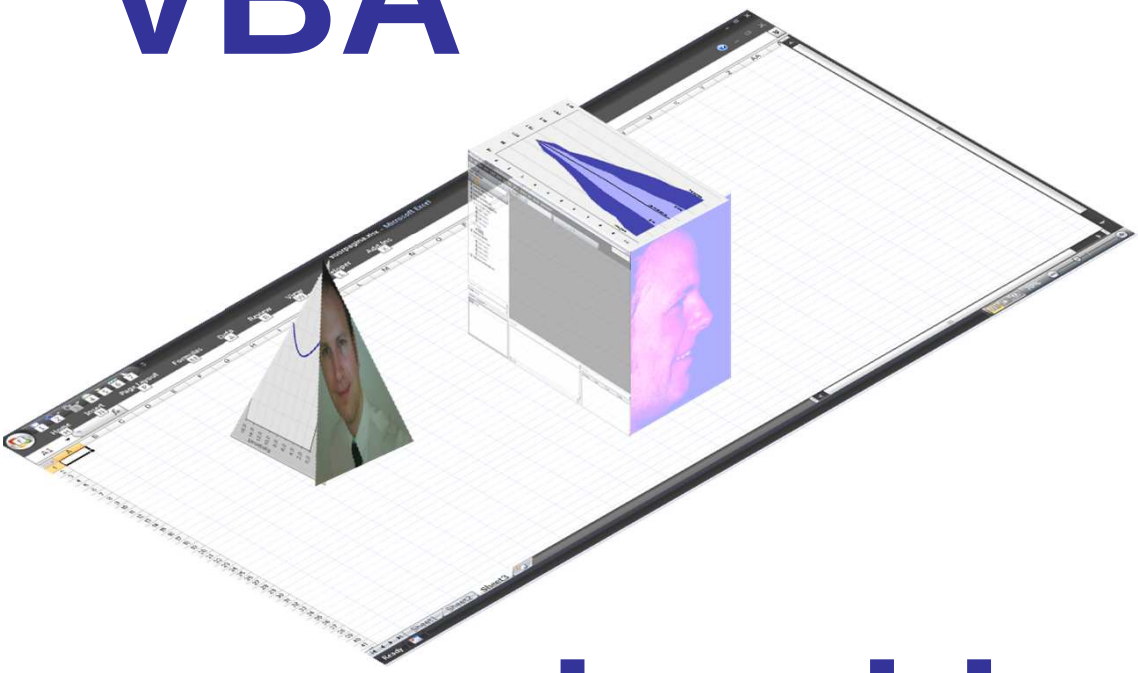
vangilst
Actuariaat & Software
Verveldhorst 28
7531 DW Enschede
freek@vangilst.com
06-408.45.45.3

VBA



boekje

Versie 1.9

Inhoud

Inleiding

Dit document geeft regels en tips voor het maken van VBA-programma's. Doel ervan is om de VBA-programma's zowel leesbaar als snel te laten zijn.

Notaties in dit document:

Tekst van een voorbeeld is cursief. `Een voorbeeld van programmacode is echter niet cursief en staat in het lettertype Courier.`

Toetsencombinaties worden aangegeven met (een afkorting van) de modificerende toets(en) en daarna de toets die tegelijk daarmee moet worden ingedrukt. Tussen de toetsen staat een minus (-). Die modificerende toetsen zijn "Ctrl", "Shift", of "Alt".

Bijvoorbeeld:

> *Het tegelijk indrukken van de Ctrl-toets en de Alt-toets en de Delete-toets wordt hier genoteerd als "Ctrl-Alt-Delete".*

Afkortingen en termen

Procedure: een Sub, Function, Property Get of Property Let.

Codemodule: een module, class module of form (d.w.z. de VBA-code achter de form).

VBE: de VBA-editor.

Stijl

In deze categorie vallen de regels die de code leesbaarder maken. Dat wil zeggen dat bij een review of bij onderhoud aan een programma gemakkelijker is te zien hoe een stukje code werkt. Daardoor kunnen fouten gemakkelijker worden gevonden of kan de code gemakkelijker worden aangepast zonder fouten te introduceren.

Het zal regelmatig voorkomen dat de code duidelijker is wanneer van deze standaards wordt afgeweken. In een dergelijk geval verdient het zeker aanbeveling om de regels niet te volgen.

Bijvoorbeeld:

> *Een variabele bevat de rekenrente. De rekenrente wordt in de financiële rekenkunde vaak aangeduid met de kleine letter "i". De regels voor stijl stellen dat de naam van een variabele begint met een hoofdletter. Maar in dit geval is het duidelijker om de variabele "i" te noemen, dus met een kleine letter.*

Waarom dit hoofdstuk?

Er zijn immers al meer beschrijvingen van zogenoemde coding standards, naming conventions e.d.

Maar die zijn er nog niet in het Nederlands en ze zijn ook niet gericht op de financiële sector. Verder maken ze vaak nog gebruik van de verouderde Systems Hungarian Notation (zie verderop voor een uitleg over Hungarian Notation).

Naamgeving

Hoofdletters/kleine letters

Wanneer we ergens een naam aan geven dan begint die naam met een hoofdletter.

Vaak bestaat een naam uit meerdere woorden. Omdat VBA niet om kan gaan met spaties in een naam schrijven we die woorden aan elkaar (dus ook niet met een underscore of verbindingsstreepje ertussen) en onderscheiden we de verschillende woorden door ze met een hoofdletter te laten beginnen. Afkortingen worden gezien als één woord.

Voorbeeld:

> *De variabele die het bedrag aan AOW-overbrugging bevat wordt genoemd*
>
> ```
> AowOverbrugging
> ```

Een naam met een verbindings-streepje is geen goede VBA-syntax. Een naam met een under_score is op zich wel correct volgens VBA, maar wanneer gebruik gemaakt wordt van inheritance dan heeft de underscore in een naam soms een bepaalde betekenis. Het is daarom beter om de underscore te vermijden.

Een naam mag met een kleine letter beginnen wanneer dat in gewone formules zo gebruikelijk is (bijvoorbeeld de rekenrente wordt vaak aangeduid met een kleine letter "i", dus is het beter om daarvoor in VBA ook de kleine letter "i" te gebruiken).

Hungarian notation

Hungarian notation houdt in dat de naam van bijvoorbeeld een variabele wordt uitgebreid met een aantal letters om aan te geven wat voor soort variabele het is.

Bijvoorbeeld:

De String variabele waarin iemands adres staat heeft bij de veelgebruikte Hungarian Notation als naam

```
strAdres
```

of

```
sAdres
```

De term “Hungarian notation” is afkomstig van de bedenker ervan, de van oorsprong Hongaarse medewerker van Microsoft genaamd Charles Simonyi, die het introduceerde in zijn artikel Program Identifier Naming Conventions (http://msdn.microsoft.com/en-us/library/aa260976(VS.60).aspx).

Hij gebruikte in zijn artikel voor de soort variabele de term “type”, wat als snel werd geïnterpreteerd als het datatype, zoals ook in het voorbeeld hiervoor (in dat voorbeeld dus string). Binnen Microsoft werden al snel lijsten gemaakt met afkortingen die voor elk datatype moesten worden gebruikt. Er moesten ook letters voor een variabele toegevoegd worden om de scope aan te geven.

Maar binnen een moderne programmeeromgeving is dat allemaal onnodig omdat snel het type van een variabele te zien is. Dat geldt ook voor de programmeeromgeving van VBA:

- In VBA staat het type van de variabele in het Watch (NL:Controle) venster.
- Als de cursor op een variabele staat dan toont Ctrl-i een tooltip met daarin het type.
- Rechts klikken op een variabele laat een menu verschijnen, kies daarin voor Definition (NL:Definitie) en VBA springt naar de declaratie ervan. Nog eens rechts klikken en voor Last Position (NL:Laatste positie) kiezen en VBA springt weer terug. Met de toetscombinaties Shift-F2 respectievelijk Ctrl-Shift-F2 gebeurt dat ook.

De laatste twee mogelijkheden werken ook als de cursor bijvoorbeeld op de naam van een functie of een procedure staat. Ctrl-i op een constante laat de waarde van die constante zien.

Ook het toevoegen van een letter om de scope aan te geven is om dezelfde redenen niet nodig. Het is zelfs vrij onhandig als je bijvoorbeeld een globale variabele lokaal wilt maken. Dan moet de naam van die variabele immers overal vervangen worden.

Zulke extra letters voegen dus voor de programmeur en de lezer van het programma geen informatie toe. Maar het programma wordt er wel onoverzichtelijker door, doordat de naamgeving er langer door wordt en er dus minder bruikbare informatie op het scherm past.

In de afgelopen jaren raakte Hungarian notation uit de mode, mede nadat Microsoft het gebruik ervan afraadde. Het oorspronkelijke idee erachter is echter helemaal zo slecht nog niet (zie bijv. het bekende artikel “Making Wrong Code Look Wrong”, http://www.joelonsoftware.com/articles/Wrong.html) en wordt Applications Hungarian genoemd (in tegenstelling tot het zogenoemde Systems Hungarian dat zich dus op het datatype en de scope richt en intussen aan populariteit heeft ingeboet).

