# 2.bölüm

**2**

# TEMELLER

Bu bölümde, bir temel oluşturması adına Visual Basic terminolojisiyle tanışacağız. Bu bölüm çok önemli, zira sonraki sayfalarda görüp okuyacağınız birçok kavramın havada kalmaması gerekiyor. Akabinde veri tiplerinin neler olduğuna ve operatörlere bakacağız. Programla kullanıcı arasındaki interaktivitenin nasıl sağlandığını göreceğiz. Son olarak da Personal.xlsb dışında alternatif kod çalıştırma alanlarına bakacağız..

## Terminoloji

### Giriş

İlk kodlarımıza geçmeden önce kısa bir terminoloji bilgisi edinmemizde fayda var. Böylece ne yaptığımızı daha iyi anlıyor olabileceğiz.

Visual Basic veya herhangi bir programlama dili kullanmış olanlar bilir, bir program yazarken programımızı belirli anahtar kelimeler üzerine kurarız. Bunlar temel olarak şu şekilde sınıflandırılabilir:

* Prosedürler
* Değişkenler
* Sabitler
* Diziler
* Koşullu yapılar
* Döngüler
* Hata ayıklayıcılar
* Kullanıcıyla iletişim
* Olaylar
* Nesneler
* Metodlar
* Özellikler
* Fonksiyonlar

Şimdi ilk etapta bilmemiz gerekenlere bir bakalım.

### Prosedür ve Modüller

#### Prosedür

Gerek kendimizin yazdığı kodlar olsun gerek makro kaydederek oluşturduğumuz kodlar olsun hepsi bir prosedürdür. İki tür prosedürümüz var.

**Sub prosedürler:** Çalışır ve bir şeyler yapar, ama bir değer döndürmezler. (Türkçede yordam olarak geçer)

Sub prosedüradı()

'Kodlar buraya yazılır

End Sub

**Function Prosedürler:** Çalışması sonucunda (genelde) değer döndüren prosedürlerdir. Bunlar da kendi içinde ikiye ayrılır. İlk grubun Excel’in yerleşik fonksiyonlarından hiçbir farkı yoktur, bunlara Kullanıcı tanımlı fonksiyonlar(UDF) denir. Genelde bir parametre/argüman alırlar. İkinci grupta ise VBA içinde kullandığımız ve döndürdüğü değeri yine VBA içinde kullanmaya devam ettiğimiz VBA Functionlar yer alır.

Function functionadı()

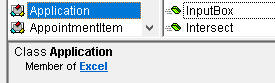
'Kodlar buraya yazılır

End Function

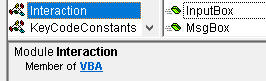
Makro konusunu işlerken ağırlıklı olarak Sub prosedürleri işliyor olacağız, yer yer Functionlara da değineceğiz.

**Fonksiyona** benzeyen bir de **metod** kavramı var, ki ikisi genelde birbiri yerine kullanılmaktadır, bununla birlikte terminolojide aralarında küçük bir fark bulunur: Fonksiyonlar bağımsız çalışabilir, ancak metodlar mutlaka bir nesneye ihtiyaç duyarlar, bağımsız çalışamazlar.

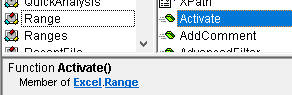
Ör: **Application.InputBox** ifadesindeki InputBox, bir **metod** olup Application nesnesine ihtiyaç duyar. Aşağıdaki tanımlardan da görüleceği üzere Application'ın hem ikonundan hem de önündeki Class ifadesinden bir alttaki Inputbox’tan farklı olduğu görülmekte.



**Normal** **Inputbox** ise bir **fonksiyon** olup nesneye ihtiyaç duymaz. Bu fonksiyon, Interaction **modülü** içinde tanımlanmıştır, bir class içinde değil.



Yukarıda, fonksiyonlar "genelde" bir değer döndürür dedik. Bazen, sebebini anlayamadığım bir şekilde, bazı kişilerin(hatta Microsoft'un kendisinin bile) değer döndürmeyen Function prosedürler yazdığını görüyoruz. Bu hem Modül fonksiyonları hem de Class metodları için geçerlidir. Mesela, aşağıda Range classının Activate metodunu görüyoruz. Deklarasyonu **Function Activate()** şeklinde yapılmış.



AddComment metodu ise sonunda **As Comment** ifadesine sahip. İşte bu “As” ile başlayan kısım bize fonksiyonun "dönüş değerini" verir.

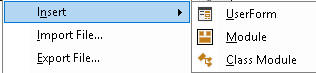


Eğer bir metod geriye bir şey döndürüyorsa bu arka planda kesinlikle bir Function prosedür olarak hazırlanmıştır, ancak değer döndürmüyorsa Sub prosedür olarak hazırlanmış olabileceği gibi sebebini hala anlamadığım bir şekilde Function prosedür olarak da hazırlanmış olabilir.

#### Modül

Prosedürlerimizi yazdığımız yerlere **Modül** denmektedir. Bunlar Standart modül, Class Modül ve Userform modüller olabilir. Ancak biz kitabın büyük kısmında standart modüllerde çalışacağız. Class modüllere çok az kod yazacağız. Eventler bölümünde ise Workbook ve Worksheet isimli class modüllere(evet bunlar da class modüldür) kod yazacağız.

VBE içinde Project penceresinde herhangi bir workbookta sağ tıklayıp Insert'e gelince gördüğümüz seçenekler modül seçenekleridir.



### Yorum Satırları

Yukarıdaki örneklerde gördüğünüz üzere ' işareti ile yorumlar yazılabilmektedir. Yorumlar yeşil renkte görünürler.

Yorumlar önemlidir, özellikle bazı makroları arada bir çalıştırıyorsak ve makronun nasıl çalıştığını aklımızda tutamıyorsak yorumlarda bunu açıklamak akıllıca olacaktır. Bir diğer gerekçe de, biz gittikten sonra yerimize gelecek kişiye de yol gösterici bir şeylerin olması gerektiğidir.

NOT: Bir diğer yorum ekleme yöntemi REM ifadesini kullanmaktır, buna neden gerek duyulmuş hiçbir fikrim yok, ben genelde ' işaretini kullanıyorum.

Sub prosedüradı()

REM açıklamalar buraya yazılır

End Sub

### Sabırsızlananlar için küçük bir ara

Hadi şimdi biraz kod yazalım. Boş bir dosya açalım. Önceki konularda gördüğümüz gibi Personal.xlsb dosyamıza gidip Modules'e sağ tıklayalım ve yeni bir modül ekleyelim. Sonra aşağıdaki kodu oraya yapıştıralım. Kodun içinde herhangi bir yerdeyken F5 tuşuna basalım ve kodumuzu çalıştıralım. (Bunun yerine üstteki araç çubuğunda Play işaretine de tıklayarak makronuzu çalıştırabilirsiniz)

Sub ilkörnek()

Range("A1").Select

Selection.Value = 10

Range("A2").Select

Selection.Value = 20

Range("A3").Formula = "=A1+A2"

Range("A4").Value = Range("A1").Value + Range("A2").Value

End Sub

Bu örnekte önce A1 hücresini seçtim, sonra mevcut olan bu seçime yani A1'e 10 değerini yazdırdım. Sonra A2 hücresini seçip oraya da 20 değerini yazdırdım. A3 hücresine ise A1+A2 toplamını formül olarak yazdırdım, ancak A4 hücresini yine aynı toplamı değer olarak yazdırdım.

Gördüğünüz üzere **Value** değerini hem değer atamada hem de değer okumada kullanabildim. İşte bazı özelliklerin hem değer yazma hem değer okuma durumu varken, bazısında sadece değer yazma, bazısında ise sadece değer okuma olabilmektedir. MSDN'de Excel Nesne modeli incelemesi yaparken bunlar şu şekilde gösterilir.

* Return or sets : Hem atanır, hem okunur
* Sets: Sadece atanır
* Returns: Sadece okunur

Şimdi bir kod daha yazalım. Bu sefer yeni modül eklemek yerine aynı modülde, yukardaki kodun hemen altına şu kodları yapıştıralım.

Sub kucukyap()

Set Aktif = Selection

For Each s In Aktif

buyuk = StrConv(s.Value, vbLowerCase)

s.Value = buyuk

Next s

End Sub

Bu kod da seçtiğimiz hücrelerdeki tüm harfleri küçük harfe dönüştürür. Deneme yapmak için A1 hücresine VOLKAN, B1 hücresine kendi adınızı, C1 hücresine de EXCEL yazın, sonra bu üçünü seçip VBE'ye gelip F5 ile kodu çalıştıralım.

Şimdilik bu kadar yeter, isterseniz devam etmeden önce siz de biraz Makro Kaydedici ile alıştırma yapın.

## Değişkenler ve Veri Tipleri

### Değişkenler

Değişkenler, adı üstünde, tüm kod çalışırken zamanla içeriği değişebilecek alanlardır.(Tabiki tüm kod boyunca sabit de kalabilirler, adları değişken diye illa değişmek zorunda değillerdir.)

#### Neden ve Nasıl?

Değişken tanımlamamızın iki temel nedeni var:

* Bilgisayarın belleğini daha verimli kullanmak için. Her değişken için uygun veri tipini tanımlarsanız bellekte o kadar az yer tutulmuş olur. Aksi halde değişkenler Variant tipte olurlar ve en yüksek hacim olan 16 Byte'lık bir hafıza işgal ederler.
* Hatalı işlem yapılmasını engellemiş olursunuz.

Birçoğunuzun bu aşamada ileri seviye programlamalar yapmayacağını varsayarak burada daha fazla bu konunun detaylarına şimdilik girmeyeceğim. **Public/Private** ayrımına da bu sayfada girmeyeceğim ancak isterseniz konularda biraz ilerleyince “İleri Terminoloji” bölümünden detaylı bilgiye ulaşabilirsiniz.

Değişkenleri tanımlamak için nerdeyse her zaman **Dim** ifadesini ve değişkenden hemen sonra **As** ifadesini ve hemen arkasından veri tipini yazarız.

Ör:

Dim İsim As String

Dim Yas As Integer

Dim Agirlik As Double

Dim Gerçekleştimi As Boolean

Bunların hiçbirini tanımlamasanız da çok büyük ihtimalle programınız çalışacaktır. Bunları tanımlayarak hem daha derli toplu hem de daha hızlı çalışan bir makroya sahip olmuş olursunuz.

Değişken kullanmak efektiftir de, mesela i=Range("A1").Value demezseniz ve Range("A1")'deki değeri birkaç yerde kullanıyorsanız ve sonradan Range("A1") yerine Range("A2")'ye referans vermek isterseniz kod içinde Range("A1") geçen her yerde değişiklik yapmak zorunda kalırsınız, ama i=Range("A1").Value demişseniz sadece bunu değiştirmeniz ve diğer yerleri "i" olarak bırakmanız yeterlidir.

Değişkenin tanımlı olması ve olmaması durumunda bellek kullanımı nasıl oluyor bir ona bakalım. Bunun için basit bir örnek üzerinden gidelim.

Sub NoVariable()

Range("A1").Value = Range("B2").Value

Range("A2").Value = Range("B2").Value \* 2

Range("A3").Value = Range("B2").Value \* 4

Range("B2").Value = Range("B2").Value \* 5

End Sub

Yukardaki kodda, VBA B2 hücresine 5 kez başvurmaktadır. Yani işlemciyi 5 kez yormuş oluyoruz. Halbuki en baştan B2'nin değerini bir değişkene atamış olsak, işlemcimiz sadece bir kez yorulmuş olacaktır. Ayrıca Eğer B2 hücresindeki değeri silip B4'e taşımamız gerekseydi, kodumuzun 5 yerinde değişiklik yapmamız gerekecekti. Bir diğer detay da bu kodda kullanılan karakter sayısı fazla ve uzun gözüküyor.

Şimdi bir de değişken kullanarak yapalım:

Sub WithVariable()

Dim i as Integer

i = Range("B2").Value

Range("A1").Value = i

Range("A2").Value = i \* 2

Range("A3").Value = i \* 4

Range("B2").Value = i \* 5  
End Sub

Şimdi ne olur? İşlemcimiz, B2 hücresine 5 kez değil 1 kez başvuracak, bu da daha az bellek kullanımı demektir. Ayrıca B2 yerine B4 kullanmak istersek sadece i değişkenini tanımladığımız satırda yani bir kez değiştirmemiz yeterli ve okunuşu daha kolay bir kod.

Bunun gibi 5-6 satırlık bir kodda çok büyük bir fark göremezsiniz, ama yüzlerce satırdan oluşan bir kodunuz olursa farkı o zaman hissedebilirsiniz.

Peki değişkenleri tanımlamadığımızda ne oluyor, program nasıl çalışıyor? VBA, bu tür değişkenleri **Variant** tipinde depolar ve her defasında bu değişkenin ne tür bir değişken olması gerektiğine karar vermeye çalışır, bu da zaman kaybıdır, yani ağır çalışan bir kod demektir. Bu nedenle Önsözde dediğim gibi işimizi doğru yapmakla kalmayalım, zarafet içinde doğru yapalım, hızlı yapalım, o yüzden değişkenlerimizi mutlaka tanımlayalım.

Aşağıda tipi belirtilerek değişken tanımladığımızda ve tanımlamadığımızda neler olduğunun süreyle ölçülmüş halini gösteren bir örneğimiz daha var. Fark oldukça açık ve net.

