# 5.bölüm

**5**

# Veri Yapıları (Diziler ve Dizimsiler)

Koşullu ifadeler ve Döngüsel yapılar tüm programlama dünyasının olduğu gibi VBA dünyasının da kilit noktalarından ikisidir. Bunlar aslında Algoritma dediğimiz kavramın omurgasını oluştururlar. Bu bölümde "x=y ise şunu yap, değilse bunu yap" tarzı sorgulamaları ve "bir şeyi yapmaya başla, bunu x defa yap veya şu şey olana kadar yap" gibi döngüsel yapıları göreceğiz.

## Diziler(Arrayler)

### Giriş

Kodumuzda aynı türden bir veya iki farklı eleman kullanacaksak standart değişkenler yeterli olacaktır. Ancak bu sayı artarsa değişken kullanımı pek pratik olmamaya başlar. İşte bu nokta, dizilerin devreye girdiği yerdir.

Mesela 20 bölgesi olan bir bankada çalışıyorsanız her bölge için bir değişken tanımlamak yerine **bölge** adında bir dizi tanımlanıp, her bölge bu diziye elaman olarak atanabilir.  Karşılaştırmaya bakalım:

'pratik olmayan yöntem

Dim bölge1 As String

Dim bölge2 As String

.....

Dim bölge20 As String

Bir de bunun şube versiyonu var, ki şube sayısı 1000 civarındaysa değişken tanımlarken ömrünüzden 1 yıl gider herhalde.

Peki şimdi nedir bu diziler, nasıl tanımlanır, başka neler yapılır, bunlara bakalım.

### Temel bilgiler

#### Tanımlama(Declaration)

Diziler, normal değişkenler gibi **Dim** ifadesi ile tanımlanırlar, ancak değişken adının yanında fazladan yuvarlak parantezlere sahiptirler. Parantezler; dizi boyutu baştan belliyse boyut numarasını içerirler, belli değilse boş bırakılır ve sonradan tanımlanır. Dizi boyutu baştan belirlenen dizilere **Statik** dizi, boyutu baştan belirtilmeyen dizilere **Dinamik** dizi denir.

Dim diziadı(boyut) As Tip

Tip belirtilmezse tıpkı normal değişkenlerde olduğu gibi dizimiz **Variant** tipli bir dizi olur ve her tür değişkeni karışık olarak depolayabilir.

Dim bölgeler(19) As String 'boyut belirtildi

Dim şubekod() As String 'boyut belirtilmedi

Bu örnekte bölge sayısı sabit olduğu için boyut baştan belirtildi, yani Statik tanımlandı ancak bir bankada şube sayısı çok sık değişebilir; yeni şubeler açılır, mevcut şubeler kapanır, şubeler birleşir; o yüzden ona baştan bir boyut belirtmesek de olur, yani Dinamik tanımlandı. Bu dinamik dizilere aşağıda ayrıca bakıyor olacağız.

#### Boyut(elaman sayısı), index, alt/üst limitler

Dizi için tanımlanan eleman sayısına boyut denir. Boyut belirtmenin de iki yolu vardır.

**Dim dizi(x to y) As Tip**

**Dim dizi(y) As Tip**

İlk yöntemde dizinin kaçıncı indexten başlayıp kaçta biteceği belirtilirken ikinci yöntemde ise doğrudan kaçta biteceği belirtilir, kaçta başlayacağı ise **Option Base** ifadesinin kullanılıp kullanılmadığına göre değişir. İndeksin biteceği son yere üst sınır denir ve bu sınır **Ubound** fonksiyonu ile elde edilir.

Dim segmentler(4) As String

segmentler(0)="Bireysel"

segmentler(1)="Birebir"

segmentler(2)="Kobi"

segmentler(3)="Ticari"

'segmentler(4)="Özel"

For i=0 to UBound(segmentler)

MsgBox segmentler(i)

Next i

Yukardaki örnekte gördüğünüz üzere son indexli elamana değer atanmadı, bu yüzden de boş olarak göründü. Anlayacağınız üzere, dizideki tüm elemanlara değer atanmak zorunda değildir. Yani, bir dizide eleman sayısı ile dolu(değer atanmış) eleman sayısı aynı olmayabilir.

Varsayılan olarak dizilerin ilk eleman indeksi 0'dır. Ancak bu index **Option Base 1** ifadesi ile 1 yapılabilir, ki ben bunu zorunda olmadığınız sürece çok kullanmanızı tavsiye etmiyorum. Veya yukarda 1. yöntemdeki gibi dizi boyutu baştan **1 to y** şeklinde belirtilerek de başlangıç indexi 1 yapılabilir. Yani şu iki ifade tamamen özdeştir.

Option Base 1

Dim bolgeler(20) As String

ve

Dim bolgeler(1 to 20) As String

İndex numarası açısından olmasa da eleman sayısı açısından şu dizi de yukardakilerle aynı kapasitededir, yani hepsi de 20 eleman içerir.

Dim bolgeler(19) As String

Dizi tanımının yönteminden ve **Option Base** kullanımından bağımsız olarak alt indexin ne olduğu ise **LBound** fonksiyonu ile elde edilir.

**Option Base 1** kullanımını tavsiye etmiyoruz dedik, zira ilgili modüldeki tüm prosedürler için indexi 1den başlatır. Bununla beraber bazı dizileri bilinçli olarak 1 nolu indeksten başlatmak gerekebilir. Mesela ayno isimli bir dizimiz olduğunu düşünün, ayno(1) diyince Ocak ayını ele almak, anlaşılırlık açısından daha makbuldür, pek tabiki ayno(1) içinde Şubat ayı da depolanabilir ancak bu yol, konuşma diline biraz aykırılık teşkil edeceği için bunu ayno(1 to 12) şeklinde tanımlamayı tercih etmek daha akıllıca olacaktır.

Bu arada **x to y** yönteminde x olarak 1'den büyük değerler de belirtilebilir ama bunun pratikte çok kullanıldığı görülmez.

Çok boyutlu dizilerde eleman sayısı, boyutlardaki elemanların çarpımına eşittir.(Bunlar aşağıda ayrıca detaylı incelenecek)

**Özetleyecek olursak;**

Dim b(10) As String şeklinde tanımlanan 1 boyutlu bir dizide;

|  |  |  |
| --- | --- | --- |
| Aranan | Yöntem | Sonuç |
| Alt limit | LBound(b) | 0 |
| Üst limit | UBound(b) | 10 |
| Eleman sayısı | Ubound(b)-LBound(b)+1 | 11 |

Dim b(1 to 10) As String şeklinde tanımlanan 1 boyutlu bir başka dizide;

|  |  |  |
| --- | --- | --- |
| Aranan | Yöntem | Sonuç |
| Alt limit | LBound(b) | 1 |
| Üst limit | UBound(b) | 10 |
| Eleman sayısı | Ubound(b)-LBound(b)+1 | 10 |

Dim b(10,5) As Integer şeklinde tanımlanan 2 boyutlu bir dizide;

|  |  |  |
| --- | --- | --- |
| Aranan | Yöntem | Sonuç |
| 1.boyutun Alt limiti | LBound(b,1) | 0 |
| 1.boyutun Üst limiti | UBound(b,1) | 10 |
| 2.boyutun Alt limiti | LBound(b,2) | 0 |
| 2.boyutun Üst limiti | UBound(b,2) | 5 |
| 1.boyutun Eleman sayısı | Ubound(b,1)-LBound(b,1)+1 | 11 |
| 2.boyutun Eleman sayısı | Ubound(b,2)-LBound(b,2)+1 | 6 |
| Dizideki Eleman sayısı | 1. boyut elaman sayısı\*2.boyut eleman sayısı | 66 |

Dikkat: Bir dizinin boyutu baştan bir kez belirtilirse bir daha asla değişmez. Boyutu değişen dizilere ise aşağıda değineceğiz.

#### Elemanlara değer atama

##### İlk değer atama

Dizi elamanlarına index numaralarıyla ulaşırız. Normalde bunlara tek tek değer atamak genelde pratikte karşılaşılan bir durum değildir, belki küçük boyutlu dizilerde olabilir ancak genelde bir döngüsel yapı ile bir hücre grubundan değer okuyup onları atamak şeklinde olmaktadır. Aksi halde kodumuz oldukça uzayacaktır.

Dim Segment(3) As String

Segment(0) = "Bireysel"

Segment(1) = "Birebir"

Segment(2) = "Kobi"

Segment(3) = "Ticari"

Döngüyle atamaya örnek olarak da şunu verebiliriz;A1: A20 arasındaki bölge kodlarını bölge dizisine atıyoruz.

For i=1 to 20

bölge(i)=Cells(i,1).Value2

Next i

##### Değer atanmamış elemanlar ve Erase fonksiyonu

Henüz değer atanmamış dizi elemanları, dizinin tipine göre default değerlerini alırlar. Bunlar;

* String diziler için sıfır uzunluklu metin, yani ""
* Nümerik diziler için 0
* Variant diziler için Empty
* Object diziler için Nothing

Statik dizilerde elemanların hepsini tek seferde varsayılan değerlerine döndürmek için **Erase** fonksiyonu kullanılır. Yani dizi elemanlarının taşıdığı değerler boşaltılır. Ancak bunlar hala hafızada yer kaplamaya devam ederler.

Sub erasestatik()

Dim segment(2) As String

segment(0) = "bireysel"

segment(1) = "birebirt"

segment(2) = "kobi"

Erase segment()

Debug.Print segment(1) '"" döndürür

End Sub

##### Array fonksiyonu

Elle tek tek tanımlama yapılması gereken durumlarda, bir diğer eleman tanımlama yöntemi de **Array** fonksiyonunu kullanmaktır, ki bu sadece elaman değeri atama değil aynı zamanda diziyi tanımlama yöntemidir de. Zira bu şekilde tanımlanan diziler **Variant** tipte olurlar ve bildiğiniz gibi **Dim** ile tanımlanmayan tüm değişkenler Variant kabul edilirler. Bu sayede elemanlar tanımlanırken dizi de yaratılmış olur. Ama biz yine de iyi bir programcı olup dizimizi Dim ile tanımlayalım. **Bu yöntemle tanımlanan dizilerde başlangıç indeksi her zaman 0 olur. Yine, bu yöntemle yaratılan diziler 1 boyutlu olur**

Dim Segment As Variant

Segment = Array("Bireysel","Birebir","Kobi","Ticari") 'tek satırda tanımlama imkanı

Debug.Print Segment(2) 'Kobi yazar

Bu yöntemin pratik bir kullanımı "ayisim" gibi bir fonksiyon şeklinde kendini gösterebilir. Fonksiyonlarla kafanızı karıştırmamak için şimdilik Sub prosedür şeklinde örneğe görelim.

Sub ayisimornek()

Dim ayisim As Variant

ayisim = Array("", "Ocak", "Şubat") 'ilk ayı boş geçtim, index 0 olduğu için

Debug.Print ayisim(2)

End Sub

#### Dizi içinde dolaşma ve elemanlara erişim

Dizilere eleman atama işini döngülerle yaptığımız gibi, dizi elamanlarını okumayı da yine genelde döngülerle yaparız. Ender olarak kod içinde bir yerde bir index numarası temin edip onu doğrudan da kullandığımız da olur.

For i=1 to 20

Cells(i,1).Value2=bölge(i)

Next i

Daha şık şekli aşağıdaki gibi olabilir, hatta sadece şık değil aynı zamanda güvenlidir de.

For i=LBound(bölge) to UBound(bölge)

Cells(i,1).Value2=bölge(i)

Next i

Evet, biraz daha fazla kod yazmış  olduk ama güvenlik ön planda olacaksa hardcoded(rakamı doğrudan belirterek) alt-üst limit vermek yerine bu fonksiyonlarla vermek daha verimlidir. Hardcoded yazıldığında, bölge sayısı 1 arttığında 20 yerine 21 yapmayı unutursanız kodunuz yine hata almadan çalışır ancak eksik çalışmış olur, ve belki akabinde çalışan kodlarla bölgelere otomatik mail gidiyorsa, son bölgeye hiç mail gitmemiş olur. Bir de içiçe bir dizi varsa ve her seviye için üst limit değişiyorsa Ubound'ı kullanmak zaten zaruri olacaktır.

Üstelik bu yazım şekli alt limitin 0 mı 1 mi olduğunu da pek önemsemez . Böylece **Option Base** var mıydı yok muydu, diziyi (1 to 20) şeklinde mi yoksa (20) şeklinde mi tanımladığınızı hatırlamak zorunda kalmazsınız.

dikkat: Dizi içinde For döngüsü ile dolaşırken diziye eleman atama işlemi sadece For i=1 to 10 şeklindeki basit For döngüsü ile yapılabilirken, eleman değerini okuma işlemi ise hem basit For ile hem de For Each ile yapılabilir.

Mesela aşağıdaki gibi bir kullanım sorunsuz çalışırken,

Dim Bölge(1 to 20) As String

For i=1 to 20

bölge(i)=Cells(i,1).Value2

Next i

bu kod da hata vermeden çalışır ancak eleman değerlerinin değişmediğini görebilirsiniz. (Eğer daha önce bir değer atanmadıysa içleri boş görünecektir.)