Voorbeeld:

Stel het programma bevat een variabele voor het salaris, “Sal” genaamd. Wie later nog eens naar het programma kijkt om te testen of om te debuggen heeft geen idee of dit een jaarsalaris betreft of een maandsalaris en of het gaat om een full-time salaris of om een parttime salaris. Dat soort informatie is ook in de naam te verwerken zodat dergelijke zaken altijd meteen duidelijk zijn. Met de afkorting “Pt” voor “parttime” en “Jr” voor “jaar” wordt de variabele dan bijvoorbeeld PtJrSal genoemd.

Nog een voorbeeld:

We maken onderscheid tussen een percentage (afgekort als Prc), een perunage (afgekort als Prn) en een factor (afgekort als Fct).

Een percentage is een hoeveelheid "per honderd". Dus bijvoorbeeld 35% staat in een percentage-variabele als 35.

Een perunage is een percentage gedeeld door honderd. Dus bijvoorbeeld 35% staat in een percentage-variabele als 35 maar in een perunage variabele als 0,35.

Een factor is een hoeveelheid die je ergens mee vermenigvuldigt. Dus de 35% staat in een factor-variabele als 1,35. Met andere woorden: 35% van iets is een factor van 1.35 maal iets.

Stel de pensioengrondslag staat in de variabele PtJrPg (parttime pensioengrondslag over een jaar). De pensioenopbouw in dat jaar valt dan te berekenen als

```
PtJrPg * 0.01 * PrcOpbouw
```

of

```
PtJrPg * PrnOpbouw
```

Merk op: een percentage ergens mee vermenigvuldigen betekent meestal dat het percentage eerst met 0.01 moet worden vermenigvuldigd. Bij het visueel controleren van een programma valt het dan meteen op wanneer dit fout gaat, bijvoorbeeld als er ergens

```
PtJrPG * PrcOpbouw
```

staat, of

```
PrJrPg * 0.01 * FctOpbouw
```

Nog een voorbeeld (Ft betekent full-time):

```
PtJrSal = FtJrSal * 0.01 * PrcPt
```

Soms kan het handig zijn om een variabele een Systems Hungarian naam te geven om onderscheid te maken met een andere variabele die anders dezelfde naam zou krijgen.

Bijvoorbeeld:

We willen de inhoud van een textbox uit een venster overnemen in een String variabele. We kunnen daarvoor dezelfde naam gebruiken, alleen krijgt de textbox nog "Txt" ervoor:

```
Dim Verzekeraar As String: Verzekeraar = TxtVerzekeraar.Text
```

Standaard afkortingen

Een lijst met afkortingen die we gebruiken bij naamgeving.

Voluit	Afkorting	Opmerkingen
Full-time	Ft	Op full-time basis, zie ook parttime
Parttime	Pt	Op parttime basis, zie ook full-time
Percentage	Prc	Een percentage is een getal dat nog door 100 gedeeld moet worden om een vermenigvuldigingsfactor te krijgen. Zie ook bij Factor.
Factor	Fct	Een factor is een getal dat ergens mee vermenigvuldigd wordt Perunage = Percentage / 100 Factor = 1 + Perunage Zie ook Percentage en Perunage

Zie voor een complete lijst Appendix 1. Het gebruik van zulke afkortingen zorgt ervoor dat de namen van variabelen, functieaanroepen etc. relatief kort blijven. Zo blijven de regels programmacode relatief kort en blijven programma's overzichtelijker. Het gebruik van de afkortingen went snel, maar voor wie daar nog niet aan gewend is zijn die afkortingen vaak nog niet gebruikt in de stukjes voorbeeldcode in dit boek.

Tekens die op elkaar lijken

Soms is het verschil tussen l, I en 1 makkelijk over het hoofd te zien (de kleine letter L, de hoofdletter i en de één). Ook het verschil tussen 0 en O (de nul en de letter O) wordt gemakkelijk over het hoofd gezien. Houd daarmee rekening bij het geven van namen.

Variabelen

De naam van een variabele begint ook met een hoofdletter.

Wanneer de variabele een private variabele is van een Class, die met een Property Get of Property Set wordt opgevraagd respectievelijk gewijzigd, dan is de naam hetzelfde als die van de Property Get/Set, maar dan met m_ ervoor (dan wordt dus wel een kleine letter gebruikt).

Bijvoorbeeld:

```
Private m_Lft As Long

Public Property Get Lft() As Long
    Lft = m_Lft
End Property
```

Merk op dat VBA zelf schakelt tussen hoofdletters en kleine letters. Als op de ene plek in de code een woord begint met een hoofdletter en elders datzelfde woord met een kleine letter begint dan bestaat het risico dat VBA één van beide woorden aanpast.

De naam van een variabele moet duidelijk maken wat er in die variabele zit. Soms is het echter voldoende om te volstaan met slechts één letter, bijvoorbeeld in het geval van een For-lus die een array doorloopt. Vaak worden dan de variabelen als "J", "K" of "N" als index gebruikt (niet "I" want de kleine letter "i" wordt in de financiële rekenkunde vaak gebruikt voor de rekenrente). In sommige gevallen zijn andere letters ook duidelijk, bijvoorbeeld "R" en "K" voor regels respectievelijk kolommen.

Bijvoorbeeld

Goed is

```
For Regel = 1 To 5
    For Kolom = 1 To 3
        Debug.Print ActiveSheet.Cells(Regel, Kolom).Value
    Next Kolom
Next Regel
```

Net zo goed is

```
For K = 1 To 5
    For J = 1 To 3
        Debug.Print ActiveSheet.Cells(J, K).Value
    Next J
Next K
```

En ook het volgende is goed

```
For R = 1 To 5
    For K = 1 To 3
        Debug.Print ActiveSheet.Cells(R, K).Value
    Next K
Next R
```

Constanten

Voor constantan gelden dezelfde naamgevingsregels als voor variabelen. Op die manier is het gemakkelijk om later van een constante een variabele te maken.

Functies

De naam van een functie beschrijft het resultaat. Het is dus meestal geen werkwoord.

Bijvoorbeeld:

Niet

```
Public Function BerekenDuurInMaanden(ByVal Begindatum As Date, ByVal
Einddatum As Date) As Long
```

Maar wel

```
Public Function DuurInMaanden(ByVal Begindatum As Date, ByVal Einddatum As
Date) As Long
```

Vooral bij functies met een Boolean als uitkomst wordt het resultaat vaak juist wel het beste beschreven door een naam waar een werkwoord in zit.

Bijvoorbeeld:

Niet

```
Public Function Partner(ByVal DeelnemerNr As Long) As Boolean
```

Maar wel

```
Public Function HeeftPartner(ByVal DeelnemerNr As Long) As Boolean
```

Maar dan weer niet

```
Public Function BepaalBurgerlijkeStaat(ByVal DeelnemerNr As Long) As Boolean
```

want uit een dergelijke functienaam wordt niet duidelijk dat het resultaat een Boolean is en wat de betekenis van True of False is.

Vermijd ontkenningen zoals “niet” of “geen” in de naam van Boolean functies. Expressies zoals `Not HeeftGeenPartner()`worden al snel onoverzichtelijk.

Subs

De naam van een procedure beschrijft wat de procedure doet en bevat daarom in tegenstelling tot functies gewoonlijk juist wel een werkwoord.

Modules

Een wat groter VBA-project kan zeer veel modules bevatten. VBA biedt weinig voorzieningen om dat overzichtelijk te houden, alleen dat ze op alfabetische volgorde in de lijst staan.

Om bij elkaar horende modules ook in de lijst bij elkaar te laten staan is het bij een groot project een goed idee om de naam van bepaalde modules te laten beginnen met een letter die aangeeft wat voor soort module het is.

Bijvoorbeeld:

> *De naam van modules met systeemtechnische code laten we beginnen met een T. De module waarin functies en procedures staan die met bestanden te maken hebben noemen we dan TFile.*

Gebruikte afkortingen zijn:

T, Technische modules (d.w.z. systeemtechnische modules)

Class, Modules waarvan de naam bestaat uit “Class” gevolgd door de naam van een class bevatten code die met die class te maken heeft maar niet in de class module zelf past. Een dergelijke module kan bijvoorbeeld gebruikt worden voor procedures om objecten van een bepaalde class aan te maken.

Een andere handige manier van naamgeving is om de naam van modules waar nog veel aan ontwikkeld wordt met “aaa_” te laten beginnen. Dan staan die modules namelijk bovenaan de lijst en zijn ze dus snel terug te vinden.

De naam van een class of een form begint niet met een speciale afkorting of letter om aan te geven wat voor soort het is. Dat gebeurt alleen bij “gewone” modules.

Gebruik voor de namen van modules zo min mogelijk afkortingen.

