# 7.bölüm

**7**

# FONKSİYONLAR

Fonksiyonlar, köken olarak matematikteki fonksiyonlara dayanır. f(x)=3x+1 gibi bir fonksiyonda x bir parametredir ve bu fonksiyon parametre olarak aldığı sayının 3 katından 1 fazlasını döndürür. İşte programlama dillerindeki fonksiyonlar da bu şekilde, genelde bir parametre alırlar ve size birşey döndürürler.

Algoritma kavramından sonra en çok önem verilmesi gereken konudur.

Bu bölümde gerek VBA’in built-in fonksiyonlarını gerek kendi VBA fonksiyonlarımızı nasıl yazacağımızı göreceğiz. Beni en çok heyecanlandıran, kendimize ait Excel fonksiyonlarımızı nasıl yazacağımızı da yine bu bölümde göreceğiz.

## String Fonksiyonları

### Karakter Fonksiyonları

**Asc ve Chr**: Excel’deki, CODE ve CHAR fonksiyonlarının benzeridir. Sırayla bir karakterin ASCII kodunu ve bir ASCII kodunun karakter karşılığını verirler. Yani birbirlerini zıttı şeklinde çalışırlar.

Debug.Print Asc("a") '97

Debug.Print Chr(64) '@

Bunların W ile biten iki versiyonu daha var. Bunlarla da ASCII'nin genişletilmiş kümesi olan UNICODE karakter işlemleri yapılır.

For i = 1 To 65535

Cells(i, 1).Value = ChrW(i)

Next i

### Parça alma ve pozisyon fonksiyonları

**Left(string,n):** Metnin solundan istenilen uzunlukta(n) parça keser.  
**Right(string,n):** Metnin sağından istenilen uzunlukta(n) parça keser.  
**Mid(string,k,[n]):** Metnin ortasından belirtilen indeksten(k) itibaren belirtilen uzunlukta(n) karakter keser. Son parametre girilmezse 2.parametreden itibaren tümünü keser.  
**Len(string):** Metnin uzunluğunu(boyutunu) verir.  
**InStr([n], string, substring, [Compare] ):** n. karakterden itibaren aramaya başlayarak bir metin içinde başka bir metni veya karakteri arar, indeks numarasını(sırasını) döndürür. Bulamazsa 0 döner. Son opsiyonel parametrenin varsayılan değeri vbBinaryCompare'dir(constant olarak 0), yani default arama şekli case-sensitivedir. Küçük/büyük harf ayrımı olmadan arama yapılması isteniyorsa bu parametreye vbTextCompare(1) girilir. Ama bu parametre girildiğinde ilk parametrenin de girilmesi gerekir.  
**InStrRev(string, substring, [n], [Compare]):** InStr fonksiyonunun aramayı sondan yapan versiyonudur. Ancak bulunan indeks yine baştan sayarak bulunan indekstir. Ör: "Ardahan"da a'yı aratırsak, sondan 2. indekste bulur, bunun da baştan sayılan indeksi 6'dır, sonuç 6 olur.  
**StrReverse(string):** Metni terse çevirir.  
**Replace(metin,aranan,neyledeğiştir,başlanacak yer,adet,compare**): İlgili metin içinde aranan metni istenilen metinle değiştirir. Başlangıç indeksi 1'den farklı verilebilir, ve sadece belirli adet değişiklik yapılması istenebilir.

String tipi ilginç bir tiptir. Programlama dünyasına aşinaysanız bilirsiniz, string yapısı referans tipli ve hantal bir yapıdır. Üzerinde manipülasyon yapıldığında yeni bir kopyası oluşturulur. Replace işleminde de bu olur. Bu da şu demek; Replace edilecek bir metin olmasa bile kopya oluşturulur. O yüzden eğer döngüsel ve uzun bir replace işlemi yapacaksanız önce replace edilecek eleman var mı diye bakmakta fayda var. Aşağıdaki gibi:

If InStr(metin, aranan) <>0 Then 'eleman varmı

metin=Replace(metin, aranan, değiştir)

End If

Bu tür optimizasyon teknikleri için <http://www.aivosto.com/vbtips/stringopt.html> sitesine bakabilirsiniz.

Yukardaki açıklamalarda metin olarak verilen herşey bir hücrenin içeriği de olabilir. Bu arada yine yukarıdaki açıklamalardan anlaşıldığı üzere []'ler içindeki parametreler opsiyonel parametrelerdir.

metin="Mustafa Kemal Atatürk"

Debug.Print Left(metin,3) 'Mus

Debug.Print Right(metin,3) 'ürk

Debug.Print Mid(metin,3,2) 'st

Debug.Print Mid(metin,3) 'stafa Kemal Atatürk

Debug.Print Len(metin) '21

Debug.Print InStr(metin,"türk") '18

Debug.Print InStr(metin,"Türk") '0, çünkü case-sensitive

Debug.Print InStr(1,metin,"Türk",1) '18

Debug.Print InStr(metin,"K") '9, Kemal'in k'si

Debug.Print InStr(metin,"k") 'Atatürk'ün sonunudaki k

Debug.Print InStr(metin,"a") '5

Debug.Print InStr(10,metin,"a") '12. Aramaya 10'dan başlar ama bulduğu konumun indeksi 1'den itibaren sayılır. yani bu örnekte 3 değil, 12 bulunur.

Debug.Print InStr(10,metin,"z") '0

Debug.Print InStrRev(metin,"a") '17

Debug.Print InStrRev(metin, "a", 16) '12

Debug.Print StrReverse("volkan") 'naklov

### Dönüşüm Fonksiyonları

**LCase(string):** Metni küçük harfe çevirir.  
**UCase(string):** Metni büyük harfe çevirir.  
**Str(string) ve CStr(expression**): Str, numerik değeri metinsel ifadeye dönüştürür. Aldığı parametrenin tamamen numerik bir değer olması gerekir. Str, bu numerik ifadeyi başında bir boşluk olacak şekilde metne çevirir. Ör: 123'ü " 123" şeklinde 4 karakterli bir metin yapar. CStr ise alfanumerik bir parametre alır ve başında boşluk olmadan dönüştürür. Bu bağlamda Str bana biraz anlamsız ve gereksiz geliyor. String dönüşümlerinde sadece CStr'yi kullanın derim.  
**Val(string):** Metinsel ifadeyi rakamsal ifadeye dönüştürür. Dönüş değeri double'dır.  
**StrConv(string,tip):** Metni istenen formattaki bir metne çevirir. Tip olarak vbUpperCase(1),vbLowerCase(2),vbProperCase(3),vbUnicode(64) ve vbFromUnicode(128) değerleri girilebilir. Hepsi de aşikar olduğu için ayrıca açıklamaya gerek bulmuyorum, aşağıdaki örnekler de zaten yeterli olacaktır.

Debug.Print LCase("VOLKAN") 'volkan

Debug.Print UCase("volkan") 'VOLKAN

Debug.Print Str(123) '"123"

Debug.Print Val("123") '123

Debug.Print StrConv("merhaba dünya",1) 'MERHABA DÜNYA

Debug.Print StrConv("MERHABA DÜNYA",2) 'merhaba dünya

Debug.Print StrConv("MERHABA DÜNYA",3) 'Merhaba Dünya

Kodunuzda kullanıcından bilgi girişi istediğinizde ve bunu bir değerle(şifre v.s) karşılaştırdığınızda küçük/büyük ayrımı önemli olacağı için girilen değeri büyük harfe çevirip, karşılaştırdığınız metni de büyük harf hazırlarsanız kullanıcı kaynaklı sorunları çözmüş olursunuz.

sifre=InputBox("Şifreyi giriniz")

If UCase(sifre)="ABCD" Then

' diğer kodlar

Else

MsgBox "Yanlış şifre girdiniz"

Exit Sub

End If

### Boşluklar

**Trim(string):** Metnin solundaki ve sağındaki tüm boşlukları siler.  
**LTrim(string):** Metnin solundaki tüm boşlukları siler.  
**RTrim(string):** Metnin sağındaki tüm boşlukları siler.  
**Space(n):** n kadar boşluk üretir.

For i = 1 To 10

Cells(i, 1).Value = Space(25 - Len(Cells(i, 1).Value)) & Cells(i, 1).Value

Next



#### Sıfır uzunluklu metin(boş metin)

""(içi boş çift tırnak) yazarak sıfır uzunluklu metin elde edebiliyoruz. Bir değerin boş metin olup olmadığını anlamak için **if degisken=""** yöntemi sık kullanılmaktadır. Ne var ki bu yöntem çok sağlıklı değildir. Özellikle büyük bir döngü içindeyken kesinlikle kaçınılmalıdır.

Alternatif olarak **Len** veya **LenB** kullanılabilir.

If LenB(x)=0 Then

'diğer kodlar

End If

Boş metin atamaları da vbNullString şeklinde yapmanızı tavsiye ederim, "" olarak değil.

### Split ve Join

**Split** ile, belirli bir ayraçla birbirinden ayrılmış kelimeleri birbirinden ayırıp tek boyutlu bir dizi elde ederiz. Mesela bir hücrede ; ile ayrılmış sicil numaralarını birbirinden ayırıp, bu kişileri mail sistemindeki alıcı listesine tek tek ekleyebiliriz. Bu mail gönderim örneğinin tamamını **Diğer Uygulamalarla İletişim** bölümündeki **Outlook Programlama** konusunda göreceğiz, ancak şuan bizi ilgilendiren kısmına bakalım.

Dim emailgrubu As Variant

'........ diğer kodlar

emailgrubu=Split(Activecell.Offset(0,1).Value, ";")

'ilgili hücredeki metin 12345;12456;12894 ise, bunlar birbirinden ayrılacak ve 3 elemanlı bir dizi elde edilecektir

'....... diğer kodlar

Bir başka örnek de bir hücredeki kelimeleri saydıran veya belirli bir kelimeyi seçen bir Function yazmak olabilir. Benim kullandığım böyle bir fonksiyon var. Microsoft geliştiricileri böyle kritik bir fonksiyonu neden hala yerel fonksiyon listesine eklemiyor, gerçekten hayret ediyorum.

Function kelimesec(hucre As Range, kaçıncı As Byte, Optional ayrac As String = " ")

Dim kelimeler As String

kelimeler= Split(hucre.Value2, ayrac)

kelimesec=kelimeler(kaçıncı - 1)

End Function

Function konusunu henüz incelemediyseniz çok dert etmeyin, öğrendiğiniz zaman gelip tekrar bu örneği inceleyebilirsiniz.

**Join** ise, Splitin tersi mantıkta çalışır. Dizi elemanlarını, belirli bir ayraç ile metin olarak birleştirir. Hemen örneğe bakalım.

Dim siciller() As Integer

Dim birlesiksiciller As Stiring

'dizi eleman sayısı bir yerden okunur, bu x olsun

Redim siciller(1 to x)

For i = 1 to x

siciller(i)=cells(i,2).Value 'dizi elemanlarına değer atanıyor

Next i

birlesiksiciller=Join(siciller, ";")

### Metin Format

String modülünün **Format** fonksiyonu oldukça faydalı bir fonksiyon olmakla birlikte metinler üzerinde kullanımından ziyada tarihsel ve numerik alanları üzerinde kullanımı daha yaygındır. O yüzden bu kısımdan ziyade Tarih ve Numerik fonksiyon sayfalarında ele alacağız. Yine de sınırlı olan metin formatlama için MSDN’ye(<https://msdn.microsoft.com/VBA/Language-Reference-VBA/articles/format-function-visual-basic-for-applications>) bakabilirsiniz.

### $ işaretli fonksiyonlar

VBA'de bazı fonksiyonların aynısının sonu $ ile biten versiyonları mevcuttur. Bunların $'sız versiyonları **string tipli variant** döndürürken, $'lı versiyonları standart string döndürür. Bunun daha derin bir anlamı var, o da şu. $'sız olanlar, üzerinde işlem yaptığı metin null değer ise hata vermezken $'lı olanlar hataya neden olur. Ayrıca $'sız olanlar string döndürdükleri için hafıza ve dolayısıyla performans avantajı sunarlar.

'Şu kod hata vermez

Dim x

x = Null

Debug.Print Left(x, 3)

'Bu kod hata verir

Dim x

x = Null

Debug.Print Left$(x, 3)

## Nümerik Fonksiyonlar

Excel VBA'de kullanılmak üzere çok fazla nümerik fonksiyon bulunmamaktadır.

Bununla birlikte Excel'in kendisiyle gelen nümerik fonksiyonların birçoğunu **WorksheetFunction** fonksiyonu aracılığı ile kullanabiliriz. Bununla ilgili detaylı bilgiye ilgili sayfada değineceğiz.

Şimdi VBA içindeki birkaç faydalı numerik fonksiyona bakalım.

### Dönüştürme fonksiyonları

**Ön bilgi**: Türkiye’de biz "," işaretini ondalık ayraç olarak kullanırken, ABD ve dolayısıyla bir ABD firmasının ürünü olan Excel ve VBA "." işaretini ondalık ayraç olarak kullanır. Binlik ayraç olarak ise tersi. Gerçi regional settings(bölgesel ayarlar) ayarlaması yapıldığında Excel’in kendisi de(Sadece Excel, VBA değil) ondalık ayraç olarak "," kullanabilmektedir. Ve nitekim Türkiye'deki bilgisayarların çoğunda bu ayar yapılı haldedir. Ancak VBA "." karakterini her zaman ondalık ayraç için kullanırken, binlik ayraç diye birşey kullanmaz. VBA dünyasında "," karakteri parametre ayracı olarak kullanılır. Dolayısıyla VBA'deki 2.22 ifadesi Türkçe 2,22 algılanırken 2,22 ifadesi ise 2 ve 22 şeklinde iki ayrı parametre olarak algılanır. Bu 2,22 ifadesi tırnak içinde değilse birçok durumda tek parametre alan fonksiyonlarda hataya neden olurken, tırnak içindeyse metinsel bir ifade olarak algılanır.

**Val(String**): Parametre olarak aldığı stringi rakama çevirir. Dönüş tipi Double'dır.

Val fonksiyonu, dönüştüreceği değer içindeki ilk sayısal olmayan kısma kadar olan kısmı sayıya çevirir. Ör: 123asr45 değerini 123'e çevirir.

