# 6.bölüm

**6**

# Debugging ve Hata Yönetimi

Debuggingi bilmediğim dönemlerde kodun bir yere kadar çalışmasını istediğim zamanlarda oraya bir MsgBox koyardım, MsgBox çıkınca da Ctrl+Break ile kodu durdururdum. Debugging araçları işte bu tür uzun ve kafa yaran süreçlerden kurtulmanızı, kodunuzu daha sağlıklı test etmenizi sağlar.

Hata Yakalama ise, özellikle başkalarının kullanımına açık olacak şekilde kodlama yaptığınızda(Ör: Bir add-in) beklenmeyen durumlarda kodunuzun uygun reaksiyonu vermesini sağlar. Doğru bir hata yönetimi uygulanmazsa kodunuz beklenmedik sonuçlara neden olabilir, yapılan işlem yarıda kalabilir, kullanıcının güveni zedelenebilir.

## Breakpointler ve İzleme Pencereleri

Kodunuzu test ederken belirli noktalarda geçici olarak duraklatan araca **Breakpoint(BP)** diyoruz. Kullanım amacı, kodu durdurduğunuz anda, o an itibarıyla değişkenlerin durumunu görmek olabileceği gibi, o ana kadar kodun ne tür etkiler yarattığını görmek de olabilir. Breakpoint konmuş durumdaki moda **Break modu** denir.

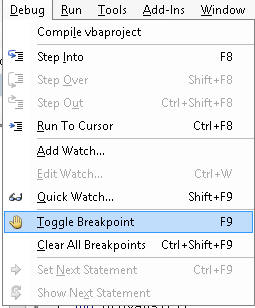
**İzleme pencereleri** ise, ister Break modunda olun ister olmayın kodunuzun o an tam olarak ne yaptığını gösterebileceği gibi tek satırlık bir kodun doğrudan burada çalıştırılmasına da imkan verir.

Şimdi bunlara detaylıca bakalım.

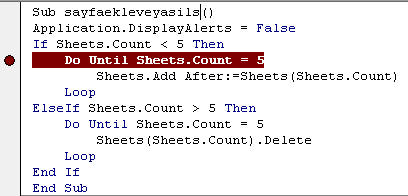
### Breakpointler

**Debug** menüsünden **Toggle Breakpoint** diyerek veya **F9** tuşuna basarak BP koyabiliriz, veya var olan BP'i kaldırabiliriz. Genelde menüyü kullanmak yerine ya F9'a tıklarız, ya da iki aşağıda bulunan resimdeki bordo renkli dairenin bulunduğu yere tıklarız, kaldırmak için bordo daireye tekrar tıklarız.

Kodumuzun farklı yerlerine birden fazla olacak şekilde BP koyabiliriz. Hepsini kaldırmak için menüde **Clear All Breakpoints** diyebilir veya **Ctrl+Shift+F9** tuşlarına basabiliriz.



Örnek bir BP konmuş satırın görüntüsü aşağıdaki gibidir.



Break moddayken izleme amacıyla, değişkenlerin üzerine gelip onların o anki değerlerini görebilirsiniz, veya izleme(debugging) pencerelerini kullanabilirsiniz.

BP'a denk geldikten sonra izleme dışında bir eylemde bulunmak isterseniz 3 seçeneğiz var:

* Reset tuşu ile durdurmak
* F5 ile tam gaz devam etmek
* F8 ile adım adım ilerlemek(F8 ve türevleri)

**NOT**: Bu BP'lar sadece ilgili oturum için geçerlidirler, yani dosya ile birlikte kaydolmazlar.

dikkat: BP'a denk gelip incelemenizi yaptığınızda kodu durdurmayı tercih ederseniz, dikkat etmeniz gereken bir nokta var. Eğer program girişinde bazı Application propertylerine(DisplayAlerts gibi) False atadıysanız bunları tekrar True yapmadan durdurmamalısınız. Aksi halde üzücü sonuçlarla karşılaşabilirsiniz.

#### Adım adım ilerlemek

##### F8 ile adım adım ilerleme

Normalde bir makroyu çalıştırmak için Standart menü çubuğundan Play butonuna veya F5 tuşuna basarız, böylece kod baştan sona tek seferde çalışmış olur.

Ancak kodumuzun, hangi aşamalardan geçtiğini ve özellikle bir yerde hata alıyorsa tam olarak nerede ve niçin hata aldığını görmek için F8 tuşu ile kodu çalıştırırız. Böylece kodunuz satır satır çalıştırılır ve o ana kadar kodun neler yaptığını, değişkenlerin hangi değerlere sahip olduğunu fareyi değişkenin üzerine gelip bekleterek görebilirsiniz.

##### Shift+F8 ile prosedürleri atlayarak adım adım ilerleme

Normal F8'le tek farkı, eğer kodumuzda bir satırda bir başka prosedürü çağırıyorsak, bunun içine de girip satır satır ilerlemek yerine o prosedürü tamamen çalıştırıp tekrar ana prosedüre dönmemizi sağlar. Alt prosedürün hatasız çalıştığından eminsek bunu kullanırız, böylece onun içinde satır satır ilerleyerek boşuna vakit kaybetmemiş oluruz.

##### Shift+Ctrl+F8 ile adım adım modundan çıkma

Uzun bir süre bu kombinasyonun amacını tam anlayamamıştım, hatta bu sayfayı ilk yazdığımda bu kombinasyonu gereksiz bulduğumu bile belirtmiştim. Ama bir gün tam da böyle birşeye ihtiyacım oldu. Şöyle ki, bir prosedür içindeyken F8 ile ilerlerken başka bir prosedüre dallandığınızda bazen orada uzun bir döngüye girmiş olabiliyorsunuz ve bir anca önce o prosedürden çıkıp ana prosedüre dönmek ve oradan F8 ile devam etmek istiyorsunuz. İşte böyle bir durumda bu kombinasyon ile o alt prosedürü hızlıca tamamlayıp ilk prosedürde kaldığınız yere gelirsiniz.

Şöyle düşünenleriniz olabilir tabi; bunu ana prosedürden alta dallandığım yerin bir satır altına breakpoint koyarak ve F5'e basarak da yapabilirdim. Evet yapabilirsiniz ama bunu en başta yapmanız gerekirdi. Bir de bazen kodunuz o kadar karmaşık olabilir ki, ordan oraya dallanarak gelmişsinizdir, tam olarak nereye breakpoint koyacağınızı bile bilemeyebilirsiniz. O yüzden en güzeli, bu kombinasyonu çalıştırmaktır.

