Spelen

met

Excel

VBA

INHOUDSOPGAVE SPELEN MET EXCEL VBA

1 Inleiding

1.1 Motto: wat kan met Excel doen we niet met Excel VBA.

Regelmatig ontmoet ik bij een cursus **Excel VBA** mensen die op zoek zijn naar bepaalde functionaliteit en verwachten deze met **VBA** te moeten toevoegen. Vaak blijkt dan dat de bedoelde functionaliteit ook uitstekend zonder **VBA** te realiseren is.

We geven een enkel voorbeeld om dit te verduidelijken.

Bij het gebruik van een draaitabel kunnen problemen ontstaan doordat het opgegeven bereik verandert. We moeten dit bereik voor de draaitabel dan handmatig aanpassen. Een enkeling kiest ervoor om het bereik bij voorbaat veel groter te maken dan nodig en de lege cellen in de draaitabel te onderdrukken.

Ligt het dan niet voor de hand dit met **VBA** op te lossen? Nee. We kunnen dit probleem in **Excel** zelf oplossen met behulp van een zogenaamde naam gebaseerd op een flexibel bereik. We definiλren bijvoorbeeld de naam DATABASE en baseren die op de formule:

```
=VERSCHUIVING($A$1;0;0;AANTALARG($A:$A);AANTALARG($1:$1))
```

De functie verschuiving laat in dit geval het bereik beginnen in A1 (met 0 naar rechts en 0 naar beneden) en vervolgens worden het aantal gevulde rijen en kolommen geteld. Voordeel is nu dat het bereik zich automatisch aanpast als er rijen en/of kolommen bijkomen!

We moeten nu wel nog de draaitabel vernieuwen en daarvoor zouden we wel **VBA** kunnen gebruiken. Zoiets:

```
ActiveSheet.PivotTables("Draaitabel1").PivotCache.Refresh
```

De kunst is naar mijn smaak **VBA** zo in te zetten dat het aanvullend is op de reeds aanwezige functionaliteit van **Excel**.

1.2 Macrorecorder en VBA

Een tweede punt is de **macrorecorder**. Veel mensen komen binnen Excel in aanraking met VBA door de **macrorecorder**. De VBA code die we dan zien schrikt eerder af dan dat ze aanmoedigt door te gaan.

Volgens mij zou de macrorecorder ook niet het uitgangspunt moeten zijn om **VBA** te leren. De **macrorecorder** is vooral nuttig om er achter te komen hoe bepaalde objecten in **Excel** heten. We nemen bijvoorbeeld op dat we op een bepaald blad op een bepaalde cel klikken en kijken naar de code:

```
Sub Macro1()
   Sheets("Blad2").Select
   Range("C4").Select
End Sub
```

De recorder heeft evenwel een aantal grote nadelen. Allereerst produceert deze vaak nogal uitgebreide deels vaak overbodige code. Bovenstaande code kunnen we sowieso al inkorten:

```
Sub Macro1()
   Sheets("Blad2").Range("C4").Select
End Sub
```

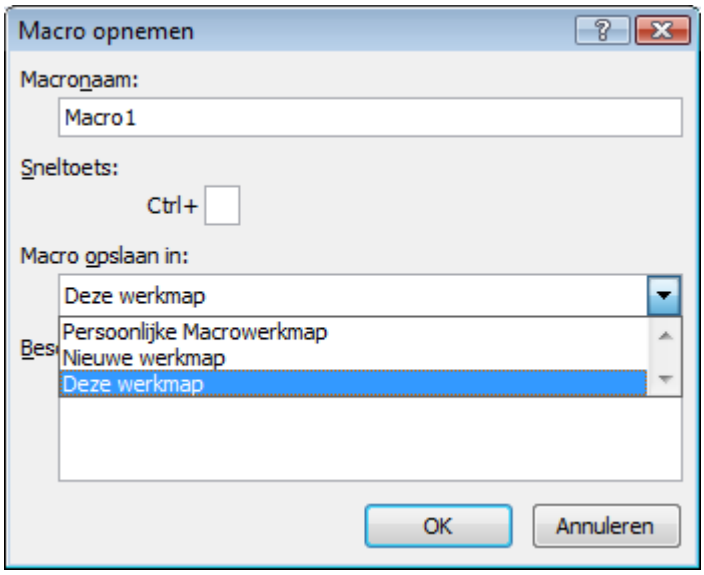
Of anders formuleren:

```
Sub Macro2()
   Sheets("Blad2").Cells(4, 3).Activate
End Sub
```

In de tweede plaats kan de **macrorecorder** niet procedureel programmeren. We zullen in de code dus geen **do … while** constructies of **if … endif** aantreffen. Daarmee is de functionaliteit tamelijk beperkt.

Ik gebruik de **macrorecorder** dan ook niet als uitgangspunt voor deze cursus, maar hooguit als geheugensteuntje of hulpje bij het vinden van de naam van een **Excel** object. Even opnemen, code bekijken en hopelijk hebbes!

Wel kunnen we de macrorecorder ook gebruiken om even snel de **Persoonlijke werkmap** op te halen. Als we **VBA** code in deze map plaatsen en opslaan, is deze code in elke nieuwe werkmap te gebruiken. Bestaat deze persoonlijke werkmap nog niet, dan moeten we deze maken door eenmalig zelf een macro op te nemen en deze op te slaan in de persoonlijke werkmap.



1.3 Organische aanpak

In deze handleiding ga ik niet uitvoerig alle onderdelen van **VBA** en de **VBA editor** ййn voor ййn bespreken. Daarvoor verwijs ik naar de talloze boeken of het internet. Vaak zijn deze boeken ook nog gratis als **PDF** te downloaden.

Ik pak een voorbeeld, werk dat in collegevorm uit en bespreek en passant de dingen die voorbij komen. Zo wordt naar ik hoop geleidelijk aan de onderlinge samenhang duidelijk.

2 Objecten in Excel

2.1 Inleiding: Excel VBA is object georiʌnteerd

Excel is opgebouwd uit objecten. We zien bijvoorbeeld dat een werkblad bestaat uit heel veel cellen. Elk van deze cellen is een object. Tezamen vormen ze een verzameling van identieke objecten (**collection**). Een cel kan weer deel uitmaken van een bereik: bijvoorbeeld een groep van cellen of zelfs een hele kolom of rij. Een bereik op zijn beurt maakt weer deel uit van een werkblad en een werkblad van een werkmap. Kortom, de objecten staan in een hierarchische verhouding tot elkaar (**hierarchy**).

Eenvoudig gezegd bestaat elk object in **Excel** uit bepaalde gegevens en de programmatuur (**VBA**) die gebruikt wordt om die gegevens te verwerken en bewerken. Zo kunnen we bijvoorbeeld van een object werkblad de naam (**property** of eigenschap) opvragen of aanpassen. Een werkblad maakt deel uit van de verzameling werkbladen. We moeten dus in de programmatuur aangeven met welk lid van de verzameling we iets willen gaan doen.

Met de volgende opdracht passen we de eigenschap naam aan van het eerste werkblad uit een werkmap.

```
Worksheets(1).name = "hessel"
```

Elk object in **Excel** heeft veelal verschillende eigenschappen. We zien hier ook al het begrip verzameling terug: Worksheet**s**. In **VBA** wordt Engels gebruikt en alle meervouden eindigen op een **s**. Verzamelingen zijn dus makkelijk te herkennen.

We hebben het **VBA** voorbeeld hier even kort door de bocht geformuleerd. We zijn voorbijgegaan aan de hierarchie. We hadden ook de volgende, volledige opdracht kunnen geven:

```
Windows.Application.Workbooks("Map1.xlsm").Worksheets(1).name = "hessel"
```

We dalen dan helemaal hierarchisch af via **Windows** naar **Application** (lees: **Excel**) via de naam van de werkmap naar het werkblad. **Excel** zal vaak uit de context begrijpen welke hierarchie bedoeld wordt. De lange formulering is dan niet nodig.

We hebben nu al de begrippen verzameling (**collection**), hierarchie (**hierarchy**) en eigenschap (**property**) de revu zien passeren. We gaan verder met gebeurtenis (**event**).

Niet elk object heeft gebeurtenissen (**events**) maar een werkblad heeft dat wel. We kunnen de inhoud ergens op een werkblad aanpassen en dat werkblad ondergaat dan de gebeurtenis verandering (**change**). In code:

```
Worksheet_change()
```

Op gebeurtenissen (**events**) kunnen we laten reageren door een zelf geschreven **procedure** of **subroutine** in **VBA**. Een voorbeeld:

```
Sub Worksheet_change(ByVal Target As Range)
  'een msgbox stuurt een bericht naar het scherm
  MsgBox "afblijven"
  'MsgBox is een functie
End Sub
```

Zodra we nu iets veranderen in het betreffende werkblad wordt de gebeurtenis geactiveerd en wordt de boodschap getoond.

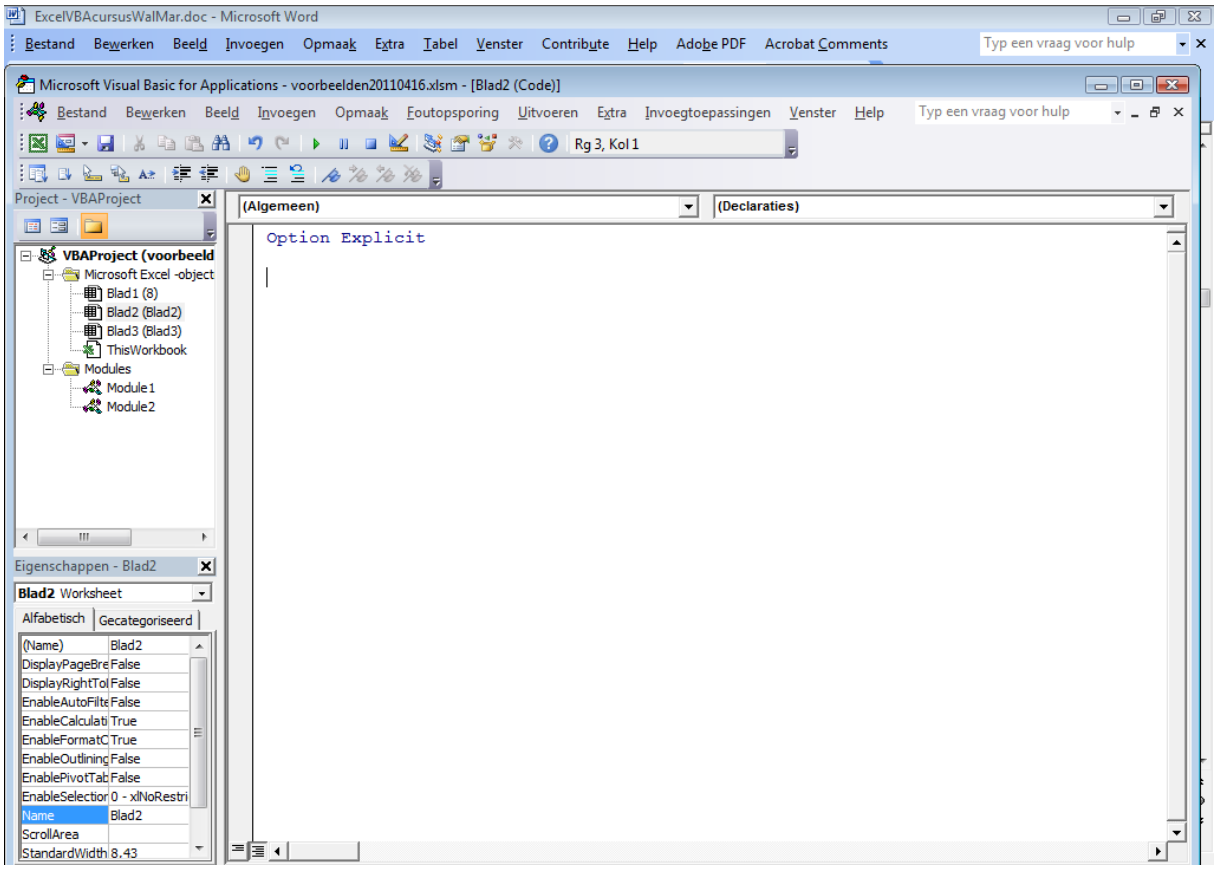
Om te voorkomen dat het allemaal te abstract wordt, gaan we het concreet toepassen.

- Maak een nieuwe werkmap en sla die op als **voorbeelden001.**

Vanaf versie 2007 moeten we dat doen als een **Excel map met macro's (*.xlsm)**; anders wordt de code niet opgeslagen en zullen de macro's later niet uitvoerbaar zijn.

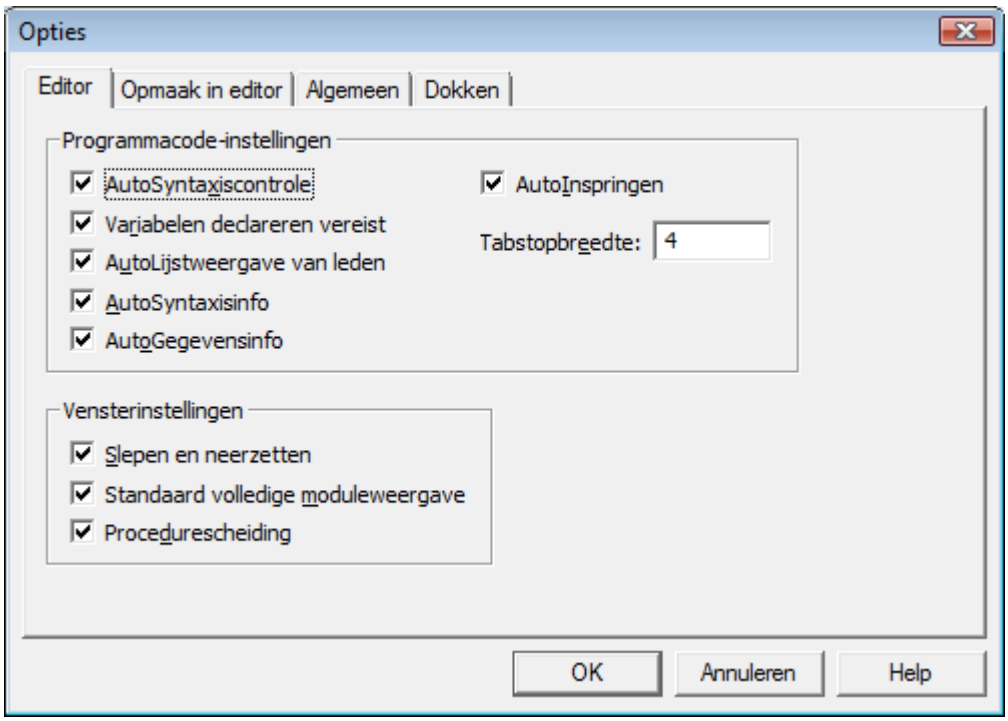
- Met de toetscombinatie **ALT F11** gaan we naar de **VBA editor.**

Dit kan ook via het menu of vanaf versie 2007 via de tab **Ontwikkelaars** die we wel eerst aan moeten zetten via **Bestand ⇒ opties**)



- Via de projectverkenner activeren we de code achter **Blad2**.

Zorg er voor dat **Option Explicit** (waarmee we afdwingen dat variabelen gedeclareerd moeten worden) aanstaat; via **Extra ⇒ opties**



De eerste keer zullen we dan even zelf bovenaan **Option Explicit** moeten typen.

- Onder **Option Explicit** typen we dan de **VBA** code:

```
Sub Worksheet_change(ByVal Target As Range)
  MsgBox "afblijven"
End Sub
```

Het inspringen van de tweede regel is niet verplicht. We doen dat omwille van de overzichtelijkheid.

- We slaan de code op via **CTRL S**.
- Terug in ons **Excel** bestand kiezen we blad2.

Nu zal de gemaakte code reageren op elke verandering die we in blad2 aanbrengen; probeer maar eens!

We gaan verder met ons experiment. De code tussen haakjes ***ByVal Target As Range*** stelt ons in staat de waarde die we invullen, op te pikken en te gebruiken. We geven een voorbeeld.

Objecten hebben zoals eerder gezegd ook eigenschappen (**properties**): het object **Worksheets("blad2")** heeft bijvoorbeeld een naam:

```
Worksheets("blad2").name
```

Deze naam kunnen we met **VBA** veranderen.

- We gaan weer met **ALT F11** naar de **VBA editor.**
- Onze eerdere code passen we aan naar:

```
Sub Worksheet_change(ByVal Target As Range)
  'de naam van het werkblad wordt gelijk aan de inhoud van de cel
  'die we aanklikken
  Worksheets("blad2").name = Target.value
  'hier passen we een property aan
End Sub
```

- Terug in **Excel** proberen we dit uit: als we wat intypen zal dit de nieuwe naam worden van **Blad2**.

Proberen we het nog eens, dan hebben we gelijk een probleem: de code geeft een foutmelding voor **Worksheets("blad2")**. Logisch: het herkent de naam van dit object niet meer. Om dit probleem te verhelpen moeten we snappen dat objecten deel uit maken van **Collections** (collecties of verzamelingen). Een **Worksheet** is deel van de collectie **Worksheets**. Leden van een collectie kunnen we aanroepen met hun naam of met hun nummer. Als we in onze code de naam **blad2** vervangen door het cijfer **2** (of het tweede lid uit de collectie) hebben we het probleem dus opgelost.

```
Sub Worksheet_change(ByVal Target As Range)
  Worksheets(2).name = Target.value
End Sub
```

We kunnen **Excel** ook laten reageren op de inhoud en deze bijvoorbeeld controleren op juiste lengte. De inhoud kunnen we opvragen via:

```
Target.Value
```

Willen we testen of de waarde wel de juiste lengte 10 heeft, dan komt de code er zo uit te zien.

```
Private Sub Worksheet_Change(ByVal Target As Range)
   'bepaal het volgnummer van de kolom van het adres
   If Target.Column = 8 Then
        'de functie len meet de lengte
     If Len(Target.Value) <> 10 Then
           MsgBox "niet correcte lengte"
           'ga terug naar hetzelfde adres
           Range(Target.Address).Select
           'hier zien we de methode Select van het object Range
        End If
   End If
End Sub
```

In het laatste voorbeeld zien we voor het eerste een methode (**method**), de methode **select** van het object **range**. Objecten kunnen ook iets uitvoeren of doen. Zo kan het object **Range** een selectie maken. Een lid van de collectie **Workbooks** kan bijvoorbeeld de actie opslaan uitvoeren:

```
Workbooks("map1.xlsx").Save
```

Of de actie sluiten:

```
Workbooks("map1.xlsx").Close
```

Zo kunnen verschillende objecten hun eigen methoden hebben.

Het eerder genoemde

```
ActiveSheet.PivotTables("Draaitabel1").PivotCache.Refresh
```

was ook een mooi voorbeeld van een methode: **refresh**. Ook een mooir voorbeeld van hierarchie en collectie trouwens.

We hebben zo gezien dat een object eigenschappen (**properties**), gebeurtenissen (**events**) en methodes (**methods**) kan hebben. Een object maakt altijd deel uit van een collectie (**collection**) en staat in een hierarchische verhouding tot andere objecten (**hierarchy**).

2.2 Een uitgebreid voorbeeld

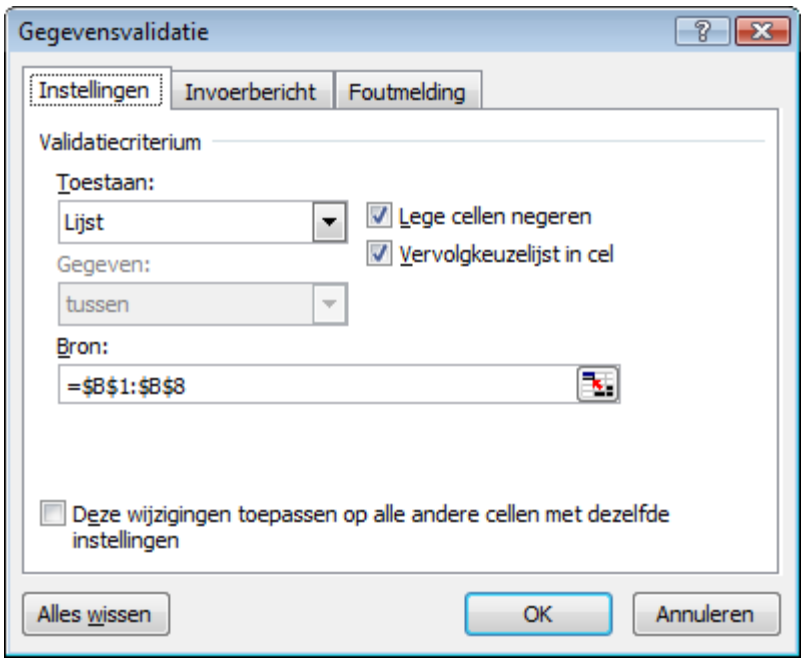
We gaan nu op **Blad1** van **voorbeelden001.xlsm** het volgende voorbeeld maken:

Dit	1	2	3	4	5	6	7
zijn	8	9	10	11	12	13	14
de	15	16	17	18	19	20	21
kleurnummmers	22	23	24	25	26	27	28
tussen	29	30	31	32	33	34	35
1	36	37	38	39	40	41	42
en	43	44	45	46	47	48	49
56	50	51	52	53	54	55	56

36

We gaan bovenstaande voorbeeld namaken.

- We typen De tekst van **A1** tot en met **A8** en de getallen 1 tot en met 56 van **B1** tot en met **H8.**
- In cel **B11** passen we gegevensvalidatie toe met een verwijzing naar **B1** tot en met **B8**:



De bedoeling is nu dat het bereik **A1:A8** steeds de kleur krijgt die hoort bij de gekozen kleurcode. We hebben hier met twee **Events** (gebeurtenissen) te maken: met **SelectionChange** als we door de getallen lopen en met **Change** als we via de lijstpijl in cel **B11** een ander getal kiezen. Beide gebeurtenissen gaan we programmeren.

We moeten ook weten hoe we met **VBA** een bereik kunnen selecteren. In dit geval kan dat met de code:

```
Worksheets(1).Range("A1:A8").Select
```

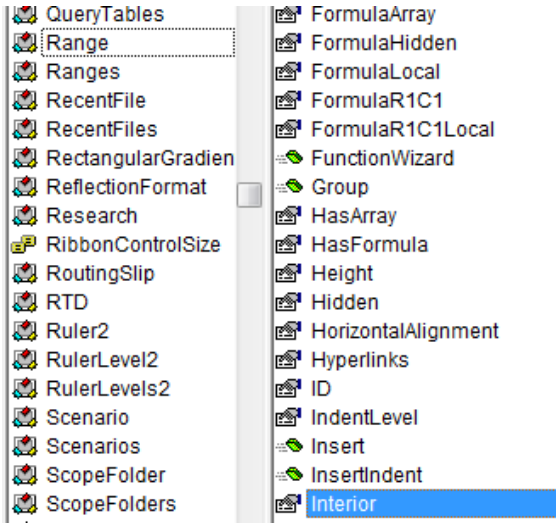
Het object **Range("A1:A8")** is hier een subobject van **Worksheets(1)**. **Select** is hier de methode (**method**) die bij dit object hoort: het zorgt er voor dat het bedoelde bereik geselecteerd wordt. Test deze code maar eens!

```
Sub test()
  Worksheets(1).Range("A1:A8").Select
End Sub
```

We geven nog wat voorbeelden van code om een bereik te selecteren.

Range("A1").Select	Selecteert A1
Range("A1:B4"). Select	Selecteert A1 t/m B4
Range("A1,B1,C1")	Selecteert A1, B1, C1
Range("A1:B4,C1:C4").Select	Selecteert A1 t/ B4 en C1 t/m C4
Range(Range("A1"), Range("D4")).Select	Selecteert A1 t/m D4
Range(Activecell,ActiveCell.OffSet(5,2)).Select	Selecteeert vanaf de actieve cell 6 rijen naar beneden en 3 kolommen naar rechts
Range("naam").Select	Selecteert het bereik met deze naam
Cells.Select	Selecteert alles
Cells(5,3).Select	Selecteert C5

Het object **Range** heeft onder meer een eigenschap **Interior,** het binnenste van de **Range**. We kunnen dit bekijken via het **Objectenoverzicht** (op te roepen vanuit de **VBA** editor met de knop met die naam of via **F2**):



Interior is zelf ook weer een object met als eigenschap onder meer **Colorindex**. Deze eigenschap kan waarden krijgen tussen de 0 en 56. Bij andere waarden krijgen we een foutmelding.

