MENGOPTIMALKAN VBA

Ketika mengembangkan aplikasi yang besar, Anda mungkin akan melibatkan banyak modul, user form, prosedur sub routine dan fungsi, Anda bukan hanya perlu memastikan bahwa aplikasi tersebut bebas dari error, tetapi Anda juga harus memastikan bahwa kinerja aplikasi tersebut akan berjalan dengan baik.

Bab ini menjelaskan berbagai tips dan trik untuk mengoptimalkan kinerja Excel VBA sehingga kode Anda dapat berjalan lebih cepat.

Latihan file dalam bab ini dapat diunduh dari tautan berikut.

http://rumpitekno.com/download-page/?did=32

Menggunakan Option Explicit

Secara default, VBA *tidak* mengharuskan Anda untuk mendeklarasikan variabel dengan menggunakan pernyataan **Dim**. Apabila compiler bertemu dengan variabel dengan nama (misalnya **NoKTP**) yang belum dideklarasikan sebelumnya sebagai variabel, atau sebagai salah satu kata kunci (*keyword*) VBA, atau properti atau metode dari typelib yang direferensikan, maka compiler *secara otomatis* akan membuat sebuah variabel baru dengan nama tersebut.

Meski dilakukan secara otomatis, tidak menutup kemungkinan terdapat kesalahan pemberian kode yang sulit untuk ditemukan. Misalnya, jika Anda mendeklarasikan sebuah variabel **Jumlah** dan kemudian menulis variabel itu sebagai **Jmlah** (kesalahan ejaan), VBA tidak akan menandainya sebagai kesalahan.

Sebaliknya, VBA akan membuat variabel baru dengan nama **Jmlah** tersebut.

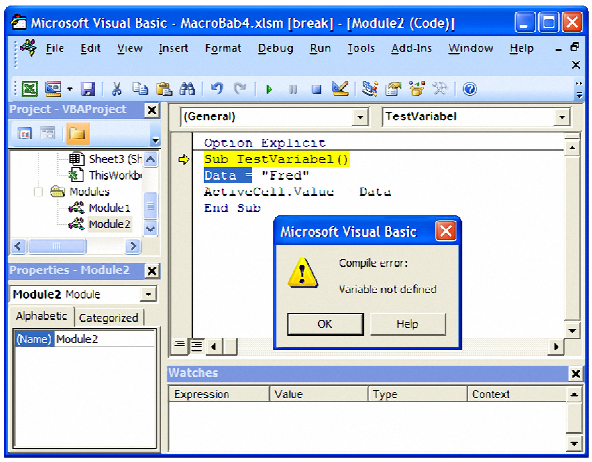
Ini berarti variabel **Jumlah** tidak akan diinisialisasi dengan nilai yang diharapkan karena nilai tersebut diberikan ke variabel **Jmlah** dan jika program itu panjang, Anda mungkin akan mengalami kesulitan menemukan kesalahan tersebut.

Anda dapat mencegah kesalahan ini dengan mengharuskan semua variabel dideklarasikan dengan pernyataan **Dim** dan menulis **Explicit Option** di baris pertama dari kode Anda.

```
Explicit Option
```

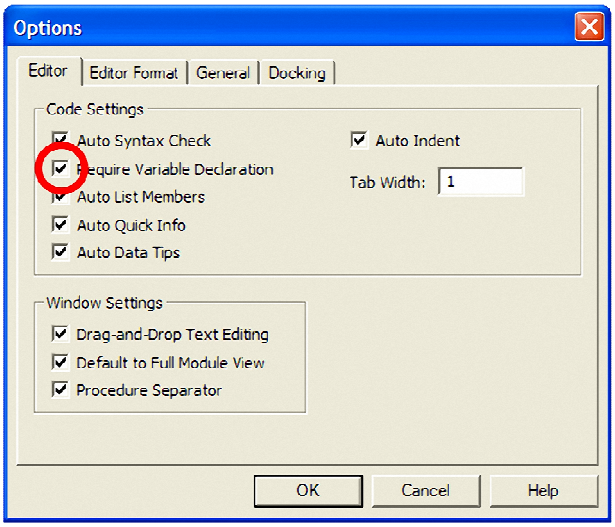
Jika Anda menulis **Explicit Option** di baris pertama macro Anda, berarti Anda harus mendeklarasikan semua variabel dan konstanta yang akan digunakan.

Jika Anda *tidak* mendeklarasikan variabel/konstanta, Anda akan mendapatkan pesan kesalahan seperti contoh berikut.



Anda dapat menambahkan pernyataan **Option Explicit** secara otomatis di semua kode baru Anda (tetapi tidak untuk kode yang sudah ada karena harus dilakukan secara manual) dengan memilih menu **Tools** > **Options** kemudian pilih tab **Editor** dan berikan tanda centang untuk **Require Variable Declaration**.