Dim Bölge(1 To 20) As String

For Each b In Bölge

b = ActiveCell.Value2

ActiveCell.Offset(1, 0).Select

Next b

### İleri seviye işlemler

#### Dinamik diziler

Yukarıdaki örnekler hep statik dizi örnekleriydi(Array fonksiyonu ile tanımlananlar hariç)

Dizimizin boyutunu baştan bilmiyorsak ve kodun gidişatına göre değişken bir şekilde karşımıza çıkma durumu varsa, diziyi boyutsuz yani dinamik tanımlarız ve zamanı geldiğinde boyutunu belirtiriz. Bunu da **ReDim** ifadesi ile yaparız. Aslında yaptığımız şey, yeni bir statik dizi yaratmaktır. Zira **ReDim** kullanıp da yeni boyutunu verdiğimiz diziyi yine yeterli bulmazsak tekrar boyut değiştirebiliriz.

Dim şubeler() As String

....

....

ReDim şubeler(800)

Statik dizilerde, genelde kaynak israfı söz konusudur, zira baştan olası en yüksek değere göre boyut belirlenir ve bu boyutların hepsi çoğu zaman kullanılmaz. Tabiki bölge sayısı gibi kesin olarak bilinen ve hepsi kullanılan durumlar istisnadır. Bu nedenle genel olarak Statik değil Dinamik dizi kullanılması önerilir.

##### Erase fonksiyonu

Dinamik dizilerde **Erase** fonksiyonu statik dizilerden farklı çalışır. Statik dizilerde Erase, elemanları default değerlerine atarken dinamik dizide diziyi boyutsuzlaştırır, yani bir nevi siler. Diziyi tekrar kullanmak isterseniz ReDim ile yeniden boyutlandırmanız gerekir.

##### İlave boyut artışı

Diyelim ki ReDim ile 10 elemanlık bir boyutlandırma yaptınız ancak öyle bir nokta geldi ki, 10 eleman yetmiyor, yani boyut artırmanız lazım, böyle bir durumda **ReDim Preserve** ifadesini kullanırız. Preserve demezsek dizi yine yeni boyuta göre boyutlanır ancak önceki elemanlar silinmiş olur. Preserve ile ilk 10 elemanın değerini de korumuş oluruz.

Boyut artırma durumların ReDim'in bir veya iki kez kullanılması önerilir. Birkaç kez boyut artırma ihtiyacı oluyorsa belki dizi değil de Collection kullanmak faydalı olabilir.

**NOT**: Variant olarak tanımlanan **değişkenler** doğası gereği dinamiktirler.(Variant tanımlanan **diziler** ise statiktir)

Dim statikVar As Variant 'standart Variant değişken

Dim dinamikVar(5) As Variant 'Variant tipli dizi

##### Variant değişken vs Variant dizi

Variantlarla ilgili detaylı bir örnek aşağıda bulunmaktadır. Bunun oldukça aydınlatıcı olduğunu düşünüyorum.

Sub variantlı()

Dim aylar1 As Variant 'içine istediğiniz tipte değer atayabileceğiniz bir DEĞİŞKENDİR, buna dizi de dahildir,

'ama dizi atayana kadar dizi değildir ve IsArray testi false döner. Ubound da kullanamazsınız

Dim aylar2() As Variant 'Parantez olduğu için bu kesinlikle DİZİDİR, IsArray true döner

'\*\*\*öncelikle parantezsiz olan yani değişken olan Aylara bakalım\*\*\*\*\*

aylar1 = 1 'number atandı

Debug.Print aylar1

'Debug.Print aylar1(1) 'hata verir, çünkü içeriği henüz bir dizi değil.

Debug.Print IsArray(aylar1) 'false

'Debug.Print UBound(aylar1) 'çalışmaz çünkü şuan için dizi değil

aylar1 = "volki" 'string

Debug.Print IsArray(aylar1) 'yine false

Debug.Print aylar1

aylar1 = Now 'tarih

Debug.Print IsArray(aylar1) 'hala false

Debug.Print aylar1

aylar1 = Array("hi", "world", "naber") 'dizi

Debug.Print IsArray(aylar1) 'artık true, çünkü Array function ile array yaptık

Debug.Print UBound(aylar1)

Debug.Print aylar1(2) 'artık hata vermez, çünkü içeriği bir dizi

aylar1 = Array(Array("hi", "world", "naber"), Array(1, "ss", 3)) 'dizi dizisi

Debug.Print UBound(aylar1)

Debug.Print aylar1(1)(1)

aylar1 = Range("a1: a5").Value ' range atadık, 2 boyutlu dizi

Debug.Print UBound(aylar1) '5 döner

Debug.Print aylar1(2, 1) '2.satırdaki değer döner

'\*\*\*şmdi de Dizi olan Aylara bakalım

Debug.Print IsArray(aylar2)

'Debug.Print UBound(aylar2) hata verir, zira henüz boyut belli değil, ya ReDim yapılmalı ya da Array ile değer atanmalı

ReDim aylar2(3)

Debug.Print UBound(aylar2)

aylar2 = Array("OCAK", "ŞUBAT", "MART", "NİSAN", "MAYIS", "HAZİRAN", "TEMMUZ", "AĞUSTOS", "EYLÜL", "EKİM", "KASIM", "ARALIK") '

Debug.Print UBound(aylar2)

Debug.Print aylar2(2)

Debug.Print UBound(aylar2)

'şimdi de farklı data tiplerinde atama yapıyoruz, Variant olduğu için sorun olmuyor

aylar2 = Array("OCAK", 2, 3, "30.04.2015", Now + 60, "HAZİRAN", "TEMMUZ", "AĞUSTOS", "EYLÜL", "EKİM", "KASIM", "ARALIK") ' variant olduğu için içine karışık tipli verilen atayabilirim

Debug.Print aylar2(4)

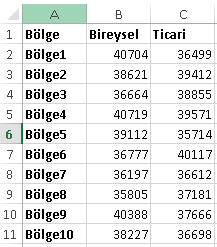
End Sub

#### Çok boyutlu diziler

Şimdiye kadar gördüğümüz dizilerin çoğu tek boyutlu idi, ama ihtiyacımıza göre birden fazla boyutlu diziler de oluşturabiliyoruz. Pratikte üçten fazla boyutun kullanıldığını ben açıkçası ne duydum ne de gördüm. Şahsen kendi kodlarımda kullandığım dizilerin büyük kısmı tek boyutludur, birkaç tane 2 boyutlu bir tane de 3 boyutlu dizim bulunmakta. 3 boyutlu dizi örneğinde aynı zamanda Dictionary de kullandığım için bu örneği o bölümde ele alacağız.

Şimdi 2 boyutlu bir dizi kullanım örneğine bakalım. Excel sayfalarının kendileri zaten satır ve sütunlardan oluşmakta olup 2 boyutludurlar ve bu durumu 2 boyutlu dizilerle birlikte çok sık kullanıyor olacağız.

Mesela 10 bölgesi olan bir bankada her bölgenin bir Bireysel bir de Ticari müdürü olduğunu düşünelim. Bunları bir diziye atama işlemi nasıl oluyor ona bakalım.



Sub ikiboyutludizi()

Dim mudur(1 To 10, 1 To 2) As Long 'ilk boyut bölge için, ikinci boyut segment tipi için

Dim i As Integer, j As Integer

For i = LBound(mudur, 1) To UBound(mudur, 1)

For j = LBound(mudur, 2) To UBound(mudur, 2)

mudur(i, j) = Cells(i + 1, j + 1).Value

Next j

Next i

Debug.Print mudur(3, 2)  
End Sub

##### Tabi bunun için aşağıda gösterdiğim başka bir yöntem var ki, bundan çok daha basittir.

##### İçiçe diziler(Dizi dizileri)

İngilizcede Array of Array ve Jagged Array olarak kullanılan bu dizi türünü Variant tipte dizi tanımlayarak elde ederiz. Bilindiği gibi Variant veri tipi içinde her şeyi tutabilir, buna diziler de dahildir. Yalnız burdaki ayrıma dikkat etmek lazım, klasik Variant bir değişken tanımlamıyoruz, Variant tipli bir dizi tanımlıyoruz. Aradaki ayrımı görmek için şu iki satıra bakalım:

Dim d As Variant 'bu Variant tipli klasik bir değişkendir  
 Dim d() As Variant 'bu ise Variant tipli bir dizidir

İçiçe dizilerin tanımlaması iki ayrı dizi şeklinde olur, ama kullanımı dizi(x)(y) şeklindedir. Yani aslında bu dizi türü tek boyutludur ama içerdiği her elaman da bir başka dizidir.

Dim dışdizi() As Variant 'ana dizi Variant olmalıdır

Dim içdizi() As String ' Alt dizinin Variant olması gerekmez

'iç diziyi dış diziye eleman olarak atama

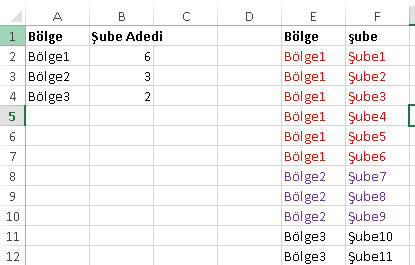
dışdizi(n)=içdizi

'nihai kullanım şekli

Debug.Print dışdizi(x)(y)

Bu dizilerin kullanımı çok boyutlu dizilere benzer ancak, kullanım ihtiyacı ve şekli küçük farklar göstermektedir. Şöyle ki, çok boyutlu dizilerde tüm boyutlar için hafızada yer ayrılır, belki o boyutlardan bazısı hiç kullanılmayacak bile. Diyelim 20 bölgeli bir banka var, bazı bölgelerde 40 şube varken bazısında 15 şube bulunabilir. Şimdi **Şubeler(1 to 20, 1 to 40)** şeklinde 2 boyutlu bir dizi tanımlarsak bazı bölgelerde bazı boyutlar israf olacak. İşte içiçe dizilerde bu israf olmuyor. Örneğin 2.bölgenin şube sayısı 18 ise, burdaki 18.şubeyi tanımlayacak son elaman bölge(2)(18) oluyor, bölge(2)(19) şeklinde bir tanımlama yapılmıyor.

Mesela aşağıdaki tabloya göre içiçe dizimizi oluşturalım.



Sub içiçedizi()

Dim bolge(1 To 3) As Variant 'dış dizi

Dim subeler() As String 'iç dizi

Dim i As Integer, s As Integer, k As Integer

For i = 1 To 3

ReDim subeler(1 To Cells(i + 1, 2)) 'iç dizinin boyutunu ayarlıyoruz, her bölgede bu boyut değişecek

'iç diziyi oluşturalım

For s = 1 To UBound(subeler)

subeler(s) = Cells(s + k + 1, 6)

Next s

k = k + s - 1 'satır sayısı resetlenmesin diye geçici bir değişkene atıyorum

'iç diziyi dış dizinin ilgili elemanına atayalım

bolge(i) = subeler

Next i

Debug.Print bolge(1)(5) 'Şube5

Debug.Print bolge(2)(2) 'Şube8

Debug.Print bolge(3)(4) 'hata verir, çünkü 3.bölgenin 4.şubesi yok

End Sub

Dizi dizisiyle yapılabilen işlemler "Collection of Collection" veya "Dictionary of Dictionary" yapılarıyla da yapılabilir. Bunları bir sonraki bölümde görüyor olacağız. Bu arada hepsinin avantaj ve dezavantajını anlatan güzel bir sayfa var, ona [http://stackoverflow.com/questions/19633937/can-you-declare-jagged-arrays-in-Excel-VBA-directly](http://stackoverflow.com/questions/19633937/can-you-declare-jagged-arrays-in-excel-vba-directly) linkinden ulaşabilirsiniz.

##### Rangeler ve Diziler

###### Bir hücre grubunu diziye atama(Sayfadan okuma)

Dizileri, bir hücre grubundan hızlı veri okuma ve yazma amacıyla da kullanırız. Bu kullanım şekli, özellikle ilgili veri kümesi üzerinde bir güncelleme(belli bir rakamla çarpmak gibi) yapmak istediğinizde idealdir.

Tabi bu amaçla kullanmak istediğimizde standart dizi tanımı yerine **Variant** olarak tanımlarız, ve tanımladığımız dizi **2 boyutlu bir dizi** olur. Zira bir sayfaya baktığınızda gördüğünüz şey satır ve sütunlardan oluşan iki boyutlu bir dizidir. Böyle bir dizide de ilk boyut satır ikinci boyut sütun olur. Bu noktada karışıklığa neden olan bir konu vardır ki o da şudur: Söz konusu hücre grubu tek kolonluk bir alandan oluşuyorsa bile iki boyutludur. İlk boyut satır sayısı kadar elemanlı, ikinci grup da sütun sayısı olan 1 elemanlıdır. **Bu tür dizilerde indeks 0dan değil 1den başlar.(Tek boyutlu variant dizilerde ise 0'dan başlıyordu)**

Mesela aşağıdaki kod ile A1: A1000 hücrelerindeki değerler 100 ile çarpılır.