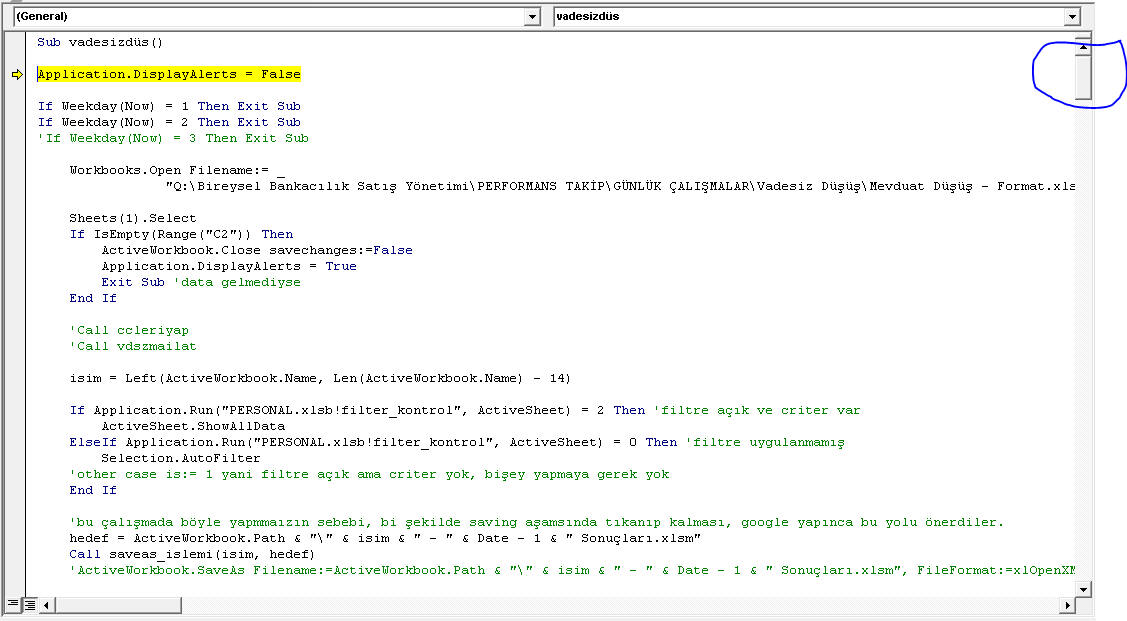
##### Ctrl+F8 ile cursor’ın bulunduğu yere kadar hızlıca ilerleme

Bence bu kombinasyona da gerek yok, onun yerine ilgili yere BP koyun ve F5 yapın, aynı görevi görür. Üstelik BP+F5'in şöyle bir avantajı da var, aynı kodu tekrar çalıştırmanız gerekebilir ve cursor farklı yerdeyse tekrar oraya gelmeye çalışacaksınız, bu da ekstra zaman kaybı demektir. BP'i bir kere koyun ve kontrolleriniz bitene kadar orada kalsın.

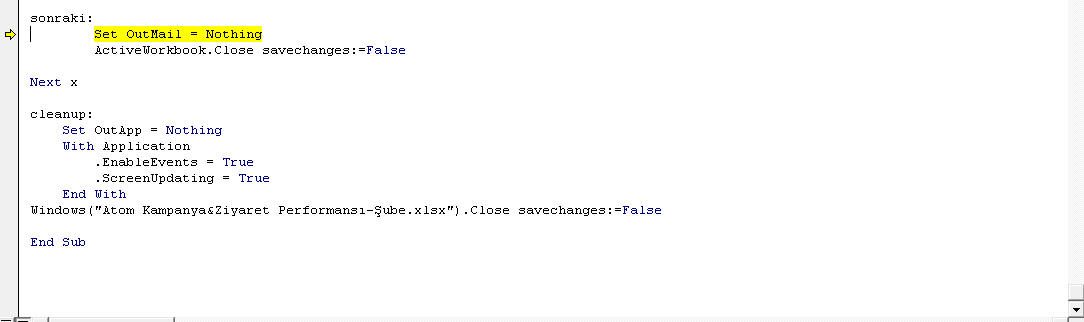
##### Ctrl+F9 ile çalıştırılacak kod kısmını belirlemek

Bu, break moddayken sarı oku mousela taşımaya benzer. Diyelim 5 sayfalık bir kodunuz var. Biliyorsunuzdur ki, kodun üst kısımlarında sorun yok, son paragrafta bir yerde sorun çıkarıyor. Kodun üst kısımlarında F8 ile ilerlediniz, sonraki birkaç sayfalık kodu çalıştırmaya gerek yok diyorsunuz. Şimdi normalde bu kombinasyon olmasaydı fareyle sarı oku 5 sayfa aşağı indirmek gerekirdi, ki bunu deneyin, çok sinir bozucu bir şeydir.

Halbuki, bu özellik sayesinde, sarı okun gelmesini istediğim yere bir yere tıklarım ve **Ctrl+F9**'a(veya debug menüsünde Set Next statement) tıklarım, böylece aradaki kodlar çalışmadan doğrudan bu satıra gelmiş olurum.



Ctrl+F9'dan sonra doğrudan istediğimiz yere geldik.



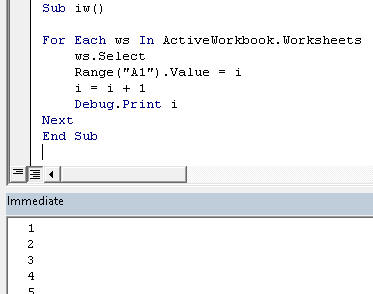
Tabi burda unutulmaması gereken birşey var; üst kısımlarda F8 ile ilerlerken, aşağıya Ctrl+F9 ile geleceğiniz noktada bir değişken varsa, üst tarafta buna değer atandığı yerleri de F8 ile geçtikten sonra buraya sıçramamız gerekir yoksa istediğiniz sonucu elde edemeyebilirsiniz.

#### Debug.Print ve Debug.Assert

##### Debug.Print

Debug.Print ile biraz aşağıda değineceğimiz **Immedaite Window'**a doğrudan birşeyler yazdırabiliyoruz. Ben genelde şöyle kullanıyorum: Debug.Print ifadesinden hemen sonraki satıra BP koyarım, Immediate Windowa ne yazdığına bakarak testimin sonucunu görürüm, veya testlerimde arka arkaya bir şeylerin değerini test edeceksem MsgBox alternatifi olarak kullanırım.

Bunu illa Break modda da kullanmak zorunda değilsiniz, normal bir F5 çalıştırmasında da kullanılabilir.



##### Debug.Assert

Bir nevi koşullu BP olarak düşünebileceğimiz Assert metodu nerede ve ne zaman kullanılacağı biraz kafa karıştırıcı bir metoddur. Kodunuzda bir hata olup olmadığını size önceden söyleyebilecek güce sahiptir.

Bunun, bir sonraki bölümde göreceğimiz, hata yakalama ifadeleriyle karıştırılmaması gerekiyor. **Hata yakalama blokları; kullanıcı, kodu çalıştırırken bir hata ile karşılaştığında ne yapılması gerektiğini söyler; Assert ise kod tasarlanırken kodu yazana nerde hata olduğunu söyler**. Yani kodunuzu çalıştırdığınızda bir yerlerde hata çıkıyor ama bir türlü nerde olduğunu bulamıyorsunuzdur. İşte böyle durumlarda Assert özelliğini baştan tanımlamanız durumunda hatayı kolaylıkla tespit edebilirsiniz.

Kullanım şekli şöyledir: **Debug.Assert koşul gerçeklenmiyorsa**

Bu şekilde kullandığımızda VBA'e şunu demiş oluyoruz: **Koşul gerçeklenmediğinde Break moda gir.**

Mesela aşağıdaki örnekte For döngüsü içinde bir yerde yeni sayfa yaratılıyor. Ama biz 5'ten fazla sayfa olsun istemiyoruz. Kodumuz bize bir şekilde 5ten fazla sayfa üretirse o anda Break moda girmiş olur ve biz de duruma müdahale ederiz.