Labels

In VBA geven labels een plek in het programma aan waar met een Goto-commando of een On Error GoTo-commando naar toe gesprongen kan worden. Sinds de jaren ’70 wordt het gebruik van het Goto-commando afgeraden. On Error Goto is aan de andere kant wel een belangrijke manier om fouten af te handelen in VBA en daarvoor is het gebruik van labels dus wel nodig.

Een dergelijk label is eenvoudig uniek te maken door het de naam te geven van de procedure of functie waar het in staat en daarachter de tekst “ErrorHandler” te zetten.

Voor het overzicht is het beter om verder niets op de regel te zetten die met een label begint. Om nog beter duidelijk te maken dat er een nieuw stukje code begint (het gedeelte dat de foutafhandeling uitvoert) is het goed om vóór de regel met het label een lege regel tussen te voegen. Het label staat aan het begin van een regel, er wordt dus niet ingesprongen vóór een label.

Bijvoorbeeld:

```
Public Function px(ByVal Lft As Long) As Double
    On Error GoTo pxErrorHandler

    px = lx(Lft + 1) / lx(Lft)

    On Error GoTo 0
    Exit Function

pxErrorHandler:
    px = 0
End Function
```

Classes
Ook voor classes gelden de eerder genoemde regels voor het gebruik van hoofdletters en kleine letters. Gebruik voor de namen van classes zo min mogelijk afkortingen.

Enums
Ook voor enums gelden de eerder genoemde regels voor het gebruik van hoofdletters en kleine letters. Gebruik voor de namen van enums zo min mogelijk afkortingen.

Inspringen
Eén niveau inspringen is vier spaties.

De inhoud van een sub of functie of type definitie wordt één niveau ingesprongen. Maar declaraties van lokale variabelen aan het begin van de sub of functie mogen helemaal links staan. Labels (het doel van een Goto-commando of On Error Goto) staan ook helemaal links.

Bijvoorbeeld:

```
Public Function px(ByVal Lft As Long) As Double
    On Error GoTo pxErrorHandler

    px = lx(Lft + 1) / lx(Lft)

    On Error GoTo 0
    Exit Function

pxErrorHandler:
    px = 0
End Function
```

Bij een For .. Next commando staan de For en de Next in dezelfde kolom.

Bij een If .. Else .. End If commando staan de Else en de End If in dezelfde kolom als de If.

Bij een Select Case .. Case .. End Select staan de Select en de Case-commando's en de End Select in dezelfde kolom.

Bij een With .. End With staan de With en de End With in dezelfde kolom.

Bij een Do (While) .. Loop (of Wend) staan ook het begin en het einde in dezelfde kolom.

Bij een Type staat de End Type in dezelfde kolom.

Lengte regels
In het verleden gold wel de richtlijn dat een regel niet langer mocht zijn dan 80 tekens. Meer paste namelijk niet op het scherm. Tegenwoordig past er veel meer op een scherm en hoeft die beperking dus niet meer te gelden. Dat neemt niet weg dat langere regels al snel onoverzichtelijk worden. Wanneer een regel langer is dan 130 tekens dan wordt het zeer zeker tijd om die op te breken. Dan kan door een underscore (_) aan het einde van de regel te zetten en op de volgende regel verder te gaan.

Bijvoorbeeld:

De regel

```
LangeVariabelenaam = LangeFunctienaam(NogEenLangeNaam, EnNogEenLangeNaam)
```

Kun je opsplitsen naar

```
LangeVariabelenaam = LangeFunctienaam(NogEenLangeNaam _
                                     , EnNogEenLangeNaam)
```

Probeer bij het opsplitsen zoveel mogelijk de zaken die bij elkaar horen onder elkaar te krijgen. Laat de vervolgregels beginnen met datgene wat de onderlinge samenhang het meest duidelijk maakt. In het bovenstaande voorbeeld is dat de komma, omdat het gaat om een lijst van parameters. In andere gevallen kan het bijvoorbeeld een operator zijn.

Bijvoorbeeld:

```
XmlUitvoer = XmlUitvoer & "<dit>is een erg lange string</dit>" _
           & "<het>wordt verdeeld over verschillende regels</het>" _
           & "<zodat>het op het scherm past</zodat>"
```

Meerdere instructies op één regel

Het is mogelijk om meerdere commando's op één regel te plaatsen door er een dubbele punt tussen te zetten. Daarmee wordt een regel al snel lang en onoverzichtelijk dus het is niet aan te bevelen.

Declaratie en initialisatie in één regel

Een geval waarbij het wel overzichtelijk kan zijn om meerdere instructies op één regel te zetten is wanneer je een variabele declareert en meteen initialiseert. In veel programmeertalen kan dat trouwens met één commando maar in VBA zijn daar twee instructies voor nodig.

Bijvoorbeeld:

```
Dim OorspronkelijkePensLft As Double: OorspronkelijkePensLft = 65
```

Commentaar

Met commentaar maak je duidelijk wat er gebeurt in het programma. Overbodig commentaar maakt het programma echter alleen maar onoverzichtelijker. Vermijd daarom commentaar waarmee een open deur wordt ingetrapt.

Bijvoorbeeld:

Het commentaar in het stukje programma

```
'Verhoog de leeftijd met 1 jaar.
LftJr = LftJr + 1
```

is overbodig en kan beter weggelaten worden.

De tekst van het commentaar begint meteen na de apostrof. Er worden dus geen spaties tussen gezet.

Een commentaarregel staat boven de regel waarop commentaar wordt geleverd en wordt net zover ingesprongen als die regel. Een dergelijke commentaarregel wordt voorafgegaan door een lege regel.

Als het commentaar te lang wordt voor een regel, ga dan verder op de volgende regel. Ook die regel begint dan met een apostrof, welke net zo ver is ingesprongen als de apostrof van de eerste commentaarregel.

Plaats in elk geval commentaar aan het begin van een procedure met een beschrijving van wat de procedure doet.

Plaats in elk geval commentaar aan het begin van een codemodule met een algemene beschrijving van wat er in de module thuishoort.

Het is ook mogelijk om commentaar te laten beginnen na andere commando's in de regel. Om te voorkomen dat een regel daarmee onoverzichtelijk wordt is het beter om dat te beperken tot

- Declaraties, om uit te leggen van een variabele doet.
- Het afsluitende commando van If..End If, of While .. Wend, of Select .. End Select, of With .. End With of Do .. Loop. Daarbij kan het commentaar gebruikt worden om aan te geven waar het afsluitende commando bij hoort. Wanneer er een lang stuk programma tussen het begin en het einde van bijvoorbeeld een With en een End With staat, dan is het daardoor sneller te zien waar het afsluitende commando bij hoort en waar de tussenliggende code op slaat. Voor procedures is dat niet nodig, want de naam van de procedure waar de cursor in staat is altijd te vinden in de rechterbovenhoek van het venster met de programmacode.
- Stuurcommando's in een blok, zoals Else of Case

Bijvoorbeeld:

```
Public Sub VerwerkDeelnemer(ByVal Id As Long, ByVal Status As Statuscode)
Dim DekkingsgraadVoldoende As Boolean 'Is dekkingsgraad voldoende om
                                      'waardeoverdracht te kunnen plegen?
    ...

    'Kies de manier om de deelnemer te verwerken, afhankelijk van de status.
    Select Case Status

    Case StatusActief, StatusArbeidsongeschikt 'Percentage arbeidsongeschiktheid
                                               'is pas van belang bij het vullen
                                               'van informatie op de nota.
        VoegToeAanIncasso Id
    Case StatusGepensioneerd
        SchrijfBetaalrecord Id
    Case StatusPremievrij 'Eventueel waardeoverdracht uitvoeren
        If HeeftTeVerwerkenWaardeoverdracht(Id) Then
            If DekkingsgraadVoldoende Then
                ...
            End If 'DekkingsgraadVoldoende
        Else 'Heeft geen te verwerken waardeoverdracht
            ...
        End If 'HeeftTeVerwerkenWaardeoverdracht
    ...
    End Select 'Status
End Sub
```

Voor commentaar gebruikten BASIC programmeurs heel lang geleden het REM-statement in plaats van de apostrofe. VBA staat dat nog steeds toe, maar het is beter om consequent voor de apostrofe te kiezen.

Lege regels

Door hier en daar een lege regel tussen te voegen wordt de structuur van een programma duidelijker.

Zet in elk geval een lege regel:

- Tussen de Option Explicit aan het begin van een codemodule en de declaraties die daarna komen.
- Tussen de declaraties aan het begin van een codemodule en de procedures die daarna komen.

- Tussen twee procedures.
- Na de declaraties aan het begin van een procedure en de rest van de code daarin.

Call

Het is mogelijk om een aanroep van een procedure (sub) vooraf te laten gaan door “Call”. In dat geval moeten de eventuele argumenten tussen haakjes staan.