Sub rakamguncelle()

Dim rakamlar As Variant 'Dizi tanımı

rakamlar = Range("A1: A10000").Value 'burada hücreden diziye okuma yaptık

For i = LBound(rakamlar) To UBound(rakamlar) 'i'ler satır boyutudur

rakamlar(i, 1) = rakamlar(i, 1) / 100 '1 ise sütun boyutu

Next i

Range("A1: A10000").Value = rakamlar 'üzerine de yazabiliriz başka bir yere de yazabilirdik

End Sub

Alanımız tek kolon değil de birden çok kolondan oluşuyorsa sütun için de bir iç döngü daha eklememiz gerekir. Bu sefer hücre grubumuzu dinamik düşünelim, buna göre kodumuzu aşağıdaki gibi olacaktır.

Dim rakamlar As Variant 'Dizi tanımı

rakamlar = Selection.Value 'burada hücreden diziye okuma yaptık

For i = LBound(rakamlar) To UBound(rakamlar) 'i'ler satır boyutudur

For j = LBound(rakamlar, 2) To UBound(rakamlar, 2) 'j'ler satır boyutudur

rakamlar(i, j) = rakamlar(i, j) \* 1000

Next j

Next i

Selection.Value = rakamlar

###### Tek boyutlu bir diziyi sayfaya yazdırma

Yazdırma işlemi yapmak için **Resize** özelliği ile hücreleri yeniden boyutlandırmayı bilmeniz gerekiyor. Resize hakkında bilgi sahibi değilseniz **Dört Temel Nesne>Range** konusundan bakıp önbilgi edinebilirsiniz.

Bu işlem için bir **Range** nesnesi yaratırız ve sonra hedef hücreyi belirleriz. Sonra bu hücreyi **Resize** ile dizi boyutu kadar genişletiriz.

' Yatay yazma(Tek satırlı çok kolonlu)

Dim dizi(1 To 3) As Integer

Dim Hedef As Range

dizi(1) = 10

dizi(2) = 20

dizi(3) = 30

Set Hedef = Range("A1").Resize(1, UBound(dizi))

Hedef.Value = dizi

Eğer dizimizi dikey şekilde yani tek kolon çok satırda yazmak isteseydik, Resize argümanlarını ters çevirmemiz ve Transpose metodunu kullanmamız gerekirdi.

' Dikey yazma(Tek kolonk çok satır)

Dim dizi(1 To 3) As Integer

Dim Hedef As Range

dizi(1) = 10

dizi(2) = 20

dizi(3) = 30

Set Hedef = Range("A1").Resize(UBound(dizi), 1)

Hedef.Value = WorksheetFunction.Transpose(dizi)

###### İki boyutlu dizileri sayfaya yazdırma

İki boyutlu bir dizimizi sayfaya yazdırmak istiyorsak yine Resize metodunu kullanırız, ilk boyutu satır olarak, ikinci boyutu da kolon olarak kullanabiliriz, veya tam tersini de istersek yerlerini değiştirip Transpoze yaparız.

Dim dizi(1 To 3, 1 To 2) As Integer

Dim alan As Range

dizi(1, 1) = 10

dizi(2, 1) = 20

dizi(3, 1) = 30

dizi(1, 2) = 100

dizi(2, 2) = 200

dizi(3, 2) = 300

Set alan = Range("A1").Resize(UBound(dizi, 1), UBound(dizi, 2))

alan.Value = dizi

Transpoze hali de şöyle:

Dim dizi(1 To 3, 1 To 2) As Integer

Dim alan As Range

dizi(1, 1) = 10

dizi(2, 1) = 20

dizi(3, 1) = 30

dizi(1, 2) = 100

dizi(2, 2) = 200

dizi(3, 2) = 300

Set alan = Range("A1").Resize(UBound(dizi, 2), UBound(dizi, 1))

alan.Value = WorksheetFunction.Transpose(dizi)

#### Dizilerin diğer kullanma şekilleri

##### Dizileri dizilere atama

Gelişmiş programlama dillerinde olduğunun aksine VBA'de bir diziyi tek seferde başka bir diziye kopyalama/atama yöntemi bulunmamaktadır. Yani şöyle bir kod çalışmaz.

Dim A(1 To 3) As Long

Dim B(1 To 3) As Long

A(1)=100000

A(2)=200000

A(3)=300000

B=A

Peki nasıl yapılır? Tabiki döngülerle;

Dim A(1 To 3) As Long

Dim B(1 To 3) As Long

A(1)=100000

A(2)=200000

A(3)=300000

For i=1 to 10

B(i)=A(i)

Next i

Debug.Print B(2)

Variant dizileri Variant dizilere atama

Dizileri başka dizilere atayamıyoruz ancak içi dizi olan bir Variant’ı başka bir Variant’a atayabiliyoruz. Zira, bu işlem aslında dizi atama değil değişken atamadır.

Dim A As Variant

Dim B As Variant

A = Array(100000, 200000, 30000)

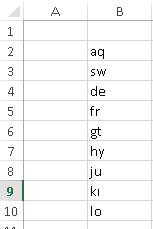
B=A 'atama başarılıdır

Debug.Print B(2) '30000 yazar

##### Range atanmış Variant dizileri normal dizilere atama

Bir hücre grubunu doğrudan bir variant diziye atama şeklini yukarda görmüştük, ama bir şekilde bunları tek boyutlu bir diziye atamanız gerekirse bunu yapmanın yolu döngü kullanmaktır. Tabi istenirse arada hiç variant dizi olmadan da doğrudan ilgili alanlar üzerinden de geçilebilir ancak, eğer bir şekilde ilgili alan kod içinde silindiyse, veya büyük bir alan ise performans sorunu yaşamamak için Variant diziden normal diziye atamak daha verimli olacaktır, zira **diziler üzerindeki işlemler range'lere göre çok daha hızlıdır**.

Şimdi aşağıdaki gibi bir hücre grubu olsun, bunu önce Varianta sonra da normal bir diziye atayalım.



Sub birkolonlu\_variant\_to\_dizi()

Dim var As Variant

Dim dizi() As String

Dim i As Integer

var = Range("B2: B10").Value2

ReDim dizi(1 To UBound(var, 1))

For i = 1 To UBound(var, 1)

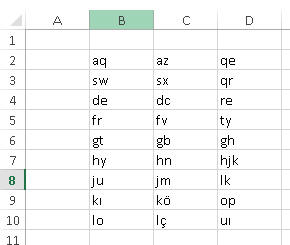
dizi(i) = var(i, 1)

Next i

MsgBox dizi(3)

End Sub

Şimdi bunu bir de çok boyutlu dizide yapalım



Sub çokkolonlu\_variant\_to\_dizi()

Dim var As Variant

Dim dizi() As String

Dim s As Integer, k As Integer, i As Integer

var = Range("B2: D10").Value2

ReDim dizi(1 To UBound(var, 1) \* UBound(var, 2))

s = 0

For k = 1 To UBound(var, 2)

For i = 1 To UBound(var, 1)

dizi(i + s) = var(i, k)

Next i

s = s + UBound(var, 1) 'her boyut geçişinde tekrar başa dönmesin diye aratoplam alınır

Next k

MsgBox dizi(15)

End Sub

Aynı diziyi iki boyutlu bir standart diziye atamak da aşağıdaki gibi olurdu.

Sub çokkolonlu\_variant\_to\_çokboyutludizi()

Dim var As Variant

Dim dizi() As String

Dim s As Integer, k As Integer, i As Integer

var = Range("B2: D10").Value2

ReDim dizi(1 To UBound(var, 1), 1 To UBound(var, 2))

For k = 1 To UBound(var, 2)

For i = 1 To UBound(var, 1)

dizi(i, k) = var(i, k)

Next i

Next k

MsgBox dizi(2, 3)

End Sub

##### Dizileri collectiona atama

Bazı durumlarda elimizdeki diziyi daha fazla esnekliği olan ve bazı yönlerden dizilere benzeyen collectionlara atamak isteriz. Ancak collection konusunu henüz işlemediğimiz için(siteyi sırayla okuduğunuz varsayımıyla hareket ederek) bu durumu collection sayfasında ele alacağız.

##### Dizileri bir prosedüre parametre olarak gönderme

Dizileri bir Sub veya Function prosedüre parametre olarak gönderebiliriz. Unutulmaması gereken nokta, diziler her zaman **ByRef** anahtar kelimesi ile gönderilir. Aşağıda basit bir örneğimiz bulunuyor.

Sub dizigonder()

Dim d(5) As String

'sadece iki elemanı dolduralım

d(0) = "volkan"

d(1) = "meltem"

Call mesajver(d)

End Sub

Sub mesajver(ByRef dz() As String)

For Each a In dz

If a <> "" Then i = i + 1

Next a

eleman = UBound(dz) - LBound(dz) + 1

MsgBox eleman & " adet elemanın " & i & " tanesi doludur"

End Sub

##### Prosedürden sonuç olarak dizi döndürme

Fonksiyonlar konusunda göreceğimiz gibi, genelde bir fonksiyondan sadece tek bir değer döndürülür. Aslında bu birçok yerde doğru diye anlatılan yanlış bir bilgidir. Genelde durum böyledir ancak fonksiyonlar tek bir değer döndürmek zorunda değildir, sadece dönüş değeri tek olmak zorundadır. Zira gerek ByRef kullanımı, gerek dönüş değerinin dizi/collection olması sayesinde fonksiyonlar pekala çoklu değer döndürebilirler. Aşağıdaki örnekte çoklu değer olarak dizi döndürüyoruz.

Sub diziata()

'boyutsuz dizi yaratılır

Dim d() As Integer

d = dizidondur '6 değerli bir dizi döner, istediğimizi kullanırız

Debug.Print d(3)

End Sub

Function dizidondur() As Integer() 'dönen değer tipi () ile biter ki dizi döndürdüğü anlaşılsın

Dim dizim(0 To 5) As Integer

For x = 0 To 5

dizim(x) = 10 \* x

Next x

dizidondur = dizim

End Function

#### Split ve join

##### Split

Dizilerle kullandığımız çok faydalı iki fonksiyon vardır.

Bunlardan ilki **Split** fonksiyonudur. Bununla, belirli bir ayraçla birleşmiş olan bir **String** ifade parçalara ayrılarak bir Variant içine veya klasik bir dizi içine aktarılır. Gördüğünüz gibi bu da aslında bir nevi dizi tanımlama yöntemi olmaktadır.

Bu fonksiyon, genelde bir hücreden ";" veya "," ile ayrılmış elemanları tek tek elde etmek için kullanılır.

Sub splitornek()

Dim s As String

s = "35516;37770;34234"

Dim dizi As Variant 'veya Dim dizi() as String şeklinde boyutsuz dizi

dizi = Split(s, ";")

Debug.Print dizi(0) '35516 döner

End Sub

Bu Split fonksiyonunu kullanarak bir UDF yazmıştım, ki bu sayfayı yazarkenki en yüksek Excel sürümü olan Excel 2016 içinde bile bu fonksiyon hala yerel olarak bulunmuyor. Bunu anlamak gerçekten zor, zira çok ihtiyaç duyulan bir fonksiyon olduğunu düşünüyorum. Neyse ki VBA ve UDF var da başımızın çaresine bakabiliyoruz.

Bu fonksiyon, bir metindeki belirtilen indeksteki kelimeyi seçiyor. Ayracı default olarak " " yani boşluk belirledim, ancak isteyen farklı bir ayraç da belirleyebiliyor. İşte bu fonksiyonum:

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

'normal bir cümlede ayrac boşluk olacağı için ayracı girmeye gerek yok, zaten default olarak " " atadım.

'ama içeriği mesela / ile ayrılmış bir hücre varsa 3.parametreyi / olarak girersiniz

Dim kelimeler As Variant

kelimeler = Split(hucre.Value2, ayrac)

kelimesec = kelimeler(kaçıncı - 1)

End Function

##### Join

Diğer faydalı fonksiyon da **Join** olup bir dizi içindeki elemanları, belirtilen ayraçla birleştirip bir String ifade döndürür, yani Splitin tersi gibi çalışır.

Sub joinornek()

Dim b As String

Dim v As Variant

v = Array(35516, 34433, 32335)

b = Join(v, ";")

Debug.Print b

End Sub

Bu fonksiyon sadece bir boyutlu dizilerde çalışır. O yüzden bir Range'den okunan ve 2 boyutlu olan bir **Variant** dizide işe yaramayacaktır. Ama bunu yapmanın da bazı yolları var. Yukarda Variant dizileri klasik dizilere çevirmeyi görmüştük, yine böyle yaparız ve gerisi kolay.

###### 1.Yöntem: Variant diziyi normal diziye dönüştürmek

1 kolonlularda

Sub birkolonlu\_variant\_to\_dizi()

Dim var As Variant

Dim dizi() As String

Dim kombine As String

Dim i As Integer

var = Range("B2: B10").Value2

ReDim dizi(1 To UBound(var, 1))

For i = 1 To UBound(var, 1)

dizi(i) = var(i, 1)

Next i

kombine = Join(dizi, ", ")

MsgBox kombine

End Sub

2 kolonlularda

Sub çokkolonlu\_variant\_to\_dizi()

Dim var As Variant

Dim dizi() As String

Dim kombine As String

Dim s As Integer, k As Integer, i As Integer

var = Range("B2: D10").Value2

ReDim dizi(1 To UBound(var, 1) \* UBound(var, 2))

s = 0

For k = 1 To UBound(var, 2)

For i = 1 To UBound(var, 1)

dizi(i + s) = var(i, k)

Next i

s = s + UBound(var, 1) 'her boyut geçişinde tekrar başa dönmesin diye aratoplam alınır

Next k

kombine = Join(dizi, ", ")

MsgBox kombine

End Sub

###### 2.yöntem: Transpoze(sadece tek kolonsa)

kombine = Join(WorksheetFunction.Transpose(Range("B2: B10").Value), ", ")

#### Varmı kontrolü ve Index bulma

##### Filtreleme

Dizilerle ilgili sık yapılan işlemlerden biri de, bir elemanın o dizide bulunup bulunmadığı ve /veya bulunuyorsa da index numarasını öğrenmek olacaktır.