Debug.Assert Sheets.Count <=5

For i=1 to x

'çeşitli kodlar

Sheets.Add

'çeşitli kodlar

Next i

Bir diğer pratik kullanımı da bir döngüde hızlıca belli bir yere kadar ilerlemek ve hatayı son bölümde incelemek olabilir. Mesela aşağıdaki kodu çalıştırdığınız aktif sayfa 1000 satırlık bir veri içeriyor olsun. Son satırda bir hata alıyorsunuz, F8 ile tek tek gitmek isteseniz 999 kere F8 yapmanız lazım, bunun yerine **Debug.Assert ActiveCell.Row < 1000** deyip F5 yaparak hızlıca 999. satıra geliriz.

Sub pagebreak()

[a2].Select

ActiveWindow.View = xlPageBreakPreview

ActiveSheet.VPageBreaks(1).DragOff Direction: =xlToRight, RegionIndex: =1

Do Until IsEmpty(ActiveCell.Offset(1, 0))

Debug.Assert ActiveCell.Row < 1000

ActiveCell.Offset(1, 0).Select

If ActiveCell.Value <> ActiveCell.Offset(-1, 0).Value Then

Set ActiveSheet.HPageBreaks(2).Location = ActiveCell

End If

Loop

End Sub

###### Debug.Assert False

Workbook\_Open eventi gibi dosya henüz açık olmadığı için manuel kontrolü ele alamadığınız veya Worksheet\_Change eventi gibi manuel debugging başlatamadığınız veya bir şekilde kodun bir yerine geldiğinde durmasını ve beklemeye geçmesini istediğiniz durumlarda kodun başlangıcında kontrolü ele almak isteyebilirsiniz. Bunu breakpointlerle de yapabilirsiniz ancak breakpointler dosya kapandığı zaman kaybolurlar. Bu breakpointleri tekrar tekrar oluşturmak istemiyorsanız işte bu tür durumlarda Debug.Assert'ün bu özel halini kullanabilirsiniz. Bu şekilde ilgili bu satıra gelindiği anda durur ve sizi bekler, bundan sonra F8 ile ilerleyebilirsiniz.

### İzleme Pencereleri

3 adet izleme penceresi bulunuyor.

* Immediate Window
* Locals Window
* Watch Window

Bunlardan en çok kullanılan dolayısıyla ana izleme penceresi ünvanı verebileceğimiz pencere Immediate Window'dur. Sırayla inceleyelim.

#### Immediate Window

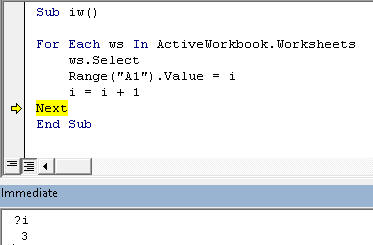
Ctrl+G tuş kombinasyonu veya View menüsünden Immediate Window butonu ile aktive edebilirsiniz.

Bununla neler yapabiliriz?

* F8 ile adım adım ilerlerken bu pencereyi açıp o an bir değerin ne olduğunu görebiliriz
* Bir kod çalıştırabiliriz
* Application seviyesinde bazı özellikleri sorgulayabiliriz.

Birkaç örnekle bakalım.

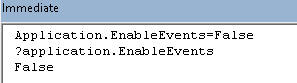
Mesela aşağıdaki kodda döngüde birkaç kez ilerledikten sonra **i**'nin hangi değere ait olduğunu sorgulamak için Immediate Window'a **"?i"** yazıp Enter'a bastım. Gördüğünüz gibi bir değişkenin değerinin ne olduğunu **sorgulamak(bilgi öğrenmek)** için başına **?** işaret koyup Enter’a basarız.



Bu sefer Application seviyesinde bir kod yazalım. Dikkat edin F8 ile giderken yapmıyorum bunu, hatta bir makro içinde bile yapmıyorum. Yine bilgi öğrenmek istediğim için başına ? koyuyorum. Öğrendiğim bilgi de XLSTART klasörünün yeridir.



Şimdi ise sorgulama yapmak yerine bir kod çalıştıralım ve bu kod yine Application seviyesinde olsun.



Gördüğünüz gibi ilk olarak EnableEvents özelliğine False değerini atadım, hemen arkasından da ? koyarak bu özelliğin durumunu öğrendim. Bu arada bazen geçici olarak bu ve bununu gibi özelliklere bir değer atamanız gerekecek, bunun için bir prosedür yaratıp içine bu kodu yazmaktansa bu pencereyi kullanabilirsiniz.

Son olarak da bir kodun içinde olalım olmayalım fark etmez, daha normal(değişken içermemesi kaydıyla) bir kod çalıştıralım.



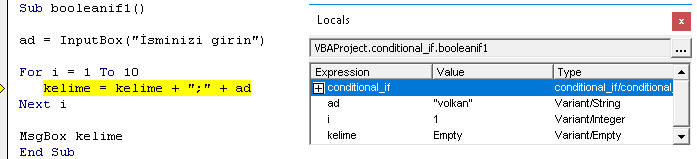
Bunu yazıp Enter'a basınca bir mesaj kutusu çıkacaktır.

#### Locals Window

**Local** penceresi, kodunuz üzerinde F8 ile tek tek ilerlerken o anda değişkenlerin değerini ayrı ayrı görebileceğiniz bir penceredir. Bu bizi aslında birsürü **Debug.Print** yazmaktan veya değişkenlerin üzerine gelip bekleyerek pop-up kutucuk ile değişkenlerin değerini tek tek görme zahmetinden kurtaran bir araçtır.

Bu pencerede local değişkenlerin değerini ve UserForm seviyesindeki tanımlanmış değişkenlerin değerini görebiliyoruz. Modül seviyesindeki değişkenler için **Watch** penceresine bakarız.

Aşağıdaki pencerede ad değişkenine volkan değeri, i değişkenine de 1 değeri atanmış durumda. Kelime değişkenine ise henüz bir atama olmadığı için Empty görünmektedir. Ayrıca fark ettiyseniz değişkenlerin tiplerini de görmekteyiz. İlk iki değişken deklare edilmediği için Variant tipindedir, ama aldıkları değer itibarıyla String ve Integer değer tutmaktadırlar.



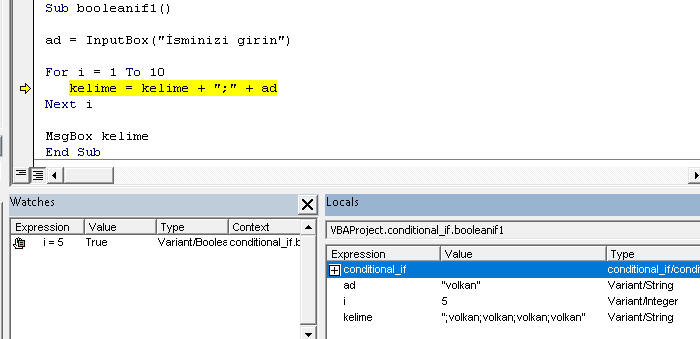
#### Watch Window

Bu pencere ile Modül seviyesindeki değişkenlerin durumunu görebiliyor, çeşitli koşullar girerek bu koşulların gerçekleşmesi veya girdiğimiz değişkenlerin değeri değiştiğinde kodun Break mod'a girmesini sağlayabiliyoruz.

