Programmeren Bulletin

23ste jaargang oktober 2016 Nummer 3

hcc! programmeren

Interessegroep

Inhoud

Redactioneel

Voor het eerst verschijnt de Bulletin op internet over alles wat met programmeren in verschillende talen te maken heeft. Deze Bulletin was eerder een BASIC Bulletin, maar in vervolg zullen de onderwerpen niet meer alleen BASIC zijn. De Bulletin zal 4 keer per jaar op internet verschijnen. Deze Bulletin is nummer 3 van dit jaar.

Hebt u vragen en zoekt u naar antwoorden? Hebt u leuke tips of onderwerpen die u wilt delen met anderen? Neem contact met mij op. Misschien kan ik uw vragen beantwoorden en is het leuk om uw tips en onderwerpen in de Bulletin te plaatsen.

Veel plezier met de nieuwe Bulletin!

Ook al heeft XNA veel mogelijkheden om objecten te kunnen gebruiken voor het maken van spellen, toch mist XNA andere mogelijkheden die ook nodig zijn. Een goede controle om een botsing vanaf een bepaalde richting te kunnen testen is beperkt. XNA kan gewoon niet vertellen hoe een botsing optreedt en vanaf welke richting het kwam. Kwam het vanaf de linkerkant? Er worden verschillende technieken bedacht om deze beperkingen weg te kunnen poetsen. Helaas is dat moeilijker dan een programmeur zou willen dromen.

VBA kennen we als een interne bibliotheek in elk office programma. In deze Bulletin laat ik zien dat VBA ook extern te gebruiken is. Het is daarom een derde grote bibliotheek naast XNA en .NET. Maar... er zit echter een addertje onder het gras!

Wie Pascal of Python wil leren kennen, kan het begin van Pascal of de introductie van Python bekijken. In de volgende Bulletins blijf ik het over deze twee programmeertalen hebben.

Marco Kurvers

BBC BASIC for Windows – Programmabesturing.

Wanneer *BBC BASIC for Windows* een FOR, REPEAT, WHILE, GOSUB, FN of een PROC statement tegenkomt, moet hij kunnen weten waar hij in het programma zit, zodat het terug kan bij een NEXT, UNTIL, ENDWHILE of RETURN statement of wanneer het einde van een procedure of een functie wordt bereikt. Dit adres vertelt *BBC BASIC for Windows* waar het in uw programma is.

Elke keer wanneer *BBC BASIC for Windows* een FOR, REPEAT, WHILE, GOSUB, FN of PROC statement bereikt, wordt er een adres in een stack geplaatst en elke keer als het een NEXT, UNTIL, ENDWHILE of een RETURN statement of het einde van een functie of procedure bereikt, wordt het laatste adres 'gepakt' van de stack en gaat de besturing daar terug.

Afgezien van de grootte van het systeemgeheugen is er geen limiet op het niveau van geneste FOR ... NEXT, REPEAT... UNTIL, WHILE ... ENDWHILE en GOSUB ... RETURN operaties. De niet af te vangen foutbericht 'No room' zal worden afgegeven als alle ruimte in de stack is opgebruikt.

Programmastructuurbeperkingen

Het gebruik van een gemeenschappelijke stack heeft een nadeel (als het een nadeel is) in de zin dat het dwingt strikte naleving te volgen van een goede programmastructuur. Het is praktisch niet goed een FOR ... NEXT lus te verlaten zonder het NEXT statement te passeren; het maakt het programma moeilijker te begrijpen. Dit geldt ook bij het verlaten van een REPEAT ... UNTIL lus of een GOSUB ... RETURN structuur. Het nadeel wordt bedoeld dat, als u een FOR ... NEXT lus verlaat zonder het uitvoeren van het NEXT statement en dan bijvoorbeeld een ENDWHILE statement wordt bereikt, *BBC BASIC for Windows* een fout rapporteert (in dit geval een 'Not in a WHILE loop' fout). Het voorbeeld onderaan resulteert een foutmelding 'Not in a REPEAT loop at line 460'.

```
400 REPEAT
410    INPUT ' "Bij welk nummer moet het stoppen", num
420    FOR i=1 TO 100
430        PRINT i;
440        IF i=num THEN 460
450    NEXT i
460 UNTIL num=-1
```

Programmalussen verlaten

Er zijn verschillende manieren om een programmalus te verlaten die niet strijdig zijn met de noodzaak om netjes programmastructuren te schrijven. Deze worden hieronder uitgelegd:

REPEAT ... UNTIL lussen
Om het verlaat-probleem van een FOR ... NEXT lus te overwinnen is om het te herstructureren naar een REPEAT ... UNTIL lus. Het voorbeeld hieronder voert dezelfde functie uit als het vorige voorbeeld, maar het verlaat de structuur goed. Het heeft als extra voordeel meer duidelijkheid over de voorwaarden waaraan de lus moet worden beëindigd (en het vereist geen regelnummers):

```
REPEAT
    INPUT ' "Bij welk nummer moet het stoppen", num
    i=0
    REPEAT
        i=i+1
        PRINT i;
```

```
    UNTIL i=100 OR i=num
UNTIL num=-1
```

De lus variabele wijzigen
Een andere manier om voortijdig een lus te verlaten is door de lus variabele een waarde aan toe te kennen die gelijk moet zijn aan de grenswaarde. Voor een uitwijkmogelijkheid kunt u de lus variabele instellen op een hogere waarde dan de limiet (ervan uitgaande dat de stap positief is). In dit geval is de waarde bij afsluiten afhankelijk van waarom de lus werd beëindigd (in sommige gevallen zou dit een voordeel kunnen zijn). In het onderstaand voorbeeld wordt deze methode gebruikt om de lus te verlaten:

```
REPEAT
    INPUT ' "Bij welk nummer moet het stoppen", num
    FOR i=1 TO 100
        PRINT i;
        IF i=num THEN
            i=500
        ELSE
            REM Meer code hier indien nodig
        ENDIF
    NEXT
UNTIL num=-1
```

Het EXIT statement gebruiken
U kunt de EXIT FOR statement gebruiken voor hetzelfde effect als het eerste voorbeeld, zonder gebruik te moeten maken van GOTO:

```
REPEAT
    INPUT ' "Bij welk nummer moet het stoppen", num
    FOR i=1 TO 100
        PRINT i;
        IF i=num THEN EXIT FOR
        REN Meer code hier indien nodig
    NEXT i
UNTIL num=-1
```

Een lus uitvoeren in een procedure
Een radicaal andere benadering is het verplaatsen van de lus in een gebruiker gedefinieerde procedure of functie. Hiermee kunt u rechtstreeks uit de lus springen:

```
DEF PROCloop(num)
    LOCAL i
    FOR i=1 TO 100
        PRINT i;
        IF i=num THEN ENDPROC
        REM Meer code hier indien nodig
    NEXT i
ENDPROC
```

Als u wilt weten of de lus voortijdig was gestopt, kunt u een andere waarde retourneren:

```
DEF FNloop(num)
    LOCAL i
    FOR i=1 TO 100
        PRINT i;
        IF i=num THEN =TRUE
```

```
        REM Meer code hier indien nodig
    NEXT i
=FALSE
```

Omdat lokale variabelen ook opgeslagen zijn op de stack, kunt u geen gebruik maken van een FOR ... NEXT lus die een lokale array maakt. Als voorbeeld, het volgende programma zal de foutmelding 'Not in a FN or PROC' geven:

```
DEF PROC_error_demo
    FOR i=0 TO 10
        LOCAL data(i)
    NEXT
ENDPROC
```

Gelukkig mag in *BBC BASIC for Windows* wel een array 'local' worden gemaakt op de volgende manier:

```
DEF PROC_error_demo
    LOCAL data()
    DIM data(10)
ENDPROC
```

Let op het gebruik van de haakjes openen en sluiten met niets ertussen.