Ketika Anda membuat sebuah macro baru, pernyataan **Option Explicit** secara otomatis akan disisipkan pada baris pertama macro Anda.



Selalu Hindari Menggunakan Tipe Variant

VBA mendukung tipe data **Variant** yang dapat menyimpan semua jenis data. Jika Anda mengabaikan klausul **As [tipe]** dalam deklarasi variabel, **Variant** akan menjadi tipe default.

Namun cara penulisan ini memerlukan waktu pemrosesan yang lebih lama karena compiler harus menambahkan kode tambahan untuk menguji jenis data yang disimpan dalam variabel tersebut.

Selain itu, variabel bertipe **Variant** membutuhkan lebih banyak memori. Oleh karena itu, aplikasi Anda akan menjadi lebih efisien jika Anda mendeklarasikan tipe data untuk masing-masing variabel yang digunakan secara *eksplisit*.

Dengan mendeklarasikan variabel secara *eksplisit* berarti Anda mengurangi kemungkinan timbulnya konflik penggunaan nama variabel yang sama atau kesalahan ejaan.

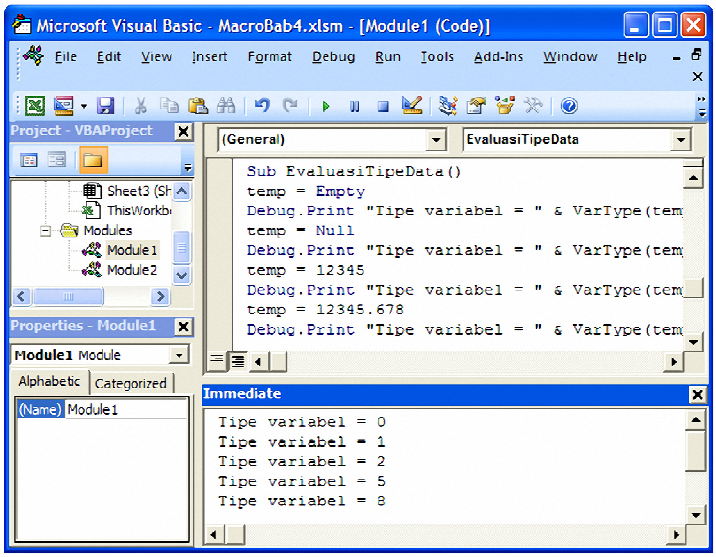
Hal ini dilakukan untuk memberikan "**Return Value**" yang berbeda untuk tiap jenis data dalam variabel tersebut.

Tipe Data	Return Value
Empty	0
Null	1

Integer	2
Long	3
Single	4
Double	5
Currency	6
Date/Time	7
String	8

Untuk mengecek nilai "Return Value" dari sebuah variabel **Variant**, Anda dapat menggunakan perintah **VarType** seperti berikut.

```
Sub EvaluasiTipeData()
temp = Empty
Debug.Print "Tipe variabel = " & VarType(temp)
temp = Null
Debug.Print "Tipe variabel = " & VarType(temp)
temp = 12345
Debug.Print "Tipe variabel = " & VarType(temp)
temp = 12345.678
Debug.Print "Tipe variabel = " & VarType(temp)
temp = ""
Debug.Print "Tipe variabel = " & VarType(temp)
temp = "Indonesia"
Debug.Print "Tipe variabel = " & VarType(temp)
temp = #8/1/1973#
Debug.Print "Tipe variabel = " & VarType(temp)
End Sub
```



Hindari Kesalahan Menggunakan Satu Dim saat Deklarasi Variabel

Anda dapat mendeklarasikan lebih dari satu variabel dengan mengunakan satu pernyataan **Dim**. Namun sebelumnya Anda harus memahami bagaimana variabel tersebut akan dideklarasi jenis tipenya. Perhatikan kode berikut.

```
Dim J, K, L As Integer
```

Anda mungkin berpikir bahwa ketiga variabel akan dideklarasikan sebagai tipe **Integer**. Tetapi sesungguhnya hanya variabel **L** yang dideklarasikan sebagai tipe **Integer**, sedangkan variabel **J** dan **K** dideklarasikan sebagai tipe **Variant**. Deklarasi ini secara fungsional akan memiliki arti yang sama seperti berikut ini.

```
Dim J As Variant, K As Variant, L As Integer
```

Anda harus menggunakan klausa **As [tipe]** untuk setiap variabel yang akan dideklarasikan apabila menggunakan satu pernyataan **Dim** sebagai berikut.

```
Dim J As Integer, K As Integer, L As Integer
```

Membuat Kode VBA Anda Lebih Mudah Dibaca dengan Notasi Hungarian

Kode VBA Anda akan lebih mudah dibaca apabila Anda menggunakan Notasi Hungarian dimana setiap nama variabel akan diawali dengan huruf yang mengidentifikasi jenis data. Misalnya, variabel bertipe **Integer** akan diberi nama **intCounter**, di mana awalan **int** menunjukkan bahwa variabel ini bertipe **Integer**.

