123;
string s = "tekst";
object obj = s;

try
{
    i = (int)obj;
    Console.WriteLine("Einde van try.");
}
catch
{
    Console.WriteLine("!EXCEPTION!");
}
finally
{
    Console.WriteLine("Uitvoeren van finally blok na het optreden van een exception.");
    Console.WriteLine("i: {0}", i);
}
```

Referenties: Exception handling

- Waarvoor dient een exception?
- Van welke klasse dien je over te erven om een custom Exception aan te maken?
- Hoe vang je een exception op?

- Het verschil tussen een try-catch en een try-finally?

Naming convention

C# Coding Standards and Naming Conventions

SQL Server Object Name	Notation	Length	Plural	Prefix	Suffix	Abbreviation	Char Mask	Underscores
Class name	PascalCase	128	No	No	Yes	No	[A-z][0-9]	No
Constructor name	PascalCase	128	No	No	Yes	No	[A-z][0-9]	No
Method name	PascalCase	128	Yes	No	No	No	[A-z][0-9]	No
Method arguments	camelCase	128	Yes	No	No	Yes	[A-z][0-9]	No
Local variables	camelCase	50	Yes	No	No	Yes	[A-z][0-9]	No
Constants name	PascalCase	50	No	No	No	No	[A-z][0-9]	No
Field name	camelCase	50	Yes	No	No	Yes	[A-z][0-9]	Yes
Properties name	PascalCase	50	Yes	No	No	Yes	[A-z][0-9]	No
Delegate name	PascalCase	128	No	No	Yes	Yes	[A-z]	No
Enum type name	PascalCase	128	Yes	No	No	No	[A-z]	No

1. Do use PascalCasing for class names and method names:

```
public class ClientActivity
{
  public void ClearStatistics()
  {
    //...
  }
  public void CalculateStatistics()
  {
    //...
  }
 }
```

Why: consistent with the Microsoft's .NET Framework and easy to read.

2. Do use camelCasing for method arguments and local variables:

```
public class UserLog
{
        public void Add(LogEvent logEvent)
        {
                int itemCount = logEvent.Items.Count;
                // ...
                }
                }
        }
```

Why: consistent with the Microsoft's .NET Framework and easy to read.

3. Do not use Hungarian notation or any other type identification in identifiers

```
// Correct
int counter;
string name;
// Avoid
int iCounter;
string strName;
```

Why: consistent with the Microsoft's .NET Framework and Visual Studio IDE makes determining types very easy (via tooltips). In general you want to avoid type indicators in any identifier.

4. Do not use Screaming Caps for constants or readonly variables

```
// Correct
public static const string ShippingType = "DropShip";
// Avoid
public static const string SHIPPINGTYPE = "DropShip";
```

Why: consistent with the Microsoft's .NET Framework. Caps grap too much attention.

5. Use meaningful names for variables. The following example uses seattleCustomers for customers who are located in Seattle:

```
var seattleCustomers = from cust in customers
where cust.City == "Seattle"
select cust.Name;
```

Why: consistent with the Microsoft's .NET Framework and easy to read.

6. Avoid using Abbreviations. Exceptions: abbreviations commonly used as names, such as Id, Xml, Ftp, Uri.

```
        // Correct
UserGroup userGroup;
Assignment employeeAssignment;
// Avoid
UserGroup usrGrp;
Assignment empAssignment;
// Exceptions
CustomerId customerId;
```

```
XmlDocument xmlDocument;
FtpHelper ftpHelper;
UriPart uriPart;
```

Why: consistent with the Microsoft's .NET Framework and prevents inconsistent abbreviations.

7. Do use PascalCasing for abbreviations 3 characters or more (2 chars are both uppercase)

```
        HtmlHelper htmlHelper;
FtpTransfer ftpTransfer;
UIControl uiControl;
```

Why: consistent with the Microsoft's .NET Framework. Caps would grap visually too much attention.

8. Do not use Underscores in identifiers. Exception: you can prefix private static variables with an underscore:

```
// Correct
public DateTime clientAppointment;
public TimeSpan timeLeft;
// Avoid
public DateTime client_Appointment;
public TimeSpan time_Left;
// Exception (Class field)
private DateTime _registrationDate;
```

Why: consistent with the Microsoft's .NET Framework and makes code more natural to read (without 'slur'). Also avoids underline stress (inability to see underline).

9. Do use predefined type names instead of system type names like Int16, Single, UInt64, etc

```
        // Correct
string firstName;
int lastIndex;
bool isSaved;
// Avoid
String firstName;
Int32 lastIndex;
Boolean isSaved;
```

Why: consistent with the Microsoft's .NET Framework and makes code more natural to read.

10. Do use implicit type var for local variable declarations. Exception: primitive types (int, string, double, etc) use predefined names.

```
var stream = File.Create(path);
var customers = new Dictionary();
// Exceptions
int index = 100;
string timeSheet;
bool isCompleted;
```

Why: removes clutter, particularly with complex generic types. Type is easily detected with Visual Studio tooltips.

11. Do use noun or noun phrases to name a class.

```
public class Employee
{
}
public class BusinessLocation
{
}
public class DocumentCollection
{
}
```

Why: consistent with the Microsoft's .NET Framework and easy to remember.

12. Do prefix interfaces with the letter I. Interface names are noun (phrases) or adjectives.

```
        public interface IShape
{
}
public interface IShapeCollection
{
}
public interface IGroupable
{
}
```

Why: consistent with the Microsoft's .NET Framework.

13. Do name source files according to their main classes. Exception: file names with partial classes reflect their source or purpose, e.g. designer, generated, etc.

```
// Located in Task.cs
public partial class Task
{
                    //...
}
// Located in Task.generated.cs
public partial class Task
{
        //...
}
```

Why: consistent with the Microsoft practices. Files are alphabetically sorted and partial classes remain adjacent.

14. Do organize namespaces with a clearly defined structure:

```
// Examples
namespace Company.Product.Module.SubModule
namespace Product.Module.Component
namespace Product.Layer.Module.Group
```

Why: consistent with the Microsoft's .NET Framework. Maintains good organization of your code base.

15. Do vertically align curly brackets:

```
// Correct
class Program
{
  static void Main(string[] args)
  {
  }
}
```

Why: Microsoft has a different standard, but developers have overwhelmingly preferred vertically aligned brackets.

16. Do declare all member variables at the top of a class, with static variables at the very top.

```
// Correct
public class Account
{
  public static string BankName;
  public static decimal Reserves;
  public string Number {get; set;}
  public DateTime DateOpened {get; set;}
  public DateTime DateClosed {get; set;}
  public decimal Balance {get; set;}
  // Constructor
  public Account()
  {
     // ...
  }
 }
```

Why: generally accepted practice that prevents the need to hunt for variable declarations.

17. Do use singular names for enums. Exception: bit field enums.

```
// Correct
public enum Color
{
   Red,
   Green,
   Blue,
   Yellow,
   Magenta,
   Cyan
}
// Exception
[Flags]
public enum Dockings
{
  None = 0,
  Top = 1,
  Right = 2,
  Bottom = 4,
  Left = 8
}
```

Why: consistent with the Microsoft's .NET Framework and makes the code more natural to read. Plural flags because enum can hold multiple values (using bitwise 'OR').

18. Do not explicitly specify a type of an enum or values of enums (except bit fields).

```
// Don't
public enum Direction : long
{
  North = 1,
  East = 2,
  South = 3,
  West = 4
}
// Correct
public enum Direction
{
   North,
   East,
   South,
   West
}
```

Why: can create confusion when relying on actual types and values.

19. Do not use an "Enum" suffix in enum type names.

```
                // Don't
public enum CoinEnum
{
  Penny,
  Nickel,
  Dime,
  Quarter,
  Dollar
}
// Correct
public enum Coin
{
   Penny,
   Nickel,
   Dime,
   Quarter,
   Dollar
}
```

Why: consistent with the Microsoft's .NET Framework and consistent with prior rule of no type indicators in identifiers.

20. Do not use "Flag" or “Flags" suffixes in enum type names.

```
//Don't
[Flags]
public enum DockingsFlags
{
   None = 0,
  Top = 1,
  Right = 2,
  Bottom = 4,
   Left = 8
}
//Correct
```

```
[Flags]
public enum Dockings
{
  None = 0,
  Top = 1,
  Right = 2,
  Bottom = 4,
  Left = 8
}
```

Why: consistent with the Microsoft's .NET Framework and consistent with prior rule of no type indicators in identifiers.

21. Do use suffix EventArgs at creation of the new classes comprising the information on event:

```
// Correct
public void BarcodeReadEventArgs : System.EventArgs
```

Why: consistent with the Microsoft's .NET Framework and easy to read.

22. Do name event handlers (delegates used as types of events) with the "EventHandler" suffix, as shown in the following example:

```
public delegate void ReadBarcodeEventHandler(object sender, ReadBarcodeEventArgs e);
```

Why: consistent with the Microsoft's .NET Framework and easy to read.

23. Do not create names of parametres in methods (or constructors) which differ only the register:

```
// Avoid
private void MyFunction(string name, string Name)
```

Why: consistent with the Microsoft's .NET Framework and easy to read, and also excludes possibility of occurrence of conflict situations.

24. DO use two parameters named sender and e in event handlers. The sender parameter represents the object that raised the event. The sender parameter is typically of type object, even if it is possible to employ a more specific type.

Why: consistent with the Microsoft's .NET Framework

Why: consistent with the Microsoft's .NET Framework and consistent with prior rule of no type indicators in identifiers.

25. Do use suffix Exception at creation of the new classes comprising the information on exception:

```
// Correct
public void BarcodeReadException : System.Exception
```

Why: consistent with the Microsoft's .NET Framework and easy to read.

Offical Reference

1. MSDN General Naming Conventions
2. DoFactory C# Coding Standards and Naming Conventions

3. MSDN Naming Guidelines
4. MSDN Framework Design Guidelines
5. Microsoft .NET - Programming Guide

OOP

Wat is een object?

Een object is een exemplaar van een bepaalde klasse en bestaat uit de waarden van alle velden of attributen waarmee we een object definiëren. Objecten van dezelfde klassen beschikken steeds over dezelfde velden. De waarden van de velden kan per object verschillen.

De naam van een object start altijd met een kleine letter.

Wat is een klasse?

Een klasse beschrijft de algemene kenmerken en een aantal algemene methodes van objecten van een bepaalde soort. Men kan een instantie maken van een klasse welke men een object noemt.

De naam van een start altijd met een hoofdletter. Uit een klasse kunnen we oneindig veel objecten maken.

Een klasse bestaat uit:

- velden (instance variables, variabelen van alle types kunnen hierbij)
- properties (voor bewerkingen uit te voeren op de velden) Wel een verschil met auto implemented properties (referentie).
- constructors (startwaarden voor de velden, initialisatie)
    - Een constructor is een methode waarvan de naam dezelfde is als de naam van de klasse.
    - De signatuur van de methode bevat enkel de methodenaam en de lijst van parameters, er is geen returntype aanwezig.
    - Een constructor heeft altijd de naam van de klasse.
    - Je kan meerdere constructors maken voor je klasse.
    - Een defaultconstructor is een constructor zonder parameters.
    - Telkens je een instantie maakt van een klasse zal de constructor uitgevoerd worden.
    - Het hangt ervan af hoeveel parameters je meegeeft om uit te maken welke constructor uitgevoerd zal worden.