Bu işlemlere geçmeden önce **Filter** fonksiyonundan bahsedelim. Bu metod ile **String** tipli bir dizinin içinde belirli bir metni arar ve bunları filtreler, eşleşmeyenleri hariç tutarız. (Burada filtrelemekten kastım, elemek değil seçmektir.)

**Syntax: Filter(dizi,aranan,[Dahilmihariçmi],[aramatipi])**

**dizi** olarak tek boyutlu bir diziyi parametre olarak veririz, bu nedenle range atanmış bir Variant dizi parametre olarak verilemez.

**Dahilmihariçmi** parametresi girilmez veya True girilirse aranan kelimeyi içeren kelimeler aranır, False girilirse aranan kelimeyi içermeyen elemanlar filtrelenir.

**aramatipi** girilmez veya **vbBinaryCompare** girilirse küçük büyük harf duyarlılığı gözetilerek arama yapılır, **vbTextCompare** girilirse küçük/büyük harf duyarlılığı olmadan arama yapılır. İstisna: ş ve Ş, ç ve Ç v.s aynı kabul edilirken bir tek İ ve i farklı algılanır.

Bununla beraber bu metodu, aranan metnin tam eşleşme durumu için kullanamazsınız. Yani "is" metnini aradığınızda "is, istanbul, mistik" kelimelerinin tamamı gelir.

Şimdi basit bir örnekle konuyu anlamaya çalışalım. Aşağıdaki dizide içinde "kan" geçen isimleri filtrelemeye çalışıyoruz.

Sub basitfiltre()

Dim isimler As Variant

Dim filtreli As Variant

isimler = Array("volkan", "erhan", "hakan", "özkan", "meltem", "serkan")

filtreli = Filter(isimler, "kan", True, vbTextCompare)

For Each f In filtreli

Debug.Print f

Next f  
End Sub

Aşağıdaki örnekte ise içinde "ankara" geçen tüm kelimeler aranır ve filtrelenir. Yalnız bu sefer kaynak dizimiz 2 boyutlu ve önce onu tek boyutlu diziye çevirmemiz gerekiyor.

Sub filtreornek()

Dim şubeler As Variant

Dim filtreli As Variant

Dim geçici() As String

Dim i As Integer

'çeşitli şubelerin olduğu alandan okuma yapılır(sırayla ankara, antalya, ankara, istanbul, sivas yazsın)

şubeler = Range("A1: A5").Value

'Variant ve 2 boyutlu olan şubeler dizisi klasik bir boyutlu bir diziye atanır

ReDim geçici(1 To UBound(şubeler, 1))

For i = 1 To UBound(şubeler, 1)

geçici(i) = şubeler(i, 1)

Next i

filtreli = Filter(geçici, "ankara", True, vbTextCompare)

Set Hedef = Range("C1").Resize(UBound(filtreli) + 1, 1)

Hedef.Value = filtreli

End Sub

##### Bir metin bir dizide var mı kontrolü?

Filter fonksiyonunu öğrendiğimize göre şimdi bunu daha spesifik bir amaçla kullanabiliriz: **Bir metnin bir dizide olup olmadığını öğrenmek için**. Bunun için birkaç yöntem bulunuyor.

İlk yöntemde diziye filtre uygular ve elde ettiğimiz yeni dizinin üst limitine bakarız. Eğer üst limit 0 veya daha üstü bir sayı ise aradığımız metin dizide var demektir, -1 ise dizide bulunmuyor demektir.

Sub varmı1sub()

'1.Yöntem: Filter, tam eşleşme aranmıyor

Dim şubeler As Variant

Dim filtreli As Variant

Dim geçici() As String

Dim i As Integer

'çeşitli şubelerin olduğu alandan okuma yapılır(sırayla ankara, antalya, ankara, istanbul,sivas yazsın)

şubeler = Range("A1: A5").Value

'Variant ve 2 boyutlu olan şubeler dizisi klasik bir boyutlu bir diziye atanır

ReDim geçici(1 To UBound(şubeler, 1))

For i = 1 To UBound(şubeler, 1)

geçici(i) = şubeler(i, 1)

Next i

filtreli = Filter(geçici, "ankara", True, vbTextCompare)

If UBound(filtreli)=-1 Then

MsgBox "aradığınız kelime dizide yok"

Else

MsgBox "aradığınız kelime dizide var"

End If

End Sub

Bunu bir fonksiyon haline getirmek daha kullanışlı yapacaktır.

'1'in function hali

Function Varmı1(aranan As String, dizi As Variant) As Boolean

Varmı1 = (UBound(Filter(dizi, aranan, True, vbTextCompare)) > -1)

End Function

'--------

Sub test1()

isimler = Array("volkan", "erhan", "hakan", "özkan", "meltem", "serkan")

Debug.Print Varmı1("volk", isimler) 'True

Debug.Print Varmı1("volkan", isimler) 'True

End Sub

Tabi bu yöntemde tam eşleşme durumu sağlanmıyor, zira ilgili metni içeren bir kelime var mı diye kontrol ediyor. Eğer ki tam eşleşme istiyorsak, **Match** fonksiyonunu kullanmalıyız veya Filter dışında çözüm sağlayan 2.yöntemi denemeliyiz. **Bu yöntemde 1.yöntemin aksine sayısal değerleri de arayabiliyoruz**. Bu yüzden hem sayıları hem stringleri kapsayacak şekilde Variant tipte bir aranan parametresi belirtiyoruz.

Function Varmı2(aranan As Variant, dizi As Variant) As Boolean

'2.yöntem)Match, tam eşleşme sağlanıyor

On Error GoTo hata

Varmı2 = Not IsError(WorksheetFunction.Match(aranan, dizi, 0))

Exit Function

hata:

Varmı2 = False

End Function

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

Sub test2()

isimler = Array("volkan", "erhan", "hakan", "özkan", "meltem", "serkan")

Debug.Print Varmı2("volk", isimler) 'False

Debug.Print Varmı2("volkan", isimler) 'True

sayılar = Array(1, 2, 3)

Debug.Print varmı3(1, sayılar) 'true

Debug.Print varmı3(10, sayılar) 'false

End Sub

3. yöntem, dizi elemanlarını **Join** ile birleştirip, yine bir String fonksiyonu olan ve bir string içinde bir alt metin arayan **InStr** fonksiyonu ile arama yapmak şeklinde olabilir, üstelik bu yöntem az önce belirtildiği gibi tam eşleşme sağlar. Bunda da 2.yöntemde olduğu gibi sayısal değerleri de arayabiliyoruz, o yüzden fonksiyonumuzu Variant tanımlıyoruz.

Şimdi bu örneği direkt Function olarak yapalım.

Function varmı3(aranan As Variant, dizi As Variant) As Boolean

ayraç = "-"

joinli = ayraç & Join(dizi, ayraç) 'tam eşleşmeyi sağlamak için başına da ilgili ayracı koyuyoruz

If InStr(1, joinli, ayraç & aranan & ayraç, vbTextCompare) Then

varmı3 = True

Else

varmı3 = False

End If

End Function

Sub test3()

isimler = Array("volkan", "erhan", "hakan", "özkan", "meltem", "serkan")

Debug.Print varmı3("volk", isimler) 'False

Debug.Print varmı3("volkan", isimler) 'True

sayılar = Array(1, 2, 3)

Debug.Print varmı3(1, sayılar) 'true

Debug.Print varmı3(10, sayılar) 'false

End Sub

4. yöntem olarak, daha basit bir yol olan **Dictionary** kullanılabilir, tabi kodun genel yapısı buna uygun ise. Bunu sonraki sayfalarda göreceğiz.

5.yöntem olarak arama algoritmalarından birini kullanarak kendi fonksiyonunuzu yazmayı önerebilirim ama çok profesyonelleşmek istemediğiniz sürece bu sulara girmenize gerek yoktur.

##### Var mı, varsa kaçta?

Yukarıdaki Match fonksiyonunu kullandığımız 2. yöntemde aslında ilgili metnin kaçıncı sırada olduğunu buldurmaya yarayan bir fonksiyon kullanmış olduk. Eğer bulamazsa hata döndürür, zaten biz de hata döndürüp döndürmediğine bakıyorduk, döndürmüyorsa elemanı içeriyor diyorduk. Peki ya hata döndürmüyorsa? Döndürdüğü şey sıra numarası olmaktadır.

#### Dizileri sıralama

Algoritma kavramına detaylı girdiğimizde karşımıza çıkan en temel algoritmalardan biri de sıralama algoritmalarıdır.(Diğeri de hemen az önce bahsettiğim arama algoritmasıdır).

Bu sıralama algoritmalarını anlamak önemli, zira dizilerle ilgili olarak sıralama yapmayı sağlayan ne yerleşik bir fonksiyon/metod, ne de arama yaparken kullandığımız gibi yardımcı yöntemler(Match, Join) bulunmaktadır. Kendi fonksiyonumuz kendimizin yazması gerekmektedir.

Bunların da kendi içinde türleri vardır. Bubble Sort, Merge Sort, Quick Sort gibi. Burada bunlardan ikisine bakacağız. [https://www.mrExcel.com/forum/Excel-questions/690718-VBA-sort-array-numbers.html](https://www.mrexcel.com/forum/excel-questions/690718-vba-sort-array-numbers.html) ve [https://stackoverflow.com/questions/152319/VBA-array-sort-function](https://stackoverflow.com/questions/152319/vba-array-sort-function) sayfalarında daha bol miktarda örnek bulunmakta, <https://www.geeksforgeeks.org/sorting-algorithms/> ve <http://bilgisayarkavramlari.sadievrenseker.com/2008/08/09/siralama-algoritmalari-sorting-algorithms/> sitelerinde ise muazzam bilgi kaynağı bulunmaktadır. Bu son sitelerden farklı yöntemleri öğrenip VBA kodlamasını yapmayı deneyebilirsiniz.

##### Bubble Sort

Bu yöntem en basit yöntem olup, aynı zamanda en yavaş çalışan yöntemdir de. Gerçi küçük dizilerde bu yavaşlık problemi çok fark edilmez. Çalışma yöntemi, tekrarlı bir şeklide komşu değerlerin sıralamasını yapmaktan ibarettir. Aşağıdaki örnekte tüm aşamaların üzerinden tek tek geçeceğiz.

Elimizdeki dizi 5,1,4,2,8 rakamlarından oluşsun.

1.aşama

Önce                   Sonra        
( **5** **1** 4 2 8 ) –> ( **1** **5** 4 2 8 )  
( 1 **5** **4** 2 8 ) –>  ( 1 **4** **5** 2 8 )  
( 1 4 **5** **2** 8 ) –>  ( 1 4 **2** **5** 8 )  
( 1 4 2 **5** **8** ) –> ( 1 4 2 **5** **8** ) 'burada bir değişiklik olmaz, sırası doğru

2.aşama

( **1** **4** 2 5 8 ) –> ( **1** **4** 2 5 8 )  
( 1 **4** **2** 5 8 ) –> ( 1 **2** **4** 5 8 )  
( 1 2 **4** **5** 8 ) –> ( 1 2 **4** **5** 8 )  
( 1 2 4 **5** **8** ) –>  ( 1 2 4 **5** **8** )  
Şu anda sıralama tamam ancak bu aşamada 1 tane de olsa bir değişim olduğu için algoritmanın devam etmesi lazım, ta ki hiç değişim olmayana kadar.

3.aşama

( **1** **2** 4 5 8 ) –> ( **1** **2** 4 5 8 )  
( 1 **2** **4** 5 8 ) –> ( 1 **2** **4** 5 8 )  
( 1 2 **4** **5** 8 ) –> ( 1 2 **4** **5** 8 )  
( 1 2 4 **5** **8** ) –> ( 1 2 4 **5** **8** )

Evet 3.aşamada artık bir değişiklik yok ve algoritmamız durur. Şimdi bunun kodunu görelim.

Function BubbleSort(arr) As Variant

Dim geçici As Variant

Dim i As Long

Dim j As Long

Dim lngMin As Long

Dim lngMax As Long

lngMin = LBound(arr)

lngMax = UBound(arr)

For i = lngMin To lngMax - 1

For j = i + 1 To lngMax

If arr(i) > arr(j) Then

geçici = arr(i)

arr(i) = arr(j)

arr(j) = geçici

End If

Next j

Next i

BubbleSort = arr

End Function

'-------

Sub bubble\_sort\_test()

Dim dizi As Variant

Dim sıralı As Variant

dizi = Array(5,1,4,2,8)

sıralı = BubbleSort(dizi)

Call Diziyazdır(sıralı)

End Sub

##### Quicksort

QuickSort(Hızlı sıralama) algoritması, böl ve fethet felsefesine dayanan bir yöntem uygular. Bu anlamda Merge Sort algoritmasına benzer.