XNA Game Studio 4.0 – De Line Intersection techniek.

XNA is een enorme bibliotheek vol met klassen die de besturing van de hardware op zich nemen. Wij hoeven alleen onze spullen erbij te maken en Klaar is Kees.

Maar Kees lijkt nog een eind ver weg te zijn. Zo we nu ondervonden hebben in het voorbeeld VBGame, is het praktisch nog niet zo gemakkelijk om een spel in elkaar te zetten. Zelfs ik ben nog niet zo ver om goed met XNA overweg te kunnen, maar om te weten hoe de klassen werken is al een pluspunt.

Wel, het terugkaatsen van de bal tegen de paddle was al een vooruitgang, maar waarom de bal zomaar in de paddle terechtkomt, is de vraag. Een vraag waar helaas niet een goed antwoord op te vinden is. Zoeken naar een goed antwoord is zoeken naar een speld in een hooiberg. Tientallen programmeurs proberen de juiste Intersection te vinden om de bal correct te laten botsen. Daar komen verschillende codefragmenten uit waarvan de meeste fragmenten niet correct werken.

De Linesegment AABB techniek

Er bestaat een techniek die het meest gebruikt wordt en goed de Intersection berekend. De code, die ik ook laat zien, is een C# versie. De meeste codefragmenten worden in C# geschreven, maar dat komt ook omdat de Content Pipeline gebruikt wordt voor de objecten en gegevensbestanden - u weet waarschijnlijk nog wel, de Content van de XNA bibliotheek die we in Visual Basic helaas moeten missen. Bovendien is er nog een probleem waar ik het nog niet over gehad heb: de sprite-eigenschappen in de Content, ook die ontbreken in Visual Basic. Om welke eigenschappen het gaat, ziet u in de volgende Bulletin.

Wat is een linesegment?
Een linesegment is een lijndeel van een rechthoek. Dit lijnsegment moet met een ander linesegment worden gecontroleerd of ze elkaar tegenkomen of niet.

Het codefragment
Om te zoeken naar een snijpunt tussen 2D lijnen, is de volgende wiskundige berekening nodig:

Line: $Ax+By = C$

$A = y_2-y_1$
$B = x_1-x_2$
$C = A*x_1+B*y_1$

Vind nu het snijpunt:

$float\ det = A_1*B_2 - A_2*B_1$

```
  if(det == 0)
  {
      lijnen zijn parallel
  } else
  {
      float x = (B2*C1 - B1*C2)/det
      float y = (A1*C2 - A2*C1)/det
  }
```

De wiskundige schets kunnen we in code opschrijven in een methode die Vector2 punten heeft voor de 2 lijnen en resulteert de intersection punt;

De methode geeft een Vector2.Zero terug als de lijnen parallel zijn en er daardoor geen intersection plaatsvindt.

```
Vector2 LineIntersectionPoint(Vector2 ps1, Vector2 pe1,
                              Vector2 ps2, Vector2 pe2)
{
    // Neem A, B, C van eerste lijn - punten : ps1 tot pe1
    float A1 = pe1.y-ps1.y;
    float B1 = ps1.x-pe1.x;
    float C1 = A1*ps1.x+B1*ps1.y;

    // Neem A, B, C van tweede lijn - punten : ps2 tot pe2
    float A2 = pe2.y-ps2.y;
    float B2 = ps2.x-pe2.x;
    float C2 = A2*ps2.x+B2*ps2.y;

    // Neem delta en controleer of de lijnen parallel zijn
    float delta = A1*B2 - A2*B1;
    if(delta == 0)
        return Vector2.Zero;    // lijnen zijn parallel
    // resulteer de Vector2 intersection punt
    return new Vector2((B2*C1 - B1*C2)/delta,
                       (A1*C2 - A2*C1)/delta);
}
```

De punten die doorgegeven worden zijn altijd start - eind punten van de lijnen. Vandaar dat de variabelen ps1, pe1 en ps2, pe2 worden genoemd.

U kunt bovenstaande AABB techniek uitproberen, maar er zijn nog veel meer soorten voorbeelden op internet te vinden. Lijkt het u niks om wiskundig bezig te zijn, dan is er nog een manier om met lijnsegmenten te controleren: Gebruik maken van een rechthoek die teruggegeven wordt door de Intersect() functie. De functie geeft namelijk aan op welke plek de overlapping heeft plaatsgevonden. Misschien is dat wel wat voor u. De functie werkt als volgt:

Cr = Rectangle.Intersect(Ar, Br)

Verwar de functie niet met de Intersects() functie, die zo werkt:

Boolean = Ar.Intersects(Br)

De Vector2.Distance() methode
In plaats van bovenstaande techniek te gebruiken, kunnen we natuurlijk ook alle vier zijden van een rechthoek controleren met Intersects(). Het is echter beter en simpeler om door middel van een afstand te bepalen of de rechthoek van de speler (de bal) de rechthoek van een object (de paddle) tegenkomt. Die manier lijkt op de wiskundige berekening, maar nu resulteert de Distance() methode de berekende afstand tussen die twee.

Eerst hebben we onderstaande lengtes nodig van beide rechthoeken, gedeeld door 2 - het centerpunt. Voor algemeen gebruik neem ik als voorbeeld een player en een enemy, de lengtes kunnen dus bij u anders zijn.

```
float playerSize = player.Width / 2;     // het centerpunt van de
                                         rechthoek
float enemySize = enemy.Width / 2;
```

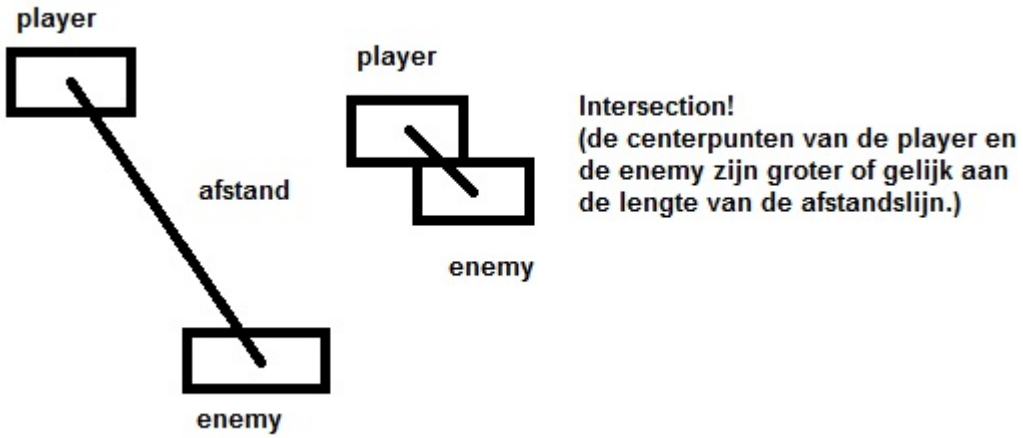
Voordat we de Intersects() methode gaan vervangen, berekenen we eerst de lokaties van de player en de enemy. De lokaties moeten centerpunten zijn om de afstand te kunnen bepalen tussen de player en de enemy:

```
Vector2 playerLocation = new Vector2(player.Width / 2,
                                     player.Height / 2);
Vector2 enemyLocation = new Vector2(enemy.Width / 2,
                                    enemy.Height / 2);
```

Onderstaande conditie bepaald of de halfsizes bij elkaar opgeteld groter of gelijk is dan de berekende afstand tussen het centerpunt van de player en het centerpunt van de enemy. Hier resulteer ik de conditie aan een boolean variabele **b**, maar u kunt de conditie ook direct gebruiken in een **if** statement:

```
b = (playerSize + enemySize >= Vector2.Distance(playerLocation,
                                                enemyLocation));
```

Als de playerSize plus de enemySize opgeteld groter of gelijk is dan de berekende afstand van de lokaties, dan is er een intersection. Waarom? Dat kan ik u hieronder laten zien.