```
1   public Person()
2   {
3   }
```

Referentie

- methodes

```
1   // Met onderstaande code maak je een klasse,
```

```
2  // voorbeeld: Person.
3  public class Person
4  {
5      // Fields
6      // Defaultconstructor
7      // Hieronder een voorbeeld van een defaultconstructor
8      public Person()
9      {
10     }
11     // Properties
12     // Methods,...
13 }
```

Abstractie

Overerving

Een klasse kan eigenschappen overnemen vanuit een andere klasse dit noemen we overerving. Men spreekt dan van een basisklasse en een superklasse.

De basisklasse/superklasse is de hoogste klasse en de subklasse of afgeleide klasse bevat dan alle attributen en methodes van de superklasse.

Referentie

Polymorfisme

Een object kan voorkomen in meerdere gedaantes. Het komt erop neer dat een object van een subklasse kan gezien worden als een element van een superklasse en dus ook de eigenschappen en methoden kan gebruiken van die superklasse of basisklasse. Polymorfisme betekent letterlijk veelvormigheid.

Referentie

Inkapsulatie of inkapseling

Bij inkapseling wil men ervoor zorgen dat men een of meer items gaat insluiten in een fysieke of logische package. Hierbij ga je er voor zorgen dat er geen toegang is tot de details van de implementatie.

Encapsulation of inkapseling in OO ga je toepassen door gebruik te maken van access specifiers. Een access specifier gaat definiëren wat de scope en visibiliteit is van een klasse member.

Volgende access specifiers zijn van toepassing:

- public, overal
- private, enkel in dezelfde klasse.

- protected, in dezelfde klasse of elke klasse die overerft van de klasse.
- internal, in dezelfde assembly, maar niet van een andere assembly.
- protected internal, in de assembly waar die gedeclareerd is of van een overervende klasse in een andere assembly.

Wat is een struct?

Een struct is qua syntax hetzelfde als een klasse maar structs zijn beperkter dan klasses. Even een overzicht van de verschillen:

- Structs zijn value types en klasses zijn reference types.
- Binnen een struct declaratie kunnen de velden of fields niet geintialiseerd worden tenzij deze gedeclareerd zijn als const of als static.
- Binnen een struct kan je geen default constructor declareren of een finalizer.
- Structs kunnen constructors bevatten met parameters.
- Als je een struct toewijst aan een nieuwe variabele dan zal alle data gekopieerd worden. Bij wijziging van de nieuwe kopie zal de data in de originele kopie niet veranderen. Dit belangrijk om weten wanneer je gebruik gaat maken van collections van value types zoals Dictionary<string, myStruct>.
- Je dient geen new te gebruiken zoals bij klassen wanneer je een instantie maakt van een struct.
- Je kan niet overerven van een andere struct of klasse, en het kan ook niet de basisklasse zijn van een klasse. Elke struct erft over van System.ValueType welke overerft van System.Object.
- Een struct kan interfaces implementeren.
- Een struct kan gebruikt worden als een nullable type and kan null waarde toegewezen worden.

```
1 // Met onderstaande code maak je een struct,
2 // voorbeeld: Postcode.
3 public struct Postcode
4 {
5     // Fields
6     // Properties
7     // Methods,...
8 }
```

Referentie

Statische klasse

Een statische klasse is een klasse die niet kan geïnstantieerd worden. Je kan er geen objecten van creëren.
Het keyword **new** kunnen we dus niet gebruiken.
Vooral voor methoden met een generieke functionaliteit, ook wel helpers genoemd.
Een voorbeeld is de Math-klasse, maar ook Console.

Abstracte klasse

Hetzelfde dan een klasse maar keyword abstract en geeft weer dat de implementatie niet compleet is.
Abstraction is het proces om details te verbergen en enkel de functionaliteit te tonen.

Statische klassen

Een statische klasse is een klasse die niet kan geïnstantieerd worden. Je kan er geen objecten van creëren.
Het keyword **new** kunnen we dus niet gebruiken.
Vooral voor methoden met een generieke functionaliteit, ook wel helpers genoemd.
Een voorbeeld is de Math-klasse, maar ook Console.

Abstracte klassen

Een abstracte klasse laat het toe om klassen en klasse members aan te maken die niet volledig zijn en die moeten geïmplementeerd worden in afgeleide (derived) klassen.

Want een abstracte klasse kan een volledige implementatie bevatten, maar meestal krijgt het een gedeeltelijke implementatie, of in het geheel geen implementatie. Gemeenschappelijke functionaliteit wordt zo ge-encapsuleerd voor de ervende klassen.

Je kan geen instantie maken een abstracte klasse alleen klassen die ervan erven kunnen dit.

Wanneer je een abstracte klasse hebt met methoden zonder implementatie (abstracte methoden), dan wordt ook de klasse abstract genoemd. Een abstracte methode is dan puur een aanduiding voor een methode die in een ervende klasse geïmplementeerd zal worden. Abstracte methoden hebben tot doel om iedere klasse die erft van de abstracte klasse een implementatie af te dwingen. Als de ervende klasse dat niet doet, wordt ook deze klasse weer abstract genoemd.

Je declareert een abstracte class met het abstract keyword.

Het keyword sealed voorkomt de overerving van een klasse of bepaalde klasse members die voorgaand gemarkeerd zijn als virtual.

Referentie: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/abstract-and-sealed-classes-and-class-members

Interfaces

Een interface definieert een set van methoden, properties, events en indexers of een combinatie van deze vier, net zoals bij klassieke klassen.

Het grote verschil met een klasse is dat een interface geen implementatie bevat, waardoor je een interface kan zien als een contract.

Een klasse of struct die een interface implementeert moet elk aspect van de interface implementeren zoals vastgelegd in de interface.

Voor een interface aan te maken (te declareren) dien je het keyword interface te gebruiken. De naam van de interface dient te starten met een hoofdletter I.

De interface in het voorbeeld hieronder voorziet in een definitie van een functie accountActive.