3 çeşit izleme çeşidi var.

* **Watch Expression**: Local Window gibi çalışır. Modül seviyesindeki değişiklikler de izlenir.
* **Break When Value is True**: Verdiğiniz koşul sağlandığında Break moda girer(Bir nevi **Debug.Assert** görevi görür)
* **Break When Value Changes**: Verdiğiniz değişkenin değeri değişince break moda girer

Aşağıdaki örnekte i=5 olunca break mod'a girsin demiş olduk. Aynı anda hem Local hem Watch windowu görüyoruz.



#### Call Stack

Bu pencereyi hiç kullanmadım açıkçası. Görevi, o anda aktif olarak çağrı yapılmış tüm prosedürleri göstermek. İhtiyacınız olmaz düşüncesiyle detaya girmiyorum. Arzu eden detayları araştırabilir.

## Hata Yakalama (Error Handling)

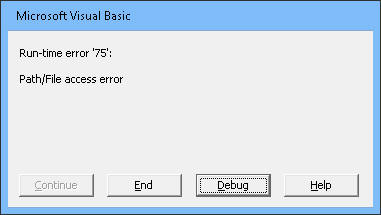
Derler ki Wright kardeşler uçağı uçurmadan önce düşmeyi öğrenmişler. "Nasıl iyi bir şekilde düşelim ki ölmeyelim" diye düşünmüşler. Uçağın icadına giden yol böyleymiş.

Biz de makro yazarken çok sık hata yapacağız, ama bu hataları öyle zarif bir şekilde ele almalıyız ki, bir terslik durumunda yumuşak bir iniş olsun.

İşte bu bölümde program akışı içinde oluşan hataları nasıl ele alacağımızı ve yumuşak iniş yapmayı öğreneceğiz. Zira iyi bir makro içinde mutlaka olası hataları ele alan bir bölüm olmalıdır.

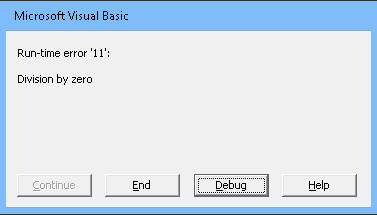
### Giriş

Kodumuzda, oluşabilecek hataları ele aldığımız satırlara Error Handler(Hata Yakalama) satırları denmektedir. Eğer kodumuzda hata yakalama satırı yoksa VBA'in aşağıdaki klasik hata mesajını görürüz. Bu da özellikle programınız kullanacak başka kişiler varsa çok şık olmaz.

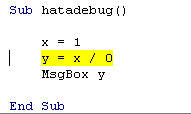


**Amacımız**, kodumuzun uygun yerlerine hata kontrol noktaları koymak, bunların bazısında program akışını uygun yere yönlendirmek, bazısında hatayı düzeltip tekrar başa(veya başka bir yere) döndürmek veya anlaşılır bir mesaj vererek programı sonlandırmak olmalıdır.

**Bununla birlikte**, geliştirme sırasında hiç bir hata yakalama kullanmamak uygun olabilir, hatta böylesi daha iyidir, böylece çıkan hata mesajlarında "Debug" düğmesine tıklayarak nerede hata aldığımızı görebiliriz. Ayrıca önceki bölümde gördüğümüz gibi **Debug.Assert** ile de tasarım sırasında da test etmiş oluruz.



Debug'a basınca hatanın olduğu yer sarılı gösteriliyor.



**NOT**: Kodunuz korumalıysa Debug düğmesi pasif olur.

#### Hata çeşitleri

Farklı kaynaklarda farklı sayılar verilmekle birlikte ben hataları genel olarak 2 kategoriye ayırıyorum:

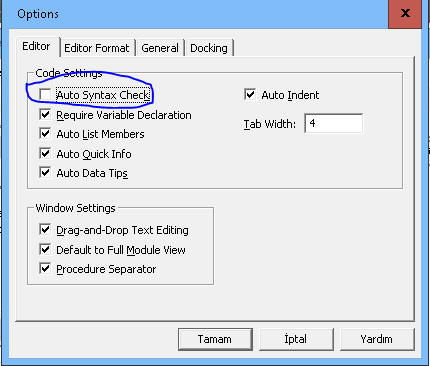
**Derleme hataları(Compile error)**: Programın derlenmesini engelleyen hatalardır. **Örnekler**:

* Option Explict açıkken tanımlanmamış değişken kullanımı
* If bloğunda Then yazmadan bir alt satıra geçmek
* Next yazılmamış bir For döngüsü
* Tüm yazım hataları(syntax error): Range yerine Rage yazmak
* Olmayan bir prosedürü çağırmak
* Bir prosedüre modülle aynı ismi vermek
* v.s

**ÇalışmaZamanı hataları(Runtime error):** Kodunuz sorunsuz derlenmiştir, ancak çalışma sırasında bir yerde hata verebilir. Örnekler:

* 0'a bölme,
* Bir değişkene taşıyabileceği değerden daha fazla değer atanması(integer'a 50.000 atanması gibi)
* Olmayan bir sayfaya/dosyaya erişmek
* v.s

**NOT**: Yazımsal kaynaklı(syntax) compile hataları sonucunda çıkan dialog kutuları bazen can sıkıcı olabilmektedir. Bunların çıkmasını engelleyebilirsiniz, tabi yine de hatalı olan kod kırmızı olarak gösterilecektir. Böylece hatanızı hala anında görmeye devam edersiniz. Bununla sadece dialog kutusunda kurtulmuş oluyoruz. Baktınız ki bir türlü hatanın nedenini çözemiyorusunuz, bu işlemi geriye alıp tekrar mesaj çıkmasını sağlayabilirsiniz.



Derleme hatası olup olmadığını **Debug>Compile** menüsünden test edebilirsiniz. Varsa tekrar yapın, taa ki bu düğme gri, yani pasif olana kadar.

Runtime hatalarını **On Error** ile başlayan cümleciklerle ele alacağız. Bunlara geçmeden önce bir de beklenti türüne göre hataları inceleyelim.

##### Beklentiye göre Hata çeşitleri

Hatalar beklentiye göre iki gruba ayrılır.

* Beklenen hatalar
* Beklenmeye hatalar

Program akışı sırasında bir yerde **hata olma olasılığı varsa**, yani hata beklenen bir hataysa bunları koşullu yapılarla yakalayabiliriz.

Örneğin bir metin dosyasından okuma yapmadan önce(veya bir Excel dosyasını açmadan önce) onun var olup olmadığını öğrenmeye çalışmak akıllıca olur. Dosya varsa okuma yaparız, yoksa yapmayız ve kullanıcıya bir mesaj çıkarırız. Böyle **spesifik** bir beklentiye karşı spesifik bir mesaj vermiş oluruz.