Bijvoorbeeld:

```
Call StartBerekening(Datum)
```

Van deze mogelijkheid wordt hier geen gebruik gemaakt, dus bovenstaande voorbeeld zou worden:

```
StartBerekening Datum
```

For .. Next

De For en Next mogen op één regel gezet worden als er maar één commando tussen wordt gezet. Anders moet de Next op een andere regel staan, net zover ingesprongen als de For. Zet de naam van de variabele die bij de For staat ook achter de Next. Zo wordt nog duidelijker dat ze bij elkaar horen. Dat is vooral handig als er veel regels tussen staan.

Bijvoorbeeld:

```
For Each DeelnemerNr In Testset: Test DeelnemerNr: Next
```

Of

```
For Each DeelnemerNr In Testset
    ...
    ...
Next DeelnemerNr
```

If .. Then

In VBA zijn er twee varianten van If .. Then. Bij één variant staat alles op één regel, zowel de condities als de uit te voeren commando’s.

Bijvoorbeeld:

```
If A Then B Else C
```

Bij de andere variant staan de commando’s niet op dezelfde regel.

Bijvoorbeeld:

```
If A Then
    B
Else
    C
End If
```

Wanneer er sprake is van uitgebreide logische condities of meerdere commando’s dan wordt variant 1 al snel onoverzichtelijk en zou de variant 2 moeten worden gebruikt. Zet in elk geval niet meer dan twee commando’s in een regel bij variant 1.

Ongeacht de gebruikte variant van het If .. Then commando, vergelijk een Boolean variabele niet met True of False.

Bijvoorbeeld

```
If HeeftPartner = True Then
```

Is niet nodig en zou moeten worden

```
If HeeftPartner Then
```

En andersom

```
If HeeftPartner = False Then
```

zou moeten worden

```
If Not HeeftPartner Then
```

Let

Het is mogelijk om een commando waarbij een waarde wordt toegekend aan een variabele vooraf te laten gaan door “Let”.

Bijvoorbeeld:

```
Let X = 5
```

Hier gebeurt dat niet, dus bovenstaande voorbeeld zou worden:

```
X = 5
```

Arrays

Het is vaak wel zo netjes om bij de declaratie van een array meteen de dimensies mee te geven. Merk echter wel op dat het onderstaande dan niet meer mogelijk is.

Bijvoorbeeld:

Het onderstaande werk wel.

```
Public Sub Ar()
Dim X(0 To 1) As Long
Dim Y() As Long

    X(0) = 1
    X(1) = 2
    Y() = X() 'Dit werkt wel omdat van y geen grenzen zijn gedeclareerd
    X(1) = 3
    Debug.Print X(0), Y(0)
    Debug.Print X(1), Y(1)
End Sub
```

Maar als het doel van de kopie gedeclareerd is met vaste grenzen dan werkt het niet:

```
Public Sub Ar()
Dim X(0 To 1) As Long
Dim Y(0 To 1) As Long
```

```
    X(0) = 1
    X(1) = 2
    Y() = X() 'Foutmelding: Kan niet aan de array toekennen
    X(1) = 3
    Debug.Print X(0), Y(0)
    Debug.Print X(1), Y(1)
End Sub
```

Techniek

In deze categorie vallen regels die direct kunnen voorkomen dat er door de programmeur een fout wordt gemaakt.

Ook hierbij zijn uitzonderingen op de regels denkbaar, met name op het gebied van scope.

ByVal/ByRef

Parameters van procedures en functies kunnen ByVal of ByRef gedeclareerd worden.

ByVal wil zeggen dat de waarde gekopieerd wordt als de procedure of functie wordt aangeroepen. Alle wijzigingen die je binnen de procedure of functie aan de waarde van die variabele doet zie je niet meer terug zodra de procedure wordt verlaten. ByRef wil zeggen dat er verwezen wordt naar de variabele die bij de aanroep is gebruikt. Alle wijzigingen die je binnen de procedure of functie aan de waarde van die variabele doet zie je ook terug zodra de procedure wordt verlaten.

Bijvoorbeeld:

```
Public Sub Mix(ByVal V As Long, ByRef R As Long)
    V = V + R
    R = R + V
End Sub



Public Sub Start()
Dim V1 As Long
Dim V2 As Long
    V1 = 5
    V2 = 6
    Mix V1, V2
    Debug.Print V1, V2
End Sub
```

Aanroep van Start heeft dan 5 en 17 als resultaat. De variabele V1 wordt door de procedure Mix niet gewijzigd. Intern wordt de parameter van de procedure Mix wel gewijzigd met het commando V = V + R (in dit geval wordt V dus gewijzigd naar 5 + 6 = 11) maar dat is slechts een kopie die alleen maar bestaat zolang de procedure Mix nog loopt. De variabele V2 wordt door de procedure Mix wel gewijzigd omdat die ByRef is meegegeven. Intern in de procedure Mix heet diezelfde variabele R en in het commando R = R + V wordt R dus gezet op 6 + 11 = 17.

Als een expressie wordt gebruikt om een ByRef parameter te vullen bij de aanroep dan wordt er niets gewijzigd aan de kant van de aanroep.

Bijvoorbeeld:

De bovenstaande procedure Start wordt nu gewijzigd naar

```
Public Sub Start()
Dim V1 As Long
Dim V2 As Long
    V1 = 5
    V2 = 6
    Mix V1, (V2 + 1)
    Debug.Print V1, V2
End Sub
```

Aanroep van Start heeft dan 5 en 6 als resultaat. De variabele V2 is niet gewijzigd.

De eenvoudigste expressie waarbij dat gebeurt is door de parameter tussen haakjes te zetten.

Bijvoorbeeld:

Ook als we de procedure Start wijzigen naar onderstaande blijven de waarden van V1 en V2 staan op 5 en 6:

```
Public Sub Start()
Dim V1 As Long
Dim V2 As Long
    V1 = 5
    V2 = 6
    Mix V1, (V2)
    Debug.Print V1, V2
End Sub
```

Aangezien VBA bij de aanroep zonder functieresultaat normaal gesproken geen haakjes verwacht om de parameters (in tegenstelling tot een aanroep als met een functieresultaat), is het zaak hierbij goed op te passen

Bijvoorbeeld:

Het resultaat van de aanroep van onderstaande procedure Test is "Begin Begin Klaar", dat wil zeggen dat alleen bij de aanroep Dummy = ByRefFun(S3) de parameter gewijzigd wordt. Bij de andere twee aanroepen zorgen de haakjes ervoor dat de aanroep toch ByVal gebeurt:

```
Public Function ByRefFun(ByRef S As String)
    S = "Klaar"
End Function

Public Function ByRefSub(ByRef S As String)
    S = "Klaar"
End Function

Public Sub Test()
Dim Dummy
Dim S1 As String
Dim S2 As String
Dim S3 As String
    S1 = "Begin"
    S2 = "Begin"
    S3 = "Begin"
    ByRefSub (S1)
    ByRefFun (S2)
    Dummy = ByRefFun(S3)
    Debug.Print S1, S2, S3
End Sub
```

Let op: Als er geen keuze wordt gemaakt dan gaat VBA uit van ByRef.

Declareer parameters van procedures altijd ByVal (doe dit dus expliciet omdat de default ByRef is)

Uitzonderingen zijn:

- Wanneer de bedoeling is dat ze gewijzigd worden door de procedure (functies wijzigen een parameter normaal gesproken niet). Dan moet dit duidelijk blijken uit de naam van de

procedure zodat de aanroeper daarin geen fout kan maken.
Of

- Wanneer dit een erg negatieve impact heeft op de snelheid van het programma. Het is bijvoorbeeld handig om strings en arrays niet ByVal te declareren in de parameterlijst omdat die groot kunnen zijn. In het geval van declaratie ByVal wordt een kopie gemaakt van de waarde en het maken van die kopie kan veel tijd kosten. Merk op dat een variabele van het datatype Variant een array kan bevatten. Als je die met ByVal declareert dan maakt VBA van die array een kopie.

Bijvoorbeeld:

Het programma

```
Public Sub Voorbeeld()
Dim GroteArray(0 To 1000000) As Double
    GroteArray(1000) = 100
    ProbeerWaardeTeVeranderen GroteArray
    Debug.Print GroteArray(1000)
End Sub



Public Sub ProbeerWaardeTeVeranderen(ByVal GroteArray As Variant)
    GroteArray(1000) = 33
    Debug.Print GroteArray(1000)
End Sub
```

toont als resultaat de waardes 33 en 100. Opvallend is dat als we geen Variant gebruiken en dus de laatste Sub aanpassen naar

```
Public Sub ProbeerWaardeTeVeranderen(ByVal GroteArray() As Double)
```

dan geeft VBA aan dat de syntax fout is. Wat wel mogelijk is:

```
Public Sub ProbeerWaardeTeVeranderen(ByRef GroteArray() As Double)
```

Dit laatste voorbeeld laat zien dat het niet mogelijk is om arrays ByVal te declareren (je kunt een array dus alleen ByVal meegeven als je er een variant van maakt).