Bu yöntemde, bir referans seçilir(bu referans en sondaki, en baştaki olabilir, ortadaki veya rasgele olabilir. Buna pivot veya dayanak noktası da denir. Bu referanstan küçük olanlar sol tarafa, büyük olanlar sağ tarafa konulur ve  daha fazla yer değiştirme işlemi yapılamayana kadar bu işlem recursive olarak devam eder. Bu konuda yukarıda bahsettiğim linklerde de bolca açıklama ve örnek bulunmaktadır.

Şimdi, algoritmanın icap ettiği şekilde ben de bir kod hazırladım. Referans noktasını "sondaki eleman" olarak belirledim. İşleyişin aşama aşama nasıl olduğunu kod içinde bulabilirsiniz. Verdiğim linklerde gerekli açıklamalar, videolarla birlikte sunulduğu için ben daha fazla uzatmamak adına ilave açıklamada bulunmayacağım. Detayla ilgilenmiyorsanız kodu doğrudan da kullanabilirsiniz.

Kodumuz iki kısımdan oluşmakta. Önce parçalama kısmı sonra sıralama kısmı. Kodu tek seferde çalıştırıp Immediate Window’da algoritmanın işleyişini aşama aşama takip edebilirsiniz.

Function Parçala(ByRef dizi As Variant, ByVal low As Integer, ByVal high As Integer) As Integer

referans = dizi(high)

Debug.Print "Referansımız: " & referans & ".Bundan küçükler sola, büyükler sağa. referans ortalarına. Soldakiler önce ele alınacak, sağdakiler daha sonra."

i = low - 1

For J = low To high - 1

If dizi(J) <= referans Then

i = i + 1

Debug.Print i, J, dizi(J) & "<= referans olduğu için " & dizi(i) & " ile " & dizi(J) & " yer değiştirecek." & dizi(i) & " yavaşça sağa kayaacak: " & Join(dizi, "-")

geçici = dizi(i) '

dizi(i) = dizi(J)

dizi(J) = geçici

'Call DiziBitişikyazdır(dizi, "yer değişim sonucunda: ")

Else

Debug.Print i, J, dizi(J) & ">" & "referans(" & referans & ") olduğu için yer değişme yok"

End If

Next J

Call DiziBitişikyazdır(dizi, "for çıkışında dizinin durumu: ")

Debug.Print dizi(i + 1) & " ile " & dizi(high) & " yer değiştirecek, yani referans ortaya girecek."

geçici2 = dizi(i + 1)

dizi(i + 1) = dizi(high) 'referansla. yani referansı ortaya koyuyoruz

dizi(high) = geçici2

Call DiziBitişikyazdır(dizi, "Bu aşamanın sonunda dizimiz: ")

Debug.Print "-----------------------------------------" & vbNewLine

Parçala = i + 1

End Function

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

Sub hızlısırala(ByRef dizi As Variant, ByVal low As Integer, ByVal high As Integer)

If low < high Then

Pi = Parçala(dizi, low, high)

hızlısırala dizi, low, Pi - 1

hızlısırala dizi, Pi + 1, high

End If

End Sub

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

'bu kodu çalıştırarak test ediyoruz

Sub quick\_sort\_test()

karışıkdizi = Array(5, 1, 4, 2, 8)

'karışıkdizi = Array(52, 3, 45, 32, 8, 19, 47, 26, 1, 100, 68, 24, 12, 9, 72, 55, 36)

Call DiziBitişikyazdır(karışıkdizi, "İlk giriş: ")

hızlısırala karışıkdizi, 0, UBound(karışıkdizi) 'byref gittiği için sıralanıp gelir

Call DiziBitişikyazdır(karışıkdizi, "Nihai sıralama sonucu: ")

End Sub

#### Diziler için Utility Modülü

Dizilerle ilgili olarak sık yapılan işlemleri, özellikle test/kontrol için yapılan işlemleri bir modül içinde toparlamak iyi bir fikir olacaktır. Bunlardan sıralamayı yukarıda görmüştük, diğerlerine birkaç örneği ise aşağıda bulabilirsiniz.

Sub Diziyazdır(dizi As Variant)

For Each d In dizi

Debug.Print d

Next d

End Sub

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

Sub DiziBitişikyazdır(dizi As Variant, Optional başlık As String = "Birleşen dizi: ")

bitişik = Join(dizi, "-")

Debug.Print başlık & bitişik & vbNewLine

End Sub

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

Function ikidizibirleştir(dizi1 As Variant, dizi2 As Variant)

Dim geçici As Variant

adet = UBound(dizi1) + UBound(dizi2) + 2

ReDim geçici(adet - 1)

For i = LBound(dizi1) To UBound(dizi1)

geçici(i) = dizi1(i)

Next i

For i = UBound(dizi1) + 1 To adet - 1

geçici(i) = dizi2(i - 1 - UBound(dizi1))

Next i

'control için

DiziBitişikyazdır (geçici)

ikidizibirleştir = geçici

End Function

## Collectionlar

Collection'lar, Diziler gibi benzer özellikli öğeleri bir arada tutan yapılardır. Dizilerden farklı olarak, tek bir veri tipleri yoktur, yani içlerinde aynı anda hem sayı hem metin tutabilirler. Ayrıca dizilerdeki boyut yerine sayısal bir **Index** ile metinsel özelliği olan opsiyonel bir **Key** unsurlarına sahiptirler. Dizilerle daha genel bir kıyaslamayı aşağıda bulabilirsiniz.

### Giriş

#### İki tür Collection

Daha derinlere dalmadan, birkaç tanıdık Collection'dan bahsetmekte fayda var. Workbook**s**, Worksheet**s**, Sheet**s** v.s. Bunlar Excel'in **yerel**(built-in) collectionlarıdır. Bunlar VB6 ve VB.Net'teki Collection class'ını baz alırlar ve çok daha fazla üyeye sahiptirler, o yüzden daha kullanışlıdırlar. Bizim burada işleyeceğimiz ise Collection classının VBA uyarlaması olup, VB6/VB.Net versiyonuna göre biraz daha çelimsizdir.

Yerel collection'lar VB'nin Collection sınıfından geldikleri için yereldirler, o yüzden ayrıca tanımlanmazlar, Excel açıldığı anda otomatikman kullanılır haldedirler(Hepsi olmasa da büyük çoğunluğu).

O yüzden şöyle bir kod yazmak çok anlamsızdır.

Dim wsc As New Worksheets

For Each ws in wsc

......

Next ws

Onun yerine Worksheets'i doğrudan kullanırız.

For Each ws in Worksheets

......

Next ws

#### Tanımlama

Şimdi, konumuz olan Collectionlara gelecek olursak bunların tanımlaması ve kullanımı iki şekilde olabilir.

Dim col As Collection 'Tanımlandı

Set col = New Collection 'Hafızada yer ayrıldı

Başka bir tanımlama şekli de tanımlama ve yaratma işleminin tek satırda olduğu şekildir.

Dim col As New Collection

İlk yöntem her zaman en güvenli yöntem olmakla birlikte eski bir bilgisayarınız yoksa performans sorunu yaşamayacağınızı düşünerek ikinci yöntemi de güvenle kullanabilirsiniz. Bir diğer fark da şudur: Eğer yarattığınız collection üzerinde **"Nothing mi?"** kontrolü yapmanız gerekiyorsa ilk yöntemde(o an sadece ilk satırı yazılmışsa) yaratılan Collection bir değer döndürmezken, yani Nothing iken, ikinci yöntemle yaratılan False döndürür. Çünkü ikinci yöntemde New diyerek atamayı da yapmış oluyoruz, ilk yöntemde ise sadece hafızada yer ayırmış olduk, henüz Collection nesnesini yaratmadık. Aşağıda bunu açıklayan bir örnek görebilirsiniz. Konuyla ilgili daha detaylı bilgiye **İleri Seviye Konular>Objeler** **Dünyası** konusunda ulaşabilirsiniz.

Sub newnothingcol()

Dim col1 As Collection

Dim col2 As New Collection

MsgBox "col1'in türü " & TypeName(col1)

MsgBox "col2'nin türü: " & TypeName(col2)

If col1 Is Nothing Then 'buraya girer

MsgBox "col1 hiçbirşeydir"

End If

If col2 Is Nothing Then 'buraya girmez

MsgBox "col2 hiçbirşeydir"

End If

Set col1 = New Collection

MsgBox "col1'in türü: " & TypeName(col1)

End Sub

#### Sınıfın Üyeleri(Property ve Metodlar)

Bu sınıfın yalnızca 4 üyesi vardır. **Add, Remove, Item, Count.**

Üye sayısı az olduğu için bunlara tek tek bakacağız.

##### Add metodu ile yeni eleman eklemek

Üyelerden en sık kullanılanıdır ve ana üyedir diyebiliriz.

Add metodu default olarak, yeni elemanları en sona ekler. Bununla beraber Before/After parametresi ile belirli bir indeksten önce veya sonra da eklenebilir.

Col.Add 10 'İçerdeki elemanlar: 10

Col.Add 20 'Şimdi 10, 20

Col.Add 30, Before: = 1 'Şimdi 30, 10, 20

Excel'in built-in Collectionlarında da yine bu Add metodunu kullandığımızı biliyorsunuz, aşağıdaki gibi:

Workbooks.Add

Worksheets.Add 'bunda After ve Before da kullanılabilir

Ekleme işlemi yapılırken opsiyonel bir parametre olan **Key** de belirtilebilir. Key'in amacı, ilgili elemana indeks yerine ismi ile de ulaşmayı sağlamaktır.

Col.Add 10, "on"

Bu parametre **string bir ifadedir ve benzersiz olmalıdır**. Yani Item olarak 10'u birden fazla ekleyebilirsiniz(niye yapasınız ki 😊), ama bunun key'i olan "on" sadece bir kez geçmeli.

Sub keyvaluepair()

Dim notlar As New Collection

'opsiyonel bir parametre olan Key de Add içine yazılabilir

notlar.Add 67, "Volkan" '67 value, volkan key

notlar.Add 67, "Meltem" 'item 2.kez geçiyor, hata vermez

'notlar.Add 12, "Meltem" 'key 2.kez geçiyor, hata verir

End Sub

Pratikte key kullanımı çok olmayacaktır diye tahmin ediyorum, daha çok Excel’in kendi Collection'larında kullanılır. Workbook veya Worksheetler’de bunu yapıyoruz; Worksheets(1) yerine Worksheets("krediler") yazmak gibi. Bu ikili bilgi ekleme şeklinde programlama literatüründe **Key-Value** pair, yani Anahtar-Değer ikilisi denir. Gerçi Collectionlar’da bunların eklenme şekli biraz terstir; Key-Value değil, Value-Key'dir.

Bununla birlikte, bölge kodu-bölge adı gibi bir ikili bilgiyi(Key-Value) aynı anda eklemek istiyorsanız Collection yerine Dictionary kullanmanızı öneririm. Açıkçası bana yerel Collectionlar’da Key kullanımı anlamlı geliyor ama Collection sınıfında bu özelliği neden koymuşlar anlayamadım, Dictionary bu ihtiyacı oldukça karşılarken!

##### Remove ile elemanları silmek

Eklenen elemanları çıkarmak için **Remove** metodunu kullanırız. Elemanın indeks numarasını(sade veya parantez içinde) veya Key'i belirtmek yeterlidir.

Sub silme()

Dim col As New Collection

col.Add 10, "on"

col.Add 20, "yirmi"

col.Add 30, "otuz"

col.Add 40, "kırk"

col.Remove 4 'indeksle

col.Remove (3) 'indeksle

col.Remove "yirmi" 'key ile

'col.Remove (10) 'hata. value ile silme yoktur

Debug.Print col.Count

End Sub

Collection’daki tüm elemanları silmek yani collection'ı boşaltmak için For Next içinde Remove kullanırız.

For i = coll.Count To 1 Step -1

coll.Remove i

Next i

Collection'ı boşaltmaya benzese de tamamen farklı şey olan iki yöntem daha vardır.

'Collection'ı yeniden tanımlamak.

Set col = New Collection

'veya onu yoketmek.

Set col = Nothing

##### Count

Belirli bir anda Collection'ımız içinde kaç eleman var bunu kontrol etmek istersek **Count** özelliğini kullanırız.

Debug.Print col.Count

Sub collornek()

Dim col As New Collection

col.Add 10, "on"

col.Add 20, "yirmi"

Debug.Print col.Item(2)

Debug.Print col(2)

Debug.Print col("yirmi")

Debug.Print col.Count

col.Remove "yirmi"

Debug.Print col.Count

Set col = Nothing

Debug.Print col.Count

End Sub

#### Elemanlara erişim

Indeks(item no) ile veya doğrudan item numarası ile elemanlara erişebiliriz. Item özelliği Collectionlar için default özellik olduğu için bunu belirtmesek de olur. Bu arada dizilerden farklı olarak Collectionlar’da indeks 0'dan değil 1'den başlar. Daha önce belirttiğimiz gibi elemanlara ismiyle ulaşmak da mümkündür, tabiki eklerken **Key** ile eklemişsek.