Jenis data	Awalan	Contoh
Boolean	bln	blnLanjut
Byte	byt	bytNilaiWeekday
Collection object	col	colWidgets
Currency	cur	curBiaya
Date (Time)	dtm	dtmTglPesan
Double	dbl	dblSales
Error	err	errNoFaktur
Integer	int	intKuantitas
Long	lng	lngJarak
Object	obj	objWordDoc
Single	sng	sngNilaiPpn

String	str	strNamaCustomer
Variant	vnt	vntColumnData

Gunakan Konstanta apabila Memungkinkan

Konstanta merupakan sebuah variabel yang tidak dapat diubah nilainya ketika program sedang berjalan. Ada dua jenis konstanta yaitu *konstanta intrinsik* dan *konstanta user*. Konstanta intrinsik adalah konstanta yang disediakan oleh VBA dengan kata kunci **vb** di bagian awal konstanta (misalnya, VbYesNo) atau konstanta yang disediakan oleh Excel dengan kata kunci **xl** di bagian awal konstanta (misalnya, xlDialogOpen).

Sedangkan konstanta user adalah konstanta yang dapat Anda buat sendiri dengan kata kunci **Const**. Apabila memungkinkan, deklarasikan nilai yang digunakan sebagai konstanta dan bukan sebagai variabel. Karena nilai konstanta tidak berubah, maka nilai konstanta hanya akan dievaluasi satu kali ketika kode Anda dikompilasi, sementara nilai variabel akan dievaluasi setiap kali digunakan pada *run time*.

Matikan Fitur yang Tidak Digunakan sebelum Menjalankan Macro Anda

Ketika macro Anda sedang berjalan, Excel secara otomatis melakukan banyak fungsi sekaligus seperti memperbarui layar spreadsheet, menghitung ulang rumus, menampilkan peringatan di layar dll. Anda dapat menonaktifkan fitur ini sebelum mulai menjalankan macro sehingga dapat mempercepat kode Anda bekerja.

```
Application.ScreenUpdating = False
Application.DisplayAlerts = False
```

Namun, jika macro yang Anda jalankan akan mengubah salah satu bagian dari spreadsheet (misalnya animasi, grafik), maka Anda tidak dianjurkan untuk menonaktifkan fitur ScreenUpdating. Pastikan Anda telah mengaktifkan kembali semua fitur pada akhir macro Anda. Gunakan kode berikut untuk mengaktifkan peringatan, ScreenUpdating dan lain-lain.

```
Application.ScreenUpdating = True
Application.DisplayAlerts = True
```

Jangan Memilih Sel dan (atau) Object

Kode berikut ini akan berjalan lebih cepat dari kode VBA kedua.

```
Range("A1").value = 10
```

Kode VBA kedua ini akan berjalan lebih lambat.

```
Range("A1").select
Selection.Value = 10
```

Umumnya pengguna akan memilih sel dan objek dalam kode ketika menggunakan perekam macro. Namun sebenarnya, Anda tidak perlu memilih sel untuk mengakses atau memperbarui nilai dari sel tersebut.

Selalu Gunakan Fungsi dan Fitur bawaan

Jangan membuat satu fungsi dari awal (user defined function) apabila telah disediakan formula dan fungsi bawaan (built-in) yang dapat Anda gunakan. Object seperti **Application** memiliki banyak metode yang berguna yang dapat melakukan apa yang Anda inginkan. Misalnya, jika Anda ingin memeriksa apakah 2 rentang tumpang tindih, Anda dapat menggunakan **Application.intersect** tanpa harus membuat satu fungsi baru dari awal.

Mengakses Sel dalam Rentang

Anda tidak perlu menggunakan metode **Cells** untuk mengakses sel tertentu dalam satu kisaran sel. Misalnya, Anda dapat referensi sel **B1** dengan kode berikut.

```
Range("MyRange")(1,2)
```

Anda juga dapat merujuk ke sel B1 dengan kode berikut.

```
Range("MyRange").Cells(1,2)
```

VBA memungkinkan Anda merujuk ke sel A1 dengan menulis **[A1]** yang lebih mudah daripada menulis **Range("A1")**. Walaupun penulisan **[A1]** lebih mudah, tetapi cara pintas ini justru akan membuat eksekusi berjalan lebih lambat dibandingkan dengan sintaks **Range("A1").**