We kunnen nu de **SelectionChange Event** als volgt programmeren:

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
  'de volgende regel gebruiken we zodat eventuele fouten genegeerd worden
   On Error Resume Next
    'interior.colorindex bepaalt de kleur van het binnenste van de cel
    Worksheets(1).Range("a1:a8").Interior.ColorIndex = Target.Value
End Sub
```

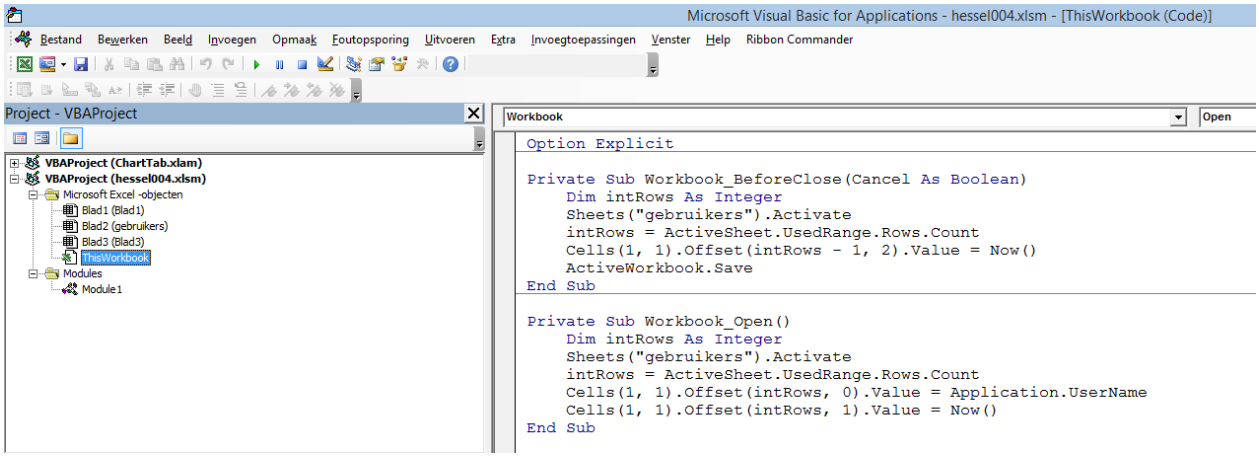
En de **Event Change** als:

```
Private Sub Worksheet_change(ByVal Target As Range)
    On Error Resume Next
    Worksheets(1).Range("a1:a8").Interior.ColorIndex = Target.Value
End Sub
```

We hebben in beide gevallen de code ***On Error Resume Next*** toegevoegd. Als we een cel kiezen zonder getal of met bijvoorbeeld tekst, dan zou dit een foutmelding opleveren. Met genoemde code, dwingen we de code verder te gaan. Slordig is het wel. Mooier zou het zijn de fout met een **IF THEN** af te vangen. Maar dat gaan we later doen.

2.3 Nog een voorbeeld: gebruikers en ingelogde tijd bijhouden

Beide stukken code hebben betrekking op gebeurtenissen van het object **Workbook**. De code hoort daarom achter **Workbook** te staan:



Voor deze code is het van belang dat we een blad met de naam **Gebruikers** gemaakt hebben.

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
   Dim intRows As Integer
   Sheets("gebruikers").Activate
   intRows = ActiveSheet.UsedRange.Rows.Count
   Cells(1, 1).Offset(intRows - 1, 2).Value = Now()
   ActiveWorkbook.Save
End Sub
```

```
Private Sub Workbook_Open()
   Dim intRows As Integer
   Sheets("gebruikers").Activate
   intRows = ActiveSheet.UsedRange.Rows.Count
   Cells(1, 1).Offset(intRows, 0).Value = Application.UserName
   Cells(1, 1).Offset(intRows, 1).Value = Now()
End Sub
```

3 Procedureel programmeren

3.1 Inleiding

In hoofdstuk 2 hebben we een eerste kennismaking gehad met objecten en hoe we deze kunnen benaderen met **VBA**.

Nu gaan we ook het procedurele programmeren van **Excel VBA** bekijken. We zullen zien dat het combineren van procedureel programmeren en **Excel** objecten zeer krachtige mogelijkheden biedt.

Procedureel programmeren betekent eigenlijk niets anders dan stap voor stap en opeenvolgend instructies opgeven.

3.2 Private, Public en Static

Een **VBA Private Sub** kan alleen vanuit dezelfde **Module** worden **aangeroepen** waar deze zich bevindt. Een **Public Sub** in de objecten, **ThisWorkbook**, **ThisDocument**, **Sheet1**, etc. kan niet elders vanuit het project worden aangeroepen.

Behalve **Private** en **Public** mogen we een **Subroutine** ook als **Static** declareren. In dat geval zullen alle gedeclareerde variabelen binnen die **Sub** hun waarden behouden als de **Sub** verlaten wordt in tegenstelling tot de normale gang van zaken: de waarde blijft niet behouden.

Variabelen die binnen een **Sub** gebruikt worden zijn altijd **Local** tenzij ze expliciet declareerd worden op het niveau van de **Module**. **Module** variabelen zijn variabelen die gedeclareerd worden voor de eerste Sub en ze behouden hun waarde door de hele **Module** heen als ze met **Dim** gedeclareerd zijn.

Als we evenwel zo'n variabele met **Public** declareren, kan deze overal in het project gebruikt worden met behoud van waarde.

3.3 Variabelen

Variabelen zijn ***dataholders*** die kunnen worden gebruikt om informatie te bewaren. Zoals de naam het aangeeft, kan deze inhoud tijdens runtime wijzigen (of variëren).

We gaan ervan uit dat voordat een programma een variabele kan gebruiken, het programma het bestaan van deze variabele dient te kennen. We kunnen het bestaan van die variabele aangeven door die variabele te gaan declareren.

Bij de declaratie van een variabele geven we ten minste de volgende zaken op:

- Identifier (lees: naam) van de variabele.
- Datatype van de variabele, opgegeven aan de hand van een type specifier in de vorm van een **As** clausule, keyword **As** gevolgd door de naam van het datatype.

Het keyword **Dim** is een afkorting voor ***Declare In Memory***.

Een tweetal veel gebruikte datatypen zijn:

- **String**: gebruikt voor alfanumerieke data, teksten dus.
- **Integer**: gebruikt voor numeriek data, meer specifiek gehele getallen.

Het datatype geeft aan welke informatie de betreffende variabele kan bevatten, en welke acties we met deze of op deze variabele kunnen toepassen. Twee numerieke variabelen bijvoorbeeld, kunnen getallen bevatten die we kunnen optellen, vermenigvuldigen, delen enzovoorts.

Stel we willen een variable van het type **String** creëren dat namen moet bevatten. We kunnen dan als naam gebruiken:

```
strNaam
```

De eerste drie letters gebruiken we hier om aan te geven dit een variabele van het type **String** is. Dit principe noemen we de **Hongaarse notatie**. De Hongaarse notatie is een afspraak voor het geven van namen bij het programmeren van computers, waarbij het eerste deel van de naam van een object een verwijzing is naar het datatype. De rest van de naam beginnen we met een hoofdletter. We noemen dit principe **Camelizing**. De engelse **Camel** telt trouwens maar ййп bult.

3.4 Beheersen van de control flow

In een procedurele programmeertaal zouden de instructies elkaar alleen maar op kunnen volgen als we de **control flow** niet kunnen beheren: instructies zouden dan alleen strikt opeenvolgend worden uitgevoerd. Voor het beheren van de **control flow** kent **VBA** verschillende elementen. We gaan er een aantal bekijken:

- **IF THEN ELSE**
- **FOR EACH NEXT**
- **FOR NEXT**
- **SELECT CASE**
- **DO UNTIL**

3.4.1 IF THEN ELSE

Laten we eens beginnen met **IF THEN ELSE**. In het vorige hoofdstuk hadden we de code:

```
Private Sub Worksheet_change(ByVal Target As Range)
  On Error Resume Next
  Worksheets(1).Range("a1:a8").Interior.ColorIndex = Target.Value
End Sub
```

We moesten daar **On Error Resume Next** gebruiken om foutmeldingen te vermijden. Wel merkten we toen op dat dit slordig is. We gaan het nu oplossen.

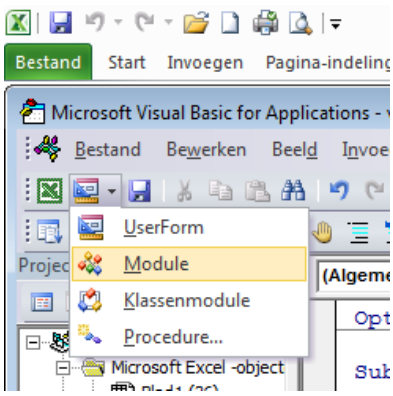
We kunnen testen of de **Target**.**Value** wel een waarde is en of het wel een waarde is tussen de 0 en 56:

```
Private Sub Worksheet_change(ByVal Target As Range)
  If Target.Value >= 0 And Target.Value <= 56 Then
      Worksheets(1).Range("a1:a8").Interior.ColorIndex = Target.Value
  End If
End Sub
```

En zie daar, het werkt.

3.4.2 FOR EACH NEXT

Dit element gebruiken we voor het benaderen van collecties (collections). Stel we willen voor alle werkbladen in een werkmap de cel **A20** vullen met het woord **gelukt**. We gaan dan met **ALT F11** naar de **VBA editor** en maken een nieuwe module aan:



Deze module geven we de naam **VBAvoorbeelden**. Het eerste voorbeeld wordt onderstaande code:

```
Sub vullenA20()
   Dim shtX As Worksheet
   'we doorlopen elke worksheet uit de verzameling worksheets
   For Each shtX In Worksheets
       shtX.Range("A20").Value = "gelukt"
   Next
End Sub
```

We hebben hier een variabele **shtX** gedefinieerd. De **X** schrijven we met een hoofdletter. Dit principe noemen we zoals eerder vermeld **camelizing**. Voordeel van deze methode: gebruiken we de variabele elders weer en schrijven we deze met een kleine letter, dan zien we dat deze onmiddellijk aangepast wordt. Dit betekent dat we deze eerder gedefinieerd hebben!

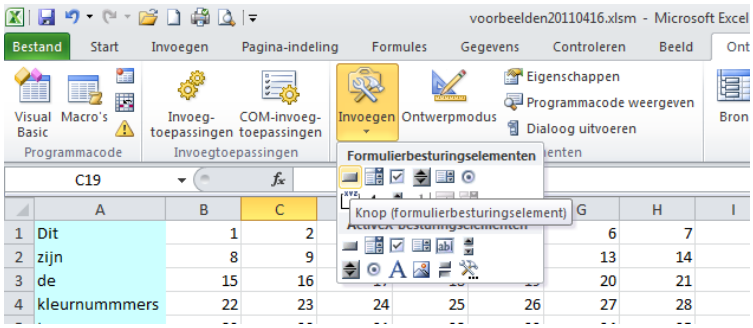
Met de constructie **For Each shtX In Worksheets** lopen we vervolgens door de **Collection** (verzameling) van alle **Worksheets** heen. In dit voorbeeld neemt de variabele als het ware de plaats in van het object **Worksheet**.

Nu een iets ingewikkelder voorbeeld. We willen op elke pagina de **Gridlines** uitzetten als deze aanstaan en omgekeerd. De eigenschap **DisplayGridlines** hoort bij het object **Window** (onderdeel van de collectie **Windows).**

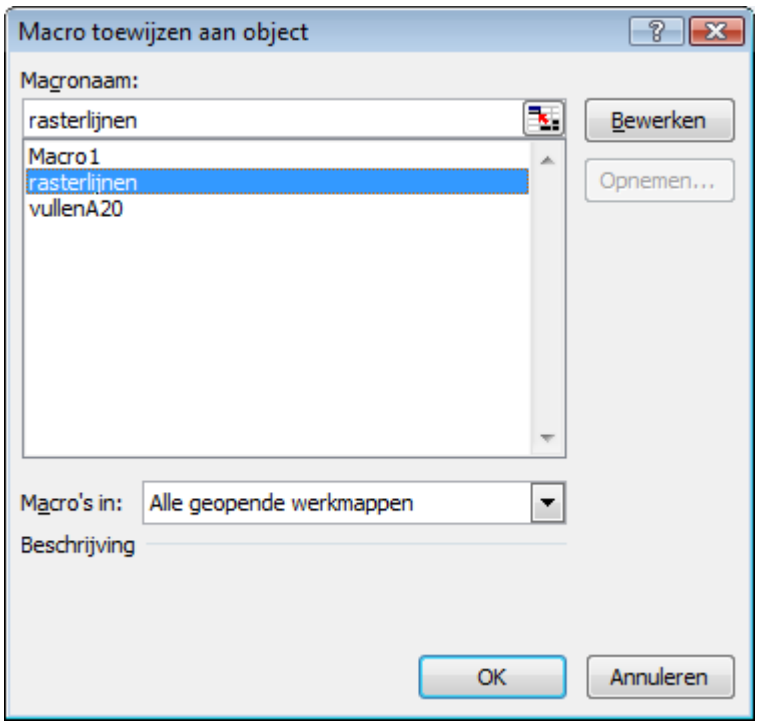
```
Sub rasterlijnen()
   Dim shtX As Worksheet
   For Each shtX In Worksheets
       shtX.Activate
       ActiveWindow.DisplayGridlines = Not (ActiveWindow.DisplayGridlines)
   Next
   Worksheets(1).Activate
End Sub
```

We hadden dergelijke code met goed gekozen trefwoorden via Google kunnen opzoeken.

- We maken op blad 1 een knop:



- En koppelen daar de procedure aan:



Nog een voorbeeld: conditioneel kleuren van tabs.

De tabs worden rood gekleurd als het blad meer dan tien rijen telt.

```
Sub TabsKleuren()
   Dim shtBlad As Worksheet
   For Each shtBlad In Worksheets
       If shtBlad.UsedRange.Rows.Count > 10 Then
           shtBlad.Tab.Color = 255
       Else
           shtBlad.Tab.Color = 0
       End If
   Next
End Sub
```

Complexer voorbeeld: printen alle grafieken.
Het **ChartObject** object verwijst naar grafieken die ingebd zijn in werkbladen. Het **Chart object** verwijst naar grafieken die hun eigen werkblad hebben.

```
Sub PrintGrafieken()
  Application.ScreenUpdating = False
  Dim chtGrafiek As Object
  Dim shtW As Worksheet
  Dim intTeller As Integer

  intTeller = 0

   'Print Chart Objects
   For Each shtW In ActiveWorkbook.Worksheets
      'loopt door de ingebedde grafieken
      shtW.Activate
       For Each chtGrafiek In shtW.ChartObjects
           If chtGrafiek.Height < chtGrafiek.Width Then
               chtGrafiek.Chart.PageSetup.Orientation = xlLandscape
           Else
               chtGrafiek.Chart.PageSetup.Orientation = xlPortrait
           End If
           intTeller = intTeller + 1
           chtGrafiek.Chart.PrintOut
       Next chtGrafiek
   Next shtW

   'Print Charts
   For Each chtGrafiek In ActiveWorkbook.Charts
        'loopt door de aparte grafiek werkbladen
       intTeller = intTeller + 1
       chtGrafiek.PrintOut
   Next chtGrafiek

   MsgBox "Printen van " & intTeller & " grafieken uit bestand " _
       & ActiveWorkbook.Name & ".", vbInformation, "Print grafieken"

  Application.ScreenUpdating = True
End Sub
```

3.4.3 FOR NEXT

In deze loop werken we met een zogenaamde teller. Zo kunnen we bijvoorbeeld alle objecten aan de hand van hun rangnummer langslopen en daar iets mee doen.
Stel we willen in een nieuwe werkmap alle bladen een naam geven van **Week01** tot en met **Week53**. Allereerst moeten we dat het aantal werkbladen aanvullen tot het aantal van 53.

Hoe tellen we het aantal bestaande werkbladen? Dat is een eigenschap van de verzameling **Worksheets**:

```
Worksheets.Count
```

We maken even een nieuwe procedure met de naam **Weeknamen**.

```
Sub WeekNamen()
   MsgBox Worksheets.Count
End Sub
```

Als we de code testen, geeft de **MsgBox** het aantal werkbladen. Hoeveel moeten we er nu bijmaken? 53 – ***Worksheets.Count***. En dat doen we met een **FOR NEXT**.

```
Sub WeekNamen()
   Dim intTeller As Integer
   For intTeller = 1 To 53 - Worksheets.Count
      'voegt een werkblad toe
      Worksheets.Add
   Next
End Sub
```

We hadden het hier korter kunnen houden door **Worksheets.Add** gelijk een parameter mee te geven voor het aantal:

```
Worksheets.Add Count:=53
```

Voor de variabele **intTeller** kiezen we als type **Integer** met een waardebereik van -32.768 tot 32.767.

Voeren we deze routine uit dan hebben we 53 werkbladen! Let op dat het toevoegen niet van voren naar achteren gaat en dat ze dus niet op volgorde liggen. Dat had weer wel gekund door **Worksheets.Add** een parameter mee te geven:

```
Worksheets.Add After:=Worksheets(Worksheets.Count)
```

Het nieuwe werkblad wordt nu achter de het laatste blad uit de collectie geplaatst.

Nu de naamgeving. We vullen de code van de sub **Weeknamen** aan met:

```
For intTeller = 1 To Worksheets.Count
  Worksheets(intTeller).Name = "Week" & Right("0" & intTeller, 2)
Next
```

Met de code:

```
Right("0" & intTeller,2)
```

plakken we steeds een 0 voor het cijfer en pakken dan de achterste 2 cijfers zodat we beginnen met 01 etc. Een programmeertrucje. **Right** is hier een functie. Op functies komen we later terug.

3.4.4 SELECT CASE

De constructie met **SELECT CASE** geniet de voorkeur boven **IF THEN ELSE** als het om meer dan 2 keuzes gaat en we met een zogenaamde geneste **IF** zouden moeten werken.

We bekijken het onderstaande voorbeeld waar cellen gekleurd worden op basis van de inhoud:

```
Sub celKleurenObvInhoud()
  Dim rngCell As Range
  Dim strKleur1 As String, strKleur2 As String, strKleur3 As String
  strKleur1 = RGB(255, 255, 200)
  strKleur2 = RGB(255, 200, 255)
  strKleur3 = RGB(200, 255, 255)
  Worksheets(1).Activate

  For Each rngCell In ActiveSheet.Range("A1:F17")
      'de waarde rngCell.value is de input voor de Select Case
      Select Case rngCell.Value
          'alles tussen de 0 en 5
          Case 0 To 5
              rngCell.Interior.Color = strKleur1
          Case 6 To 10
              rngCell.Interior.Color = strKleur2
          Case 11 To 15
              rngCell.Interior.Color = strKleur3
          Case Else
              rngCell.Interior.ColorIndex = xlNone
      End Select
  Next
End Sub
```

- Vul op het eerste werkblad het bereik **A1:F17** met de getallen tussen de 1 en de 20.
- Maak bovenstaande procedure.
- Bekijk het resultaat.

3.4.5 DO WHILE etc.

Simpel voorbeeld voor het toevoegen van bladen op basis van de inhoud van blad 1.

```
Sub Bladentoevoegen()
  Dim rngCel As Range

  Application.DisplayAlerts = False
  'bestaande bladen wissen
  Do While True
      If Sheets.Count = 1 Then
          Exit Do
      End If

      Sheets(Sheets.Count).Delete
  Loop
  Application.DisplayAlerts = True

  'nieuwe bladen toevoegen
  For Each rngCel In Sheets(1).UsedRange.Columns(1).Cells
      Sheets.Add After:=Sheets(Sheets.Count)
      Sheets(Sheets.Count).Name = rngCel.Value
  Next
  Sheets(1).Activate
End Sub
```

3.5 Opgaven

- Pas nu zelf het laatste voorbeeld uit paragraaf 2.2 met **SelectionChange** aan
- Maak een werkboek met 53 bladen waar bij elk blad een eigen kleurcode krijgt
  tip: gebruik **worksheets**(**intT**).**tab**.**colorindex**
- Maak een werkmap met een aantal werkbladen met een knop waarbij alle werkbladen behalve de huidige verborgen worden
  tip: gebruik de eigenschap **visible** van een **worksheet** (true of false)

4 Excel object uitgelicht: UsedRange

4.1 Inleiding

In het laatste voorbeeld in hoofdstuk 3.4.4 hebben we expliciet het bereik aangegeven:

```
Range("A1:F17")
```

Zouden we het bereik verplaatsen, dan werkt onze **VBA** code niet meer. Een probleem dus. We kunnen dit oplossen met een zeer handig **Excel** object: **UsedRange**. **UsedRange** verwijst naar het gebruikte bereik op het actieve werkblad en beslaat de gehele rechthoek tussen de eerste gebruikte cel links boven en de laatst gebruikte cel rechtsonder. Wissen we de cel rechtsonder dan blijft dit punt actief totdat het werkboek wordt opgeslagen. Daarna loopt de **UsedRange** tot de nieuwe laatst gebruikte cel rechtsonder.

Nu even een idee van de mogelijkheden van dit object:

ActiveSheet.UsedRange	Stelt de UsedRange opnieuw in na een verandering
ActiveSheet.UsedRange.Cells	Alle cellen in de UsedRange
ActiveSheet.UsedRange.Cells(2,1).Select	**Methode**: selecteert de eerste cel van de tweede rij
ActiveSheet.UsedRange.Columns.Count	**Eigenschap**: geeft het aantal gebruikte kolommen
ActiveSheet.UsedRange.Rows.Count	Geeft het aantal gebruikte rijen
ActiveSheet.UsedRange.Columns(1).Select	Selecteert de eerste kolom
ActiveSheet.UsedRange.Rows(1).Columns(1).Select	Selecteert de cel linksboven
ActiveSheet.UsedRange.Columns(6).Delete	**Methode**: wist kolomnummer 6
ActiveSheet.UsedRange.Columns(1).Cells	Verzameling cellen uit kolom 1
ActiveSheet.UsedRange.Columns(2).Rows.Count	Geefft de rijen in de tweede kolom
ActiveSheet.UsedRange.EntireRow.EntireColumn.AutoFit	**Methode**: brengt alle kolommen en rijen op passende bereedte
ActiveSheet.UsedRange.Copy	**Methode**:zet de UsedRange op het klembord
ActiveSheet.UsedRange.Columns.Count	Geeft het nummer van de laatst gebruikte kolom
ActiveSheet.UsedRange.SpecialCells(xlCellTypeFormulas).Select	Selecteer alle formules in de UsedRange
ActiveSheet.UsedRange.SpecialCells (xlCellTypeFormulas).Interior.ColorIndex=1	**Eigenschap**: stelt de kleuren index in voor alle formules in de UsedRange
ActiveSheet.UsedRange.Cells(1).Value	**Eigenschap**: geeft de waarde van de eerste cel uit de UsedRange
Activesheet.UsedRange.Range(Cells(2,1),Cells(10,10)).Select	Selecteert binnen de UsedRange vanaf de tweede rij en eerste kolom tot aan de tiende kolom en rij
ActiveSheet.UsedRange.Range(Columns(1), Columns(5)).Select	Selecteert kolom 1 tot en met 5

We kunnen het voorbeeld uit hoofdstuk 3.4.4 nu simpel aanpassen door

```
Range("A1:F17")
```

te vervangen door

```
UsedRange
```

```
Sub celKleurenObvInhoud()
  Dim rngCell As Range
  Dim strKleur1 As String, strKleur2 As String, strKleur3 As String

  'de functie RGB geeft een rood-groen-blauw kleur
  strKleur1 = RGB(255, 255, 200)
  strKleur2 = RGB(255, 200, 255)
  strKleur3 = RGB(200, 255, 255)
  Worksheets(1).Activate

  'we doorlopen elke cel uit het gebruikte blok
  'van links naar rechts en boven naar beneden
  For Each rngCell In ActiveSheet.UsedRange
      Select Case rngCell.Value
          Case 0 To 5
              rngCell.Interior.Color = strKleur1
          Case 6 To 10
              rngCell.Interior.Color = strKleur2
          Case 11 To 15
              rngCell.Interior.Color = strKleur3
          Case Else
              'zet de kleur op niks
              rngCell.Interior.ColorIndex = xlNone
      End Select
  Next
End Sub
```

Daarna levert het geen probleem meer op als we het bereik verplaatsen: de procedure blijft gewoon werken.

4.2 Toepassingen UsedRange

Stel we importeren gegevens in een werkblad en verschillende kolommen blijken leeg te zijn. Die kunnen we dan handmatig gaan wissen. We kunnen daar evenwel ook een procedure voor schrijven.

Het gebied waar de data in staan is de **UsedRange**. De breedte ervan stellen we vast met de code: **ActiveSheet.UsedRange.Columns.Count**.