Sub erişim()

Dim col As New Collection

col.Add 10, "on"

col.Add 20, "yirmi"

Debug.Print col.Item(2) '20

Debug.Print col(2) '20

Debug.Print col("on") '10

End Sub

##### İlk ve Son üyeye erişmek için

Collectionlar sıralı yapılar oldukları için ilk ve son eleman özel öneme sahiptirler, en azından bazı durumlarda böyledir. İlk elemana ulaşmak basit, indeksi 1 veririz. Son elemana ulaşmak için eleman sayısını saydırır, ve bu sayıyı indeks olarak veririz.

col(1) 'ilk eleman

col(col.count) 'son eleman

Tüm elemanlar üzerinden geçmek için For döngüsü kullanırız. Klasik veya For Each.

'Klasik For

For i = 1 To coll.Count

Debug.Print coll(i)

Next i

'For Each

For Each m In meyveler

Debug.Print m

Next m

Fark ettiyseniz elemanları sadece okuduk, yani onları çağırdık ve değerlerini ekrana yazdırdık. Onlara bir değer atamadık, değerlerini değiştirmedik. Bunun bir sebebi var: **Collectionlar read-onlydir**. Yani eklediğimiz elemanların değerini değiştiremeyiz, onları sadece okuruz. Bu önemli bir dezavantaj gibi görünmekle birlikte kullanım amacına uygun kullandığımızda çok da aradığımız bir özellik değildir. Ama siz ille de değiştirilebilir bir dizi yapısı kurmak istiyorsanız, ya normal dizi ya da Dictionary kullanmalısınız.

NOT: Collectionlar’la hiçbir zaman Key'i elde edemeyiz. Bu ifade, Collectionlar’ın Readonly olmasından farklı bir şeydir. Key'i elemana ulaşmak için kullanırız ama ona değer atayamayız,  değerini de elde edemeyiz. Yani 1.indekste yer alan Item=10, Key="On" olan bir elemana Col(1) diyerek 10 değerine ulaşabilirken "On" değerine hiçbir şekilde ulaşamayız. "On"u sadece bu elemana ulaşırken indeksin alternatifi olarak kullanırız.

##### "Elemanlar arasında X var mı?" kontrolü

Collection sınıfının çok az üyeye sahip olduğunu gördük. Ve bunların arasında da başka dillerde olan **Contains**, **Exists** gibi "Var mı?" kontrolü yapacak bir metodu yok. Bunun yerine bu işi görecek bir fonksiyon yazılmaktadır.

Bu örnekte Item'ın Collection'da olup olmadığına bakıyoruz.

Function ColdaVarmı(col As Collection, kontrol As Variant) As Boolean

On Error Resume Next

ColdaVarmı= False

Dim x As Variant

For Each x In col

If x = kontrol Then

ColdaVarmı= True

Exit Function

End If

Next

End Function

'kullanımı da şöyledir

Sub VarmıCol()

Dim sayılar As New Collection

sayılar.Add 10

sayılar.Add 20

sayılar.Add 30

Debug.Print ColdaVarmı(sayılar, 20) 'True döner

Debug.Print ColdaVarmı(sayılar, 40) 'False döner

End Sub

Item yerine Value değerinin olup olmadığına bakmak içinse şöyle bir fonksiyon yazabiliriz. Bu fonksiyon [http://stackoverflow.com/questions/137845/determining-whether-an-object-is-a-member-of-a-collection-in-VBA](http://stackoverflow.com/questions/137845/determining-whether-an-object-is-a-member-of-a-collection-in-vba) sayfasından alıntıdır. Bu daha pratiktir ve tüm collection'ı dolaşmak zorunda kalmadığı için daha hızlıdır. Ama bunu kullanabilmek için Value özelliğini kullanmanız gerekiyor. Yaptığı iş şudur. Contains'e önce doğrudan True atıyor, sonra obj değişkenine col(key) ile atama yapmaya çalışıyor, sanki varmış gibi. Eğer varsa atar ve fonksiyondan çıkar, Contains'e de en son True atandığı için True döner; obj=col(key) satırı hata alırsa, yani bu key bu Collection içinde yoksa, hata ele alma bloğunda Contains'e False atanır.

Public Function Contains(col As Collection, key As Variant) As Boolean

Dim obj As Variant

On Error GoTo err

Contains = True

obj = col(key)

Exit Function

err:

Contains = False

End Function

'kullanım şekli yukardakine benzer, sadece Value parametresi string olmak zorunda olduğu için stringe çevrilerek yazılır.

Sub VarmıCol()

'kullanımı da şöyledir

Dim sayılar As New Collection

sayılar.Add 10, CStr(10)

sayılar.Add 20, CStr(20)

sayılar.Add 30, CStr(30)

Debug.Print ColdaVarmı(sayılar, CStr(20)) 'True döner

Debug.Print ColdaVarmı(sayılar, CStr(40)) 'False döner

End Sub

#### Collectionları sıralama

Collection sıralama için maalesef hazırda gelen bir sıralama metodu bulunmamaktadır. Bu nedenle aşağıdaki gibi bir Sub prosedür yazarız. Bunu istersek Function olarak yazıp dönen değeri de atayabilirdik.

Sub ColSırala(mycol As Collection)

Dim i As Long, j As Long

Dim geçici As Variant

For i = 1 To mycol.Count - 1

For j = i + 1 To mycol.Count

If LCase(mycol(i)) > LCase(mycol(j)) Then 'büyük küçük harf ayrmı olmasın diye

'küçük elemanı depolayalım

geçici = mycol(j)

'küçük elemanı çıkaralım

mycol.Remove j

'küçük elemanı büyük elemandan önceye yerleştirelim

mycol.Add geçici, geçici, i

End If

Next j

Next i

End Sub

Aşağıdaki örnekle de bir collectionımızı sıralayalım.

Sub ColSıralamaÖrneği()

Dim col As New Collection

col.Add "volkan"

col.Add "meltem"

col.Add "doruk"

col.Add "doga"

Call ColSırala(col)

'Sıralama sonrasında yazıralım

For i = 1 To col.Count

Debug.Print col(i)

Next i

End Sub

Tabi sayılardan oluşan bir collection'ı sıralamak istediğinizde yukarıdaki fonksiyonu biraz değiştirmek gerekir. Aşağıdaki değişen bloğu görebilirsiniz. İlk olarak LCase dersek sayıları metin gibi algılar ve 150yi 20'nin öncesine koyabilir, zira ilki "1" ile ikincisi "2" ile başlıyordur. Bir diğer değişiklik de Key kısmında "k" şeklinde bir prefix(önek) ekleriz, zira bu parametre String olmalıdır.

If mycol(i) > mycol(j) Then 'Lcase kullanılmaz

geçici = mycol(j)

mycol.Remove j

mycol.Add geçici, "k" & geçici, i 'k prefixi eklenir

End If

Bu arada siz de Diziler konusunda gördüğümüz diğer sıralama algoritmalarını Collectionlara uyarlamaya çalışabilirsiniz.

#### Dizi ve Collection karşılaştırması

Dizilerde Index 0'dan da 1'den de başlayabilir, Collectionlar’da her zaman 1'den başlar.

Dizilerin aksine Collectionlar’da yeni eleman ekleme ve çıkarmaoldukça basittir, herhangi bir boyut, index v.s belirtmeden eleman eklenebilmektedir.

Diziler genellikle belirli bir boyuta sahiptirler, bu ya statik dizlerdeki gibi baştan belirlenir veya dinamik dizilerdeki gibi sonradan belirlenir, her halükarda genelde dizinin boyutu sabittir. Tabiki dinamik dizilerde **ReDim Preserve** deyimi kullanılarak dizinin içeriği korunacak şekilde boyut artırılabilir ancak bu birkaç kez kullanılırsa verimsiz bir yöntem olur. İşte böyle durumlarda collection kullanmak daha mantıklıdır, zira Collectionlar’da boyut diye birşey yoktur.

Collectionlar, eleman sayısı artınca hantallaşır ve performans kayıpları yaşanabilir. (Binlerce elemandan bahsediyorum tabi)

Collectionlar read-only'dir. Yani eğer elemanları değiştirmeyi düşünüyorsanız veya en azından böyle bir ihtimal varsa Collection yerine dizi kullanmalısınız.

**Genelleme yapacak olursak, boyut baştan bilinmiyor ve çok sık değişecekse Collection kullanmak gerekirken, boyut baştan (veya ilerde bir noktada) biliniyor ve sabit ise dizi kullanmak daha doğru bir yol olacaktır .**

#### Dictionary ile karşılaştırma

Dizilerle kıyastan ziyade, Collectionları aslında Dictionary'ler ile kıyaslamak daha mantıklıdır. Dictionary'lerin çok esnek yapıları olmasına rağmen neden Collection kullanalım ki? diye soranlar için bir karşılaştırma listesi Dictionary bölümünde yer almaktadır.

#### Bir örnek

İşyerinde, sık güncelleme yaptığım bazı Access dosyalarım bulunmakta. Bilen bilir, Access dosyalarında tablolara yeni alan ekledikçe boyut büyür, üstelik eskisini silip yenisini yükleseniz bile. İçerdeki data büyümediği halde boyut büyümesini engellemek için Compact işleminin yapılması gerekir. Bunlar da büyük dosyalar için vakit alan işlemlerdir. Benim de Application konusunda ele aldığım gibi Schedule edilmiş bir sürü rutinim var. Onlardan biri de bu accessleri compact edip boyutlarını küçülten bir makro. Schedule ediyorum, çünkü neden bu iş gece olabilecekken ben PC başındayken olsun ki!

Sub accessleri\_compact()

'tüm dosyaların kapalı olması lazım, çünkü exclusively açılıyor: Bu işi acceslerin içine timer koyarak hallettim

On Error Resume Next

Dim app As Object

Dim DBler As New Collection 'dosyalara tek tek aynı işlemi yapmamak için bir collectiona atayacağız

Dim çalıştımı As Boolean

Dim IMsgFilter As Long 'ole mesajını yoketmek için. Buna takılmayın şimdi.

CoRegisterMessageFilter 0&, IMsgFilter 'ole mesajını yokediyoruz

adet = 0

Set app = CreateObject("Access.Application")

'dosya isimlerini değiştirirerek veriyorum

DBler.Add ("C: \..........accdb")

DBler.Add ("C: \..........accdb")

DBler.Add ("C: \..........accdb")

DBler.Add ("C: \..........accdb")

'şimdi de collection içinde geziniyor ve her eleman için aynı işlemi yapıyoruz

For Each d In DBler

cmp = Left(d, Len(d) - 6) & "\_cmp.accdb"

okmi = app.CompactRepair(d, cmp, False) 'boolean döndürdüğü için böyle yapıyoruz

If okmi = True Then 'başarılı şekilde compact olduysa

If FileLen(d) = FileLen(cmp) Then 'eğer compact sonucunda dosya daha da küçülmediyse, boşuna işlem yapmaya gerek yok,

'sadece yeni üretilen dosyayı silelim, böylece dosyamızın son erişim tarihini de değiştirmemiş oluruz

Kill cmp

Else

Kill d 'orijinal dosyayı siliyoruz

Name cmp As d 'Kompakt edilen dosyayı orijinal ismi ile rename ediyoruz

adet = adet + 1

End If

End If

Next d

Set app = Nothing

If adet > 0 Then

rapor = "access compact " & adet & " out of " & DBler.Count

alici = "12345" 'benim sicilim ve aynı zamanda mail adresim

Call Mailat2(rapor, alici) ' bu kendime bilgi maili gönderen bir başka prosedür

End If

CoRegisterMessageFilter IMsgFilter, IMsgFilter 'ole mesajını restore ediyoruz

Exit Sub

hata:

CoRegisterMessageFilter IMsgFilter, IMsgFilter 'ole mesajını restore ediyoruz

rapor = "accesler compact"

alici = "12345"

Application.Run "Personal.xlsb!mailnogo", rapor, alici

End Sub

### İleri konular

#### Collectionları prosedürlere parametre olarak göndermek

Tıpkı dizilerde olduğu gibi Collection'ları da parametre olarak başka bir prosedüre gönderebiliriz. Dizilerdeki örneği buraya da uyarlayabiliriz.

Sub colgonder()

Dim col As New Collection

col.Add "volkan"

col.Add "meltem"

Call Mesajver(col)

End Sub

Sub Mesajver(coll As Collection)

MsgBox "Bu Collection’da " & coll.Count & " adet elaman var"

End Sub

Bu yöntemin güzel bir örneği de hemen bir alttaki kısımda yer almaktadır.

#### Collectionları Dizilere dönüştürme

Bazı durumlarda elde ettiğimiz Collection'ı, Dizi özelliklerinden faydalanmak veya parametre olarak Dizi alan bir fonksiyonda kullanmak için diziye çevirmemiz gerekir. Aşağıdaki fonksiyon bu işi yapmaktadır.