Mode Calculation

Biasanya Excel akan menghitung ulang (secara otomatis) sebuah sel/rentang sel ketika nilai pada sel/rentang sel yang terkait berubah. Jika buku kerja Anda sering menghitung ulang, hal ini dapat memperlambat kinerja aplikasi Anda. Untuk mencegah modus perhitungan ulang secara otomatis, gunakan pernyataan berikut.

```
Application.Calculation = xlCalculationManual
```

Namun pada akhir kode Anda, pastikan Anda telah mengembalikan mode perhitungan ke otomatis dengan pernyataan berikut.

```
Application.Calculation = xlCalculationAutomatic
```

Ketika mode perhitungan dalam mode manual (xlCalculation Manual), Excel tidak akan memperbarui nilai di sel. Jika macro Anda bergantung pada nilai sel yang diperbarui, Anda harus memaksa event **Calculate** dengan metode **Calculate**, untuk dapat diterapkan ke satu rentang sel tertentu (Range ("MyRange"). Calculate) atau ke seluruh Workbook (Calculate).

Koleksi Indeks

Item individual dari satu koleksi objek dapat diakses melalui nama atau indeks koleksinya.

Misalnya, jika Anda memiliki tiga lembar kerja ("Sheet1", "Sheet2", dan "Sheet3") dalam buku kerja, Anda dapat referensi "Sheet2" dengan menulis kode berikut.

```
Worksheets("Sheet2")
```

Anda juga dapat referensi "Sheet2" dengan menulis kode berikut.

```
Worksheets(2)
```

Secara umum, metode indeks angka, Worksheets(2) jauh lebih cepat daripada metode indeks nama, Worksheets("Sheet2").

Namun jumlah dan urutan item dalam koleksi objek dapat berubah-ubah, sehingga biasanya akan lebih aman dan mudah untuk merujuk pada item dalam koleksi berdasarkan nama daripada berdasarkan nomor indeksnya.

Gunakan Objek Range dan Bukan Selection

Pada saat mulai bekerja dengan kisaran sel, umumnya Anda tidak perlu memilih kisaran yang terkait. Sehingga akan lebih efisien untuk menulis kode sebagai berikut.

```
Range("A1").Font.Bold = True
```

Kode berikut bekerja lebih lambat karena melibatkan dua langkah.

```
Range("A1").Select
Selection.Font.Bold = True
```

"Early Binding"

Jika Anda akan bekerja dengan aplikasi lain, misalnya dengan Word, deklarasikan objek OLE Anda secara langsung dan bukan sebagai variabel bertipe Object karena kebanyakan *overhead* akan dilakukan pada waktu kompilasi ("Early Binding") dan bukan pada saat macro dijalankan ("Late Binding") dengan menulis kode sebagai berikut.

```
Dim WordObj As Word.Application
```

Hindari menulis kode seperti berikut.

```
Dim WordObj As Object
```

Mencegah Penghitungan Ketika Menjalankan Kode Macro

Jika pelaksanaan kode dalam macro Anda melibatkan banyak penghitungan, Anda dapat mengoptimalkan kode VBA dengan menonaktifkan penghitungan otomatis dan aktifkan kembali penghitungan setelah selesai menjalankan macro Anda.

```
Sub NonAktifkanPenghitungan()
Application.Calculation = xlCalculationManual
'Kode macro
Application.Calculation = xlCalculationAutomatic
End Sub
```

Screen Updating

Anda dapat menonaktifkan fitur *screen updating* sehingga Excel tidak memperbarui gambar di layar setiap kali kode Anda dijalankan.

```
Application.ScreenUpdating = FALSE
```

Hal ini dapat mempercepat kinerja kode Anda. Pastikan untuk mengembalikan pengaturan ke **True** di akhir macro Anda.

```
Application.ScreenUpdating = TRUE
```

Memberikan Nilai Nol pada Variabel String

Apabila Anda ingin memberikan nilai nol pada satu variabel string, akan lebih mudah jika Anda menggunakan konstanta vbNullString daripada mendeklarasikan variabel string (misalnya, strWords="").

vbNullString akan diproses sedikit lebih cepat daripada "" karena vbNullString sebenarnya bukanlah string, tetapi satu konstanta yang berukuran 0 byte, sedangkan "" adalah satu variabel string yang berukuran 4-6 byte.

```
Sub TeksKosong()
Dim strWords As String
'Memberikan varibel string dengan nilai "Elex"
strWords = "Elex"
MsgBox strWords
'Memberikan varibel string dengan nilai nol
strWords = vbNullString
MsgBox strWords
End Sub
```

Pengulangan FOR EACH

Jika Anda bermaksud untuk membuat pengulangan melalui satu koleksi, biasanya akan lebih cepat dengan menggunakan pernyataan **For/Each** daripada menggunakan cara indeks (**For/Next**). Berikut ini contoh dimana pengulangan kode pertama bekerja lebih cepat daripada contoh yang kedua.