Bu kod, k değişkeni tanımlanmadan çalıştırıldığında 5,88 saniye sürmekteyken k'nın önündeki ' işareti kaldırılıp tanımlama yapıldığında ise 2,13 saniye sürmekte. İşte bellek yönetimi budur!

Sub timerkontrol()

Dim başlangıç As Single

Dim bitiş As Single

Dim i As Long

'Dim k As Long

başlangıç = Timer 'bu fonksiyon, kodunuzun ne kadar sürede çalıştığını tespit etmek için kullanılır

For i = 1 To 100000000 'Bu yapı For-Next döngüsüdür. Şimdilik bu döngünün nasıl kullanıldığını bilmiyor olabilirsiniz, buna takılmayın. Sonraki bölümlerde detaylıca incelenecek.

k = k + 1

Next i

bitiş = Timer

MsgBox ("İşlem süresi: " & vbNewLine & Round(bitiş - başlangıç, 2) & " saniyedir.")

End Sub

Bu arada bazı zamanlar olacaktır ki değişkenin içeriği sürekli değişebiliyordur, bazen sayısal bazen karakter bir içeriğe sahip olabiliyordur, hatta sadece sayısaldır ama bazen integer, bazen long integer veya bazen double olabiliyordur veya bir sebepten dolayı tipi bilinmiyordur, bu durumda değişkeni **Variant** olarak tanımlamaktan başka bir çare yoktur, sadece tipini Variant olarak belirtiyoruz, tip belirtmezsek de Excel onu Variant olarak algılar. Yani şu ikisi de aynı şekilde algılanır

Dim deger As Variant

Dim deger

#### Değişken tanımlama kuralları

Değişkenleri tanımlamanın bir faydası da şudur: Değişkenlerinizde mutlaka bir tane Büyük harf bulunursa ve bundan sonra değişkenleriniz hep küçük harfle yazıp space'e veya Enter'a bastığınızda, Excel ilgili harfi otomatikman büyük harfe çevirir, eğer büyük harf olmuyorsa anlarsınız ki değişkeninizi yanlış yazmışsınız. Bu nedenle, burda bu uyarıyı yapmakta da fayda görüyorum.

**UYARI!**: Değişken isimlerinizde iki kelime veya daha çok varsa mutlaka bir harf, mümkünse ortadan bir harf, büyük olsun. Kod içinde diğer her yerde küçük harfle yazıp, VBA'in büyütmesini bekleyin. Buna programcılık dilinde camelCase notasyonu denmektedir. Genel olarak tüm programlama dillerinde en çok önerilen değişken tanımlama geleneği camelCase olarak bilinen yöntemdir. Ör:

Dim okulNo

Dim ayAdı

enAltSatırNo

Böylece siz programınızın başka bir yerinde **okulno** yazdığınızda otomatikman **okulNo** olacaktır, aynı şekilde **ayadı** da otomatikman **ayAdı** olacaktır.

Değişken tanımlamada dikkat edilecek diğer hususlar şöyledir:

* İlk karakter bir harf olmalı
* boşluk, nokta(.), ünlem(!), ve şu karakterler kullanılmaz (@, &, $, # )
* Çok geçeğini sanmam ama karakter uzunluğu 255i geçmemeli
* VB rezerv kelimeleri kullanılmamalı (for, next, String vs.)
* Türkçe karakter serbest ancak başka birçok dilde geçersiz olduğu için kullanmamaya çalışın

Program boyunca sabit kalacak bir değişkeniniz varsa bunu **Const** ifadesi ile tanımlayabilirsiniz. Örneğin, mağaza sayısı 15 olan bir firma için bunu sabit olarak tanımlayabilir ve döngülerde bu sabiti kullanabilirsiniz.

Const magaza As Integer = 15 'aynı satırda tanımlanmak zorunda

For i = 1 to magaza

'Kodlar

Next i

**DİKKAT:** Sık yapılan hatalardan biri de şudur. Şimdi iki String değişken tanımlamak istediğimizi düşünelim. Eğer kelimeden tasarruf edeyim deyip şu şekilde tanımlarsak hata yaparız:

Dim metin1, metin2 as String

Çünkü bu şekilde aslında ilk değişkenin tipi belirtilmemiş oldu ve bu yüzden Variant oldu, String değil. Bu nedenle şu şekilde tanımlama yapmalıyız.

Dim metin1 as String, metin2 as String

'veya daha güvenli olsun isterseniz

Dim metin1 as String

Dim metin2 as String

#### Option Explicit

Eğer buraya kadar okuduklarınızdan değişkenleri tanımlamanın gerçekten iyi bir fikir olduğunu düşünüyorsanız, bütün modüllerinizin başında **Option Explicit** ifadesi bulunsun, bu sizi değişkenleri tanımlamaya zorlayacaktır.

Option Explicit

Sub zorunlu()

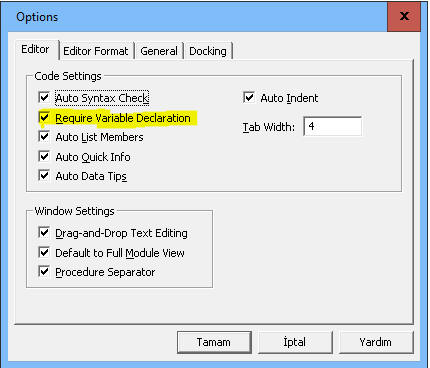
mesaj="Merhaba"

MsgBox mesaj

End sub

Yukardaki kodu çalıştırdığınızda hata verecektir, çünkü “mesaj” değişkeni tanımlanmamıştır.

Her modülün başına tek tek bu ifadeyi yazmak istemiyorsanız, şu ayarlamayı yapın. VBE içinde **Tools>Options** düğmesine basın ve aşağıdaki seçeneği işaretleyin.



#### Değer atama ve Default(Varsayılan) değerler

Değişkenleri tanımladıktan sonra onlara bir de değer atamak gerekir. Değişkenler, değer atanana kadar varsayılan değerlere sahip olurlar. Buna göre;

* Sayısal tipte bir değişken için varsayılan değer **0'dır.**
* Karekter/String tipinde bir değişken için varsayılan değer **""** yani sıfır uzunluklu stringtir.
* Nesne tipinde bir değişken için varsayılan değer **Nothing’dir**

Bu bölümün sonunda Empty, Null ve Nothing kavramlarına ait detaylı bir açıklama bulabilirsiniz.

Şimdiye kadarki örneklerde gördüğünüz üzere sayısal veya karakter tipte bir değişkene değer ataması için = işaretini kullanırız. Ör:

i=0

ay="Ocak"

Bir nesneye(Range, sheet, collection v.s) değer atamak içinse **Set** ifadesini kullanırız. Ör:

Dim hucre As Range

Dim ws As Worksheet

Set hucre=Range("A1")

Set ws=Activesheet

Nesne tanımlamalarında, kod bitiminde bu nesneleri tekrar **Nothing**olarak atamak bellek yönetimi açısından faydalıdır, böylece Excel bu nesneler için bellekte gereksiz yer ayırmayacaktır.

Sub Ornek()

Dim hucre as Range

Set hucre = Range("A1")

'Diğer Kodlar

Set hucre = Nothing

End Sub

#### Static deyimi ile değişken tanımı

Static kavramı biraz daha ileri seviye konularındandır, ancak tek başına ileri seviye konularının arasında sırıtacağı ve anlam bütünlüğü açısından da buraya daha uyduğu için buraya almak durumunda kaldım. Ayrıca ileri seviye terminolojik konuların ele alındığı bölümde Global değişkenlerle kıyaslaması da bulunmaktadır.

Bu deyimle tanımlanmış değişkenlere ben zombi değişken diyorum, zira tanımlandıkları prosedür çalışmayı tamamlasa bile yaşamaya devam ederler, ta ki içinde bulundukları workbook kapanana kadar. Bu yüzden bunlara hafızalı değişkenler de denmektedir.

##### Neden tanımlanır?

Dim ifadesi ile tanımladığımız tüm değişkenler ilgili prosedür çalıştırıldıktan sonra bellekten silinir. Ancak bazı durumlarda, tanımladığımız değişkenin prosedür çalıştıktan sonra bile bir önceki değerini tutmasını bekleriz.

Aşağıdaki örnek kodu 3 kez çalıştırdığımızda sırayla şunu görürüz: i: 1, j: 1, sonra i: 1, j: 2 ve en son i: 1,j: 3.

Sub statictest()

Dim i As Integer

Static j As Integer

i = i + 1

Debug.Print "i: " & i

j = j + 1

Debug.Print "j: " & j

End Sub

Aşağıda ise günlük hayat içinden bir örnek var.

UYARI: Bundan sonrasına devam etmeden önce **Application.OnTime** metodunun öğrenilmesinde veya genel bir fikir edinilmesinde fayda var.

Mesela her 5 dakikada çalışacak şekilde bir ayarlanmış bir prosedür düşünün. Sabah işe geldiğinizde 9: 00 gibi çalışmaya başlatıyorsunuz, akşam 18: 00'e kadar da 5 dk aralıklarla kendisi Refresh olup çalışıyor. Saat 17: 00 olduğunda belli alıcılara bir mail atsın istiyoruz diyelim. Sabah çalıştırma saatimiz her zaman tam net 9: 00 olmayacağı için "saat=17: 00 mı" diye kontrol edemeyiz. Bunun yerine saat 16'dan büyük mü diye kontrol etmeliyiz. Diyelim ki bir önceki çalıştırma işlemi 16: 58: 23'te oldu, bir sonraki 17: 03: 23te olacak, tam bu anda yakalarız, ama bir sonraki de 17: 08: 23te olacak, yine kontrole takılır ve bu şekilde 12 kez mail gitmiş olur, ki böyle birşey istemeyiz.

İşte böyle bir durumda statik bir değişken tanımlayabiliriz. Aşağıdaki örneğe bakalım. Saat 16'dan büyükse i'yi her defasında 1 artırıyoruz ancak sadece i=1 ise yani ilk kez gidecekse mail gönderimi yapıyoruz.

Sub statikornek()

Static i As Integer

If Hour(Now) > 16 Then 'Bu yapı koşullu karşılaştırma yapmamızı sağlayan IF yapısıdır. Şimdilik bu yapının nasıl kullanıldığını bilmiyor olabilirsiniz, buna takılmayın. Sonraki bölümlerde detaylıca incelenecek.

i = i + 1

If i = 1 Then Call mailproseduru

End If

Application.OnTime Now + TimeValue("00: 05: 00"), "statikornek" ' 5 dk sonra kendisini tekrar çalıştırıyor

End Sub

Sub mailproseduru()

MsgBox "mail gönderimi"

'diğer kodlar

End Sub

Bu işlemi pek tabiki saatin 17: 05: 00'ten küçük olup olmadığına bakarak da yapabilirdik ama Static konusunu anlamak adına uygun bir örnek olacağını düşündüm.

Bu arada bir diğer alternatif de i'nin değerini sayfada boş ve görünmeyen bir hücreye yazıp bunun değerini kontrol etmek olabilirdi ama static değişken kullanmak daha şık bir yöntemdir.

### Veri Tipleri ve Boyutlar

Excel’deki temel veri tiplerini iki gruba ayırabiliriz, sayısal ve sayısal olmayan. Toplamda 14 çeşit temel veri tipi bulunur. Neden bu kadar veri grubu var dersek, birinci sebep kullandıkları hafıza miktarı, ikincisi ise belli işlemlerin sadece belli veri tipleriyle yapılmasını sağlayarak hatalı işlemlerin olmasını engellemek diyebiliriz. Şimdi bunlara yakından bakalım.

#### Sayısal veri tipleri

|  |  |  |
| --- | --- | --- |
| Tip | Alabileceği değerler | Hafıza Kullanımı (Byte) |
| Byte | 0 ile 255 | 1 |
| Integer | -32.768 ile 32.767 | 2 |
| Long | -2.147.483.648 ile 2.147.483.648 | 4 |
| Single | -3.4\*10^38 ile 3.4\*10^38 (Küsuratlar ihmal edilmiştir) | 4 |
| Double | -1.7\*10^308 ile 1.7\*10^308 (Küsuratlar ihmal edilmiştir) | 8 |

**Currency** diye bir tip daha var ama kariyer hayatım boyunca hiç kullanmadım, o yüzden buraya koymuyorum.

Bunlardan en çok küsurata sahip olanı Double'dır. İhtiyacınıza göre birini kullanırsınız. Aşağıda küsuratları gösteren bir örnek var.

Sub tipler()

Dim d As Double, s As Single, c As Currency, l As Long, i As Integer, t As Byte

a = 9

b = 7

d = b / a

s = b / a

c = b / a

l = b / a

i = b / a

t = b / a

Debug.Print d, s, c, l, i, t 'sırasıyla 0,777777777777778 0,7777778 0,7778 1 1 1

End Sub

Bir de **Decimal** diye bir tip var, bu en yüksek küsurat duyarlığına sahip veri tipidir. Ancak bunun kullanımı biraz daha alengirli, o yüzden burada detaya girmeyeceğim, zaten bunu da Currency tipi gibi kariyer hayatım boyunca kullanma ihtiyacım hiç olmadı.