Debug.Print Val("00123") '123

Debug.Print Val("123asr45") '123

Debug.Print Val("2,22") '2 döndürür. çünkü virgülü non-numerik algılar ve ilk non-numerik karakterden önceki kısmı döndürür

Debug.Print Val("2.22") '2,22 döndüdür

'Debug.Print Val(2, 22) 'hata verir, sanki iki parametre var gibi algılar

Debug.Print Val(2.22) '2 döndürür, noktayı non-numerik algılar. Burada 2,22 dönmesini beklediyseniz yukarıda yazıanları tekrara okuyun lütfen

Val'in tarihleri nasıl değiştirdiğini de görelim.

Debug.Print Val("21.01.1979") '21,01 döndürür, çünkü ikinci noktayı saysal olmayan karakter olarak algılar

**CInt, CDbl v.s(İfade**): Bunlar da dönüştürme görevi görürler. **C**'den sonraki tipe dönüşüm yaparlar. Ör: CInt, içindeki değeri Integer'e dönüştürken, CDbl, Double'a, CLng Long'a. İfade olarak bir string olabileceği gibi daha küçük boyutlu bir numerik değer de olabilir.

Şunu merak etmiş olabilirsiniz. CDbl da Val de double türüne dönüştürüyor. Ne fark var? Neden 2 tane fonksiyon var? Aşağıdaki kodları ve sonuçları ile yukardaki Val'in sonuçlarını incelerseniz farkı görebilirsiniz.

Debug.Print CDbl("00123") '123

'Debug.Print CDbl("123asr45") 'hata alır. numerik olmasını bekler

Debug.Print CDbl("2,22") '2,22 döndürür

Debug.Print CDbl("2.22") '222 döndürür

'Debug.Print CDbl(2, 22) 'hata verir, sanki iki parametre var gibi algılar

Debug.Print CDbl(2.22) '2,22 döndürür

Biraz karışık gelmiş olabilir. Özeti şu: Val'i sanki Str fonksiyonu ile string hale gelmiş sayısal ifadeleri sayıya çevirmek için kullanmanız gerekirken, CDbl'i kullanıcı tarafından girilen bir sayısal metni sayıya çevirmek için kullanın.

### Yuvarlama Fonksiyonları

**Int(Number) ve Fix(Number**): İkisi de küsurlu sayıların küsuratını atar. Fark şu: Int, negatif sayılarda aşağı doğru yuvarlarken, Fix sadece küsurat atar. Ör: 3,85 için ikisi de 3 döndürürken, -3,85 için Int -4, Fix ise -3 döndürür.

**Round(Number,Duyarlılık**);Excel'in built-in(yerleşik) Round fonksiyonundan farklıdır. Excel fonksiyonunda negatif değer girerek sayı 10'un katları şeklinde de yazılabilirken, VBA'de sadece pozitif rakamlar girilebilmektedir. **Round(95.458, 1)** bize 95.5 değerini verir. Ondalık olarak virgül yerine nokta karakteri yazıldığına dikkat edin.

Negatif değer girip 10'un katları şeklinde yuvarlamak ve RoundUp/RoundDown gibi seçenekleri ele almak için WorksheetFunction'dan faydalanabiliriz.

Debug.Print Round(95.458, 1) '95,5

Debug.Print worksheetfunction.Round(95.498, -1) '100

Debug.Print worksheetfunction.RoundUp(95.498, 0) '96

Debug.Print worksheetfunction.RoundDown(95.498, 0) '95

### Matematiksel fonksiyonlar

NOT: Trigonometrik fonksiyonlar gibi MIS dünyasında kullanımı az olan veya hiç olmayan fonksiyonlara yer verilmemiştir.

**Randomize ve Rnd(Number**): Rasgele sayı üretmek için kullanılırlar. Rasgele sayı üretimi bilgisayarın sistem saati baz alınarak üretilir. Rnd ile 0-1 arasında rasgele sayı üretilir. Sayı üretildikten sonra, teknik olarak bir sonraki rasgele sayının ne olacağı tahmin edilebilir, bu da sonraki sayının rastgeleliğine şüphe düşürür. Bununla birlikte her Rnd işleminden önce **Randomize** fonksiyonu tek başına kullanılırsa sistemin baz alacağı değer bir nevi resetlendiği için bir sonraki rasgele sayının tahmini imkansızlaşır ve gerçek bir rasgele sayı üretilmiş olur. Özellikle birden fazla rasgele sayı üretmeniz gerektiği durumlarda Rnd öncesinde Randomize kullanmanız önerilir, diğer durumlarda tek başında Rnd iş görecektir.

Rnd, Excel’in RAND fonksiyonu gibi işlemektedir. Aşağıda örnekte 1 ile 100 arasında bir sayı üretilip x değişkenine atanmaktadır, bu da RANDBETWEEN(1,100) gibi.

Randomize

x=CInt(Rnd\*100)

'ancak 10-20 gibi daha üst seviyelerde bir rasgele rakam istenirse

x=WorksheetFunction.RandBetween(10,20)

**Abs(number**): Mutlak değer üretir. Ör. **Abs(-100)** 100 değerini verir.

### Sayı formatlama

Sayıları formatlamak için iki fonksiyon bulunmakta. **Format** ve **FormatNumber**. Bunların ikisi de aslında String modülünün bir fonksiyonudur. Ama konsept olarak sayılarla ilgili olduğu için burada ele almayı uygun buldum.

Ben bunlardan FormatNumber yerine Format'ı kullanmayı tercih ediyorum. Zaten FormatNumber üzerine de çok fazla online bilgi de bulunmuyor. Format fonksiyonuyla ilgili detay bilgilere ise MSDN’den([https://msdn.microsoft.com/en-us/VBA/language-reference-VBA/articles/format-function-visual-basic-for-applications](https://msdn.microsoft.com/en-us/vba/language-reference-vba/articles/format-function-visual-basic-for-applications)) ulaşabilirsiniz.

Sayılarda formatlama yaparken dikkat edilecek hususlar bellidir. Binlik ayraç, ondalık ayracı, yuvarlamalar ve gerekirse belirli miktarda fazladan 0 gösterimi.

Formatlamada iki temel karakter kullanıyoruz. **0** ve **#**. Ben bunlardan en çok #'i kullanıyorum. 0'ın farkı şu: Eğer sayının başında belirli miktar 0 olsun istenirse bu kullanılır. Neden böyle birşey gereksin? Mesela BT ekibine bir liste hazırlıyorsunuzdur, listeyi sizden 10 haneli sayılar şeklinde(başta 0 olacak şekilde) isterler. Mesela sizin göndereceğiniz listedeki sayılardan biri 123 ise bunu 0000000123 şeklinde isterler. 0, aynı zamanda küsuratlarda da fazladan 0 gösterebilir. Mesela 4 haneli küsurat olsun isteniyorsa 3,18'in gösterimi 3,1800 şeklinde olur.

Gerek 0 gerek #, eğer gerekenden az miktarda kullanılmışsa otomatikman gereken adede tamamlanır. (Aşağıda hem 0'ın hem #'in ilk örneklerinde görüldüğü gibi). Ondalık ayracın solunda sadece 1 adet 0/# olmasına rağmen dört rakamın dördü de gösterilmiştir.

Aşağıdaki örneklerle anlattıklarımız pekiştirelim. Başlamadan önce "ön bilgi" bölümünde yazan '.' ve ',' işaretlerinin kullanımına tekrar bakmanızı öneririm.

Sub formatting()

Debug.Print Format(8315.4, "0.000") '8315,400

Debug.Print Format(8315.4, "0,000") '8.315

Debug.Print Format(8315.4, "0.0") '8315,4

Debug.Print Format(8315.4, "0,0") '8.315

Debug.Print Format(8315.4, "0000.0") '8315,4

Debug.Print Format(8315.4, "0000000.0") '0008315,4

Debug.Print vbNewLine

Debug.Print Format(8315.4, "#.###") '8315,4. 0 formatından farklı olarak takip eden 0lar görünmez.

Debug.Print Format(8315.4, "#,###") '8.315

Debug.Print Format(8315.4, "#.#") '8315,4

Debug.Print Format(8315.4, "#,#") '8.315

Debug.Print Format(8315.4, "####.#") '8315,4

Debug.Print Format(8315.4, "#######.#") '8315,4. 0 formatından farklı olarak baştaki fazla 0lar görünmez.

'nokta ve virgül beraber

Debug.Print vbNewLine

Debug.Print Format(8315.4, "0,000.00") '8.315,40

Debug.Print Format(8315.4, "#,###.##") '8.315,4

Debug.Print Format(8125648315.486, "#,###.##") '8.125.648.315,49

End Sub

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

## Tarihsel Fonksiyonlar

### Ön Bilgiler

Excel’de tüm tarihlerin sayısal bir karşılığı vardır. Aslında karşılığı vardır demekten ziyade tüm tarihler bir sayıdır demek daha doğru olur. Excel bunları gösterirken tarih formatında gösterir. Tarihler tam sayı depolanırken saatler küsurlu formda tutulur. Bu sayı 1.1.1900'den itibaren geçen gün sayısıdır. Ör: 17.05.2015 arka planda 42141 sayısı olarak tutulur.

Ayraçları, bölgesel ayarlar belirler, ancak tarih “**#**” işaretleri arasında yazılırsa bölgesel ayarlardan bağımsız olarak Amerikan tarih formatına göre belirtilmiş olur: **Ay/gün/yıl**. Bunu pek kullanmanızı önermem, ancak görürseniz de şaşırmayın. Biz bildiğimiz formatta kullanalım, yoksa 5 Temmuz yazacağınız yerde 7 Mayıs gibi sonuçlar elde edebilirsiniz, bu da önemli hatalara neden olabilir.

Tarihlerle ilgili dikkat edilmesi gereken bir husus var. Formatlı Gösterim tipi, kaynak tipi veya dönüş tipi. Formatlı Gösterim şekli her zaman stringtir. Ancak kaynak tip veya dönüş tipi farklı tipler de olabilir.

Tarihlerin kaynak/dönüş tipi aşağıdakiler olabilir.

* String
* Nümerik
* Tarihsel
* # karakterleri arasında
* Tarihsel bir fonksiyondan dönen değer olarak. (Excel hücresinden veya Date, DateSerial gibi VBA fonksiyonlarından)

Konuyla ilgili daha detaylı bilgiye MSDN’den(<https://msdn.microsoft.com/en-us/library/aa227472(v=vs.60).aspx>) ulaşabilirsiniz.

### DateTime modülü

Tarihsel işlemlerimizi yaparken DateTime modülünü kullanıyor olacağız. Bu modüldeki fonksiyonlara geçmeden önce sık kullanılan property'lere bir bakalım.

#### Propertyler

**Date ve Date**$: O anki günü verirler. İlki Variant döndürürken ikincisi String döndürür. Aralarındaki fark, Variant olanda null işlemi yapabilmenizdir. Bir diğer fark da Date ile bir sayıyı matematiksel işleme tabi tutabilirsiniz ancak Date$ ile bunu yapamazsınız.

Debug.Print Date+1 'yarının tarihini verir

Debug.Print Date$+1 'hata verir. String olduğu için.

Aşağıdaki kod, üzerinde çalıştığımız dosyanın son değişikliğin tarihinin bugünden küçük olması durumunda çalışır.

'fso tanımının yapıldığı kodlar

If fso.DateModified < Date Then

'diğer kodlar

End if

**Time ve Time**$: Şu anın saatini verir. Aralarındaki fark Date/Date$ arasındaki farkın aynısıdır.

**Now**: Şu anın tarih ve saatini verir. Date ve Time'ın birleşimi gibi düşünülebilir.

#### Değişken tanımlamalar

DateTime modülünün içindeki **Date** tipi ile hem tarih hem saat tanımlanabilir. Aşağıdaki örneklerde tüm değişkenleri Variant tanımlıyoruz.

Dim tarih1 'As Date

Dim tarih2 'As Date

Dim stringtarih1 'As String

Dim stringtarih2 'As String

Dim numeriktarih1 'As Long

Dim numeriktarih2 'As Double

Dim diyezlitarih1 'As Date

Dim diyezlitarih2 'As Date

Dim hucredentarih1 'As Date

Dim hucredentarih2 'As Date

Örnek kodlara geçmeden önce, Range nesnesinin **Value** ve **Value2** özelliklerinin farkını tekrar edelim. Value, hücrenin içeriğini alırken, Value2 tarihleri numerik değere çevirerek depolar. (Tabi siz bir değişkeni Date olarak tanımladıysanız, ona Value2 sonucunu aktarsanız bile sayısal değer değil yine tarihsel formattaki değeri depolanır.)

#### Dönüş tipleri

tarih1 = Now

tarih2 = DateSerial(1979, 1, 21)

stringtarih1 = "21.01.1979"

stringtarih2 = "01.01.1979"

numeriktarih1 = 42574

numeriktarih2 = 42574.127

diyezlitarih1 = #1/21/1979# 'amerikan formatında olmak zorunda, önce ay ve ayraç oalrak da "/" girili "." değil. gün girerken 01

girsen bile enter’a basıp alta geçince kaybolurlar.

diyezlitarih2 = #1/21/1979 4: 30: 00 AM# 'amerikan

formatında olmak zorunda, önce ay ve ayraç olarak da /

hucredentarih1 = range("a1").Value 'hücre içeriği: 24.07.17 13: 14

hucredentarih2 = range("a1").Value2 'hücre içeriği: 21.01.1979

Şimdi bunların çeşitli özelliklerini yazdıralım.