Contoh 1:

```
Dim WS as Worksheet
For Each WS In Worksheets
MsgBox WS.Name
Next WS
```

Contoh 2:

```
Dim I as Integer
For I = 1 To Worksheets.Count
MsgBox Worksheets(i).Name
Next i
```

Gunakan Objek Sederhana, Hindari Objek Compound

Jika Anda harus mereferensikan satu objek berulang kali, seperti satu rentang sel, deklarasikan sebuah objek dari tipe itu, lalu set ke objek target dan kemudian gunakan objek tersebut untuk merujuk ke target yang diinginkan.

Contoh:

```
Dim MyCell As Range
Set MyCell = Workbooks("Book2").Worksheets("Sheet3").Range("C3")
...'perintah lanjutan
MyCell.Value = 123
```

Dengan merujuk langsung ke MyCell, VBA dapat mengakses objek secara langsung.

Cara ini lebih mudah daripada setiap kali Anda harus melengkapi path lengkap ke objek. Metode ini hanya berguna saat Anda perlu mengakses sebuah objek beberapa kali selama eksekusi kode.

Deklarasi Objek sesuai Jenisnya

Jika memungkinkan hindari penggunaan data yang bertipe **Object** atau **Variant**. Jenis data ini membutuhkan overhead yang cukup banyak untuk menentukan jenisnya. Sebaliknya, gunakan deklarasi *eksplisit* seperti berikut.

```
Dim MySheet As Worksheet
```

Hindari menulis kode sebagai berikut, di mana Anda mendeklarasikan data sebagai Object.

```
Dim MySheet As Object
```

Gunakan deklarasi eksplisit seperti mendeklarasikan NumRows bertipe Long.

```
Dim NumRows As Long
```

Hindari mendeklarasikan variabel sebagai Variant.

```
Dim NumRows As Variant
```

Menggunakan Pernyataan WITH

Jika Anda menggunakan beberapa pernyataan dalam satu baris yang berlaku untuk objek yang sama, akan lebih baik jika Anda

menggunakan pernyataan **WITH** daripada harus setiap kali merujuk ke objek tersebut.

Prosedur berikut akan memformat sel yang dipilih dengan font "Times Roman ', ukuran font 12, Bold, Italic dan berwarna biru.

```
Sub ChangeFont()
Selection.Font.Name = "Times New Roman"
Selection.Font.Size = 12
Selection.Font.Bold = True
Selection.Font.Italic = True
Selection.Font.ColorIndex = 5
End Sub
```

Prosedur di atas dapat ditulis ulang menggunakan blok **WITH**.

```
Sub ChangeFont()
With Selection.Font
.Name = "Times New Roman"
.Size = 12
.Bold = True
.Italic = True
.ColorIndex = 5
End With
End Sub
```

Menghapus Undo Stack dengan Macro

Ketika Anda bekerja dengan Microsoft Word, Anda dapat menggunakan metode **UndoClear** dengan objek **ActiveDocument** untuk menghapus tumpukan Undo.

Namun, Excel VBA tidak menyediakan metode seperti UndoClear karena tumpukan Undo secara otomatis dihapus (*dibersihkan*) oleh Excel setiap kali macro Anda membuat perubahan pada buku kerja.

Jika macro Anda tidak membuat perubahan dan Anda masih ingin membersihkan tumpukan Undo, maka yang perlu Anda lakukan adalah membuat sebuah perubahan pada lembar kerja.

Macro berikut ini akan menyalin isi sel A1 kembali ke A1 dan selama proses ini secara tidak langsung akan membersihkan tumpukan undo:

```
Sub ClearUndo ()
Range("A1").Copy Range("A1")
End Sub
```

Fungsi Lembar Kerja

Anda dapat menggunakan fungsi bawaan Excel dalam kode VBA Anda. Instruksi yang menggunakan fungsi bawaan Excel akan dieksekusi dengan lebih cepat daripada fungsi UDF yang harus ditafsirkan terlebih dahulu setiap kali macro berjalan.