Het is ook niet mogelijk om een user defined type ByVal te declareren. Je kunt die ook niet ByVal meegeven als je er een Variant van maakt zoals bij arrays.

Het wel mogelijk om een object ByVal te declareren maar dat wil niet zeggen dat er een kopie van het hele object gemaakt wordt.

Bijvoorbeeld:

```
    Dim ActieveDeelnemer As Persoon: Set ActieveDeelnemer = New Persoon
    ActieveDeelnemer.Geboortedatum = #1/1/1970#
    ActieveDeelnemer.Bsn = "1234567890"
    ProbeerPersoonTeVeranderen ActieveDeelnemer, "9876543210"
    Debug.Print ActieveDeelnemer.Bsn

Public Sub ProbeerPersoonTeVeranderen(ByVal Deelnemer As Persoon, ByVal
NieuweBsn As String)
    Deelnemer.Bsn = NieuweBsn
    Debug.Print Deelnemer.Bsn
```

```
End Sub
```

Dit stukje code toont gewoon twee keer 9876543210. Maar als we die laatste procedure veranderen naar

```
Public Sub ProbeerPersoonTeVeranderen(ByVal Deelnemer As Persoon, ByVal
NieuweBsn As String)
    Set Deelnemer = New Persoon
    Deelnemer.Bsn = NieuweBsn
    Debug.Print Deelnemer.Bsn
End Sub
```

dan wordt er wel 9876543210 en vervolgens 1234567890 getoond.

Dit komt doordat het object dat als parameter wordt meegegeven eigenlijk alleen een verwijziging is naar het object. Door het object ByVal te declareren in de parameterlijst wordt er een kopie gemaakt van die verwijzing. Maar die kopie verwijst uiteraard naar hetzelfde object, vandaar dat in het eerste voorbeeld het Bsn ook buiten de procedure gewijzigd bleek te zijn. In het tweede voorbeeld werd de kopie van de verwijzing aangepast door `Set Deelnemer = New Persoon`. Daardoor werd er binnen die procedure ook echt naar een ander object verwezen en werd het oorspronkelijke object dus niet veranderd.

Scope

Variabelen worden bij voorkeur gedeclareerd met een zo klein mogelijke scope, dus bij voorkeur lokaal in een procedure, of desnoods private in een codemodule. Globale variabelen worden zo min mogelijk gebruikt.

In procedures worden bij voorkeur alleen de parameters en lokale variabelen gebruikt en aangepast. Als een de waarde van een parameter wordt aangepast dan moet dat expliciet (ByRef) worden gedeclareerd en duidelijk blijken uit bijvoorbeeld de naam van de procedure.

In functies wordt bij voorkeur de waarde van de parameters niet aangepast, alleen gelezen. Verder worden alleen de voor die functie lokale variabelen gebruikt en aangepast.

Option Explicit

Wanneer aan het begin van een codemodule het commando

```
Option Explicit
```

is opgenomen, dan controleert VBA bij het compileren van de codemodule of alle gebruikte variabelen wel gedeclareerd zijn.

Anders loop je het risico dat je per ongeluk een naam verkeerd intypt en er een nog onbekende variabele gebruikt wordt, met een default waarde.

Bijvoorbeeld:

```
Public Sub StatusVeranderen()

Dim WordtArbeidsongeschikt As Boolean

    ...

    WordtArbeidsongeschikt = True

    ...
```

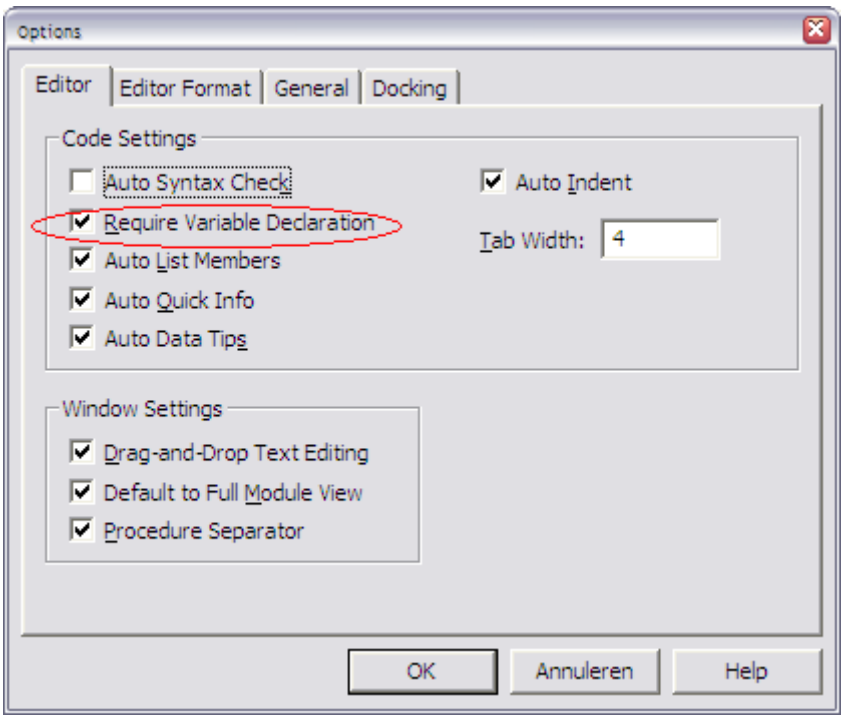
```
If WordArbeidsongeschikt Then

...
```

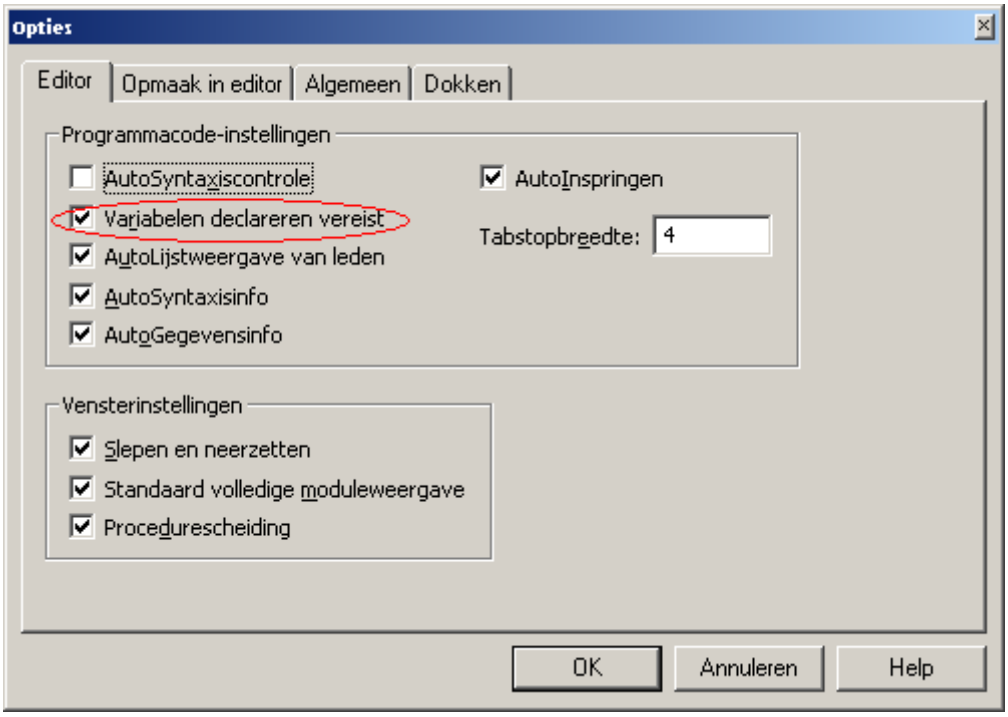
In het `If`-commando wordt de niet-gedeclareerde variabele `WordArbeidsongeschikt` (zonder t) gebruikt. VBA gaat voor deze niet-gedeclareerde variabele uit van de default waarde `False`, in plaats van de waarde `True` van de variabele `WordtArbeidsongeschikt` die de programmeur op die plek had willen gebruiken.

Gebruik daarom altijd `Option Explicit` (dit is een uitzondering op de regel dat er uitzonderingen mogelijk zijn op regels).

Je kunt VBA zo instellen dat het automatisch wordt toegevoegd als je een nieuwe codemodule maakt. Kies daartoe in het menu “Tools” (NL: “Extra”) voor de menuoptie “Options” (NL: “Opties…”). Je krijgt dan een venster waarin je kunt aangeven dat het declareren van variabelen verplicht is.



Nederlandse versie van het venster:



Declaraties van variabelen

Het is mogelijk om in één Dim commando meerdere variabelen achter elkaar te declareren. Let daarbij op dat bij iedere variabele het type gezet moet worden.

Bijvoorbeeld

Na de declaratie

```
Dim x, y, z As Long
```

zijn x en y een Variant en is z een Long. Wat mogelijk door de programmeur bedoeld was is

```
Dim x As Long, y As Long, z As Long
```

Het is daarom beter om bij declaraties van variabelen iedere declaratie op een eigen regel te zetten. Daardoor wordt het ook makkelijker om een kort commentaar er achter te zetten om te beschrijven wat de variabele doet.