Sub OpenFile()

Dim dsy As String

dsy = "C: \deneme.txt"

' DosyaVarmı diye bir fonksiyon yazmış olalım, bu fonksiyon ile onun varlığını sorguluyoru

If Not DosyaVarmı(sFile) Then 'DosyaVarmı(dsy)=False da yazabilrdik

MsgBox "Dosya mevcut değil, dosya adının doğru olduğundn emin olup tekrar deneyin"

Exit Sub

Else

'yapılacak işler

End If

End Sub

Keza yine iki sayının toplamını aldıracağız diyelim. Burada kullanıcı sayılardan birini veya ikisini de nümerik olmayan bir değer girebilir. Bu beklenen bir durum olup bunu da If blokları ile yakalarız.

Aşağıda göreceğimiz hata kodlarını ise genelde beklenmeyen hata durumlarında kullanırız. Yani programa, **genel** bir hata durumunda ne yapacağını söylemiş oluruz.

### On Error ifadesi

On Error kalıbı hata yakalamanın kalbidir ve bu kalıbı takip eden bir dizi ifade vardır. Bunlar, "hata olması durumunda şunu yap" şeklinde özetlenebilecek ifadelerdir.

* On Error Resume Next
* On Error GoTo etiket
* On Error GoTo 0
* On Error GoTo -1

Şimdi bunlara tek tek bakalım

#### On Error Resume Next

Bu kalıp, VBA'e hatayı görmezden gelmesini ve bir sonraki satıra geçmesini söyler. Yani hatayı düzeltmek yerine yola devam eder. O yüzden çok dikkat edilmesi gereken bir kalıptır. Birçok durumda bunu kullanmaktan sakınmanız gerekir, bütün üstatlar da bunu söyler zaten, ancak belli bazı durumlar vardır ki bu kalıbı kullanmak akıllıca olmaktadır.

Mesela bankada çalışıyoruz diyelim ve bir otomatik mail makromuz olsun. Bankadaki belli bir roldeki tüm personele, kişiye özel olacak şekilde mail gönderiyor olalım. Ancak elimizdeki kişi listesi güncel olmayabilir, yani bu kişilerden bazıları artık bankadan ayrılmış olabilir. Böyle bir durumda programa şunu söyleyebiliriz: "Sırayla herkese mail at, maili bulunmayanlar için durma, bir sonrakine devam et". Bunu demezsek, program, ilk ayrılmış kişide durur. Bunu ele almanın başka yöntemleri de var tabi; Resolve metodunu kullanmak, On Error GoTo etiket deyip, ilgili kaydı formatlamak ve bir sonrakine devam etmek gibi ama konuyu anlatmak adına basit düşünüyoruz.

İkinci bir örnek, her gün belirli bir saate schedule edilmiş ve o günün sonucunu tarih adıyla kaydeden bir rutininiz olsun. Bir süre sonra diskinizde şişkinlik olmasın diye her gün t-30 tarihli raporları silmeye yarayan da başka bir rutininiz olsun. Yani bize sadece son 30 günün raporu yeter diyoruz. Ama bazı günler raporlarınız oluşmamış olabilir, o yüzden var olmayan bir t-30 raporunu silmeye çalışabilirsiniz. Böyle günlük schedule edilmiş 40 raporunuzun ikisi için rapor oluşmadığını düşünelim. Diğer 38ini tehlikeye atmamak için yine bu kalıbı kullanabiliriz. Tabi bunu bir alternatifi de dosya mevcut mu diye kontrol etmek de olabilir ama bu kontrol de kodun süresini uzatacaktır. Böyle bir durumda bu kalıbı kullanmak daha pratik bir çözüm olacaktır.

Aşağıda ilk örneğimize ait kod bloğunu görüyoruz.

Sub onerrorresumeornek()

'Dim ile değişken deklerasyonları

'1000 diye sabit bir rakam olmaz tabi, bunu son satır gibi bir yöntemle yakalamanız lazım \_

örnek basit olsun diye 1000 yazdım

For i=1 to 1000

On Error Resume Next 'sadece döngü içinde hata olursa bir sonraki satıra geçsin

'mail gönderim kodları

Next i

'varsa diğer kodlar

End Sub

#### On Error GoTo Etiket

Bu kalıp, tüm hata türlerini yakalar ve programı etiketin olduğu yere yönlendirir. Bu kısımda genelde şunlar yapılır:

* Hata mesajları veririz
* Olası hatayı düzeltebiliyorsak düzeltir ve hata aldığımız yere veya hemen sonraki satıra yönlendiririz
* Programın başında False atadığımız bazı özellikleri burada tekrar True'ya döndürürüz
* Sayfa veya Workbook seviyesinde protection'ı geçici kaldırdıysak tekrar protection uygularız
* Log tutarız

Etiketin hemen öncesinde bir **Exit Sub** ifadesinin olmasında fayda vardır, yoksa hatasız ilerleyen kod son olarak buraya gelir ve gereksiz yere burdaki kodları da çalıştırır, ki bunu istemeyiz.

Hata mesajında genelde **Err.Description** ve **Err.Number** özelliklerini kullanırız. Err.Number için bir koşullu yapı kullanılarak, "Hata numarası şuysa şu mesajı, buysa bu mesajı, diğer durumlar için Err.Description & "Volkanla görüşün" " gibi bir mesaj verdirebiliriz.

Sub Hataornek()

On Error Goto hata

With Application

.DisplayAlerts=False

.ScreenUpdating=False

End With

'çeşitli kodlar

'program bitişi öncesinde bu ayarları eski haline getriyoruz

With Application

.DisplayAlerts=True

.ScreenUpdating=True

End With

'hata etiketinden hemen önce programdan çıkıyoruz

Exit Sub

hata:

'hata durumunda bu ayarları eski haline getriyoruz

With Application

.DisplayAlerts=True

.ScreenUpdating=True

End With

MsgBox Err.Description & vbcrlf & "Bu bilgi yeterli değilse Vlkanla görüşün lütfen"

End Sub

Dikkat: Bu kalıbı kullanırken akılda bulundurulması gereken önemli bir husus var, o da bir anda sadece tek bir hata yakalama bloğunun aktif olabileceğidir. Yani bir hata için **On Error GoTo hata1** dediniz diyelim, hata1 bloğunda da bir hata meydana gelirse oraya da **On Error GoTo hata2** yönlendirmesi yaparsanız bu etkisiz kalacaktır, çünkü ilk hata yakalayıcı hala aktiftir. Bu durumu aşmanın bir yolu var ve biraz sonra bunu göreceğiz. Siz şimdi sadece, bunun gerçekten işe yaramadığını görmek için şu kodu çalıştırmayı deneyin.

Bu kodda, önce 0'a bölme hatası yapıyoruz. Bu bizi hata1 etiketine yönlendiriyor. Hata1 Error Handler’ında da, bu hatayı erişimimiz olmayan bir dosyaya yazmaya ve kaydını tutmaya çalışıyoruz. Yine hata alınca bu sefer hata mesajı görüntüleniyor.