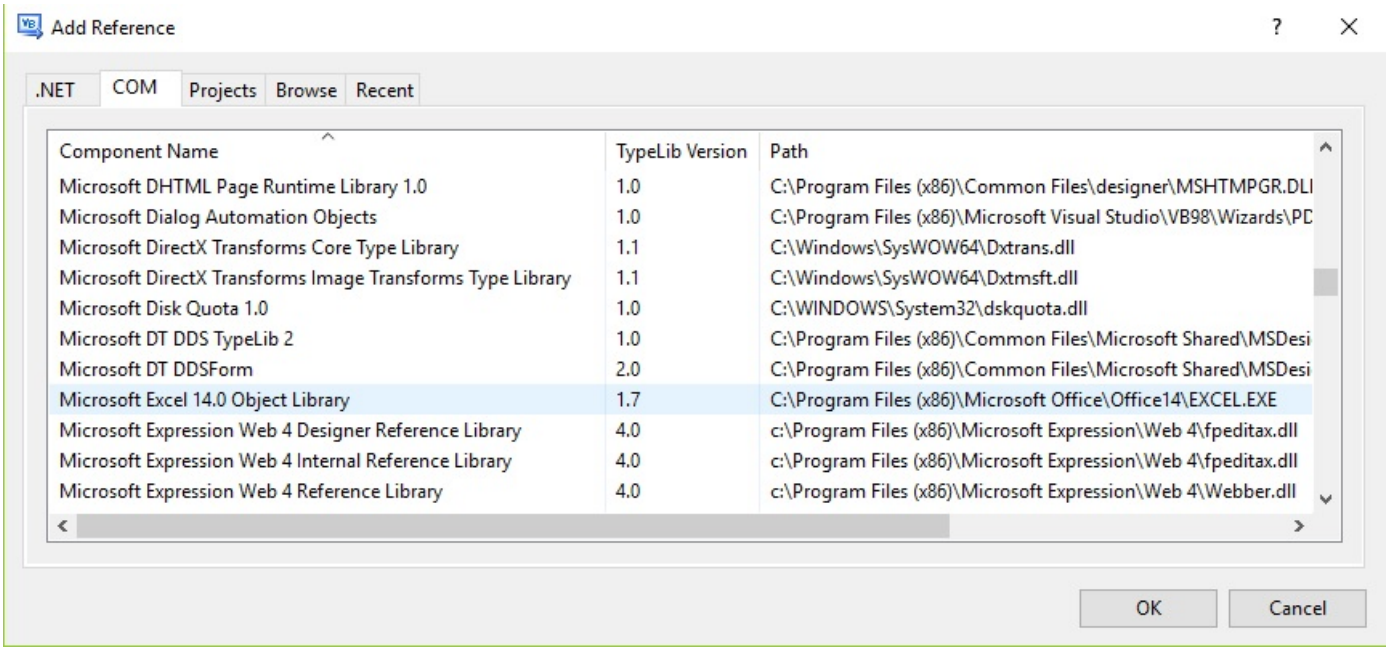
Op deze tekening kunt u het nu duidelijker zien wat er precies gebeurd. Hoe dichter de enemy bij de player komt, hoe kleiner de afstand wordt. Tot het zo dichtbij komt dat de afstand kleiner zal worden dan de halve lengtes van de player en de enemy. Vandaar dat die ook nodig zijn.

Mocht de grootte van de twee rechthoeken verschillen, dan nog zou de conditie prima werken. De conclusie is dat het om de totale som gaat van de twee halve lengtes van twee rechthoeken met een berekende afstand van twee centerpunten van de twee rechthoeken.

Excel – VBA als een Visual Studio Library.

VBA is een objectbibliotheek in Office programma's. Elk Office programma heeft zijn eigen VBA objectbibiliotheek. Het is mogelijk om de programma's als Excel op uw eigen manier te laten werken. U kunt uw eigen UserForms maken en met VBA aansturen met de werkbladen. Er is echter één probleem: VBA werkt intern. U kunt niet de Excel code compileren als een zelfstandig eigen EXE applicatie. U hebt altijd Excel nodig om uw project uit te laten voeren.

Maar misschien is er toch een oplossing. In de referentielijst van de programmeertalen in Visual Studio is een Excel COM library waarmee we met de VBA objecten in Visual Basic kunnen programmeren, en zelfs in Visual C# en in Visual C++. Wanneer we in Visual Basic 2010 een nieuwe solution aanmaken met een project, genaamd ExcelTest, kunnen we de Excel library toevoegen aan het project.



Kies in het menu **Project** het menu-item **Add Reference...**. Een dialoogformulier verschijnt. Open het tabblad COM en rol met de muis totdat u bij de library komt. Klik die aan, zie voorbeeld. Klik dan op OK.

> **Het kan voorkomen dat u een andere library versie tegenkomt. Dat heeft te maken met wat voor Excel versie u hebt.**
>
> **Houd er rekening mee dat de library alleen aanwezig kan zijn als u zelf Office met Excel heeft. De library is namelijk geen onderdeel van Visual Studio.**

Klik op de lege Form1 en open de klasse Form1. U ziet onderstaande klasse:

```
Public Class Form1

End Class
```

Voordat de library gebruikt kan worden, moet die geïmporteerd worden in de klasse. Neem onderstaande Imports regel over boven de Form1 klasse:

```
Imports Excel = Microsoft.Office.Interop.Excel
```

In de klasse zijn drie objecten nodig om gebruik te kunnen maken van de Excel objecten. Allemaal hebben ze een verband met elkaar. Het werkboek kunnen we niet aanmaken zonder een

applicatie object van Excel en dan zouden we ook geen werkblad aan kunnen maken als we geen werkboek zouden hebben. Neem onderstaande declaraties over in de klasse:

```
Private app As New Excel.Application
Private workbook As Excel.Workbook
Private worksheet As Excel.Worksheet
```

Open de event Form1_Load() en neem onderstaande code over:

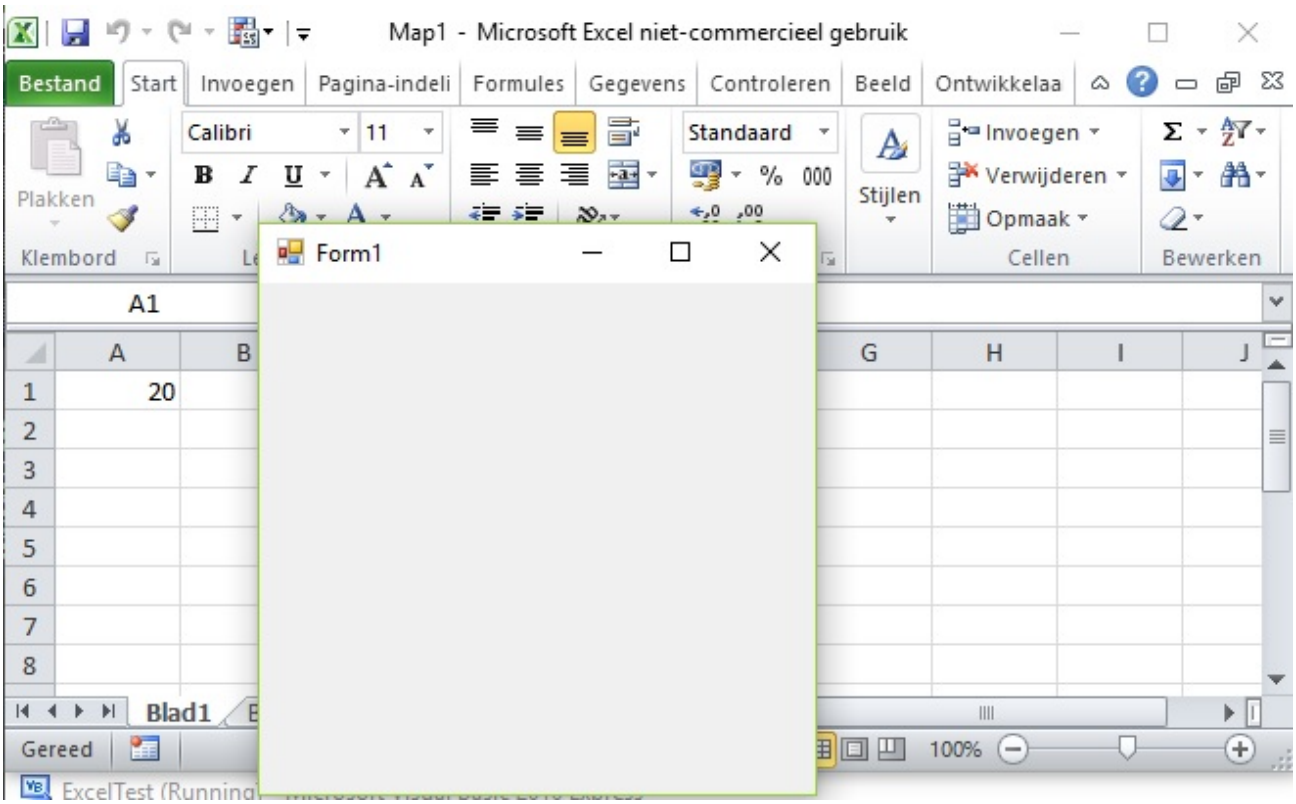
```
app.Visible = True
workbook = app.Workbooks.Add()
worksheet = workbook.Sheets(1)
worksheet.Range("A1").Value = 20
```

Als u vergeet de app.Visible = True regel op te nemen, zult u het werkboek niet zien. Toch is het werkboek onzichtbaar ook te gebruiken, en dat kan best nuttig zijn (daarover later meer).

Willen we Form1 sluiten, dan moeten we ook het werkboek sluiten en de Excel objecten opruimen om foutmeldingen te voorkomen. Plaats onderstaande code in de Form1_FormClosed() event:

```
workbook.Close()
app.Quit()
worksheet = Nothing
workbook = Nothing
app = Nothing
```

Als u het project start, zult u wat zien wat u niet zou verwachten. Althans, niet omdat het een object bibliotheek is!



Niet alleen Form1 van Visual Basic 2010 is aanwezig, maar ook het hele Excel programma! Hoe kan dat? We willen enkel een werkboek hebben zonder enige poespas.

Het probleem is dat de library een COM library is en geen zelfstandige klassen library. Zonder gebruik van Excel.Application kan er geen werkboek aangemaakt worden.

Ook als u gebruik zult maken van Visual C# of Visual C++ heeft u hetzelfde probleem.

Toch kan het gebruik van Excel in uw project nuttig zijn, bijvoorbeeld als u functies en formules wilt gebruiken die in Visual Studio niet aanwezig zijn. U kunt dan die functies en formules van Excel gebruiken door ze in de objecten aan te roepen.

Het project sluiten. Of toch eerst Excel?
Klikt u op het sluiten knop van Form1, dan zult u dezelfde vraag krijgen als wanneer u Excel sluit, of u Map1 wilt opslaan of niet. Bij de knoppen Opslaan en Niet opslaan werkt alles prima en uw project wordt samen met Excel keurig afgesloten. Klikt u echter op Annuleren, dan zal toch uw Form1 afgesloten worden, maar niet Excel. De reden is dat het berichtvenster niet van Visual Studio is, maar bij Excel vandaan komt, *ook al werkt u alleen met Form1!* Alle objecten worden opgeruimd, maar Excel blijft aanwezig. Merk op dat het berichtvenster na een klik op Annuleren nogmaals zal verschijnen zonder Form1. Waarschijnlijk doordat er Workbook.Close() en een app.Quit() uitgevoerd worden. Het zou kunnen dat de app gesloten kan worden zonder het werkboek te hoeven sluiten.

Sluiten we echter eerst Excel en daarna pas Form1, dan heeft dat grote gevolgen en kan uw project gaan hangen of zelfs crashen. Het sluiten van Excel zorgt voor het ongeldig maken van het app object. Als we die proberen te sluiten en te vernietigen met Nothing, dan zal het project meteen hangen en Form1 zal niet meer reageren. In de IDE kunt u het project onderbreken naast de uitvoerknop. Als het een uitvoerbaar bestand is zult u dat niet meer kunnen doen en is de enige mogelijkheid om het programma via taakbeheer af te breken.

Het is daarom zeer belangrijk gebruik te maken van foutafhandelingen als u objectbibliotheken gaat gebruiken die deze problemen kunnen veroorzaken. COM library's moet u goed beschermen, want het gebruik ervan is heel anders dan bij klassen library’s.

Toch nog een voordeel.
Ook al is deze manier niet zoals u zou willen, toch kan het gebruik van het Excel COM library nuttig zijn. Door gebruik te maken van de Formula eigenschap, kunt u elke Excel formule in het worksheet object aanroepen. Jammer genoeg moet u de formules in het Engels programmeren, terwijl ze in Excel zelf Nederlandstalig zijn.

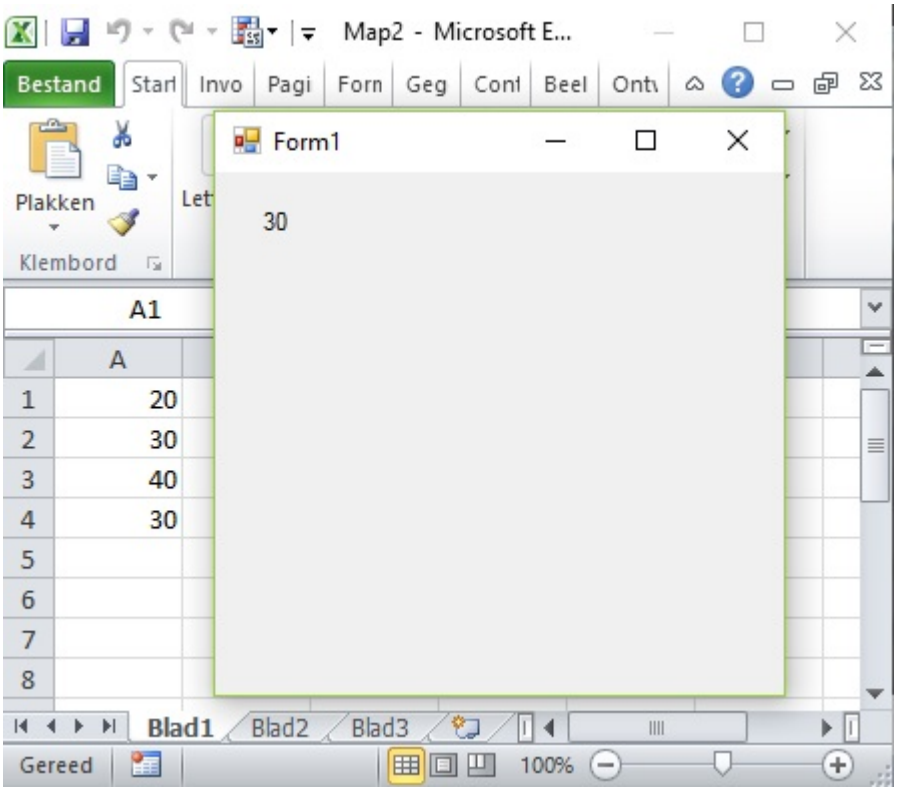
Om zo'n voordeel te kunnen zien, gaan we een testje maken. We breiden het project uit door nog twee extra getallen in de cellen A2 en A3 te zetten en een gemiddelde van de waarden in de cellen A1, A2 en A3 in A4 uit te voeren. Het resultaat plaatsen we in een label in Form1.