Concatenatie van strings

Het is in VBA mogelijk om strings achter elkaar te knopen met een plus “+”. Doe dat niet. VBA interpreteert dat namelijk in sommige gevallen als een gewone optelling.

Bijvoorbeeld:

```
x = 5
y = "6"
z = x + y
Debug.Print z
```

Dit heeft als resultaat 11

Maar:

```
x = 5
y = "6"
```

```
z = x & y
Debug.Print z
```

Dit heeft als resultaat "56"

Gebruik dus altijd "&" als je strings achter elkaar wilt plakken.

Long in plaats van Integer

Een Integer getal loopt in VBA van -32.768 tot 32.767. Dat wil zeggen dat je al heel snel per ongeluk buiten het bereik valt, bijvoorbeeld als je door de rijen van een groot spreadsheet loopt. Het datatype Long is niet langzamer dan Integer (vaak juist sneller!) maar heeft een veel groter bereik. Gebruik dus geen Integers.

Variant

Gebruik zo min mogelijk Variant als datatype. Daarmee is immers pas op run-time duidelijk wat er in zit. De controle die de compiler vooraf doet kan dan niet zien of een variabele juist gebruikt wordt. Daarnaast zijn Variants vaak langzamer (omdat het geheugengebruik groter is).

Soms is het gebruik van Variants onvermijdelijk, bijvoorbeeld in een For .. Each lus, of als het resultaat van een ingebouwde functie een Variant is.

Afronden

De in VBA ingebouwde functie voor afronden werkt anders dan de meeste mensen op school hebben geleerd. VBA werkt met het zogenoemde "Bankers rounding", zie http://en.wikipedia.org/wiki/Rounding#Round_half_to_even

Bijvoorbeeld

```
?Round(2.5)
 2

?Round(3.5)
 4
```

Het is beter om zelf een functie te schrijven voor het afronden, of om `Application.Worksheetfunctions.Round` te gebruiken (maar dat laatste is vrij traag).

Deze afrondmethode gebruikt VBA ook in de functies CInt, CLng, CByte en CCur.

IIf

De functie IIf lijkt een korte manier te zijn om een If commando te schrijven. Maar het grote verschil met een If-commando is dat in de IIf functie zowel het True-gedeelte als het False-gedeelte worden uitgevoerd. Dat kan soms problemen opleveren:

Bijvoorbeeld:

> *Onderstaande geeft een foutmelding dat er gedeeld wordt door 0 als* `lx(Lft) = 0`
>
> ```
> px = IIf(lx(Lft) = 0, 0, lx(Lft + 1)/lx(Lft))
> ```
>
> *Dat komt doordat zowel de expressie* `lx(Lft + 1)/lx(Lft)` *toch wordt berekend, ook al is de conditie* `lx(Lft) = 0` *waar. Maar in dit geval meldt VBA in elk geval nog een fout.*

Ander voorbeeld:

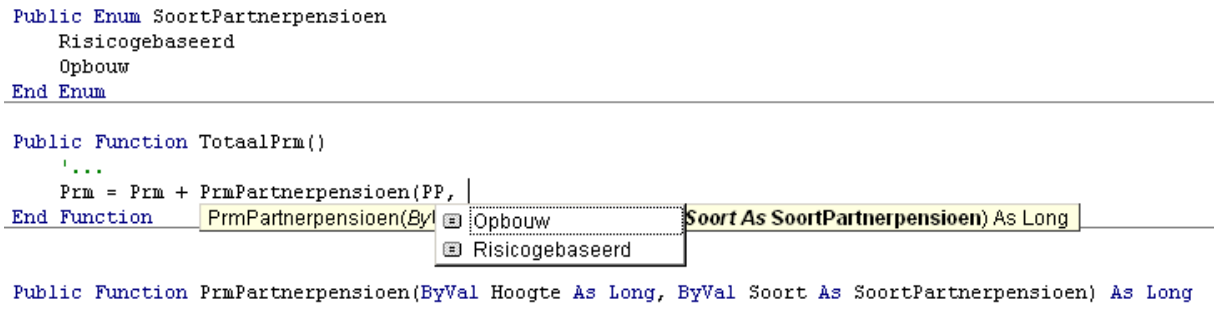
```
Gegevens = IIf(Persoon.IsActief, RegelUitActievenbestand,
RegelUitInactievenbestand)
```

In dit voorbeeld wordt zowel de functie RegelUitActievenbestand als de functie RegelUitInactievenbestand uitgevoerd. Als in die functies wordt bijgehouden waar ze zijn gebleven met inlezen, dan heeft één van beide functies hierna een regel teveel ingelezen. Die wordt in een volgende stap dus onverwacht overgeslagen.

Enums in plaats van constantes

Vaak is het handiger om een enum te gebruiken in plaats van een constante. Bij een enum is duidelijk welke waardes een variabele of parameter kan krijgen. De VBA editor helpt dan ook bij het kiezen van de juiste waardes door code completion. Je krijgt dan namelijk door op Ctrl-Spatie te drukken een lijstje met mogelijke waarden

Bijvoorbeeld:



Dat is handiger dan er een string of een constante voor te gebruiken zoals:

```
Public Function PrmPartnerpensioen(ByVal Hoogte As Long, ByVal Soort As
String) As Long
```

Of

```
Const Risicogebaseerd As Long = 1
Const Opbouw As Long = 2
Public Function PrmPartnerpensioen(ByVal Hoogte As Long _
                                   , ByVal Soort As Long) As Long
```

Performance

In deze categorie vallen regels die zorgen dat de code snel is.

vbNullString

Gebruik de ingebouwde constante vbNullString om een string leeg te maken in plaats van "". De lege string "" kost 6 bytes geheugenruimte en daarmee ook meer tijd om te verwerken dan de ingebouwde constante vbNullString,

Bijvoorbeeld:

```
Uitvoer = ""
```

is niet zo snel als

```
Uitvoer = vbNullString
```

Is een string leeg?

Er zijn verschillende manieren om te kijken of een String leeg is (dus of de lengte nul tekens is).

```
If Uitvoer = "" Then ..

If Uitvoer = vbNullString Then ..

If LenB(Uitvoer) = 0 Then ..
```

De laatste is het snelste. VBA hoeft dan alleen maar de lengte van de String op te halen en met 0 te vergelijken.

$-functies

Een aantal ingebouwde functies van VBA heeft twee versies. Eén waarbij het resultaat een String is en één (de 'normale' versie) waarbij het resultaat een variant is waarin een String zit. De naam van de functie met de String als resultaat is dezelfde als de andere functies, alleen is er een dollarteken ($) achter geplakt.

Het gaat om de functies LCase$, UCase$, Left$, Right$, Mid$, Chr$, ChrW$, LTrim$, RTrim$, Trim$, Space$, String$, Format$, Str$, Hex$, Oct$

Deze werken verder hetzelfde als hun tegenhangers LCase, UCase, Left, Right, Mid, Chr, ChrW, LTrim, RTrim, Trim, Space, String, Format, Str, Hex, Oct.

Het is het efficiëntst om de functies met het dollarteken te gebruiken als het resultaat ervan in een String-variabele wordt gezet en om de functies zonder dollarteken te gebruiken als het resultaat in een Variant-variabele wordt gezet.

As New

Het is mogelijk om de declaratie van een object meteen een nieuw exemplaar aan te laten maken. Dus in plaats van

```
Dim ActieveDeelnemer As Persoon: Set ActieveDeelnemer = New Persoon
```

kun je ook schrijven

```
Dim ActieveDeelnemer As New Persoon
```

Dat is echter af te raden want daardoor wordt het gebruik van dat object langzamer en gebeurt er ook iets onverwachts. Wanneer een object op die manier wordt gedeclareerd wordt er niet alleen op die plek een nieuw exemplaar aangemaakt, maar ook wordt overal waar dat object wordt gebruikt een extra controle toegevoegd of het object al bestaat. Die extra controle maakt het werken met dat object wat langzamer. Maar wanneer het object nog niet bestaat dan wordt het op die plek automatisch aangemaakt.

Bijvoorbeeld:

```
Dim ActieveDeelnemer As New Persoon

....

If ActieveDeelnemer.Lft(#1/1/2010#) >= 65 Then
    Set ActieveDeelnemer = Nothing
End If

If ActieveDeelnemer Is Nothing Then
    ....
```

In bovenstaande stukje programma zou je verwachten dat `ActieveDeelnemer Is Nothing` inderdaad waar is. Dat is immers een paar regels eerder zo toegekend. Echter, ActieveDeelnemer is "As New" gedeclareerd dus `ActieveDeelnemer Is Nothing` is altijd Onwaar. Er wordt namelijk meteen een nieuw exemplaar aangemaakt zodra de variabele wordt gebruikt. Bovenstaande programma wordt door VBA als het ware automatisch vertaald naar:

```
Dim ActieveDeelnemer As Persoon: Set ActieveDeelnemer = New Persoon

....

If ActieveDeelnemer Is Nothing Then Set ActieveDeelnemer = New Persoon
If ActieveDeelnemer.Lft(#1/1/2010#) >= 65 Then
    Set ActieveDeelnemer = Nothing
End If

If ActieveDeelnemer Is Nothing Then Set ActieveDeelnemer = New Persoon
If ActieveDeelnemer Is Nothing Then
    ....
```

With/.