Contoh:

Anda dapat menggunakan fungsi **SUM()** dengan menulis kode berikut.

```
MySum = Application.WorksheetFunction.Sum(Range("A1:A100"))
```

Anda tidak perlu menulis ulang kode untuk penjumlahan sebagai berikut.

```
For Each C In Range("A1:A100")
MySum = MySum + C.Value
Next C
```

Hindari Menggunakan Pernyataan If Else

Seringkali pengguna akan menggunakan pernyataan **If Else** untuk menguji apakah satu keadaan itu **TRUE** atau **FALSE**. Namun ada satu cara yang bekerja sedikit lebih cepat yang dapat Anda gunakan. Contoh pertama menunjukkan cara yang umum digunakan, sedangkan cara yang kedua akan bekerja lebih cepat.

```
Sub CaraYesNoLambat()
Dim bYesNo As Boolean
Dim i As Integer
If i = 5 Then
  bYesNo = True
Else
  bYesNo = False
End If
MsgBox bYesNo
End Sub
```

Cara yang lebih cepat dapat ditulis dengan kode VBA yang lebih optimal seperti berikut.

```
Sub CaraTrueFalseCepat()
Dim bYesNo As Boolean
Dim i As Integer
bYesNo = (i = 5)
  MsgBox bYesNo
End Sub
```

Seringkali dalam keadaan tertentu pengguna perlu untuk mengubah nilai satu variabel dari True ke False seperti contoh kode VBA berikut.

```
Sub ToggleTrueFalseLambat()
Dim bYesNo As Boolean
If bYesNo = False Then
  bYesNo = True
Else
  bYesNo = False
End If
MsgBox bYesNo
End Sub
```

Kode VBA tersebut dapat ditulis dengan lebih optimal seperti berikut.

```
Sub ToggleTrueOrFalseCepat()
Dim bYesNo As Boolean
bYesNo = Not bYesNo
MsgBox bYesNo
End Sub
```

Menganalisis Logika Macro Anda

Sebelum mengoptimalkan kode VBA Anda, Anda harus terlebih dahulu mengoptimalkan logika macro Anda. Tanpa logika yang baik, kode VBA yang ditulis tidak akan memiliki nilai yang optimal. Luangkan waktu untuk meringkas logika program Anda sehingga Anda bisa mendapatkan performa terbaik dari macro Anda.

Tulis Beberapa Pernyataan pada Baris yang Sama

Ketika macro dijalankan, prosesor VBA hanya akan membaca satu baris macro pada satu waktu. Kode pada baris tersebut akan diinterpretasikan dan dieksekusi sesuai dengan urutan baris penulisan kode. Ketika satu baris selesai dibaca dan dieksekusi, baris berikutnya akan dibaca dan dijalankan, dan seterusnya sampai akhir macro. Untuk mempercepat pembacaan macro, Anda dapat menggabungkan beberapa pernyataan dalam satu baris dan memisahkannya dengan tanda titik dua.

Akibatnya waktu yang dibutuhkan untuk membaca kode macro akan lebih sedikit dan ada lebih banyak waktu yang dialokasikan untuk menjalankan macro tersebut. Dengan demikian, keputusan untuk "tidak mengoptimalkan" atau "mengoptimalkan" kode VBA

akan menghasilkan kode VBA yang lebih sulit dibaca atau menghasilkan kode VBA yang lebih cepat dijalankan.

Namun waktu yang dihemat ketika Anda menggabungkan beberapa pernyataan dalam satu baris yang sama tidak akan berpengaruh besar terutama jika Anda menggunakan komputer yang cepat. Penggabungan beberapa pernyataan dalam satu baris juga tidak dianjurkan apabila bagian dari kode tersebut hanya akan dijalankan sekali saja.

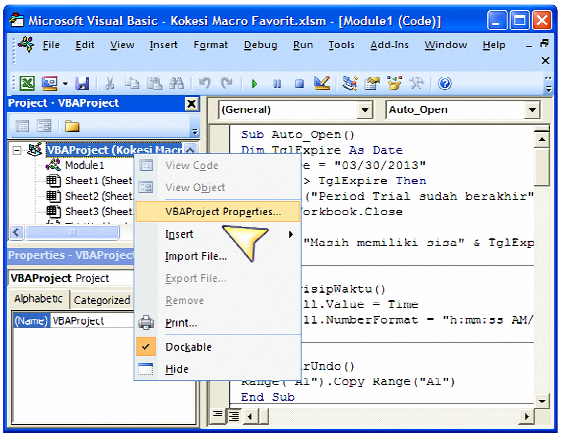
Namun jika kode itu akan diulang ratusan atau ribuan kali selama macro dijalankan, penghematan waktu dari setiap iterasi kode mungkin kecil, tetapi penghematan waktu secara kumulatif akan terlihat cukup substansial. Jika Anda memutuskan untuk "memadatkan" kode VBA Anda dengan menggabungkan beberapa pernyataan dan kata kunci pada satu baris yang sama, yang harus Anda ingat bahwa jumlah maksimal karakter dalam satu baris adalah 255 karakter. Selain itu, pernyataan **Sub** yang digunakan untuk memulai satu macro, harus ditulis pada baris terpisah, seperti juga pernyataan **With** dan pernyataan **End**.