Function CollectionToArray(col As Collection) As Variant()

Dim arr() As Variant, i As Long, t As Variant

'Collection’daki elemans sayısından 1 çıkarıp dizi boyutunu belirliyoruz

ReDim arr(col.Count - 1) As Variant

'tüm elemanlar tek te diziye atanır

For Each t In col

arr(i) = t

i = i + 1

Next t

CollectionToArray = arr

End Function

'Aşağıda da kullanım örneği bulunuyor

Sub TestCollectionToArray()

Dim sayıcol As Collection, sayıdizi() as Variant

Set sayıcol = New Collection

sayıcol.Add 10

sayıcol.Add 20

sayıcol.Add 30

sayıdizi= CollectionToArray(sayıcol)

Debug.Print UBound(sayıdizi) '2(indeks 0dan başladığı için)

End Sub

#### Collection Collectionı(İçiçe Collection)

İçiçe dizilerde böyle bir kullanım şeklinin amacını ve yöntemini görmüştük. Henüz bakmadıysanız oraya bakmanızı tavsiye ederim. Benim böyle bir kullanım şekline şimdiye kadar ihtiyacım olmadı, ama yine de sizlerin olabilir diye aşağıya [http://bytecomb.com/collections-of-collections-in-VBA/#jagged-collection](http://bytecomb.com/collections-of-collections-in-vba/#jagged-collection) sitesinden aldığım bir örneği koyuyorum. Sitede belirtildiği gibi, eğer içteki kümeye sık sık eleman ekleyecekseniz içiçe dizi yerine içiçe collection kullanmak daha mantıklıdır. Onun dışında pek bir kullanım farkı yok gibi görünüyor.

Dim cAnimals As New Collection

' Let's add stats on the Cheetah

Dim cCheetah As New Collection

' Easy to add inner collections to the outer collection. Also, cCheetah refers

' to the same collection object as cAnimals(1).

cAnimals.Add cCheetah

' Easy to add items to inner collection.

' Working directly with the cCheetah collection:

For Each vMeasurment In GetMeasurements("Cheetah")

cCheetah.Add vMeasurement

Next

' Working on the same collection by indexing into the outer object

For i = 1 To cAnimals.Count

For j = 1 To cAnimals(i).Count

cAnimals(i)(j) = cAnimals(i)(j) \* dblNormalizingFactor

Next

Next

#### Fonksiyonlarda dönen değer olarak

Bazen bir Collection'ı birkaç farklı prosedür içinde kullanmak gerekebilir. o yüzden bunu bir fonksiyon olarak yazıp, dönen değer olarak da ilgili Collection'ın gelmesini sağlayabiliriz.

Aşağıda benim işte kullandığım bir fonksiyon ve bunun kullanım örneği bulunmakta. Her ay Database şifrelerimizin süresi dolmaktadır ve bu yüzden şifrelerin aylık olarak yenilenmesi gerekmektedir. Benim SQL'lerin çoğu da Excel'e gömülü ve schedule edilmiş durumdalar. Şimdi yaklaşık 50 civarı rapor olduğunu ve bazısında birden çok connection olduğunu düşünecek olursak toplam connection sayısı 100 civarında olduğunu söyleyebilirim. Bunların bazısı 5-10 sn'de çalışırken bazısı 10-15 dk sürebiliyor. Bunların her birinde manuel değişiklik yapmak çooooooook uzun zaman alacaktır, sadece connection sayısı çok olduğu için değil aynı zamanda her manuel değişiklik sonrasında sorguların çalışacak olması ve benim bunların bitmesini beklemem gerektiği için.

İşte bu kod ile, öncelikle dosyaları collection'a atıyorum. Bu collection'ı hem şifre değiştirmede hem de sonrasında değiştirdiğim şifrelerin hepsinin değişip değişmediğini görmek için iki kez kullanıyorum. Bu yüzden bir fonksiyona atamak daha mantıklı oldu. Dosyaları açmadan önce otomatik refresh olmasınlar diye EnableEvents=False yapıyorum, varsa protectionları geçici kaldırıyorum, bu detayları aşağıdaki koda koymadım, sadece kafanızda soru işareti olabilir diye belirtmek istedim. Kodun diğer detaylarını Connection'ları işlediğim sayfada veriyor olacağım.

'fonksiyonu çağırma ve kullanma

Sub collfunc()

Dim files As Collection

Set files = dosyacoll()

For Each file In files

Debug.Print file

Next file

End Sub

'Fonksiyonun kendisi

Function dosyacoll() As Collection

Const gunlukyol As String = "…………."

Const haftalıkyol As String = "…………."

Const bbsp As String = "…………."

Const hgtakip As String = "…………."

Dim files As Collection

Dim DBtür As Byte

DBtür = Application.InputBox("DB türünü girin. Oracle 1.grup için 1, 2.grup için 2, DB2 için 3", Type: =1)

'collecitionı oluşturalım

Set files = New Collection

With files

If DBtür = 1 Then

.Add (hgtakip + "Miy access data.xlsb")

'Diğer 20 küsur rapor

'.....

ElseIf DBtür = 2 Then

'10 küsur rapor

'.....

ElseIf DBtür = 3 Then

'10 küsur rapor

'.....

Else

MsgBox "yanlış DB türü girdiniz"

Application.EnableEvents = True

Exit Function

End If

End With

Set dosyacoll = files

End Function

## Dictionary

Dizilerden sonra en çok kullanılan toplu değer saklama aracı Dictionaryler’dir. Collectionlar’a benzerler, onlar gibi değerleri **Key/Value** ikilisi şeklinde tutarlar. Gerçi Collectionlar’da ikili yapı yerine tekli yapı kullanımı daha yaygındır; ikili yapı gerektiğinde çoğunlukla Dictionary kullanılır. Bu kadar çok benzerlik gösterdikleri için bu sayfa boyunca yeri geldikçe Collectionlar’la olan benzerlik ve farklılıklara değinilecek, en aşağıda da genel bir karşılaştırma yapılacaktır.

### Giriş

#### Ne, nasıl, nerede

Adı üzerinde, bu yapıları bir sözlük gibi kullanırız. Mesela çeşitli kategorideki kelimeler için Türkçe-İngilizce karşılıkları şeklinde bir liste oluşturabiliriz, veya bölge kodu-bölge adı gibi lookup listeleri oluşturabiliriz.

Key, Item

---- ----

bir, one

iki, two

.......

veya

5001, Akdeniz

5002, Başkent

.....

Diğer örnekleri şöyle sıralayabiliriz:

* Key: Ürün kodu, Item: ürün açıklaması(ürün yerine şube, bölge, stok v.s birçok şey konulabilir)
* Key: Ürün kodu, Item: ürün alış/satış fiyatı
* Key: Ürün kodu, Item: ürün cirosu/stok adedi v.s
* Key: Müşteri kodu, Item: Telefon no
* Key: Kısaltma, Item: Kısaltmanın uzun/açık hali(CHP, Cumhuriyet Halk Partisi gibi)

vb.

#### Tanımlama şekilleri

Collectionlar’dan farklı olarak Dictionaryler’i kullanmak için projemize **Scripting.Runtime** kütüphanesini(Library) eklemek gerekir. (**Tools>References>Microsoft Scripting Runtime**)

Dictionaryler, birçok object türü gibi hem Late hem Early binding şeklinde tanımlanabilirler. (Early ve Late binding hakkında detay bilgi ve birbirlerine göre avantaj/dezavantajları için **İleri Seviye Konular>Objeler Dünyası** konusuna bakınız)

'Early binding olarak tanımlanırsa

Dim sayılar As New Scripting.Dictionary

'veya

Dim sayılar As Scripting.Dictionary

Set sayılar = New Scripting.Dictionary

'Late binding olarak tanımlanırsa

Dim sayılar As Object

Set sayılar = CreateObject("Scripting.Dictionary")

Early Binding'in tek satır ve iki satır versiyonu arasındaki farkı Collectionlar’da anlattığımız için burada tekrar aynı detaya girmiyorum.

#### Property ve Metodları

##### Eleman ekleme

Collectionlar’da olduğu gibi eleman eklemek için **Add** metodu kullanılır. Dictionaryler’de zorunlu olan iki parametre vardır, bunlar sırasıyla şöyledir: **Key** ve **Item**. (Collectionlar’da Key opsiyoneldir)

Bunların ikisi de Variant tiptedirler, yani her değeri taşıyabilirler, buna Dictionary dahil, yani Dictionary Dictionary'si diye bir kavram teknik olarak mümkündür ve aşağıda da bir örneğini yapıyor olacağız. (Collectionlar’da Key'ler string olmak zorundaydı)

Şimdi iki basit örnek yapalım, ilki sayıların Türkçe-İngilizce karşılığı, ikincisi de lookup liste olarak, bölge kodları ve bölge adları olsun.

Dim sayılar As New Scripting.Dictionary

sayılar.Add "bir", "one"

sayılar.Add "iki", "two"

Debug.Print sayılar("bir") 'one yazar

İkinci örneğimiz de şöyle:

Dim bölgeler As New Scripting.Dictionary

bölgeler.Add 5001, "Akdeniz"

bölgeler.Add 5002, "Başkent"

Debug.Print bölgeler(5001)

**Key** parametresi Collectionlar’da olduğu gibi benzersiz bir parametre olmalıdır, yani sadece bir kere kullanılmalıdır. Collectionlar’da vurgu Item'dadır, yanisözel ifade etmek gerekirsebiz **Item'**ı depolarız ve ona istenirse Key adında bir isim veririz. Dictionaryler’de ise vurgu Key'dedir, Key ile ona karşılık gelen yani onun lookup'ı olan Item birlikte depolanır.

Key de Item de Variant tipli parametrelerdir demiştik ancak Key'in bir istisnası var, dizi(array) olamaz. Item ise array dahil herşey olabilir.

**Item** **propertysi/özelliği**: Collectionlar’dan farklı olarak Dictionaryler’de eleman eklemenin farklı bir yolu daha vardır: Doğrudan atama yöntemi(Implicit adding). Eğer Dictionary içinde olmayan bir anahtara atama yapılmaya çalışılırsa onu doğrudan eklemiş oluruz. Bunun için Item property'sini kullanırız, veya bu property düşürülerek de yazılabilir.

'madenler isimli dict içinde şuan sadece altın ve gümüş var olsun

madenler.Item("diamond")="elmas" 'madenler.Add "diamond", "elmas" ile aynı

Bu özellik, parametre olarak Key'i alır. Yani Key'i "diamond", Item'ı "elmas"tır. Mesela bu elemanda Item'ın baş harfini büyük olarak değiştirmek için şu kodu yazarız.

madenler.Item("diamond")="Elmas"

'veya kısaca

madenler("diamond")="Elmas"

ÖNEMLİ: Collectionlar’daki aynı mantıkla, elemanlara ters yönden erişim mümkün değildir. Yani nasıl ki Collectionlar’da Key'ler benzersizdir ve Item belirtilerek Key'e ulaşamıyorduk, Dictionaryler’de de Item belirterek Key'lere ulaşamayız.

##### Exists metodu ile "Varmı" kontrolü

Bir Key'in Dictionary'de olup olmadığını **Exists** metodu ile kontrol ederiz. Varsa True, yoksa False döndürür. Genel kullanımı "Eleman yoksa onu ekle" şeklindedir.

If Not dict.Exists("elma") Then dict.Add "elma", "apple"

Daha sadece olarak şöyle de diyebilirdik. Zira bu yöntemle, dictionary içinde "olmayan" bir elemanı doğrudan dictionary'ye ekliyorduk.

dict("elma")= "apple"

Ama "Varmı" kontrolünü yaptığımız sırada başka işlemler de yapmamız gerekirse Exists kullanmalıyız.

Collectionlar’da doğrudan böyle bir metod yoktur. Bu işlem, hata kontrolü ile dolaylı olarak yapılmaktadır.

##### Items metodu

Bu metod, Dictionary içindeki tüm itemları döndürür ve bunu 0 tabanlı bir dizi olarak depolar.

meyveler=dict.Items 'elemanları diziye atadık

Debug.Print Join(meyveler,"-") 'dizideki elemanları - ile birleştirdik

Döngüyle tamamına erişebiliriz.

For Each i In dict.Items

Debug.Print i

Next i

Herhangi bir indeksteki Item'a ulaşmak için de kullanılır.

Debug.Print dict.Items(0)

##### Key propertysi

Dictionary içindeki belli bir Key'in değerini değiştirmek için kullanılır.(Key'in karşılık geldiği Item'ı değil. Bunun için Item özelliğini kullanıyoruz)

dict.Add "elma", "apple"

dict.Key("elma") = "Elma"

Bu yukardaki gibi bir örnekteki tekil bir elemanı değiştirmekten ziyade tüm Key'lerin önüne sabit bir ifade eklemek gibi çoklu değişiklikler yapılması daha olasıdır, ki bunu da döngü ile yaparız.

For Each k In dict.Keys

dict.Key(k) = "M\_" & k 'tüm meyvererin önünde Meyvenin M'si ve \_ kodyduk

Next k

Elemanlara erişimi ve döngü detaylarını aşağıda göreceğiz.

Dikkat: Bu özellik Write-Only olup sadece Key'in değerini değiştirmek için kullanılır, Item'ı elde etmek için kullanılmaz. Item'ı elde etmek için nasıl kullanacağımıza elemanlara erişim bölümünden bakabilirsiniz.

##### Keys metodu

Dictionary içindeki tüm Keyleri döndürür ve bunu 0 tabanlı bir dizi olarak depolar.

meyveler=dict.Keys

Debug.Print Join(meyveler,"-")

Herhangi bir indeksteki Key'e ulaşmak için de kullanılır.