```
interface IAccount
{
   bool accountActive(Account a);
}
```

Een klasse of struct kan meerdere interfaces implementeren, maar een klasse kan maar van een klasse overerven.

Een interface kan geen constanten bevatten, velden, operators, (instantie) constructors, finalizers of types. Members van een interface zijn automatisch publiek (public) en kunnen geen enkele access modifiers invoegen. Members kunnen ook niet static zijn.

To implement an interface member, the corresponding member of the implementing class must be public, non-static, and have the same name and signature as the interface member.

Wat is het verschil tussen een interface en een abstract klasse? Wat is het voordeel van een interface bij overerving?

Referenties:

- https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/interfaces/
- https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/interface

Interfaces versus abstract klasse

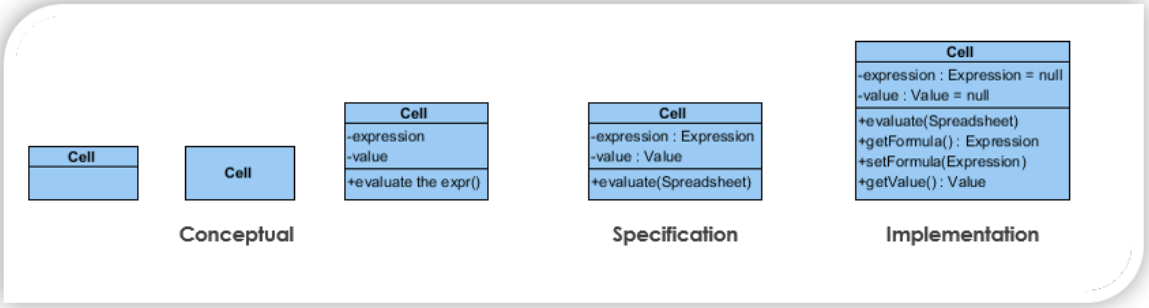
Interface	**Abstracte klasse**
Kan alleen abstracte methoden bevatten (impliciet); gebruik van abstract keyword mag niet.	Kan abstracte en niet abstracte methoden bevatten.

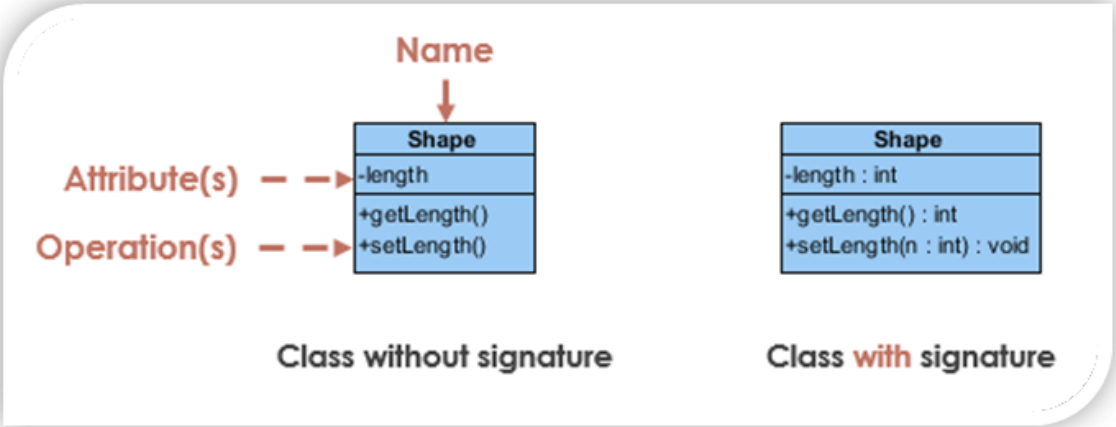
Niet bedoeld voor fields.	Fields zijn mogelijk.
Instantie via implementatie.	Instantie via subklasse.
Geen uitbreiding op klasse, kan wel een enkele interface of meerdere uitbreiden	Kan een enkele klasse uitbreiden en een of meerdere interfaces implementeren.
Geen onderdeel van klasse hiërarchie, zelfde interface kan je gebruiken in klassen die niets met elkaar te maken hebben	Abstracte klassen zijn wel onderdeel van de klasse hiërarchie.
Kennen geen constructor.	Kunnen een constructor bevatten.
Interface	**Abstracte klasse**
Multiple inheritance, meerdere interfaces implementeren mogelijk.	Een klasse kan slechts ervan van een andere klasse.
Alleen publieke methoden en constanten zijn mogelijk, zonder implementatie.	Statische methoden, protected members en gedeeltelijke implementatie zijn allemaal mogelijk.

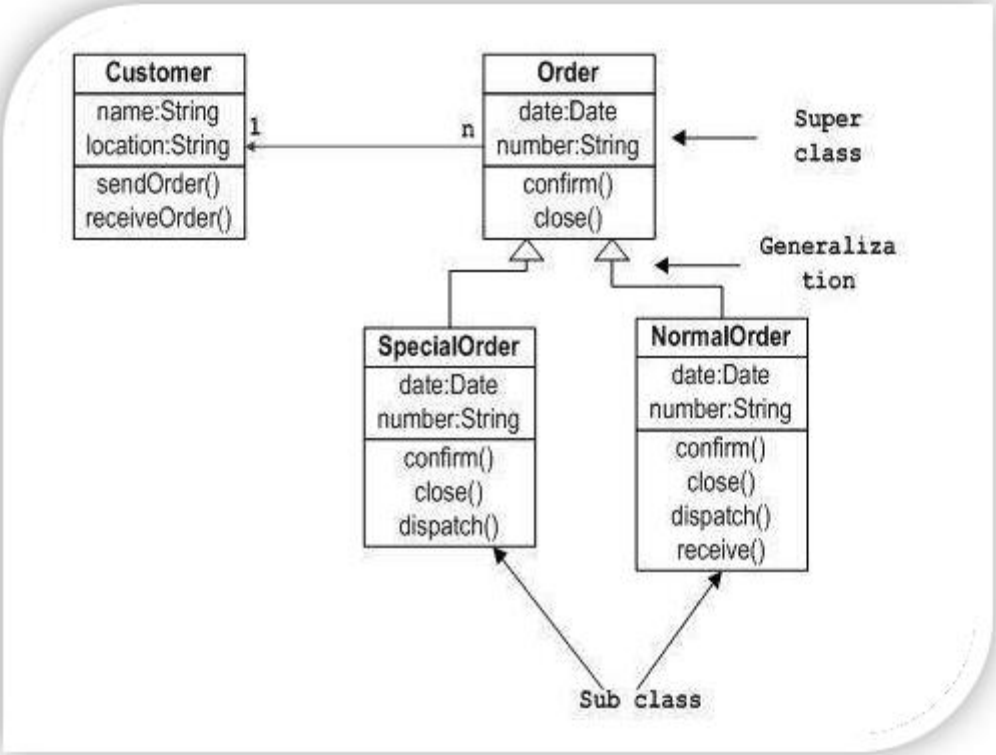
UML

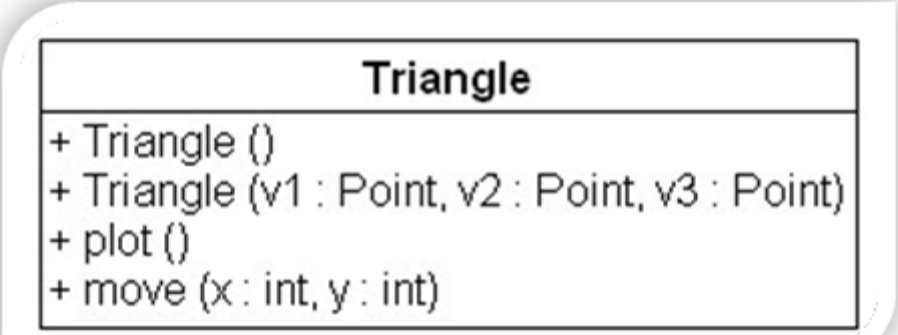
UML staat voor Unified Modeling Language.
Standaard sinds 1997 voor objectgeoriënteerde analyse.
UML-model = grafische weergave om relatie tussen verschillende klassen weer te geven.
Een klasse wordt voorgesteld als een blok met bovenaan de naam gevolgd door eventuele velden, properties en methoden.
Je ontwerpt eigenlijk eerst de klassestructuur alvorens te beginnen met coderen. Je zorgt zo voor structuur.

\+ public attributen of operaties
\- private attributen of operaties
\# protected attributen of operaties










