**Konstanten**

Deklarieren Sie, wo immer möglich, Konstanten statt Variablen.

**[**[**Bearbeiten**](http://de.wikibooks.org/w/index.php?title=VBA_in_Excel_-_Grundlagen:_Code-Optimierung&action=edit&section=2)**] Objektindex**

Wenn es die Klarheit des Codes nicht stört, verwenden Sie bei Objekt-Schleifen den Index des Objektes, nicht den Namen.

Worksheets(intCounter)

ist schneller als

Worksheets("Tabelle1")

Allerdings gehen For-Each-Schleifen vor, denn

For Each wksData In Worksheets:Next

ist schneller als

Worksheets(intCounter)

**[**[**Bearbeiten**](http://de.wikibooks.org/w/index.php?title=VBA_in_Excel_-_Grundlagen:_Code-Optimierung&action=edit&section=3)**] Direkte Objektzuweisungen**

Verwenden Sie keine allgemeinen Objektzuweisungen wie:

Dim wksData As Object

Deklarieren Sie korrekt:

Dim wksData As Worksheet

Dies hat auch den Vorteil, dass IntelliSense nach Eingabe eines Punktes Vorschläge machen kann, welche Eigenschaften und Methoden zu dem Objekt passen. Wenn die Objekte einer anderen Anwendung entstammen (z.B. Word oder Access), muss zunächst der Verweis auf die Objektbibliothek eingefügt werden, damit Intellisense funktioniert.

**[**[**Bearbeiten**](http://de.wikibooks.org/w/index.php?title=VBA_in_Excel_-_Grundlagen:_Code-Optimierung&action=edit&section=4)**] Selektieren**

Wählen Sie keine Arbeitsmappen, Blätter, Bereiche oder andere Objekte aus:

Workbooks("Test.xls").Activate

Worksheets("Tabelle1").Select

Range("A1").Select

ActiveCell.Value = 12

Referenzieren Sie stattdessen exakt:

Workbooks("Test.xls").Worksheets("Tabelle1").Range("A1").Value = 12

**[**[**Bearbeiten**](http://de.wikibooks.org/w/index.php?title=VBA_in_Excel_-_Grundlagen:_Code-Optimierung&action=edit&section=5)**] Keine eckigen Klammern**

Verwenden Sie für Zellbereiche nicht die Schreibweise in eckigen Klammern:

[b3] = [d4]

Schreiben Sie stattdessen (Ausführungszeit ca. 66% von vorigem):

Range("B3").Value = Range("D4").Value

Noch etwas schneller (Ausführungszeit ca. 90% von vorigem bzw. 60% von ersterem):

Cells(3,2).Value = Cells(4,4).Value ' Cells(ZeilenNr, SpaltenNr)

*Hinweis: Beachten Sie, dass bei Angabe des Zellbezug als String die Range-Eigenschaft verwendet werden muss,* *wohingegen bei der Angabe als Zahlen die Cells-Eigenschaft verwendet werden muss.*

**[**[**Bearbeiten**](http://de.wikibooks.org/w/index.php?title=VBA_in_Excel_-_Grundlagen:_Code-Optimierung&action=edit&section=6)**] Direkte Referenzierung**

Referenzieren Sie - wenn der Programmablauf es nicht erforderlich macht - nicht hierarchieweise:

Set wkbData = Workbooks("Test.xls")

Set wksData = wkbData.Worksheets("Tabelle1")

Set rngData = wksData.Range("A1:F16")

Referenzieren Sie stattdessen direkt das Zielobjekt:

Set rngData = Workbooks("Test.xls").Worksheets("Tabelle1").Range("A1:F16")

**[**[**Bearbeiten**](http://de.wikibooks.org/w/index.php?title=VBA_in_Excel_-_Grundlagen:_Code-Optimierung&action=edit&section=7)**] Dimensionierung**

Dimensionieren Sie die Variablen nicht größer als dies erforderlich ist:

Dim intCounter As Integer

ist schneller als:

Dim varCounter as Variant

Vorsicht: Wenn eigentlich der Datentyp Byte ausreichen sollte, kann eine Subtraktion manchmal einen Unterlauf verursachen. Die Gefahr besteht vor allem bei FOR-Schleifen mit einem negativen Argument für STEP. In diesem Falle bei INTEGER bleiben.