Debug.Print "tarih1", vbTab, tarih1, TypeName(tarih1),VarType(tarih1), IsDate(tarih1)

Debug.Print "tarih2", vbTab, tarih2, vbTab, TypeName(tarih2), VarType(tarih2), IsDate(tarih2)

Debug.Print "stringtarih1", vbTab, stringtarih1, vbTab, TypeName(stringtarih1), VarType(stringtarih1), IsDate(stringtarih1)

Debug.Print "stringtarih2", vbTab, stringtarih2, vbTab, TypeName(stringtarih2), VarType(stringtarih2), IsDate(stringtarih2)

Debug.Print "numeriktarih1", vbTab, numeriktarih1, vbTab,TypeName(numeriktarih1), VarType(numeriktarih1), IsDate(numeriktarih1)

Debug.Print "numeriktarih2", vbTab, numeriktarih2, vbTab,TypeName(numeriktarih2), VarType(numeriktarih2), IsDate(numeriktarih2)

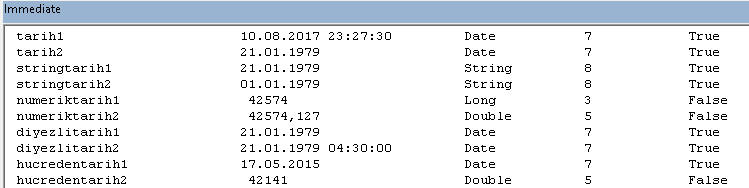
Debug.Print "diyezlitarih1", vbTab, diyezlitarih1, vbTab,TypeName(diyezlitarih1), VarType(diyezlitarih1), IsDate(diyezlitarih1)

Debug.Print "diyezlitarih2", vbTab, diyezlitarih2, TypeName(diyezlitarih2), VarType(diyezlitarih2), IsDate(diyezlitarih2)

Debug.Print "hucredentarih1", hucredentarih1, vbTab,TypeName(hucredentarih1), VarType(hucredentarih1), IsDate(hucredentarih1)

Debug.Print "hucredentarih2", hucredentarih2, vbTab,TypeName(hucredentarih2), VarType(hucredentarih2), IsDate(hucredentarih2)

Çıktısı aşağıdaki gibi olacaktır.



### Dönüşüm İşlemleri

Bazen, gösterim tipi metinsel olan tarihleri tarih tipine çevirip onlar üzerinden işlemlerinize devam etmek istersiniz. Bunun için VBA bize 4 yöntem sağlamaktadır. Dördü de küçük nüanslar göstermektedir.

**1.Yöntem: CDate(Exp)**: CDate, bir ifadeyi tarih yaparken tipini de tarih yapar. Üstelik saat v.s bilgisi varsa bunları da korur. İçine her tür ifadeyi alabilir.

x1 = CDate(tarih1)

x2 = CDate(tarih2)

x3 = CDate(stringtarih1)

x4 = CDate(stringtarih2)

x5 = CDate(numeriktarih1)

x6 = CDate(numeriktarih2)

x7 = CDate(diyezlitarih1)

x8 = CDate(diyezlitarih2)

x9 = CDate(hucredentarih1)

x10 = CDate(hucredentarih2)

Şimdi bunları yazdıralım.

Debug.Print x1, TypeName(x1), VarType(x1)

Debug.Print x2, TypeName(x2), VarType(x2)

Debug.Print x3, TypeName(x3), VarType(x3)

Debug.Print x4, TypeName(x4), VarType(x4)

Debug.Print x5, TypeName(x5), VarType(x5)

Debug.Print x6, TypeName(x6), VarType(x6)

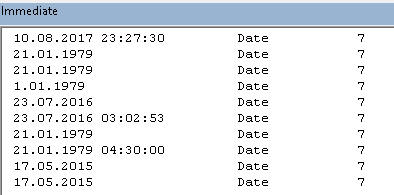
Debug.Print x7, TypeName(x7), VarType(x7)

Debug.Print x8, TypeName(x8), VarType(x8)

Debug.Print x9, TypeName(x9), VarType(x9)

Debug.Print x10, TypeName(x10), VarType(x10)

Çıktısı şöyle olacaktır:



**2.Yöntem: DateValue(Metin**): Bu fonksiyon ile metin formatındaki tarihler gerçek tarihe çevrilir. Saat v.s detayını korumaz. Argüman olarak sadece metin alır.

DateValue("21.01.1979") '21 Ocak 1979 tarihini verir

DateValue(28876) 'hata verir. Çünkü argüman olarak sadece metin alır.

**3.Yöntem: Format(exp,** format): sadece gösterim şeklini tarihsel yapar, dönüş tipi String’tir. Dönen değerle tarihsel işlem yapılamaz.

İkinci parametre olarak önceden tanımlanmış formatlar da girilebilir, kullanıcı tanımlı formatlar da. Bunların detayına <https://msdn.microsoft.com/VBA/Language-Reference-VBA/articles/format-function-visual-basic-for-applications> sayfasından ulaşabilirsiniz.

y1 = Format(tarih1, "Short Date") 'önceden tanımlanmış parametre

y2 = Format(tarih2, "dd.mm.yyyy") 'bu ve aşağıdakiler ise kullanıcı tanımlı parametredir

y3 = Format(stringtarih1, "dd.mm.yyyy")

y4 = Format(stringtarih2, "dd.mm.yyyy")

y5 = Format(numeriktarih1, "dd.mm.yyyy")

y6 = Format(numeriktarih2, "dd.mm.yyyy")

y7 = Format(diyezlitarih1, "dd.mm.yyyy")

y8 = Format(diyezlitarih2, "dd.mm.yyyy")

y9 = Format(hucredentarih1, "dd.mm.yyyy")

y10 = Format(hucredentarih2, "dd.mm.yyyy")

Şimdi de bunların çıktısını alalım

Debug.Print y1, TypeName(y1), VarType(y1)

Debug.Print y2, TypeName(y2), VarType(y2)

Debug.Print y3, TypeName(y3), VarType(y3)

Debug.Print y4, TypeName(y4), VarType(y4)

Debug.Print y5, TypeName(y5), VarType(y5)

Debug.Print y6, TypeName(y6), VarType(y6)

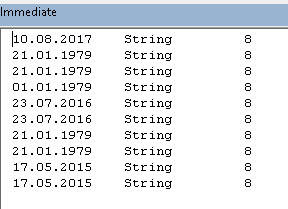
Debug.Print y7, TypeName(y7), VarType(y7)

Debug.Print y8, TypeName(y8), VarType(y8)

Debug.Print y9, TypeName(y9), VarType(y9)

Debug.Print y10, TypeName(y10), VarType(y10)

Çıktı sonucu aşağıdaki gibidir:



**4.Yöntem: FormatDateTime(d,constant**)): FormatDateTime da string döndürür, ama kullanımı daha basittir, seçenekler sınırlıdır. Seçeneklerde constantlar var, ön tanımlı veya kullanıcı tanımlı parametre yok.

z1 = FormatDateTime(tarih1, vbShortDate)

z2 = FormatDateTime(tarih2, vbLongDate)

z3 = FormatDateTime(stringtarih1, vbLongDate)

z4 = FormatDateTime(stringtarih2, vbLongDate)

z5 = FormatDateTime(numeriktarih1, vbLongDate)

z6 = FormatDateTime(numeriktarih2, vbLongDate)

z7 = FormatDateTime(diyezlitarih1, vbLongDate)

z8 = FormatDateTime(diyezlitarih2, vbLongDate)

z9 = FormatDateTime(hucredentarih1, vbLongDate)

z10 = FormatDateTime(hucredentarih2, vbLongDate)

Şimdi de çıktı alalım.

Debug.Print z1, vbTab, TypeName(z1), VarType(z1), IsDate(z1) 'short olduğu ve içinde gün ismi geçmediği için True, aşağıdakiler false false false

Debug.Print z2, TypeName(z2), VarType(z2), IsDate(z2)

Debug.Print z3, TypeName(z3), VarType(z3), IsDate(z3)

Debug.Print z4, TypeName(z4), VarType(z4), IsDate(z4)

Debug.Print z5, TypeName(z5), VarType(z5), IsDate(z5)

Debug.Print z6, TypeName(z6), VarType(z6), IsDate(z6)

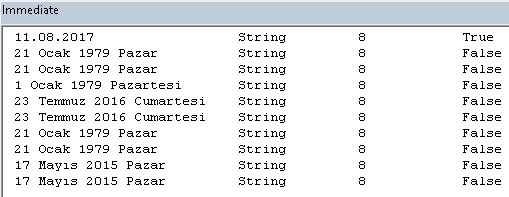
Debug.Print z7, TypeName(z7), VarType(z7), IsDate(z7)

Debug.Print z8, TypeName(z8), VarType(z8), IsDate(z8)

Debug.Print z9, TypeName(z9), VarType(z9), IsDate(z9)

Debug.Print z10, TypeName(z10), VarType(z10), IsDate(z10)

Sonuç aşağıdaki gibi olacaktır:



### Toplama/Çıkarma

VBA'de, Excel’deki yaptığımız gibi +1/-1 diyerek her zaman toplama çıkarma yapılamıyor. Bu operatörleri belli durumlarda kullanabiliyoruz. Matematiksel işlemler için belirli fonksiyonlar kullanılmalıdır.

* DateDiff/DateAdd
* DateValue("21.01.1979")+1
* Dateserial(1979,1,21)+1
* CDate("21.01.1979")+1
* #"li üzerinde işlem: #5/22/97# - #1/10/97#
* Date döndüren herhangi bir fonksiyonla. (Date/Now veya UDF gibi)

c1 = diyezlitarih1 + 1 'datevalue veya DateAdd demeye gerek kalmadan doğrudan kullanılabilir

c2 = DateAdd("d", 1, stringtarih1)

c3 = stringtarih1 + 1 'istenmeyen sonuç: noktaları uçup ekleme yapar, yani yıla eklenmiş olur, 21011980

c4 = DateValue(stringtarih1) + 1

c5 = CDate(stringtarih1) + 1

c6 = tarih1 - tarih2

c7 = DateDiff("d", tarih2, tarih1)

c8 = (tarih1 - tarih2) / 30 'yaklaşık ay sayısı

c9 = DateDiff("m", tarih2, tarih1) 'kesin ay sayısı

c10 = (tarih1 - tarih2) / 365

Şimdi de çıktılarını alalım.

Debug.Print vbNewLine

Debug.Print c1, vbTab, TypeName(c1), IsDate(c1)

Debug.Print c2, vbTab, TypeName(c2), IsDate(c2)

Debug.Print c3, vbTab, TypeName(c3), IsDate(c3)

Debug.Print c4, vbTab, TypeName(c4), IsDate(c4)

Debug.Print c5, vbTab, TypeName(c5), IsDate(c5)

Debug.Print c6, TypeName(c6), IsDate(c6)

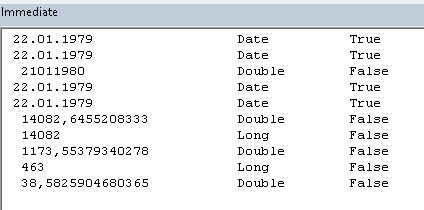
Debug.Print c7, vbTab, TypeName(c7), IsDate(c7)

Debug.Print c8, TypeName(c8), IsDate(c8)

Debug.Print c9, vbTab, TypeName(c9), IsDate(c9)

Debug.Print c10, TypeName(c10), IsDate(c10)

Çıktı şöyle olacaktır:



### Diğer işlemler

#### IsDate ile "Tarih mi?" sorgulaması

Bir değerin tarih olup olmadığını sorgulanması için içeriğin Short Date veya saatli Short Date olması lazım. Long Date veya numerik olursa  tarih olarak algılamaz.

Aşağıdaki örnekte, schedule edilmiş bir makro ile, içindeki dosya isimleri "Falanfilan raporu - 25.07.2017 Sonuçları.xlsm" gibi olan dosyalara bakıyor ve belirtilen süreden önce eski olanları sildiriyorum. Belirli süreyi tespit etme işlemini "Diğer kodlar" bölümünde yapıyorum, şuan bu kısım önemli değil. Her rapor için tespit ettiğim bu süre değerini bir dictionary’de depoluyorum. Mid(isim, Len(isim) - 19, 10) ile kestiğim kısım 25.07.2017'a denk gelen kısım ve bunun tarih olup olmadığını sorguluyorum. Ancak bazen ilgili klasörde manuel kaydedilmiş dosyalar olabiliyor, bunların silinmesini istemiyorum, keza içinde Format ifadesi geçen dosyaların da silinmesini istemiyorum.

Sub eskilerisil()

'Diğer kodlar

For Each d In klasor.Files

isim = fso.GetBaseName(d)

If Len(isim) > 19 Then

If IsDate(Mid(isim, Len(isim) - 19, 10)) Then tarih = DateValue(Mid(isim,Len(isim) - 19, 10))

Else

tarih = DateSerial(2099, 12, 31)

End If

If Not aysonumu( Date - dict(kls)) = True And tarih < Date - dict(kls) And InStr(isim, "Format") = 0 Then

Kill d

End If

Next d

End Sub

#### Saat karşılaştırma

Hour'la, TimeSerial veya TimeValue ile yapabiliriz.

If Hour(Now)>13 Then 'dakika detayı gerekli değilse

If TimeValue(Now) > #1: 00 PM# Then

If TimeValue(Now) > #13: 00# Then 'otomatik PM'e çevrilir

If TimeValue(Now) > #13: 30: 00# Then 'otomatik PM'e çevrilir

If TimeValue(Now) > TimeSerial(13, 0, 0) Then

#### Bir tarihin belirli kısımlarını alma

##### Parça fonksiyonları

tarih="31.12.2016 13: 35: 00"

Debug.Print Year(tarih) '2016

Debug.Print Month(tarih) '12

Debug.Print Day(tarih) '31

'Hour, Minute ve Second ile de saat, dakika e saniye döndürülür

##### DatePart

Yukarıdaki parça parça alan fonksiyonların hepsini tek bir fonksiyon ve parametre ile de yapabiliriz, mesela yıl almak için DatePart("yyyy",tarih) kullanılabilir. Ancak yukarıdakileri kullanmak daha kolaydır. Yani basit parça alma için bu fonksiyonu kullanmayacağız. Bununla beraber yılın kaçıncı günü, yılın hangi çeyreği, yılın hangi haftası v.s için bu fonksiyon lazım. Mesela bu söylediğim son 3 amaç için sırayla y,q,ww parametrelerini kullanırız.

Dikkat: **y** ile **yyyy** karıştırmayın. yyyy yılı verirken tek y yılın gününü verir, d de ayın gününü verir. Parametre olarak kullanılabilecek tüm ifadelere yine yukarda linkini verdiğimiz MSDN sayfasından erişebilirsiniz.

##### Ay ve hafta adı

Debug.Print MonthName(Month(tarih)) 'Aralık

Debug.Print WeekdayName(Weekday(tarih)) 'Pazar

##### TimeSerial ve DateSerial

Yıl,Ay, Gün veya Saat, Dakika, Saniye bilgilerinin bir yerden temin edilmesi durumunda da belirli tarih ve saatler elde edilebilmektedir.