We testen:

```
Sub kolomWissen()
  Worksheets(1).Activate
  MsgBox ActiveSheet.UsedRange.Columns.Count
  'eigenschap
End Sub
```

We wissen een specifieke kolom 6 met de code:

```
ActiveSheet.UsedRange.Columns(6).Delete
'een methode
```

Om te bepalen of een kolom uit de UsedRange leeg is, tellen we het aantal argumenten in die kolom. We gebruiken daarvoor een **Excel** functie die we aanroepen met ***WorksheetFunction.CountA()***. Dan krijgen we deze code

```
WorksheetFunction.CountA(ActiveSheet.UsedRange.Columns(1))
```

Na enig puzzelen komen we dan met de oplossing:

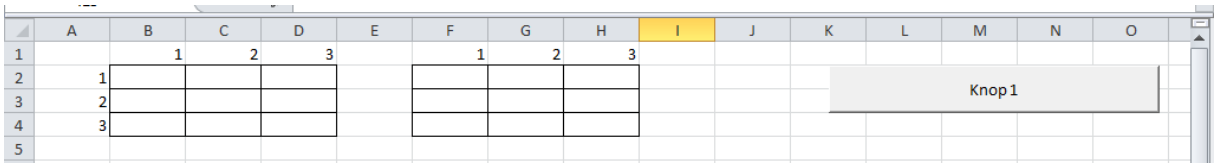
```
Sub kolomWissen()
  Worksheets(1).Activate
  Dim intT As Integer
  For intT = ActiveSheet.UsedRange.Columns.Count To 1 Step -1
    'met WorksheetFunction kunnen we de functies uit Excel gebruiken
    If WorksheetFunction.CountA(ActiveSheet.UsedRange.Columns(intT))=0 Then
        ActiveSheet.UsedRange.Columns(intT).Delete
    End If
  Next
End Sub
```

Een complexer probleem
Werknemers moeten aan de hand van twee variabelen in een grid met 9 vakken worden ingedeeld:

1	Rob	1	3
2	Hessel	2	2
3	Tim	3	1
4	Ron	1	3
5	Walter	2	2
6	Christel	3	1
7	Aap	1	3
8	Noot	2	2
9	Mies	3	1
10	Wim	1	3
11	Zus	2	2
12	Jet	3	1
13	Teun	1	3
14	Vuur	2	2
15	Gijs	3	1
16	Jan	1	1
17	Feb	2	2
18	Mrt	3	3
19	Apr	1	1
20	Mei	2	2
21	Jun	3	3
22	Jul	1	1
23	Aug	2	2
24	Sep	3	3
25	Okt	1	1
26	Nov	2	2

Data | Grid | Cat

Het Grid, ййn voor de namen en ййn voor de percentages



De gecombineerde mogelijkheden zetten we op een derde blad:

	A
1	11
2	12
3	13
4	21
5	22
6	23
7	31
8	32
9	33
10	
11	

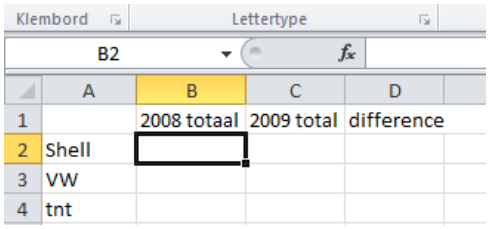
De code waar we stap voor stap naar toe kunnen werken, komt er zo uit te zien.

```
Sub opDelenInCat()
  Application.ScreenUpdating = False
  'ScreenUpdating is een eigenschap
  Dim rngCell As Range, rngCell2 As Range
  Dim strAll As String
  Dim intAll As Integer, intTot As Integer

  'ga naar het laatste werkblad Cat
  Worksheets(Worksheets.Count).Select
  For Each rngCell In ActiveSheet.UsedRange
  'loopt langs de mogelijkheden 11-33 in de range A1:A9
    Worksheets(1).Select
    'gaat naar blad Data
    intTot = ActiveSheet.UsedRange.Rows.Count
    'telt in data het aantal rijen; nodig om percentage te berekenen

    For Each rngCell2 In ActiveSheet.UsedRange.Columns(1).Cells
      'trim verwijdert de spaties
      'met de eigenschap Offset bepalen we het punt vanaf de gekozen startcel
      If Trim(rngCell2.Offset(0, 1).Value) & Trim(rngCell2.Offset(0,
2).Value) = Trim(rngCell.Value) Then
        'chr(10) voegt een harde Return (druk op ENTER) toe
          strAll = strAll & rngCell2.Value & Chr(10)
          intAll = intAll + 1
          'aantal voorkomende gevallen geteld voor berekenen percentage
      End If
    Next 'ga naar de volgende rij op het blad Data

    Worksheets(2).Select
    'gaat naar werkblad Grid en voeg de resultaten in
    Range("a1").Offset(Left(rngCell, 1), Right(rngCell, 1)).Value = strAll
    Range("e1").Offset(Left(rngCell, 1), Right(rngCell, 1)).Value = intAll /
intTot
    strAll = ""
    intAll = 0
  Next 'haal de volgende waarde op van het blad Cat
  Application.ScreenUpdating = True
End Sub
```

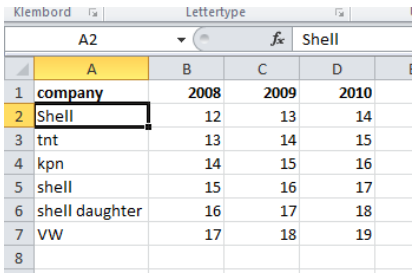
Het resultaat ziet er dan zo uit:

	A	B	C	D	E	F	G	H	I	J	K	L
1		1	2	2		1	2	3				
2	1	Jan Apr Jul Okt		Rob Ron Aap Wim Teun		15%	0%	19%		Vullen		
3	2		Hessel Walter Noot Zus Vuur Feb Mei Aug Nov			0%	33%	0%				
4	3	Tim Christel Mies Jet Gijs		Mrt Jun Sep Dec		19%	0%	15%				
5												

Nog een complex probleem.
Op het getoonde totaalblad willen we de totalen van de andere werkbladen hebben. Daarbij moeten alle resultaten van bedrijven waarin de woorden **Shell**, **VW** en **tnt** in voorkomt bijelkaar geteld worden.



Kijk maar eens naar het volgende blad:



Voor Shell levert dit dus voor 2008 12 + 15 + 16 op. Zo zijn er meerdere bladen met elk hun eigen inhoud.

Hoe pakken we dit aan? We beginnen op het laatste blad met de resultaten. Dat activeren we met de code:

```
Worksheets(Worksheets.Count).Activate
```

Vervolgens loopen we hier door de eerste kolom van de **UsedRange** met een **FOR EACH**:

```
For Each rngCell In ActiveSheet.UsedRange.Columns(1).Cells
  If rngCell.Value <> "" Then
    strCompany = rngCell.Value
  End If
```

Met de eerst gevonden waarde activeren we de eerste Worksheet:

```
For Each shtSheet In Worksheets
  If shtSheet.Name = Worksheets(Worksheets.Count).Name Then
      'Name is een eigenschap
      Exit For
  End If
```

Het voorgaande zou ook zo kunnen:

```
For intTeller = 1 to Worksheets.Count -1
  Worksheets(intTeller).Activate 'Methode
```

We houden er in beide gevallen rekening mee dat we laatste werkblad buiten de loop moeten laten. In actuele Worksheet loopen we dan door de eerste kolom:

```
For Each rngCell2 In ActiveSheet.UsedRange.Columns(1).Cells
  If InStr(UCase(rngCell2.Value), UCase(strCompany)) > 0 Then
    'Instr en Ucase zijn functies
    dblTotal = dblTotal + rngCell2.Offset(0, 1)
    dblTotal2 = dblTotal2 + rngCell2.Offset(0, 2)
    'Offset is een eigenschap
  End If
Next
```

We gebruiken de **InStr** functie om te bepalen of de tekst van het resultaten blad gevonden wordt. De **InStr** is hoofdlettergevoelig en daarom moeten we ook de **Ucase** functie gebruiken. Vinden we de waarde dan nemen we de waarde uit de naastliggende cellen mee en hogen een variabele er mee op.

Zijn we door alle bladen tot aan de laatste heen, dan keren we terug naar het laatste blad plakken het eindresultaat achter de waarde die we als zoekwaarde hebben gebruikt:

```
Worksheets(Worksheets.Count).Activate
   rngCell.Offset(0, 1).Value = dblTotal
   rngCell.Offset(0, 2).Value = dblTotal2
```

Vervolgens pakken we de volgende waarde uit kolom 1 weer op als zoekwaarde en begint het verhaal opnieuw.

Arrays
In de volgende opgaven komen we het begrip **Array** tegen. Met een **Array** bedoelen we een meervoudige, eventueel multidimensionale variabel. In een **Arry** variabele met de naam

```
Dim arrTotaal(3) 'ééndimensionale array variabele
```

Kunnen we 4 waarden stoppen horende bij **arrTotaal**(0) t/m **arrTotaal**(3).

```
Redim arrTotaal(3)
```

In **Excel** 2003 werkt het een en ander nog anders dan in latere versies.

Met dit laatste commando maken we de **Array** weer leeg.

Meer over **Arrays** vinden we in paragraaf 20.3.5.

4.3 Opgaven

- Probeer dit laatste voorbeeld uit te werken.
- Maak een blok van 10 bij 10 met willekeurige getallen en kleur deze met behulp van een randomizer die willekeurige getallen produceert
  tip: gebruik de eigenschap **.Color = RGB(Rnd() * 256, Rnd() * 256, Rnd() * 256)** (**RGB** en **RND** zijn functies)
- Maak analoog naar het voorbeeld van de kolommen een procedure die lege rijen wist.
- Probeer een tekst van blad 1 een aantal keren naar blad 2 te kopiëren zodat het op blad2 steeds onderaan het bestaande bereik wordt toegevoegd
  tip: het aantal rijen van de **Usedrange** kunnen we tellen met UsedRange.rows.count; het startpunt bepalen we met de **Offset** eigenschap.
- We kunnen de vorige oefening verfijnen door de bovenste rij erbuiten te laten
  tip: met **UsedRange.cells(2,1)** beginnen we op de tweede rij in de eerste kolom; met **UsedRange.Range(Cells(2, 1), Cells(intR, intC))** kunnen we het juiste bereik bepalen.
- Het laatste voorbeeld van paragraaf 4.2 hadden we ook een **Array** kunnen gebruiken; we kunnen dan in 1 keer definiëren hoeveel kolommen we willen tellen.
  tip: definieer een Array als dim **arrTotaal**() en redim deze daarna met het aantal kolommen uit de **UsedRange** van het laatste blad -1

5 Excel: tabellen

5.1 Maken van een tabel

```
Sub sbCreatTable()
'Create Table in Excel VBA
Sheet1.ListObjects.Add(xlSrcRange, Range("A1:D10"), , xlYes).Name =
"myTable1" 'eigenschap
End Sub
```

5.2 Tabel ontkoppelen

```
Sub sbReset_Table_BackTo_Range()
'Reset Table Back to Original Range
  On Error Resume Next 'If there are no Table ignore the below Statement
  Sheet1.ListObjects("myTable1").Unlist
End Sub
```

5.3 Sorteren van een tabel

```
Sub sbSortTable()
  'Naming a range
  Sheet1.Sheets("Sheet1").ListObjects("myTable1").Sort.SortFields.Clear
  Sheet1.Sheets("Sheet1").ListObjects("myTable1").Sort.SortFields.Add
Key:=Range("myTable1[[#All],[EmpName]]"), SortOn:=sortonvalues,
Order:=xlAscending, DataOption:=xlSortNormal
Range("myTable1[#All]").Select
With Sheet1.Worksheets("Sheet1").ListObjects("myTable1").Sort
  .Header = xlYes
  .MatchCase = False
  .Orientation = xlTopToBottom
  .SortMethod = xlPinYin
  .Apply
End With
End Sub
```

5.4 Filteren van een tabel

```
Sub sbFilterTable()
  ActiveWorkbook.Sheets("Sheet1").ListObjects("myTable1").Range.AutoFilter
field:=2, Criteria1:="DDD"
  'matched with 4 in column c2 records will be shown
End Sub
```

5.5 Weghalen van filters

```
Sub sbClearFilter()
'Check Filter is Exists or Not
If ActiveWorkbook.Sheets("Sheet1").FilterMode = True Then
  ActiveWorkbook.Sheets("Sheet1").ListObjects("myTable1").Range.AutoFilter
End If
End Sub
```

5.6 Selecteren in een tabel met VBA

Select	VBA Coding
Gehele tabel	ActiveSheet.ListObjects("Table1").Range.Select
Tabel rijkop	**ActiveSheet.ListObjects("Table1").HeaderRowRange.Select**
Tabel data	**ActiveSheet.ListObjects("Table1").DataBodyRange.Select**
Derde kolom	**ActiveSheet.ListObjects("Table1").ListColumns(3).Range.Select**
Data derde kolom	**ActiveSheet.ListObjects("Table1").ListColumns(3).DataBodyRange.Select**
Data rij vier	**ActiveSheet.ListObjects("Table1").ListRows(4).Range.Select**
Derde kop	**ActiveSheet.ListObjects("Table1").HeaderRowRange(3).Select**
Datapunt in rij 3, kolom 2	**ActiveSheet.ListObjects("Table1").DataBodyRange(3, 2).Select**
Subtotalen	**ActiveSheet.ListObjects("Table1").TotalsRowRange.Select**

5.7 Rijen en kolommen toevoegen aan een tabel

Toevoegen	VBA
Nieuwe kolom vier	**ActiveSheet.ListObjects("Table1").ListColumns.Add Position:=4**
Kolom aan het eind	**ActiveSheet.ListObjects("Table1").ListColumns.Add**
Rij toevoegen boven rij elf	**ActiveSheet.ListObjects("Table1").ListRows.Add (5)**
Rij onderaan	**ActiveSheet.ListObjects("Table1").ListRows.Add AlwaysInsert:= True**
Totalenrij	**ActiveSheet.ListObjects("Table1").ShowTotals**

5.8 Inlezen van gegevens in een array variable

5.8.1 Tabel met ййn kolom

```
Sub SingleColumnTable_To_Array()
  Dim myTable As ListObject
  Dim myArray As Variant
  Dim TempArray As Variant
  Dim x As Long

  'Set path for Table variable
  Set myTable = ActiveSheet.ListObjects("Table1")

  'Create Array List from Table
  TempArray = myTable.DataBodyRange

  'Convert from vertical to horizontal array list
  myArray = Application.Transpose(TempArray)

  'Loop through each item in the Table Array (displayed in Immediate Window
[ctrl + g])
  For x = LBound(myArray) To UBound(myArray)
    Debug.Print myArray(x)
  Next x
end Sub
```

5.8.2 Tabel met meerdere kolommen

```
  Sub MultiColumnTable_To_Array()
    Dim myTable As ListObject
    Dim myArray As Variant
    Dim x As Long

    'Set path for Table variable
    Set myTable = ActiveSheet.ListObjects("Table1")

    'Create Array List from Table
    myArray = myTable.DataBodyRange

    'Loop through each item in Third Column of Table (displayed in Immediate
  Window [ctrl + g])
    For x = LBound(myArray) To UBound(myArray)
    Debug.Print myArray(x, 3)
    Next x
End Sub
```

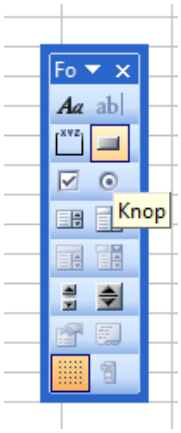
5.9 Links

http://www.jkp-ads.com/articles/**Excel**2007Tables**VBA**.asp

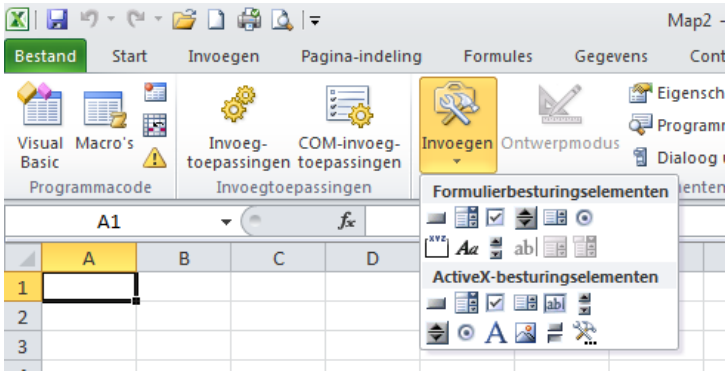
6 (Menu)knoppen en VBA

6.1 VBA voor knoppen en het verbergen van werkbladen

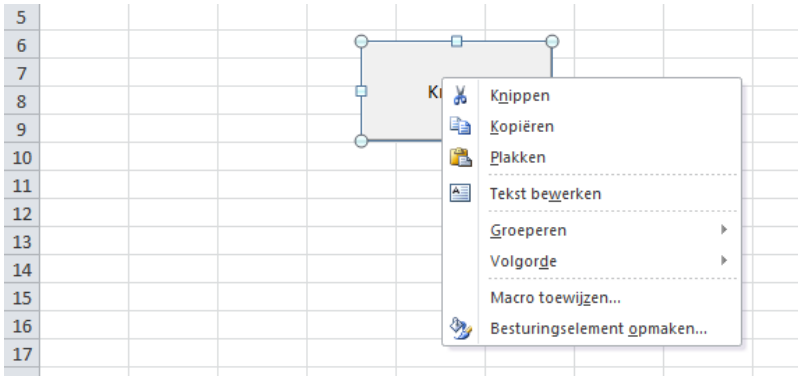
In **Excel 2003** kunnen we knoppen maken via de werkbalk **Formulieren**:



Vanaf versie **2007** doen we dat met de tab **Ontwikkelaars**:



Aan een dergelijke knop kunnen we een macro koppelen:



Als we hiervoor nu de volgende code zouden maken:

```
Sub home()
  Sheets("Home").Select
End Sub
```

Zal een klik op deze knop ons naar een blad met de naam **Home** sturen.

We kunnen de code nog iets verder uitbreiden:

```
Sub home()
  Sheets("Home").visible = True
  Sheets("Home").Select
End Sub
```

Zo wordt het blad waar we heen springen eerst weer zichtbaar gemaakt. Met een beetje fantasie kunnen we nu de knoppen maken om heen en weer te springen door de verschillende bladen en daarbij alleen het blad in beeld zichtbaar te houden.

Dat komt er dan zo uit te zien, inclusief het uitzetten van de communicatie met het scherm:

```
Sub menu()
   Application.ScreenUpdating = False 'eigenschap
   Dim shtBlad As Worksheet
   For Each shtBlad In Worksheets
       shtBlad.Visible = True 'eigenschap
   Next
   Worksheets("home").Activate 'methode
   For Each shtBlad In Worksheets
       If shtBlad.Name <> ActiveSheet.Name Then
           shtBlad.Visible = False
       End If
   Next
   Application.ScreenUpdating = True
End Sub
```

Met **VBA** kunnen we een spreadsheet ook altijd bij het desgewenste blad laten beginnen. Achter **ThisWorkbook** plaatsen we dan de volgende code:

```
Private Sub Workbook_Open()
   Sheets("home").Select
End Sub
```

6.2 Checkboxen

Ook een leuke optie is het automatisch laten toevoegen van checkboxen. Stel we hebben een tabel waarbij we naast de laatste kolom steeds checkboxen willen plaatsen, precies evenveel als er rijen in de kolom zijn. We kunnen dat als volgt doen:

```
Dim rngCell As Range
For Each rngCell In ActiveSheet.UsedRange.Columns(6).Cells 'collectie
   ActiveSheet.CheckBoxes.Add(rngCell.Offset(0, 1).Left, rngCell.Offset(0,
1).Top, rngCell.Offset(0, 1).Width, rngCell.Offset(0, 1).Height).Select
   With Selection
        .LinkedCell = rngCell.Offset(0, 1).Address 'eigenschappen
        .Characters.Text = "" 'eigenschap
        .Name = rngCell.Offset(0, 1).Address 'eigenschappen
   End With
Next
```

We kunnen ze weer automatisch wissen met:

```
'wissen vinkvakjes
ActiveSheet.CheckBoxes.Delete 'methode
```

7 Functions

7.1 Inleiding

Bij een functie komt er kort gezegd een waarde binnen, ondergaat deze binnen de functie een bewerking en geeft de functie vervolgens een waarde terug. Laten we eens wat voorbeelden gaan bekijken.

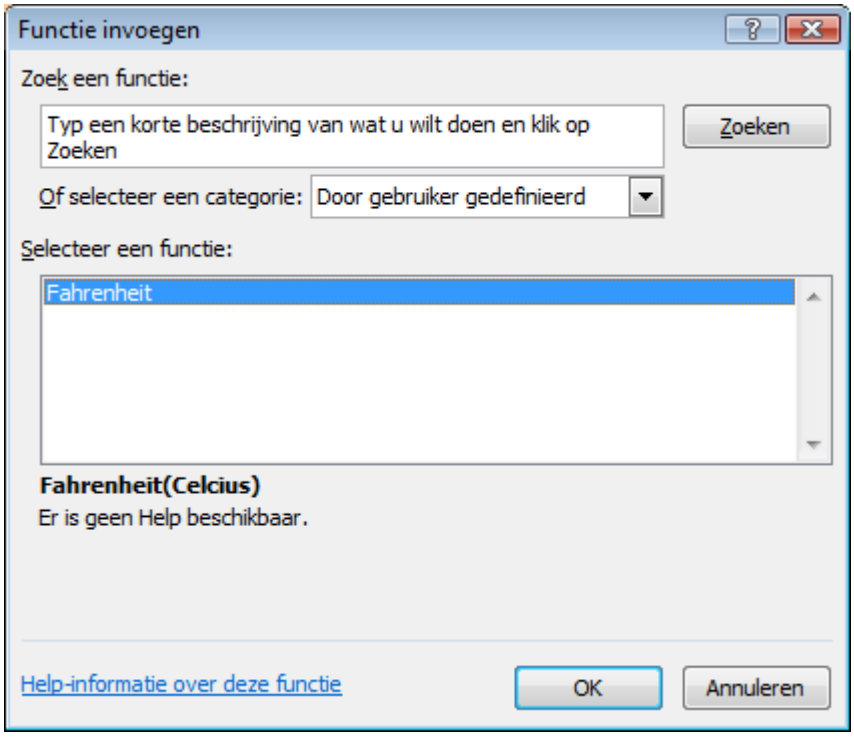
7.2 Van Celsius naar Fahrenheit

Een simpel voorbeeld: het omrekenen van graden Celsius naar Fahrenheit.

```
Function Fahrenheit(Celsius As String) As Variant
   'IsNumeric test of een waarde numeriek is
   If Not IsNumeric(Celsius) Then
       Fahrenheit = "geen waarde"
   Else
       'CDbl zet een waarde om naar een Double
       Fahrenheit = CDbl((9 / 5) * Celsius + 32)
   End If
End Function
```

In dit voorbeeld verwachten we een waarde van het type **String** omdat we niet weten wat de gebruiker gaat invullen. We laten de functie een variabele van het type **Variant** opleveren omdat we ook niet weten wat eruit gaat komen.

Deze functie kunnen we vanuit het werkblad aanroepen. We vinden hem onder de rubriek ***Door gebruiker gedefinieerd***:



7.3 Controleren van burgerservicenummer

Dit nummer moet voldoen aan de zogenaamde 11-proef. Als we het burgerservicenummer voorstellen door de reeks ABCDEFGHI, dan moet: (9 x A) + (8 x B) + (7 x C) + (6 x D) + (5 x E) + (4 x F) + (3 x G) + (2 x H) - (1 x I) een veelvoud van 11 zijn. We kunnen met deze combinatie bijna 91 miljoen nummers creëren. Geldige voorbeelden zijn: 111222333 en 123456782.

We gaan dit uitwerken in een procedure:

```
Function bsn(bsnummer As String) As String
  Dim intT As Integer, intTotaal As Integer

  If Len(bsnummer) <> 9 Then
    bsn = "mispoes: korter dan 9"
    'verlaat de functie
    Exit Function
  Else
    For intT = 1 To Len(bsnummer) 'functie
      'met Mid kunnen we bepalen waar in de string we willen beginnen en welk
      'deel we kiezen
        If Not IsNumeric(Mid(bsnummer, intT, 1)) Then
          'IsNumeric is een functie
          bsn = "mispoes: zit een letter in"
          Exit Function
        End If
    Next

    For intT = Len(bsnummer) To 1 Step -1
        If intT = 1 Then
          intTotaal = intTotaal + Mid(bsnummer, 10 - intT, 1) * - intT
        Else
          intTotaal = intTotaal + Mid(bsnummer, 10 - intT, 1) * intT
        End If
    Next

    If intTotaal Mod 11 = 0 Then
        bsn = "bingo"
    Else
        bsn = "goed fout"
    End If
  End If
End Function
```

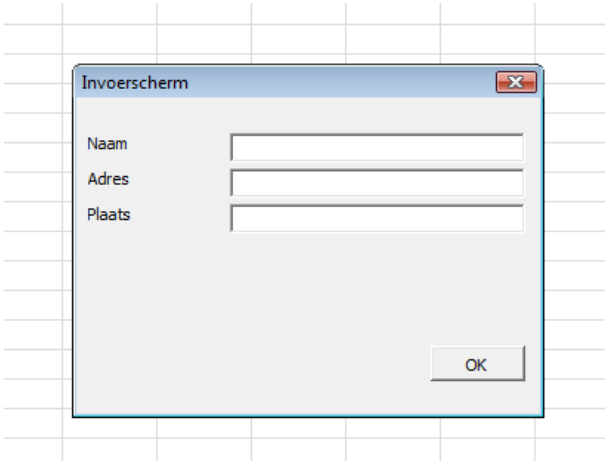
7.4 Opgaven

- Maak een functie die Fahrenheit omzet naar Celsius
- Maak een functie die banknummers controleert
  tip: dan geldt (9 x A) + (8 x B) + (7 x C) + (6 x D) + (5 x E) + (4 x F) + (3 x G) + (2 x H) + (1 x I) is deelbaar door 11
- Maak een functie die uit een willekeurige datum het bijbehorende weeknummer afleidt
  tip: we hebben binnen deze functie **Datepart**() nodig
- Maak een functie die bepaalt of een jaar wel of geen schrikkeljaar is
  tip: jaren zijn schrikkeljaren als ze deelbaar zijn door 4, maar niet als ze deelbaar zijn door 100 maar weer wel als ze deelbaar zijn door 400
- Maak een functie die de correcte leeftijd berekent op basis van de geboortedatum en de huidige datum (date)

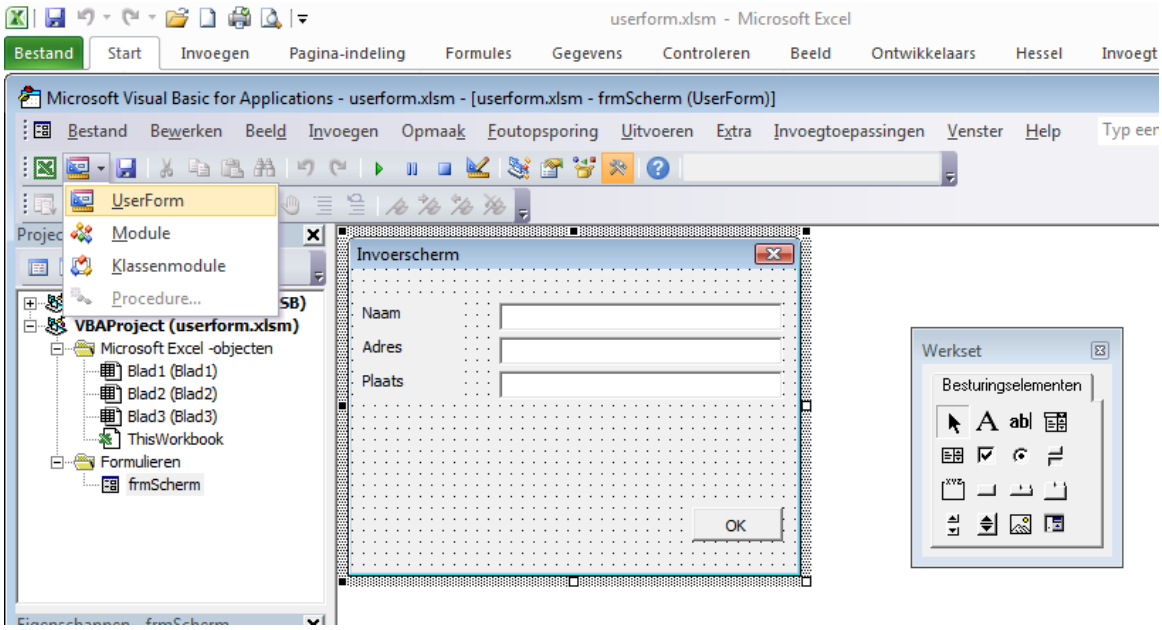
8 Dialoogvensters: Userforms

8.1 Inleiding

We kunnen in **Excel** ook zogenaamde dialoogvensters maken en deze met **VBA** ondersteunen.