Tipp: Noch etwas schneller als der Integer ist der Datentyp Long! Das liegt vermutlich daran, dass Integer 16-bittig ist während Long 32-bittig ist und alle neueren Prozessoren für 32-Bit optimiert sind.

**[**[**Bearbeiten**](http://de.wikibooks.org/w/index.php?title=VBA_in_Excel_-_Grundlagen:_Code-Optimierung&action=edit&section=8)**] With-Rahmen**

Verwenden Sie With-Rahmen. Langsam ist:

Worksheets("Tabelle1").Range("A1:A16").Font.Bold = True

Worksheets("Tabelle1").Range("A1:A16").Font.Size = 12

Worksheets("Tabelle1").Range("A1:A16").Font.Name = "Arial"

Worksheets("Tabelle1").Range("A1:A16").Value = "Hallo!"

Schneller ist:

With Worksheets("Tabelle1").Range("A1:A16")

With .Font

.Bold = True

.Size = 12

.Name = "Arial"

End With

.Value = "Hallo!"

End With

**[**[**Bearbeiten**](http://de.wikibooks.org/w/index.php?title=VBA_in_Excel_-_Grundlagen:_Code-Optimierung&action=edit&section=9)**] Excel-Funktionen**

Ziehen Sie Excel-Funktionen VBA-Routinen vor. Langsam ist:

For intCounter = 1 To 20

dblSum = dblSum + Cells(intCounter, 1).Value

Next intCounter

Schneller ist:

dblSum = WorksheetFunction.Sum(Range("A1:A20"))

Wenn Sie große, zusammenhängende Zellbereich berechnen müssen, setzen Sie zur eigentlichen Berechnung Excel-Formeln ein. Die Formeln können Sie danach in absolute Werte umwandeln:

Sub Berechnen()

Dim intRow As Integer

intRow = Cells(Rows.Count, 1).End(xlUp).Row

Range("C1").Formula = "=A1+B1/Pi()"

Range("C1:C" & intRow).FillDown

Columns("C").Copy

Columns("C").PasteSpecial Paste:=xlValues

Application.CutCopyMode = False

Range("A1").Select

End Sub

Dasselbe Ergebnis hat folgende Prozedur, die auch With-Klammern verwendet und bei der Ersetzung der Formeln durch Werte ohne Copy/PasteSpecial auskommt:

Sub Berechnen2()

Dim lngRow As Long

lngRow = Cells(Rows.Count, 1).End(xlUp).Row

With Range("C1:C" & lngRow)

.Formula = "=A1+B1/Pi()" ' trägt die Formeln ein

.Formula = .Value ' ersetzt die Formeln durch Werte; .Value = .Value geht auch

End With

Range("A1").Select ' nur, wenn das nötig/erwünscht ist

End Sub

Tipp: Wenn Sie auf eine große Anzahl Zellen zugreifen müssen, dann ist es am Schnellsten, wenn Sie die Werte mit einem Befehl in ein Array kopieren und dann aus dem Array lesen:

Sub Berechne3()

dim a

dim i as long, j as long, sum as long

a = me.Range("A1:H800").value

for i=1 to 8

for j=1 to 800

sum=sum+a(j,i) ' a(ZeilenNr, SpaltenNr)

next j

next i

debug.print sum

End Sub

**[**[**Bearbeiten**](http://de.wikibooks.org/w/index.php?title=VBA_in_Excel_-_Grundlagen:_Code-Optimierung&action=edit&section=10)**] Array-Formeln**

Setzen Sie temporäre Excel-Array-Formeln zur Matrixberechnung ein. Wenn Sie in VBA zwei Zellbereiche auf Übereinstimmung überprüfen wollen, müssen Sie einzelne Zellvergleiche vornehmen. Mit Einsatz einer Excel-Array-Formel sind Sie schneller. Im nachfolgenden Code werden zwei große Zellbereiche auf Übereinstimmung überprüft. Über VBA müsste man jede einzelne Zelle des einen mit der des anderen Bereiches vergleichen. Die Excel-Array-Formel liefert das Ergebnis unmittelbar nach dem Aufruf:

Function MatrixVergleich(strA As String, strB As String) As Boolean

Range("IV1").FormulaArray = "=SUM((" & strA & "=" & strB & ")\*1)"

If Range("IV1").Value - Range(strA).Cells.Count = 0 Then

MatrixVergleich = True

End If

Range("IV1").ClearContents

End Function

Sub Aufruf()

MsgBox MatrixVergleich("C1:D15662", "E1:F15662")

End Sub