Klik op de toolbox de label control en sleep deze linksboven op Form1.

Plaats onderstaande code onder de Worksheet.Range("A1").Value = 20 regel:

```
worksheet.Range("A2").Value = 30
worksheet.Range("A3").Value = 40
worksheet.Range("A4").Formula = "=AVERAGE(A1:A3)"
Label1.Text = worksheet.Range("A4").Value.ToString
```

Als u het project start, ziet u onderstaand voorbeeld.



Het gemiddelde wordt prima uitgevoerd. Schuift u Form1 aan de kant, dan zult u merken dat in cel A4 gewoon de fomule =GEMIDDELDE in de invoerbalk staat en niet de opgegeven =AVERAGE.

Maakt u een groter project met gebruik van de handige Excel functies, dan zal het verschijnen van Excel zelf kunnen irriteren, de applicatie staat gewoon in de weg. Het is mogelijk om de applicatie onzichtbaar te laten en gewoon de objecten te laten werken.

Dit doet u door de regel:

```
app.Visible = True
```

in commentaar te zetten, of True in False te wijzigen.

Samenvatting

U kunt hiermee uw projecten functioneler maken. Door uw project samen met de Excel objecten te laten werken, kunt u formulieren ontwerpen dat normaal gesproken op een UserForm in Excel niet kan, vanwege gebrek aan materiaal (besturingselementen). Bovendien werkt VBA alleen als een versie 6.

Maar pas op! U kunt van uw project ook een rommel maken. Als u de Excel applicatie zichtbaar laat, dan is het mogelijk dat u per ongeluk VBA opent. In VBA kan ook een compleet project ontwikkeld worden. Het uitvoeren van de code van Visual Studio kan het VBA project in Excel verstoren, of andersom. Wees dus voorzichtig met het gebruik van Excel in Visual Studio. Als u echter weet wat u doet, is er niets aan de hand.

Liberty BASIC API reference.

ShellExecute

Deze functie voert een applicatie uit die gekoppeld is aan een opgegeven bestandsextentie. Als bijvoorbeeld de standaard applicatie voor bestanden met de extentie "TXT" het Windows Kladblok is, zal de aanroep het kladblok openen met het opgegeven bestand. Wanneer een programma een andere applicatie start, zal deze wachten totdat de applicatie gesloten wordt.
Zie WaitForSingleObject.

```
showWindowFlag = _SW_SHOWNORMAL

lpszOp$ = "open"            'kies "open" of "print" of "explore"

lpszFile$ = "readme.txt"    'bestandsnaam die geladen moet worden

lpszDir$ = DefaultDir$      'standaard directory

lpszParams$ = ""            'maak deze null

calldll #shell32, "ShellExecuteA", _
    hWin as ulong, _            ' het ouder venster handle
    lpszOp$ as ptr, _           ' "open" of "print"
    lpszFile$ as ptr, _         ' bestand om te openen of af te drukken
    lpszParams$ as ptr, _       ' null
    lpszDir$ as ptr, _          ' directory
    showWindowFlag as long, _   ' modus om applicatie te starten
    result as long              ' resulteert de instantie handle van de
                                ' applicatie die was gestart
```

Hoe de pad en de bestandsnaam gescheiden wordt vanuit een filedialog aanroep:

```
filedialog "Kies bestand...", "*.*", file$

index = len(file$)
length = len(file$)         'maak de lengte klaar voor de while/wend lus

while mid$(file$, index, 1) <> "\"    'zolang geen backslash
    index = index-1         'kijk verder voor de meest rechtse backslash
wend

'de bestandsnaam zonder station/pad informatie
filename$ = right$(file$, length - index)
directory$ = left$(file$, index)
```

Mogelijke showWindowFlags:

_SW_HIDE	Verbergt het venster en zorgt voor het activeren van een ander venster.
_SW_MINIMIZE	Minimaliseert het venster en activeert het top-level venster in de systeemlijst.
_SW_RESTORE	Activeert en toont een venster. Als het venster geminimaliseerd of gemaximaliseerd is, zal Windows het naar de originele grootte en positie herstellen (zelfde als

	_SW_SHOWNORMAL).
_SW_SHOW	Activeert een venster en toont het in zijn huidige grootte en positie.
_SW_SHOWMAXIMIZED	Activeert een venster en toont het als een gemaximaliseerd venster.
_SW_SHOWMINIMIZED	Activeert een venster en toont het als een icoon.
_SW_SHOWMINNOACTIVE	Toont een venster als een icoon. Het venster dat momenteel actief is blijft actief.
_SW_SHOWNA	Toont het venster in zijn huidige staat. Het venster dat momenteel actief is blijft actief.
_SW_SHOWNOACTIVATE	Toont een venster in zijn meest recente grootte en positie. Het venster dat momenteel actief is blijft actief.
_SW_SHOWNORMAL	Activeert en toont een venster. Als het venster geminimaliseerd of gemaximaliseerd is, zal Windows het naar de originele grootte en positie herstellen (zelfde als _SW_RESTORE).

Sleep

De Sleep functie schorst de uitvoering van de huidige thread voor een opgegeven interval. De waarde is gegeven in milliseconden. De functie resulteert geen waarde.

```
calldll #kernel32, "Sleep", _
    value as long, _     'milliseconden om te pauzeren
    r as void
```

WaitForSingleObject

De WaitForSingleObject functie resulteert zodra één van de twee plaatsvindt: Het opgegeven object is in de gesignaleerde status of de time-outinterval is verstreken. Het vereist een handle naar het uitvoeringsproces, dat verkregen kan worden met de ShellExecuteExA functie. Die functie vereist een SHELLEXECUTEINFO structuur. Gebruik deze methode om een externe toepassing te starten binnen uw programma en wacht tot het voltooid is voordat u uw programma hervat.

```
filedialog "Open", "*.txt", file$
if file$ = "" then end

SEEMASKNOCLOSEPROCESS = 64    '0x40

struct s, _
    cbSize as ulong, fMask as ulong, hwnd as ulong, _
    lpVerb$ as ptr, lpFile$ as ptr, lpParameters$ as ptr, _
    lpDirectory$ as ptr, nShow as long, hInstApp as ulong, _
    lpIDList as long, lpClass as long, hKeyClass as ulong, _
    dwHotKey as ulong, hIcon as ulong, hProcess as ulong

s.cbSize.struct = len(s.struct)
s.fMask.struct = SEEMASKNOCLOSEPROCESS
s.hwnd.struct = 0
s.lpVerb$.struct = "Open"
s.lpFile$.struct = file$
s.lpParameters$.struct = ""
s.lpDirectory$.struct = DefaultDir$
```

```
s.nShow.struct = _SW_RESTORE

calldll #shell32, "ShellExecuteExA", s as struct, r as long

if r <> 0 then
    hProcess = s.hProcess.struct
else
    print "Fout."
    end
end if

waitResult = -1
while waitResult <> 0
    calldll #kernel32, "WaitForSingleObject", _
        hProcess as long, 0 as long, waitResult as long
wend
print "Uitvoeringsproces is beëindigd"
end
```