#### Sayısal olmayan tipler

|  |  |  |
| --- | --- | --- |
| Tip | Alabileceği değerler | Hafıza Kullanımı (Byte) |
| String(sabit boyutlu) | Max 65400 karakter | sabit boyut |
| String(değişken boyutlu) | Max 2 milyar karakter | 10+karakter sayısı |
| Date | Makul bir tarih girin yeter | 8 |
| Boolean | True ve False | 2 |
| Object | Çok çeşitli(Ör: Range, Workbook, Collection) | 4 |
| Variant(Varsayılan veri tipi) | Herşey olabilir | 16 |

Burada String biraz enteresan görünüyor. Bununla ilgili bir örnek yapalım:

Dim isim As String 'Boyut belirtmediğimiz için değişken boyutludur

isim="Volkan" 'şuan 6 karakter içerir+10=16 Byte

isim="Mustafa Kemal" 'şuan 13 karakter içerir+10=23 Byte

Şimdi diyebilirsiniz ki, Variant tipi 16 Byte tutuyor, değişken boyutlu String'i neden kullanalım ki, yukardaki örnekte 23 Byte oldu. Variant'ın 16 Byte tutma olayı, ilk tanımlama anındadır. Sonrasında, içerisindeki değişkenin tipine göre boyutu artabilir.

Devam edelim, şimdi de sabit boyutlu String nasıl tanımlanır ona bakalım.

Dim isim as String\*15

isim="Volkan" 'Hafızada şöyle tutulur "Volkan " '15i tamamlayacak kadar boşluk eklenir

#### Temel olmayan veri tipleri

Bu yukarıdaki temel veri tipleri dışında Range, Collection, Worksheet gibi çeşitli classlardan(sınıf) üretilen nesneler var, o yüzden bunlar da veri tipi olarak düşünülebilir. **Dim hucre As Range** ifadesinde olduğu gibi. Bunun dışında ileri VBA sayfalarında göreceğimiz gibi kendi tanımladığımız sınıfları da veri tipi olarak düşünebiliriz. Mesela Student diye bir class tanımladıysanız **Dim st As Student** şeklinde bir değişken tanımlayabilirsiniz.

#### Enumeration/Constant

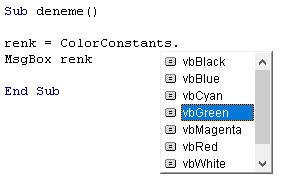
Bazı sabitler vardır ki bunlar Excel açılır açılmaz yüklenirler ve kullanıma hazırdırlar. Bunların kimisinin kullanımı faydalı iken kimisi zorunludur. Mesela **vbNullString** boş string amaçlı olarak "" yerine kullanılabilir. Böylece boşuna bellekte yer ayrılmamış olur, zira bu vbNullString zaten o anda bellektedir, yani ilave bellek tüketmez. Ancak mesaj kutularına verilen cevabın evet mi olduğunu anlamak için **vbYes** sabitini kullanmak zorunludur.

Biz bu sabitlerden 3 çeşidini kullanıyor olacağız.

* vb ile başlayanlar: VBA kütüphanesinin(library'sinin) sabitleri
* xl ile başlayanlar: Excel library'sinin sabitleri
* mso ile başlayanlar: Office library'sinin sabitleri

Bunların bir de numerik karşılıkları vardır, bunlara **Enumeration** değerleri denir. Örneğin vbYes'in değeri 6'dır. Bunların ikisi de kullanılabilir.

Mesela aşağıdaki kodu yazıp çalıştırdığınızda size bu sabitin sayısal karşılığı olan 65280’i verir.



Ayrıca kendi Enumeration tiplerinizi de oluşturabilirsiniz.

Private Enum Bolgeler

Akdeniz '0

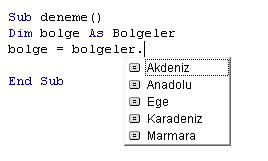
Karadeniz '1

Marmara '2

Ege '3

Anadolu '4

End Enum



### Empty, Null, Nothing kavramları

#### Empty

Konu programlamaya geldiğinde karşımıza null ve Nothing kavramları da çıkar. VBA’de değer atanmamış tüm değişkenlere otomatik olarak default değerler atanır. Bu, numerik değişkenler için 0, Stringler için ZLS(zero length string, yani sıfır uzunluklu metin), Object(nesne) tipli değişkenler için Nothing ve Variant değişkenler içinse Empty'dir.

Empty, başlangıç değeri atanmamış demektir. Null ise geçerli bir data içermeyen değişken demektir. Empty, Variant alt tipi olarak karşımıza çıkar. İstenirse bir değişkene başlangıç anında da Empty değeri verilebilir.

VBA'de yerel değişken(metotların içindeki) ve field(class seviyesindeki global eleman) ayrımı olmadığı için makro çalıştığında tüm değişkenler default değerlerine otomatik atanır.

Başlangıç değeri atanmamış veya bilinçli şekilde Empty değeri atanmış bir değişkeni yakalamak için **IsEmpty** fonksiyonunu kullanırız. Bununla birlikte bir değişkenin Empty olması çok karşılaşılan bir durum olmayacak, daha çok bir hücre değerinin içinin boş olup olmadığını sorgulayacağız. Aslında bir hücrenin içinin boş olup olmadığını sorgularken Range objesinin default değeri olan Value'yu sorgulamış oluyoruz. Yani If IsEmpty(Range("A1") ile If IsEmpty(Range("A1").Value) aynı şeydir, ve Value da Variant değer döndürdüğü için IsEmpty ile sorguladığımızda gerçekten boşsa True döndürür. Bu sorgulamayı **Len** fonksiyonu ile de yapabiliriz.

If Len(degisken)=0 Then

'veya Excel hücresi için

If Len(Range("A1"))=0 Then

ZLS tıpkı Excel’de olduğu gibi "" şeklinde ifade edilir. Bu bir stringtir ve sıfır uzunluktadır. Çoğu durumda vbNullString ile ZLS, VBA tarafından aynı şekilde yorumlanır. vbNullString tam bir string olmamakla birlikte birçok durumda ZLS yerine kullanılabilir ve hatta kullanılmalıdır da çünkü ZLS'ye göre performans açısından daha verimlidir, özellikle büyük bir döngü içinde sürekli bir "" ataması olacaksa. Çünkü ZLS, bellekte 6 byte yer kaplarken vbnullstring ise ilave yer kaplamaz, zira vbnullstring bir constant olup zaten VBA tarafından baştan yaratılmıştır ve yeniden yaratımına gerek yoktur.

Bu arada Excel’de olduğunun aksine VBA'de Blank diye bir kavram dolayısıyla IsBlank diye bir sorgulama şekli de yoktur.

Son olarak, bir şekilde başlangıç değeri olmayan Variant tipli bir değişkene IsNumeric sorgulaması yapıldığında True değeri döndürür.

#### Null

Bir değişken veri içermiyorsa bu değişken Null değere sahiptir diyebiliriz. Bir değişkenin değerinin Null olabilmesi için ya bilinçli bir şekilde Null ataması ya da Null içeren başka bir şeyle etkileşime girmesi gerekir. Empty gibi Null da sadece Variant tipinin bir özelliğidir, yani sadece Variant tipteki bir değişken Null değer alabilir. Başka tipteki bir değişkene Null atanmak istendiğinde hata alınır. Ancak Stringlere vbNullString ile null atama yapılabilir. Bir değişkenin içeriğinin Null olma olasılığı varsa ve bunu başka bir değişkenle işleme sokacaksanız, işleme sokmadan önce o anda Null içerip içermediğini IsNull ile kontrol etmeniz gerekir, aksi halde yine hatayla karşılaşırsınız.

#### Nothing

Ve son olarak bir de Nothing var. Tanımlanmış ancak henüz yaratılmamış **objelerin** değeri Nothing’dir. Bir objeye bu değer atandığında ise, objenin kendisiyle obje değişkeni arasındaki bağı koparmış oluruz.

Nothing sadece object tipindeki değişkenlere atanan bir özelliktir ve Set ifadesi ile kullanılır. Bir nesnenin Nothing olup olmadığını anlamak için "Is Nothing"(iki kelime ayrı) sorgulaması yapılır. Burada "is"(olmak) kullanımı önemlidir, eşitlik ("=") yerine "olmak" ile sorguluyoruz.

If Degisken Is Nothing Then

Şimdi de VBA'de tüm bunlar nasıl kullanılıyor, sonuçları ne oluyor, ona bir bakalım.

Buradaki kod çok uzun olduğu için buraya almadım. Buna **github**’daki kod dosyasından(**null\_empty\_nothing\_zls**) bakmanız gerekecek.

## Operatörler

### Aritmetik Operatörler

Matematikte bildiğimiz 4 işlem operatörü VBA'de de aynen geçerlidir. Bunlara ek olarak;

"\" Mod işareti olup, bir tamsayılı bölme sonunda kalanı verir. Ör: 9 \ 6=3, başka bir örnek: 8 \ 3 =2. \ işareti yerine doğrudan Mod kelimesi de yazılabilir. Küçük bir sayının büyük sayıyla Mod'u küçük sayının kendisidir.

^ işareti: Üs aldırır. 3^2=9

### Karşılaştırma Operatörleri

|  |  |  |
| --- | --- | --- |
| Amaç | Operatör | Örnek |
| Eşit mi | = | If A=B then .... |
| Büyük mü | > | If A>B then .... |
| Küçük mü | < | If A<B then .... |
| Büyük eşit mi | >= | If A>=B then .... |
| Küçük eşit mi | <= | If A<=B then .... |
| Eşitsizlik | <> | If A<>B then .... |

Özel yazımı olan bir kontrol şekli var, o da True/False kontrolü. Bu kontrolü yaparken direkt boolean tipli değişkenin kendisini yazarak sorgulayabiliriz. Ör:

Sub bool\_andor()

Dim a As Boolean

a = True

If a Then 'if a=True demek yerine

MsgBox "Doğru"

Else

MsgBox "yanlış"

End If

End Sub

### Mantıksal Operatörler

|  |  |  |
| --- | --- | --- |
| Amaç | Operatör | Örnek |
| Ve | And | If A=B and A>0 then .... |
| Veya | Or | If A>B or A=0 then .... |
| Değil | Not | If Not obj Is Nothing then .... |

**Not** operatörünün ilginç bir kullanımı da boolean tipli değişkenleri tersine döndürmek içindir. Özellikle toggle işlemlerinde(Ör: Bi düğmeye defalarca basıldığında True/False döngüsüne girme durumu) çok kullanılır.

Sub bool\_not()

Dim a As Boolean

a = True

a= not a ' a şimdi False oldu

End Sub

### Birleştirme operatörleri

İki tür birleştirme operatörü var.

+: Bu operatör iki numerik ifadeyi toplarken iki string ifadeyi birleştirir.

&: Bu hem numerik hem string değişkenleri birleştirir

+ işareti kullanıldığında değişkenlerden biri string tipte olsa bile eğer içeriği sayı ise birleşme yerine toplama olur. Aşağıda örnekler mevcut.

Sub birlestirme()

Dim a As String

Dim b As String

Dim c As Integer

Dim d As Integer

Dim e As String

a = "10"

b = "20"

c = 300

d = 5000

e = "volkan"

Debug.Print "merhaba " + e 'iki string + ile birleşir

Debug.Print "merhaba " & e 'iki string & ile birleşir

Debug.Print a + b 'iki sayısal içerikli string + ile birleşir>1020

Debug.Print a & b 'iki sayısal içerikli string & ile birleşir>1020

Debug.Print a + c 'bir sayısal içerikli string ve bir numerik + ile toplanır>310

Debug.Print a & c 'bir sayısal içerikli string ve bir numerik & ile birleşir>10300

Debug.Print c + d 'iki numerik + ile toplanır>5300

Debug.Print c & d 'iki numerik & ile birleşir>3005000

'Debug.Print c + e 'hata verir, numerik ve sayısal içerikli olmayan string toplanamaz

End Sub

### Değişkenleri kendisiyle toplama/birleştirme

Bir InputBox/MsgBox içindeki veya otomatik maildeki Body metni çok uzun ise bu metni parçalar halinde yazıp bunları sürekli kendisiyle birleştirerek ilerlemek yaygın bir yöntemdir.

Sub satırgeçiş()

mesaj = "Müşteri segmenti için bir değer giriniz. " & vbCrLf

mesaj = mesaj + "Bireysel müşteriler için 1," & vbCrLf

mesaj = mesaj + "Ticari müşteriler için 2," & vbCrLf

mesaj = mesaj + "Kurumsal müşteriler için 3"

a = InputBox(mesaj)

End Sub

Bir başka örnek de şöyle olabilir

Sub mailbodyornek()

bodymsj="Değeri arkadaşlarımız" & Chr(10) & Chr(10)

bodymsj=bodymsj+"........."

bodymsj=bodymsj+"........"

bodymsj=bodymsj+"........"

'Diğer kodlar

End Sub

Bir de sayısal değişkenlerin kendisiyle toplanması vardır. Bu yöntemi de özellikle döngüsel yapılar içinde kullanırız. Değişkenin kendisini 1 ile toplayarak, değerini artırmış oluruz.

Sub sayıartır()

Dim i As Integer

i=0

Do

'Diğer kodlar

i=i+1 'burada i'yi her defasında bir artırmış oluyoruz. Gelişmiş dillerdeki i++ ifadesinin aynısıdır

Loop Until i=100

End Sub

|  |  |  |
| --- | --- | --- |
|  |  |  |

## İnteraktivite

Makrolar kullanıcı ile belli başlı 4 şekilde iletişim kurar.

* Mesaj kutuları(MsgBox)
* Bilgi sorma kutuları (InputBox)
* Formlar(Userforms)
* File/Folder dialog kutuları

Son ikisini ayrı bölümlerde işleyeceğiz, biz şimdilik burada MsgBox ve InputBox ile haşır neşir olacağız.