```
+=============================+
| ClassName                   |
+-----------------------------+
| +<<get>> Id : int           |
| -<<set>> Id : int           |
| +<<get>> IsSomething : bool |
+-----------------------------+
| + Method1(arg1 : string)    |
+=============================+
```

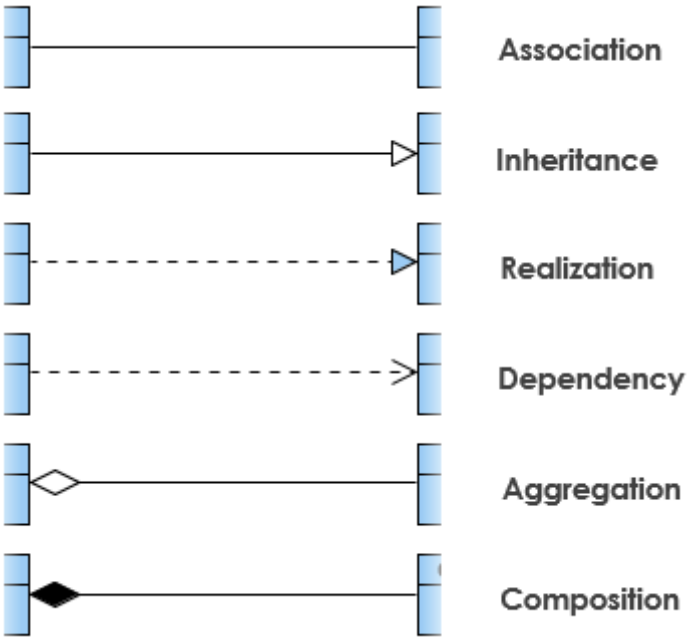
Perspectieven

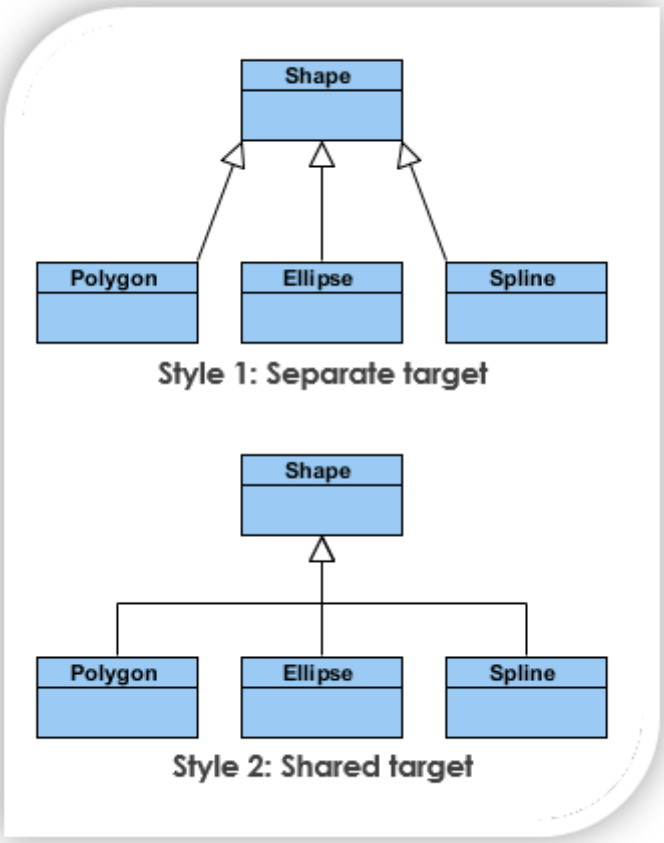
- **Conceptual**: het presenteren van de concepten
- **Specification**: focus is hierbij op de inface van de abstract data types
- **Implementation**: beschrijft hoe klassen de interfaces gaat implementeren

In sommige gevallen gaat men ook opteren om een prefix te gebruiken:

- «property»
- «constructor»

Relatie





Referentie: Links to an external site.https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/

Gebruik voor het modelleren van een UML-schema een modelleertool:

- UMLet https://www.umlet.com/
  - Online: http://www.umletino.com/umletino.html
- StarUML.io http://staruml.io/

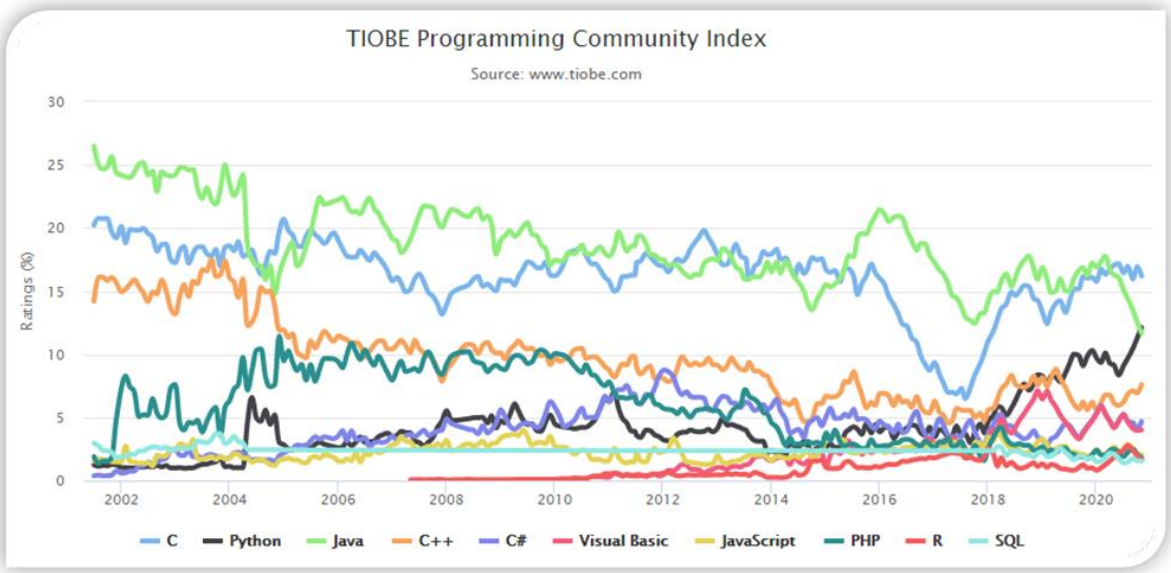
Porgrammeertalen

Overzicht van programmeertalen.

Er zijn verschillende mogelijkheden om een blik te werpen op welke programmeertalen er momenteel een belangrijke rol nemen in de uitwerking van projecten.

TIOBE's Programming Community Index.

De eerste bron van informatie is de TIOBE's Programming Community Index.
TIOBE staat voor **T**he **I**mportance **O**f **B**eing **E**arnest.



Nov 2020	Nov 2019	Change	Programming Language	Ratings	Change
1	2	^	C	16.21%	+0.17%
2	3	^	Python	12.12%	+2.27%
3	1	v	Java	11.68%	-4.57%
4	4		C++	7.60%	+1.99%
5	5		C#	4.67%	+0.36%
6	6		Visual Basic	4.01%	-0.22%
7	7		JavaScript	2.03%	+0.10%
8	8		PHP	1.79%	+0.07%
9	16	^^	R	1.64%	+0.66%
10	9	v	SQL	1.54%	-0.15%
11	14	^	Groovy	1.51%	+0.41%
12	21	^^	Perl	1.51%	+0.68%
13	20	^^	Go	1.36%	+0.51%
14	10	vv	Swift	1.35%	-0.31%
15	11	vv	Ruby	1.22%	-0.04%
16	15	v	Assembly language	1.17%	+0.14%
17	19	^	MATLAB	1.10%	+0.21%
18	13	vv	Delphi/Object Pascal	0.86%	-0.28%
19	12	vv	Objective-C	0.84%	-0.35%
20	32	^^	Transact-SQL	0.82%	+0.44%

Bron: https://www.tiobe.com/tiobe-index/

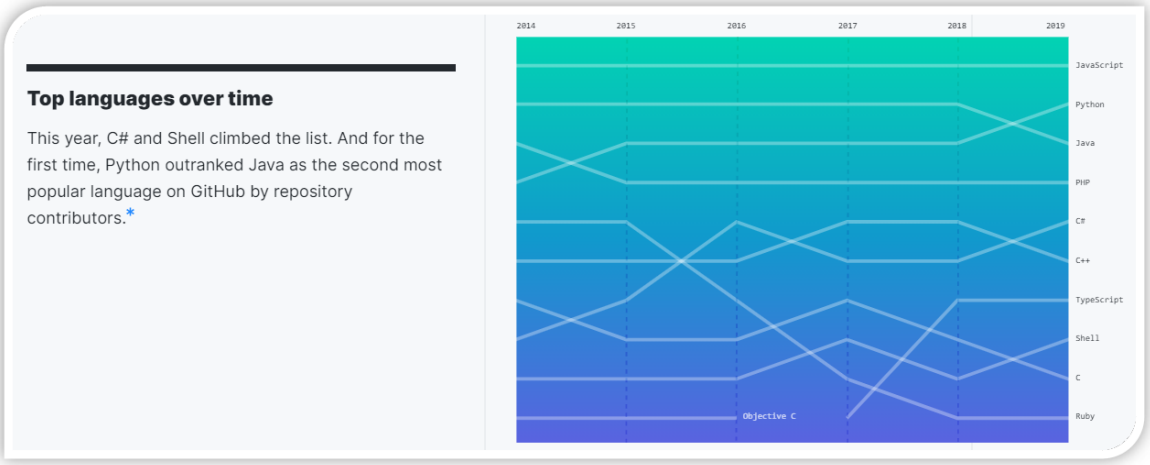
Als je de grafiek bekijkt zijn de belangrijkste programmeertalen de volgende:

- Java
- C
- Python
- C++
- C#
- Visual Basic .NET
- Javascript
- PHP
- SQL

TIOBE 2018 versus TIOBE 2019.

GitHub Octoverse.

Een andere bron is de programmeertalen die gebruikt worden voor projecten op GitHub.



Hieruit blijkt dat de meest populaire programmeertalen de volgende zijn:

- Javascript
- Java
- Python
- PHP
- Ruby
- C++
- C#
- TypeScript

Bron: https://octoverse.github.com/#top-languages

PYPL index.

PYPL staat voor PopularitY of Programming Language.

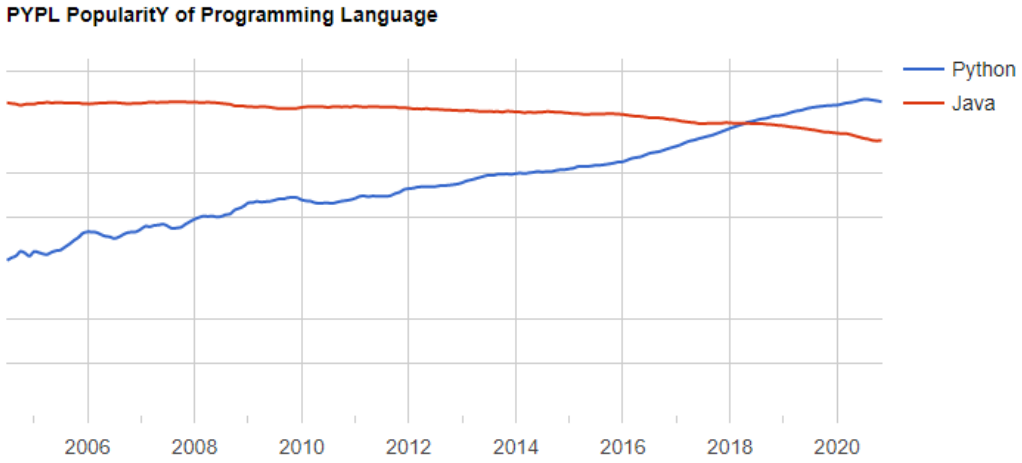
1		Python	30.8 %	+1.8 %
2		Java	16.79 %	-2.3 %
3		JavaScript	8.37 %	+0.3 %
4		C#	6.42 %	-0.9 %
5		PHP	5.92 %	-0.2 %
6		C/C++	5.78 %	-0.2 %
7		R	4.16 %	+0.4 %
8		Objective-C	3.57 %	+1.0 %
9		Swift	2.29 %	-0.2 %
10		TypeScript	1.84 %	-0.0 %
11		Matlab	1.65 %	-0.1 %
12		Kotlin	1.64 %	-0.0 %
13	↑↑	Go	1.43 %	+0.2 %
14	↓	Ruby	1.2 %	-0.2 %
15	↓	VBA	1.11 %	-0.2 %

Bron: https://pypl.github.io/PYPL.html

Als je naar deze index kijkt dan zie je dat volgende programmeertalen de populairste zijn:

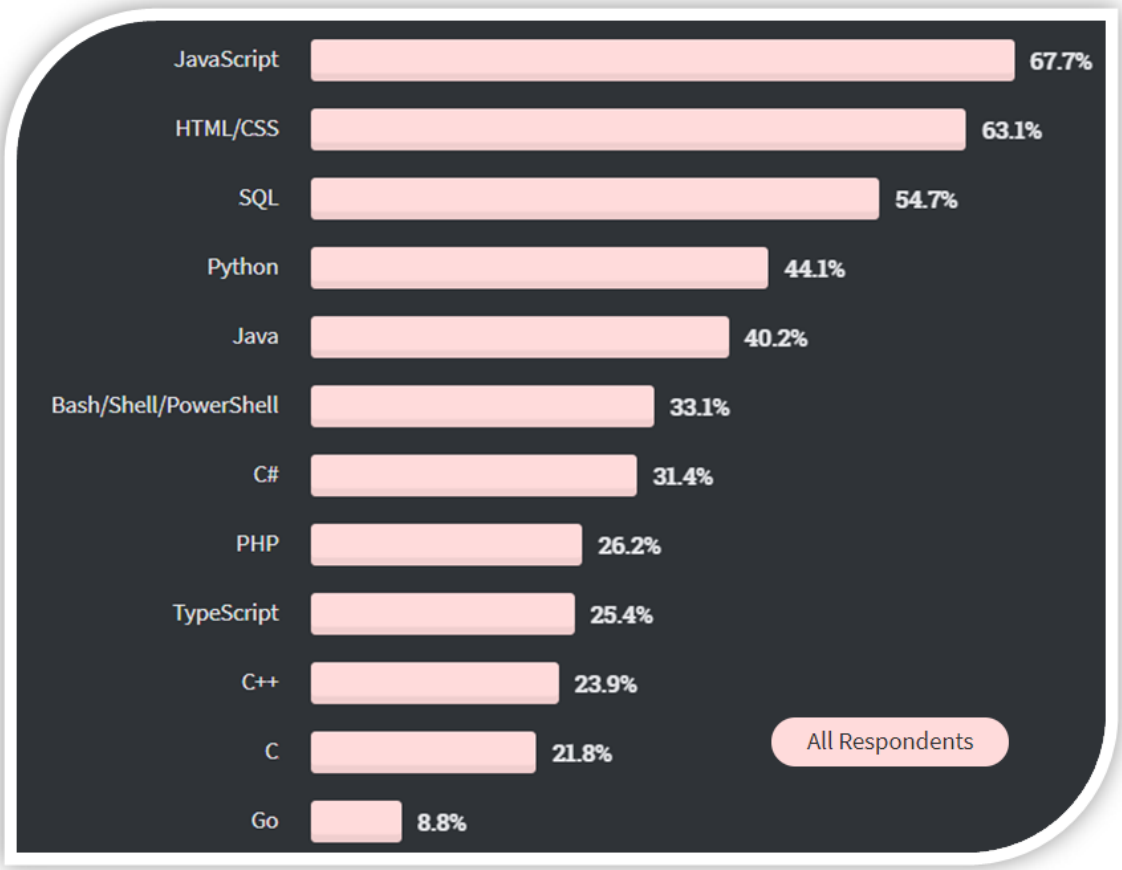
- Python
- Java
- Javascript
- C#
- PHP
- C/C++

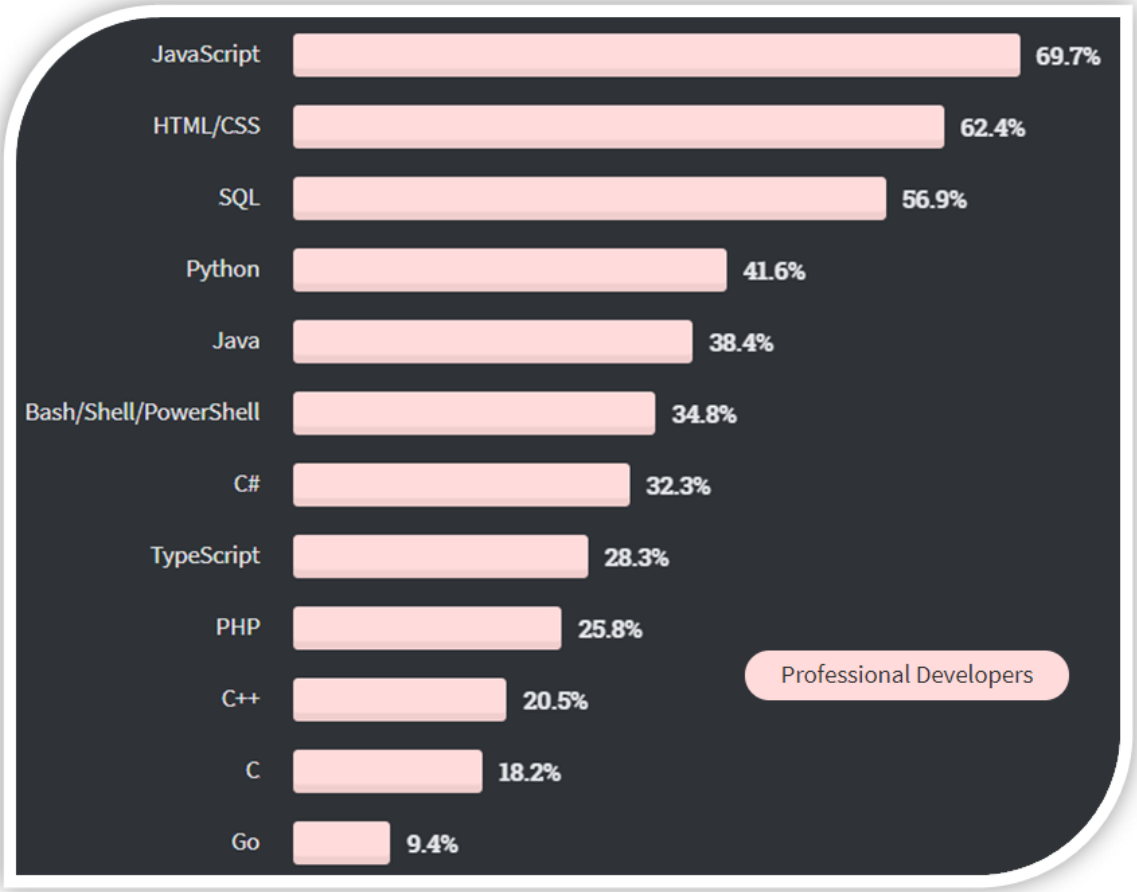
Worldwide, Python is the most popular language, Python grew the most in the last 5 years (19.1%) and Java lost the most (-8.7%)

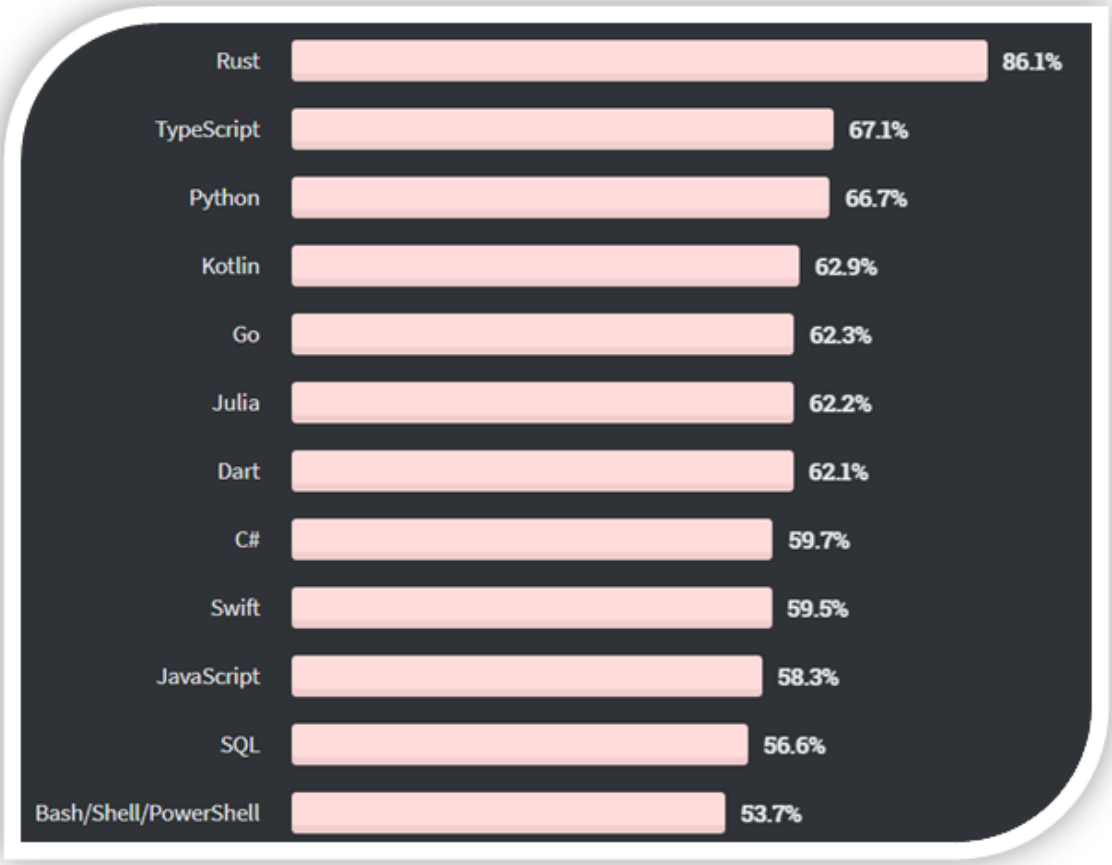


PYPL 2018 versus PYPL 2019.

Stackoverflow Survey




JavaScript
69.7%
HTML/CSS
62.4%
SQL
56.9%
Python
41.6%
Java
38.4%
Bash/Shell/PowerShell
34.8%
C#
32.3%
TypeScript
28.3%
PHP
25.8%
C++
20.5%
C
18.2%
Go
9.4%
Professional Developers



Bron: https://insights.stackoverflow.com/survey/2020#technology

Conclusie

De meeste programmeertalen die telkens terugkeren voor de specifieke doeleinden:

- Front-end web development: Javascript,…
- Back-end web development: Javascript, Java, Python, PHP, Ruby…
- Mobile development: Swift, Java, C#,…
- Game development: C++, C#,…
- Desktop applications: Java, C++, Python, C#,…