Mengoptimalkan Macro sebagai Add-ins

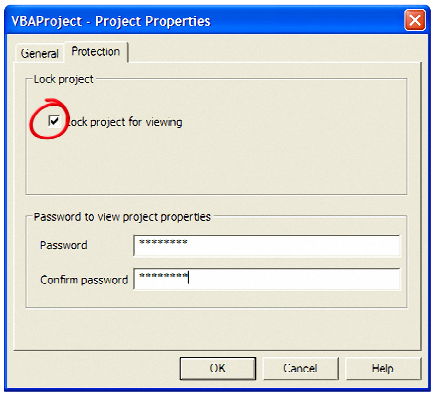
Pasti Anda akan bertanya-tanya bagaimana Anda dapat membuat koleksi macro dan fungsi favorit Anda selalu tersedia setiap kali Anda (atau orang lain) membuka Excel (bahkan pada jaringan lain) tanpa memungkinkan macro tersebut diubah tanpa seijin Anda?

Anda dapat menulis semua macro dan fungsi dalam satu buku kerja lalu menyimpannya sebagai **Add-in**. Untuk membuat Add-in (yang diproteksi dengan kata sandi), buka buku kerja yang digunakan sebagai Add-in. Tekan tombol [**ALT**] + [**F11**] untuk membuka VBE.

Pada jendela **Project** Explorer, klik kanan pada nama projek, misalnya, **VBAProject (Book** 1) lalu pilih **VBAProject Properties** dari menu yang muncul.



Pada kotak dialog **VBAProject-Project Properties** yang muncul, pilih tab **Protection** dan beri tanda centang untuk **Lock project for viewing**, lalu masukkan kata sandi yang diinginkan dan klik **OK** untuk melanjutkan.

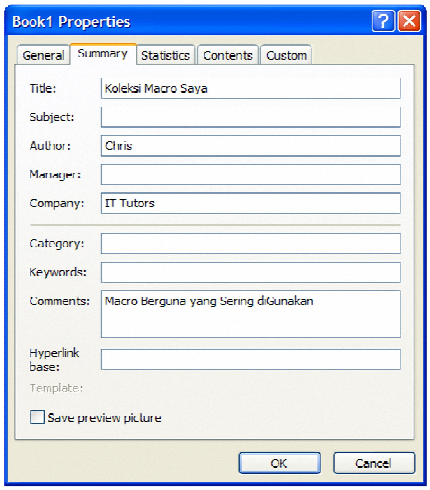


Tekan tombol [**ALT**] + [**Q**] untuk menutup Visual Basic Editor dan kembali ke lembar kerja Excel.

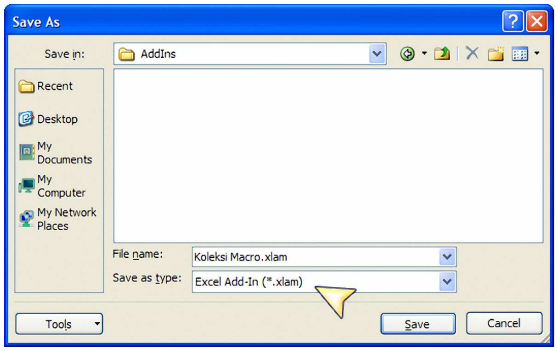
Pilih tombol **Office button** > **Prepare** > **Properties.** Klik panah bawah pada **Document Properties** dan pilih **Advanced Properties**.

Pada kotak dialog **Properties** yang muncul, pilih tab **Summary** lalu masukkan informasi untuk **Title** (semua yang Anda ketik akan muncul dalam kotak dialog Add-Ins yang digunakan oleh Excel) dan **Comments** (semua yang Anda ketik akan muncul di bagian penjelasan kotak dialog Add-Ins yang digunakan oleh Excel).

Klik **OK** untuk melanjutkan.



Lalu simpan workbook tersebut sebagai **Add-in** dengan memilih **Office button** > **Save As** > **Other Formats** lalu pilih **Excel Add-In (*.xlam)** dari **Save as type** dan masukkan nama Add-in Anda pada **File name** dan klik **Save** untuk melanjutkan.

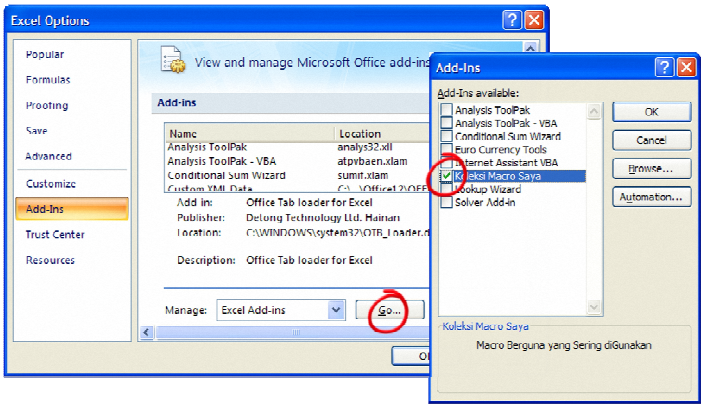


Memuat Add-in secara Otomatis

Jika Anda telah menyimpan koleksi macro favorit Anda sebagai **Add-in**, Anda dapat memuatnya secara otomatis setiap kali Anda membuka Excel.