### InputBox

Inputbox ile kullanıcıya çeşitli sorular sorar, ondan bir şeyler yazmasını veya sayfa üzerinde bir şeyleri seçmesini bekleriz. Kullanıcının bu girdiği değeri de bir değişken içinde depolarız. O yüzden Inputbox’ları tek başına kullanmak yerine her zaman bir değişkene atama şeklinde kullanırız.

Dim ad As String

ad = InputBox("Adınızı girin")

Inputbox'a girilen her değer bir metindir, sayı olsa bile bu metin olarak depolanır. Girilen değeri sayı olarak kullanmak istiyorsanız bunu **Val** veya buna benzer bir dönüştürme metodu (**Int, CInt, CLng, CDbl gibi**) ile sayıya çevirmeniz gerekir. Aksi halde istenmeyen sonuçlar ortaya çıkabilir. Bir örnekle bakalım:

Sub input1()

a = InputBox("bir sayı girin")

b = InputBox("ikinci bir sayı girin")

Range("A1").Value = a + b

End Sub

Kodu çalıştıralım, a için 5, b için 7 girelim. A1 hücresinde 12 rakamını görmeyi bekleriz ama 57 yazar. Çünkü VBA metinler için birleştirme operatörü olarak + işaretini de kullanılır(bir de & işareti var).

Şimdi aynı kodu aşağıda gibi çalıştıralım, aynı değerleri girelim, bu sefer 12 sonucunu görebiliriz.

Sub input2()

a = InputBox("bir sayı girin")

b = InputBox("ikinci bir sayı girin")

Range("A1").Value = Val(a) + Val(b)

End Sub

Bir diğer alternatif de, numerik olmasını istediğimiz değişkenleri baştan numerik olarak tanımlamaktır.

Sub input3()

Dim a As Integer

Dim b As Integer

a = InputBox("bir sayı girin")

b = InputBox("ikinci bir sayı girin")

Range("A1").Value = a + b '12 yazar

End Sub

#### Kullanım şekli ve diğer InputBox

Nesne Modelini anlatırken Class’lardan ve Library'lerden bahsetmiştim. İşte bu yukarda gördüğümüz InputBox da **VBA Library**'si içinde **Interaction** class'ına ait bir fonksiyondur. Birçok fonksiyon gibi bu da parametre alır.

Syntax'ı şöyledir: **InputBox(Prompt[,title][,default][,xpos][,ypos][,helpfile,context])**

Bu parametrelerden sadece köşeli parantez içine alınmamış olan Prompt parametresi zorunlu olup diğerleri opsiyoneldir. Önemlilerin açıklaması ise şöyledir:

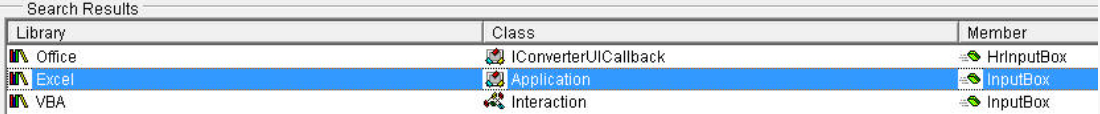
**Prompt**: Kullanıcıya ne girişi yapmasını söyleyeceğimiz ifade. Ör: Adınızı giriniz.

**Title**: Inputbox kutusunun başlığını set edebilirsiniz

**Default**: Faydalı bir özelliktir, kullanıcıya bazı durumlarda kutu içinde hazır bir değer sunabilirsiniz. Genelde, en çok girilen değerleri tahmin ederek girebilirsiniz. Örneğin "Bir il kodu girin" deyip, default değer olarak da İstanbul’un kodu olan 34ü yazabilirsiniz.

Yukarda Inputbox'ın bize aktif sayfadan bir seçim de yaptırabileceğini söylemiştim. Ancak yukardaki kodları çalıştırdığınızda bunu yapamazsınız, isterseniz bir deneyin, sonra tekrar gelin. Peki neden böyle söyledim. Çünkü bir Inputbox'ımız daha var, bu işlemi o yapar ve kendisi **Excel Library**'sindeki **Application** nesnesinin bir metodudur. Terminoloji konusunda belirttiğimiz gibi, bu Inputbox metodu diğer metodlar gibi bir nesneye ihtiyaç duyar, yani tek başına kullanılamaz, o nesne de Application nesnesidir. İlki function iken ikincisi metoddur.

Zaten aşağıdaki resimden de görüleceği üzere bağlı oldukları classların iconları bile farklı.



İkinci Inputbox'ımızın syntax'ı ise şöyledir: **Application.InputBox(Prompt,Title,Default,Left,Top,HelpFile,HelpContextID,Type)**

Bir önceki InputBox'tan farklı olarak en sonda bir Type parametresi görüyoruz. Bu parametrenin alabileceği değerleri ve anlamları aşağıda verilmiştir. En sık kullanacaklarımız koyu gösterilmiştir.

|  |  |
| --- | --- |
| Değer | Anlam |
| 0 | Formül |
| 1 | Sayı |
| 2 | Metin |
| 4 | True/False |
| 8 | Range(Bir hücre grubu) |
| 16 | Hata değeri |
| 64 | Dizi |

Tablodan da görüleceği üzere kullanıcıya bir hücre grubu seçtirmek için Type parametresini 8 tipinde belirtmemiz gerekiyor. Eğer kullanıcı hem metin hem sayısal birşey girebilecekse Type değerine toplam değer olan 3(1+2) yazılır.

Hemen bir örnek yapalım.

Dim sonHucre As Range

Set sonHucre = Application.InputBox(Prompt: ="Son hücreyi seçin", Type: =8)

Değişkenlerle ilgili [sayfadan](file:///E:\OneDrive\Uygulama%20Geliştirme\web%20sitelerim\excelinefendisi\Konular\VBAMakro\Temeller_DegiskenlerveVeriTipleri.aspx) hatırlayacağınız üzere nesnelere değer atamak için Set ifadesini kullanıyorduk, burada da öyle yaptık.

#### Boş geçilen kutular(Cancel veya Esc ile iptal)

Bazen kullanıcılar hiçbir değer girmeden çıkmak ister, o zaman ne olur.

Klasik Inputbox'ın dönüş değeri olan Stringtir ve bu durumda ilgili değişkene String tipinin default değeri atanır, yani **""**. O yüzden değişkenin değerinin "" olup olmadığı kontrol edilir.

Application.Inputbox metodunun dönüş değeri Varianttır, o yüzden default değer olarak **Empty** bekleriz ancak MSDN bize bu Inputbox'ta boş geçilen değerler için atanan değerin **False** olduğunu söylüyor. O yüzden değişkenin değerinin False olup olmadığı kontrol edilir, ama bu klasik Inputbox’a göre biraz daha alengirlidir. Aşağıdaki örneklere bakalım.

Kodumuzda hatalı birşey olmaması için bazı kontroller yapmamız gerekiyor. Bundan sonrasına devam etmeden önce koşullu yapıları bildiğinizden emin olun, bilmiyorsanız kısa bir bilgi edinip tekrar buraya gelin.

'klasik Inputbox

a=Inputbox("Bir değer girin")

If a<>"" Then

Msgbox "Giriş yapıldı"

'Diğer kodlar buraya

Else

Msgbox "Bir giriş yapılmadan çıkmayı tercih ettiniz"

End If

'Application'lı, String

Dim a As String

a=Application.Inputbox("Adınızı girin", Type: =2)

If a<>"False" Then 'False'ın tırnak içinde yazıldığına dikkat edin

Msgbox "Giriş yapıldı"

'Diğer kodlar buraya

Else

Msgbox "Bir giriş yapılmadan çıkmayı tercih ettiniz"

End If

'Application'lı, Integer(değişken tanımlanmaz, yani Varianttır)

a=Application.Inputbox("Yaşınızı girin", Type: =1)

If a<>False Then 'Variant her değeri alabileceği için False ifadesi aynen yazılır

Msgbox "Giriş yapıldı"

'Diğer kodlar buraya

Else

Msgbox "Bir giriş yapılmadan çıkmayı tercih ettiniz"

End If

'Application'lı, Integer(değişken tanımlanır)

Dim a As Integer

a=Application.Inputbox("Yaşınızı girin", Type: =1)

If a<>0 Then 'Sayısal ifadelerde False veya False'ın rakamsal karşılığı olan 0 kullanılabilir

Msgbox "Giriş yapıldı"

'Diğer kodlar buraya

Else

Msgbox "Bir giriş yapılmadan çıkmayı tercih ettiniz"

End If

'Applicationlu, Range

'Range seçiminde eğer kullanıcı seçim yapmazsa hata oluşur, bu yüzden bir hata kontrol mekanizması da ekleriz

've ayrıca bir seçim yapıp yapmadığını da Nothing ile kontrol ederiz

On Error Resume Next 'burayı yazmazsak hata alırız. Hata yönetim mekanizmaları için ilgili sayfaya gidip bilgi edinebilirsiniz

Dim a As Range

Set a = Application.InputBox("Bir hücre seçin", Type: =8)

If Not a Is Nothing Then

Msgbox "Seçim yapıldı"

'Diğer kodlar buraya

Else

Msgbox "Bir seçim yapılmadan çıkmayı tercih ettiniz"

End If

Şimdi son olarak tam bir örnek yapalım. Bu örnekte kullanıcıdan açık olan dosyaya kaç sayfa eklemek istediğini soracağız, detaylara takılmayın, sadece yukardaki anlatılanları pekiştirmeye çalışın.

Sub Sayfaekle()

Dim i As Integer, syf As Integer

syf = Application.InputBox("Kaç sayfa ekleyelim", Default: =3, Type: =1)

If syf = False Then 'escape'e basılıysa veya Cancel'a tıklandıysa. Bunu ayrıca if syf= 0 diye de yapabilirdik

Exit Sub

Else

For i = 1 To syf

Worksheets.Add

Next i

End If

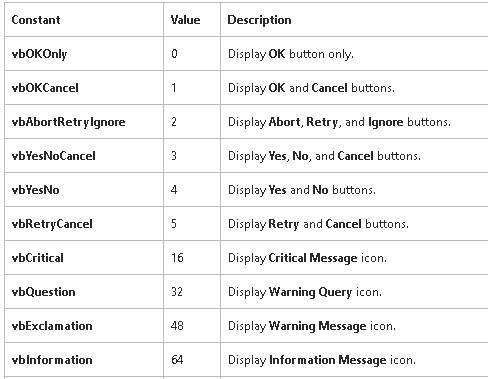
End Sub

### MsgBox

MsgBox ile ya bilgilendirme yaparız, ya da cevabı Evet/Hayır gibi sorular sorup bilgi ediniriz. Bilgilendirme yaptığımızda bunu bir değişkene atamaya gerek yoktur, ancak bilgi topladığımızda Inputbox’ta olduğu gibi bir değişkene atamamız lazım.

MsgBox da InputBox gibi VBA Library'sindeki Interaction sınıfı içinde yer alır ve syntax'ı şöyledir: **MsgBox(prompt[, buttons] [, title] [, helpfile, context])**

Burda prompt ve title InputBox’taki gibidir, son iki parametreden bahsetmeyeceğim, arzu eden araştırabilir. Burda önemli bir parametre var: buttons parametresi. Bu parametrenin alabileceği değerler şöyledir(Liste daha uzun ama çoğu gereksiz olduğu için buraya almadım, hatta bunlardan da en çok YesNo ve YesNoCancel düğmelerini kullanacağımızı söyleyebilirim)



Aşağıda bilgilendirmeye örnek bir kod var

Sub MessageBox()

'Uzunca bir kod bloğu

MsgBox "İşlem tamamdır"

End sub

Bilgi edinme örneği ise şöyle birşey olabilir.

Sub MessageBox()

cvp = MsgBox("Ana diskinizde(Ör: 'C: ') 'böl' isminde bir klasörünüz var mı?", vbYesNo) ' bu bilgi toplama mesajı

If cvp= 6 Then 'yes demek oluyor

GoTo ilerle

Else

MsgBox "O ZAMAN O KLASÖRÜ YARATIP TEKRAR ÇALIŞTIR" 'bu bilgi mesajı

Exit Sub

End If

ilerle:

'diğer kodlar

End sub

Gördüğünüz üzere cvp değerinin değerini 6 gibi bir sayıyla ölçtük. İşte VBA’de bazı sabitlerin(constant) böyle sayısal değerleri vardır, ikisi de kullanılabilir. Tüm düğmeler ve değerleri şöyle.

|  |  |
| --- | --- |
| Sabit | Değer |
| vbOK | 1 |
| vbCancel | 2 |
| VBAbort | 3 |
| vbRetry | 4 |
| vbIgnore | 5 |
| vbYes | 6 |
| vbNo | 7 |

InputBox'ta olduğu gibi MsgBox'ın da iptal edilmesi söz konusu olabilmektedir. Tabi eğer buton türü olarak Cancel varsa. Aksi halde Esc tuşu da işe yaramamaktadır.

Bu örnekten çıkış mümkün değilken,

Sub msgbox1()

On Error GoTo hata

a = MsgBox("Cevap verirmisin", vbYesNo)

'Diğer kodlar

Exit Sub

hata:

Debug.Print Err.Description

End Sub

bunu iptal edebilirsiniz.