- Systems programming: C, Rust,…

Cloud computing

Cloudoplossingen

Op IT vlak kan je kiezen voor verschillende cloudoplossingen.

De 3 meest voorkomende vormen zijn: IaaS, PaaS en SaaS. Er zijn ook varianten zoals BaaS,…

Je kan samenvatten dat deze allen eindigen op “aaS” welke staat “voor as a Service”. Je kan de verschillende oplossingen onderscheiden door hun niveau van service die ze aanbieden en hoeveel u of uw bedrijf zelf of via een partner dienen te voorzien om tot een oplossing te komen voor IT-noden.

De noden kunnen volgende zijn:

- Hosting voor een WordPress website.
- IDE in de cloud.
- Desktop in de cloud.

Infrastructure as a Service

Infrastructure as a Service of afgekort IaaS staat voor het niveau van de fysieke laag (men spreekt ook wel van "server storage network"). Deze bestaat uit volgende onderdelen: servers met opslagcapaciteit (datacenter) met volgende noodzakelijke basisvoorzieningen: airconditioning, elektriciteit, internet/netwerk,… Indien je zelf niet wil voorzien in infrastructuur voor je IT dan je opteren voor IaaS. Hierbij kan je kiezen om volledig of deels het beheer uit te voeren en ben je verantwoordelijk voor het applicatiebeheer en de applicaties die op die specifieke omgeving zal draaien.

Referentie:

- Best Infrastructure as a Service (IaaS) Providers: https://www.g2.com/categories/infrastructure-as-a-service-iaas

Platform as a Service

Platform as a Service of afgekort PaaS staat voor het niveau waarbij de leverancier of serviceprovider meer zal aanbieden dan enkel de infrastructuur. De serviceprovider zal voorzien in de IaaS-laag, het OS (Windows, Linux,…) eventueel ook middleware (webserver (Apache, IIS,…), frameworks (.NET,…), databaseserver (MySQL, MS SQL Server,…)) en eventueel back-up voorzien in het aangeboden servicemodel.

Referenties:

- 10 Top PaaS Providers for 2020: https://www.devteam.space/blog/10-top-paas-providers/
- Best PaaS providers of 2020: https://www.techradar.com/news/best-paas-provider
- Best Cloud Platform as a Service (PaaS)Software: https://www.g2.com/categories/cloud-platform-as-a-service-paas

Backend as a Service

Backend as a Service of afgekort BaaS staat voor het niveau waarbij een backend framework klaar staat om direct te gebruiken.

Referenties:

- Backend as a Serivce - What is a BaaS?: https://blog.back4app.com/2019/07/24/backend-as-a-service-baas/
- BaaS providers: https://blog.back4app.com/2019/09/30/baas-providers/

Software as a Service

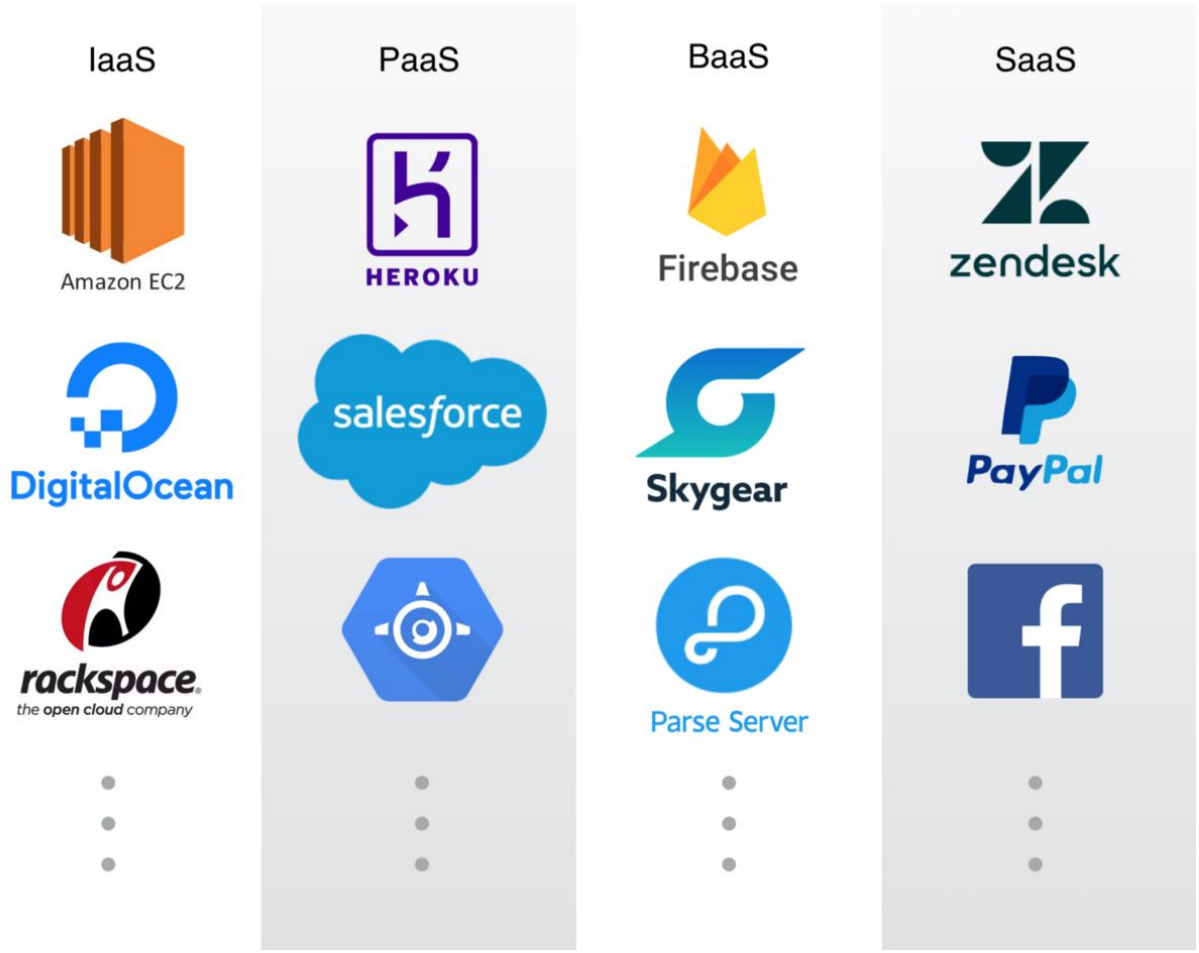
Software as a Service of afgekort SaaS staat voor het niveau waarbij een (afgewerkte) applicatie aangeboden wordt in de cloud. Dit is de meestgebruikte vorm van cloudoplossing. De eindgebruiker dient niet te voorzien in de technische kennis en kan direct van start met het gebruik van de applicatie voor het uitvoeren of helpt bij het uitvoeren van zijn dagdagelijkse taken.

Voorbeeld: Teamleader, Google Forms, Doodle,…

Referenties:

- The Top 30 SaaS Companies & Products to Watch in 2019: https://blog.hubspot.com/service/top-saas-companies
- 20 SaaS Companies to Watch out in 2020: https://www.aithority.com/ait-featured-posts/20-saas-companies-to-watch-out-in-2020/
- 50 Leading SaaS Companies: https://www.datamation.com/cloud-computing/50-leading-saas-companies.html

Praktische voorbeelden:



- Welke soorten cloudoplossingen zijn er?
- Waarvoor staat IaaS?
- Waarvoor staat PaaS?
- Waarvoor staat SaaS?
- Wat is een SLA?
- Welke omgeving is interessant als developer om snel een testomgeving, developmentomgeving en/of productieomgeving op te zetten?

Datacollecties

Array

Een array is een collectie welke bestaat uit een vast aantal elementen van hetzelfde datatype. Elk element in de collectie is een variabele en heeft een adres of ook wel index genoemd. Doormiddel van de index kan je een waarde toekennen of opvragen.

Snelle toegang via een nummer. Grootte staat vooraf vast, resizen is duur. Een tweedimensionale array (toegang via twee nummers) heet een matrix.

Aanpassen van de grootte is moeilijk.

Een dimensionale array

Toegang via single index in een lineaire vorm.