Sub cokluhata()

On Error GoTo hata1

x = 0

MsgBox 1 / x

MsgBox "İşlem sonucu başarıyla gösterildi"

Exit Sub

hata1:

On Error GoTo hata2

dosya = "C: \deneme.txt" 'bu dosyayı yönetici izni sağlayarak oluşturdum, yani dosya var,

'ancak izin isteyen bir konum olduğu içn dosyaya yazma sırasında hata alacağım

Open dosya For Append As #1

Write #1, Now, Err.Number, Err.Description, Environ("username")

Close #1

Exit Sub

hata2:

MsgBox Err.Description, , "Hata"

End Sub

#### On Error GoTo 0

Hiçbir hata yakalama kodu yazmazsak, VBA ilk hatada bize hata mesajı gösterir demiştik. İşte **On Error GoTo 0** da, VBA'in default modu olan "Hata yakalama bloğu yok, tüm hatalar için uyarı göster"e dönüş yapmamızı sağlar. Böyle bir şeyi neden isteyelim ki? Birazdan göreceğiz. Ama şunu anlamışsınızdır ki, bu kalıbı direkt programın başında kullanmak anlamsızdır, zaten default ayar öyledir.

Bu kalıbı daha çok, var olan bir hata yakalama rutini iptal etmek için kullanırız. Neden böyle bir şeye ihtiyaç duyacağımızı bir düşünün! Sonra devam edin.

Mesela yukarıda bahsettiğimiz otomatik mail gönderim makrosunu düşünün. Yaklaşık 1000 kişiye mail gidecek, ama gönderim yapacağınız kişilerden bazılarına mail gönderilemiyorsa bir sonrakine geçsin demiştik. Bunun için de **On Error Resume Next** demiştik. Ancak mail gönderememe dışında başka bir sorun varsa bunun size veya kullanıcıya gösterilmesini istemiş olabilirsiniz. Ne tür hatalar çıktığını gördükten sonra bunlara özgü çözümlerinizi de ayarlayabilesiniz diye On Error GoTo kalıbını kullanabilirsiniz. Sonrasında ya If blokları ile yakalarsınız veya bir etikete göndererek işlem yaparsınız. Örnek bir kod bloğu şöyle olacaktır:

Sub onerrorgotosfır()

'Dim ile değişken deklerasyonları

'1000 diye sabit bir rakam olmaz tabi, bunu son satır gibi bir yöntemle yakalamanız lazım \_

örnek basit olsun diye 1000 yazdım

For i=1 to 1000

On Error Resume Next 'sadece döngü içinde hata olursa bir sonraki satıra geçsin

'mail gönderim kodları

ooMail.Send

'On Error GoTo 0 buraya da konabilir, aşağıya da

Next i

'buradan sonra bir hata çıkarsa hata mesajını görelim

On Error GoTo 0

'diğer kodlar

End Sub

#### On Error GoTo -1

Yukarda bir programda **sadece** **bir** adet hata yakalama bölümü aktif olabilir demiştik, dolayısıyla bir Error Handler bloğunda başka bir Error Handler’a yönlendirme yapılırsa bu işe yaramıyordu. İşte **On Error GoTo -1** ile bu sorunu aşmış oluyoruz. Bu kalıp, var olan Error Handler'ı resetler(**Err** nesnesini yok eder) ve sizin yeni bir Error Handler yaratmanıza imkan verir. (Bunun, aşağıda anlatacağımız **Err.Clear** ile karıştırılmaması lazım. **Err.Clear** sadece hata bilgilerini yok eder, **Err** nesnesini değil). **Sözün özü**; bu kalıp, başka bir Error Handler bloğu içinde kullanılır ve var olan Error Handler içinden başka bir Error Handler’a yönlendirmenin tek yolu budur. On Error Resume Next bile burada işe yaramaz.

Şimdi yukarda örneği tekrar alalım ve hata1'den hemen sonra kalıbımızı yerleştirelim.

Sub cokluhata()

On Error GoTo hata1

x = 0

MsgBox 1 / x

MsgBox "İşlem sonucu başarıyla gösterildi"

Exit Sub

hata1:

On Error GoTo -1 'bu satırın başına yorum işareti koyarak etkisini görün

On Error GoTo hata2

dosya = "C: \deneme.txt" 'bu dosyayı yönetici izni sağlayarak oluşturdum, yani dosya var,

'ancak izin isteyen bir konum olduğu içn dosyaya yazma sırasında hata alacağım

Open dosya For Append As #1

Write #1, Now, Err.Number, Err.Description, Environ("username")

Close #1

Exit Sub

hata2:

MsgBox Err.Description, , "Hata"

End Sub

#### İkisi bir arada örnek

Inputbox konusunu anlatırken görmüştük ki, dönüş tipi Range olan bir Inputbox sorusuna Cancel deyip çıkarsak program hata alıyordu. Bunun için "On Error Resume Next" diyerek hata mesajını yok ediyorduk. Peki ya sonraki hatalarda devam etmesini istemiyorsak, o zaman yeni bir hata yakalama bloğu set etmeliyiz. Ör:

Sub hataikili()

Dim a As Range

On Error Resume Next 'geçici olarak tanımlanır

Set a = Application.InputBox("Bir hücre seçin", Type: =8) 'buna esc dersek diye, commenteleyip görelim

On Error GoTo hata 'esas hata yakalama kodumuzu yazıyoruz

If Not a Is Nothing Then

MsgBox "Seçim yapıldı"

'Diğer kodlar buraya

Else

MsgBox "Bir seçim yapılmadan çıkmayı tercih ettiniz"

End If

MsgBox 1 / 0

MsgBox 2 / 4 'buraya ulaşmaz

Exit Sub

hata:

MsgBox Err.Description

End Sub

#### Kapsamlı bir örnek

Şimdiki örnekte 4 çeşit **On Error** ifadesini de kullanıyoruz. Bu açıdan bu örneğin çok faydalı olacağını düşünüyorum. Özeti şu: Networkte duran ve her gece refresh olması beklenen bir dosya schedule edilmiş durumda. Schedule saati gelmiş ve dosya açılınca onun içindeki ThisWorkbook\_Open makrosu devreye girecek ve dosyadaki tablolar refresh olacak. Refresh sonucunda data dönüyorsa(yani veritabanına ilgili data yüklenmişse) mailing işlemine devam edecek, henüz data yüklenmediyse dosyayı kaydetmeden çıkacak. Aradaki detaylar kafanızı karıştırmasın, sadece hata bloklarına odaklanın.

NOT:Kod çok büyük olduğu için **github**’dan(**musteri\_degisim**) indirebilirsiniz.

#### Error Handler’dan çıkma yolları

GoTo -1 bölümünde yaptığım açıklamalar şöyle bir yanlış anlaşılmaya sebep olmamalı. GoTo -1, **bir Error Handler içinden başka bir Error Handler içine göndermenin** tek yoludur, **Error Handler içinden çıkmanın** tek yolu değil. Error Handler içinden On Error GoTo -1 kalıbına ilave olarak **Resume, Resume Next, Resume etiket** veya **Exit Sub** diyerek de çıkabilirsiniz. Bu şekillerde çıkış yapıldığında da Error Nesnesi resetlemiş olur.

Şimdi bunlara bakalım:

##### Resume

Resume tek başına kullanıldığında hata olan yerden tekrar devam etmeye çalışır. Bunun için tabi Error Handler’da hatayı düzeltmeniz gerekir. Duruma göre kullanıcıya düzeltmeyle ilgili bilgi vermek de gerekebilir.