Debug.Print DateSerial(2017,1,5)

Debug.Print TimeSerial(8,3,12)

#### Bazı özel tarihler

Aybaşı: DateSerial(year(Date),month(Date),1)

Aysonu: DateSerial(Year(Date), Month(Date)+1 , 1)-1 veya daha sade olarak DateSerial(Year(Date), Month(Date) + 1, 0)

Haftabaşı: Date - WeekDay(Date, vbUseSystem) + 1 ‘veya vbMonday

Haftasonu: Date - WeekDay(Date, vbUseSystem) + 7

Yılbaşı: DateSerial(Year(Date), 1, 1)

Yılsonu: DateSerial(Year(Date)+1, 1, 0)

#### Timer ile geçen süreyi hesaplama

Timer property'sini genelde bir prosedürün ne kadar sürede çalıştığını bulmak amaçlı kullanırız.

Sub timerkontrol()

Dim başlangıç As Single

Dim bitiş As Single

Dim i As Long

başlangıç = Timer

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.")

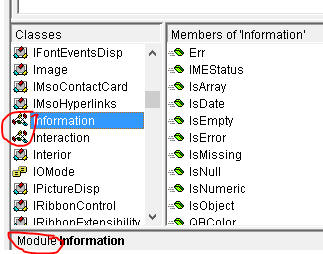
End Sub

## Diğer Fonksiyonlar

VBA'de kullanılan birçok fonksiyon var elbette. Bunların bir kısmını önceki 3 bölümde gördük. Bunun dışında farklı bölümlerde ele aldığımız(Interaktivite bölümündeki **InputBox** gibi) ve sonraki bölümlerde ele alacağımız(Dosya işlemlerindeki **ChDir** gibi) fonksiyonlar da var.

Bizim bu bölümde ele alacağımız fonksiyonlar da tıpkı bu yukarıda bahsettiklerim(**metinsel, numerik ve tarihsel**) gibi kendi başına kullanılan önemli fonksiyonlar olacak. Bunlar da öncekiler gibi çeşitli bağımsız Modüller(Information ve Interactive modülleri) içinde bulunuyor, Class modüllerinde değil. O yüzden bunları da tıpkı diğerleri gibi, önlerinde bir nesne olmadan kullanacağız.

Aşağıdaki görselden de fark edileceği üzere Modüllerin ikonu Class Modüller’inden farklıdır.(Bunlar başka dillerdeki Static/Shared sınıflara benzerler)



**IsArray, IsDate,IsEmpty,IsError,IsNull,IsNumeric,IsObjcet**: Bunlar parametre olarak aldıkları ifadelerin sırayla dizi mi, tarihsel ifade mi, boş mu, hata mı, null mı, sayısal mı obje mi olduğunu döndürür. Çoğunun kullanımı farklı yerlerde gösterildiği için burada ayrıca detaya girmiyorum.

**TypeName**: Değişkenin tipini verir.(String, Integer, Range vs). Genelde rutin kodlarımız içinde bulunmak yerine bir şeyleri kontrol ederken test amaçlı kullanılır.

**VarType**: Değişkenin tip numarasını verir. Alacağı değerler şöyledir. 0: empty, 1: null, 2: int, 3: long,....7: Date, 8: string, 9: object, 11: boolean, 12: Variant (sadece variant arraylerde), 8192: Array(normal değer + 8192). Bu da TypeName gibi genelde test amaçlı kullanılır.

Bunların hepsini bir arada ele alındığı bir örneği **github’taki(null\_empty\_nothing\_zls)** kod dosyasında görebilirsiniz.

**Environ**: İşletim sistemiyle ilgili bilgiler verir. Ya bir indeksle ya da ifade ile kullanılır. Tüm indekslerin değerlerini aşağıdaki kod ile bulabilirsiniz.

Sub env()

For i = 1 To 46

Debug.Print "i: " & i & ": " & Environ(i)

Next i

End Sub

'İfade kullanımı da şöyledir

Debug.Print Environ("USERNAME")

Ben şahsen bunlardan özellikle **COMPUTERNAME** ve **USERNAME'**i sıklıkla kullanma ihtiyacı duyuyorum. Mesela ortak kullanılan bir dosya var diyelim, bu bende açıldığında farklı bir işleve sahip olsun başkalarında açıldığında farklı işleve sahip olsun istiyorsam, bunu şöyle hallederim:

If Environ("USERNAME")=12345 Then 'kullanıcı adımın 12345 olduğunu varsayın

'diğer kodlar

Else

'Exit Sub

End Sub

Birden fazla bilgisayarla çalıyorsam ve sadece birinde açılan dosyada işlem olsun istersem de şu kod işimi görür:

If Environ("COMPUTERNAME")="A12345" Then 'kullanıcı adımın 12345 olduğunu varsayın

'diğer kodlar

Else 'B12345 ve L12345'te birşey yapmadan çıkar

'Exit Sub

End Sub

**CreateObject ve GetObject**: Bunlar objeler bölümünde ele alınıyor.

**SendKeys**: Klavyeden belli tuş veya tuş kombinasyonlarının basılması taklidini yapar.

SendKeys "^{F2}" 'Ctrl+F2 kombinasyonuna basılmış sayar

Tüm kullanılabilecek parametreler [MSDN’de](https://msdn.microsoft.com/en-us/vba/excel-vba/articles/application-sendkeys-method-excel)(https://msdn.microsoft.com/en-us/VBA/Excel-VBA/articles/application-sendkeys-method-Excel ) bulunmaktadır.

**Shell**: Belirli bir programı açar. Hesap makinesi, Windows Explorer en yaygın olanlarıdır. Mesela aşağıdaki örnekte diyelim ki bir dosyayı parçalara ayırdınız, dosyaların bölündüğü yer de Böl klasörü olsun. Kullanıcıya en son bir mesaj verip, ilgili klasörün açılması sağlanır.

Sub shellornek()

'kod bloğu

'

'

MsgBox "İşlem tamam. Dosyları görmek için tıkayınız"

Call Shell("explorer.exe" & " " & "C: \böl", vbNormalFocus)

'veya Shell "explorer.exe" & " " & "C: \böl", vbNormalFocus

End Sub

## Kullanıcı Tanımlı Fonksiyonlar(UDF)

### Neden?

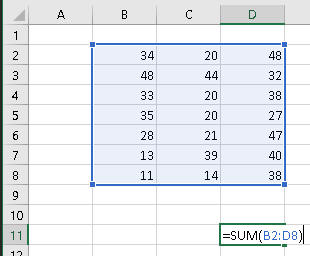
Bazı anlar olur ki "Ya bunun formülü nasıl yazılıyordu" dersiniz, o formülü daha önce yazmışsınızdır ama o kadar uzun bir formüldür ki tekrar hatırlamak biraz zamana mal olur, hele o parantezler yok mu, "ya bu parantezlerden hangisi fazla" diye düşünür durursunuz. Burada bir alternatif, daha önceden yaptığınız ve bir yerlere(!) kaydettiğiniz çalışmayı bulmak ve formülü kopyalamak, bir diğeri de kendi fonksiyonunuzu yani UDF(User Defined Functions) tanımlamak ve bunu her ihtiyacınız olduğunda çok kolay şekilde kullanmaktır.

UDF'in tek kullanım amacı bu değildir tabiki, bir diğer kulanım amacı ise, Excel'in mevcut fonksiyonlarıyla yapmanın zor olduğu hatta imkansız olduğu şeyleri bunlarla yapabilmektir. Evet, her ne kadar Excel'in çok geniş bir fonksiyon kütüphanesi olsa bile bunlar bazen yetersiz kalabilmektedir. Gerçi her yeni versiyonda bazı eksiklikler tamamlanmaktadır. Şahsen benim yazdığım en az bir düzine fonksiyon artık son durumda(bu sayfayı yazarken son versiyon 2016 idi) boşa çıkmıştır. Bunlara da yeri gedikçe bakacağız.

### Giriş

Şimdi UDF konusuna girerken olaylara bakışımızı biraz değiştirmekte, kendimizi Microsoft çalışanıymış gibi düşünmekte fayda var. Kendimize şunu soralım: Yerel fonksiyonları Excel mühendisleri nasıl hazırlamış olabilir?

Mesela **SUM** fonksiyonuna bakalım:



Bu yukarıdaki gibi bir fonksiyonu biz nasıl hazırlardık? Bunu önce sözlü dile getirelim. **Seçili alandaki tüm hücrelerin değerini tek tek topla**. Algoritmik hali de şöyledir.

* İlk hücreden toplamaya başla
* Sonraki hücreye geç, bir önceki değerle topla
* Bunu son hücreye kadar devam ettir

O zaman son minvalde kodumuz şöyle olacaktır:

Function Topla(alan As Range) As Double

Dim a As Range

Dim gecici As Double

For Each a In alan

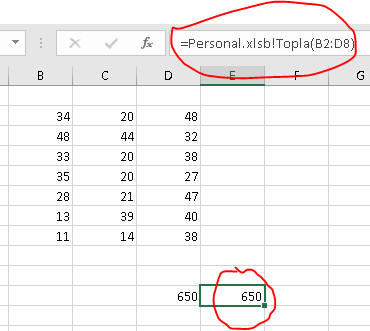
gecici = gecici + a.Value

Next a

Topla = gecici

End Function

Sonuca bakalım.



Ne yaptığımıza bir bakalım

* Fonksiyon ismini Function kelimesi ile belirttim, tipini de Double.(Tip belirtmezsem Variant olur)
* Fonksiyonuma parametre olacak olan "alan"ı veri tipi Range olacak şekilde belirttim.(Normalde yerel SUM fonksiyonu kendisine bir alan değil de adedi belirsiz olan sayısal değerleri kabul eder, biz basit olsun diye alan belirttik. Yerel SUM'ın yaptığı gibi de yapabilirdik, bunu biraz ileride ParamArray kısmında göreceğiz)
* Sonra geçici bir değişken tanımladım, hücreler içinde dolaşırken ara toplamı hep bu geçici değişkende tuttum.(Geçici değişkenler Function tanımlamalarında sıklıkla kullanılırlar)
* Sonra bir For Each döngüsü ile tüm hücrelerin arasında dolaşıp toplamı hesapladım
* En son da fonksiyon ismi olarak belirttiğim ifadeye(Topla) geçicinin değerini atadım

Fonksiyon tanımlamanın jenerik yapısı aşağıdaki gibidir:

Function fonksiyonadı(parametre1 As veritipi,parametre2 As veritipi,...) As DönüşTipi

'Gerekliyse değişken tanımlamaları

'Kod bloğu

'varsa geçicideğer

fonksiyonadı=geçicideğer

'geçici değer yoksa hesabı direk fonksiyon adı üzerinde yaparız

'fonksiyonadı=hesaplama kodları

End Function

Mesela aynı mantıkla **COUNT** fonksiyonunu düşünün. Bu fonksiyon bildiğiniz gibi sayı içeren hücreleri sayar. Hadi biz de aynı görevi gören bir UDF hazırlayalım. VBA, boş hücreleri de 0 gibi düşündüğü için bunları numerik sayar, o yüzden dolu hücrelere sayı içeriyor mu diye bakacağız.

Function NumerikSay(alan As Range)

Dim a As Range

Dim gecici As Double

For Each a In alan

If IsNumeric(a) And Not IsEmpty(a) Then gecici = gecici + 1

Next a

NumerikSay = gecici

End Function

Aynı mantıkla COUNTA fonksiyonunu düşünün. Bu fonksiyon bildiğiniz gibi içi dolu hücreleri sayar. Bunun da UDF versiyonunu hazırlayalım.

Function DolularıSay(alan As Range)

Dim a As Range

Dim gecici As Double

For Each a In alan

If Not IsEmpty(a) Then gecici = gecici + 1

Next a

DolularıSay= gecici

End Function

#### Performans

Hazırladığınız fonksiyonu binlerce satırdan oluşan listelerde kullanacaksanız mutlaka hem fonksiyonun parametrelerini hem de kodda kullanılacak tüm değişkenleri uygun veri tipinde tanımlayın. Buna rağmen fonksiyonunuz çok komplike ise yeterince hızlı çalışmayabilir. Yerel fonksiyonlarla yapmak çok zor değilse yerel fonksiyonları kullanmanız gerekebilir. Yerel fonksiyonlar sonuçta en temel seviyede çalışırlar. Kendi UDF'lerimiz ise bir seviye üstte çalışır, o yüzden temel seviyeye çevrilmeleri gerekir, en son da makine diline çevrilirler. Bu performans farkını test ettikten sonra görebilir, duruma göre karar verirsiniz.

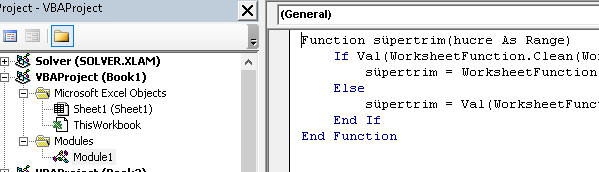
### Yerleşim ve Erişim

#### Yerleşim

##### Tekil kullanımlık UDF'ler

Yazdığımız fonksiyonları bazı durumlarda sadece ilgili dosya içinde çalışmasını isteriz, çünkü o dosyaya özgü çözüm sunarlar. Bunları başka bir yerde kullanmayacağımız için genel UDF'leri tutacağımız yerde bulundurmamıza gerek yoktur.

İlgili dosyada UDF yazmak için o dosyaya Modül eklemeli, modül sayfasına yazmalıyız. ThisWorkbook ve Sheet modülleri UDF'ler için uygun değildir.

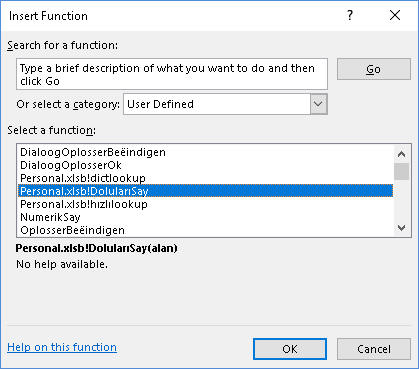


**Dikkat**: Tekil kullanımlık UDF'in kaydolduğu dosyayı **xlsm** veya **xlsb** uzantısıyla kaydetmek gerekmektedir.

##### Genel kullanım UDF'lerinin yerleşimi

Eğer kodlarınızı genele yaygın bir şekilde kullanmak istiyorsanız bunun için bir alternatif Personal.xlsb dosyasıdır ama bu genelde kötü bir alternatiftir. Normal Sub prosedürler için Personal.xlsb güzel bir adrestir ancak Function'lar için aynısını söyleyemeyiz. Zira bunlara ne **Insert Function** menüsündeki **User Defined** kategorisinden ulaşmak kolaydır, ne de bir hücreye doğrudan ismini yazarak ulaşabiliriz.