```
// Array met bedragen (double datatype), grootte van de array is 10.
double[] bedragen = new double[10];
// Array met getallen (int datatype), grootte van de array is 5.
int[] getallen = new int[5] { 1, 2, 3, 4, 5 };
// Opvragen van het 2de getal van de array getallen.
Console.WriteLine("Het 2de getal van de array getallen: {0}", getallen[1]);
// Waarde opgeven voor de 1ste en 2de plek in de array bedragen.
bedragen[0] = 2.50;
bedragen[1] = 4.12;
// Uitlezen van het 2de getal van de array bedragen.
Console.WriteLine("Het 2de getal van de array bedragen: {0}", bedragen[1]);
```

Multidimensionale array

Matrix van waarden van hetzelfde type men spreekt ook wel eens van arrays van arrays.

```
// 2 dimensionale array
string[,] 2darray = new string[3, 2] { { "BMW", "520" }, { "Golf", "1,9 TDI" },
                                       { "Mercedes-Benz", "E 200" } };
// Afdrukken type van de Golf.
Console.WriteLine(2darray[1, 1]);
// 3 dimensionale array
int[, ,] getallen = new int[2, 2, 3] { { { 1, 2, 3 }, { 4, 5, 6 } },
                                       { { 7, 8, 9 }, { 10, 11, 12 } } };
```

Referentie multidimensionale array

Algemene referentie: https://docs.microsoft.com/nl-nl/dotnet/api/system.array?view=netframework-4.7

Sorteren Array

Het sorteren van een Array kan via de standaard daarvoor voorziene methode Sort.

Array.Sort();

https://docs.microsoft.com/nl-nl/dotnet/api/system.array.sort?view=netframework-4.7

BitArray

Een BitArray is een array bestaande uit bit waarden welke bestaan uit booleans.

Referentie: https://docs.microsoft.com/nl-nl/dotnet/api/system.collections.bitarray?view=netframework-4.7

```
1 bool[] booleans = new bool[5] { true, false, true, true, false };
2 BitArray ba = new BitArray( booleans );
3 Console.WriteLine("Aantal: {0}", booleans.Count );
4 Console.WriteLine("Lengte: {0}", booleans.Length );
5
```

ArrayList

Is een datacollectie van een bepaalde object die geordend is. Opvragen van een item kan via zijn index. Het verschil met een array is dat je items kan toevoegen en verwijderen zonder de grootte aan te passen. Sorteren, zoeken zijn ook mogelijk via methodes.

Referentie: https://docs.microsoft.com/nl-nl/dotnet/api/system.collections.arraylist?view=netframework-4.7

```
1  // Aanmaak arraylist met toevoeging van 3 elementen.
2  ArrayList al = new ArrayList();
3  al.Add("Hello");
4  al.Add("World");
5  al.Add("!");
6  // Opvragen van het 2de element.
7  Console.WriteLine("2de waarde:  {0}", al[1]);
8  // Het aantal, de capaciteit en de elementen opvragen.
9  Console.WriteLine("Aantal: {0}", al.Count );
10 Console.WriteLine("Capaciteit: {0}", al.Capacity );
```

Lijst

Een lijst, list in C#, is een collectie welke bestaat uit een x aantal elementen van een bepaald object. Elk element is toegankelijk via zijn index. Zoeken, sorteren en manipuleren is mogelijk via voorziene methodes.

Referentie: https://docs.microsoft.com/nl-nl/dotnet/api/system.collections.generic.list-1?view=netframework-4.7

```
1  // Aanmaak van een lijst met strings.
2  List<string> namen = new List<string>();
3  // Toevoegen van namen tot de lijst.
4  namen.Add("Bart");
5  namen.Add("Geert");
6  namen.Add("Kristof");
7  Console.WriteLine();
8  foreach (string naam in namen)
9  {
10     Console.WriteLine(naam);
11 }
12 Console.WriteLine("");
13 Console.WriteLine("Capaciteit: {0}", namen.Capacity);
14 Console.WriteLine("Aantal: {0}", namen.Count);
15 // Een element verwijderen.
16 namen.Remove("Bart");
17 // Ledigen van lijst.
18 namen.Clear();
```

Sortedlist

Dit is een datacollectie waarbij je gebruik kan maken van de key of de index om elementen op te vragen in de lijst. Een SortedList is een combinatie van een hashtable (key) en een arraylist (index). De collectie is gesorteerd via zijn key waarde.

Referentie: https://docs.microsoft.com/nl-nl/dotnet/api/system.collections.sortedlist?view=netframework-4.7

```
1  SortedList sl = new SortedList();
2  sl.Add("B", "World");
3  sl.Add("C", "!");
4  sl.Add("A", "Hello");
5  Console.WriteLine("Count:    {0}", mySL.Count );
6  Console.WriteLine("Capacity: {0}", mySL.Capacity );
7  Console.WriteLine("Key van 2de element: {0}", myList.GetKey(1));
8  Console.WriteLine("Value van 2de element: {0}", myList.GetByIndex(1));
```

Hashtable

Hashtable is een datacollectie die bestaat uit elementen van een bepaald datatype welke toegankelijk zijn via hun key. Dit gebruik je best als je elementen wenst op te vragen aan de hand van een key. Elk element in de hashtable bestaat uit een key/value paar.

Referentie: https://docs.microsoft.com/nl-nl/dotnet/api/system.collections.hashtable?view=netframework-4.7