Deze schermen zijn via de **VBA editor** te maken:



In dit voorbeeld heb ik een dialoogvenster gemaakt met drie simpele tekstvelden. De namen van de labels beginnen steeds met ***lbl*** en die van de tekstvelden met ***txt***.

Om het scherm te starten plaatsen we de volgende code achter **ThisWorkbook**:

```
Private Sub Workbook_Open()
  'toon het het scherm
  frmScherm.Show 'methode
End Sub
```

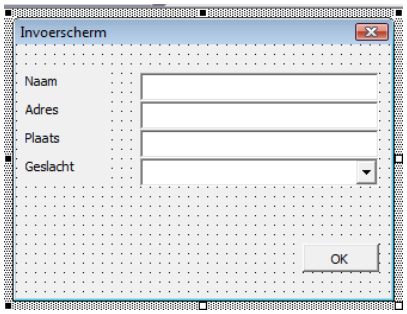
Achter het dialoogvenster heb ik de code geplaatst:

```
Private Sub UserForm_Initialize()
   'plaats cursor in eerste vakje
   Me.txtNaam.SetFocus 'methode
End Sub
```

En verder de code om het klikken op de knop **OK** af te handelen:

```
Private Sub cmbOK_Click()
  Dim rngC As Control
  Dim intC As Integer
  intC = 0

  'plaatsen cursor
  Cells(1, 1).Select

  'uitlezen controls en plaatsen in Excel
  For Each rngC In Controls 'collectie
      If Left(rngC.Name, 3) = "txt" Then 'Left is een functie
          ActiveCell.Offset(0, intC).Value = rngC.Value
          intC = intC + 1
      End If
  Next

  'verberg scherm
  Me.Hide
End Sub
```

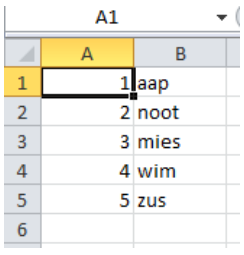
We kunnen het dialoogvenster verder uitbreiden met bijvoorbeeld een combobox:



Om deze te initialiseren breiden we de **UserForm_Initialize** uit:

```
Private Sub UserForm_Initialize()
  'plaatsen cursor in eerste vakje
  Me.txtNaam.SetFocus

  'initialiseren combobox geslacht
  cboGeslacht.AddItem "heer" 'AddItem is een methode
  cboGeslacht.AddItem "mevrouw"
  cboGeslacht.AddItem "onbekend"

  'listindex geeft de default aan, in dit geval heer
  cboGeslacht.ListIndex = 0 'Eigenschap
End Sub
```

We kunnen de waarden ook vanuit een **Range** in **Excel** inlezen:

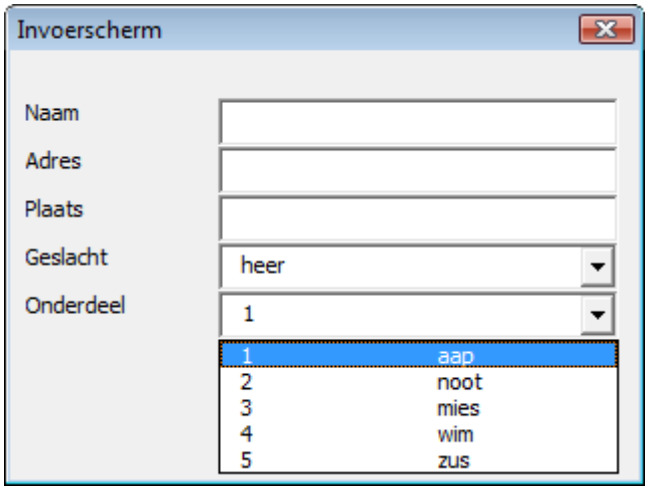


De eerste kolom geven we dan de naam ***onderdelen***.

De **Initialize** code ziet er dan zo uit:

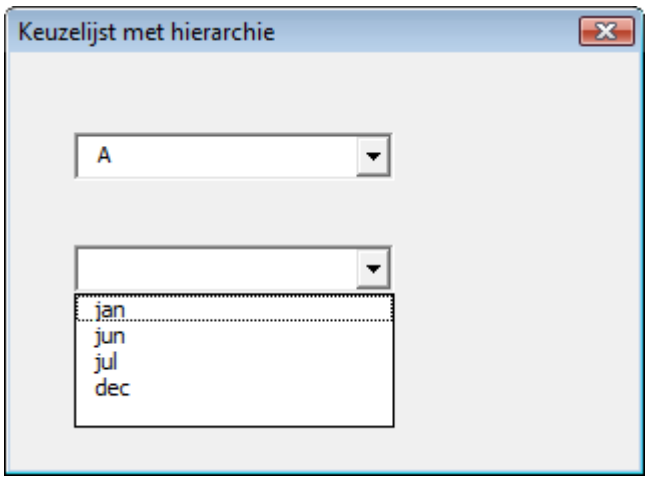
```
For Each rngCell In Worksheets(2).Range("Onderdelen").Cells
      With Me.cboOnderdeel
           .AddItem rngCell.Value
           .List(.ListCount - 1, 1) = rngCell.Offset(0, 1).Value
      End With
Next
```

En het eindresultaat zo:



Twee keuzelijsten met hierarchie
Ten slotte nog een voorbeeld met twee keuzelijsten waarbij de bovenste de onderste moet filteren. De waarden worden hierbij door middel van een **array** vanuit **VBA** uit een **Excel** sheet gelezen:

	A	B
1	A	jan
2	B	feb
3	C	mrt
4	D	apr
5	E	mei
6	A	jun
7	A	jul
8	B	aug
9	B	sep
10	D	okt
11	C	nov
12	A	dec
13		



De volgende code hoort hierbij:

```
Private Sub CmbTeam_Change()
   'MsgBox Me.CmbTeam.Value
   Me.CmbMedewerker.Visible = True
   Dim arrTotaal() As String
   Dim intTeller As Integer, intNieuw  As Integer
   Dim rngCell As Range
   Dim strBingo As String

   For Each rngCell In ActiveSheet.UsedRange.Columns(1).Cells
       If rngCell.Value = Me.CmbTeam Then
           intTeller = intTeller + 1
           ReDim Preserve arrTotaal(intTeller)
           arrTotaal(intTeller - 1) = rngCell.Offset(0, 1).Value
       End If
   Next
   Me.CmbMedewerker.List() = arrTotaal
End Sub

Private Sub UserForm_Initialize()
   Dim arrTotaal() As String
   Dim intTeller As Integer, intNieuw  As Integer
   Dim rngCell As Range
   Dim strBingo As String
   For Each rngCell In ActiveSheet.UsedRange.Columns(1).Cells
       strBingo = False
       For intNieuw = 0 To intTeller - 1
           If arrTotaal(intNieuw) = rngCell.Value Then
               strBingo = True
               Exit For
           End If
       Next
       If strBingo = False Then
           intTeller = intTeller + 1
           ReDim Preserve arrTotaal(intTeller)
           arrTotaal(intTeller - 1) = rngCell.Value
       End If
   Next
   Me.CmbTeam.List() = arrTotaal
End Sub
```

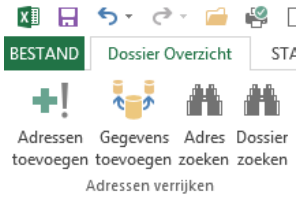
8.2 Opgaven

- Maak een eigen voorbeeld met behulp van het bovenstaande

9 Eigen tabs

Vanaf Excel 2007 kunnen we eigen tabs alleen nog met behulp van XML maken. We hebben hierbij de **Custum UI Editor for Microsoft Office** nodig. De is gratis te downloaden.

Voorbeeld van een zelfgemaakte tab:



Het XML achter het Excel bestand ziet er zo uit:

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
 <ribbon startFromScratch="false">
  <tabs>
    <tab id="MaxIlze" label="Dossier Overzicht" insertBeforeMso="TabHome">
      <group id="customGroup1" label="Adressen verrijken">
        <button id="customButton1" label="Adressen toevoegen" size="large"
 onAction="toevoegenpchnr" imageMso="QueryAppend" />
       <button id="customButton2" label="Gegevens toevoegen" size="large"
 onAction="vullen" imageMso="DatabaseAccessBackEnd"/>
       <button id="customButton3" label="Adres zoeken" size="large"
 onAction="zoekenpostcode" imageMso="FindDialog"/>
       <button id="customButton4" label="Dossier zoeken" size="large"
 onAction="zoekendossier" imageMso="FindDialog"/>
      </group>
    </tab>
  </tabs>
 </ribbon>
</customUI>
```

De inhoud van de eigenschap **imageMso** verwijst naar bestaande **Microsoft** buttons. De eigenschap **onAction** verwijst naar **VBA** procedures achter het **Excel** bestand.

Zo'n procedure moet dan wel zo beginnen:

```
Sub toevoegenpchnr(control As IRibbonControl)
```

10 Excel VBA en Access

10.1 Inleiding

Willen we vanuit **Excel** met **VBA** gegevens uit Access of een andere database halen dan moeten we gebruik maken van DAO of ADO.

DAO
Uit Wikipedia: *In computer software, a data access object (DAO) is an object that provides an abstract interface to some type of database or persistence mechanism, providing some specific operations without exposing details of the database. It provides a mapping from application calls to the persistence layer. This isolation separates the concerns of what data accesses the application needs, in terms of domain-specific objects and data types (the public interface of the DAO), and how these needs can be satisfied with a specific DBMS, database schema, etc. (the implementation of the DAO).*

ADO
Uit Wikipedia: *ActiveX Data Objects, afgekort ADO, is een verzameling van COM-objecten die gebruikt worden om gegevensbronnen aan te spreken. Bij gegevensbronnen moet men denken aan bijvoorbeeld databases. ADO is ontwikkeld door Microsoft. Het doel van ADO is om een laag te vormen tussen een programmeertaal en een database. Een programmeur die in zijn programma data wil ophalen hoeft dan niet te weten welke technieken zijn gebruikt om de database te bouwen. De programmeur hoeft dan alleen kennis te hebben van de ADO-functies om data op te halen. Wanneer gebruikgemaakt wordt van ADO om databronnen aan te spreken, hoeft de programmeur bijvoorbeeld geen SQL te kennen.*

10.2 Van Access naar Excel met DAO

We gaan nu binnen **Excel** een **VBA**-module maken en daarbinnen de DAO-verwijzing activeren. Vervolgens leren we hoe we alle gegevens uit een tabel in een **Excel**-werkblad plaatsen.

- Start **Excel** op.
- Activeer de **Excel**-Visual Basic-editor via toetscombinatie **Alt F11**.
- Voeg een nieuwe module toe via de menuopties **Invoegen** ⇒ **Module**.

De module wordt automatisch Module1 genoemd.

- Voeg voor dit project (Map1) een verwijzing toe naar de DAO-bibliotheek via de menuopties **Extra** ⇒ **Verwijzingen**.
- Vink vervolgens het keuzevakje voor **Microsoft DAO 3.51 Object Library** aan (3.51 is afhankelijk van de geïnstalleerde officeversies).
- Klik op **OK**.
- Voer de volgende code in:

```
Sub accessinlezen()
  'communicatie met scherm uitzetten
  Application.ScreenUpdating = False

  'DAO library moet aangevinkt via Extra ⇒ verwijzingen
  Dim dbNwind As DAO.Database
  Set dbNwind = DAO.OpenDatabase("c:\VBA Excel\noordenwind.mdb")
  Dim rs As DAO.Recordset

  Worksheets(1).Activate
  Dim strTabel As String

  'er van uitgaande dat in cel B1 de naam van de tabel staat
  strTabel = Range("b1").Value

  Set rs = dbNwind.OpenRecordset(strTabel)

  'om te testen of het werkt
  'Range("a1").Value = rs.Fields(0).Value
  Range("a4:z5000").Clear 'methode
  Range("a4").Select

  Dim intTeller As Integer, intTellerVert As Integer
  intTellerVert = 1

  'voor het afdrukken van de veldnamen
  For intTeller = 0 To rs.Fields.Count - 1
      ActiveCell.Offset(0, intTeller).Value = rs.Fields(intTeller).Name
  Next

  'voor het afdrukken van de rijen
  Do Until rs.EOF
      For intTeller = 0 To rs.Fields.Count - 1
        ActiveCell.Offset(intTellerVert, intTeller).Value =
rs.Fields(intTeller).Value
      Next
      intTellerVert = intTellerVert + 1
      rs.MoveNext 'methode
  Loop

  'voor het aanpassen van de kolombreedten
  Cells.EntireColumn.AutoFit 'methode

  'sluiten van de objecten en verwijderen uit geheugen
  rs.Close
  dbNwind.Close
  Set rs = Nothing
  Set dbNwind = Nothing

  'communicatie met scherm weer aanzetten
  Application.ScreenUpdating = True
End Sub
```

- Test de code uit via **F8** en bekijk het resultaat in het **Excel**-sheet.
- Bewaar het **Excel**-werkboek onder de naam **AccessImport**.

10.3 Van Access naar Excel met ADO

Nu doen we hetzelfde maar dan met **ADO**

```
Sub AccessRead()
  'communicatie met scherm uitzetten
  Application.ScreenUpdating = False

  'Microsoft ActiveX Data Objects 2.x moet aangevinkt
  'via Extra ⇒ verwijzingen
  Dim strCnn As String
  Dim dbNwind As ADODB.Connection
  strCnn = "DRIVER={Microsoft Access Driver (*.mdb)};DBQ=c:\northwind.mdb"

  'driver Access ACCDB: Driver={Microsoft Access Driver (*.mdb, *.accdb)}

  'voor koppeling aan SQL Server
  'strCnn = "Provider=sqloledb; _
  '"Data Source=SERVERNAAM;Initial Catalog=pubs;User Id=sa;Password=; "

  Set dbNwind = New ADODB.Connection
  dbNwind.Open strCnn

  Dim rs As ADODB.Recordset
  Set rs = dbNwind.Execute("SELECT * FROM customers") 'methode
  Dim i As Integer, t As Integer
  t = 0
  rs.MoveFirst 'methode
  Range("a1").Select

  'voor het afdrukken van de veldnamen
  For i = 0 To rs.Fields.Count - 1
      ActiveCell.Offset(0, i).Value = rs.Fields(i).Name
  Next i
  i = 0
  Range("a2").Select

  'voor het afdrukken van de rijen
  Do Until rs.EOF
      For i = 0 To rs.Fields.Count - 1
        ActiveCell.Offset(t, i).Value = rs.Fields(i).Value
      Next i
      t = t + 1
      'recordpointer verplaatsen
      rs.MoveNext 'methode
  Loop

  'sluiten van de objecten en verwijderen uit geheugen
  rs.Close 'object sluiten
  dbNwind.Close
  Set rs = Nothing 'object opruimen
  Set dbNwind = Nothing

  'communicatie met scherm weer aanzetten
  Application.ScreenUpdating = True
End Sub
```

10.4 Van Excel naar Access

```
Sub vanExcelnaaraccess()
  Dim db As DAO.Database
  Set db = DAO.OpenDatabase("c:\hessel.mdb")

  Dim strSQL As String
  Dim naam As String, adres As String, woonplaats As String
  naam = Cells(ActiveCell.Row(), 1).Value
  adres = Cells(ActiveCell.Row(), 2).Value
  woonplaats = Cells(ActiveCell.Row(), 3).Value

  strSQL = "Insert into adressen(naam, adres, woonplaats) "
  strSQL = strSQL & " values ('" & naam & "','" & adres & "','" & woonplaats
& "')"

  db.Execute (strSQL) 'methode

  db.Close 'object sluiten
  Set db = Nothing 'object opruimen
End Sub
```

11 Van Excel naar Word

In het volgende voorbeeld wordt de inhoud van een aantal **Excel** cellen naar formuliervelden in **Word** overgebracht.

```
Sub Excelword()
  If Cells(ActiveCell.Row, 1).Value = "" Then
      MsgBox "Naam is leeg, sukkel"
  Else
      Dim wrd As Word.Application
      Set wrd = CreateObject("Word.Application")

      wrd.Visible = True 'eigenschap
      wrd.Activate 'methode
      wrd.WindowState = wdWindowStateMaximize 'eigenschap

      wrd.Documents.Add "c:\explainit\wordExcel.dotm" 'methode

      wrd.ActiveDocument.FormFields("naam").Result = Cells(ActiveCell.Row,
1).Value
      wrd.ActiveDocument.FormFields("adres").Result = Cells(ActiveCell.Row,
2).Value
      wrd.ActiveDocument.FormFields("woonplaats").Result = _
Cells(ActiveCell.Row, 3).Value

      wrd.ActiveDocument.SaveAs Filename:="c:\explainit\" & _
      Cells(ActiveCell.Row, 1).Value & _
      Cells(ActiveCell.Row, 4).Value + 1 & _
      ".docx"
      wrd.ActiveDocument.Close 'methode
      wrd.Quit 'methode
      Set wrd = Nothing 'object opruimen
      Cells(ActiveCell.Row, 4).Value = Cells(ActiveCell.Row, 4).Value + 1
  End If
End Sub
```

12 Email vanuit Excel

12.1 Inleiding

Er zijn verschillende manieren om vanuit **Excel** email te versturen met **VBA**. We kunnen de ***sendmail*** methode gebruiken voor direct verzenden; we komen dan in de e-mail applicatie (bijv. **Outlook**) terecht:

```
Workbooks(1).sendmail
```

We kunnen ook op de achtergrond Outlook aanroepen de mail zonder tussenkomst versturen:

```
'outlook aanroepen en mailtje maken
Set OutApp = CreateObject("Outlook.Application")
Set OutMail = OutApp.CreateItem(0)


'versturen mail
With OutMail
 .To = hessel@walmar.nl 'eigenschap
 .CC = "" 'eigenschap
 .BCC = "" 'eigenschap
 .Subject = "This is the Subject line" 'eigenschap
 .Body = "Hi there" 'eigenschap
 .Attachments.Add Dest.FullName 'methode
 .Send 'methode
End With
```

12.2 Voorbeeldcode van direct versturen

Onderstaand geven we een voorbeeld van code waarmee we een bestand vanuit **Excel** direct naar **Outlook** sturen waar we verder kunnen mailen.

```
Sub Mail_workbooksendmail()
   'werkt in 97-2010
   Dim wb As Workbook
   Set wb = ActiveWorkbook
   wb.SendMail "", "This is the Subject line"

   'onderstaande maakt e-mail direct klaar voor verzenden
   'wb.SendMail "hessel@walmar.nl", "This is the Subject line"
End Sub
```

12.3 Voorbeeldcode van emailen via Outlook

Onderstaand geven we een voorbeeld van code waarmee we een bestand vanuit **Excel** via Outlook kunnen mailen.

```
Sub Mail_Range()
  Dim rngSource As Range
  Dim wbDest As Workbook, wb As Workbook
  Dim strTempFilePath As String, strTempFileName As String,
  strFileExtStr As String
  Dim lngFileFormatNum As Long
  Dim OutApp As Object, OutMail As Object

  'Set rngSource = Nothing
  Set rngSource = ActiveSheet.UsedRange

  Set wb = ActiveWorkbook
  Set wbDest = Workbooks.Add(xlWBATWorksheet)
  rngSource.Copy
  wbDest.Sheets(1).Cells(1).Activate
  ActiveSheet.Paste
  Application.CutCopyMode = False

  'bepalen van naam en pad van tussenbestand
  strTempFilePath = Environ$("temp") & "\"
  'Environ$("temp") geeft de naam van de temp map
  strTempFileName = "Selection of " & wb.Name & " " _
                & Format(Now, "dd-mmm-yy h-mm-ss")

  If Val(Application.Version) < 12 Then
    strFileExtStr = ".xls": lngFileFormatNum = -4143
    'You use Excel 2000-2003
  Else
    strFileExtStr = ".xlsx": lngFileFormatNum = 51
    'You use Excel 2007-2010
  End If

  'outlook aanroepen en mailtje maken
  Set OutApp = CreateObject("Outlook.Application")
  Set OutMail = OutApp.CreateItem(0)

  'versturen mail
  With wbDest
    .SaveAs strTempFilePath & strTempFileName & strFileExtStr, _
      FileFormat:=lngFileFormatNum
    With OutMail
        .To = "hessel@walmar.nl"
        .CC = ""
        .BCC = ""
        .Subject = "This is the Subject line"
        .Body = "Hi there"
        .Attachments.Add wbDest.FullName
        .Send
    End With
    .Close SaveChanges:=False
  End With

  'verwijderen tussenbestand
  Kill strTempFilePath & strTempFileName & strFileExtStr

  'verwijderen objecten uit geheugen
  Set OutMail = Nothing
  Set OutApp = Nothing
End Sub
```

13 Excel VBA en bestandsbeheer

13.1 Inleiding

We kunnen **Excel VBA** ook gebruiken om iets te doen met onze bestanden.

13.2 Bestanden inlezen en openen

Het navolgende voorbeeld leest alle bestanden uit een de map D:\data en plaatst de namen in een **Excel** bestand.

```
Sub bestanden()
  Cells.ClearContents
  Cells(1, 1).Select
  Dim strBestanden As String
  Dim intT As Integer
  intT = 0

  'lees de bestanden uit de gegeven map
  strBestanden = Dir("d:\data\*.*")

  'loop door deze bestanden heen
  Do Until LenB(strBestanden) = 0
      ActiveCell.Offset(intT, 0).Value = strBestanden
      intT = intT + 1
      strBestanden = Dir() 'reset de variabele
  Loop
End Sub
```

Het navolgende voorbeelden opent ййn voor ййn de **Excel** bestanden uit de map D:\data die op .**xlsx** eindigen en sluit deze vervolgens weer.