Wanneer een object andere objecten bevat (een property heeft als type weer een object) en die ook weer objecten bevatten etc. dan kun je daar iets uit opvragen of veranderen met de bekende puntjes.

Bijvoorbeeld

```
Partnerpens = Ouderdomspens * Deelnemer.HuidigeRegeling.Partnerpens.Prc / 100
```

Op dat moment moet VBA de hele weg nalopen van het bevattende object (in het voorbeeld is dat Deelnemer) naar het diepste object (in het voorbeeld is dat Partnerpens).

Dus

VBA neemt een pointer naar het Deelnemer-object, haalt daaruit de pointer naar het HuidigeRegeling-object en haalt daaruit vervolgens de pointer naar het Partnerpens-object.

Als je zo meerdere keren hetzelfde object wilt gebruiken dan kost dat onnodig veel tijd. Je kunt ook de pointer naar het diepste object bewaren met het With-commando of door het diepste object in een variabele te zetten. Als je het With-commando gebruikt dan kun je volstaan met alleen een punt te gebruiken voor het eerste object.

Bijvoorbeeld.

```
If Deelnemer.HuidigeRegeling.Partnerpens.Soort = "Perc. per jaar " Then
    PP = Deelnemer.HuidigeRegeling.PG _
        * Deelnemer.HuidigeRegeling.Diensttd _
        * Deelnemer.HuidigeRegeling.Partnerpens.Prc / 100 _
ElseIf Deelnemer.HuidigeRegeling.Partnerpens.Soort = "Perc. van OP " Then
    PP = Ouderdomspens * Deelnemer.HuidigeRegeling.Partnerpens.Prc _
                                / 100
End If
```

Zou je met een With statement kunnen herschrijven naar

```
With Deelnemer.HuidigeRegeling
    If .Partnerpens.Soort = "Perc. per jaar " Then
        PP = .PG _
            * .Diensttd _
            * .Partnerpens.Prc / 100 _
    ElseIf.Partnerpens.Soort = "Perc. van OP " Then
        PP = Ouderdomspens * .Partnerpens.Prc _
                                    / 100
    End If
End With 'Deelnemer.HuidigeRegeling
```

Of, wanneer je gebruik maakt van variabelen

```
Dim Regeling As PensioenRegeling
Dim PPRegeling As RegelingPartnerpensioen
Set Regeling = Deelnemer.HuidigeRegeling
Set PPRegeling = Regeling.Partnerpens
If PPRegeling.Soort = "Perc. per jaar " Then
    PP = Regeling.PG _
        * Regeling.Diensttd _
        * PPRegeling.Prc / 100 _
ElseIf PPRegeling.Soort = "Perc. van OP " Then
    PP = Ouderdomspens * PPRegeling.Prc _
                                / 100
End If
```

And/Or

In een logische expressie van de vorm $(b_1$ `And` $b_2)$ of $(b_1$ `Or` $b_2)$ worden beide operands door VBA uitgerekend, zelfs als op grond van de eerste operand al bepaald kan worden wat de uitkomst van de logische expressie zou worden. Als bijvoorbeeld b_1 `= False` in de expressie $(b_1$ `And` $b_2)$ dan zou b_2 niet uitgerekend hoeven te worden, het resultaat is dan immers altijd `False`. Sommige programmeertalen rekenen b_2 in een dergelijk geval dan ook niet uit, VBA doet dat echter wel. Wanneer b_2 een functie is die relatief veel tijd kost om uit te rekenen dan is dat jammer.

Soms kan het programma iets herschreven worden zodat het toch mogelijk is om onnodige berekening te voorkomen. Zo is het mogelijk om in een If-commando de And te vervangen door nog een If.

Bijvoorbeeld

```
If SnelleConditie() And LangzameConditie() Then …
```

kan worden vervangen door

```
If SnelleConditie() Then If LangzameConditie() Then …
```

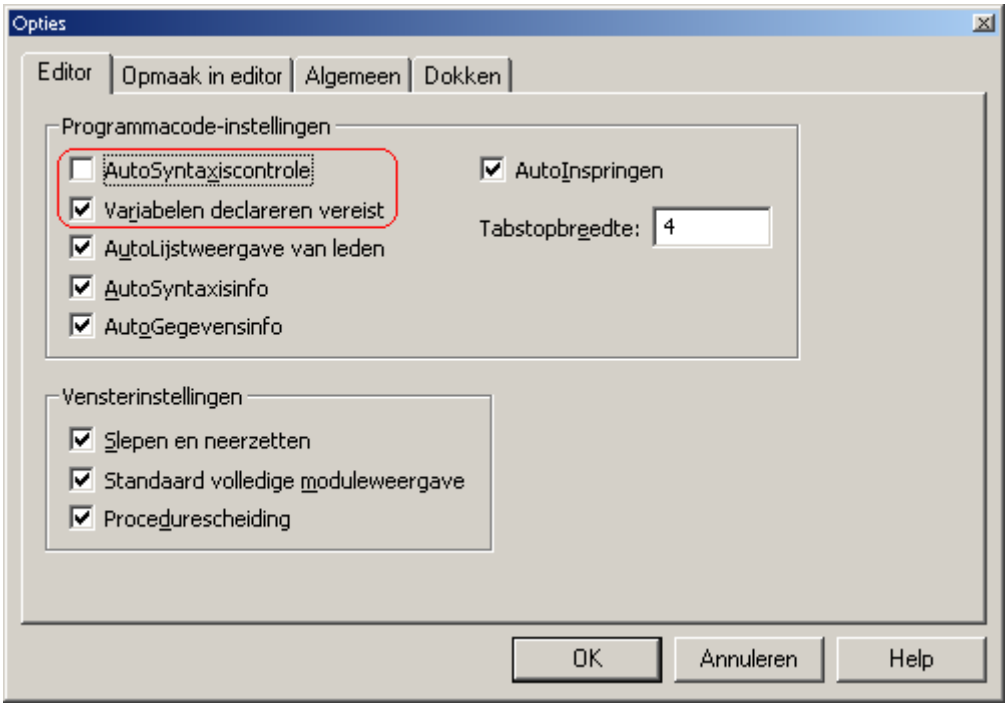
Overige tips

Zichtbaar maken van wat verborgen is in de libraries waar het project references naar heeft.

Druk in de VBA-IDE op F2, je krijgt dan een overzicht van de codemodules, enum's, constanten etc. die in je project zitten of door references (NL: verwijzingen) toegankelijk zijn. Dat is handig als je wilt weten wat er allemaal in een library zit waar je een reference naar hebt gemaakt. Met de knoppen bovenin het venster kun je snel zoeken. Als je rechts klikt in het venster en in het menu dat verschijnt kiest voor "Show Hidden Members" (NL:"Verborgen leden weergeven") dan verschijnen er nog meer codemodules etc.

Instellingen

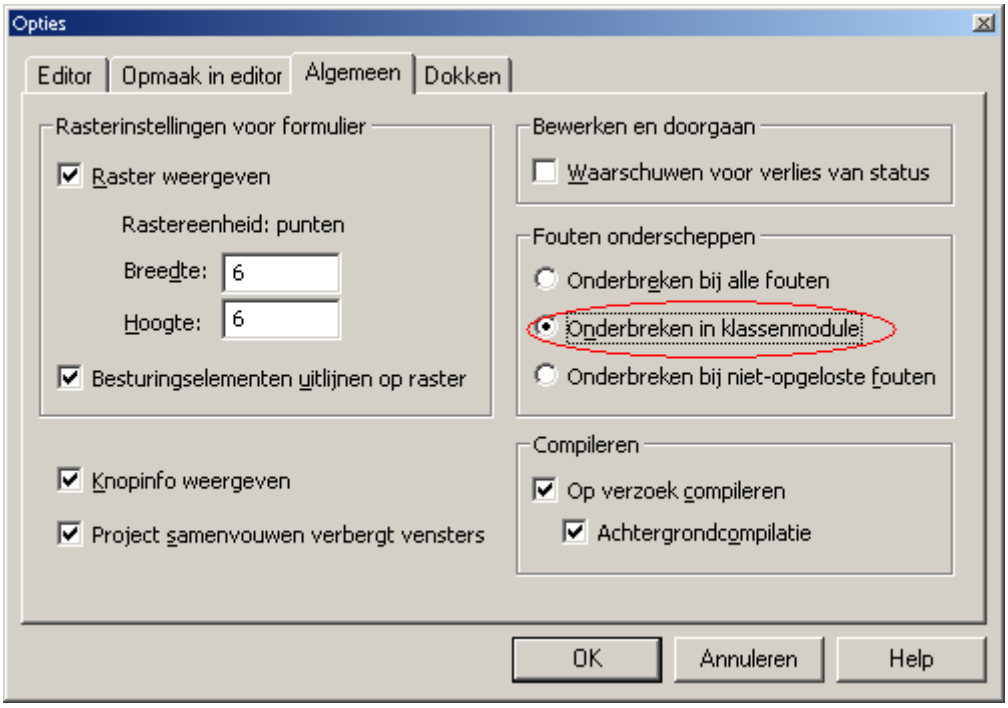
Met het kiezen van menuoptie Tools\Options…" (NL: "Extra\Opties…") verschijnt onderstaande scherm.