```
1 Hashtable openenMet = new Hashtable();
2 openenMet.Add("rtf", "wordpad.exe");
3 openenMet.Add("jpg", "paint.exe");
4 openenMet.Add("txt", "notepad.exe");
5 openenMet.Add("pdf", "AcroRd32.exe");
6 openenMet["jpg"] = "Illustrator.exe";
7 Console.WriteLine("Opvragen waarde voor key jpg, waarde: {0}", openenMet["jpg"]);
8 Remove("pdf");
```

Queue

Een queue is een FIFO collectie van een bepaald object, waarbij FIFO staat voor first-in, first-out. Is handig als je een de items in de collectie wil raadplegen volgens het FIFO principe. Wanneer je een item toevoegt tot de queue dan dien je enqueue uit te voeren, wanneer je een item wist van de queue dan dien je dit te doen via deque.

Referentie: https://docs.microsoft.com/nl-nl/dotnet/api/system.collections.queue?view=netframework-4.7

```
1 Queue q = new Queue();
2 q.Enqueue("Hello");
3 q.Enqueue("World");
4 q.Enqueue("!");
5 Console.WriteLine("Aantal: {0}", q.Count);
6 Console.WriteLine("Een element uitlezen: " + q.Dequeue(););
7 q.Enqueue("!");
```

Stack

Een stack is een LIFO collectie van een bepaald object, waarbij LIFO staat voor last-in, first-out. Wanneer je een item toevoegt tot de stack dan zal push moeten uitvoeren, wanneer je een item van de lijst wist dan zal je pop moeten uitvoeren (popping).

Referentie: https://docs.microsoft.com/nl-nl/dotnet/api/system.collections.stack?view=netframework-4.7

```
1 Stack st = new Stack();
2 st.Push("Hello");
3 st.Push("World");
4 st.Push("!");
5 Console.WriteLine("Aantal: {0}", st.Count );
6 Console.WriteLine( "Weergeven element, (Pop): {0}", st.Pop() );
7 Console.WriteLine( "Weergeven element, (Peek): {0}", st.Peek() );
```

Associatieve array

Dictionary in C#; elementen in de datacollectie kan je zoeken via hun unieke key.

Referentie:

- Dictionary
- Dictionary Base

```
1 Dictionary<string, string> openenMet = new Dictionary<string, string>();
2 openenMet.Add("rtf", "wordpad.exe");
3 openenMet.Add("jpg", "paint.exe");
4 openenMet.Add("txt", "notepad.exe");
5 openenMet.Add("pdf", "AcroRd32.exe");
6 openenMet["jpg"] = "Illustrator.exe";
7 Console.WriteLine("Opvragen waarde voor key jpg, waarde: {0}", openenMet["jpg"]);
```

HashSet

Je kan elementen in de hashset opzoeken doormiddel van hun unieke key.

Referentie:

- HashSet

```
1  HashSet<int> hs = new HashSet<int>();
2  hs.Add(1);
3  hs.Add(3);
4  hs.Add(5);
5  hs.Add(7);
6  Console.WriteLine("Aantal: {0}", hs.Count);
7  Console.Write("Elementen: ");
8  Console.Write("{");
9  foreach (int i in hs)
10 {
11     Console.Write(" {0}", i);
12 }
13 Console.WriteLine(" }");
```

ConcurrentBag

Een concurrentbag is een datacollectie met waarden waar je resultaten kunt toevoegen en uitpikken.

Referentie: https://docs.microsoft.com/nl-nl/dotnet/api/system.collections.concurrent.concurrentbag-1?view=netframework-4.7