```
Sub bestanden()
  Dim strBestanden As String
  Dim intT As Integer
  Dim wbWb As Workbook
  intT = 0

  strBestanden = Dir("d:\data\*.xlsx")

  Do Until LenB(strBestanden) = 0
      Set wbWb = Workbooks.Open("d:\data\" & strBestanden) 'methode
       'doe er iets mee
      wbWb.Close 'methode
      strBestanden = Dir() 'reset de variabele
  Loop
End Sub
```

Nog een voorbeeld: inhoud kopiлren uit meerdere **Excel** bestanden

```
Sub meerExcelbestanden()
  Cells.ClearContents
  Cells(1, 1).Select
  Dim strBestanden As String
  Dim intT As Integer
  intT = 0
  Dim exlWb As Workbook

  strBestanden = Dir("c:\explainit\Excel basis\*.xlsx")
  Application.ScreenUpdating = False
  Do Until LenB(strBestanden) = 0
      Set exlWb = Workbooks.Open("c:\explainit\Excel basis\" & strBestanden)
      'MsgBox exlWb.Worksheets(1).Cells(1, 1).Value
      'Exit Sub
      Workbooks("voorbeelden009.xlsm").Worksheets(1).Cells(1,
1).Offset(intT, 0).Value = _
      exlWb.Worksheets(1).Cells(1, 1).Value
      intT = intT + 1

      exlWb.Close False 'sluiten zonder opslaan
      strBestanden = Dir()

  Loop
  Application.ScreenUpdating = True
End Sub
```

13.3 Excel bestand wegschrijven naar een tekstbestand

```
Sub WriteToATextFile()
  Dim strFile, MyFile, fnum, strWholeLine As String
  Dim intI As Integer, intT As Integer, intRow As Integer, intCol As Integer
  Dim rngR as Range, rngC As Range

  'first set a string which contains the path to the file
  'you want to create.
  'this example creates one and stores it in the root directory

  strFile = InputBox("name of the file")
  MyFile = "d:\data\tx" & strFile & ".txt"
  'set and open file for output
  fnum = FreeFile()
  Open MyFile For Output As fnum

  Set rngR = ActiveSheet.UsedRange
  rngR.Offset(0, 0).Select

  intRow = rngR.Rows.Count
  intCol = rngR.Columns.Count

  For intT = 0 To intRow - 1
      For intI = 0 To intCol - 1
          strWholeLine = strWholeLine & ActiveCell.Offset(intT, intI).Value
& ";"
      Next intI
      'write the line
      Print #fnum, strWholeLine
      'reset
      strWholeLine = ""
  Next t

  'use Write when you want the string with quotation marks
  'Write #fnum, "I printed this"
  Close #fnum
  Range("a1").Select
```

```
End Sub
```

13.4 FileSystemObject gebruiken om bestanden uit zips te kopiʌren

```
Sub zipcopier()
  Dim fs As Object, f As Object
  Dim FileInFolder As Object, sh As Object, ZipFile As Object, fileInZip As
Object
  Dim FolderTo As Variant, FolderFrom As Variant

  FolderTo = "d:\data\end"
  FolderFrom = "d:\data\start"
  Set fs = CreateObject("Scripting.FileSystemObject")
  Set f = fs.GetFolder(FolderFrom)

  'procedure for zip-files
  For Each FileInFolder In f.Files
      If Right(FileInFolder.Name, 4) = ".zip" Then
          Set sh = CreateObject("Shell.Application")
          Set ZipFile = sh.Namespace(f & "\" & FileInFolder.Name)
          For Each fileInZip In ZipFile.items
              'Create folder if not there already
              If Not fs.FolderExists(FolderTo) Then
                  fs.CreateFolder (FolderTo)
              End If


              'copies files from zip to destination one by one
              If Not fs.fileexists(FolderTo & "\" & Mid(fileInZip.Path,
InStrRev(fileInZip.Path, "\") + 1)) Then
                 sh.Namespace(FolderTo).moveHere (fileInZip.Path)
              End If

          Next
      End If
  Next
End Sub
```

13.5 Opgaven

- Maak een routine die uit een bepaalde directory alle .**XLSX** bestanden opent en sluit en van allemaal de waarden uit de eerste kolom bij elkaar telt; op het eind willen we de einduitslag via een **MSGBOX** zien
  tip: gebruik **UsedRange** en **WORKSHEETFUNCTION.SUM**()

14 Foutafhandeling

14.1 Inleiding

De foutafhandeling verloopt in grote lijnen net zo als in **Word** en **Access**. In hoofdstuk 24 kunnen we hier meer over lezen. In dit hoofdstuk zullen we alleen een korte samenvatting geven.

14.2 Algemene VBA-foutafhandelding

Fouten in code zijn op te sporen via het statement **ON ERROR**. Voorbeelden zijn:

```
On error resume next
On error goto 0
On error goto Foutje
```

Een volledig voorbeeld ziet er zo uit:

```
Sub Foutje()
  On Error GoTo Err_Foutje

  Dim intT As Integer
  For intT = 1 To 10
    MsgBox 10 / (5 - intT)
  Next intT
  Exit Sub

  Err_Foutje:
  Select Case Err.Number
    Case 11
      intT = intT + 1
      Resume
    Case Else
      MsgBox Err.Number & " " & Err.Description
  End Select
End Sub
```

15 Performance

15.1 Functionaliteit aan- en uitzetten

Als we in de **VBA** code bepaalde **Excel** functionaliteit uitzetten, wordt onze code sneller:

```
Application.ScreenUpdating = False
Application.DisplayStatusBar = False
Application.Calculation = xlCalculationManual
Application.EnableEvents = False
ActiveSheet.DisplayPageBreaks = False
```

We moeten er uiteraard wel aan denken het een en ander weer aan te zetten:

```
Application.ScreenUpdating = True
Application.DisplayStatusBar = True
Application.Calculation = xlCalculationAutomatic
Application.EnableEvents = True
ActiveSheet.DisplayPageBreaks = True
```

15.2 Diversen

Declareren van variabelen met expliciete types voorkomt de overhead, veroorzaakt door het steeds opnieuw moeten bepalen van het datatype.

Als het om simpele functies gaat, kunnen we ze beter in **VBA** zelf maken in plaats van **WorksheetFunction** object te gebruiken.

16 Oplossingen bij de opgaven

16.1 Oplossingen van paragraaf 3.5

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
  If Target.Value >= 1 And Target.Value <= 56 Then
      Worksheets(1).Range("a1:a8").Interior.ColorIndex = Target.Value
  End If
End Sub
```

```
Sub TabsKleuren()
  Dim intTeller As Integer
  For intTeller = 1 To 53
      Worksheets(intTeller).Tab.ColorIndex = intTeller
  Next
End Sub
```

```
Sub verbergen()
  Application.ScreenUpdating = False
  Dim shtX As Worksheet
  For Each shtX In Sheets
      If shtX.Name <> ActiveSheet.Name Then
          shtX.Visible = False
      End If
  Next
  Application.ScreenUpdating = True
End Sub
```

16.2 Oplossingen van paragraaf 4.3

```
Sub total()
  Dim rngCell as Range, rngCell2 As Range
  Dim strCompany As String
  Dim dblTotal as Double, dblTotal2 As Double
  Dim shtSheet As Worksheet

  Worksheets(Worksheets.Count).Activate

  For Each rngCell In ActiveSheet.UsedRange.Columns(1).Cells
    If rngCell.Value <> "" Then
      strCompany = rngCell.Value

      'go to the first sheet
      For Each shtSheet In Worksheets
        If shtSheet.Name = Worksheets(Worksheets.Count).Name Then
            Exit For
        End If
        shtSheet.Activate

        For Each rngCell2 In ActiveSheet.UsedRange.Columns(1).Cells
          If InStr(UCase(rngCell2.Value), UCase(strCompany)) > 0 Then
            dblTotal = dblTotal + rngCell2.Offset(0, 1)
            dblTotal2 = dblTotal2 + rngCell2.Offset(0, 2)
          End If
        Next
      Next
      Worksheets(Worksheets.Count).Activate
      rngCell.Offset(0, 1).Value = dblTotal
      rngCell.Offset(0, 2).Value = dblTotal2
    End If
    dblTotal = 0
    dblTotal2 = 0
  Next
End Sub
```

```
Sub randomcoloring()
  Dim intTeller As Integer
  Dim rngC As Range

  'remove colors
  ActiveSheet.UsedRange.Interior.ColorIndex = xlNone

  For intTeller = 1 To 10
      For Each rngC In ActiveSheet.UsedRange
          'paint each cell in the used range
          rngC.Interior.Color = RGB(Rnd()* 256, Rnd() * 256, Rnd() * 256)
          'rnd() generates a number between 0 and 1
      Next
  Next intT
End Sub
```

```
Sub copyusedrangeplakken()
  Worksheets(1).UsedRange.Copy
  Dim intRij As Integer
  intRij = Worksheets(2).UsedRange.Rows.Count
  Worksheets(2).UsedRange.Rows(intRij+1).Columns(1).PasteSpecial
End Sub
```

```
Sub SelectUsedRangeLessTopRow()
  Dim intRij As Integer, intKolom As Integer
  With ActiveSheet.UsedRange
      intRij = .Rows.Count
      intKolom = .Columns.Count
      .Range(Cells(2, 1), Cells(intR, intC)).Copy
  End With

  intR = Worksheets(2).UsedRange.Rows.Count
  Worksheets(2).UsedRange.Rows(intR+1).Columns(1).PasteSpecial
End Sub
```

Of:

```
Sub SelectUsedRangeLessTopRow()
  Dim intRij As Integer, intKolom As Integer
  With ActiveSheet.UsedRange
      intRij = .Rows.Count - 1
      intKolom = .Columns.Count
      .Offset(1, 0).Resize(intRij, intKolom).Copy
  End With

  intR = Worksheets(2).UsedRange.Rows.Count
  Worksheets(2). UsedRange.Rows(intR+1).Columns(1).PasteSpecial
End Sub
```

```
Sub rijWissen()
  Worksheets(1).Activate
  Dim intTeller As Integer
  For intTeller = ActiveSheet.UsedRange.Rows.Count To 1 Step -1
    If WorksheetFunction.CountA(ActiveSheet.UsedRange.Rows(intTeller)) = 0
Then
       ActiveSheet.UsedRange.Rows(intTeller).Delete
    End If
  Next
End Sub
```

```
Sub total()
  Worksheets(Worksheets.Count).Select
  Dim arrTot()
  Dim rngCell As Range, rngCell2 As Range
  Dim intT As Integer, intA As Integer, intC As Integer
  intC = Worksheets(Worksheets.Count).UsedRange.Columns.Count - 1
  ReDim arrTot(intC)

  For Each rngCell In Worksheets(Worksheets.Count).UsedRange.Columns(1)
    If rngCell.Value <> "" Then
      For intT = 1 To Worksheets.Count - 1
          For Each rngCell2 In Worksheets(intT).UsedRange.Columns(1).Cells
            If InStr(UCase(rngCell2.Value), UCase(rngCell.Value)) > 0 Then
              For intA = 0 To intC
                arrTot(intA) = arrTot(intA) + rngCell2.Offset(0, intA).Value
              Next
            End If
          Next
      Next
      For intA = 1 To intC
          rngCell.Offset(0, intA).Value = arrTot (intA)
      Next
    End If
  ReDim arrTot (intC)
  Next
End Sub
```

16.3 Oplossingen van paragraaf 7.4

```
Function Celsius(Fahrenheit As String) As Variant
   If Not IsNumeric(Fahrenheit) Then
       Celsius = "geen waarde"
   Else
       Celsius = CDbl((5 / 9) * (Fahrenheit - 32))
   End If
End Function
```

```
Public Function ElevenProof(Account As String) As String
   Dim intTeller As Integer, intSom As Integer, intRest As Integer

   If Len(Account) = 9 Then
       For intTeller = 9 To 1 Step -1
           intSom = intSom + (intTeller * Val(Mid(Account, (10 - intTeller),
1)))
       Next i

       intRest = intSom Mod 11 'modulus: rest van de deling
       If intRest = 0 Then
           ElevenProof = "correct"
       Else
           ElevenProof = "incorrect"
       End If
   Else
       ElevenProof = "nummer is te lang of te kort"
       Exit Function
   End If

End Function
```

```
Function weekNummerNL(datum As Date) As String
  Dim intWeeknr As Integer
  If IsDate(datum) Then
      intWeeknr = DatePart("ww", datum, 2, 2)
      weekNummerNL = "week " & Right("0" & intWeeknr, 2)
  Else
      weekNummerNL = "datum is fout"
  End If
End Function
```

```
Function Schrikkeljaar(Jaar As Integer) As Boolean
  Dim intUitKomst As Integer
  intUitKomst = Jaar Mod 4

  If intUitKomst = 0 Then
      Schrikkeljaar = True
  End If

  If Jaar Mod 100 = 0 Then
      Schrikkeljaar = False
  End If

  If Jaar Mod 400 = 0 Then
      Schrikkeljaar = True
  End If
End Function
```

```
Function leeftijd(datum)
  If Not IsDate(datum) Then
      leeftijd = ""
      Exit Function
  End If

  If Month(Date) > Month(datum) Then
      leeftijd = Year(Date) - Year(datum)
  Else
      If Month(Date) < Month(datum) Then
          leeftijd = Year(Date) - Year(datum) - 1
      Else
          If Day(Date) < Day(datum) Then
              leeftijd = Year(Date) - Year(datum) - 1
          Else
              leeftijd = Year(Date) - Year(datum)
          End If
      End If
  End If
End Function
```

16.4 Oplossingen van paragraaf 13.3

```
Sub bestanden()
  'zet meldingen naar het scherm uit
  Application.DisplayAlerts = False
  Application.ScreenUpdating = False

  Dim strBestanden As String
  Dim intTeller As Integer
  Dim wkbWb As Workbook, wksWs As Worksheet
  Dim dblTotaal As Double

  intTeller = 0
  strBestanden = Dir("d:\data\*.xlsx")
  Do Until LenB(strBestanden) = 0
    Set wkbWb = Workbooks.Open("d:\data\" & strBestanden)
      For Each wksWs In Worksheets
      dblTotaal = dblTotaal + _
      WorksheetFunction.Sum(wksWs.UsedRange.Columns(1))
      Next
      wkbWb.Close
      strBestanden = Dir() 'reset de variabele
  Loop
  'zet meldingen naar het scherm aan
  Application.DisplayAlerts = True
  Application.ScreenUpdating = True
  MsgBox dblTotaal
End Sub
```

17 Wat is Visual Basic For Applications

17.1 Achtergrond

Visual Basic for Applications, kortweg **VBA** genoemd, is een afgeleide van de programmeertaal **Visual Basic**. Deze taal is ontstaan uit de taal **Basic** (wellicht roept dat bij sommigen herinneringen op: in deze taal kon bijvoorbeeld op de Commodore 64 worden geprogrammeerd, spelletjes of andere programma's) en is in de loop der jaren uitgebreid tot de taal van nu. De taal **Visual Basic** is in 1992 op de markt verschenen, waarna in 1994 de taal **Visual Basic for Applications** het licht zag. Pas in 1997, met het verschijnen van **Microsoft Office 97**, is **VBA** volledig geïntegreerd in het Office pakket. Daarvyyr had elk programma een eigen, aparte, basic macrotaal.

17.2 VBA is meer dan een macrotaal

In de eerste versies van **Excel** en **Word** was het al mogelijk om met behulp van de basic talen bepaalde handelingen als het ware op te nemen, om later weer *geautomatiseerd* af te laten spelen. Dit kwam de snelheid van werken natuurlijk zeer ten goede. Helaas was de integratie en daarmee standaardisatie ver te zoeken. Met de introductie van **VBA** zijn tevens een aantal functionaliteiten toegevoegd die **VBA** tot een ware programmeertaal maken. Zo zijn er nu standaard besturingsfuncties opgenomen (If…Then…Else, Do…Loops, For…Next en dergelijke). In **VBA** zijn geavanceerde voorzieningen getroffen om fouten in een programma op te sporen (debugging). Bovendien vindt het schrijven van code plaats in een speciaal daarvoor ingerichte editor, waarmee het mogelijk is op een gestructureerde wijze code te schrijven. Dit komt de overzichtelijkheid en dus het onderhoud van code zeer ten goede.

Zoals de naam al doet vermoeden richt **VB(A)** zich met name op het grafische aspect van een programma: de gebruikers-interface (**Graphical User Interface** of **GUI**): dit gedeelte van een programma is in vergelijking met andere talen snel en eenvoudig op te zetten. Ook het schrijven van code gaat relatief eenvoudig: dit lijkt soms veel weg te hebben van *verkort Engels*. Tevens is de methode van werken, in tegenstelling tot veel andere talen, vrij eenvoudig te noemen (vandaar ook de term *basic*). Dit houdt echter niet in dat Visual Basic (for Applications) een beperkte taal is: integendeel, we zullen snel ontdekken hoe krachtig deze taal kan zijn. Met behulp van **VBA** is het mogelijk functionaliteit toe te voegen aan de Microsoft Office producten. Met enige kennis van zaken, kan zo'n opgenomen ***macro*** al snel uitgroeien tot een volledig zelfstandig functionerend programma (en kan bijvoorbeeld de hele **Excel** of **Word** omgeving zijn aangepast).

In **Access** kunnen helaas geen macro's worden opgenomen. Macro's in **Access** bestaan uit reeksen commando's die in een speciaal scherm moeten worden ingevoerd. Deze macro's zijn ook niet standaard in **VBA** gecodeerd, maar kunnen wel worden geconverteerd naar **VBA**-code. Macro's en **VBA** zijn in **Access** eigenlijk twee verschillende dingen!

Niet alleen in Microsoft Office wordt **VBA** gebruikt. Juist vanwege de eenvoud van deze taal en de snelheid waarmee een en ander kan worden gebouwd, ziet men deze taal ook terug in bijvoorbeeld **AutoCAD**: een geavanceerd technisch tekenprogramma van AutoDesk.

18 Basisbegrippen VBA

18.1 VBA terminologie

Voor een helder basis inzicht in de **VBA** taal lichten we eerst een aantal Engelstalige termen toe. Deze laten we later uitgebreider aan bod komen.

18.1.1 Algemene termen

Term	Omschrijving
Design-Time	De ontwerpfase van een applicatie, de tijd die wordt doorgebracht in de ontwikkelomgeving van **VBA**.
Run-time	De tijd dat de applicatie wordt uitgevoerd. De applicatie gedraagt zich nu zoals de gebruiker zal ervaren.
UserForms	De grafische voorstelling van de applicatie (formulieren) die gebruikt worden om informatie van een gebruiker te verkrijgen.
Controls	De grafische gereedschappen die op het formulier geplaatst kunnen worden, zoals textboxes, buttons. Hiermee communiceert de gebruiker met de applicatie.
Objects	Een algemene verzamel term die wordt gebruikt om de zaken die het programma maken te beschrijven, zoals UserForms, Controls.
Properties	Kenmerken/eigenschappen van een object. (kleur, lengte, opschrift).
Methods	Acties die door het object kunnen worden uitgevoerd (toevoegen van een werkblad).
Events	Gebeurtenissen (van buitenaf) die door het object worden herkend (het drukken op een knop door de gebruiker).

Het is belangrijk te beseffen dat **VB(A)** in ййn belangrijk opzicht afwijkt van overige (oudere) programmeertalen zoals **Pascal**, **C**. Deze talen zijn ***procedureel*** opgebouwd, wat inhoudt dat het programma de code stap voor stap (dat wil zeggen: regel voor regel) uitvoert. Wel kan vaak naar andere delen van het programma worden *gesprongen*, of kunnen bepaalde regels code op basis van een voorwaarde worden overgeslagen. Dit brengt vaak uitgebreide stukken code met zich mee, elke mogelijke situatie moet immers worden afgevangen.

Visual Basic (en dus yyk **VBA**) is echter gebeurtenis gestuurd (**event-driven**). Dit houdt in dat de applicatie in staat is te reageren op bepaalde gebeurtenissen van buitenaf. Bijvoorbeeld een gebruiker die op de **OK** knop klikt. Pas dбn wordt de code die bij die gebeurtenis van die knop hoort, uitgevoerd. Gebeurtenissen (**events**) kunnen evenwel ook door het besturingssysteem of door andere code in de applicatie worden opgeroepen. Het programma is hierdoor in staat zich veel flexibeler voor de gebruiker op te stellen. De gebruiker kan nu bijvoorbeeld informatie in een willekeurige volgorde op het formulier invoeren.

18.1.2 Bestandstypen

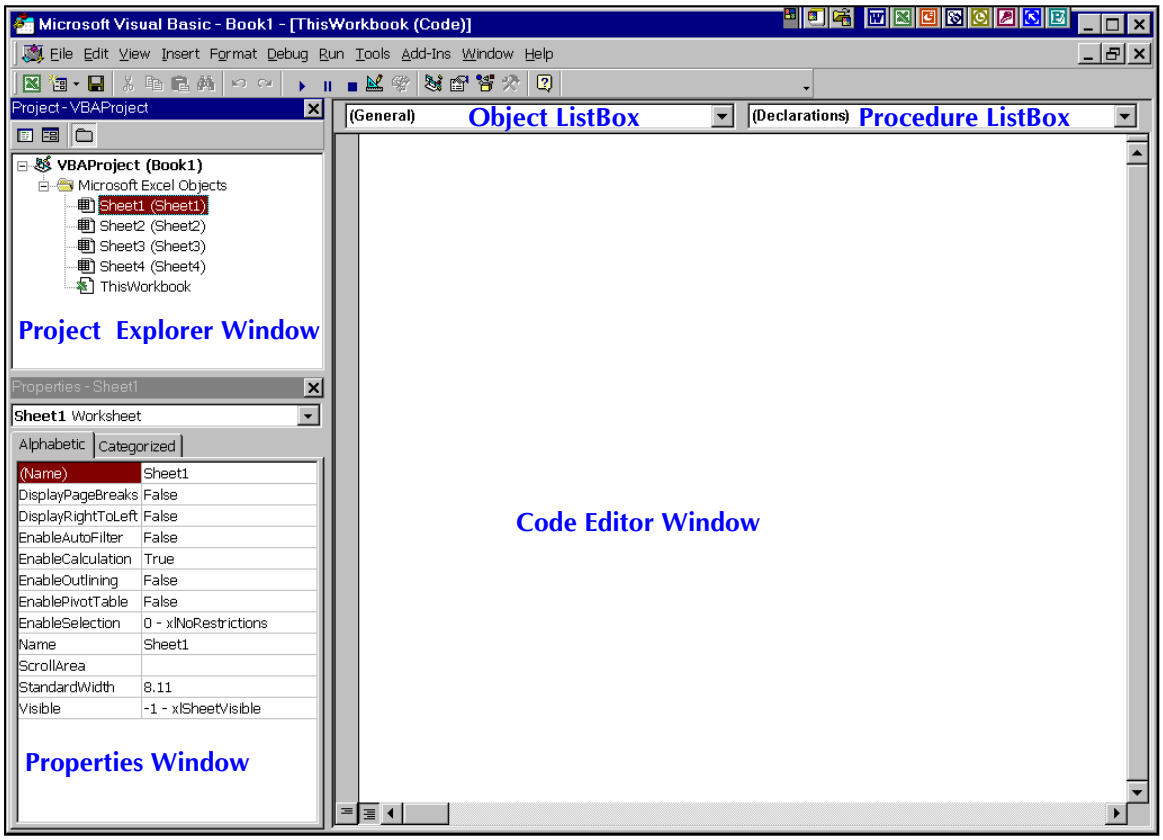
De Visual Basic Editor geeft ons de mogelijkheid om allerlei functionaliteit toe te voegen aan onze Office applicatie. Deze informatie over bijvoorbeeld formulieren wordt opgeslagen in het Office document zelf. Het is echter ook mogelijk formulieren en dergelijke te exporteren en apart als bestand op te slaan. We kunnen deze bestanden dan later weer hergebruiken in andere Office applicaties.

Hieronder volgt een lijst met de bestanden en hun extensies.

Bestand	Extensie	Omschrijving
UserForms	.frm	Bevat alle code en instellingen (properties) van het formulier.
	.frx	Bevat alle grafische elementen van het formulier.
Standard Modules	.bas	Hierin staan de algemene functionaliteiten van de applicatie (zoals functies, procedures, constanten), die overal vanuit de applicatie aangeroepen moeten kunnen worden.
Class Modules	.cls	Hierin staat de definitie van het te maken object. Bevat properties, methods, events, errors.

18.2 De Visual Basic Editor

De Visual Basic Editor is op te roepen via **Tools, Macro, Visual Basic Editor (ALT F11)** in versie 2003 of via de ribbon/lint Developer/Ontwikkelaar in latere versies.



In de linker bovenhoek vinden we de zogenaamde **Project Explorer** waarin de benodigde bestanden zijn te vinden waaruit de applicatie is opgebouwd. Dat kunnen bijvoorbeeld rekenbladen (sheets) zijn, maar ook ***modules***, ***classes*** en ***userforms***.

In de linker onderhoek vinden we de zogenaamde **Properties Window** waarin de kenmerken/eigenschappen van een geselecteerd object staan vermeld. We zullen merken dat elk object zijn eigen kenmerken bezit. Het object is te kiezen door een keuze te maken uit de ***listbox*** (ook in het properties window).

Aan de rechterkant is een groter scherm zichtbaar: de **Code Editor**. Dit is de plaats waar we onze code zullen gaan schrijven (ontwerpfase/designtime).

18.3 Objecten

Een object is een combinatie van code, data en eventueel een grafische schil die we als een zelfstandig functionerend geheel kunnen gebruiken en aansturen. Denk bijvoorbeeld aan een **Excel** werkboek: hierin bevindt zich alle informatie en functionaliteit die een gebruiker nodig heeft om met **Excel** te kunnen werken. Het werkboek bevat echter ook weer subobjecten: de rekenbladen zelf, eventuele grafieken op het rekenblad, de diverse cellen en ga zo maar door.

In **VBA** zijn er nog veel meer objecten. Besef dat ze volledige functionaliteit bezitten. Bijvoorbeeld een knop, waarin de verschillende gebeurtenissen waarop de knop kan reageren al in zijn meegebakken. De **Object Browser**, op te roepen via **View, Object Browser (F2)** geeft een overzicht van de diverse objecten in Office. Hierover later meer.