WinExec

Deze functie voert een ander programma uit binnen een applicatie. Als de tekenreeks een applicatie bevat zonder de pad, zoekt Windows de directory's af in onderstaande volgorde:

1. De huidige directory.
2. De windows directory (de directory met WIN.COM); de GetWindowsDirectory functie haalt de pad van de directory.
3. De Windows systeem directory (de directory die systeembestanden bevat, zoals gdi.exe); de GetSystemDirectory functie haalt de pad van de directory.
4. De directory die het uitvoerbare bestand voor de huidige taak bevat; de GetModuleFileName functie haalt de pad van de directory.
5. De directory's die als een lijst in de PATH omgevingsvariabele staan.
6. De directory's toegewezen in een netwerk.

```
calldll #kernel32, "WinExec", _
    winFile$ as ptr, _          'bestand om uit te voeren
    _SW_SHOWNA as long, _       'zie ShellExecute voor de waarden van de
vlag
    result as long              'succes als >= 31
```

In de volgende Bulletin komen de API's over het gebruik van de INI bestanden, het Windows Register en het klembord.

Turbo Pascal – De start om te beginnen.

De Free Pascal Compiler en Turbo Pascal lijken op elkaar. Zelf denk ik dat Turbo Pascal een programmeertechniek kent dat Free Pascal nog niet kent. Omdat ik nog nooit met Free Pascal geprogrammeerd heb, weet ik eerlijk gezegd daar ook geen antwoord op. Zelf neem ik aan dat de meeste voorbeelden, die hier zullen verschijnen, in Free Pascal ook gebruikt kunnen worden.

In begin jaren ’90 leerde ik Turbo Pascal in alleen de eerste klas van de MTS in Ede. Al snel merkte ik nog meer van de taal te weten dan de anderen en, nadat ik de tweede klas helaas niet haalde maar ze mij toch als ‘stageloper’ lieten werken, begon ik les te geven in Pascal aan dertig leerlingen uit de 3de klas. Daarom is programmeren ook mijn sterkste punt. Ook 5de generatietalen heb ik op de MTS gebruikt voor het programmeren van robots. Tien robotarmen programmeerde ik met BBC BASIC en één grote robot met een rolbaan moest geprogrammeerd worden met Scara.

Het hoofdblok of ook wel genoemd de programmakern
In Pascal moeten we ons aan een hele andere structuur houden dan we in andere programmeertalen gewend zijn. Codeblokken moeten binnen een **begin** en een **end** staan, net zoals de accolades in C. Basic kent geen statements of symbolen die de codeblokken moeten bepalen. Toch is er in Visual Basic .NET wel zoiets bijgekomen, zoals de accolades die nodig zijn om waarden in declaraties te initialiseren.

Het hoofdblok begint optioneel met een **program**. Daarna een **begin**, het codeblok en als laatst een **end** met een punt. De punt is uniek. Voor zover ik een aantal programmeertalen ken, heeft alleen Pascal een **end** met een punt. Als er toch nog code eronder staat, wordt dat door de compiler genegeerd.

```
program Welkom;
begin
     Writeln('Welkom bij de programmeertaal Pascal!');
     Readln
end.
```

Het statement dat voor een **end** staat hoeft geen puntkomma te hebben. Het kan geen kwaad om achter **Readln** een puntkomma te plaatsen, maar de compiler beschouwd het dan wel als een null-statement.

Als u een foutje maakt, bijvoorbeeld ‘Writelm’ dan zal de compiler melden:
    Error 3: Unknown identifier

Strings
Een string hoort tussen enkele aanhalingstekens te staan. Hoewel de lengte van een string 255 tekens is, kan Turbo Pascal alleen regels met een lengte van 127 tekens compileren. Toch is het in de editor van Turbo Pascal mogelijk 248 tekens per regel in te tikken. Andere editors handhaven soms andere lengtes. Dit soort details kunnen u grijze haren bezorgen, besteed er maar niet teveel aandacht aan.

Om lange strings te kunnen gebruiken, kunnen we deze splitsen met het plus-teken. Dit is handig om lange regels te voorkomen, zoals hieronder:

```
Writeln('Een voor allen ' +
          'en allen ' +
          'voor een.');
```

Het concateneren (samenvoegen) kan ook met variabelen worden gedaan.

Wanneer we een enkel aanhalingsteken willen afdrukken, moeten we deze nog eens intikken en erachter met de rest van de string verder gaan, bijvoorbeeld:

```
Writeln('Daar rijdt weer zo''n mooie auto!');
```

Variabelen
De plaatsen waar variabelen gedeclareerd moeten worden is altijd voor een **begin**, maar nooit in een codeblok zelf. Ze worden gedeclareerd na een **program**, na een **procedure** en na een **functie**. Er zijn nog meer plaatsen waar variabelen gedeclareerd mogen worden, maar dat is altijd weer voordat een codeblok begint.

```
program WieBentU;
var
     UwNaam: string[40];
begin
     Write('Hoe heet u? ');
     Readln(UwNaam);
     Writeln('Hallo ', UwNaam, '!');
     Readln
end.
```

Onthoud dat u codeblokken mag nesten, maar het is dan niet toegestaan variabelen voor een binnenst **begin** te declareren.

```
begin
     var A: Integer;
     begin
          ...
     end
end.
```

De compiler zal tegenstribbelen en de foutmelding ‘Illegal expression’ geven.

Constanten
Wanneer u vaak met dezelfde waarden te maken heeft, is het zinvol om constanten te gebruiken. Ook de constantendeclaratie moet voor een **begin** staan, maar ook voor de variabelendeclaratie.

```
program Constanten;
const
     Naam = 'Anne';
     Gewicht = 59;
     Spaargeld = 375;
begin
     Writeln('Naam   : ', Naam);
     Writeln('Gewicht: ', Gewicht, ' kilo');
     Writeln(Naam, ' heeft € ', Spaargeld, ' gespaard.');
     Readln
end.
```

Subranges
Pascal kent een mogelijkheid om waarden binnen een bepaalde grens te houden. Buiten de grens zal de compiler niet accepteren. Het declareren van subranges werkt als volgt:

```
var Gewicht: 2..125;
```

Probeert u een toekenning te geven als:

```
Gewicht := 130;
```

dan zal de compiler een foutmelding geven, omdat de waarde 130 buiten de grens ligt.

Toch kan het gewicht bij sommige mensen hoger zijn dan 125. Het gebruik van constanten is dan weer nuttig, zoals onderstaand programma laat zien.

```
program HetGewicht;
const
     ZwaarsteGewicht = 140;
var
     Gewicht: 2..ZwaarsteGewicht;
begin
     Write('Wat is uw gewicht? ');
     Readln(Gewicht);
     Writeln('Uw gewicht is: ', Gewicht);
     Readln
end.
```

Typt u een getal in die kleiner is dan 2 of groter is dan ZwaarsteGewicht, dan zal de compiler een waarschuwing geven:

Warning: Range check error while evaluating constants

Op internet is heel veel te vinden over hoe Pascal werkt. Hebt u een leuk stukje over Pascal, stuur deze dan op.