Sub msgbox1()

On Error GoTo hata

a = MsgBox("Cevap verirmisin", vbYesNoCancel)

If a = vbYes Then

MsgBox "Evet denildi"

ElseIf a = vbNo Then

MsgBox "Hayır denildi"

Else

MsgBox "Seçimi iptal ettiniz"

End If

Exit Sub

hata:

Debug.Print Err.Description

End Sub

Önemli bir detay da, MsgBox'ın bilgi toplama formundayken mutlaka ()'ler içinde kullanılmasıdır. Mesaj verirken ise genelde () olmadan kullanılır, ama parantezli kullanımı da sorunsuz çalışır.

### Kullanıcı dostu mesajlar

Şimdi kod yazmada biraz deneyim kazandığımıza göre uzun kodlar yazarken nelere dikkat etmemiz gerekir ona bir bakalım.

#### Kullanıcı dostu kodlama

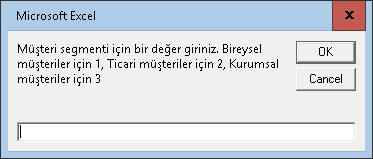
Kullanıcılara bazen MsgBox ile bazen Inputbox ile çeşitli mesajlar yayınlamak gerekecek. Kullanıcı bu mesajları rahat okusun diye gerekli yerlerde satır geçişlerini yapmanız lazım. Bir örnekle ne demek istediğimiz daha iyi anlatabilirim.

Şimdi aşağıdaki kodu, bir modül içine yazıp F5 ile çalıştıralım. Görüntü aşağıdaki gibi olup, kullanıcının okuması açısından çok kolay değildir.

Sub satırgeçiş()

a = InputBox("Müşteri segmenti için bir değer giriniz. Bireysel müşteriler için 1, Ticari müşteriler için 2, Kurumsal müşteriler için 3")

End Sub

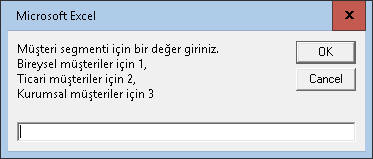


Şimdi bir de bu kod nasıl daha düzenli hale getirilir ona bakalım: Her cümle ve seçenek arasına bir ifade koyarak. Bu ifade **vbCrLf** ifadesidir ve cümleleri bir alt satıra taşır, bunun yerine vbCr veya **vbLf** veya **vbNewLine** veya **Chr(10)** ifadeleri de kullanılabilir. (Bunların dördü de MsgBox ve InputBox kullanımında aynı etkiye sahiptir, ancak hücre içine birşey yazdırırken farklı etkilere sahiptir, bunu deneyip görebilirsiniz.)

Sub satırgeçiş2()

a = InputBox("Müşteri segmenti için bir değer giriniz. " & vbCrLf & "Bireysel müşteriler için 1," & vbCrLf & "Ticari müşteriler için 2," & vbCrLf & "Kurumsal müşteriler için 3")

End Sub



#### Kodlamacı dostu kodlama

Şimdi yeri gelmişken bir de kullanıcı dostu olmakla ilgili değil ama kodlamacı dostu olmakla ilgili bir notum olacak. Yine yukardaki kodu örnek alalım, bu kod biz kodlamacılar için de okuması zor, çünkü VBE içinde kod sağa doğru uzuyor, ama kodlamacı olarak benim bunu ekranda, scrollbarı sağa sürüklemeden görebilmem lazım. Hadi gelin bunu okunaklı hale getirelim.

Yapacağımız şey basit, cümleyi nerden kesmek istiyorsak oraya bir boşluk ve sonrasında bir alt çizgi(\_) koymak. Buna **Line Contination Character** adı verilir.

Sub satırgeçiş3()

a = InputBox("Müşteri segmenti için bir değer giriniz. " & vbCrLf & \_

"Bireysel müşteriler için 1," & vbCrLf & \_

"Ticari müşteriler için 2," & vbCrLf & \_

"Kurumsal müşteriler için 3")

End Sub

Görüldüğü gibi, kod şimdi bizim için de daha okunaklı hale geldi.

Bunu yapmanın bir yolu daha var, o da metni parçalara ayırmak.

Sub satırgeçiş4()

mesaj = "Müşteri segmenti için bir değer giriniz. " & vbCrLf

mesaj = mesaj + "Bireysel müşteriler için 1," & vbCrLf

mesaj = mesaj + "Ticari müşteriler için 2," & vbCrLf

mesaj = mesaj + "Kurumsal müşteriler için 3"

a = InputBox(mesaj)

End Sub

Bu iki yöntemi sadece interaktivite sağlayan yerlerde değil başka yerlerde de kullanacağız.

Bu arada hemen iki yöntem arasındaki küçük farka da değinelim. İlk yöntem yani \_ yöntemi ile sadece metin birleştirme değil, içinde metin bile olmayan tam bir VB kodunu da parçalara ayırabiliriz, amaç yine aynı: Sağa doğru uzayan kodu tek bir ekranda tutmak. Aşağıdaki gibi.

Sub blabla()

Cells.Find(What: ="Volkan", After: =ActiveCell, LookIn: =xlValues, \_

LookAt: =xlPart, SearchOrder: =xlByRows, SearchDirection: =xlNext, \_

MatchCase: =False).Activate

End Sub

İkinci yöntemin ise tek amacı uzun metinleri parçalara ayırmaktır.

## İleri Terminoloji

Yeni başlayanlar isterse bu kısmı şimdilik geçebilir. Burayı biraz ilerleme kaydedip de ilave teknik bilgiye ve aslına bakılacak olursa daha sağlam bir altyapıya sahip olmak isteyenler için hazırladım. İsterseniz şöyle hızlıca bir göz atın ve ne hakkında olduğunu görün, ilerde kafanıza sorular takıldığında tekrar dönüp bakarsınız.

### Parametre, Argüman, ByVal, ByRef

#### Argüman vs Parametre

Bazı prosedürlerimiz hiçbir parametre olmadan, çağrıldıklarında doğrudan çalışırlar. Bazıları ise parametre alırlar. Parametreler prosedür isminden sonra gelen parantezlerin içinde yer alırlar. Ör:

Function KareAl(rakam As Integer)

KareAl=rakam\*rakam

End Function

Şimdi bu yukardaki örnekte 'rakam', KareAl fonksiyonunun bir parametresidir.

Peki argüman nedir? Argüman da işte bu KareAl fonksiyonunu çağırırken parantez içine yazdığım ifadelere denir.

Yani aslında aynı şeyden yani 'rakam'dan bahsediyoruz. İşte bu rakam ifadesi, **çağrılan** prosedür için **parametre**, **çağıran** prosedür için ise **argüman** adını alır.

Sub test()

Dim a As Integer

a=Application.Inputbox("Bir sayı girin", Type: =1)

sonuc=KareAl(a)

MsgBox "Girdiğiniz sayınını karesi şudur: " & sonuc

End Sub

Bu örnekte çağıran prosedür test() yordamıdır(Sub), ve bu yordam 'a' değişkenini KareAl fonksiyonuna argüman olarak gönderiyor. Fark ettiyseniz, argüman ve parametrenin aynen yazılması gerekmiyor. a argümanı, KareAl içine girdiğinde rakam parametresi olarak algılanıyor.

Gördüğünüz gibi aynı şeyin farklı yönlerden görünüşüne benziyor bu iki ifade, ve birçok yerde birbiri yerine bile kullanıldığını görebilirsiniz.

#### Çağırma şekilleri

Sub ve Function prosedürlerin çağrılmalarında küçük farklar bulunmaktadır. Hemen bakalım:

Bir **Sub** prosedürü çağırmanın da kendi içinde iki yolu var. Call terimini kullanarak veya kullanmayarak.

Call ifadesini yazdıysanız parantez kullanmak zorundasınız

Call yazmadıysanız parantez olmadan yazarsınız. (Ben şahsen Call yazmadan Sub çağırmanızı önermem, zira bu ifade ile o satırda başka bir prosedürün çağrıldığı hemen belli olmaktadır.)

**Function** çağırmanın da iki yolu var;

* Eğer dönen değeri bir değişkende depolayacaksanız parantez zorunludur
* Dönen değerle ilgilenmiyorsanız parantez gerekmez

Yukarıda KareAl fonksiyonumuza ek olarak aşağıda bir Sub prosedürümüz ve bir fonksiyonumuz daha var.

Sub Bilgiler(isim as String, yas as Integer, hayattamı as Boolean)

MsgBox "Kişi " & yas & " yaşında,ismi " & isim & " ve kendisi " & IIf(True, "hayatta", "hayatta değil")

End Sub

Function CokSelamSoyle(rakam As Integer)

'bu aslında bir Sub olsa daha iyi olurdu, ancak örnek olması adına function kullandım

CokSelamSoyle=rakam\*100

MsgBox CokSelamSoyle & " kez selam olsun"

End Function

Bunları aşağıdaki şekillerde çağırabiliriz;

Call Bilgiler("Volkan", 37, True) 'Sub çağırırken call varsa parantez zorunlu

Bilgiler "Volkan", 37, True 'Sub çağırırken call yoksa parantez zorunlu değil

a=KareAl(10) 'Function çağırırken bir değişkende depolayacağımız için parantez var

CokSelamSoyle 10 'dönen değerle ilgilenmiyoruz, parantez zorunlu değil

Fonksiyonlardan dönen değerle ilgilenmeye ve ilgilenmemeye çok güzel bir örnek MsgBox fonksiyonudur aslında. Hemen örneğe bakalım.

'Kullanıcıya mesaj vermek istiyorum, kullanıcıdan bir cevap toplamayacağım

Msgbox "İşlem tamam" 'parantez yok

'Kullanıcıdan bilgi toplayacağım

cevap=MsgBox("Devam edeyim mi?", vbYesNo)

If cevap=vbYes then

'kodlar buraya

End If

#### İsimli argümanlar(Named arguments)

Bir Sub veya Function prosedürü çağırdığınızda, bu prosedürün parametre tanımında görüldüğü sırasıyla argüman sağlamamız gerekir. Parametre pozisyonuna bakmadan argüman yazmak istiyorsak **isimli argümanlar** kullanabiliriz. İsimli argümanı yazmak için **Argümanismi: =Değer** syntaxını kullanırız.

Mesela aşağıdaki gibi 3 parametreli bir Sub prosedürümüz olsun.

Sub Bilgiler(isim as String, yas as Integer, hayattamı as Boolean)

'kodlar buraya

End Sub

Bu prosedürü isimli veya isimsiz nasıl çağırabiliyoruz bir bakalım(Sadece Call''lu versiyonları yazalım, Call'suz da yapılabilirdi)

'Klasik yöntem

Call Bilgiler("Volkan", 37, True) 'sadece bu sırada

'İsimli argüman yöntemi, argüman sırası önemli değildir

Call Bilgiler(hayattamı: =True,yas: =37,isim: ="Volkan")

Call Bilgiler(yas: =37,hayattamı: =True,isim: ="Volkan")

Call Bilgiler(yas: =37,isim: ="Volkan",hayattamı: =True)

'3 parametre için toplam 6 kombinasyon var, biz üçünü yazmış olduk

İsimli argümanlar, özellikle **Opsiyonel** argümanlar çok sayıda varsa oldukça kullanışlı olmaktadır. Şimdi önce Opsiyonel argümanlara sonra da ikisinin bir arada kullanılmasına bakalım.

#### Opsiyonel argümanlar

Bazen prosedürünüze öyle parametreler koymak istersiniz ki, bunlar seçime bağlı olsun, kullanıcı isterse girsin isterse girmesin. Bunlara ayrıca varsayılan değer de girebilirsiniz. Yani özellikle kullanıcıların büyük çoğunluğunun aynı değeri gireceğini umduğunuz değişkenleri opsiyonel ayarlayıp değer olarak da bu en olası değeri girebilirsiniz.

Opsiyonel parametresi olan bir prosedürü çağırdığımızda bu parametreyi kullanmazsak, ve varsayılan değer varsa bu değer dikkate alınır, varsayılan değer yoksa ilgili data tipinin varsayılan değeri kullanılır, mesela String için "", Integer için 0 gibi.

Opsiyonel parametrelere argüman sağlanıp sağlanmadığını **IsMissing** fonksiyonu ile kontrol edebilirsiniz. Ancak bu fonksiyon sadece Variant tipteki değişkenler için kullanılıyor. String, Integer gibi data tiplerindeki bir opsiyonel parametreye argüman sağlanmış mı diye bakmak için bunların default değerlerini kontrol ederiz. String için **If x=""** , integer için **If x=0** gibi. IsMissignle ilgili MSDN örneği aşağıdaki gibidir:

Sub OptionalArgs(strState As String, Optional varRegion As Variant, \_

Optional varCountry As Variant = "USA")

If IsMissing(varRegion) And IsMissing(varCountry) Then

Debug.Print strState

ElseIf IsMissing(varCountry) Then

Debug.Print strState, varRegion

ElseIf IsMissing(varRegion) Then

Debug.Print strState, varCountry

Else

Debug.Print strState, varRegion, varCountry

End If

End Sub

Aşağıdaki kodu çalıştırarak bu örneğin ne sonuçlar döndürdüğünü inceleyin

Sub test\_ops()

OptionalArgs "Teksas"

OptionalArgs "Teksas", "East"

OptionalArgs "London", "England", "UK"

End Sub

Opsiyonel parametreler, parametre listesinde [] içinde görünürler. Atlamayı düşündüğünüz opsiyonel parametreleri virgül(,) koyarak atlayabilirsiniz.