Aangezien de applicatie tot leven moet komen, moeten we code schrijven waarmee we de objecten kunnen besturen. We zullen begrijpen dat de communicatie tussen het grafische gedeelte (daar waar de objecten op zijn geplaatst, dat kan zowel op een formulier als op een werkblad zijn!) en het code gedeelte (zie de code editor) van essentieel belang is. Weten welk object in de code wordt bedoeld, verhoogt de leesbaarheid en verkleint de kans op fouten. We zorgen er dus voor dat we consequent zijn in de naamgeving. Gelukkig heeft elk object een *name* property, zodat elk object uniek is te identificeren.

Om direct op te kunnen maken, om wat voor object het gaat, is er een naamconventie afgesproken: de Hongaarse notatie. Dit houdt in dat elk object een voorzetsel krijgt bestaande uit bijvoorbeeld drie letters, gevolgd door een eigen gekozen naam (beginnend met een hoofdletter). Hieronder volgt een klein gedeelte van die lijst.

Object	Voorzetsel	Voorbeeld
Form	frm	frmKlantGegevens
Label	lbl	lblHulpBericht
TextBox	txt	txtGebruikersNaam
ListBox	lst	lstLanden
ComboBox	cbo	cboAutos
OptionButtons	opt	optTermijn
Checkbox	chk	chkPapier

Voor het geven van namen (aan objecten, variabelen en dergelijke) gelden de volgende richtlijnen:

- De naam moet beginnen met een letter.
- De tekens **. , ! @ & $ #** mogen in de naam niet voorkomen, net zo min als de spatie.
- Namen mogen niet lager zijn dan 255 tekens.
- De naam mag niet al door VB(A) in gebruik zijn.
- De naam mag niet al eerder zijn gedefinieerd in hetzelfde bereik (scope).

Visual Basic for Applications is niet hoofdlettergevoelig (*case-sensitive)*, maar zal wel de wijze van (hoofdletter)gebruik van het moment van aanmaken bewaren.

18.4 Properties

Onder properties van een object worden de eigenschappen of kenmerken van dat object verstaan. Zo kunnen we bijvoorbeeld denken aan de naam van een werkblad (sheet), of het lettertype van een bepaalde cel. Properties kunnen zowel worden ingesteld als worden opgevraagd.

18.4.1 Instellen van properties in Design Time

Het instellen van properties in design time gebeurt via het properties window in de **VBA Code Editor**. Via het keuzelijstje kan het juiste object worden gekozen (mits het in design time bekend is), bijvoorbeeld de knop op een formulier. Kies vervolgens de gewenste property. Denk aan de *name* property van elk object. Dit kunnen we het beste maar zo snel mogelijk in orde maken; als we dat later doen, bestaat de kans dat we de gehele code dient aan te passen. Dit doet **VBA** namelijk niet zelf.

Om snel naar een bepaalde property toe te gaan dienen we de combinatie **[CTRL]-[SHFT]** ingedrukt te houden, terwijl we vervolgens de eerste letter van de property indrukken.

We kunnen hulp vragen over een bepaalde property door deze te selecteren en op **[F1]** te drukken.

18.4.2 Instellen van properties in Run Time

Het instellen van properties die gedurende de uitvoering van het programma veranderen zal moeten gebeuren via code. Ook kunnen waarden aan properties worden toegekend die voor de rest binnen het programma niet veranderen (en daarmee is eigenlijk hetzelfde bereikt als het instellen van een property via het property window). Dat kunnen we bijvoorbeeld doen wanneer we properties gebruiken die andere programmeurs niet verwachten, of omdat we onze code duidelijker wil opbouwen. Wees er op bedacht dat niet alle properties zijn in te stellen, sommige zijn namelijk alleen maar op te vragen. Ook in dit geval krijgen we hulp: bij properties die in design time bekend zijn, krijgen we een uitklaplijstje te zien. We kunnen hieruit de property kiezen door de eerste paar letters ervan in te tikken. Geven we een spatie dan krijgen we het woord, gevolgd door een spatie. Een tab zet alleen het woord neer en met **enter** gaan we meteen naar de volgende regel toe.

De syntax is als volgt:

```
Object.Property = Value
```

Voorbeeld*:*

```
ActiveCell.Value = "Hello world"
ActiveCell.HorizontalAlignment = xlHAlignRight
```

Het uitklaplijstje is niet altijd beschikbaar. We kunnen ook proberen het lijstje op te roepen (na de punt) met **CTRL J**. Voor het tonen voor een uitklaplijst van de objecten drukken we op **CTRL SPATIE**.

18.4.3 Meerdere waarden van properties gelijk instellen

Wanneer we nu van een bepaald object meer properties moeten instellen, kunnen we natuurlijk elke keer het object herhalen, gevolgd door de gewenste property. Dit kost echter veel typwerk, komt de leesbaarheid niet ten goede en maakt bovendien de code langzamer omdat elke keer weer in het object gedoken moet worden. Sneller is het, gebruik te maken van **With…End With** statements.

Voorbeeld*:*

```
With Worksheets("Sheet1").Range("B5")
   .Value = "Hello World"

   With .Font
        .Size = 14
        .Bold = True
        .Italic = True
        .Color = vbRed
   End With

End With
```

Denk aan de punt voor de properties!

18.4.4 Het opvragen van properties in Run Time

Wanneer we waarden aan properties kan toekennen, kunnen we ze uiteraard ook opvragen. Bijvoorbeeld om te achterhalen wat een gebruiker als wachtwoord heeft ingetikt… De syntax is bijna identiek aan dat van het instellen (let op de positie van het = teken – kenmerkend voor properties – ten opzichte van de property):

```
Variabele = Object.Property
```

Voorbeeld*:*

```
strPassword = ActiveCell.Value
lngNumberOfCells = ActiveSheet.Cells.Count
```

18.5 Methods

Methods zijn acties die door het object zelf kunnen worden uitgevoerd. Bijvoorbeeld het opslaan van een werkboek, het toevoegen van rijen of kolommen…Sommige methods hebben argumenten nodig, waarvan een aantal optioneel kan zijn. Syntax is als volgt:

```
Object.Method [argument1, …]
```

Voorbeeld*:*

```
Application.Calculate     = zonder argumenten
ActiveWorkbook.SaveAs ("C:\My Documents\Testje.xls")    = met argumenten
```

18.6 Events

Events zijn gebeurtenissen die door het object worden herkend. Deze gebeurtenissen zijn altijd afkomstig van buiten af (vanuit het object gezien), bijvoorbeeld omdat deze door een gebruiker worden opgeroepen (een *click* event op het object *cmdOK*) of door het besturingssysteem zelf. Tevens kan een event vanuit de applicatie worden opgeroepen (via code). Door code te schrijven voor deze events krijgen we een interactief geheel. Zo kunnen we code voor de ***open*** event van het werkboek schrijven, die we elke keer bij het openen een plezierige **Excel** sessie wensen. Syntax is als volgt:

```
Private Sub Object_Event()
  'code voor plezierige voortzetting komt hier.
End Sub
```

Voorbeeld:

```
Private Sub Workbook_Open()
  Msgbox "Een plezierige voortzetting gewenst!"
End Sub
```

Een kenmerk van event procedures is dat deze een *underscore* (liggend streepje) hebben in de naam van de procedure. We krijgen een event procedure door in de **ObjectListBox** het gewenste object te kiezen (bijvoorbeeld workbook), en in de **ProcedureListBox** te kiezen voor de gewenste event (open). Dat kan soms een hele moeilijke keuze zijn, omdat we precies op het juiste moment (lees: gebeurtenis) een bepaalde handeling of instelling willen laten optreden.

Let op de Sub…End sub constructie. Deze wordt automatisch voor ons door **VBA** geplaatst.

18.7 Overzicht objecten, properties, methods en events

Hoewel in de voorgaande paragrafen de kenmerken en de verschillen van de bouwstenen van VB(A) zijn genoemd, kan het soms even duren voordat we echt aanvoelen, ervaren wat het nu allemaal inhoudt.

Ter verduidelijking het volgende voorbeeld:

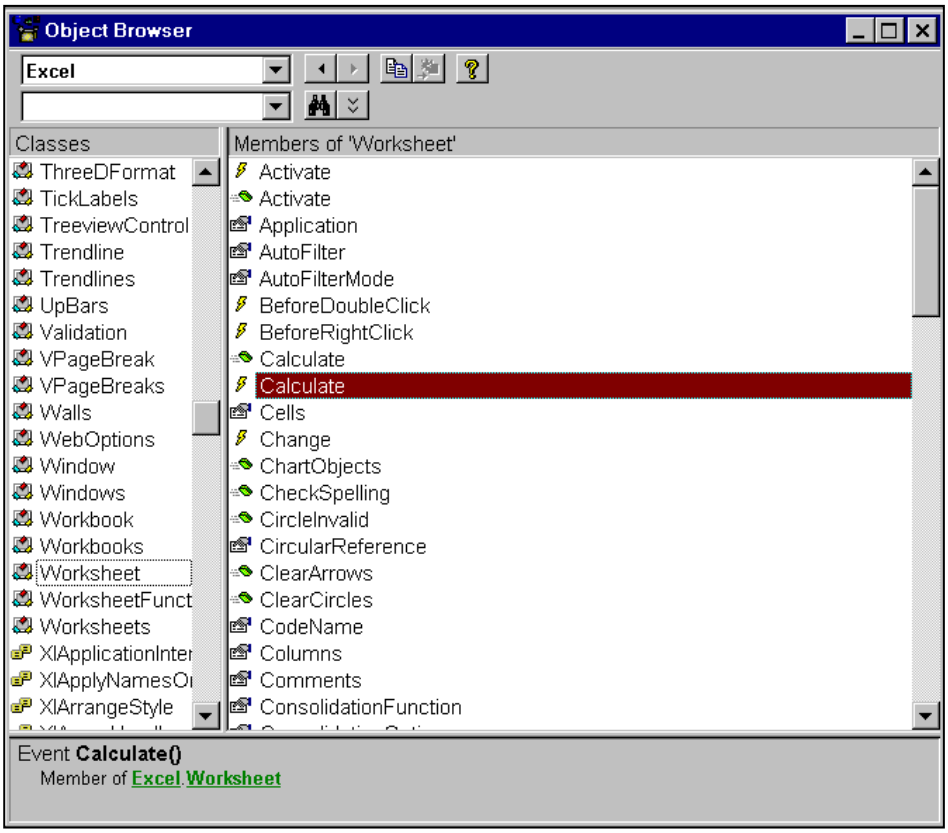
Beschouw de docent als object. Als het goed is, heeft deze docent bepaalde functionaliteiten: bepaalde kenmerken zoals haarkleur, lengte, schoenmaat e.d. Ook heeft de docent bepaald gedrag: de docent deelt boeken uit (geen strafwerk), beantwoordt vragen, legt uit en kan reageren op gebeurtenissen van buitenaf (als het goed is) zoals het stellen van vragen door een cursist (maar de cursist is weer een ander object!), het rinkelen van de pauzebel en noem maar op.

Stel, een **badgast** ligt lekker in de zon te bakken. Na verloop van tijd wordt deze badgast enigszins rood. Maar…jawel, deze zonaanbidder reageert op die zonnestralen door actie te ondernemen haalt dus zonnebrandcrиme factor 30 en smeert zich in.

Object: Badgast
Property: Huidskleur
(Instellingswaarde: Rood)
Method: HalenZonnebrandCrиme
(Argument: Factor 30)
Event: Badgast_SchijntFel (!)

18.8 De Object Browser

Met behulp van de objectbrowser, via **View, Object Browser (F2)** op te roepen, kunnen we inzicht krijgen in de diverse objecten in **Excel** en de daarbij horende properties, methods en events. We kunnen de objectbrowser ook gebruiken om te achterhalen of iets nu een object, property, method of event is:



- Objecten: Een grijs vlakje met daarop een driehoek
- Properties: een handje dat een blaadje vasthoudt
- Methods: een gummetje met gumstreepjes
- Events: een bliksemflits

We kunnen bovenaan de Objectbrowser voor de gewenste *library* kiezen: dit is de bibliotheek waar de diverse functies in staan omschreven. Tevens kunnen we aan de rechterkant een method, property of event uit kiezen. Via de knop met het vraagteken kunnen we ook op deze manier informatie (met eventueel voorbeeldcode) opvragen.

Bovendien kunnen we, door gebruik te maken van de Object Browser, inzicht krijgen in het object model van **Excel**: de structuur van hoofdobjecten en subobjecten (probeer via **Help** het objectenmodel van **Excel** eens te vinden).

19 Het schrijven van code

19.1 De plaats

De code in een **VBA** applicatie kan op verschillende plaatsen worden ondergebracht, echter altijd op een zogenaamde **module**. Dit is in feite niet meer dan een tekstverwerker (vergelijkbaar met Word), die speciaal geschikt gemaakt is voor programmeer werkzaamheden. In een module kunnen zaken voorkomens zoals:

- Het declareren van constanten, variabelen
- Het toekennen van waarden aan properties
- Diverse procedures en functies
- Commentaar

Echter, er zijn diverse verschijningsvormen van modules, te weten:

- Document modules
- UserForm modules
- General Modules
- Class Modules

Deze plaatsen zullen we stap voor stap behandelen, met daarbij de kenmerken van elke plaats. Welke moeten we nu gebruiken?

19.1.1 Document Modules

Deze zijn in principe bedoeld om code neer te zetten die specifiek voor een bepaald document is bedoeld. We kunnen in **Excel** bijvoorbeeld gebruik maken van het object *Worksheet* (objectenlijst) en de daarbij horende events. Bijvoorbeeld de *Activate* event, om de gebruiker bij het kiezen van het werkblad te verblijden met een bericht.

Tevens kunnen we in **Excel** gebruik maken van **ThisWorkbook** (in de Project Explorer). Wanneer we hier op dubbelklikken, kunnen we in de **Code Editor** kiezen uit het object ***Workbook***. Zoals we kunnen zien, zijn er nu ook weer andere events te selecteren.

19.1.2 UserForm modules

Over het algemeen wordt in een **UserForm** module puur code gezet die betrekking heeft op de grafische elementen op dat zelfde **form** (formulier). Hiermee is het dan immers mogelijk geworden de communicatie tussen de controls op het vorm en de bijhorende code te regelen. In een UserForm kan weliswaar ook code voorkomen die niet direct betrekking heeft op een grafisch object (bijvoorbeeld een opslaan functionaliteit), maar liever wordt dat soort code dan in een General Module gezet (zie volgende paragraaf). Voorop blijft staan de *herbruikbaarheid* van de code: als programmeur willen we immers zo min mogelijk hoeven te kopiʌren. Mochten we er later achter komen dat we geschreven code vaker (op andere forms) ook nodig hebben, verplaats het dan naar een **General Module**.

Het toevoegen gaat via **Insert, UserForm**. Met de rechtermuisknop in de Project Explorer gaat dit wat sneller.

19.1.3 General Modules

Een General Module (kort ook wel gewoon **module** genoemd) bevat in principe alle code voor de macro's die zijn opgenomen. Echter, wanneer we dat nog niet hebben gedaan, dienen we deze eerst toe te voegen via **Insert, Module**. Aangezien er hier geen grafische elementen aanwezig zijn (zoals op de UserForm), zijn er in de ObjectListBox gййn objecten te vinden. (alleen General, maar dat is geen object). Wel kunnen hier algemene functionaliteiten in worden gezet, die door het hele programma kunnen worden benut. Deze modules hebben namelijk de eigenschap meteen geladen te worden bij het opstarten van het programma. Als advies bedoeld: stop alleen zaken in een module die met elkaar te maken hebben. Voeg zonodig extra modules toe.

19.1.4 Class Modules

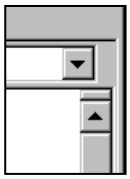
Class Modules vallen buiten het bereik van deze cursus. Kort gezegd worden deze ingezet om zelf objecten met eigen functionaliteit te maken. Zo'n class module is dan te vergelijken met bijvoorbeeld een Word sjabloon (bijvoorbeeld voor een brief), waarvan meer kopieën gemaakt kunnen worden.

19.2 De opmaak van code

Zoals gezegd, is de Code Editor van **VBA** (of van VB) eigenlijk meer een aangepaste tekstverwerker. Heel veel zaken, die we in Word gewend zijn te doen, zoals het zoeken en vervangen naar tekst, of het verplaatsen (slepen) van tekst, kunnen hier op dezelfde wijze worden uitgevoerd. Toch zijn er een aantal kleine zaken die handig in het gebruik kunnen zijn.

19.2.1 De Split Bar

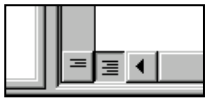
De split bar kennen we eigenlijk al van **Excel**: wanneer we het **Excel** venster willen opsplitsen in meerdere vensters, kunnen we dit verdelen (**Windows, Split**). In de **VBA** Editor zit een zelfde split bar, net boven de meest rechter schuifbalk. Door deze bar te verschuiven kunnen we het scherm in tweeën delen.



19.2.2 Views

In de **VBA** Editor kan men kiezen voor twee views:

1. Procedure View
2. Full Module View



Deze knop zit aan de linker onderkant van de Code Editor.

19.2.3 Inspringen

Het is een goede gewoonte om code te voorzien van witregels (enters) en inspringingen. Hiermee wordt de code leesbaarder en dat verkleint de kans op fouten.

19.2.4 Commentaar

Om later ook nog te achterhalen, wat nu precies het doel/werking is van een functie of procedure, dienen we onze code rijkelijk te voorzien van commentaar. Een goede gewoonte is om boven elk procedureblok te vermelden wat de naam is, eventuele argumenten, doel, door wie en wanneer het gecodeerd is, bijzonderheden, aanpassingen en dergelijke. Commentaar in Visual Basic (for Applications) wordt voorafgegaan door een enkelvoudig aanhalingsteken ' (single quote). Deze regel zal in het groen worden weergegeven. **REM** of **REMARK** heeft hetzelfde effect (maar is meer typewerk).

19.2.5 Het Line-Continuation teken

Sommige opdrachten (statements) in **VBA** zijn dusdanig lang dat deze als het ware het scherm uitlopen. Om nu te voorkomen dat de lezer/programmeur veel naar links en rechts moet scrollen om de code te kunnen lezen, kan gebruik gemaakt worden van het liggend streepje _ (underscore). Dit wordt ook wel het Line Continuation Character genoemd. Vergeet niet ййrst een spatie te geven, vervolgens het liggend streepje in te tikken, en daarna pas de enter om naar de volgende regel te gaan.

Voorbeeld:

```
MsgBox "Hier wordt gebruik gemaakt van de Line Continuation Character", _
       vbOKOnly + vbInformation, _
       "Afbreken"
```

19.2.6 Het Concatenatie teken

Wat nu te doen wanneer we een dusdanig lange tekst hebben (string), dat deze eigenlijk moet worden afgebroken. Wanneer we de string proberen te splitsen met behulp van de Line Continuation Character, krijgen we een syntax fout en een rode regel. Er moet dan dus nog iets plaatsvinden: het weer aan elkaar plakken van de twee afzonderlijke strings (of een string met een variabele, dat kan ook). Dit gebeurt met het ampersand teken **&** (concatenatie).

Voorbeeld*:*

```
MsgBox  "Hier wordt gebruik gemaakt van de LineContinuationCaracter" _
       & " en het Concatenate teken om drie strings en een variabele" _
       & " aan elkaar te plakken: " & variabele, _
       vbOKOnly + vbInformation, _
       "Afbreken"
```

Deze tekst zal echter achter elkaar in het berichtenvenster komen. Om dit te voorkomen kunnen we gebruik maken van de ingebouwde Visual Basic constante **vbCrLF** (CarriageReturn – LineFeed) zoals in onderstaand voorbeeld.

Voorbeeld*:*

```
MsgBox  "Hier wordt gebruik gemaakt van de LineContinuationCaracter" _
       & vbCrLf _
       & " en het Concatenate teken om drie strings en een variabele" _
       & vbCrLf _
       & " aan elkaar te plakken: " & variabele, _
       vbOKOnly + vbInformation, _
       "Afbreken"
```

19.2.7 Opmaak instellingen

Via **Tools, Options** kunnen in het tabblad **Editor Format** allerlei kleurinstellingen ten behoeve van de code worden gemaakt. Tevens kunnen we lettertypen en lettergrootte wijzigen.

19.2.8 Hulp bij het schrijven van code

Via **Tools, Options** kunnen we in het tabblad **Editor** allerlei zaken instellen die het schrijven van foutloze code dichterbij brengt.

- *Auto Syntax Check*: bepaalt of **VBA** automatisch bij het verlaten van de lijn op een juiste syntax moet checken.
- *Require Variable Declaration*: hiermee dwingen we onszelf om altijd eerst een variabele expliciet te declareren, voordat we deze in de code kunt gebruiken. Wanneer deze optie aan staat zullen bij alle nieuwe modules de woorden **Option Explicit** worden opgenomen bovenaan in elke module (niet de al bestaande, daar eventueel zelf intikken). Met deze optie aan is het niet mogelijk door een tikfout een nieuwe (lege) variabele te introduceren. Dit is dus veiliger. Hierover later meer.
- *Auto List Members*: de lijst die verschijnt wanneer we na een bestaand object een punt in tikken. Of bij het declareren van variabelen (As keyword). Helpt ons dus om een geldig onderdeel uit de lijst te selecteren.
- *Auto Quick Info*: geeft uitleg over de gebruikte functie, laat ons zien met welk argument we bezig zijn (bijvoorbeeld de msgbox functie). Zichtbaar in een gele box onder de functie, tijdens intypen.
- *Auto Data Tips: laat in debug/break mode de inhoud van een variabele zien.*

Het uitklaplijstje is niet altijd beschikbaar. We kunnen ook proberen het lijstje op te roepen (na de punt) met **[CTRL] – [J]**. Voor het tonen voor een uitklaplijst van de objecten drukken we op **[CTRL] – [SPATIE]**.

Het item uit de lijst kan worden afgemaakt met **[TAB]**: alleen het woord is neergezet, met **[SPATIE]**: woord plus spatie, of met **[ENTER]**: woord en direct naar de volgende regel.

20 Variabelen

20.1 Wat zijn variabelen

Variabelen worden gebruikt om tijdelijk bepaalde informatie op te slaan die voor de uitvoering van het programma belangrijk is. Bijvoorbeeld om de gebruikersnaam, die de gebruiker van de applicatie al aan het begin gegeven had, in op te slaan. Doel van zo'n variabele is dan, om deze inhoud (de gebruikersnaam dus), later weer op te halen uit het geheugen. Door gebruik te maken van variabelen wordt de code sneller: nu wordt de waarde ergens op een vaste plek in het geheugen weggezet, en kan snel opnieuw weer worden opgehaald. Dit is sneller dan elke keer de waarde opvragen (uit bijvoorbeeld een **Excel** cel, of textbox).

Hoewel het in VB(A) niet persй noodzakelijk is, om de variabelen als het ware eerst bekend te maken aan VB(A) – we noemen dit **declareren** – is dit toch een goede gewoonte. We kunnen dit afdwingen door boven aan in elke module de woorden **Option Explicit** te typen. Aangezien dit lastig is, kan dat ook door de VB(A) omgeving zelf worden geregeld: **Tools, Options,** tabblad **Editor,** optie **Require Variable Declaration**. Hiemee wordt voorkomen dat we per ongeluk een variabele introduceren die niet bekend is (en dus leeg is!).

Voorbeeld: (je kunt onderstaande code overnemen in een module)

```
  Sub Salaris()
    Dim curSalary As Currency
    curSalary = 1000
    curSalary = curSalry * 1.1
    MsgBox curSalary
End Sub
```

Wat zal de **msgbox** tonen? En wat gebeurt er als Option Explicit aanstaat?

20.2 Variabelen gebruiken

20.2.1 Variabelen declareren

Variabelen waarin data wordt opgeslagen worden aangemaakt met het woordje **DIM**. (afkomstig van Dimension). **PRIVATE** kan ook, is gelijkwaardig. Het declaratie gedeelte houden we altijd bij elkaar (overzichtelijk). Aangezien het de bedoeling is dat we de (inhoud van) de variabelen vaker gaan gebruiken, moet de gekozen naam duidelijk zijn, maar niet te lang (typewerk). Ook hier gelden conventies voor, waarover zo meteen meer. Bij het definiлren van de variabelen (en dat gebeurt dus in het declaratie statement) is het ook van belang alvast mee te geven wat voor **datatype** er in komt. Deze keuze bepaalt immers de gebruikte hoeveelheid geheugen en daarmee dus de snelheid van het programma. Wanneer we het datatype weglaten, kan een gegeven van elk datatype worden opgeslagen in die variabele, de variabele wordt dan van het type *variant* gemaakt. Dit kost veel extra geheugen!