User Defined kategorisinden ulaşmak biraz karışıktır. Diyelim ki fonksiyon adı DolularıSay olsun. Bu fonksiyon listede **Personal.xlsb!DolularıSay**  şeklinde görünür ancak bunun için P harfine gidip önce Personla.xlsb'yi bulmanız gerekmez. UDF'ler, içinde bulundukları dosyadan bağımsız olarak kendi adlarına göre alfabetik sıralıdır. Dolularısay için de D harfiyle başlayan fonksiyonlara gitmeniz gerekir. Aşağıda görüldüğü üzere N harfiyle başlayan NumerikSay fonksiyonu P harfiyle başlayan Personal.xlsb!DolularıSay 'dan sonra geliyor, önce değil.

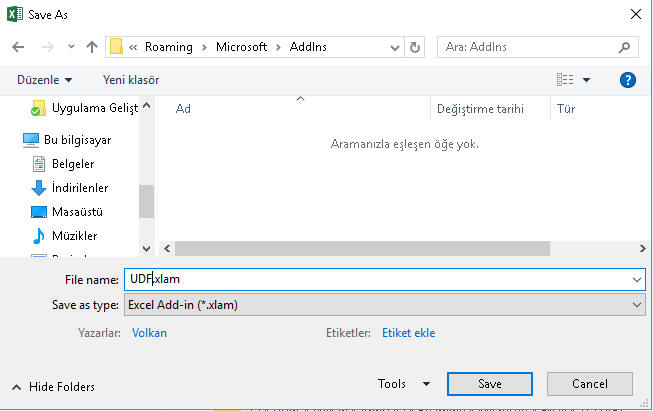


Bunu seçtiğimizde yukardaki ilk örnekte gördüğümüz gibi Excel içindeki görünümü **=Personal.xlsb!DolularıSay(B2: D7)** şeklinde olup biraz garip bir görünüme sahiptir. Normal bir dosya içindeki tekil kullanımlık UDF'lerde ise bu sorun yoktur.

Peki soru şu: UDF'imizi hem tekil kullanımlık UDF'teki gibi önünde dosya ismi olmadan kullanmak hem de genele yaygın kullanmak istiyorsak ne yapmalıyız? **Cevap**: UDF'lerimizi **Add-in** içinde kaydetmek.

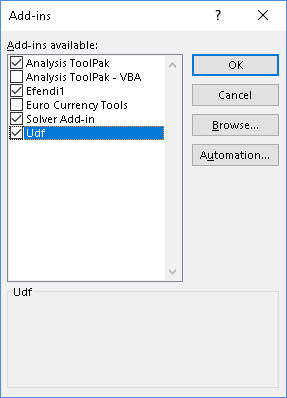
##### Add-in içinde yerleşim

Boş bir dosya açın, kaydet düğmesine basın, dosya tipini Excel Add-in olarak değiştirin.

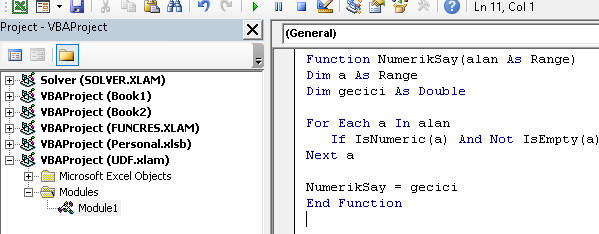


Otomatikman adresin yukardaki gibi değişmesi lazım. Sizdeki adres Excel versiyonunuza göre değişebilir. Bende şöyle: **C: \Users\Volkan\AppData\Roaming\Microsoft\AddIns\**

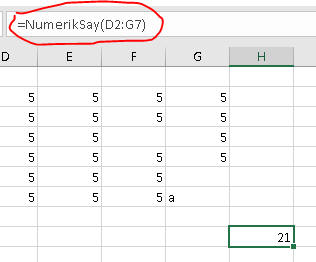
Bu add-in'i **Developer menüsü>Excel Add-in** menüsünden aşağıdaki gibi aktive edebilirsiniz. Bir kere aktive olduktan sonra Excel her açıldığında bu dosya da açılacaktır. Dosyanın kendisi görüntülenemez. Sadece VBE ortamında görünür. Yalnız bunlar Personal.xlsb'den biraz farklıdır. Personal.xlsb'yi istersek unhide yaparak görebiliriz ancak add-in dosyaları asla normal worbooklar gibi görüntülenemezler. Aşağıdaki görselde fark ettiyseniz Excel’le hazır gelen Anlaysis Toolpak ve Solver gibi add-inler de var. Bunların kullanımıyla ilgili detaylara yandaki ana menüde Excel altındaki Data menüsünden ulaşabilirsiniz.(NOT: Developer menüsündeki COM Add-in butonu VBA ile ilgili olmadığı için ona bu kitapta değinmeyeceğiz.)



Bundan sonrasında yapılması Gereken bu dosyaya VBE'de modül ekleyip UDF kodlarımızı oraya yazmaktır.



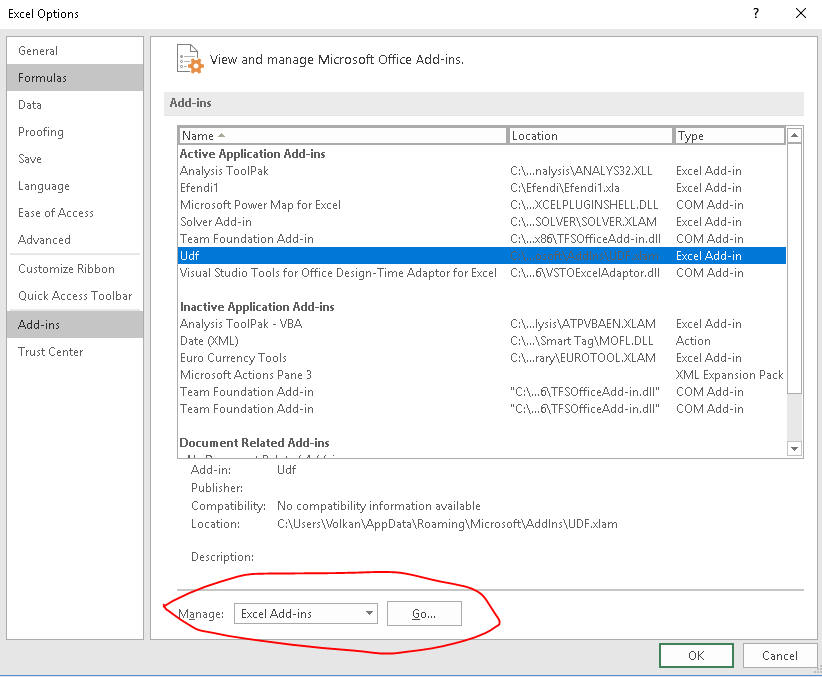
Gördüğünüz gibi artık UDF'in önünde dosya adı yok ve de herhangi bir dosyada kullanabiliyoruz.



#### Erişim

Add-in'lerinizi başka kişilerle de paylaşmaya karar verdiyseniz bunların nerede olduğuna ulaşmak için VBE'de Immediate Window'a şunu yazın: **?Application.UserLibraryPath.** Add-inlerinizin aslında istediğiniz yere kaydedebilirsiniz, ancak daha sonra aktive(veya pasifize) ederken  kolaylık olması adına bu adreste olması daha iyidir.

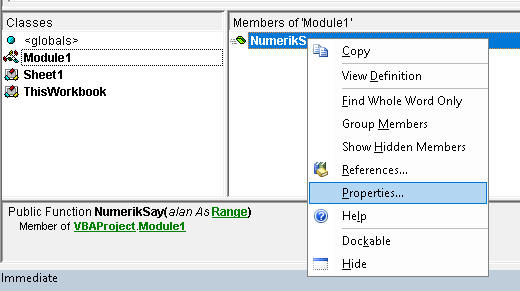
Aktivasyon işlemini yukarda göstermiştik. Bir diğer alternatif de, **File> Options>Add-ins** menüsündendir.



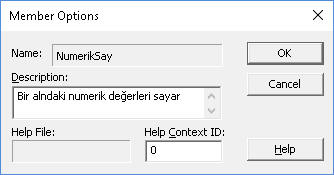
Son olarak, Add-inler sadece UDF depolamak için kullanılan araçlar değillerdir. Ayrıca menü yaratıp, bu menüye çeşitli düğmeler ve alt menülere ekleyerek Sub prosedürlerinizi yani makrolarınızı çalıştırmak için de bir arayüz sağlarlar. Bu konuya İleri Seviye işlemler bölümünde değineceğiz.

### UDF Açıklamaları ve Fonksiyon Kategorisi

Fonksiyonlara açıklama eklemek özellikle başkalarının kullanımı için faydalı olabilir. Bunun için aşağıdaki gibi bir ayarlama yapılabildiği gibi toplu tanımlama yapma imkanı da varır, ki bunun için tabiki yine VBA kodu kullanırız.



Çıkan kutuya direkt açıklama yazılır.



Toplu işlem için kullanacağımız kodda ise fonksiyonun ait olacağı kategoriyi de belirtebiliriz. Mesela NumerikSay için İstatistiki kategorisine koyabiliriz, ki yukardaki yöntemde bu yapılamamaktadır.

Mevcut kategoriler aşağıdakiler olup bunların dışında yeni kategoriler de yaratabiliriz.

'0 Kategori yok, All içinde görünür  
'1 Financial  
'2 Date & Time  
'3 Math & Trig  
'4 Statistical  
'5 Lookup & Reference  
'6 Database  
'7 Text  
'8 Logical  
'9 Information  
....  
'14 User Defined default  
'15 Engineering (Analysis Toolpak add-in'i kuruluysa kullanılabilir)

Aşağıdaki kod, NumerikSay fonksiyonunu Sayısal Fonksiyonlar isimli kategori içine atar, bu kategori mevut değilse bunu yaratır.

Sub UDFaçıklaması()

Application.MacroOptions \_

Macro: ="NumerikSay", \_

Description: ="Belli bir alandaki sayısal değerlerin toplam adedini verir", \_

Category: ="Sayısal Fonksiyonlar"

End Sub

Excel 2010 ile birlikte fonksiyonların argümanlarına da açıklama girme imkanı gelmiştir. MacroOptions metoduna **ArgumentDescriptions** parametresini ekleyerek bu işi hallederiz.(Normalde VBA ölü bir dildir, yani yeni güncellemeler, özellikler eklenmez ancak ender de olsa böyle küçük iyileştirmeler yapılmakta)

Önemli bir nokta var ki, o da bu girdiğiniz açıklamalar sadece sizde geçerli olur. UDF'erinizi bir add-in haline getirip gönderirseniz ve onlarda da görünmesini isterseniz bu tanım ekleme kodlarını Workbook\_Open içine yazmakta fayda var, ki kişinin add-ini yüklenir yüklenmez açıklama kodu da çalışsın. UDF'lerin açıklamaları ve parametrelerin açıklamalarıyla birlikte kodunuz çok uzayacak gibi olursa bunları bir text dosyasından veya Excel dosyasından okutabilirsiniz.(Biraz aşağıda örneği var)

Şimdi tek bir fonksiyon için nasıl yapıyoruz ona bakalım.

Sub UDFaçıklaması()

Dim fonkAd As String

Dim fonkTanım As String

Dim fonkKategori As String

Dim argumanAdedi As Integer

Dim argumanlar() As String

argumanAdedi = 1

ReDim argumanlar(1 To argumanAdedi)

fonkAd = "NumerikSay"

fonkTanım = "Belli bir alandaki sayısal değerlerin toplam adedini verir"

fonkKategori = 4

argumanlar(1) = "Toplanacak sayıların olduğu alanı seçin."

Application.MacroOptions \_

Macro: =fonkAd, \_

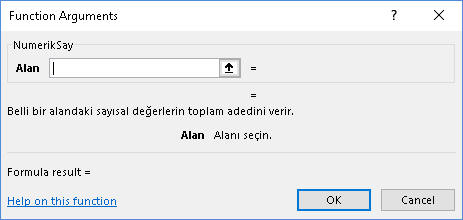
Description: =fonkTanım, \_

Category: =fonkKategori , \_

ArgumentDescriptions: =argumanlar

End Sub

Herhangi bir hücreye fonksiyonumuzu yazdığımızda aşağıdaki gibi görünür:



Aşağıda da Workbook\_Open makrosuna yazılı olan kod var. Bu kod ile, fonksiyon açıklamalarını bir Excel dosyasından okuyoruz.

Private Sub Workbook\_Open()

Application.ScreenUpdating = False

Workbooks.Open "C: \makrolar\kılavuz.xlsx", UpdateLinks: =0 'herkes dosyayı buraya koymalı

Workbooks("kılavuz.xlsx").Sheets("UDFDesc").Select

Call macrodesc

Workbooks("kılavuz.xlsx").Close savechanges: =False

Application.ScreenUpdating = True

End Sub

'bu da çaprılan macrodesc prosedürü

Sub macrodesc()

Dim fonkAd As String

Dim fonkTanım As String

Dim fonkKategori As String

Dim argumanAdedi As Integer

Dim argumanlar() As String

Dim i As Integer

Range("A2").Select

Do

fonkAd = ActiveCell.Value2

fonkTanım = ActiveCell.Offset(0, 1).Value2

fonkKategori = ActiveCell.Offset(0, 2).Value2

argumanAdedi = Range(ActiveCell, ActiveCell.End(xlToRight)).Cells.Count - 3

ReDim argumanlar(1 To argumanAdedi)

For i = 1 To argumanAdedi

argumanlar(i) = ActiveCell.Offset(0, i + 2).Value2

Next i

Application.MacroOptions \_

Macro: =fonkAd, \_

Description: =fonkTanım, \_

Category: =fonkKategori, \_

ArgumentDescriptions: =argumanlar

Loop Until ActiveCell.Value = ""

End Sub

### Diğer Hususlar

#### ParamArray ile belirsiz sayıda parametre temini

Yukarıda Excel’in yerel SUM fonksiyonunu, kaynağı bir alan olacak şekilde tasarlamış ve demiştik ki, aynı yerel SUM gibi sayısı belirsiz olan elemanları içerecek şekilde de yapabiliriz. İşte bunun yolu **ParamArray** ifadesini kullanmaktır. Önce örneği yapalım sonra açıklayalım.