```
1  ConcurrentBag<int> cb = new ConcurrentBag<int>();
2  cb.Add(1);
3  cb.Add(3);
4  cb.Add(5);
5  cb.Add(7);
6  int element;
7  while (!cb.IsEmpty)
8  {
9      if (cb.TryTake(out element))
10         Console.WriteLine(element);
11 }
```

Databanken

Wat is een databank?

Een databank is een digitale verzameling van data welke je kan vergelijken met een bestand waarin gegevens opgeslagen en met elkaar verbonden worden.

Binnen een hogeschool kan men gebruik maken van een database die alle data bijhoudt die belangrijk zijn voor de onderwijsactiviteiten van de hogeschool. Denk daarbij aan de gegevens van de studenten, lectoren, lessen, lokalen, studiegelden,…

Voordelen

- Je kan eenvoudig gegevens in de databank toevoegen.
- Het is eenvoudig om gegevens te wijzigen en verwijderen in de databank.
- Je kan snel gegevens opzoeken in de databank.
- Je kan filteren op gegevens.

Onderdelen

Als je gegevens gaat bijhouden van specifieke objecten kan je dit gelijkstellen aan tabellen. Elke eigenschap van het object komt dan overeen met een kolom. En elk element van het type object zal dan overeenkomen met een rij in de tabel men noemt dit ook een record.

RDBMS

RDBMS staat voor relational database management system en wijst op het relationele model van de data. Meest gekende RDBMS is MySQL, Microsoft SQL Server, SQLite, Oracle,…

RDBMS

NoSQL

Een NoSQL databank verwijst naar een “niet-SQL” of “niet-relationele database”. Een NoSQL databank gaat voor het opslaan en ophalen van gegevens niet werken met tabellarische relaties zoals in een relationele databank

Voorbeelden van NoSQL databanken

Key-/Value-Stores

Bestaat uit een grote tabel van KV pairs.

Voorbeeld:

- Amazon's Dynamo

Document Databases

Deze databank slaat documenten op welke samengesteld zijn door tagged elementen.

Voorbeelden:

- MongoDB
- Apache CouchDB

Column-Oriented Databases

Elk blok bestaat uit gegevens afkomstig uit slechts één kolom.

Voorbeel:

- Google's Bigtable

Exploring the different types of NoSQL databases. NoSQL explained.

Big Data

Wat is big data? Big data is de term die men gebruikt om zeer grote, complexe, snel veranderende gegevensreeksen te beschrijven.

Hoewel big data wordt gedefinieerd op basis van grootte, wordt open data gedefinieerd door het gebruik ervan.

Belangrijk hierbij zijn de 3 V's of bij uitbreiding 5 V's:

- Volume (hoeveelheid): de hoeveelheid van data...
- Velocity (snelheid): de snelheid waarmee de data wijzigt...
- Variety (variëteit): de variatie aan data... (datums, cijfers, tekst,...) we hebben het dan vooral over gestructureerde (heeft een bepaalde lengte en formaat), niet gestructureerde of semi gestructureerde data.
- Veracity (waarheidsgetrouwheid): de data afkomstig van eigen bronnen of van andere partijen... Welke data is te vertrouwen in zekere zin.
- Value (waarde creatie): is de verzamelde data nuttig, ben je er iets mee... daarvoor dient de data omgevormd te worden tot informatie om zo tot bepaalde inzichten te komen en dan is het resultaat inderdaad waardevol.

Open Data

Wat is open data?

Open data verwijst naar alle informatie die beschikbaar is gemaakt voor iedereen om toegang te krijgen, te wijzigen en te delen. Het kan van een openbare bron of van een bedrijf zijn en kan worden gebruikt voor zowel commerciële als niet-commerciële doeleinden.

Open data kan je omschrijven als toegankelijke openbare gegevens die mensen, bedrijven en organisaties kunnen gebruiken om patronen en trends te analyseren, gegevensgestuurde beslissingen te nemen, nieuwe ondernemingen te lanceren en complexe problemen op te lossen.

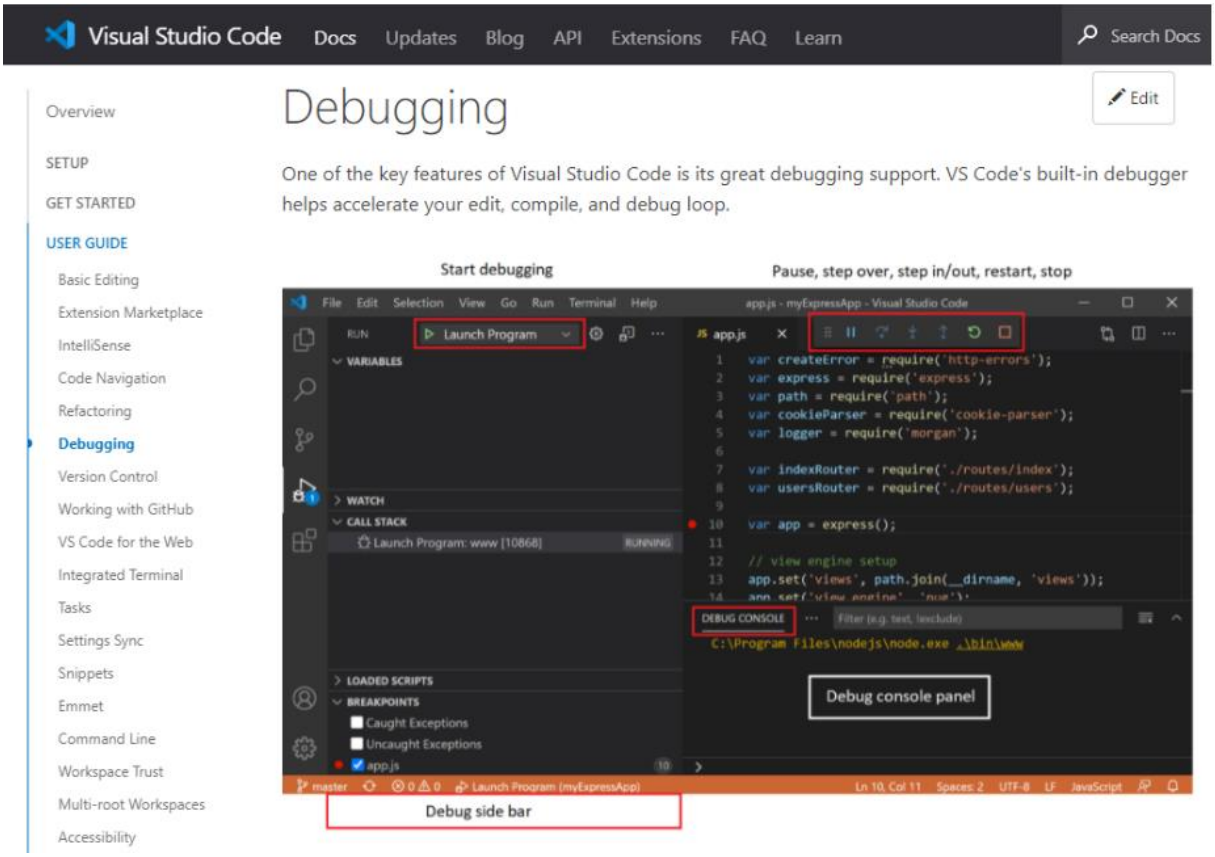
Alle definities van open data omvatten de twee basisfuncties:

- De gegevens moeten openbaar beschikbaar zijn voor iedereen om te gebruiken en moeten op een zodanige manier worden gelicentieerd dat hergebruik mogelijk is
- Open data moeten ook relatief gemakkelijk te gebruiken zijn, hoewel er niveaus van “openheid” zijn. En er is algemene overeenstemming dat open data gratis of tegen minimale kosten beschikbaar moeten zijn.
  Dus vrij van auteursrechten.
- Beschikbaar zonder beperkingen.
- Computer-leesbaar, gebruikmakend van open standaarden.

Debuggen

Wanneer je code schrijft is het belangrijk te weten wat die code precies uitvoert. Daarbij is het altijd handig om dan ook te **debuggen** via jouw **IDE** (ook wel **integrated development environment** genoemd).

Hieronder kan je het stappenplanvinden hoe je aan Debugging kan doen via Visual Studio Code.



Variabelen binnen Visual Studio Code: https://code.visualstudio.com/docs/editor/variables-reference

Debugging: https://code.visualstudio.com/docs/editor/debugging

.NET Core Debugger: https://github.com/OmniSharp/omnisharp-vscode/blob/master/debugger.md

Patronen

Wat zijn patronen?

Kan je vergelijken met blauwdrukken dat je kan aanpassen om veelvoorkomende ontwerpproblemen op te lossen in je code.

Ontwerppatronen of design patterns bieden een flexibele oplossing voor veelvoorkomende designproblemen.

Deze leveren een algemene oplossing of flexibele manier voor het oplossen van veel voorkomende problemen bij de ontwikkeling van software.

Bij OOP => hergebruik van oplossingen voor een probleem.

Overzicht

- Design patterns
  - Creational design patterns
    - *Focus op de manier hoe objecten gecreëerd worden.*
    - *Verhogen van flexibiliteit en hergebruik van code.*
    - *Patronen:*
      - Factorymethod ☆ ☆ ☆
      - Abstract factory ☆ ☆ ☆
      - Builder ☆ ☆ ☆
      - Prototype ☆ ☆
      - Singleton ☆ ☆
  - Structural design patterns
    - *Focus op het construeren van objecten en klassen in een grotere structuur (geheel).*
    - *Het aspect van efficiëntie en flexibiliteit mogen hierbij niet verloren gaan.*
    - *Patronen:*
      - Adaptive ☆ ☆ ☆
      - Bridge ☆
      - Composite ☆ ☆
      - Decorator ☆ ☆
      - Façade ☆ ☆
      - Flyweight ☆
      - Proxy ☆
  - Behavioral design patterns
    - *Focus op toewijzen van verantwoordelijkheden tussen objecten en algoritmes.*
    - *Patronen:*
      - Chain of responsibility ☆
      - Command ☆ ☆ ☆
      - Interpreter ☆

- Iterator ☆☆☆
- Mediator ☆☆
- Memento ☆
- Observer ☆☆☆
- State ☆☆
- Strategy ☆☆☆
- Visitor ☆
- Template method ☆☆☆

Cheat sheet

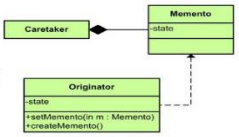
Hierbij een overzicht van alle ontwikkelpatronen...

C Abstract Factory
S Adapter
S Bridge
C Builder
B Chain of Responsibility
B Command
S Composite
S Decorator
S Facade
C Factory Method
S Flyweight
B Interpreter
B Iterator
B Mediator
B Memento
C Prototype
S Proxy
B Observer
C Singleton
B State
B Strategy
B Template Method
B Visitor

Memento

Type: Behavioral

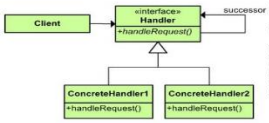
What it is:
Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.



Chain of Responsibility

Type: Behavioral

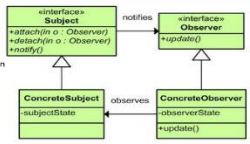
What it is:
Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.



Observer

Type: Behavioral

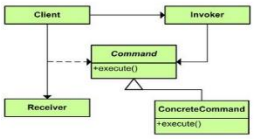
What it is:
Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.



Command

Type: Behavioral

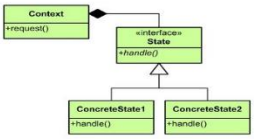
What it is:
Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.



State

Type: Behavioral

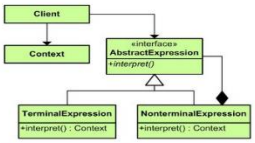
What it is:
Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.



Interpreter

Type: Behavioral

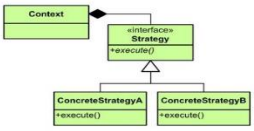
What it is:
Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.



Strategy

Type: Behavioral

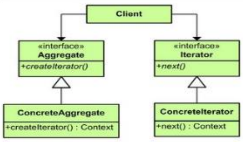
What it is:
Define a family of algorithms, encapsulate each one, and make them interchangeable. Lets the algorithm vary independently from clients that use it.



Iterator

Type: Behavioral

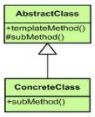
What it is:
Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.



Template Method

Type: Behavioral

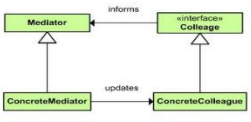
What it is:
Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.



Mediator

Type: Behavioral

What it is:
Define an object that encapsulates how a set of objects interact. Promotes loose coupling by keeping objects from referring to each other explicitly and it lets you vary their interactions independently.

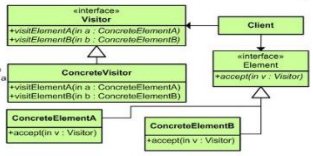


Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, Massachusetts: Addison Wesley Longman, Inc.

Visitor

Type: Behavioral

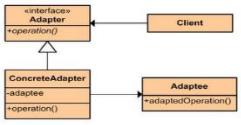
What it is:
Represent an operation to be performed on the elements of an object structure. Lets you define a new operation without changing the classes of the elements on which it operates.



Adapter

Type: Structural

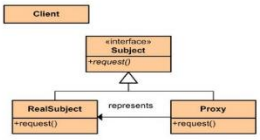
What it is:
Convert the interface of a class into another interface clients expect. Lets classes work together that couldn't otherwise because of incompatible interfaces.



Proxy

Type: Structural

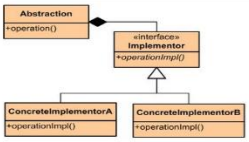
What it is:
Provide a surrogate or placeholder for another object to control access to it.



Bridge

Type: Structural

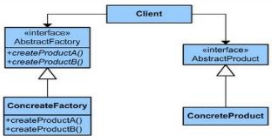
What it is:
Decouple an abstraction from its implementation so that the two can vary independently.



Abstract Factory

Type: Creational

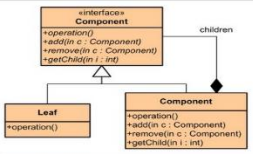
What it is:
Provides an interface for creating families of related or dependent objects without specifying their concrete class.



Composite

Type: Structural

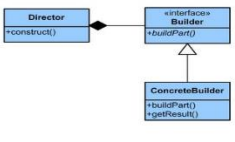
What it is:
Compose objects into tree structures to represent part-whole hierarchies. Lets clients treat individual objects and compositions of objects uniformly.



Builder

Type: Creational

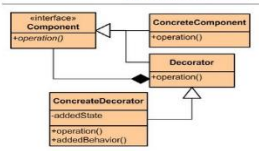
What it is:
Separate the construction of a complex object from its representing so that the same construction process can create different representations.



Decorator

Type: Structural

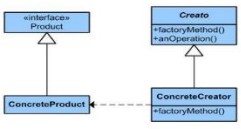
What it is:
Attach additional responsibilities to an object dynamically. Provide a flexible alternative to sub-classing for extending functionality.



Factory Method

Type: Creational

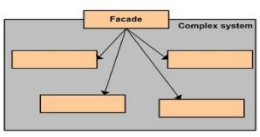
What it is:
Define an interface for creating an object, but let subclasses decide which class to instantiate. Lets a class defer instantiation to subclasses.



Facade

Type: Structural

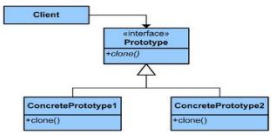
What it is:
Provide a unified interface to a set of interfaces in a subsystem. Defines a high-level interface that makes the subsystem easier to use.



Prototype

Type: Creational

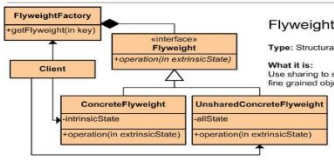
What it is:
Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.



Flyweight

Type: Structural

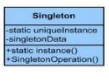
What it is:
Use sharing to support large numbers of fine grained objects efficiently.



Singleton

Type: Creational

What it is:
Ensure a class only has one instance and provide a global point of access to it.



Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, Massachusetts: Addison Wesley Longman, Inc.

To do

Lees aandachtig het naslagwerk op https://refactoring.guru/design-patterns

- Wat is een ontwerppatroon? > https://refactoring.guru/design-patterns/what-is-pattern
- De voordelen van ontwerppatronen > https://refactoring.guru/design-patterns/why-learn-patterns
- Onderverdeling van ontwerppatronen > https://refactoring.guru/design-patterns/classification
- Overzicht van de ontwerppatronen > https://refactoring.guru/design-patterns/catalog

Referenties:

- Design Patterns : http://refactoring.guru/design-patterns
- Overzicht van design patterns: http://refactoring.guru/design-patterns/catalog
- Overzicht van design patterns in C#: http://refactoring.guru/design-patterns/csharp

Snippets

Om snel code te kunnen schrijven kan je gebruikmaken van de stukjes code die je snel kan invoegen in de IDE Visual Studio Code, dit kan je doen door custom snippets aan te maken.

Stappenplan kan je vinden op:
https://code.visualstudio.com/docs/editor/userdefinedsnippets