Hieronder volgt een overzicht:

Data type	Geheugen ruimte	Bereik waarden
Byte	1 byte	0 van 255
Boolean	2 bytes	**True** of **False**
Integer	2 bytes	-32,768 van 32,767
Long (long integer)	4 bytes	-2,147,483,648 van 2,147,483,647
Single (single-precision floating-point)	4 bytes	-3.402823E38 van -1.401298E-45 voor negatieve waarden; 1.401298E-45 van 3.402823E38 voor positieve waarden
Double (double-precision floating-point)	8 bytes	-1.79769313486231E308 van -4.94065645841247E-324 voor negatieve waarden; 4.94065645841247E-324 van 1.79769313486232E308 voor positieve waarden
Currency (scaled integer)	8 bytes	-922,337,203,685,477.5808 van 922,337,203,685,477.5807
Decimal	14 bytes	+/-79,228,162,514,264,337,593,543,950,335 geen decimal teken; +/-7.9228162514264337593543950335 met 28 plaatsen rechts t.o.v. het decimal teken; kleinste getal, niet nul is: +/-0.0000000000000000000000000001
Date	8 bytes	Januari 1, 100 van December 31, 9999
Object	4 bytes	Elke verwijzing naar een **Object**
String (variabele-lengte)	10 bytes + string length	0 tot ongeveer 2 miljard
String (vaste-lengte)	Lengte van de string	1 tot ongeveer 65,400
Variant (met nummers)	16 bytes	Elke numerieke waarde in het bereik van een **Double**
Variant (met tekst characters)	22 bytes + string lengte	Zelfde bereik als een **String** van variabele lengte
User-defined (jesing **Type**)	Hoeveelheid die nodig is voor de elementen	Het bereik van elk element is hetzelfde als het bereik van zijn datetype

De syntax voor declaraties is als volgt:

```
DIM variabelenaam As datatype
```

Voorbeeld*:*

```
Dim strUserName as String
Dim rngSelectie As Range  'Dit is een object datatype!
```

Ook hier gelden weer de regels ten aanzien van naamgeving:

DataType	Voorzetsel	DataType	Voorzetsel
Boolean	Bln	Long	lng
Byte	Byt	Object	obj of voorzetsel van object zelf
Currency	Cur		
Date	Dt	Single	sng
Double	Dbl	String	str
Integer	Int	Variant	Var

Meerdere variabelen kunnen we als volgt declareren:

```
Dim a As Integer, b As Integer, c As Integer
```

In dit geval zijn a, b en c allemaal van het type Integer.

Pas op voor de valkuil:

```
Dim a, b, c As Integer
```

In dat geval wordt alleen de laatste variabele als Integer gedeclareerd. De andere twee zijn van het type Variant. We kunnen dat testen met de constructie:

```
If VarType(a) = vbInteger Then
  MsgBox "Integer "
End if
```

20.2.2 Toekennen van variabelen

Voor het toekennen van **datatype-variabelen** (alles wat in bovenstaande tabel staat, behalve het *object* type) is een eenvoudige toewijzing voldoende.

Voorbeeld: (je kunt onderstaande code overnemen in een module)

```
Sub Jarig()
 Dim strUserName As String
 Dim dtDateOfBirth As Date

 strUserName = "Patrick"
 dtDateOfBirth = #12/7/1970#
 MsgBox strUserName & " is jarig op: " & dtDateOfBirth
End Sub
```

Bij het toewijzen van zogenaamde **object-variabelen** is er ййn ding waar we op moeten letten:

het woordje **SET** gebruiken vyyr de toewijzing.

Voorbeeld: (je kunt onderstaande code overnemen in een module)

```
Sub Objecten()
  Dim rngSelectie As Range
  Set rngSelectie = ActiveSheet.Range("A1:C5")
  rngSelectie.Select
End Sub
```

20.2.3 Het declareren en toewijzen van constanten

Bij constanten is het declaratie statement en het toewijzingsstatement samengesmolten tot ййn statement. Aangezien constanten gedurende de uitvoering van het programma niet kьnnen veranderen, moet direct de toewijzing plaatsvinden. De ideale plaats om constanten te definiлren is bovenaan in een module, in de General Declarations sectie.

Voorbeeld*:*

```
Const PI As Long = 22 / 7
```

Constanten worden per afspraak in hoofdletters geschreven. De combinatie DIM werkt niet met **CONST** (van constante). Wel kan hiervoor bijvoorbeeld **Private** of **Public** worden gebruikt.

20.3 Het bereik van variabelen

De plaats waar en de wijze waarop een variabele wordt gedeclareerd, bepaalt het bereik (zichtsveld) en de levensduur van deze variabele. Overigens geldt dit niet alleen voor variabelen, maar ook voor procedures en functies. Er zijn drie soorten bereiken (**scope**):

- Local
- Module / UserForm
- Public

20.3.1 Local scope

Dit gebeurt wanneer een variabele binnen een procedure of functie met het woordje **DIM** wordt gedeclareerd. Gevolgen zijn:

- De variabele is alleen maar zichtbaar voor alles binnen die procedure of functie waarin hij is gedeclareerd. Kan dus niet van buitenaf worden opgevraagd of worden gewijzigd.
- De levensduur van de variabele is beperkt tot de tijd waarin de procedure of functie wordt uitgevoerd. Wanneer de procedure of functie is afgelopen, wordt het geheugenstukje, waarin de variabele was opgeslagen, weer vrijgegeven.

20.3.2 Module / UserForm scope

Dit gebeurt wanneer een variabele in het **General Declarations** gedeelte (dit is helemaal bovenaan in de code editor, onder **Option** Explicit) met het woordje **DIM** of **PRIVATE** wordt gedeclareerd. Gevolgen zijn:

- De variabele is nu zichtbaar voor alles binnen de module of form waarin deze is gedeclareerd. Kan dus niet van buitenaf, vanuit een andere module, worden opgevraagd of worden gewijzigd. Wel kunnen alle procedures en functies bij de inhoud van de varaibele!
- De levensduur van de variabele is nu langer: net zo lang als de module (of formmodule) in de lucht is. Aangezien de general modules in **Excel** altijd in de lucht zijn, is hun levensduur lang. Tenzij de module expliciet wordt gesloten.

Dit geldt ook voor procedures en functies, echter, zij kunnen niet met DIM bekend worden gemaakt. Dit gebeurt dan met **PRIVATE**.

20.3.3 Public scope

Dit gebeurt wanneer een variabele, procedure of functie wordt voorafgegaan door het woordje **PUBLIC**. De variabele moet, net als bij module scope, in het **General Declarations** gedeelte zijn gedeclareerd. Maar nu als public.

Gevolgen zijn:

- De variabele, procedure of functie is nu voor alle overige programma onderdelen bekend. Dat houdt in dat een variabele vanuit het hele programma is op te vragen of is te wijzigen, terwijl een procedure of functie overal vandaan is aan te roepen.
- De levensduur is nu net zo lang als de applicatie leeft.

Wanneer een procedure of functie niet expliciet met scope wordt aangegeven is zijn scope altijd public.

20.3.4 Static variabelen

Wanneer het nu toch wenselijk is om de zichtbaarheid van een variabele te beperken tot local scope (en dus is de variabele binnen de procedure aangemaakt), maar we de inhoud van die variabele toch langer beschikbaar willen houden (zoals bijvoorbeeld het geval is bij de public scope), dan is er een uitdaging. Met bovenstaande is dat niet te realiseren. Kan er dus bijvoorbeeld niet worden bijgehouden hoe vaak er op een knop is gedrukt. Of toch wel? Met het woordje **STATIC** kan dit wel. De variabele moet binnen een procedure of functie worden gedeclareerd, net als bij local scope, maar in plaats van **DIM** wordt nu **STATIC** gebruikt.

Voorbeeld: (je kunt onderstaande code overnemen in een UserForm)

```
Private Sub cmdTellertje_Click()
  Static intTeller As Integer

  intTeller = intTeller + 1
  lblTeller.Caption = intTeller
End Sub
```

20.3.5 Arrays in VBA

VBA kent **fixed arrays** en **dynamic arrays**. Fixed arrays hebben een vaste grootte die niet veranderd kan worden. Dynamic arrays kunnen we na hun declaratie nog vergroten of verkleinen. We spelen eerst wat met een fixed array:.

```
Function fixedArray()
   Dim rij(1 To 4) As Integer
   For i = 1 To 4 'indextelling begint bij 1
       rij(i) = i
       Debug.Print rij(i); " ";
   Next i
End Function
```

Merk als eerste op, dat we de indextelling bij 1 laten beginnen en niet bij 0, zoals in Java. Dat stellen we in met het argument 1 To 4. We hadden ook 0 To 3 kunnen geven. De puntkomma's in het Debug.Print-statement geven aan dat we op dezelfde regel blijven. Het meegeven van het argument 1 To 4 in de declaratie van rij maakt het array fixed.

Als we geen argument meegeven, declareren we een dynamic array, zoals in de volgende functie:

```
Function dynamicArray()
   Dim rij() As Integer
   ReDim rij(1 To 4)
   For i = 1 To 4
       rij(i) = i
       Debug.Print rij(i); " ";
   Next i
   Debug.Print
   ReDim Preserve rij(1 To 5)
   For i = 1 To 5
       Debug.Print rij(i); " ";
   Next i
   Debug.Print
   ReDim Preserve rij(1 To 3)
   For i = 1 To 3
       Debug.Print rij(i); " ";
   Next i
   Debug.Print
   ReDim rij(1 To 3)
   For i = 1 To 3
       Debug.Print rij(i); " ";
   Next i
End Function
```

Met *ReDim* geven we de array een grootte.

Om naar de volgende regel te gaan met printen geven we een *Debug.Print*-statement zonder ; . Vervolgens maken we het array 5 elementen groot. *Preserve* geeft aan dat de huidige waarden bewaard moeten blijven. Het vijfde element heeft geen waarde en krijgt standaard de waarde 0. Dan maken we het array 3 elementen groot, met behoud van de waarden en vervolgens zonder behoud van waarden.

De output van de functie *fixedArray*() (als we die aanroepen in het Immediate Window) is de volgende:

```
1   2   3   4
```

En van de functie dynamicArray():

```
1    2    3    4
1    2    3    4    0
1    2    3
0    0    0
```

21 Procedures en functies

21.1 Inleiding

In Visual Basic For Applications zijn er een drietal soorten procedures:

- Event Procedures
- General Procedures
- Property Procedures

In de volgende paragrafen worden deze toegelicht.

21.2 Event Procedures

Deze zijn al ter sprake gekomen. Event Procedures worden opgeroepen op het moment dat het object op een bepaalde gebeurtenis reageert. Zo'n procedure wordt, hoe dan ook, altijd uitgevoerd. Dat niet alle events tot zichtbare resultaten leiden, komt omdat de programmeur zйlf nog code voor dat event moet schrijven. Bijvoorbeeld de feitelijke afhandeling van het opslaan van een bestand, wanneer de gebruiker op de ***opslaan*** knop drukt (en daarmee dus de *click* event van een *commandbutton* oproept.). Om event procedures te gebruiken en daarvoor code te schrijven, moet eerst in de **ObjectListBox** het gewenste object worden gekozen (waarop moet de gebeurtenis plaatsvinden, of wat dient waarop te reageren), om vervolgens in de **ProcedureListBox** de juiste event te kiezen.

De syntax is als volgt:

```
Private Sub Object_Event()
  'code statements komen hier.
End Sub
```

Voorbeeld: (we kunnen onderstaande code overnemen in een **UserForm**)

```
Private Sub cmdOpslaan_Click()
  Dim strFileName As Variant

  strFileName = Application.GetSaveAsFilename( _
    fileFilter:="Microsoft Excel Workbook (*.xls), *.xls")
    If strFileName <> False Then
        ActiveWorkbook.SaveAs strFileName
    End If

End Sub
```

Zoals we zien heeft bovenstaand voorbeeld geen argumenten: tussen de haken, direct nб de event naam, staat niets ingevuld. Het is echter ook mogelijk, dat er tussen de haken bepaalde argumenten worden meegegeven. Deze kunnen dan in de procedure zelf worden gebruikt. Niet altijd hoeven de argumenten te worden ingevuld of te worden meegegeven: optionele argumenten.

De syntax is als volgt:

```
Private Sub Object_Event(argement, [optioneel argument])
  'code statements komen hier.
End Sub
```

Voorbeeld: (we kunnen onderstaande code overnemen in een **UserForm**)

```
Private Sub cmdOpslaan_MouseDown(ByVal Button As Integer, ByVal Shift As
Integer, ByVal X As Single, ByVal Y As Single)
  MsgBox "Button = " & Button & vbCrLf & _
  "Shift = " & Shift & vbCrLf & _
  "X = " & X & " Y = " & Y
End Sub
```

Waarom zijn X en Y (de courdinaten van de muisaanwijzer) niet eerst met DIM gedeclareerd?
Waarom is de scope van event procedures PRIVATE?

21.3 General Procedures

21.3.1 Inleiding

General procedures zijn functionaliteiten die aan het programma worden toegevoegd. Deze zaken zitten dus niet meegebakken in het object zelf en dienen door de programmeur zelf te worden gemaakt. Het doel van een general procedure kan tweeledig zijn:

- Het maken van een procedure die een aantal handelingen (en dat kunnen dus ook wijzigingen van properties zijn) bij elkaar groepeert: SUB PROCEDURES.
- Het maken van een procedure waaruit een resultaat of waarde komt: FUNCTION PROCEDURES

21.3.2 Sub Procedures

Kenmerk van Sub procedures is dat deze handelingen uitvoeren of kenmerken van objecten wijzigen. De procedure zelf kan nooit een resultaat/waarde teruggeven.

De Syntax is als volgt:

```
Sub ProcedureNaam()
  'code komt hier.
End Sub
```

Voorbeeld: (we kunnen onderstaande code overnemen in een Module)

```
Sub BerekenOmtrek()
  Dim intStraal As Integer
  Dim dblOmtrek As Double

  intStraal = InputBox("Geef de straal van de circel:", "Straal", 10)
  dblOmtrek = 2 * 22 / 7 * intStraal
  ActiveCell.Value = dblOmtrek
  ActiveCell.NumberFormat = "[blue]#,##0.00_ ;[Red]-#,##0.00 "
End Sub
```

Wat is de scope van deze sub procedure?

Hoewel het lijkt alsof deze procedure iets berekent en dus een waarde teruggeeft aan de functie, is dat niet zo. Er wordt inderdaad iets berekend (namelijk de omtrek), maar de uitkomst wordt in de procedure zelf naar de actieve cel gestuurd: er worden alleen eigenschappen gewijzigd.

21.3.3 Function Procedures

Kenmerk van Function procedures is dat deze gййn handelingen uitvoeren of kenmerken van objecten wijzigen. De procedure zelf kan juist wel een resultaat/waarde teruggeven. Over het algemeen zullen functies ййn of meerdere argumenten bevatten. Zelfs optionele argumenten (niet verplicht om in te vullen) zijn mogelijk. Wel moeten deze als laatste йn van het type *Variant* zijn gedefinieerd (in verband met de *IsMissing* functie). We moeten er op letten dat we ook de juiste datatypen specificeren: een functie geeft immers een waarde terug!

Syntax is als volgt:

```
Function FunctieNaam(Argument As Datatype) As Datatype
  'code komt hier.
End Function
```

Voorbeeld: (we kunnen onderstaande code overnemen in een Module)

```
Function Omtrek(Straal As Integer) As Double
  Omtrek = 2 * 22 / 7 * Straal
End Function
```

Wat is de scope van deze function procedure?

```
Function FunctieNaam(Argument As Datatype, Optional Argument As DataType)  _
As Datatype,)
  'code komt hier.
End Function
```

Voorbeeld: (we kunnen onderstaande code overnemen in een Module)

```
Function Optellen(Waarde1 As Variant, Waarde2 As Variant, _
                  Optional Waarde3 As Variant) As Variant
  If IsMissing(Waarde3) Then
      Optellen = Waarde1 + Waarde2
  Else
      Optellen = Waarde1 + Waarde2 + Waarde3
  End If
End Function
```

21.3.4 Procedures en Functies aanroepen

Procedures kunnen op dezelfde manier getest of gebruikt worden als de normale *macro's*, in de **Excel** en Word omgeving: via **Tools, Macro, Macro's...** . Selecteer vervolgens de gewenste macro. Tevens kunnen deze macro's ook worden hergebruikt in andere macro's door de naam van de macro op te nemen op de plaats waar deze moet worden uitgevoerd in de andere macro.

Functies daarentegen kunnen worden uitgevoerd in de **Excel** omgeving via **Insert, Function...**. Selecteer vervolgens de groep/categorie **User Defined.** De functie staat nu aan de rechterkant in de lijst. De schermpjes krijgen we van **Excel** cadeau. Functies kunnen ook in code worden aangeroepen. Ook hier geldt dat de naam van de functie dan moet worden opgenomen, maar nu ook met de argumenten.

```
Sub Aanroep()
  MsgBox Optellen(10, 20)
  MsgBox Optellen(10, 20, 30)
End Sub
```

21.4 Property Procedures

Property procedures worden gebruikt om eigenschappen te maken in objecten. Hier spelen classes een belangrijke rol. Aangezien dit buiten deze cursus valt, gaan we hier niet verder op in.

22 Communiceren met de gebruiker

22.1 Berichten weergeven

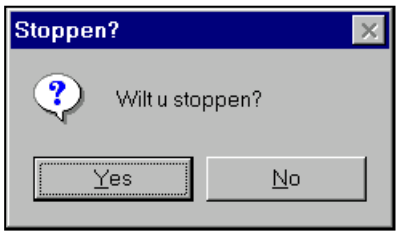
Berichten weergeven aan de gebruiker gebeurt door een **msgbox** functie. Het kan zijn dat er puur een melding op het scherm moet verschijnen (1) of dat er een melding moet komen, waarop de gebruiker verschillend op kan reageren(2). De syntax lijkt op elkaar, maar vorm (2) heeft de haakjes nodig:

- MsgBox prompt, buttons, title, helpfile, context
- variabele = MsgBox(prompt, buttons, title, helpfile, context)

Alleen het ***prompt*** argument is verplicht. Hiermee tonen we de melding aan de gebruiker. Met het ***buttons*** argument geven we op welke knoppen (en eventuele icons we op het dialoogvenster willen hebben). Het ***title*** argument is de string die in de blauwe titelblak van het dialoogvenster verschijnt. Tevens kunnen we via de argumenten ***helpfile*** en ***context*** aangeven welk helpfile en item aan deze melding gekoppeld zijn.

Voorbeeld:

```
Sub Berichten()
  Dim intAntwoord As Integer

  MsgBox "Het is mooi weer vandaag", vbOKOnly + vbInformation, "Melding"
  intAntwoord = MsgBox("Wilt U stoppen?", vbYesNo + vbQuestion, "Stoppen?")
      If intAntwoord = vbYes Then
        MsgBox "U stopt!", vbCritical
      Else
        MsgBox "U stopt niet!", vbExclamation
      End If
End Sub
```



Ook hebben we de mogelijkheid gebruik te maken van de **named arguments**: in plaats van het argument in te vullen op de juiste positie in de functie, kunnen we flexibeler werken met het eerst benoemen van het argument. De locatie maakt dan niet meer uit, en de code wordt, zeker bij ingewikkelde – niet vaak gebruikte functies – veel leesbaarder.

Voorbeeld*:*

```
Sub Aflossing()
  Dim curAflossing As Currency
  curAflossing = Pmt(Rate:=0.06 / 12, NPer:=30 * 12, PV:=-100000)

  MsgBox prompt:="Uw aflossing per maand bedraagd: " & curAflossing, _
      Title:="Aflossing", _
      Buttons:=vbOKOnly + vbInformation
End Sub
```

Let op de := bij de argumenten!

22.2 Vragen stellen

We kunnen ook invoer van de gebruiker nodig hebben. Bijvoorbeeld de naam, leeftijd en noem maar op. Ook dan kunnen we gebruik maken van voor gedefinieerde dialoogvensters. In dit geval de **inputbox** functie. Ook hier zijn er twee vormen: de standaard inputbox functie van **VBA** (1) en de inputbox functie van de toepassing **Excel** (2). De laatste is veel beter toegespitst op de mogelijkheden van **Excel** (extra argument). Ook hier is het de bedoeling dat er iets gedaan wordt met de teruggegeven waarde (van de gebruiker), dus vergeet de haakjes niet na de functie!

(1) **variabele = InputBox _**
prompt, title, default, xpos, ypos, helpfile, context

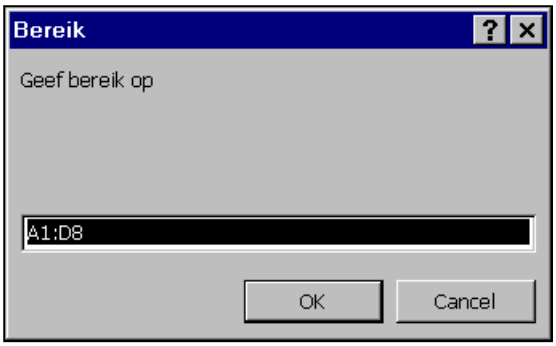
Alleen het ***prompt*** argument is verplicht. Hiermee tonen we de melding aan de gebruiker. Het ***title*** argument is de string die in de blauwe titelblak van het dialoogvenster verschijnt. Met het ***default*** argument geven we een standaard waarde mee. Ook de plaatsing van het dialoogvenster kan worden geregeld, via ***xpos*** en ***ypos***. Tevens kunnen we via de argumenten ***helpfile*** en *context* aangeven welk helpfile en item aan deze melding gekoppeld zijn.

(2) **variabele = Application.InputBox _**
(prompt, title, default, left, top, helpfile, helpcontextid, type)

Application staat hier voor de toepassing **Excel**: de functie is alleen te gebruiken in **Excel**. Alleen het ***prompt*** argument is verplicht. Hiermee tonen we de melding aan de gebruiker. Het ***title*** argument is de string die in de blauwe titelblak van het dialoogvenster verschijnt. Met het ***default*** argument geven we een standaard waarde mee. Ook de plaatsing van het dialoogvenster kan worden geregeld, via ***left*** en ***top***. Tevens kunnen we via de argumenten ***helpfile*** en ***helpcontextid*** aangeven welk helpfile en item aan deze melding gekoppeld zijn. Als extra argument kent deze inputbox ook het ***type*** argument: hiermee kunnen we bepalen wat voor type invoer gewenst is (bijvoorbeeld een bereik, type 8).

Voorbeeld*:*

```
Sub Invoer()
  Dim strName As String
  Dim rngBereik As Range
  Dim intAntwoord As Integer

  strName = InputBox("Geef je naam", "Naam", "Patrick")
  Set rngBereik = Application.InputBox("Geef bereik op", "Bereik", "A1:D8",
, , , , 8)

End Sub
```



Wanneer we gebruik maken van de inputbox functie (beide versies), kunnen we geen knoppen specificeren. Dit zijn altijd de **OK** en **Cancel/Annuleren** knop. Wanneer we dit anders willen, zullen we onze eigen form moeten maken. Wanneer we op **OK** klikken, zal in de variabele de waarde zitten die door de gebruiker in de textbox is ingevoerd. Drukt de gebruiker op Cancel/Annuleren, dan zal er een lege string ('') worden teruggegeven bij vorm (1), bij vorm (2) wordt **false** teruggegeven.

23 Besturingsstructuren

23.1 Vergelijkings- en logische operatoren

23.1.1 Vergelijkingsoperatoren

<	Kleiner dan
>	Groter dan
<=	Kleiner dan of gelijk aan
>=	Groter dan of gelijk aan
=	Gelijk aan
<>	Niet gelijk aan

23.1.2 Logische operatoren

Voor het combineren van meerdere expressies (condities) zijn logische operatoren vereist om deze afzonderlijke condities tot ййn expressie samen te voegen. **VBA** gebruikt zogenaamde booleaanse logica (werken met 0 – **False** en 1 – **True**) om tot een eindresultaat te komen.

AND	Combineert twee expressies. Beide afzonderlijke expressies moeten waar zijn, om de hele expressie waar te laten zijn.
OR	Combineert twee expressies. Ййn van beide afzonderlijke expressies moet waar zijn, om de hele expressie waar te laten zijn.
NOT	De ontkenning van een enkele expressie.

Er zijn nog meer logische operatoren, maar deze worden zelden gebruikt.

23.2 Conditionele structuren

Met behulp van conditionele structuren is het mogelijk het programma te laten reageren, afhankelijk van de uitkomst van een bepaalde conditie. Een statement (of meerdere statements) kunnen nu, afhankelijk van de uitkomst van deze conditie, al dan niet worden uitgevoerd.

23.2.1 If…Then

In het geval zowel de conditie als het statement, dat moet worden uitgevoerd wanneer de conditie waar/true is, (de zogenaamde *default*, dit hoeven we dus niet expliciet te noemen) op een regel past, is de syntax als volgt:

```
IF conditie [= TRUE] THEN single statement
```

Voorbeeld:

```
If intCPUTemperature >= 100 Then msgbox "CPU Temerature is too high!"
```

Voor het geval het vervolg uit meerdere acties moet bestaan (en dus meerdere statements bevat), is de syntax als volgt (let op de afsluiting!):

```
IF conditie THEN
  één of meerdere statements
END IF
```

Deze structuur kan altijd, terwijl de eerste (zonder de END IF) alleen is toegestaan bij ййn statement.

Voorbeeld:

```
If intCPUTemperature >= 100 Then
  msgbox "CPU Temerature is too high!"
  intCPUTemperature = intCPUTemperature - 50
End If
```

23.2.2 If…Then…Else

Bij de IF…THEN structuur wordt alleen iets uitgevoerd wanneer de conditie waar is. Als er ook iets uitgevoerd moet worden indien de conditie niet waar is, kan dat niet met de eerste structuur maar is de IF…THEN…ELSE structuur nodig. Hiermee worden ook alle andere situaties (ELSE) afgevangen. Controle of het afvragen vindt plaats op maar ййn conditie: er kan dus maar uit ййn alternatief worden gekozen.