Function ToplaParamarray(ParamArray sayılar()) As Double

Dim a As Variant

Dim gecici As Double

For Each a In sayılar

gecici = gecici + a

Next a

ToplaParamarray = gecici

End Function

ParamArray ifadesinden sonra bu sayısı belirli olmayan elemanları içerecek bir dizi adını gireriz. Bu Variant tipli bir değişken olmak zorundadır, yani başka bir veri tipiyle tanımlayamayız. Zaten mantıklısı da budur. Mesela Toplam örneğinde hem 0,25 gibi double tipindeki sayıları hem de 5 gibi IntegerlarWı input olarak alabiliriz, böyle bir durumda Variant’tan başka çare yoktur.

Diğer kısıtlamalar şöyle;

* Fonksiyonun parametre listesinde başka parametreler de varsa Paramarray sonucu parametre olmalıdır. Ör: ToplamınXinciÜssü şeklindeki fonksiyon şöyle tanımlanırdı. **Function ToplamınXinciÜssü(üs As Integer,ParamArray sayılar())**
* Bir fonksiyonda sadece 1 tane paramarray tanımlanabilir
* Bir fonksiyonda Optional ve Paramarray parametrelerden yalnız biri kullanılabilir.
* İlgili dizinin tabanı 0'dır. İsterseniz Option Base ile genel dizi tabanını 1 yapmış olun fark etmez. ForEach yerine klasik for kullanılacaksa Lbound(dizi) to Ubound(dizi) şeklinde kullanılabilir.

NOT: Bu yukardaki örnekte bir alan seçimi yapılamayacağına dikkat edin; tek tek sayı temin etmek zorundasınız. Excel'in SUM fonksiyonu ise hem alan kabul edebiliyor hem de tek tek sayılar kabul edebiliyor. SUM'ın bunu nasıl yaptığını düşünün ve her iki versiyonu da kapsayacak bir fonksiyonu yazmayı deneyin.

#### Optional ile opsiyonel parametre temini

Bazen girdiğimiz parametrelerin sık kullanılan değerini biz baştan gireriz ve son kullanıcıya bunu değiştirme imkanı veririz. Bu, yerel Excel fonksiyonlarında da var olan default(varsayılan) değerlerle aynı şeydir. Mesela VLOOKUP fonksiyonunun son parametresi opsiyonel olup default değeri TRUE(1)'dur, yani girilmezse TRUE(1) algılanır.

İşte biz de bu opsiyonel parametreyi Optional ifadesi ile sağlarız, istersek varsayılan değer de girebiliriz, girmezsek de bunun girilip girilmediğini IsMissing fonksiyonu ile test ederiz. Ancak IsMissing sorgulaması yapabilmek için ilgili parametrenin data tipi Variant olarak girilmelidir. (ParamArrayda datatipi Variant olmak zorundayken Opsiyonellerde ise tavsiyedir, olur da IsMissing ile sorgularız diye, ama zorunlu değildir). Bu değer Variant değilse ve kullanıcı tarafından girilmezse bunlara default değerleri atanır; String için "", sayısal tipler için 0, boolean için false.

Dikkat edilmesi gereken diğer hususlar şöyledir:

* Bunlar da ParamArray gibi son parametre olarak girilmelidir
* ParamArray'de belirttiğimiz gibi aynı anda hem ParamArray hem Optional ifadesi kullnılamaz.
* Birden fazla Optional ifadesi kullanılabilir

##### Opsiyonel Örnek 1

Aşağıda varsayılan değerin girilmiş olduğu bir örnek bulunuyor. Bu örnekte bir hücredeki metni kelimelere ayırıyoruz. Normal bir cümlede ayraç boşluk olacağı için ayracı girmeye gerek yok, o yüzden default olarak " " atadım. Ama kullanıcı isterse kelimeleri farklı bir ayraçla da ayırabilir. Mesela - ile ayrılmış kelimeler varsa ayraç olarak - kullanılabilir. Fonksiyonda Split fonksiyonu kullanarak kelimeleri parçalıyor ve bir diziye atıyorum. Kaçıncı parametresi ile de istediğim kelimeyi elde ediyorum.

Function kelimesec(hucre As Range, kaçıncı As Byte, Optional ayrac As String = " ")

Dim kelimeler As Variant

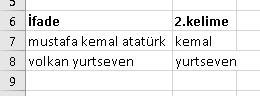
kelimeler = Split(hucre.Value2, ayrac)

kelimesec = kelimeler(kaçıncı - 1)

End Function

Örnek kullanım  aşağıdaki gibidir. B7'deki formül şöyle:

=kelimesec(A7,2) //son parametreyi girmedim



##### Opsiyonel Örnek 2

Şimdi üç tane opsiyonel değişkeni olan bir fonksiyonumuz var. Bunlardan birinin varsayılan değeri girilmiş, birinin girilmemiş, biri de Variant olarak tanımlanmış.

Bu örnek bankacılık dünyasından bir örnek olacak. Mevduat/Kredi gibi hacimsel bir büyüklük ile bu hacimden ne kadar kar elde ettiğimizi gösteren spread bilgisini zorunlu olarak giriyoruz. Bu fonksiyonu iki şekilde kullanabiliyoruz. Eğer belirli bir alan seçilmezse şube bazında NFG(Net Faiz Geliri) hesaplayacağız, ancak ilgili alanı seçersek o alandaki MT(Müşteri Temsilcisi) adedi başına NFG hesaplanmış olacak. İşte bu alan bilgisini Variant olarak girdim, ki IsMissing ile alanın girilip girilmediğini sorgulayabileyim. İkinci opsiyonel seçenek Kur bilgisi olup, ilgili hacim türünün TL mi yoksa döviz mi olduğuna göre değer alacak. Varsayılan olarak döviz tipinin TL olduğunu düşünerek değeri 1 girdim, ancak kullanıcı isterse farklı bir kur değeri girebilir. Son değişken ise Para birimi olup, varsayılan değer girilmemiştir. Kullanıcı isterse TL, USD gibi değerler girebilir, eğer girmezse Stringler’in varsayılan değeri olan "" atanacak olup herhangi bir para birimi yazmayacaktır.

Son olarak, fonksiyon için dönüş tipi belirtmedim, yani Variant olacak. Zira Birim parametresi girilirse sonuç String, girilmezse double olacak,  yani ikisini de kapsayan  bir tip olmalı, ki bu da Variant oluyor.

Function NFGHesapla(hacim As Double, spread As Double, Optional alan As Variant, Optional Kur As Double = 1, Optional Birim As String)

Dim adet As Integer

If IsMissing(alan) = True Then

NFGHesapla = (hacim \* spread \* Kur / 1200) & Birim

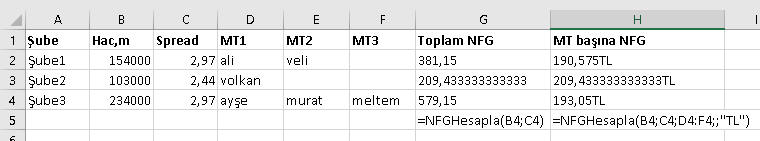
Else

NFGHesapla = (hacim \* spread \* Kur / 1200) / WorksheetFunction.CountA(alan) & Birim

End If

End Function

Kullanım şekli aşağıdaki gibidir. G kolonunda şube başına ve para birimsiz versiyonunu görürken, H kolonunda MT başına ve para birimli versiyonunu görüyoruz.



**NOT**: Bu örnekle, aslında modern programlama dillerinde olan ancak VBA'de olmayan "method overloading" kavramını (tam olarak olmasa da) bir nevi taklit etmiş olduk. Yani bir metodun ismi aynı olup farklı parametre veya dönüş tipi alıyorsa bu işleme method overloading denir. Biz de buna benzer birşey yapmış olduk. Hem alan tipini seçip seçmemeye göre hem de sonuna para birimi koyup koymamaya göre 4 farklı kullanım şekli sunduk.

#### Volatile

Yazdığımız fonksiyon, eğer sayfada bir güncelleme olduğunda bundan etkileniyorsa anında güncellenmez. Volatile ile bu güncellemeyi anlık olarak yapmış oluruz. Ancak üstatlar der ki, UDF'inizi öyle bir hazırlayın ki bu fonksiyonu kullanma ihtiyacınız hiç olmasın.

İnternette bu konuyla ilgili çeşitli tartışmalar da yapılmış olduğunu görebilirsiniz, İngilizceniz varsa bakabilirsiniz.

Özet  tavsiyem: Araştırmalarınızda bu ifadeyi görürseniz ne olduğunu bilin ama bunu kullanmayın. UDF'inizde bunun kullanımına gerek olmayacak şekilde tüm parametreleri dahil edin.(Örnek vererek kafanızı da karıştırmak istemiyorum)

#### Default Değerler

Fonksiyonların da tıpkı değişkenler gibi default değerleri bulunur. Örneğin, Boolean tipli bir fonksiyon için bir koşul bloğu içinde sadece True ataması yapıyorsanız ve bu koşul sağlanmıyorsa, siz açıkça Else bloğu içinde False ataması yapmasanız bile fonksiyon False döndürür. (Bu açıklama hem Excel UDF'leri hem VBA UDF'leri için geçerlidir)

Aşağıdaki örneği inceleyelim.

Function aysonumu(tarih As Date) As Boolean

If Month(tarih) <> Month(tarih + 1) Then

aysonumu = True

Else

aysonumu=False

End If

End Function

Bu fonksiyonu daha kısa bir şekilde aşağıdaki gibi yazabiliriz. Zira fonksiyona ilk girildiği anda, fonksiyon booelan tipli bir fonksiyon olduğu için giriş anında default değeriyle yani False olarak hayatına başlar. Sonrasında kendisine bir atama olmazsa da bu değerini korur, yani False döndürür.

Function aysonumu(tarih As Date) As Boolean

If Month(tarih) <> Month(tarih + 1) Then aysonumu = True

End Function

NOT: Bu fonksiyon hem VBA hem Excel UDF'i olarak kullanılabilen bir fonksiyondur.

### Çeşitli örnekler

#### Süpercombine ile hücreleri birleştirin

Bu fonksiyon, ardışık bir hücre grubunu belirli bir işaret/ayraç ile birleştirir. 2016 ile gelen TEXTJOIN ve CONCAT fonksiyonlarıyla gereksiz hale gelmiştir ancak eski Excel versiyonlarını kullananlar için hala geçerlidir.

Function Süpercombine(Hucre\_grubu As Range, isaret As String)

x = vbNullString

For Each k In Hucre\_grubu

If Not IsEmpty(k) Then

x = x & k.Value & isaret

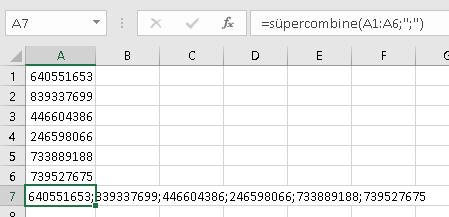
End If

Next k

Süpercombine= Mid(x, 1, Len(x) - 1)

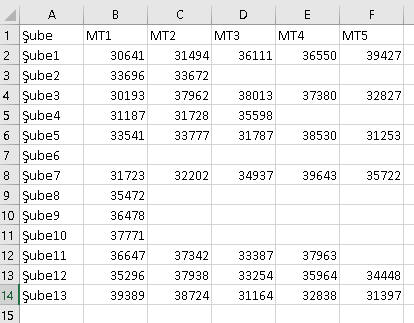
End Function

Örnek kullanım aşağıdaki gibidir. Müşteri numaraları ";" işareti ile birleştirilmiştir.



#### Süperlookup ile kayan lookup işlemi yapın

Sheet2'de şöyle bir listemiz var,



Sheet1'de bulunan aşağıdaki listeye(sadece A kolonu olduğunu düşünün) lookup çekmek isteseydik şu formülü yazardık.(veya yardımcı bir index satırı kullanırdık, ama şık olanı bu aşağıdakidir)

=VLOOKUP($A2;Sheet2!$A: $F;MATCH(B$1;Sheet2!$A$1: $F$1;0);0)

İşte bizim süperlookup fonksiyonumuz bizi bu uzun formülden kurtarmış oluyor.

Function süperlookup(alan As Range, sütun As Range, aranan As Range)

On Error GoTo hata

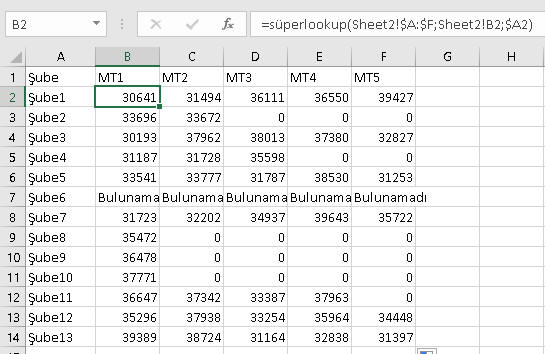
süperlookup = alan.Columns(1).Find(aranan, lookat: =xlWhole).Offset(0, sütun.Column - alan.Columns(1).Column).Value

Exit Function

hata:

süperlookup = "Bulunamadı"

End Function



#### Çok eşleşmeli lookup

Bazen, lookup yaptığımız alanda birden fazla eşleşme varsa hepsinin sonucunun gelmesini isteriz. Normal Excel fonksiyonları ile bunu yapmak çok olası değil. Aslında oldukça dolambaçlı yollardan bunu yapanlar var ama ben buna girmek istemiyorum. Bunun yerine aşağıdaki UDF'i kullanmak çok daha kolaydır.