Dikkat: Hatayı düzeltmeden Resume dersek program sonsuz bir kısırdöngüye girer.

Sub resume1()

On Error GoTo hata

x = InputBox("0 ile 100 arasında bir sayı girin") '0 dahil mi belli dğeil

sayı = 1 / x

Debug.Print sayı

Exit Sub

hata:

If x=0 Then

x=1

MsgBox "Girdiğiniz sayı aralık dışı olup en düşük 1 girmeniz gerekirdi. Sayı otomatikman 1e yuvarlandı""

Resume

Else

MsgBox Err.Description 'mesela bir harf girilirse hata versin

End If

End Sub

Başka bir örnek de şu olabilir. Biz biliriz ki nerdeyse tüm Workbook’larda Sheet1 diye bir dosya vardır, ve kodumuz içinde kullanıcıdan bir dosya seçmesini istedik diyelim, sonra gidip Sheet1 isimli sayfaya bazı şeyleri otomatik yazacağız. Eğer Sheet1 diye bir sayfa yoksa hata verecektir. Bunu aşağıdaki gibi ele alabiliriz.

Sub resume3()

On Error GoTo hata:

'Kullanıcıdan dosya seçtiren kodlar

Worksheets("Sheet1").Activate

'diğer işlemler

hata:

If Err.Number = 9 Then 'sayfa yoksa

Worksheets.Add.Name = "Sheet1"

Resume 'Worksheets("Sheet1").Activate satırına geri döner

End If

End Sub

Aşağıdaki örnekte de Error Handler içinde hatalı kaydı düzeltip aynı kayıtla(sonraki değil) devam etmesini sağlıyorum:

Sub resume4()

On Error GoTo hata

Dim kareal As Variant

kareal = Array(9, 16, -64, 100) 'yanlışlıkla 64ün önüne - yazmışım diyelim

For Each a In kareal

MsgBox a & " sayısının kökü: " & Sqr(a)

Next a

Exit Sub

hata:

a = Abs(a)

Resume

End Sub

##### Resume etiket

Hata bloğunda gerekli bir mesaj verdikten sonra bir etikete yönlendirme işlemini **Resume Etiket** kalıbı ile yaparız. Burada alternatif olarak **GoTo Etiket** kalıbı kullanıldığı görülmekle birlikte bu kullanım beklenmeyen hatalara neden olabilir. Bu yüzden Error Handlerlar içindeyken GoTo yerine Resume kullanmanızı öneriyorum.

Sub resume2()

On Error GoTo hata

yeniden:

x = InputBox("0 ile 100 arasında bir sayı girin") '0 dahil mi belli dğeil

sayı = 1 / x

Debug.Print sayı

Exit Sub

hata:

If x=0 Then

x=1

MsgBox "Girdiğiniz sayı aralık dışı olup en düşük 1 girmeniz gerekirdi. Sayı otomatikman 1e yuvarlandı""

Resume 'burada da istersek kullanıcıya tekrar seçim hakkı vermek isteyebiliriz

Else

If Err.Number=13 Then 'Type Mismatch hatasıysa

MsgBox "Lütfen geçerli bir sayı giriniz"

Resume yeniden 'GoTo etiket de alternatif olmakla birlikte kullanmayın Else

Else 'başka bir hataysa

MsgBox Err.Description

End If

End If

End Sub

##### Resume Next

Mesela otomatik mail programınızda sicili olmayan kişiler çıkarsa program akışı durmasın diye **On Error Resume Next** demiştik ya, bunun bir diğer ve şık alternatifi de şu olabilir. **On Error Goto hata** dersiniz, hata bölümünde bir log kaydı oluşturabilir ve/veya mail gitmeyen kişilerin listesi o an açık olan bir Excel dosyasında ise onları renklendirebilirsiniz, böylece kimlere mail gitmediğini de görmüş olursunuz. Zira belki sorun To'ya koyduğunuz kişide değil, cc'deki kişiden kaynaklı da olabilir.

Tabi bu örnekte başka bir hata durumunda istenmeyen sonuçlar elde edebilirsiniz, o yüzden kodunuzu bu duruma göre ayarlamanız gerekecektir. Burada basitlik olması adına sadece olmayan bir adrese mail gitme ihtimali olduğunu düşündük.

Sub resumenext1()

'Dim ile değişken deklerasyonları

On Error GoTo hata

'1000 diye sabit bir rakam olmaz tabi, bunu son satır gibi bir yöntemle yakalamanız lazım \_

örnek basit olsun diye 1000 yazdım

For i=1 to 1000

'mail gönderim kodları

Next i

'varsa diğer kodlar

hata:

'Loglama veya formatlama işlemi

Resume Next

End Sub

Aşağıdaki örnekte ise Error Handler’da hatayı yakaladıktan sonra durmasını istemiyorum ve diğer sayılar için devam etmesini istiyorum, ama devam etmeden önce de bir mesaj vermesini istiyorum.

Sub resumenext2()

On Error GoTo hata

Dim kareal As Variant

kareal = Array(9, 16, -1, "A", 81, "B", 100)

For Each a In kareal

MsgBox a & " sayısının kökü: " & Sqr(a)

Next a

Exit Sub

hata:

MsgBox a & ": Karekökü alınacak bir sayı değil"

Resume Next

End Sub

#### Err nesnesi

Bir runtime hatası oluştuğunda Err nesnesi oluşturulur ve bu nesnenin özellikleri bu hatanın detayıyla doldurulur.

Bu nesnenin üyeleri, **Exit**'li ifadelerden(Exit Sub gibi) ve **Resume Next**'ten sonra resetlenir, yani 0 ve ""'a döner. Veya manuel resetlemek için Err.Clear metodu kullanılır.

Err'in default property'si(özelliği) Number’dır. Bir de Description vardır. Bunları yukardaki örneklerde bolca kullandık. Genelde bu descriptiondaki bilgi son kullanıcıya yeterli bilgi vermez, hele bir de İngilizceleri yoksa birşey anlamazlar, o yüzden hata numarasına göre kendi mesajlarınızı belirtebilirsiniz.

Select Case Err.Number

Case Is = 5

MsgBox "Negatif sayıların karekökü alınamaz"

Case Is = 13

MsgBox "Metinsel ifadelerin karekökü alınamaz"

End Select

Çıkan hatanın Number özelliği yoksa Description özelliğinde "Application-defined or object-defined error" mesajı gösterilir. Herhangi bir anda hata yoksa Number=0 ve Description="" değerindedir. O yüzden belirli bir anda hata olup olmadığını If Err.Number=0 ile kontrol edebilirsiniz.

##### Err.Raise

Err nesnesi oluşturmanın bir yolu da **Raise** metodunu kullanmaktır. Bu metodla birlikte kendimize özgü Hata Kaynağı, Hata Numarası ve Hata açıklaması üretiriz. Hata numarasını 513 ile 65535 arası bir değer atayabiliyoruz ve bunu vbObjectError ile birlikte kullanıyoruz.

Err.Raise'in bir alternatifi **GoTo hata** diyerek ilerlemek olabilir ancak bu şekilde hatayı kayıt altına alamayız. Özellikle birden fazla yönlendirme olacaksa GoTo etiket yerine Err.Raise deyip açıklayıcı bilgilerle de donatmak gerekir.