Aşağıda VBA'ın MsgBox fonksiyonunu görüyoruz. 1 adet zorunlu parametresi var, o da gösterilecek mesajı barındıran Prompt parametresi, diğerler hep opsiyoneldir.



**DİKKAT:** Opsiyonel parametreler, parametre sıralamasında en sonda yer almalıdır, ondan sonra opsiyonel olmayan bir parametre kullanırsanız kodunuzu çalıştırmadan önce hata alırsınız.

#### Opsiyonel ve İsimli argümanlar bir arada

Şimdi diyelim ki çok sayıda opsiyonel değişkeni olan bir fonksiyonu çağıracaksınız ama sadece zorunlu olanı bir de aradaki parametrelerden birini girmek istiyorsunuz. Nasıl yapardınız?

MsgBox'ı alalım yine; Sizin için sadece mesaj ve başlık göstermek yeterli diyelim. Aşağıdaki yöntemleri uygulayabilirsiniz.

MsgBox "İşlem tamam", vbOKOnly, "Rapor Sonucu" 'Named argüman kullanmıyoruz. Sıra önemli. Mecburen ilgili sırada tek tek yazarız

MsgBox "İşlem tamam", , "Rapor Sonucu" 'Named argüman kullanmıyoruz. Sıra önemli. Gereksizleri virgülle atlarız

MsgBox Title: ="Rapor Sonucu", Prompt: ="İşlem tamam" 'Named argüman: Sıra fark etmez

MsgBox "İşlem tamam", Title: ="Rapor Sonucu" 'Prompt parametresi zaten sırasında olduğu için isimli argüman kullanmadı

Sizi bilmem ama bana göre en kullanışlısı ve basiti son seçenektir, özellikle 8-10 tane parametresi olan ve sadece ilk parametresiyle son parametresini kullanmak istiyorsam, araya 6 tane virgül koymakla uğraşmak istemem, onun yerine ilk parametresi için isim kullanmayıp sonrasında sadece son parametre için isim kullanırım, bu kadar basit. Mesela **Application.InputBox** fonksiyonunu çok kullanacaksınız, bunda aşağıdaki gibi 2 argüman girmek yeterlidir.

Cevap = Application.InputBox("Bir sayı girin", Type: =1)

#### ByVal vs ByRef

Bu konu benim uzun zaman kafamı karıştıran bir konuydu, araştırma yaptığımda karşıma çıkan örnekler hep kafamda canlandırmakta zorlandığım, i ve j'lerin kullanıldığı soyut örneklerdi. Tanımlar ise MSDN çevirisi veya varyasyonu olmaktan öteye gitmiyordu. O yüzden ben hem örnekleri verirken hem de tanımı yaparken biraz daha açıklayıcı olmaya çalışacağım, umarım bunda başarılı olabilirim.

Şimdi, eğer çağırdığımız prosedüre argüman olarak gönderdiğimiz değişkenleri bu prosedürde değişikliğe uğratmayı planlıyorsak amma ve lakin bunlar geri döndüğünde hala aynı değeri korumasını istiyorsak **ByVal** ile göndeririz, geldiğinde değeri değişerek gelsin istiyorsak **ByRef** ile göndeririz. Biliyorum şuan çok karışık geldi bu ifade, o yüzden hemen aşağıdaki örneği inceleyelim sonra gelip tekrar bu tanıma bakarsınız.

Sub çağrılan(ByVal dondugunde\_degismeyen\_rakam As Integer, ByRef dondugunde\_degisen\_rakam As Integer)

dondugunde\_degismeyen\_rakam = 100

dondugunde\_degisen\_rakam = 500

'çeşitli kodlar

End Sub

Sub çağıran()

Dim a As Integer, b As Integer 'özellikle b'yi integer olarak deklare etmemiz lazım, yoksa b'yi Variant algılar ve Type Mismatch hatası verir

a = 1

b = 2

Call çağrılan(a, b)

Debug.Print a, b '1 500 yazar

End Sub

Gördüğünüz gibi, a değeri döndüğünde hala 1 değerini korudu, b değeri ise değişerek geldi. Peki neden böyle birşey isteyelim ki? Üstelik bu örnek ilk paragrafta eleştirdiğim i ve j'li örneklere benzemedi mi, evet benzedi 😊 Tabiki bu örnek giriş niteliğindeydi, şimdi örneği biraz daha mantıklı hale getirelim.

Diyelim ki bir fonksiyona yaş değişkenini göndereceğim, gönderdiğim yerde bir hesaplamada kullanacağım(garip bir hesaplama olacak ama rakamın değişmesini istediğim için böyle döngüsel bir işlem olacak), sonra bu hesap sonucuyla da kişinin hak ettiği maaşı hesaplayacağım, ama geri geldiğinde ona hala **yaş olarak ihtiyacım varsa** işte o zaman ByVal ile gönderirim. Bi de doğumyeri bilgisini göndereceğim. Gittiği prosedürde, doğumyerine göre kişiyi iki bölgeden birine tayin eden bir hesaplamada kullanacağım, ve döndüğünde bu yerleşim yeri bilgisi ile gelmesini istiyorum, iyi de peki orijinal doğumyeri bilgisine de ihtiyacım varsa ne olacak, o zaman onu çağrıyı yapan prosedürde geçici bir değişkene atarız. Hadi örneğe bakıp daha iyi anlayalım.

Function maashesapla(ByVal y As Integer, ByRef d As String)

If d = "İstanbul" Then

d = "Doğu"

Else

d = "Batı"

End If

'maaş hesaplama katsayısını elde yöntemi

For i = y To 1 Step -1

y = y + i

Next i

maashesapla = y \* 100

End Function

'çağıran prosedürümüz

Sub byvalref\_ornek()

Dim dogumyeri As String

yas = 20 'bunu inputboxla sorup daha parametrik de yapabilirdik, ancak örnek basit olsun istedim

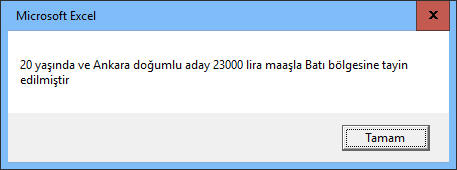
dogumyeri = "Ankara" 'bunu da inputboxla sorabilirdik, siz böyle yapın isterseniz

orijinaldogum = dogumyeri ' az sonra değişeceği için şimdiden başka bir değişkene atıyorum

MsgBox yas & " yaşında ve " & orijinaldogum & " doğumlu aday " & maashesapla(yas, dogumyeri) & " lira maaşla " & dogumyeri & " bölgesine tayin edilmiştir"

End Sub

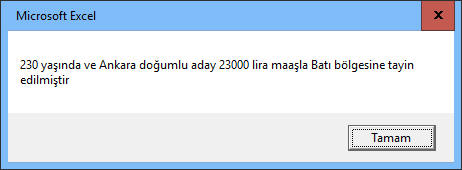
Prosedürü çalıştırınca aşağıdaki gibi bir mesajla karşılaşırız.



Burada ne oldu şimdi ona bir bakalım. Yas argümanı y parametresi olarak **maashesapla** fonksiyonuna girdi ve bir For-Next döngüsünde 230'a kadar yükseldi, yani y'nin son değeri 20 değil 230dur. Sonra da maaş hesaplandı(23000), ama ByVal olarak gönderdiğimiz için geri dönerken yine 20 olarak geldi, 230 değil.

Dogumyeri argümanı da d parametresi olarak fonksiyona girdi ve if kontrolü ile yine kendisine bölge ataması yapıldı ve değeri Batı oldu, ana prosedüre de bu değerle döndü. Ama biz başta dedik ki, Ankara değerine de ihtiyacımız olacak, o yüzden ana prosedürde orijinaldogum diye bir değişken tanımladık ve fonksiyona göndermeden önce dogumyerini bu değişkene atadık, fonksiyona girdikten sonra atasaydık hiçbir anlamı olmazdı zira çoktan değişmiş olarak gelecekti.

Peki, diyelim ki yas parametresini ByVal değil de ByRef tanımlasaydık ne olurdu. İşte şu olurdu 😊



İstenmeyen bir sonuç. Doğum yerinde olduğu gibi bunda da geçici bir değişken tanımlayıp 20 değerini koruyabilirdik ama sizce gerek var mı, elimizde ByVal diye bir seçenek varken, ne diye fazladan bir değişken yaratalım ve belleği dolduralım ki, aynı zamanda kodumuzu uzatalım ki?

Son olarak ByVal ve ByRef konusunda akılda tutulacak bazı hususlar var, onlara da bakalım:

##### Akılda tutulacaklar

* Parametreleri varsayılan gönderme şekli ByRef'tir. Yani ByVal veya ByRef olarak belirtilmemiş bir parametrenin değeri ByRef olarak yorumlanır(Birçok yüksek seviyeli dilde, Vb.Net dahil, varsayılan gönderi şekli ByValue'dur. Eğer ilerde VBA kodlarınızı VB.NET veya C# diline taşıyıp bir VSTO uygulaması yapma gibi planınız olursa sorun çıkmaması adına ByRef ifadelerini mutlaka yazın, "Nasıl olsa default değer ByRefmiş, yazmaya gerek yok" deyip boş bırakırsanız kodlarınızı Vb.Net'e kopyaladığınızda onlar otomatik olarak ByVal olarak algılanacaktır, ki bu da sıkıntılara neden olabilir, çünkü siz onların ByRef olarak işleyeceğini düşünmüştünüz).
* Çağrıda bulunduğumuz prosedürde ByRef olarak gönderilen argüman mutlaka Dim ile aynı veri tipinde tanımlanmalıdır, yoksa hata alırsınız. Buna Variant tipler de dahildir. O yüzden Dim ile tanımlanmamış bir değeri string tipteki parametreye atamaya çalışırsanız hata alırsınız. Aşağıdaki örnekte olduğu gibi:

'Çağrılan prosedür

Sub rutinkod(ByVal rapor As String)

....

End Sub

'Çağıran kod

Sub kredi ()

Rpr="kredi" 'Rpr varianttir

Rutinkod(Rpr) 'type mismatch hatasi

End Sub

* Gönderim şekliniz %90-95 oranlarında ByVal olacaktır, herkesin söylediği oran yaklaşık bu civardadır. İhtiyaca göre değişmekle birlikte, eğer gittiği yerde değişme riski yoksa boş da bırakabilirsiniz yani ByRef kalabilir. Belki ilk maddede gözünüzü çok korkutmuş olabilirim ama pratikte bu kadar korkuya gerek yok, zira çoğu durumda gönderim şeklinin ByVal veya ByRef olması sizin için bişey fark etmeyecektir.
* Parametre, ByVal olarak tanımlandığında bir nevi ilgili değişkenin kopya alma işi yapıldığı için ByRef'e göre daha yavaş çalışır. Gerçi çok büyük döngülerden oluşmadığı sürece bu farkı hissetmezsiniz. Ancak büyük döngüleriniz varsa ve kodunuz uzun çalışıyorsa, gereksiz ByVal var mı diye kontrol etmelisiniz.
* Parametre olarak Obje kullanılıyorsa bunlar mecburen ByRef tanımlanır. Çünkü Objeler referans tipli değişkenlerdir. (Referans tipli-Değer tipli arasındaki fark, çok ileri seviyeler olduğu için burada detaya girmeyeceğim)
* **ÖNEMLİ:** Fark ettiniz mi bilmiyorum ama normalde bir değer döndüren prosedürlere Fonksiyon diyoruz, ancak bir Sub prosedürde ByRef yöntemiyle ilettiğimiz değerden başka değerler de döndürebiliyoruz. Üstelik normal bir fonksiyon sadece tek değer döndürürken ByRef ile gönderi yaparak istediğimiz kadar değer döndürebiliriz. Yukardaki örnekte hem maaş tutarı hem de atama bölgesi olmak üzere 2 değer döndürmüş olduk.
* **ÖNEMLİ:** ByRef'li parametresi olan bir prosedüre ByVal'miş gibi bir değer gönderebilme teknikleri vardır. Normalde bir prosedüre %99 oranında ByRef olarak ihtiyaç duyuyorsunuz diyelim, çünkü dönen değerin değişerek gelmesini istiyorsunuz, ama ender de olsa dönen değerin değişmesini istemediğiniz anlar olabilir. Bunun için kalkıp da mevcut prosedürün bir kopyasını alıp ByRefleri ByVal yapmanın anlamı yok, zira bunun için birkaç teknik var.
  + Çağrılan prosedür Sub ise, Call olmadan ama argümanı () içine koyarak. (Bu teknikte "isimli argüman" kullanılmaz).
  + İkinci yöntem, argüman olarak değişken değil doğrudan değerin kendisini vermek. Yukarıdaki örnek için **Çağrılan(10)** demek gibi.

Sub Çağrılan(ByRef rakam as Integer)

'Kodlar burada

End Sub

'1.yöntem

Sub Çağıran()

sayı=10

Çağrılan (sayı)

End Sub

'2.yöntem

Sub Çağıran()

Çağrılan (10)

End Sub

* + Gönderilen argüman bir classa aitse(Class konusu çok ileri seviyedir, eğer başlangıç düzeyindeyseniz bunu atlayabilirsiniz). MSDN örneği şöyle:

Class Customer

Public MyValue

End Class

Sub TestMethod (ByRef MyParam)

MyParam = 5

End Sub

Dim cust

Set cust = New Customer

cust.MyValue = 123

TestMethod cust.MyValue

' cust.MyValue is still 123.