Function çok\_eşleşmeli\_vlookup(aranan As Range, alan As Range, kaçıncıkolon As Integer, Optional distinctmi As Boolean = True)

Dim a As Range

Dim dict As New Scripting.Dictionary 'Reference olarak eklenmiş olmalı, ekli değilse Late binding olarak yaratılabilir

If alan.Columns(1).Rows.Count = 1048576 Then

Set alan = Range(alan(1, 1), alan(1, 1).End(xlDown))

End If

For Each a In alan.Resize(, 1)

If Not dict.Exists(a.Value) Then

dict.Add a.Value, a.Offset(0, kaçıncıkolon - 1).Value

Else

geçici = dict(a.Value)

dict.Remove (a.Value)

x = a.Offset(0, kaçıncıkolon - 1).Value

If distinctmi = True Then

dict.Add a.Value, geçici & IIf(InStr(1, geçici, x, vbTextCompare) > 0, "", ";" & x)

Else

dict.Add a.Value, geçici & ";" & a.Offset(0, kaçıncıkolon - 1).Value

End If

End If

Next a

çok\_eşleşmeli\_vlookup = dict(aranan.Value)

End Function

NOT: Yukarıda sonunda x bulunan satırı kaldırıp aşağıdaki 2 satırı eklersek, eşleşen değerleri bize distinct(benzersiz) olarak getirir.

x = a.Offset(0, kaçıncıkolon - 1).Value

dict.Add a.Value, geçici & IIf(InStr(1, geçici, x, vbTextCompare) > 0, "", ";" & x)

Aslında kodu daha kullanışlı hale getirebiliriz, bunun için bir parametre daha ekleyelim.

Function çok\_eşleşmeli\_vlookup(aranan As Range, alan As Range, kaçıncıkolon As Integer, Optional distinctmi As Boolean = True)

Dim a As Range

Dim dict As New Scripting.Dictionary 'Reference olarak eklenmiş olmalı, ekli değilse Late binding olarak yaratılabilir

If alan.Columns(1).Rows.Count = 1048576 Then

Set alan = Range(alan(1, 1), alan(1, 1).End(xlDown))

End If

For Each a In alan.Resize(, 1)

If Not dict.Exists(a.Value) Then

dict.Add a.Value, a.Offset(0, kaçıncıkolon - 1).Value

Else

geçici = dict(a.Value)

dict.Remove (a.Value)

x = a.Offset(0, kaçıncıkolon - 1).Value

If distinctmi = True Then

dict.Add a.Value, geçici & IIf(InStr(1, geçici, x, vbTextCompare) > 0, "", ";" & x)

Else

dict.Add a.Value, geçici & ";" & a.Offset(0, kaçıncıkolon - 1).Value

End If

End If

Next a

çok\_eşleşmeli\_vlookup = dict(aranan.Value)

End Function

#### Gün no’sundan(1-7) gün adını veren fonksiyon

Normalde bunu aşağıdaki formülle yazabiliriz ama TEXT fonksiyonu bazen karışık olabilmekte ve bazı kişilerin yıldızı bu formülle barışık olmayabiliyor.

=TEXT(A1+1;"gggg") //A1'de 2 yazıyorsa salı döndürür. 1 ekleme sebebi günlerin Amerikan formatına göre Pazardan başlamasıdır

Gün UDF'i çok daha basit gibi duruyor.

Function gün(hucre As Range)

Select Case hucre.Value

Case 1

gün = "Pazartesi"

Case 2

gün = "Salı"

Case 3

gün = "Çarşamba"

Case 4

gün = "Perşembe"

Case 5

gün = "Cuma"

Case 6

gün = "Cumartesi"

Case 7

gün = "Pazar"

Case Else

gün = "\*\*\*\*\*Hata\*\*\*\*\*,Ben sadece 1 ve 7 arasındaki günler için çalışırım"

End Select

End function

Hücredeki formül:

=gün(A1)

#### Uçhariçortalma ile bir veri kümesindeki uç değerleri elimine edin

Diyelim ki hedefleme veya tahminleme yapıyorsunuz. Hedef verdiğiniz birimin (bölge/şube/bayi/mağaza/v.s) son 1 yıllık satışlarına bakacak ve ortalama alacaksınız, ama bir seferlik aşırı düşük/büyük rakamların da ortalamayı etkilemesini istemiyorsunuz. O yüzden en küçük ve en büyük rakamları çıkartıp kalan 10 ayın ortalamasını almak istiyorsunuz. Bunun için normalde şu Excel fonksiyonunu yazardınız.

=(SUM(B2: B13)-SMALL(B2: B13;1)-LARGE(B2: B13;1))/(COUNT(B2: B13)-2)

Ama biz şu UDF ile işimizi kısaca hallederiz.

Function uçhariçort(alan As Range, Uç As Variant)

Dim aratoplam As Double

Dim enbüyükler As Double

Dim enküçükler As Double

For i = 1 To Uç

enbüyükler = enbüyükler + WorksheetFunction.Large(alan, i)

Next i

For i = 1 To Uç

enküçkler = enküçkler + WorksheetFunction.Small(alan, i)

Next i

aratoplam = WorksheetFunction.Sum(alan) - enbüyükler - enküçkler

uçhariçort = aratoplam / (alan.Count - Uç \* 2)

End Function

UDF'imizi **=uçhariçort(B2: B13;1)** şeklinde kullanınca sonuç aşağıdaki gibidir.



Sona girdiğimiz 1 parametresi en küçük/büyük 1 değeri hariç tutmamızı sağlar. Eğer 2’şer küçük/büyük hariç tutmak istersek buraya 2 gireriz, bu sefer kalan 8 ayın ortalaması alınmış olur.

#### Süpertrim ile TRIM yapılamayan karakterleri de silin

Bu fonksiyon Outlook gibi farklı bir ortamdan kopyalanarak alınan rakamların önündeki ve sonundaki boşlukları yok etmek için kullanılır. Genelde herkes Excel'in TRIM fonksiyonunu bilir ve bununla bu boşlukları kaldırabileceğini düşünür ancak aslında bunlar her zaman bildiğimiz boşluk olmayabiliyor, o yüzden TRIM yetersiz kalıyor. Metinsel fonksiyonlarda bu durumu açıklamıştık. Detayı oradan okuyabilirsiniz. İşte orada verilen formülü yazmak yerine aşağıdaki UDF oldukça iş görecektir

Function süpertrim(hucre As Range)

süpertrim = Val(WorksheetFunction.Trim(WorksheetFunction.Substitute(hucre, Chr(160), "")))

End Function

#### Sapmaoranı ile veri kümesindeki sapmayı bulun

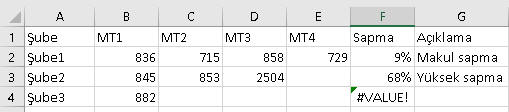
Bu fonksiyon ile incelediğiniz küme içinde bir dengesizlik var mı, rakamlar birbirinden çok mu farklı yoksa makul sınırlarda mı sapma var, bunu görmüş olursunuz. Yapılan işlem aslında kümenin standart sapmasının ortalamaya bölümüdür. Bu oran ne kadar küçükse o kadar dengelidir, yani rakamlar birbirine yakındır, ne kadar büyükse sapma miktarı o kadar yüksektir. Tek elemanlı bir kümede hata alınır, siz eleman sayısının tek olup olmamasına bakarak fonksiyonu iyileştirin.

Function sapmaoranı(alan As Range)

sapmaoranı = WorksheetFunction.StDev\_S(alan) / WorksheetFunction.Average(alan)

End Function

Aşağıdaki örnekte şubelerdeki müşteri temsilcilerine bağlanmış müşteri adetleri görülmekte. 2 nolu şubede portföyler arasında dengesizlik söz konusu ve müşterilerin yeniden dağıtılmasında fayda var diyebiliriz.



## VBA için UDF

### Giriş

Excel'de kullanım için hazırladığımız UDF'lerden başka VBA prosedürleri içinde çalışıp bir sonuç döndüren fonksiyonlar da vardır. Bunlar da tıpkı VBA'in yerel fonksiyonları gibidirler. Genelde belirli bir(bazen birkaç) sonuç döndürmek üzere hazırlanırlar. Ender olarak bazı kaynaklarda sonuç döndürmeyen versiyonların kullanıldığını da görebilirsiniz ama bence bu tamamen yanış kullanımdır. Sonuç döndürmeyen bir iş istiyorsak bunu Function olarak değil Sub olarak hazırlamalıyız.

Aşağıda bu yanlış kullanıma bir örnek bulunmaktadır. Bu, doğru çalışan tamamen düzgün bir koddur, ancak dediğim gibi Functionların amacı bu değildir, olmamalıdır.

Sub YanlışFuncOrnek()

Call YanlışFunc

End Sub

Function YanlışFunc()

[A1] = 12

End Function

Fonksiyonların amacı nedir diyecek olursak, iki temel amacı vardır.

Ana Sub prosedürünüzün çok uzaması durumunda bunu belli yerlerde kesip fonksiyon olarak yazmak ve ana koddan bu fonksiyonu çağırmak

Bir diğeri de, belirli bir işi farklı zamanlarda, farklı kodlar içinde sürekli yapmak durumunda kalıyorsanız, bunu bir kez fonksiyon olarak yazarsınız, sonra her yerden bu fonksiyonu çağırırsınız. Böylece gereksiz tekrardan kurtulmuş olursunuz. Üstelik fonksiyonda küçük bir değişklik gerekse bile sadece bir kere yapılması yeterli olacaktır.

Son olarak belirtmek istediğim bir husus var, VBA kullanımı için yazdığımız functionlara genelde kimse UDF demez, UDF denince akla genelde Excel için yazdığımız UDF'ler gelir, ama aslında teknik olarak bakıldığında VBA içi kullanım amacıyla yazdığımız functionlar da UDF'tir. Zira VBA'de yerel olarak gelmediği için biz bunları kendimiz tanımlarız.

#### Tanımlama

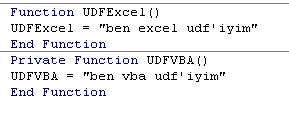
Bunların tanımlanması da tıpkı Excel UDF tanımı gibidir.

Function fonksiyonadı(Pamaratre1 As datatipi,...) As DönüşTipi

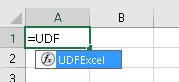
.......

End Function

Sadece kullanırken Excel içinde değil de VBA içinde kullanıyoruz. Bu arada Excel'den böyle bir fonksiyonu yazmaya çalıştığınızda yazabilirsiniz, buna bir engel yoktur. O yüzden bunların görünmesini istemiyorsanız **Private** olarak tanımlamanız gerekir. Aşağıdaki örneklerden bunu görebilirsiniz.



Aşağıda görüldüğü üzere Excel’de bir hücreye **=UDF** yazınca sadece UDFExcel çıktı.



Tabi ilgili fonksiyonu Private tanımlamanın da bir dezavantajı var, onu prosedürlerin erişim seviyesi bölümünde görmüştük. Özetle bu fonksiyonlara sadece bulunduğu modülden erişilebilir. Eğer bu bir sıkıntı olmayacaksa private tanımlayın, sıkıntı olacaksa public tanımlayın. (Tabi bu arada bunları bir Add-in içine yazdığınız senaryoya göre söylüyorum. Add-in dışında başka bir yere mesela Personal.xlsb içine yazılanlarda zaten fonksiyon ismini Excel’e yazınca hemen çıkmıyordu, başında **Personal.xlsb!** olması gerekiyordu veya Insert function deyip seçmek gerekiyordu)

Zaten genel tavsiyem şudur: Özellikle Excel UDF'lerinizi başkalarına da dağıtıyorsanız UDF.xlam dosyası içine sadece Excel UDFlerini koyun, kendi kullanımınız için oluşturduğunuz VBA UDF'lerini ise Personal.xlsb içine koyun.

#### Kullanım

Şimdi bir örnekle konuyu pekiştirelim.

Diyelim ki ana makronuzda(veya birçok makronuzda) öyle bir yer geliyor ki, o anda ilgili alanda filtre uygulanmış mı uygulanmamış mı bunu kontrol etmek ve sonrasında duruma göre de bir işlem yaparak ilerlemek istiyorsunuz. Bunun için aşağıdaki gibi bir Function yazarız. Bunu Personal.xlsb içine yazdırıyorum.

Function filter\_kontrol(ws As Worksheet) As Byte

If ws.AutoFilterMode = True Then

If ws.FilterMode = False Then

filter\_kontrol = 1 'filtre açık ama criter yok

Else

filter\_kontrol = 2 'filtre açık ve criter var

End If

Else

filter\_kontrol = 0

End If

End Function

Sub filtrekullan()

'örnek kullanım şekli

If Application.Run("PERSONAL.xlsb!filter\_kontrol", ActiveSheet) = 2 Then 'filtre açık ve criter var

ActiveSheet.ShowAllData

ElseIf Application.Run("PERSONAL.xlsb!filter\_kontrol", ActiveSheet) = 0 Then 'filtre uygulanmamış

Selection.AutoFilter

'diğer duurmlarda yani 1, yani filtre açık ama criter yok, bişey yapmaya gerek yok

End If

End Sub

#### Dönüş Değeri

Genelde fonksiyonların sadece bir adet dönüş değeri olur. Ancak bazen bir fonksiyonu çağırdığımızda birden fazla dönüş değeri isteyebiliriz. Bunun için çeşitli alternatifler olmakla birlikte ben ikisinden bahsedeceğim.

* ByRef ile çoklu değer döndürme (**Temeller>İleri Terminoloji’de** anlattık)
* Diziler ve Collectionlar aracılığı ile çoklu değer döndürme(Kendi konu sayfalarında anlattık)

Konuyu tamamlamak adına İleri Terminoloji sayfasındaki Argüman ve Parametre ile Prosedürlere Erişim maddelerini gözden geçirmenizi tavsiye ederim.

#### Yinelemeli(Recursive) fonksiyonlar

Bir fonksiyonun içinde kendisine başvuru yapmamız da mümkündür. Buna recursive başvuru denir. Tabi bunu sonsuza kadar değil de belirli bir şart sağlanana(mesela bir limite ulaşmak gibi) kadar kurgulamak gerekir, yoksa kodumuz kısır döngüye girer.

Recursive fonksiyonlara klasik örnek matematikteki faktöriyel hesabıdır. Aşağıdaki örnekte bu hesabı bulabilirsiniz. Aldığı parametre 1 olana kadar kendisini -1 değeriyle çağırıp parametre ile çarpıyor. Parametrenin değeri 1'e ulaştığında(indiğinde) yineleme sonlanmış oluyor.

Function faktoriyel(ByVal n As Integer) As Integer

If n <= 1 Then

faktoriyel= 1

Else

faktoriyel= faktoriyel(n - 1) \* n

End If

End Function

'---------------

Sub faktoriyel\_yaz()

Debug.Print faktoriyel(5) '120

End Sub

Bu fonksiyonun ele alınışı aşama aşama şöyledir.