Sub errraise()

On Error GoTo hata

x = InputBox("1000den küçük bir sayı girin") 'çok net değiliz, belki kullanıcı 0 girecek, belki 1000 dahil sanacak

'bunu direkt select case içinde kullanamadığımız için if blok içinde kullanıyorum

If Not IsNumeric(x) Then

Err.Raise 517 + vbObjectError, "Sayısal olmayan veri", "Kullanıcı sayısal olmayan bir değer girdi"

End If

Select Case x

Case 0

Err.Raise 513 + vbObjectError, "x nedeniyle 0'a bölme hatası", "Kullanıcı x=0 girdi"

Case 1000

Err.Raise 514 + vbObjectError, "x nedeniyle 0'a bölme hatası", "Kullanıcı x=1000 girdi"

Case Is > 1000

Err.Raise 515 + vbObjectError, "aralık dışı değer", "Kullanıcı x>0 girdi"

Case Is < 0

Err.Raise 516 + vbObjectError, "aralık dışı değer", "Kullanıcı x<0 girdi"

Case Else

sayı = (1 / x) \* (1 / (1000 - x))

End Select

Debug.Print sayı

Exit Sub

hata:

Debug.Print Err.Source, Err.Number, Err.Description

End Sub

##### Err.Source

Bu özellik, özellikle Log tutulurken faydalı olabilmektedir ve yukarıdaki örnekte görüldüğü gibi, Raise ile manuel yaratılan hata mesajlarında bizim tarafımızdan belirtildiğinde anlamlıdır, yoksa her zaman VBAProject gibi birşey ifade etmeyen bir sonuç üretir. (E neden var o zaman? Çünkü bu özellik VB dilinin kendisinde var ve orada anlamlı ama VBA içinde anlamsızlaşıyor, eğer ki biz kendimiz belirtmezsek!)

##### Err.Clear

Bu metod yukarda da gördüğümüz gibi hata nesnesinin üyelerini sıfırlar. Pratik olarak buna ihtiyaç duyabileceğimiz tek yer On Error Resume Next dendiği durumlarda, karşılaşılan hata adedini saymak için kullanmak olacaktır. Zira ben başka bir kullanımını görmedim.

Şöyle ki, döngüsel bir kodumuz var diyelim, döngünün bir yerinde Err.Number<>0 kontrolüne takılır ise hata oluşmuş demektir, bu durumda hatasayısı=hatasayısı+1 diyerek sayacı arttırırız. Ama arada birçok yerde hatasız geçiş olacağı için ama en sonki kod numarası da hala hafızada olacağı için bir kez hata oluştuğunda döngünün sonraki turlarında her yerde tekrar hata kontrolüne takılır. O yüzden sayacı arttırdıktan sonra hata numarasını sıfırlamak gerekir.

Aşağıdaki örnekte, yine otomatik mailing yapacağız ve kaç adet mailin gitmediğini MsgBox ile bildireceğiz. Örneği özellikle basit tuttum, sadece bu kısma odaklanmanız için. Tabiki yukardaki örneklerle birleştirilerek mail gitmeyen satırlar için renklendirme de yapılabilir.

On Error Resume Next

son=Range("A1").End(xlDown).Row

For i=1 to son

'mail kodları

If Err.Number<>0 The

hata=hata+1

Err.Clear

End If

Next i

MsgBox hata & " adet mail gönderilememiştir"

### Prosedürler arası hata yakalama

A1 ve A2 isminde iki prosedürümüz oluğunu düşünelim. A1'in içinde bir yerde A2'yi çağırıyoruz. A2'de bir hata meydana geldiğinde, VBA A2'nin içinde Error Handler var mı diye bakar. Varsa buna girer, yoksa geriye gidip A1'e bakar, A1'de varsa A1'in Error Handler'ına gider.

Aşağıdaki örnekleri F8 ile çalıştırıp görelim. İlk olarak A2'de Error Handler olmadığında A1'deki Error Handlera geldiğini görelim.

Sub A1()

On Error GoTo hata

MsgBox "selam"

Call A2

Exit Sub

hata:

MsgBox "A1'deki hata mesajı"

End Sub

Sub A2()

MsgBox 1 / 0

End Sub

Şimdi de A2'de varsa A2'de kaldığını.

Sub A1()

On Error GoTo hata

MsgBox "selam"

Call A2

Exit Sub

hata:

MsgBox "A1'deki hata mesajı"

End Sub

Sub A2()

On Error GoTo eh

MsgBox 1 / 0

Exit Sub

eh:

MsgBox "A2'deki hata mesajı"

End Sub

Tabi bu durum **sadece On Error Goto** ifadeleri için geçerlidir, Resume için geçerli değildir.

Aşağıdaki örnekte sadece A1'de Resume satırı var, dolayısıyla A2'de ilk hatayı alır almaz A1'e dönecek ve "devam" mesajını çıkaracak.

Sub A1()

On Error Resume Next

MsgBox "selam"

Call A2

MsgBox "devam"

End Sub

Sub A2()

MsgBox 1 / 0

MsgBox "selam2"

End Sub

Halbuki biz A2'de kaldığı yerden devam etmesini istersek A2 içine de ayrıca **On Error Resume Next** yazmalıyız.

Sub A1()

On Error Resume Next

MsgBox "selam"

Call A2

MsgBox "devam"

End Sub

Sub A2()

On Error Resume Next

MsgBox 1 / 0

MsgBox "selam2"

End Sub

### Hata kaydı tutma

Yukardaki bazı örneklerde Logger isminde bir prosedür görmüştük. Bununla ortaya çıkan hataları detay bilgilerle kayıt altına alabiliriz. Hata takibi açısından oldukça faydalı bir yöntemdir. Özellikle ortak kullanılan ve ara sıra hata veren bir dosyanız varsa şiddetle tavsiye ederim. Böylece kim hangi aşamada ne yapmış da hata meydana gelmiş bunu görebilirsiniz.(Dosya refreshi sırasında mı hata oluştu, bir dosyaya ulaşmaya çalışırken mi ha oluştu v.s)

Tabi bunu sadece hata kaydı tutma amacıyla kullanırsanız onun hakkını vermemiş olursunuz. O yüzden Logger prosedürünü programın kilit noktalarına yerleştirerek tam bir **olay kaydı** tutabilirsiniz. Bu konunun detay anlatımı başka bir sayfada olduğu için burada tekrar örneklendiremeye gerek görmüyorum**.** [**Dosya İşlemleri>Dosya Okuma Yazma**](file:///E:\OneDrive\Uygulama%20Geliştirme\web%20sitelerim\Publishlerim\EE\precompile\Konular\VBAMakro\Dosyaislemleri_LogcuUygulamasi.aspx) konusuna bakıp incelemenizi tavsiye ediyorum.

## Neler Öğrendik

Kodumuzu adım adım çalıştırmayı, belirli kısımları pas geçmeyi ve tekrar devam ettirmeyi öğrendik.

Farklı hata türlerinde nasıl davranmamız gerektiğini gördük.

Hataların kaydını(log) da tutan profesyonel hata yakalama süreçlerinin nasıl kurgulandığını öğrendik.