Zoals ook al aangegeven bij de coding standards wordt de optie "Variabelen declareren vereist" aangezet. Dan vult de VB-editor bij iedere nieuwe module automatisch "Option Explicit" in.

Het is handig om AutoSyntaxiscontrole uit te zetten. Als een regel een syntaxfout bevat dan maakt de VB-editor die regel automatisch rood, dus een fout valt al snel op. AutoSytaxiscontrole aanzetten zorgt ervoor dat er ook nog een dialoogvenster verschijnt waarin de fout wordt gemeld. Dat venster moet weggeklikt worden voordat je verder kunt gaan. Dat is niet altijd handig, bijvoorbeeld wanneer een regel bewerkt en je iets uit een andere regel wilt kopiëren.

In het tabblad "General" (NL: "Algemeen") is het handig om de optie "Break in Class Module" (NL: "Onderbreken in klassenmodule") aan te zetten.



Daarmee zorg je ervoor dat, wanneer een fout optreedt die niet wordt afgevangen, je kunt beginnen met debuggen op de plek waar de fout optreedt. Default staat dit in VBA op “Onderbreken bij niet-opgeloste fouten” maar dan begin je bij een fout in een klassenmodule niet op de plek waar de fout optreedt maar op de plek waar de code van die klassenmodule werd aangeroepen.

Appendix 1: Afkortingen voor naamgeving

De naam van een variabele kan worden gekozen of samengesteld uit onderstaande afkortingen.

Inhoudelijk

Voluit	Afkorting	Verwacht type	Opmerkingen
Aantal	Ntl	Numeriek (Long)	Bijvoorbeeld aantal beleggingsfondsen dat wordt aangeboden in een beschikbarepremieregeling: NtlFnd
Arbeidsongeschiktheidspensioen	Aop		
Backservice	Bs	Numeriek	
Bedrag	Bdr	Numeriek	
Begin..	Bgn		Bijvoorbeeld in combinatie met “Datum” krijg je BgnDt. Of in combinatie met leeftijd krijg je BgnLft.
Belasting	Blst		
Berekenings..	Ber		Bijvoorbeeld in combinatie met “Datum” krijg je BerDt. Of in combinatie met leeftijd krijg je BerLft.
Bruto	Br		
Brutokoopsom	Bk		
Brutopremie	Bp	Numeriek	
Code	Cd	String	
Coming service	Cs	Numeriek	
Contante waarde	Cw	Numeriek	
Correctie	Cor	Numeriek	
Dag	Dg	Numeriek	
Datum	Dt	Date	
Decimalen	Dc	Numeriek (Long)	Aantal decimalen
Diensttijd	Dtd	Numeriek	Volledige diensttijd van datum in dienst tot pensioendatum. Normaal gesproken in jaren tenzij anders aangegeven, zoals DtdMn is diensttijd in maanden. Zie ook bij “Toekomstige duur …”
Doelvermogen	Dv	Numeriek	
Doorlopende kosten	Dk		
(schade)Driehoek	Dh		Schadedriehoek
Eerste kosten	Ek		
Eind..	End…		Bijvoorbeeld in combinatie met “Datum” krijg je EndDt. Of in combinatie met leeftijd krijg je EndLft. End kun je niet los gebruiken.
Excassokosten	Exc		
Extra	Xtr		

Factor	Fct	Numeriek	Een factor is een getal dat ergens mee vermenigvuldigd wordt. Voor een percentage/perunage gebruik je meestal Prn, d.w.z.: Perunage = Percentage / 100 Factor = 1 + Perunage Zie ook Percentage en Perunage.
Fonds (beleggingsfonds)	Fnd		
Franchise	Fran	Numeriek	
Full-time	Ft	Numeriek	Op full-time basis, zie ook parttime
Garantie	Gar		
Geboorte..	Geb		Bijv. GebDt of GebJr
Huidige	Hdg		
Incassokosten	Inc		
Indexatie (van aanspraken)	Indx		Dus een andere afkorting dan Ix, die bedoelt is voor de index in een array of collection.
Intrest	i	Numeriek	
Jaar	Jr	Numeriek	Ook “in jaren”.
Kapitaal	Kap		
Koopsom	Kps		Maar brutokoopsom Bk, nettokoopsom Np.
Korting	Kng		
Kwartaal	Kw	String	Kwartaal als string in de vorm jjjjQq, dus bijvoorbeeld 2013Q1.
Leeftijd	Lft		
Loon	Sal		
Maand	Mn	Numeriek	Ook “in maanden”.
Naam	Nm	String	
Netto	Net		Let op: bruto/netto koopsom/premie hebben eigen nog kortere afkortingen.
Nettokoopsom	Nk		
Nettopremie	Np		
Nieuwe	Nwe		
Ongehuwdenpensioen	Ohp		
Oude	Oud		
Ouderdomspensioen	Op		
Overbruggingspensioen (tijdelijk)	Top		
Overlevingskans (1 jaar)	px	Numeriek	
Overlevingskans (n jaar)	npx	Numeriek	
Partnerpensioen	Pp		
Parttime	Pt	Numeriek	Op parttime basis, zie ook full-time
Pensioen	Pens	Numeriek	Let op: voor ouderdomspensioen, partnerpensioen, wezenpensioen (tijdelijk) overbruggingspensioen en arbeidsongeschiktheidspensioen zijn eigen afkortingen.
Pensioengrondslag	Pg	Numeriek	

Percentage	Prc	Numeriek	Een percentage is een getal dat nog door 100 gedeeld moet worden om een vermenigvuldigingsfactor te krijgen. Zie ook bij Factor en Perunage
Persoon	Prsn		
Perunage	Prn		Een perunage is een percentage dat al door 100 is gedeeld, dus Perunage = Percentage / 100 Factor = 1 + Perunage Zie ook bij Factor en Percentage.
Polis	Pls		
Premie	Prm		Maar brutopremie Bp, nettopremie Np.
Premievrij	Pv		
Premievrijstelling bij invaliditeit	Pvi		
Provisie	Prov		
Regeling	Rgl		
Rendement	Rend		
Rentestandskorting	Rsk		
Reserve	Res		
Restitutie	Rst		
Risico	Ris		
Risicokapitaal	RisKap		
Salaris	Sal		
Sterftekans (1 jaar)	qx	Numeriek	
Sterftekans (n jaar)	nqx	Numeriek	
Tabel	Tab		
Tarief	Tar		
Termijn/periode	Trm		
Toekomstige duur of diensttijd tot pensioen	Tdp	Numeriek	Diensttijd van berekeningsdatum tot pensioendatum. Normaal gesproken in jaren tenzij anders aangegeven, zoals TdpMn is toekomstige diensttijd in maanden.
Tot en met/laatste	Tm		
Totaal	Ttl		
Uitkering	Uitk		
Vanaf/eerste	Van		
Verzekering	Verz		
Volgende	Vlg		
Voorziening	Voorz		
Vorige	Vrg		
Wezenpensioen	Wzp		

Techisch

Voluit	Afkorting	Verwacht type	Opmerkingen
Bestand	File		Dus bestandsnaam is FileNm
Directory	Dir		Dus directorynaam is DirNm. Alleen in combinatie met andere dingen te gebruiken, want “Dir” is losgeschreven ook een VBA-functie.
File	File		Dus filenaam is FileNm
Formaat	Fmt	String	
Index (in een array)	Ix	Numeriek (Long)	Dus een andere afkorting dan Indx, die bedoeld is voor indexatie van aanspraken
Initialisatie	Init		De procedure waarmee een module geinitialiseerd wordt heet Init. Je kunt zo meerdere procedures genaamd Init krijgen, die houd je uit elkaar door de naam van de module ervoor te zetten met een punt ertussen. Dus Modulenaam.Init()
Kolom	Kol	Long	
Naam	Nm	String	
Printer	Prt		
Regel	Rgl	Long	
Rij	Rij	Long	
Workbook	Wb	Workbook	Een geopend Excelbestand (ook wel werkmap genoemd) heet in het Engels een workbook.
Worksheet	Ws	Worksheet	Een werkblad (ook wel tabblad genoemd) in een workbook, heet in het Engels een worksheet.