* Önce 5 parametresini verdik, sonuç=faktoriyel(4)\*5
* Şimdi parametre olarak 4 gitmiş oldu, sonuç=**faktoriyel(3)\*4**\*5(Koyu kısım aslında bir üst satırın açılmış hali
* Sonra 3 gider, sonuç=**faktoriyel(2)\*3**\*4\*5(Koyu kısım yine bir üsttekinin açılımı)
* Sonra 2, sonuç=**faktoriyel(1)\*2**\*3\*4\*5
* Son olarak 1 gider, sonuç**=faktoriyel(1)\*2**\*3\*4\*5
* Nihai sonuç=1\*2\*3\*4\*5=120

Bunun dışında parent-child tarzı oluşumlarda(Ör: klasör-dosya veya HTML tag'i ve alt tag'i)  da çok sık kullanılır. Bu kullanım şekline bir örnek de aşağıda bulunmaktadır.

'ana prosedür

Sub recursive\_fulldosya()

Dim fso As New Scripting.FileSystemObject

Dim anaklasorStr As String

anaklasorStr = "C: \windows"

Recursiveİlerle fso.GetFolder(anaklasorStr)

End Sub

'recursive prosedür

Sub Recursiveİlerle(kls As Variant) 'variant çünkü ilk girereken Folder sonra Folders olacak

Dim altKlasorler As Variant

Dim dosya As file

Dim i As Integer

On Error Resume Next 'erişim izni olmayan yerlerde hata almasın diye

For Each altKlasorler In kls.SubFolders

Recursiveİlerle altKlasorler 'burada recursive olarak başvuru var

Next

i = 1

For Each dosya In kls.Files

ActiveCell(i, 1).Value = dosya.ParentFolder

ActiveCell(i, 1).Offset(0, 1).Value = dosya.Name

ActiveCell(i, 1).Offset(0, 2).Value = dosya.Size

i = i + 1

Next

End Sub

### Çeşitli Örnekler

#### Excel hücre grubunu mail body'sine koymak

Bir başka örnek de meşhur RonDeBruin'in belirli bir Excel hücre grubunu mail bodysi haline getiren fonksiyondur(<https://www.rondebruin.nl/win/s1/outlook/bmail2.htm>), efsanedir, hayat kurtarıcıdır. Kod aşağıdaki gibidir:

Function RangetoHTML(rng As Range)

' Changed by Ron de Bruin 28-Oct-2006

' Working in Office 2000-2013

Dim fso As Object

Dim ts As Object

Dim TempFile As String

Dim TempWB As Workbook

TempFile = Environ$("temp") & "/" & Format(Now, "dd-mm-yy h-mm-ss") & ".htm"

'Copy the range and create a new workbook to past the data in

rng.Copy

Set TempWB = Workbooks.Add(1)

With TempWB.Sheets(1)

.Cells(1).PasteSpecial Paste: =8

.Cells(1).PasteSpecial xlPasteValues, , False, False

.Cells(1).PasteSpecial xlPasteFormats, , False, False

.Cells(1).Select

Application.CutCopyMode = False

On Error Resume Next

.DrawingObjects.visible = True

.DrawingObjects.Delete

On Error GoTo 0

End With

'Publish the sheet to a htm file

With TempWB.PublishObjects.Add( \_

SourceType: =xlSourceRange, \_

Filename: =TempFile, \_

Sheet: =TempWB.Sheets(1).Name, \_

source: =TempWB.Sheets(1).UsedRange.Address, \_

HtmlType: =xlHtmlStatic)

.Publish (True)

End With

'Read all data from the htm file into RangetoHTML

Set fso = CreateObject("Scripting.FileSystemObject")

Set ts = fso.GetFile(TempFile).OpenAsTextStream(1, -2)

RangetoHTML = ts.ReadAll

ts.Close

RangetoHTML = Replace(RangetoHTML, "align=center x: publishsource=", \_

"align=left x: publishsource=")

'Close TempWB

TempWB.Close savechanges: =False

'Delete the htm file we used in this function

Kill TempFile

Set ts = Nothing

Set fso = Nothing

Set TempWB = Nothing

End Function

Örnek kullanımı ise şöyledir. Uzun olmaması adına tüm kodu buraya koymadım, tam örnek koda outlook programlama bölümünde değineceğiz.

....

Set rng = Range("A1").CurrentRegion.SpecialCells(xlCellTypeVisible)

....

....

.htmlbody = "Değerli MİY'imiz," & Chr(14) & Chr(14)

.....

.....

.htmlbody = .htmlbody + Application.Run("PERSONAL.xlsb!RangetoHTML", rng) & vbCrLf & Chr(14)

.......

#### İlk visible alan ve sonrasını seçmek

Function ilkvisiblesec(erim As Range) As Range

Set ilkvisiblesec = erim.Offset(1, 0).SpecialCells(xlCellTypeVisible).Cells(1)

End Function

Function ilkvisiblesonrasıalansec(erim As Range) As Range

Dim ilk As Range

Dim son As Range

Dim n As Integer, r As Integer

n = erim.Columns.Count

r = erim.SpecialCells(xlCellTypeVisible).Cells.Count / n - 1 'tek satırlık bir alan olup olmadığını kontrol etmek için

Set ilk = erim.Offset(1, 0).Resize(erim.Rows.Count - 1, 1).SpecialCells(xlCellTypeVisible).Cells(1, 1) 'bu kısım ilk görünen hücreyi verir

Set son = ilk.Offset(0, n - 1)

Set ilktoright = Range(ilk, son)

If r > 2 Then

Set ilkvisiblesonrasıalansec = Range(ilktoright, ilktoright.End(xlDown))

Else

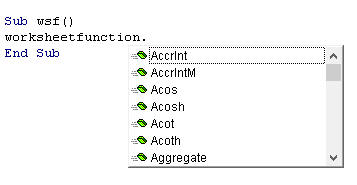
Set ilkvisiblesonrasıalansec = ilktoright

End If

End Function

## WorksheetFunction

VBA'in en önemli fonksiyonlarından bir grup da WorksheetFunction class'ı içinde bulunan metotlardır. Bunlar, adı üzerinde, Excel içinde kullanılan fonksiyonlardır. Burada tüm fonksiyonlar bulunmamakla birlikte önemli bir kısmı dahil edilmiştir. Nelerin olduğunu, aşağıdaki gibi Worksheetfunction yazıp nokta koyduktan sonra intellisense aracılığıyla görebilirsiniz(veya F2'ye basıp object browserdan da arayabilirsiniz). Genelleme yapacak olursak VBA eşleniği olan fonksiyonlar burada yoktur. Now, Left, Mid gibi.



Bunlar aslında UDF'in tersi gibi düşünülebilirler. Nasıl ki VBA aracılığıyla Excel'e yeni fonksiyonlar kazandırabiliyorsak Excel aracılığıyla da VBA'e ek fonksiyonlar kazandırmış oluyoruz.

Mesela bir lookup işlemi yapmak istediğinizde tıpkı Excel içinde VLOOKUP yazarmış gibi VBA içinden bu lookup işlemini yapabilirsiniz. **Bunun sonucunu bir hücrede görmek yerine bir değişkenin içine depolamış olursunuz,** o kadar.

**ÖNEMLİ NOT**: Bazen WorksheetFunction yerine direkt Application nesnesinin kullanıldığını görebilirsiniz. Bu bağlamda ikisi özdeştir diyebiliriz, aşağıdaki örnekteki gibi. Ancak siz yine de WorksheetFunction'ı kullanın, zira yapılan testler bunun %20 civarında daha hızlı olduğunu söylüyor. Üstelik Application'lı versiyonda Intellisense de çıkmamaktadır. Hata yakalama bağlamında da farkları var ancak o detaya girmeyeceğim, arzu eden [https://www.mrExcel.com/forum/Excel-questions/584913-application-vs-application-worksheetfunction.htm](https://www.mrexcel.com/forum/excel-questions/584913-application-vs-application-worksheetfunction.htm) sayfasından bakabilir.

x=Application.Sum(Range("A1: A10"))

x=WorksheetFunction.Sum(Range("A1: A10"))

### Örnekler

#### Örnek 1

İlk örneğimizde sistemden çektiğimde 12 haneden küçük portföy kodlarını 12 haneye tamamlayan bir UDF var. Bu örnekte en başa gereken miktarda 0 konulmaktadır. Bunu Excel’de yapmak için kendisine çok benzeyen şu formülü girmem gerekirdi.

=REPT("0",12-LEN(A2)) & A2

Function portföy12(pk As Range)

portföy12 = WorksheetFunction.Rept("0", 12 - Len(pk)) & pk

End Function

Hangisini yazmak daha kolay, Excel formülünü mü, UDF'i mi? Böylece bir önceki bölüme ithafen UDF'lerin gücünü de anmış olalım.

Bu arada bunu alternatif olarak şöyle de yapabilirdik ancak WorksheetFunction'ı örneklemek adına böyle yaptık.

Function portföy12(pk As Range)

portföy12 = String("0", 12 - Len(pk)) & pk

End Function

Ama diyelim String yöntemi VBA'de yok, siz de WorksheetFunction'ı bilmiyorsunuz. Bu durumda bu UDF'i şöyle yazardık.

Function portfoy12(pk As Range)

ilave = 12 - Len(pk)

For i = 1 To ilave

ek = ek & 0

Next i

portfoy12 = ek & pk

End Function

Gördüğünüz gibi WorksheetFunction bizi gereksiz bir For döngüsü yazmaktan kurtarmış olur(Tabi bu örnekte String alternatifimizin olduğunu bir kez daha altını çizelim, ama her zaman böyle alternatifler olmaz)

#### Örnek 2

İkinci örneğimizde vlookup kullanımı var. Bu örnekte de 'aranan' isimli bir değeri 'table' isimli bir range içinde arıyoruz. Bunu normal VBA ile yapmak için [Dictionary](file:///E:\OneDrive\Uygulama%20Geliştirme\web%20sitelerim\Publishlerim\EE\precompile\Konular\VBAMakro\DizilerveDizimsiYapilar_Dictionaryler.aspx) tanımlamak gerekirdi, ki bu da kodlarımızı oldukça uzatırdı.

.....

sonuc = WorksheetFunction.VLookup(aranan, table, 1, False)

Gördüğünüz gibi Excel'in içinde herhangi bir hücreye Vlookup formülü yazdırmış olmuyoruz, tamamen VBA tarafındayız ve **sanki** Excel'de Vlookup yapmış gibiyiz ve sonucunu da bir değişkene atıyoruz. Excel'de bir hücreye formül yazdırmak için **Range** nesnesinin **Formula** property'si kullanılır. Bu ayrım, özellikle ilk başlarda kafa karıştırıcı olabilir. Birkaç kez kullandıkça farkı anlayacaksınız.

#### Örnek 3

Aşağıda ise çifte vlookup yapmayı sağlayan bir UDF var, burada **COUNTIF** fonksiyonunun kullanımı görüyorsunuz.

Function Çiftevlookup(alan As Range, sütun As Long, İlk\_kriter, İkinci\_kriter)

Dim rCheck As Range, bFound As Boolean, lLoop As Long

On Error Resume Next

Set rCheck = alan.Columns(1).Cells(1, 1)

For lLoop = 1 To WorksheetFunction.CountIf(alan.Columns(1), İlk\_kriter)

Set rCheck = alan.Columns(1).Find(İlk\_kriter, rCheck, xlValues, xlWhole, xlNext, xlRows, False)

If UCase(rCheck(1, 2)) = UCase(İkinci\_kriter) Then

bFound = True

Exit For

End If

Next lLoop

End With

If bFound = True Then

Çiftevlookup= rCheck(1, sütun)

Else

Çiftevlookup= "#N/A"

End If

End Function

#### Örnek 4

Yine bir önceki UDF bölümünde gördüğümüz bir fonksiyonda **LARGE** ve **SMALL** fonksiyonlarının kullanımına şahit oluyoruz.

Function uçhariçort(alan As Range, Uç As Variant)

Dim aratoplam As Double

Dim enbüyükler As Double

Dim enküçükler As Double

For i = 1 To Uç

enbüyükler = enbüyükler + WorksheetFunction.Large(alan, i)

Next i

For i = 1 To Uç

enküçkler = enküçkler + WorksheetFunction.Small(alan, i)

Next i

aratoplam = WorksheetFunction.Sum(alan) - enbüyükler - enküçkler

uçhariçort = aratoplam / (alan.Count - Uç \* 2)

End Function

### Evaluate

WorksheetFunction'a benzer bir de Evaluate metodu vardır. Bazı durumlarda WorksheetFunction'ın özdeşi olup buna göre yazımı daha kolay olduğu için tercih edilebilir. Mesela aşağıdaki iki ifade özdeştir.

WorksheetFunction.Sum(Range("A1: A10"))

Evaluate("SUM(A1: A10)")

Fark ettiyseniz bunda Range yok, aslında tırnak içindeki formül tamamen Excel ortamında yazdığımız formül gibi, içerde hiçbir VBA terimi yok.

Ama Evaluate sadece bu kadar iş yapmıyor, onun üstün basan yönleri de var. Bence bunların en önemlisi içeriğindeki formülü dizi formülü olarak üretebilmesidir. Ve bu da Evaluate'i dizi formülü üreten UDF'ler yazmada oldukça kullanışlı kılar. <http://www.decisionmodels.com/calcsecretsh.htm> linkinde bu metod ile ilgili çok daha detaylı bilgi bulabilirsiniz.

Aşağıdaki UDF ile belirli bir alandaki En Büyük/Küçük X değerin toplamını alırız. X=True ise En büyük, False ise En küçük x değere bakılır.

Function EnXTopla(alan As Range, N As Long, Optional x As Boolean) As Single

Dim strAddress As String

On Error Resume Next

strAddress = alan.Address

If x = False Then

EnXTopla = Evaluate("=SUMPRODUCT((" \_

& strAddress & ">=LARGE(" & strAddress & "," & N & "))\*(" & strAddress & "))")

Else

EnXTopla = Evaluate("=SUMPRODUCT((" \_

& strAddress & "<=SMALL(" & strAddress & "," & N & "))\*(" & strAddress & "))")

End If

End Function

## Neler Öğrendik

Bu bölümde gerek VBA’in kendi fonksiyonlarını kullanmayı gerek kendi fonksiyonlarımızı yaratmayı öğrendik.

Fonksiyonların, algoritmik yapılardan sonra bir programlama dilindeki en önemli yapılar olduğunu gördük.