De volgende keer komen recordtypes en objecttypes aan bod en laat ik een andere Pascal programmeertaal zien – Lazarus. Lazarus is een Windows IDE object georiënteerde taal met veel componenten. Naast de recordtypes en objecttypes worden ook de structuren gebruikt, zoals lussen, keuzes, meervoudige keuzes, condities en expressies, procedures en functies.

Python – Een introductie

Python is een programmeertaal met een interpreter, maar vanuit de interpreter kunnen we ook naar de File/Line. Het venster zal dan er anders uit zien.

Als we Python starten, krijgen we de versie te zien met de datum en meer informatie. Onder de informatie staan deze tekens: **>>>**. Achter deze tekens kunnen we Python regels invoeren. Regels, onder andere expressies en statements. Python kan bijvoorbeeld als een rekenmachine werken. In andere interpreters moeten we met een **print** statement beginnen als we een expressie uit willen voeren, maar in Python is dat niet nodig.

Laten we eens kennismaken met Python. Volg onderstaande introductie om te begrijpen hoe het werkt.

Python gebruiken als een rekenmachine

Numerieke waarden

In elke programmeertaal kunnen we expressies gebruiken met getallen. De operatoren +, -, * en / werken op dezelfde manier. Ook de haakjes kunnen we gebruiken om te groeperen.

Typ achter de **>>>** `2 + 2`. Het antwoord zal onder direct 4 zijn. Gebruik geen = en ook geen ? of print om sommen uit te rekenen.

Typ de volgende expressies net zo:

```
>>> 50 - 5 * 6
20          Dit geeft Python als antwoord.
>>> (50 - 5 * 6) / 4
5.0
>>> 8 / 5  # deling geeft altijd een floating point waarde
1.6
```

De waarden 4 en 20 zijn van het type **int**. De waarde 1.6 is van het type **float**. Later meer over soorten numerieke types.

Het #-teken kunt u gebruiken voor commentaarregels. In Python worden ze standaard schuingedrukt.

De deling (/) retourneert altijd een float. Om een integer resultaat te krijgen kunt u de // operator gebruiken. Voor een rest deling gebruikt u de % operator.

```
>>> 17 / 3  # klassieke deling retourneert een float
5.666666666666667
>>> 17 // 3  # dit kapt de nummers rechts van de punt af
5
>>> 17 % 3  #  de % operator retourneert de rest van de deling
2
>>> 5 * 3 + 2  # resultaat * deler + rest
17
```

Python kent ook een macht-operator als twee **:

```
>>> 5 ** 2  # kwadraat
25
>>> 2 ** 7  # 2 tot de macht van 7
```

```
128
```

Het gelijkheidsteken = wordt gebruikt om waarden toe te kennen aan variabelen:

```
>>> width = 20
>>> height = 5 * 9
>>> width * height
900
```

Als u probeert een variabele te gebruiken die nog niet bestaat, geeft Python (met een hoop informatie) een foutmelding:

```
>>> n  # probeer toegang te krijgen tot een ongedefinieerde variabele
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'n' is not defined
```

In directe mode wordt het laatste gedrukte resultaat toegekend aan de variabele _. Hiermee wordt bedoeld dat als u Python gebruikt als een rekenmachine, het soms gemakkelijker is om met het laatste resultaat verder te kunnen rekenen:

```
>>> tax = 12.5 / 100
>>> price = 100.50
>>> price * tax
12.5625
>>> price + _
113.0625
>>> round(_, 2)
113.06
```

Deze variabele moet u zien als een alleen-lezen variabele. Ken geen waarde er aan toe – u zult automatisch een lokale variabele met dezelfde naam aanmaken, waardoor de ingebouwde variabele zijn magie kwijtraakt.

Strings

Python kan ook strings manipuleren die in verschillende manieren uitgedrukt kunnen worden. Ze kunnen tussen enkele aanhalingstekens (‘…’) of dubbele aanhalingstekens (“…”) worden getypt met hetzelfde resultaat. Het \ teken kan worden gebruikt om de aanhalingstekens te vermijden:

```
>>> 'spam eggs'  # enkele aanhalingstekens
'spam eggs'
>>> 'doesn\'t'  # gebruik \' om het enkele aanhalingsteken te vermijden
"doesn't"
>>> "doesn't"  # of gebruik in plaats daarvan dubbele aanhalingstekens
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
>>> "\"Yes,\" he said."
'"Yes," he said.'
>>> '"Isn\'t," she said.'
'"Isn\'t," she said.'
```

In de interpreter, de directe mode, wordt in de uitvoerstring tussen aanhalingstekens de speciale tekens vermeden met backslash es. Dit maakt soms verschillen met de invoer (de

aanhalingstekens kunnen veranderen), de twee strings zijn hetzelfde. De string staat tussen dubbele aanhalingstekens als de string als inhoud een enkel aanhalingsteken bevat en geen dubbele aanhalingstekens, anders staat het tussen enkele aanhalingstekens. De **print()** functie kan een beter leesbare uitvoer geven door de eerste en laatste aanhalingstekens niet af te drukken en door het af te drukken van ontsnapte tekens en speciale tekens:

```
>>> '"Isn\'t," she said.'
'"Isn\'t," she said.'
>>> print('"Isn\'t," she said.')
"Isn't," she said.
>>> s = 'First line.\nSecond line.'  # \n is nieuwe regel
>>> s  # zonder print(), \n wordt ook afgedrukt
'First line.\nSecond line.'
>>> print(s)  # met print(), \n zorgt voor een nieuwe regel
First line.
Second line.
```

Wilt u geen tekens achter een \ als speciale tekens, gebruik dan *raw strings* door een **r** voor het eerste aanhalingsteken toe te voegen:

```
>>> print('C:\some\name')  # hier betekent \n nieuwe regel!
C:\some
ame
>>> print(r'C:\some\name')  # onthoud de r voor het aanhalingsteken
C:\some\name
```

Letterlijke stringwaarden kunnen uit meerdere regels bestaan. Een mogelijkheid is gebruik te maken van 3 aanhalingstekens: """…""" of '''…'''. Het einde van de regels zijn automatisch ingevoegd in een string, maar het is mogelijk om dat te voorkomen door een \ aan het einde van een regel toe te voegen, zoals onderstaand voorbeeld:

```
print("""\
Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname               Hostname to connect to
""")
```

produceert de volgende uitvoer (merk op dat de geïnitialiseerde nieuwe regel niet ingevoegd is):

```
Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname               Hostname to connect to
```

Strings kunnen samengevoegd worden (aan elkaar worden gezet) met de + operator en worden herhaald met *:

```
>>> # 3 keer 'un', gevolgd door 'ium'
>>> 3 * 'un' + 'ium'
'unununium'
```

Twee of meer letterlijke strings (dat wil zeggen, de een tussen aanhalingstekens) gevolgd met elke andere zullen automatisch samengevoegd worden:

```
>>> 'Py' 'thon'
'Python'
```

Dit werkt alleen met elke letterlijke string, maar niet met variabelen of expressies:

```
>>> prefix = 'Py'
>>> prefix 'thon'  # kan niet samengevoegd worden met een variabele
                   en een letterlijke string
  ...
SyntaxError: invalid syntax
>>> ('un' * 3) 'ium'
  ...
SyntaxError: invalid syntax
```

Wilt u toch waarden van variabelen of van een variabele en een letterlijke string samenvoegen, gebruik dan +:

```
>>> prefix + 'thon'
'Python'
```

Deze mogelijkheid is nuttig wanneer u lange strings wilt breken:

```
>>> text = ('Put several strings within parentheses '
...         'to have them joined together.')
>>> text
'Put several strings within parentheses to have them joined together.'
```

Strings kunnen worden *geïndexeerd* (subscripted) met index 0 als eerste teken. Er is geen apart teken-type; een teken is simpelweg een string met als grootte 1:

```
>>> word = 'Python'
>>> word[0]  # teken in plaats 0
'P'
>>> word[5]  # teken in plaats 5
'n'
```

De indexnummers mogen ook negatief zijn om aan de rechterkant te beginnen:

```
>>> word[-1]  # laatste teken
'n'
>>> word[-2]  # tweede laatste teken
'o'
>>> word[-6]
'P'
```

Onthoud dat -0 hetzelfde is als 0, negatieve indexnummers beginnen met -1.