##### Gerçek dünya örneği

Bu örnek, Fonksiyon ve Collection konularının iyice anlaşılmasını gerektirdiği için bu konuları öğrendikten sonra tekrar bakmanızı tavsiye ederim.

Şimdi diyelim ki, belirli sayda rapor adını kullanan belli sayıda prosedürünüz var. Bunların hepsinde tek tek bu rapor isimlerini kullanan kodları yazmak anlamsız. O yüzden sonuç olarak bunların adını döndüren bir fonksiyon yazmaya karar verdiniz. Raporları bir Collection'a atayacak ve prosedürlerinizden de bu fonksiyonu çağıracaksınız. Çağıracağınız fonksiyonda çalışan bir If bloğu var, kullanıcıya sorulan sorunun cevabına göre de şifre değişkenine değer atanıyor. Tüm prosedürlerimizde bu IF bloğunu yazmak da anlamsız olacağı için bu kodu fonksiyon içinde tutmaya karar vermişsiniz, ki mantıklı olan da budur. O yüzden çağırdığınız bu fonksiyondan collection ile birlikte aynı zamanda bu şifreyi de döndürmeniz gerekiyor. Yani çoklu değer döndüren bir collectiona ek olarak ikinci bir değer daha döndürüyorsunuz.

Örneğimiz basit olsun diye 2 çağırıcı prosedür, fonksiyon içinde de 4 rapor yazdık. Kodlarımız şöyle:

'1.prosedür

Sub pro1()

Dim myCol As New Collection

Dim myŞifre As String

myŞifre = ""

Set myCol = col(myŞifre)

'myCol üzerinde diğer işler

Debug.Print myŞifre 'fonksiyondan değişerek gelir

End Sub

'2.prosedür

Sub pro2()

Dim myCol As New Collection

Dim şfr As String

şfr = ""

Set myCol = col(şfr)

'myCol üzerinde diğer işler

Debug.Print şfr 'fonksiyondan değişerek gelir

End Sub

'Fonksiyonumuz

Function col(ByRef şifre As String) As Collection

Dim c As New Collection

c.Add "rapor1"

c.Add "rapor2"

c.Add "rapor3"

c.Add "rapor4"

a = InputBox("DB türü girin," & vbCrLf & \_

"Oracle için 1" & vbCrLf & \_

"DB2 için 2" & vbCrLf & \_

"SQL Server için 3")

Select Case a

Case 1

şifre = "volki1144"

Case 2

şifre = "volki1234"

Case 3

şifre = "volki7788"

Case Else

şifre = ""

End Select

Set col = c

End Function

Dikkat ettiyseniz geçici şifre değişkenini her iki prosedürde de "" şeklinde argüman olarak koyduk, aslında istediğimiz şekilde koyabilirdik, nasıl olsa fonksiyon içinde değişerek geleceğini biliyoruz.

### Değişkenlerin Erişim Tipleri, Kapsamı(Scope'u) ve Yaşam Ömürleri(Lifetime)

#### ****Kapsam****

Kapsam, değişkene nereden erişilebileceği ile ilgilidir. Ör: A prosedürü içinde tanımlanmış bir değişkene sadece A prosedürü içinde erişilebilir.

Değişkenlerin tanımlandığı yere göre iki ana türü olmak üzere toplamda 3 türü vardır.

##### ****Yerel Değişkenler(Prosedür seviyesi)****

Sub ve End Sub(veya Function) arasında tanımlanan değişkenlere yerel değişken denir. Mesela aşağıdaki örnekte yol1 değişkenini sadece bu yasam1 prosedüründe, yol2 değişkenini de sadece yasam2 prosedüründe kullanabiliriz.

Sub yerel1()

Dim yol1 As String

yol1 = "C: \makrolar"

End Sub

Sub yerel2()

Dim yol2 As String

yol2 = "C: \hedefler"

End Sub

Bu arada şöyle birşey var ki, Sub-End Sub arasında tanımlanmış bir değişken if blokları veya döngüler arasında tanımlanmış veya ilk kez orada kullanılmış olsa bile bunların dışında çıktığında da yaşamaya devam eder, yani yerel değişkenlerin yaşam ömrü tüm prosedürdür, sadece tanımlı ve kullanıldığı yer değil. Örneğin aşağıdaki kodda i değişkeni For-Next döngüsünden sonra da yaşamaya devam eder.

Sub scope()

For i = 1 To 10

Debug.Print i

Next i

Debug.Print i \* 5

End Sub

Bu durum özellikle C#, C, Java gibi yüksek seviyeli dillerden gelenler için biraz şaşırtıcı olabilir, zira bunlarda yerel değişkenler sadece tanımlandıkları bloklarda geçerlidir.

##### Global Değişkenler(Modül seviyesi)

Bir de global değişkenler vardır, bunlar genelde modülün en üstünde tanımlanır, tüm Sub'lardan önce. Diyelim ki birçok prosedürümde "şubeadet" diye bir değişken olacak, o zaman hepsinde tek tek tanımlamak yerine global tanımlarız, bir kere tanımlarız. Global değişkenin amacı da budur zaten.

Dim subeadet As Integer 'global tanımlandı

Sub yerel1()

Dim yol1 As String

yol1 = "C: \performans"

'subeadet kullanılabilir

End Sub

Sub yerel2()

Dim yol2 As String

yol2 = "C: \hedefler"

'subeadet kullanılabilir

End Sub

Şimdi bu örnekte subeadet değişkenini her iki prosedür içinde de kullanabiliriz.

###### (Global değişkenlerin) Erişim seviyesi

Şimdi bu yukardaki örnekte global değişkeni aynı modülün tepesinde Dim ifadesi ile tanımladık. Dim ifadesi yerine **Private** ifadesi de kullanılabilirdi, ki global değişken tanımlarken daha çok Private ifadesini kullanmakta fayda var(Hatalı bir durum olacağından değil, sadece kodlamacılar arasında bu daha çok tercih edilir). Dim'i daha çok yerel değişkenleri tanımlarken kullanalım.

İşte bu Private(Dim) ifadesi o değişkenin sadece o modül içindeki prosedürlerde kullanılabileceği anlamına gelir. Başka modüller bu değişkeni kullanamaz.

Başka modüllerin de bu değişkeni kullanabilmesini istersek o zaman onu **Public** ifadesi ile tanımlarız.

Şimdi şöyle bir örnek yapalım. Yeni iki yeni modül oluşturalım ve aşağıdaki kodları yazalım

'Modül1 içeriği

Public publics As String

Dim dims As String

Private privates As String

Sub erisim1()

Debug.Print TypeName(publics) 'String yazar

Debug.Print TypeName(dims) 'String yazar

Debug.Print TypeName(privates) 'String yazar

End Sub

'Modül2 içeriği

Sub erisim2()

Debug.Print TypeName(publics) 'String yazar, çünkü public değişkeni kullandık

Debug.Print TypeName(dims) 'Empty yazar, çünkü modül1deki Dimle tanımlanan değişkeni değil, bu modülde henüz tanımlanmamış olan yani Variant tipteki dims değişkeninin tip adını yazar, tanımlanmamışların TypeName değieri de Emptydir.

Debug.Print TypeName(privates) 'bunda da bir yukardakiyle aynı nedenden Empty yazar

End Sub

**NOT:** Tabi bir de değişkenler konusunda gördüğümüz gibi **Const** ifadesi ile sabit tanımlama var. Sabitler de Private veya Public tanımlanabilmektedir.

Bir de **Global** keywordü(ifadesi) var ki artık pek kullanılmaz, onun yerine Public yeterlidir.(Detayı şu: Global sadece standart modüllerde kullanılabilirken, Public her yerde kullanılabilir. O zaman ne gerek var Globale? Haklısınız, çok gerek yok, ben de hiç kullanmadım ama bilin ve gördüğünüzde şaşırmayın diye dahil ettim. Bu sayfada "global" kelimesini gördüğünüz her yerde, kelimenin seviye anlamını kullanılmıştır, keyword(ifade) anlamı değil.)

##### Workbooklar arası değişken kullanımı

Peki bir Public değişkene başka bir Workbooktan erişilebilir mi? Evet. Bir örnek tasarlayalım.

Diyelim ki, birçok workbookta bölge sayısı şeklinde bir değeri kullanıyoruz. Bunu tüm workbooklarda tek tek tanımlamak yerine Personal.xlsb içinde bir kez tanımlarız. Sonra bu değişkeni döndüren public bir fonksiyon tanımlarız. Sonra tüm diğer workbooklardan bu fonksiyonu çağıracak bir kod yazarız. Böylece bankadaki bölge sayısı değiştiğinde sadece Personal.xlsb içindeki bu constantın değerini değiştirmek yeterli olacaktır.

Kodlarımız şöyle:

'Personal.xlsb içeriği

Public Const bolgeadet As Integer = 22

Public Function GetBolgeAdet()

GetBolgeAdet=bolgeadet

End Function

'herhangi bir workbook içeriğiden başvuru

Sub deneme()

blg=Application.Run("Personal.xlsb!GetBolgeAdet") 'Workbooklar arası prosedür erişimini aşağıda göreceğiz.

End Sub

**Özet**

|  |  |  |  |
| --- | --- | --- | --- |
| Konum | KeyWord | Kapsam \* | Açıklama |
| Prosedür içinde | Dim | Yerel | Sadece o prosedür içinde |
| Tepede | Dim/Private | Global | Bulunduğu modüldeki tüm prosedürler |
| Tepede | Public | Süper Global | Bulunduğu workbooktaki tüm prosedürlerde |

\* Bu sınıflandırmadaki ifadeler bana aittir. Bunlar genelde şöyle ifade edilir: Prosedür seviyesi, Modül seviyesi ve Proje Seviyesi/Workbook Seviyesi/Public Modül Seviyesi

Public SüperGlobalDeğişken As Integer

Private GlobalDeğişken As Integer

Sub Kapsam()

Dim YerelDeğişken As Integer

End Sub

#### ****Yaşam Ömrü****

Yaşam Ömrü, değişkenin değeri ne kadar süreyle koruyacağı ile ilgili bir kavramdır.

Genelde tanımladığımız değişkenlerin çalıştığı prosedür içinde değerini koruması bizim için yeterlidir. Ancak bazen prosedür bittikten sonra bile değerini korumasını bekleriz. Bunun için bir alternatif, değişkeni Statik tanımlamakken, diğeri de Global tanımlamaktır.

Statik değişkenleri Değişkenler ve Veri Tipleri konusunda anlatmıştık. O sayfayı okumadıysanız önce oraya bakmanızı tavsiye ederim.

Global değişkenler de Statik değişkenler gibi hafızalı değişkenlerdir. İçinde bulundukları workbook kapanana kadar önceki değerlerini muhafaza ederler. Ancak biz hafızalı değişken istiyorsak ve buna global seviyede ihtiyacımız yoksa  bunu static tanımlamayı tercih etmeliyiz. İyi bir programcı olmak bunu gerektirir.

Static değişkenlere ilgili prosedür içindeyken ve global değişkenlere aynı modül içindeyken sıfır ataması(değişkenimizin integer olduğunu varsayalım) yapıldığında değeri sıfırlanırken, global değişkenlere farklı modülden 0 ataması yapıldığında sıfırlanmaz, değerini korumaya devam eder. Farklı modülden de resetleme yapmak istiyorsak **End** ifadesi kullanırız.

'1.modül

Dim globalDeğişken As Integer

Sub global\_vs\_static()

'bunu 5 kez çalıştıralım

Static statikLocal As Integer

globalDeğişken = globalDeğişken + 1

statikLocal = statikLocal + 1

Debug.Print globalDeğişken

Debug.Print statikLocal

If statikLocal = 5 Then

statikLocal = 0

Debug.Print statikLocal

End If

End Sub

Sub sıfırla\_aynımodül()

globalDeğişken = 0 'aynı modüldeyken etkilidir

End Sub

'2.modül

Sub sıfırla\_farklımodül()

'globalDeğişken = 0 'etkisiz

End 'etkili

End Sub

#### Ömür ve Kapsam kıyaslaması

Aşağıdaki örnekte Prosedür1 içindeki "yerelDeğişken" değişkenin ömrü hem prosedür1 hem prosedür2 içinde de devam eder, zira prosedür1'den prosedür2'ye dallanma yaşanmış ve sonra geri gelinmiştir, ve yerel değişkenimiz hayatına devam etmektedir. Bununla beraber yerel değişkenin kapsamı sadece Prosedür1'dir, o yüzden Prosedür2 içinde onu yazdırmaya çalışırsak, bu o Prosedür içinde bambaşka bir yerelDeğişken olarak ele alınır. Prosedür2 içinde tanımı yapılmadığı için de Variant gibi algılanır ve default değeri Empty olduğu için ekrana görünmeyen bir boş karakter yazar.

Public Sub Prosedür1()

Dim yerelDeğişken As Integer

yerelDeğişken = 5

Debug.Print yerelDeğişken '5

Call Prosedür2

yerelDeğişken = 10

Debug.Print yerelDeğişken '10

End Sub

Public Sub Prosedür2()

Debug.Print yerelDeğişken 'tanımsız olduğu için Variant, Variant olduğu için de default değeri olan Empty yazar, ama "Empty" değil, görünmeyen bir karekter

End Sub

#### Kullanım şekli nasıl olacak

"Tamam iyi diyorsun da, bunlar çok havada, nasıl kullanacağız" diye düşünüyor olabilirsiniz. İşte size kendimden örnekler ve tahmini bir senaryo:

Ana makro dosyamız Personal.xlsb olduğuna göre, onun ilk modülünün tepesine birkaç değişken ve sabit tanımlayın. Mesela Mağazalar zincirinde çalışıyorsanız mağaza adedi gibi. Neden bunu Global(Modülün tepesinde) ve Public tanımlayayım ki diye mi sordunuz? Global tanımlayalım çünkü birden çok prosedürde kullanacağız, hepsinde tek tek aynı şeyi tanımlamayalım. Public tanımlayalım çünkü birden çok modülde kullanacağız.