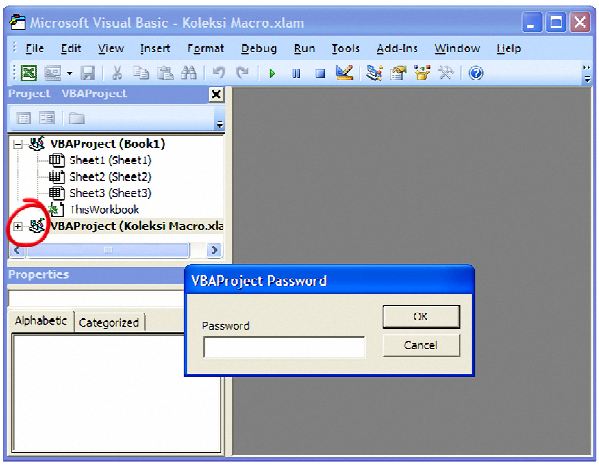
Pilih **Office button** > **Excel Options** > **Add-ins**. Lalu pada bagian **Manage**, pilih **Excel Add-ins** dan klik **Go**.

Ketika kotak dialog **Add-Ins** muncul, beri tanda centang untuk Add-ins, **Koleksi Macro Saya** dan klik **OK** untuk melanjutkan.



Setiap kali Anda membuka Excel, koleksi macro favorit Anda dalam bentuk Add-in akan dimuat secara otomatis.

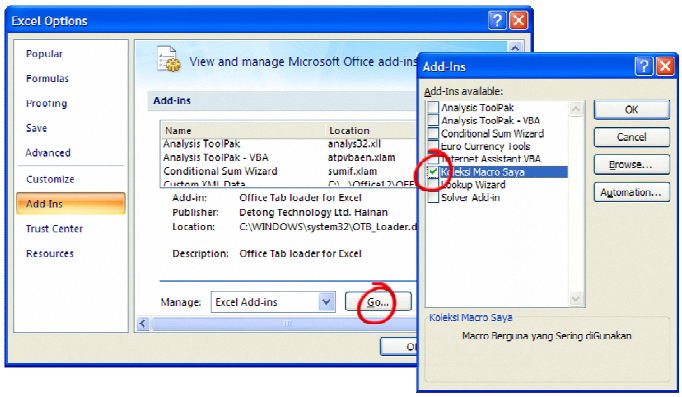
Untuk **memodifikasi koleksi macro** Anda, tekan tombol **[ALT]** + **[F11]** untuk menampilkan VBE. Pada jendela **Project Explorer**, klik **VBAProject (Koleksi Macro.xlam).** VBE akan menampilkan kotak dialog untuk meminta kata sandi Anda sebelum macro tersebut dapat dibuka.



Ketika macro sudah dibuka, maka macro tersebut dapat diakses (masuk kembali ke VBE) selama Excel belum ditutup walaupun Anda sudah keluar dari VBE.

Apabila Anda tidak menginginkan macro dan fungsi Anda dimuat secara otomatis sebagai Add-in, pilih **Office button** > **Excel Options** > **Add-ins**. Lalu pada bagian **Manage**, pilih **Excel Add-ins** dan klik **Go**.

Ketika kotak dialog **Add-Ins** muncul, hilangkan tanda centang untuk Add-ins, **Koleksi Macro Saya** dan klik **OK** untuk melanjutkan.



Untuk memuat Add-in Anda secara *selektif* hanya untuk worksheet tertentu, buka lembar kerja yang diinginkan untuk memuat Add-in. Tekan tombol **[ALT]** + **[F11]** untuk menampilkan VBE. Klik ganda pada objek "**This Workbook**" di jendela **Project Explorer**.

Ketika Excel membuka jendela **Code** untuk **This Workbook**, tulis macro berikut di jendela Code.

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
AddIns("Add-In Name").Installed = False
End Sub
Private Sub Workbook_Open()
AddIns("Add-In Name").Installed = True
End Sub
```

Dalam kode ini, ubah nama Add-in ("**Add-In Nama**") dengan nama Add-in yang ingin Anda gunakan dalam lembar kerja. Tutup Editor VBA dan simpan buku kerja Anda.

* * *