In tegenstelling tot indexering is *slicing* (snijden) ook toegestaan. Door slicing te gebruiken, kunt u substrings (gedeelde strings) weergeven:

```
>>> word[0:2]  # tekens vanaf plaats 0 (opgenomen) tot 2 (uitgesloten)
'Py'
>>> word[2:5]  # tekens vanaf plaats 2 (opgenomen) tot 5 (uitgesloten)
'tho'
```

Onthoud dat de start altijd opgenomen is en het eind altijd uitgesloten is.
Dit zorgt ervoor dat `s[:i] + s[i:]` altijd gelijk is aan `s`:

```
>>> word[:2] + word[2:]
```

```
'Python'
>>> word[:4] + word[4:]
'Python'
```

Een indexnummer dat te hoog is geeft als resultaat een foutmelding:

```
>>> word[42]  # word heeft maar 6 tekens
  ...
IndexError: string index out of range
```

Hoewel, een buiten slice indexnummer zal soepel verlopen, wanneer u slicing gebruikt:

```
>>> word[4:42]
'on'
>>> word[42:]
''
```

Python strings kunnen niet gewijzigd worden – ze zijn onveranderbaar. Daarom zal een toekenning aan een indexpositie in een string een foutmelding retourneren:

```
>>> word[0] = 'J'
  ...
TypeError: 'str' object does not support item assignment
>>> word[2:] = 'py'
  ...
TypeError: 'str' object does not support item assignment
```

Als u een andere string nodig heeft, moet u een nieuwe maken:

```
>>> 'J' + word[1:]
'Jython'
>>> word[:2] + 'py'
'Pypy'
```

De ingebouwde functie **len()** retourneert de lengte van een string:

```
>>> s = 'supercalifragilisticexpialidocious'
>>> len(s)
34
```

Lijsten

Python kent een aantal samengestelde gegevenstypes die gebruikt worden voor het groeperen van andere waarden. De meest veelzijdige is de *lijst* die geschreven kan worden als een lijst uit komma-gescheiden waarden (items) tussen vierkante haken. Lijsten mogen als inhoud verschillende item-types hebben, maar meestal hebben alle items hetzelfde type:

```
>>> squares = [1, 4, 9, 16, 25]
>>> squares
[1, 4, 9, 16, 25]
```

Lijsten kunnen, net zoals bij strings en bij alle andere ingebouwde types, geïndexeerd en gesliced (in delen teruggegeven) worden:

```
>>> squares[0]  # geïndexeerd retourneert het item
1
```

```
>>> squares[-1]
25
>>> squares[-3:]  # slicing retourneert een nieuwe lijst
[9, 16, 25]
```

Alle slice bewerkingen retourneren nieuwe lijsten met als inhoud de gevraagde elementen. Dit betekent dat de volgende slice een nieuwe (ondiepe) kopie van de lijst retourneert:

```
>>> squares[:]
[1, 4, 9, 16, 25]
```

Lijsten ondersteunen ook samenvoegingen:

```
>>> squares + [36, 49, 64, 81, 100]
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Hoewel strings onveranderbaar zijn, zijn lijsten een veranderbaar type. Het is toegestaan hun inhoud te wijzigen:

```
>>> cubes = [1, 8, 27, 65, 125]  # er is wat fout gegaan hier
>>> 4 ** 3  # de kubus van 4 is 64, niet 65!
64
>>> cubes[3] = 64  # vervang de foute waarde
>>> cubes
[1, 8, 27, 64, 125]
```

U kunt ook nieuwe items aan het einde van de lijst toevoegen met gebruik van de **append()** methode:

```
>>> cubes.append(216)  # voeg de kubus van 6 toe
>>> cubes.append(7 ** 3)  # en de kubus van 7
>>> cubes
[1, 8, 27, 64, 125, 216, 343]
```

Toekennen aan slices is ook mogelijk, de lengte van de lijst kan worden gewijzigd en ook worden schoongemaakt:

```
>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> letters
['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> # vervang sommige waarden
>>> letters[2:5] = ['C', 'D', 'E']
>>> letters
['a', 'b', 'C', 'D', 'E', 'f', 'g']
>>> # verwijder ze nu
>>> letters[2:5] = []
>>> letters
['a', 'b', 'f', 'g']
>>> # maak nu de hele lijst schoon door ze allemaal te verwijderen
>>> letters[:] = []
>>> letters
[]
```

De ingebouwde functie **len()** werkt ook op lijsten:

```
>>> letters = ['a', 'b', 'c', 'd']
```

```
>>> len(letters)
4
```

Het is ook mogelijk lijsten te nesten. Elk item zal dan een lijst zijn:

```
>>> a = ['a', 'b', 'c']
>>> n = [1, 2, 3]
>>> x = [a, n]
>>> x
[['a', 'b', 'c'], [1, 2, 3]]
>>> x[0]
['a', 'b', 'c']
>>> x[0][1]
'b'
```

De eerste stappen naar programmeren

Natuurlijk kunnen we Python voor gecompliceerdere dingen gebruiken dan het toevoegen van twee aan twee aan elkaar. We kunnen bijvoorbeeld een sub-volgorde *Fibonacci* serie schrijven, als volgt:

```
>>> # Fibonacci series:
... # de som van twee elementen definieert het volgende
... a, b = 0, 1
>>> while b < 10:
...     print(b)
...     a, b = b, a+b
...
1
1
2
3
5
8
```

De eerste regel is een *meervoudige toekenning*: de variabelen **a** en **b** krijgen tegelijk de nieuwe waarden 0 en 1. In de laatste regel wordt dit opnieuw gedaan, hier gedemonstreerd dat een expressie aan de rechterkant voorrang heeft voordat een toekenning plaatsvindt. De expressies aan de rechterkant worden van links naar rechts uitgevoerd.

De **while** lus voert alles uit zolang de conditie (hier: **b < 10**) waar is. Zoals in C is in Python ook elke niet-nul integer waarde waar, een nul is onwaar. De conditie mag ook een string zijn of een lijstwaarde. Van elke volgorde is een niet-nul lengte een waar, een lege is een onwaar. De test in het voorbeeld is een simpele vergelijking. De standaard vergelijking operatoren zijn hetzelfde als in C.

Samenvatting

Python is een programmeertaal om prima als beginner ermee te starten. U hebt kunnen zien hoe de structuur van Python eruit ziet. Maar we kunnen nog meer doen in Python, zoals het bovenstaande voorbeeld laat zien met een while lus. We kunnen met gegevensstructuren werken, modules maken en zelfs eigen klassen bouwen. In de volgende Bulletin ga ik daarom verder met Python om nog meer te kunnen ontdekken wat we met deze programmeertaal kunnen doen.