Yine sık sık kayıt yaptığınız bir klasör varsa(veya ilerde olacaksa) bunu Public Const olarak tanımlayabilirsiniz. (Mesela her gün çalışıp o günün tarihiyle belli bir klasöre dosya kaydediyorsunuzdur)

Benim Personal.xlsb dosyamdaki global değişken ve sabitlerim şöyle. Nerdeyse onlarca makromda bölge sayısını bir döngü içinde kullanıyorum (**For i= 1 to bolgeadet** gibi). Keza otomatiğe bağlanmış tüm raporlarımın kaydolduğu bir günlükklasör path'imi de tanımlamak çok akıllıca oldu, böylece her prosedürde tek tek bu path'i belirtmem gerekmiyor(NOT: Bir ara günlük çalışan 38 raporum bulunmakta idi, düşünsenize hepsine tek tek bu klasörün adresini yazdığımı, üstelik bu adres çok uzun görünüyor, yaklaşık 80 karakter, halbuki değişkenim 12 karakter)

'ana modlümdeki global değişken ve sabitler

Public Const gunlukyol As String = "C: \.......\PERFORMANS TAKİP\GÜNLÜK ÇALIŞMALAR"

Public Const bolgesayısı As Integer = 22

Public subeadet As Integer 'şube açılış ve kapanışı nedeniyle değişebilir, o yüzden Const değil

'diğer modüllerden örnek bir prosedür

Sub tmvgelisim()

If Weekday(Now, vbMonday) = 7 Then Exit Sub

'eğer dosyayı önceki runlarda oluştuysa mail atmadan çık, yoksa çalıştır

If Application.Run("PERSONAL.xlsb!FileFolderExists", gunlukyol + "\TMV Gelişim\TMV Gelişim - " & Date - 1 & " Sonuçları.xlsm") Then Exit Sub

Workbooks.Open Filename: = \_

gunlukyol + "\TMV Gelişim\TMV Gelişim - Format.xlsm"

'dosya save olduysa mailat yoks atma

If Not Application.Run("PERSONAL.xlsb!FileFolderExists", gunlukyol + "\TMV Gelişim\TMV Gelişim - " & Date - 1 & " Sonuçları.xlsm") Then

ActiveWorkbook.Close savechanges: =False

Exit Sub

Else

ActiveWorkbook.Close

Kill "C: \geçici\geçici2.xlsm"

rapor = "TMV Gelişim"

alici = "12345;23456"

'Call Mailat2(rapor, alici) 'mail atma prosedürüdür

End If

End Sub

Mağaza sayınız/şube adediniz v.s dönem dönem değişmiyorsa(açılış ve kapanışlar nedeniyle) bir tane magazaadet/subeadet sabiti tanımlayabilirsiniz. (Ör: Public Const magazaadet as Integer=540)

Mağaza sayınız değişkense Public magazaadet as Integer dersiniz ve her prosedürünüzde bu değişkeni kullanırsınz ve o anki açık mağaza sayısını da InputBox ile kullanıcıya sordurursunuz veya şubeler/mağazalar listeniz varsa oradan okutturursunuz.

#### Altın kural

Altın kural şu: **Değişkenlerinizi olabildiğince yerel tanımlayın**, global olacaksa Private olsun, en son seçenek Public olsun. Şimdi  şöyle düşünebilirsiniz. Madem ki Public diye birşey var ve her yerde geçerli neden Private'a ihtiyaç olsun ki, tüm değişkenlerimi Public tanımlarım.

Cevap biraz uzun ve karışık, ama özetleyecek olursak, hafıza problemleri, karışık kod düzeni, hataya açık olma durumu v.s gibi nedenler yüzünden böyle yapmalısınız diyebilirim. Detaylı bilgiye eğer İngilizceniz varsa [http: //c2.com/cgi/wiki?GlobalVariablesAreBad](http://c2.com/cgi/wiki?GlobalVariablesAreBad) sayfasından ulaşabilirsiniz.

#### Çatışma

Peki, aynı anda hem global değişkeniniz hem de yerel değişkeniniz aynı isimdeyse o zaman ne olacak. Hangisi hangisi nereden bileceksiniz?

Eğer, sadece birini kullanırsanız, VBA en düşük seviyedekini yani yerel seviyedekini kullanmış olur. Eğer ikisini birden kullanmanız gerekiyorsa, global değişkeni kullanmak için başına modül adını yazmanız gerekir. Ör:

Private subeadet As Long ' global değişken

Sub bolgerapor()

Dim subeadet As Long ' yerel değişken

subeadet = 50 ' yerel değişkeni kullanıyoruz, bölgenin sube sayısını verir

Module1.subeadet= 700 ' global kullanıyoruz, bankanın sube sayısını verir

End Sub

Bana kalırsa böyle bir kullanıma da çok ihtiyacınız olmayacaktır. Bunun yerine ilgili değişkenlere farklı isim vermeye çalışın. Yine de olur da ihtiyacınız olursa çözüm yukarıdaki gibidir.

### Prosedürlerin erişim seviyesi ve çağrılması

#### Erişim seviyeleri

Prosedürlerin de değişkenler gibi erişim seviyesi bulunmaktadır ve varsayılan erişim şekli Public'tir. Varsayılandan kastım, önünde hiçbir erişim belirtecinin bulunmadığı tanımlama şeklidir.

Private olanlara sadece ilgili modülden erişilebilirken Public olanlara(Public denmemişse de Public sayılırlar) tüm modüllerden ulaşılabilir. Private tanımlanmış modüller, Excel’in Makrolar dialog kutusunda listelenmezler.

Değişkenlerde olduğu gibi prosedürlere de dışardan yani başka bir workbooktan erişilebilir, ki bu durum değişkenlerde olduğu gibi çok nadir değil, oldukça olağandır. Bunu yapmak da oldukça kolaydır, aşağıda örnekleri göreceksiniz. Örneklere geçmeden önce bir noktaya daha değinmek istiyorum. Diyelim ki bir workbookta birden fazla modül ve prosedür var, prosedürlerden bir tanesini o workbooktaki tüm modüller kullanıyor olsun ancak buna diğer workbookların erişmesini istemeyebilirsiniz(gerekçeniz ne olur bilmiyorum ama teorik olarak bunu isteyebilirsiniz). Böyle bir durumda bu prosedürü farklı bir modüle alın ve en tepeye **Option Private Module** yazın. İşte bu kod, ilgili prosedürünüzü dışardan kullanıma kapatır, yani sadece kendi workbook içindeki modüller ulaşabilir.

Farklı modüllerdeki private prosedürleri çağırırken ise **Application.Run** metodunu kullanıyoruz. Bu metodu kullanırken, parametre de atayabiliriz, parametreleri virgül ile ayırırız.

İşte bu parametre gönderme işi bize güzel bir fayda sağlıyor. Workbook ve Worksheets olaylarında detaylıca göreceğiz gerçi ama, bunların olayları(event) da Private Sub üretir ve Application.Run ile bu private sublara da ulaşabiliriz, üstelik parametre gönderme imkanına sahip olarak.

Private Sub Deneme()

Application.Run "Sheet1.Worksheet\_Change", Range("A1")

End Sub

Farklı workbooklardaki prosedürlere de ulaşabiliriz, tabiki Public tanımlandılarsa. Bunlara da yine Application.Run metodu ile ulaşırız, bu sefer prosedür önüne Workbook adını yazar ve ! işareti ekleriz. Ulaşacağımız prosedür bir **Sub** değil de **Function** ise ve döndürdüğü değeri sorgulayacaksak bu sefer her şeyi parantez içine yazarız.

Bu anlattıklarımızın özet görüntüsü aşağıdaki tabloda bulunmaktadır. Aşağıdaki ifadelerin hepsinde Call opsiyoneldir, çıkartılabilir, ancak bir prosedüre çağrıda bulunduğunu hızlıca algılayabilmemiz adına kullanmanızı tavsiye ederim. Call çağrılarında Aplication.Run'dan farklı olarak parametreler parantez içine yazılır. Call ifadesi kullanmayacaksanız parametreyi prosedürden bir boşluk sonrasına da yazabilirsiniz.

|  |  |
| --- | --- |
| Amaç | Yöntem |
| Aynı Modül içindeki bir prosedürü çağırmak | * Call private\_prosedür * Call public\_prosedür * Call prosedür(parametre1, parameter2) * prosedür parametre1, parametre2(önermiyorum) |
| Aynı Workbook, başka modüldeki prosedür | * Call public\_prosedür * Call Modül2.**public**\_prosedür(Modül1 içinde de **public**\_preosedür isminde bir prosedür varsa karışmasın diye modül ismiyle betimleriz) * Application.Run "Modül2.**private**\_prosedür"(Call ile olmaz) |
| Farklı workbooktaki prosedür (public veya private fark etmez) | * Application.Run "DiğerWB.xlsm!public\_prosedür" * Application.Run "DiğerWB.xlsm!public\_prosedür", parametre * If Application.Run("DiğerWB.xlsm!public\_function", parametre)= x Then |

Aşağıda tüm durumları içeren bir örnek var.

Sub PublicProsedür() 'publictir

MsgBox "selam ben public prosedür"

End Sub

Private Sub PrivateProsedür()

MsgBox "selam ben private prosedür"

End Sub

Sub parametreli(x As Integer, y As String)

MsgBox x & " " & y

End Sub

Function kareal(sayı As Integer)

kareal = sayı ^ 2

End Function

Sub test()

'bu modülden

Call PublicProsedür

Call PrivateProsedür

Call parametreli(1, "defa")

parametreli 1, "defa"

Debug.Print kareal(5)

x = kareal(5)

'başka modülden

Call publicmesajver

'Call privatemesajver 'hata alırız

Application.Run "Prosedür\_gizleme1.privatemesajver"

'başka workbooktan

Application.Run "schedule.xlsb!StartTimer" 'standart prosedür

Application.Run "schedule.xlsb!privatedeneme" 'privatelar da çağrılabilir

Application.Run "schedule.xlsb!Mailat", "kredi raporu" 'parametreli prosedür

x = Application.Run("schedule.xlsb!dolusay", dizi, 10, 15) 'function

End Sub

#### Makrolarınızı gizlemek

Bazen kullanıcıların Makro diyalog kutusunu açıp da kazalara neden olabilecek makroları çalıştırmasını istemezsiniz. Bunu yapmanın birkaç yolu var. Gelin onlara bir bakalım.

* İlgili makroları Private yapmak. Evet, Private prosedürler Makro penceresinde görünmezler. Ama diyelim ki makrolarınızın public olması gerekiyor, o zaman diğer yöntemlere devam edelim.
* İlgili tehlikeli makroları ayrı bir modüle alıp en tepeye **Option Private Module** demek. (Yukarda söylemiştim, bu yöntemle aynı zamanda başka workbbokların da bu modüle erişimini engellemiş oluruz)
* Prosedürünüze opsiyonel dummy bir parametre eklemek. Parametre alan prosedürler Makrolar dialog kutusunda görünmezler.

'1.modül içeriği

Sub publicmesajver()

MsgBox "selam"

End Sub

Private Sub privatemesajver()

MsgBox "selam"

End Sub

Sub dummylisub(Optional dummy As Byte)

MsgBox "selam"

End Sub

'2.Modül içeriği

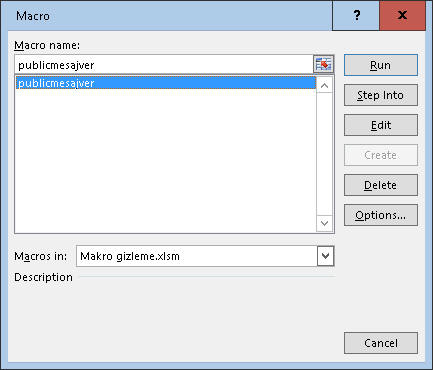
Option Private Module

Sub optionprivatelısub()

MsgBox "selam"

End Sub

Şimdi gidelim Developer menüsüne ve Macros düğmesine tıklayalım. Evet, beklediğimiz gibi sadece publicmesajver prosedürü listeye geldi. privatemesajver gelmedi çünkü private(1.yöntem), dummylisub gelmedi çünkü parametresi var(3.yöntem) optionprivatelısub da gelmedi çünkü option private modül açık(2.yöntem)



## Neler Öğrendik

VBA dünyasında kullanılan terminolojiyi gördük, ki bunlar birçok programlama dilinde de kullanılan ifadelerdir. Aslında bunları öğrenerek programlama dünyasına yavaştan girmiş oluyoruz.

Değişken kavramını ve veri tiplerini gördük.

Çeşitli operatörleri gördük.

Kullanıcıya mesaj verme ve kullanıcıdan bilgi alma yöntemlerini öğrendik.

Son olarak da biraz ileri kavramları gördük. Özellikle bu kısım ağır gelmiş olabilir, ilerleyene zamanlarda bu sayfaya tekrar dönmeyi ve bilgilerinizi hazmetmeyi ihmal etmeyin.