Syntax:

```
IF conditie THEN
  één of meerdere statements
ELSE
  één of meerdere statements
END IF
```

Voorbeeld*:*

```
If strPassword = "cursus" Then
  msgbox "Access granted!"
  '[code voor toegang tot systeem]
Else
  msgbox "Access denied!"
End If
```

23.2.3 If…Then…ElseIf

Bij deze structuur kan vanwege de ELSEIF gekozen worden uit meer dan ййn conditie of alternatief. Deze condities kunnen verschillend zijn ten opzichte van elkaar.

Syntax:

```
IF conditie ①  THEN
  één of meerdere statements
ELSEIF conditie ②  THEN
  één of meerdere statements
ELSE
  één of meerdere statements
END IF
```

Voorbeeld*:*

```
If strPassword = "cursist" Then
  msgbox "Access granted to cursist!"
  '[code voor toegang tot systeem voor cursist]
ElseIf strPassword = "docent" Then
  msgbox "Access granted to docent!"
  '[code voor toegang tot systeem voor docent]
Else
  msgbox "Access denied!"
End If
```

Het is mogelijk om, indien er op meerdere condities gecontroleerd moet worden (dus niet op twee, zoals in het voorbeeld), het ELSEIF … THEN statement te herhalen. Dit komt echter de leesbaarheid en snelheid van uitvoering niet ten goede. Helemaal niet wanneer er steeds op dezelfde variabele (in voorbeeld strPassword) moet worden getest: de waarde steeds worden opgehaald, waardoor de code trager wordt.

23.2.4 Select Case

Wanneer er gekozen moet worden uit meerdere alternatieven, maar er in feite sprake is van ййn testexpressie (bijvoorbeeld de waarde van een variabele) is het efficiʌnter gebruik te maken van een SELECT CASE statement. De testexpressie wordt nu maar ййn keer doorlopen. Bovendien is het nu ook mogelijk om te testen op bereiken (vooral handig bij waarden).

Syntax:

```
SELECT CASE testexpressie
  CASE expressie ①
    statement(s)
  CASE expressie ②
    statement(s)
  CASE ELSE
    statement(s)
END SELECT
```

Zo kan bovenstaand voorbeeld (IF…THEN…ELSEIF) worden herschreven in een SELECT CASE structuur:

Voorbeeld*:*

```
Select Case strPassword
   Case "cursist"
       msgbox "Access granted to cursist!"
       '[code voor toegang tot systeem voor cursist]
   Case "docent"
       msgbox "Access granted to docent!"
       '[code voor toegang tot systeem voor docent]
   Case Else
       msgbox "Access denied!"
End Select
```

Ook kan er op bereiken worden getest.

Voorbeeld*:*

```
Function Bonus(Performance As Integer, Salary As Currency) As Currency
   Select Case Performance
       Case 1
           Bonus = Salary * 0.3
       Case 2, 3
           Bonus = Salary * 0.2
       Case 4 To 6
           Bonus = Salary * 0.1
       Case Is > 8
           Bonus = 100
       Case Else
           Bonus = 0
   End Select
End Function
```

23.3 Lus structuren

Om er voor te zorgen dat een statement, of meerdere statements, herhaaldelijk kunnen worden herhaald, kunnen lusstructuren worden ingezet. Aangezien er nogal uit een aantal gekozen kan worden, is het goed te beseffen, wat nu precies nodig is. Als we goed naar onszelf luisteren, is vaak de oplossing al gevonden. In feite komt het neer op het stellen van de volgende vragen:

1) Is van tevoren al bekend hoe vaak iets moet worden doorlopen? Dit kan zijn doordat de programmeur hierover al een beslissing kan nemen, of doordat het programma zelf is staat is, dit te bepalen (bij het aflopen van een verzameling bijvoorbeeld).
2) Als dat niet het geval is, moet de code dan mimimaal ййn keer worden doorlopen, of niet. Er moet dan voorwaardelijk een lus gemaakt worden. En willen we dan dat de actie(s) worden uitgevoerd *zolang* of *totdat* aan de voorwaarden is/wordt voldaan?

23.3.1 For…Next

Deze lusstructuur wordt gebruikt als van tevoren al vaststaat hoe vaak de code in de lus moet worden uitgevoerd. Door middel van een **teller** kan worden bijgehouden, hoe vaak de lus al is uitgevoerd.

Syntax:

```
FOR teller = start TO end [stapgrootte]
  statement(s)
NEXT [teller]
```

Tussen haken staan de niet verplichte zaken zoals stapgrootte (standaard is deze 1) en het herhalen van de tellervariabele bij het **NEXT** statement. Dat laatste is aan te raden om wйl op te nemen, puur voor de leesbaarheid.

Voorbeeld*:*

```
Sub Lus()
  Dim intTeller As Integer

  For intTeller = 1 To 5
      MsgBox "De lus is nu " & intTeller & " keer doorlopen."
  Next intTeller
End Sub
```

23.3.2 For…Each

Deze lus wordt gebruikt wanneer onderdelen van een collectie (verzameling) moeten worden afgelopen. Bijvoorbeeld elk werkblad (worksheets) uit de verzameling werkbladen (worksheet**s**) moet worden voorzien van een voettekst waarin de datum en de documentnaam staan.

Syntax:

```
FOR EACH itemvariable IN itemcollection
  statement(s)
NEXT [teller]
```

Voorbeeld:

```
Sub KoppenEnVoeten()
  Dim strPath As String
  Dim strDate As String
  Dim shtSheet As Worksheet

  strPath = ActiveWorkbook.FullName
  strDate = Format(Date, "dd-mmm-yyyy")

  For Each shtSheet In ActiveWorkbook.Worksheets
      shtSheet.PageSetup.LeftFooter = strDate
      shtSheet.PageSetup.RightFooter = strPath
  Next shtSheet

End Sub
```

23.3.3 Do…Loop While

Deze structuur zorgt ervoor dat de code in de lus **minimal ййn keer** wordt doorlopen: de controle op de testexpressie vindt plaats nб uitvoering van de lus. Deze code wordt net zo vaak doorlopen **totdat** de testexpressie **false** teruggeeft.

Syntax:

```
DO
   statements
LOOP WHILE testexpressie
```

Voorbeeld*:*

```
Sub VoorwaardelijkeLus1()
  Dim strPassword As String
  Do
    strPassword = InputBox("Password:")
  Loop While strPassword <> "cursist"
End Sub
```

23.3.4 Do…Loop Until

Deze structuur zorgt ervoor dat de code in de lus **minimal ййn keer** wordt doorlopen: de controle op de testexpressie vindt plaats nб uitvoering van de lus. Deze code wordt net zo vaak doorlopen **totdat** de testexpressie **true** teruggeeft.

Syntax:

```
DO
   statements
LOOP UNTIL testexpressie
```

Voorbeeld*:*

```
Sub VoorwaardelijkeLus2()
  Dim strPassword As String
  Do
    strPassword = InputBox("Password:")
  Loop Until strPassword = "cursist"
End Sub
```

Let op! Beide lusstructuren zijn in elkaar om te zetten, maar dan moet de testexpressie worden aangepast.

23.3.5 Do While…Loop

Deze structuur zorgt ervoor dat de code in de lus **eerst wordt gecontroleerd**: de controle op de testexpressie vindt plaats vyyr uitvoering van de lus. Pas wanneer de controle gelukt is (testexpressie geeft **true** terug) wordt begonnen aan de lus. Het kan dus gebeuren dat de code nooit wordt doorlopen! Herhaling vindt alleen plaats **totdat** de testexpressie **false** teruggeeft.

Syntax:

```
DO WHILE testexpressie
   statements
LOOP
```

Voorbeeld*:*

```
Sub VoorwaardelijkeLus3()
  Dim strPassword As String
  Do While strPassword <> "cursist"
    strPassword = InputBox("Password:")
  Loop
End Sub
```

23.3.6 Do Until…Loop

Deze structuur zorgt ervoor dat de code in de lus **eerst wordt gecontroleerd**: de controle op de testexpressie vindt plaats vyyr uitvoering van de lus. De code in de lus wordt alleen uitgevoerd wanneer de testexpressie **false** teruggeeft. Het kan dus gebeuren dat de code nooit wordt doorlopen! Herhaling vindt alleen plaats **totdat** de testexpressie **true** teruggeeft.

Syntax:

```
DO UNTIL testexpressie
   statements
LOOP
```

Voorbeeld*:*

```
Sub VoorwaardelijkeLus4()
  Dim strPassword As String
  Do Until strPassword = "cursist"
    strPassword = InputBox("Password:")
  Loop
End Sub
```

23.3.7 Een lus direct verlaten

Een lus kan altijd via het **EXIT** statement worden afgebroken. Afhankelijk van het type lus moet gebruik gemaakt worden van EXIT FOR of EXIT DO. Dit is vooral handig wanneer de lus direct moet worden verlaten en het niet meer nodig is om verdere uitvoering plaats te laten vinden.

24 Foutopsporing en Foutafhandeling

24.1 Inleiding

Foutopsporing of **debugging** van een applicatie wordt gebruikt wanneer de code anders reageert dan we zouden verwachten. Ergens in de code zit een foutje of onvolkomenheid, die de juiste uitvoering van het programma in de weg staat. Het regel voor regel napluizen van code is nu, met de komst van een aantal geavanceerde debugging tools in **VBA**, niet persй meer nodig. Hoewel een goede dosis gezond verstand voor het snel oplossen van een probleem nooit weg is natuurlijk.

24.2 Foutopsporing

24.2.1 De break mode

Deze break mode (staat), ook wel debug mode genoemd, wordt gebruikt om fouten op te sporen en kan worden bereikt op een aantal manieren.

- Wanneer er in de code een onoplosbare fout (voor **VBA**) optreedt, en gekozen wordt voor debug in het dialoogvenster;
- Wanneer op [CTRL]-[BREAK] yf op de stopknop wordt gedrukt
- Wanneer **VBA** een breakpoint tegenkomt;
- Wanneer **VBA** een STOP statement in de code tegenkomt;
- Wanneer er een WATCH EXPRESSIE afgaat.

24.2.2 Breakpoints

Breakpoint zijn zogenaamde markeringspunten in de code, waarop **VBA** de uitvoering zal onderbreken. Het is dan mogelijk hierna de code op een aantal manieren te vervolgen (hierover later meer). Het instellen kan via:

- Het Debug menu, optie Toggle Breakpoint;
- De functietoets **F9**;
- De Toggle Breakpoint knop op de toolbar (handje)
- Via het contextgevoelige snelmenu en kies voor de optie Toggle Breakpoint;
- Klikken in de grijze Marge Balk.

Deze methoden hebben als voordeel dat ze de code niet aantrasten, maar het kan dus nadelig zijn indien de instellingen bewaard moeten blijven. Gebruik dan het STOP statement. Dit kan overal in de code worden geplaatst. Echter, verwijder dit als het niet meer nodig is.

24.2.3 De debug toolbar

De debug toolbar geeft ons een aantal mogelijkheden om gericht een fout op te zoeken en op te lossen. Van links naar rechts:

Design Mode	Schakelt design mode in of uit
Run Sub/UserForm of **Run Macro**	Voert de huidige procedure uit wanneer de cursur in een procedure staat, toont een userform wanneer dit actief is, of laat het macro dialoogvenster zien, wanneer de cursor niet in een procedure of userform staat.
Break	Stopt de uitvoering van het programma als dat in uitvoering is en gaat naar break mode.
Reset/Stop	Schoont de stack en alle variabelen op module niveau
Toggle Breakpoint	Stelt een breakpoint in, of haalt deze weg
Step Into	Voert de code regel voor regel uit. Aangeroepen procedures worden ook regel voor regel uitgevoerd.
Step Over	Voert de code binnen de procedure regel voor regel uit. Aangeroepen procedures worden direct (niet regel voor regel) uitgevoerd.
Step Out	Voert de rest van de procedure uit.
Locals Window	Toont het Locals Window. Hiermee is het mogelijk variabelen, maar ook eigenschappen van bijvoorbeeld grafische objecten op een formulier in break mode op te vragen йn te wijzigen.
Immediate Window	Toont het Immediate Window. Hiermee kunnen waarden van variabelen in break mode worden opgevraagd of gewijzigd. Dit kan worden gedaan met het **PRINT** statement of het vraagteken **?**: *PRINT strUserName* Tevens kunnen procedures of functies worden getest, of zelfs complete stukjes code worden uitgevoerd. Invoer is identiek als direct via code.
Watch Window	Toont het Watch Window. Hiermee kunnen waarden van variabelen in break mode worden opgevraagd of gewijzigd. Het moment wanneer naar break mode wordt overgegaan is in te stellen: ▪ *watch expression*: alleen de waarde bekijken, lijst genereren ▪ *break when value is true*: in break mode wanneer aan de expressie wordt voldaan ▪ *break when value changes*: in break mode wanneer de waarde van een expressie verandert Let wel op de ingestelde context: procedure en module.
Quick Watch	Toont het Quick Watch dialoog venster, met daarin de huidige geselecteerde variabele. Geeft de mogelijkheid snel variabelen toe te voegen aan de Watch Window.
Call Stack	Toont de Call Stack. Deze geeft een lijst met daarin alle procedures die door elkaar zijn aangeroepen, mits deze al niet zijn beλindigd.

Tevens kunnen resultaten ook direct naar het Immediate Window worden weggeschreven: **DEBUG.PRINT**. Vaak wordt voor testdoeleinden namelijk ook de MsgBox functie gebruikt. Deze moet echter wel weer worden verwijderd. Debug.print zal aan de gebruiker geen hinderlijke meldingen laten zien. Overigens is het wel veiliger om ook, na het testen, het debug.print statement te verwijderen: eventuele code wordt namelijk wel uitgevoerd.

24.3 Foutafhandeling

In VB(A) worden grofweg drie soorten fouten onderscheiden:

- Syntax fouten
- Logische fouten
- Run-time fouten

Fouten van het eerste type kunnen eenvoudig worden vermeden door de juiste instellingen bij Tools, Options te plegen. De Visual Basic Editor (en deze zit dus ook in **VBA**) geeft dan een melding wanneer de syntax niet correct is wanneer de optie **Auto Syntax Check** is ingesteld.

Fouten van het tweede type zijn wat lastiger. **VBA** is niet in staat deze te detecteren, het programma functioneert volgens de regels van de syntax gewoon goed. Maar toch krijgen we niet de juiste uitkomst. Denk bijvoorbeeld aan het optellen van twee waarden in een textbox: 12+13 geeft dan 1213 in plaats van 25. (een textbox behandelt de gegevens zonder extra aandacht van de programmeur als strings!). Gelukkig zijn er de diverse **debugging tools** om deze problemen te lijf te gaan. Testen is en blijft de enige remedie.

Fouten van het derde type vergt inlevingsvermogen van een programmeur. Immers, bij het testen kan alles wel goed gaan (omdat de programmeur immers bekent is met de werking van het programma!), maar bij een gebruiker kan plotseling van alles misgaan. Daarom is het nodig om de meest vreemde mogelijkheden te hebben afgevangen. Dit proces staat ook bekent onder **error handling**.

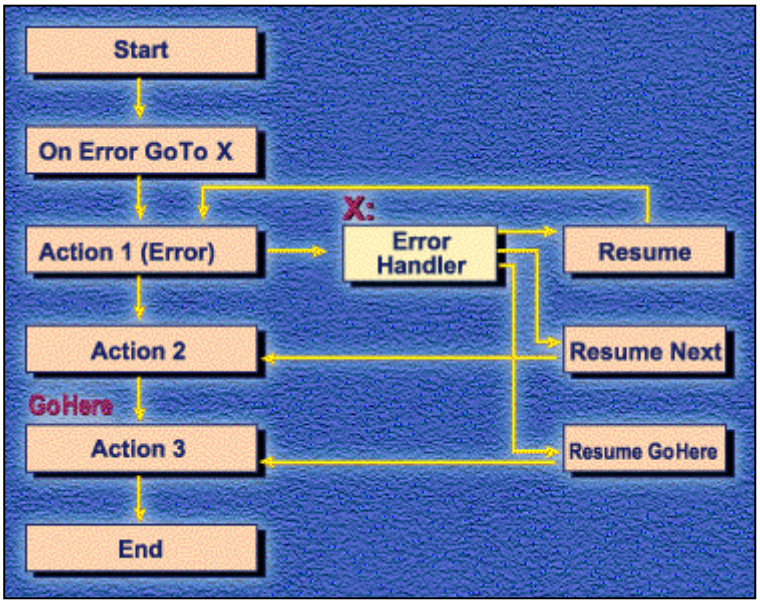
24.3.1 De stappen

Om error handling aan te zetten dienen een aantal stappen te worden doorlopen:

- Vertel Visial Basic For Applications dat de error handler aan staat: nu is bekend wat er gedaan moet worden voor 6ls het fout gaat! Dit wordt gedaan met het On Error statement. Dit statement kan alleen maar voorkomen in de body van een procedure en bijvoorbeeld niet in het General Declarations gedeelte. Ook is het niet mogelijk om hiermee gelijk naar een andere procedure te springen, of om tegelijkertijd meerdere soorten afhandelingen met On Error aan te zetten: er kan er maar ййn tegelijk actief zijn.
- Geef aan waar de fout kan worden opgelost. Dit gebeurt meestal met zogenaamde labels, die aan het einde van een procedure zijn geplaatst (nog wel in de body!). Vergeet niet om boven het label een Exit Sub of Exit Function op te nemen.
- Los de fout daadwerkelijk op en verlaat de foutafhandelingroutine met ййn van de volgende statements:

- Resume — ga verder met dezelfde regel als waar de fout optrad,
- Resume Next — ga verder met de regel na die regel waar de fout optrad,
- Resume label — ga verder met label,
- Exit Sub — verlaat de sub procedure onmiddellijk,
- Exit Function — verlaat de function procedure onmiddellijk,
- End — sluit de applicatie af.

Schematisch:



Voorbeeldcode:

```
Sub Foutje()
  On Error GoTo Err_Foutje

  Dim i As Integer

  For i = 1 To 10
    MsgBox 10 / (5 - i)
  Next i

  Exit Sub

  Err_Foutje:
   Select Case Err.Number
       Case 11
           MsgBox "Er heeft zich een fout voorgedaan. " _
                           & vbCrLf & "Er is door nul gedeeld."
           i = i + 1
           Resume
       Case Else
           MsgBox Err.Number & " " & Err.Description
   End Select

End Sub
```

Opmerkingen:

- Het On Error statement kent wat aanvullingen:
- On Error Goto 0 de error handling routine weer uitzetten, (0=nul);
- On Error Resume Next ongeacht de fout, gewoon doorgaan. Wees daarmee voorzichtig, fouten kunnen zich nu makkelijk opstapelen!
- Het On Error statement moet in de body van een procedure zijn opgenomen. Dit geldt ook voor de verschillende Resume statements.
- Pas op met labels: ze maken de code al gauw ondoorzichtig, waardoor de kans op fouten toeneemt.
- Gebruik Exit Sub/Exit Function om de procedure te verlaten wanneer alles goed gegaan is: laten we dit weg dan zal ook de foutafhandeling worden uitgevoerd!
- Let op de dubbele punt achter het label!

24.3.2 Het Err object

Om te achterhalen welke fout nu is opgetreden, moeten we wat extra informatie aan de fout meegeven. Dit gebeurt door middel van het zogenaamde **Err** object. Dit object kan gebruikt worden om extra informatie over de opgetreden fout op te vragen.

Property: number	Het nummer van de fout
Property: description	De omschrijving van de fout
Property: source	De bron/locatie van de fout
Method: raise	Het opwekken van zelfgedefinieerde fouten
Method: clear	Het weghalen van de fout uit het geheugen.

24.3.3 Een centrale Error Handler

De taal Visual Basic schrijft voor dat we een **On Error** statement alleen maar mogen gebruiken in de body van een procedure. Dit is lastig, want hierdoor zullen we heel veel code moeten herhalen (vooral als in een aantal procedures dezelfde fout kan optreden, die op dezelfde manier moet worden opgevangen).

In VB(A) zijn hiervoor een aantal oplossingen bedacht. Wanneer in een aangeroepen procedure (een procedure die vanuit een andere procedure is aangeroepen) een fout optreedt, zal **VBA** eerst in de eigen (de aangeroepen) procedure kijken, of daar een error handling routine in staat. Zo ja, dan wordt gekeken of de fout daarmee kan worden opgelost. Kan dit niet, of is de routine niet aanwezig, dan wordt teruggegaan in de **calling chain** en dus naar de procedure, die de andere waar de fout optrad, aanriep. Als er in deze procedure een error handler aanwezig is, йn de fout kan worden opgelost, dan blijft het programma gewoon functioneren. Ander is het over: een melding en wegwezen. Deze methode werkt alleen wanneer er inderdaad sprake is van procedures die elkaar aanroepen. Dit is niet altijd het geval!

Vandaar een tweede techniek. Wanneer nu de opgetreden fout kan worden meegestuurd vanuit de procedure naar een algemene functie? Deze functie bekijkt dan wat er aan de hand is, een geeft vervolgens een waarde terug. Op basis van deze waarde weet de procedure hoe verder te gaan. Voordeel is duidelijk: op ййn centrale plaats staan de soorten fouten die in de applicatie kunnen optreden, terwijl de code voor elke foutafhandeling gekopieerd kan worden (die verandert niet meer).

Voorbeeld: (we kunnen de code overnemen in een UserForm – let op naamgeving)

```
Dim intFoutActie As Integer      'Var. voor afhandelingsnummer - zie module-

Private Sub cmdOmzetten_Click()
  Dim curInvoer As Currency    'Var. voor bewaren ingevoerde bedrag.
  Dim curUitkomst As Currency  'Var. voor bewaren omgezette bedrag.

  On Error GoTo Fout

  If IsNumeric(txtInvoer.Text) Then     'Controle op nummerieke invoer.
    curInvoer = CCur(txtInvoer.Text)  'Toekennen waarde.
      curUitkomst = Omzetten(curInvoer) 'Uitrekenen omgezette bedrag.

      'Tonen van het omgerekende bedrag in msgbox en statusbalk.
      'Inclusief opmaak.
      lblUitkomst = "Euro: " & Format(curUitkomst, "##,##0.00")

  Else    'Geen nummeriek gegeven.

    MsgBox "U dient een getal in te vullen!", vbOKOnly + vbInformation,
"Bedrag invullen a.u.b"

  End If  'Einde controle nummeriek gegeven.

   'plaatsen van de focus, nieuwe invoer kan gelijk worden ingeveven.
   With txtInvoer
       .SetFocus
       .SelStart = 0
       .SelLength = Len(.Text)
   End With

  Exit Sub

  Fout:
  intFoutActie = FoutAfhandeling(Err.Number)
  Select Case intFoutActie
      Case cEND
       End
      Case cRESUME
       Resume
      Case cRESUMENEXT
       Resume Next
      Case cEXIT
       Exit Sub
  End Select

End Sub
```

```
Private Function Omzetten(curBedrag As Currency) As Currency
  'Algemene omrekenfunctie voor het omzetten van Guldens naar Euro.

  Const dblFactor As Double = 2.20371 '1 EURO = 2.20371 GULDEN.
  Omzetten = curBedrag / dblFactor
End Function
```

Voorbeeld - vervolg: (we kunnen de code overnemen in een algemene Module – let op naamgeving)

```
Option Explicit
' * Algemene variablen

Public Const cEXIT As Integer = 3
Public Const cRESUMENEXT As Integer = 2
Public Const cRESUME As Integer = 1
Public Const cEND As Integer = 0

Public Function FoutAfhandeling(ErrNum As Integer) As Integer
Dim intAntwoord As Integer

Select Case ErrNum
  Case 6 'Overflow.

      MsgBox "Het ingevoerde bedrag is te groot." & vbCrLf & _
             "Voer een kleiner bedrag in!", vbOKOnly + vbInformation, _
             "Te groot bedrag"
      With frmEuro.txtInvoer
          .SetFocus
          .SelStart = 0
          .SelLength = Len(.Text)
      End With

      FoutAfhandeling = cEXIT

  Case 13 'Conversieprobleem.

      MsgBox "Er is een conversieprobleem." & vbCrLf & _
             "Wil je een ander bedrag proberen?", vbOKOnly + vbInformation,
_
             "Conversieprobleem"
      With frmEuro.txtInvoer
          .SetFocus
          .SelStart = 0
          .SelLength = Len(.Text)
      End With

      FoutAfhandeling = cEXIT

  Case Else 'Overige situaties.

      MsgBox "Neem contact op met Parity Solutions B.V." & vbCrLf & _
             "Er is een onherstelbare fout opgetreden!" & vbCrLf & _
             Err.Number & vbCrLf & Err.Description, _
             vbOKOnly + vbCritical, "Oeps..."
      FoutAfhandeling = cEND

End Select
End Function
```

25 Nuttige Excel VBA links

http://www.rondebruin.nl/sendmail.htm	Voor meer voorbeelden voor het verzenden van e-mail
http://www.fhvzelm.com/	Voorbeeldbestanden **VBA**
http://www.linkedin.com/groups/**Excel**-en-**VBA**-Nederland-3792893?mostPopular=&gid=3792893	LinkedIn forum voor **Excel** problemen