Debug.Print dict.Keys(0) 'kısaca dict(0) da yazılabilir

##### Remove ve RemoveAll ile eleman çıkarma

Belirtilen key'deki elemanı çıkarmak için **Remove**, tüm elamanları çıkarmak için yani Dictionary'yi boşaltmak için **RemoveAll** metodunu kullanırız.

meyveler.Remove "elma"

meyveler.RemoveAll

Collectionlar’da RemoveAll yoktu, bunun yerine tüm Collection içinde dolaşıp elemanları tek tek kaldırmak gerekiyordu veya yeni Collection atama veya Nothing ataması yapmak gibi dolaylı yollara başvuruluyordu.

**NOT**: Nothing ataması veya New Dictionary ataması da ilgili dictionary'nin içini boşaltır.

Set meyveler = Nothing

Set meyveler = New Dictionary

RemoveAll ve New Dictionary arasındaki ayrımı görmek için **PivotTable, Slicer ve Chartlar** bölümündeki Pivot Tablolarla ilgili kısımda "Birden çok fieldda filtre uygulama" başlığı altındaki örneği inceleyin.

##### CompareMode ile küçük/büyük harf duyarlılığı

Dictionaryler default olarak küçük/büyük harf ayrımına duyarlıdır(case-sensitive'dir), yani "elma" ve "Elma" farklı olarak algılanır. Bu duyarlılığı kaldırmak için aşağıdaki kod yazılır.

Dim dict As New Dictionary

dict.CompareMode = vbTextCompare 'veya numerik değer olarak 1

'tekrar case sensitive yapmak için şöyle yazılır

dict.CompareMode = vbBinaryCompare 'veya 0

##### Count ile eleman saymak

Collection’daki gibi içerdeki toplam eleman sayısını verir.

Debug.Print dict.Count

#### Elemenlara tekil erişim ve Dictionary içinde dolaşma

Itemlara erişim ile Key'lere erişim bazen kafa karıştırıcı olabilmektedir. Bu kısımda bunların detaylarına değinmeye çalışacağım.

Collectionlar’da olduğunun aksine Dictionary'lere doğrudan Index numarası ile ulaşılmaz, zira Index diye birşey yoktur. Hatırlayacak olursak Collectionlar sıralı bir yapıya sahipken Dictionary'lerde sıra yoktur. Sonuç olarak, Dictionary'lerde elemanlara erişim onun Key'i aracılığı ile olur, bu da ya **Indexli Key** belirterek ya da doğrudan **Key'in kendisi**(Stringse) yazılarak olmaktadır. Indexli Key ile ulaştığımız şey Key'in kendisi iken, Key'in kendisini yazarak eriştiğimiz şey ise bu Key'in lookup değeridir. Collectionlar’da Index ile Item'a ulaşıyorduk, ki buna Key adı verilerek de ulaşılabilir demiştik, Dictionaryler’de ise Indexli key vererek Item'a ulaşıyoruz.

Dİkkat: Key ismini **Key** propertysi ile kullanamayız. Zira bu property write-only'dir, yani sadece Key'in değerini değiştirmek için kullanılır.

Farkındayım, bütün bunlar şuan çok karışık geliyor olabilir. Aşağıdaki örnekler biraz daha aydınlanmanıza yarayacaktır. Biraz aşağıda tüm bunları derleyip toplayan bir örnek ve bir tablo daha göreceksiniz. Ondan sonra kendi örneklerinizi de yapınca konu iyice pekişecektir.

Dim dict As New Dictionary

dict.Add "elma", "apple"

'Tekil elemana read erişimi

Debug.Print dict("elma") 'Key'in kendisi ile Itema erişim. apple değerini verir

Debug.Print dict.Keys(0) 'Indeksli Key ile Key'e erişim. elma değerini verir

Debug.Print dict.Key("elma") 'Hata verir. Çünkü Key propertysi write-onlydir.

'Tekil elemanlara write erişimi

dict.Key("elma")="Elma" 'Key'in değeri değişti, onun lookupı olan Itemın değil. Yani elme Elme oldu.

dict("elma")="apple" 'Itemın değerini değişti. Eleman yoksa Implicit ekleme olur. Yani elma, apple ikilisi eklenir

dict.Item("apple")="Apple" 'Item'ın değerini değiştirdik.

Aşağıda ise erişim yöntemleriyle ilgili küçük bir collection/dictionary karşılaştırması bulunuyor.

Dim col As New Collection

Dim dict As New Scripting.Dictionary

'Collection örneği

col.Add "Elma" '1.index

col.Add "Armut" '2.index

col.Add "Erik" '3.index

Debug.Print col(2) 'veya col.Item(2). Armut yazar

'Key'li Coll örneği. ilk yazılan Item, ikincisi Key'dir

col.Add "Elma", "Apple"

col.Add "Armut", "Pear"

col.Add "Erik", "Plum"

Debug.Print col("Plum") 'Erik yazar

Debug.Print col("Erik") 'Hata verir. Erişim Item'la olmaz,

'Dictionary örneği. ilk yazılan Key, ikincisi Item'dır

dict.Add "Elma", "Apple"

dict.Add "Armut", "Pear"

dict.Add "Erik", "Plum"

Debug.Print dict("Armut") 'Pear yazar

Debug.Print dict.Key("Armut") 'Hata verir. Çünkü Key özelliği Write-Only

##### Tüm elemanları dolaşma

Dictionary'lerin tüm elemanları dolaşmak için tüm dizimsi yapılarda olduğu gibi For Next döngülerini kullanıyoruz.

For Each döngüsünü hem Early Binding hem Late Binding için kullanabilirken, For Next döngüsünü sadece Early Binding tanımlama yapıldığında kullanabiliriz.(Binding çeşitleri hakkında bilgi için **İleri Seviye Konular>Objeler dünyası** konusuna bakınız)

Dim k As Variant

For Each k In dict.Keys

Debug.Print k, dict(k)

Next k

**For Each k In dict.Keys** satırını **For Each k In dict** şeklinde de yazabilirdik. Yani sadece Dictionary'nin adını yazmak onun Keys propertysine bakacağız diye algılanır. Collectionlar’da böyle bir yazımla Items kastedilir.

Klasik For döngüsünde ise dikkat edilecek husus, başlangıcın 0'dan başlaması, bitiş indeksinin de eleman sayısı - 1 olmasıdır.

For i=0 To dict.Count-1

Debug.Print dict.Keys(i), dict.Items(i)

Next i

Şimdi başka bir örneğe bakalım

Dim dict As New Scripting.Dictionary

dict.Add Key: ="Apple", Item: ="Elma"

dict.Add Key: ="Orange", Item: ="Portakal"

dict.Add Key: ="Plum", Item: ="Erik"

'Keylerin değerini değiştirebiliyoruz, yukardaki örnekte tek elemanın değerini

'değiştirmiştik, şimdi tüm elemanların başına "A\_" koyuyoruz.

For Each k In dict.Keys

dict.Key(k) = "A\_" & k

Next k

'tüm keyler ve bunların lookup değeri olan itemları yazdırıyoruz

For Each k In dict.Keys

Debug.Print k, dict(k)

Next k

Şimdi bir de iki kolonlu bir listeyi döngüsel olarak Dictionary'ye ekleyelim, ama eklerken Varmı diye de kontrol edelim. Bu liste, Şube kodu ve şube adından oluşan bir liste olabilir.

Dim dict As Object 'Late binding ile yaratıyoruz

Set dict = CreateObject("Scripting.Dictionary")

Dim anahtar, deger 'tip belirtmeye gerek yok, Varianttırlar

Do

anahtar = ActiveCell.Value

deger = ActiveCell.Offset(0, 1).Value

If Not dict.Exists(anahtar) Then

dict.Add anahtar, deger

End If

ActiveCell.Offset(1, 0).Select

Loop Until IsEmpty(ActiveCell)

Debug.Print dict.Count

#### Key, Keys, Item ve Items birlikte kullanımı

Aşağıdaki iki örnek ile tüm bu öğrendiklerimizi pekiştirelim.

Sub foreach\_in\_dict()

Dim madenler As New Scripting.Dictionary

madenler.Add "gold", "altın"

madenler.Add "iron", "demir"

madenler.Item("diamond") = "elmas" 'madenler.Add "diamond", "elmas" ile aynı

madenler("cupper") = "bakır" 'üsttekinin kısa yöntemi

Debug.Print "--------sadece key---------"

For Each K In madenler.Keys 'Keys yazmasak da olur

Debug.Print K

Next K

Debug.Print "--------sadece item---------"

For Each i In madenler.Items

Debug.Print i

Next i

Debug.Print "------Keysden giderek ikili yazım----------"

For Each K In madenler.Keys

Debug.Print K, madenler(K)

Next K

Debug.Print "--------Items'dan giderek ikili yazım olmaz---------"

For Each i In madenler.Items

Debug.Print i, madenler.Item(i) 'Item belirterek key'lere ulaşılamaz

Next i

End Sub

Bu örnek ise pekiştirme için çok daha kuvvetli bir örnek.

Sub key\_değiştirme()

Dim iller As New Dictionary

iller.Add "01", "adana"

iller.Add "02", "adıyaman"

iller.Add "03", "afyon"

iller.Item(0) = "Adana" 'Key'i 0, Item'ı Adana olan yeni bir eleman ekler

iller.Item(0) = "İstanbul" 'Var olan bir kayıt olduğu için update yapar

iller.Items(0) = "hey" 'Items ile atama değil okuma yapılır, bu satır etkisizdir

Debug.Print iller.Item(0) 'Key'i 0 olan item okunur, yani İstanbul

Debug.Print iller.Items(0) 'Item indeksi 0 olan kayıt okunur

'iller.Key("06") = "Ankara" 'hata: olmayan bir key'e erişemeyiz

iller.Key("01") = "001" 'Key'in kendisini değiştirdik, write-only

iller.Keys(0) = "hey" 'item gibi etkisiz. keys sadece okumada kullanılır

Debug.Print iller.Keys(0) 'hey değil 001 yazar

'Debug.Print iller.Key(0) 'hata alrız, çünkü write-only

'şimdi baştan bir dolaşaım

Debug.Print "--------değişm öncesi------"

For Each K In iller.Keys

Debug.Print K, iller(K)

Next K

'şimdi de hem keyleri hem itemları aynı anda değiştirelim

For Each K In iller.Keys

YK = Val(K)

iller.Key(K) = YK

iller.Item(YK) = UCase(iller.Item(YK))

Next K

'şimdi değişiklikleri kontrol edelim

Debug.Print "--------değişim sonrası------"

For Each K In iller.Keys

Debug.Print K, iller(K)

Next K

End Sub

##### Nihai Özet Tablo

Aşağıdaki tablo ile de yukarıdaki örnekleri bir tablo şeklinde görüyoruz.

İller isimli bir dictionary'miz olduğunu ve 01-Adana ile 33-İçel arasındaki kayıtların eklendiğini düşünün. Buna göre;

|  |  |  |  |
| --- | --- | --- | --- |
| Üye | Ekleme | Erişim | Update |
| Item() propertysi | Olmayan kayıt key ile eklenir.Ör: Item(«34»)=«istanbul» | Dict içinde bulunan bir Item, Key kullanılarak okunur. **Erişilen şey Item'dır.** Ör: Item(«34»)-->istanbul | Dict içinde bulunan bir Item, Key kullanılarak değiştirilir. Ör: Item(«34»)=«**İ**stanbul» |
| Item**s**() metod | Ekleme yapılamaz | **Indeks** no ile erişilir. **Erişilen şey Item'dır**. Ör: Items(0)-->adana | Etkisizdir. Update yapılamaz. |
| Key() propertysi **(write-only)** | Ekleme yapılamaz. Kullanılırsa hata alınır | Olmayan key’e erişemeyiz. Olanın ise içeriğini değiştiririz. **Erişilen şey Key'dir.** | Update yapılamaz |
| Key**s**() metod | Ekleme yapılamaz | **Indeks** no ile erişilir. **Erişilen şey Key'dir.** Ör: Keys(0)--> «01» | Update yapılamaz |

### Kıyaslamalar ve ileri örnekler

#### Collectionların ve Dictionarylerin karşılaştırılması

Bu iki yapının benzerlikleri, farklılıkları ve birbirine göre avantaj/dezavantajları bulunmaktadır. "Şu daha iyidir" diye doğrudan bir söylem doğru değildir. Her araçta olduğu gibi, o an ihtiyacımızı en iyi hangisi görüyorsa onu kullanmamız gerekmektedir. Ben burada bir karşılaştırma vereceğim, kararı siz verin. Tabiki benim de naçizane bazı tavsiye ve yönlendirmelerim olmayacak değil.

* Collectionlar VBA içinde yerel olarak bulunurken, Dictionary'leri kullanmak için bunu ya reference olarak eklemeli ya da CreateObject ile Late Binding şekilde yaratmalıyız.
* Dictionaryler’de Key de Item da hem okunabilir hem yazılabilirdir. Ancak Collectionlar’da Item'ın değerini değiştiremezsiniz, yani read-onlydir. Bunu yapmak için önce onu kaldırmalı sonra yeni değerle tekrar eklemeniz gerekir. Ayrıca Collectionlar’da Keyler ne read-only ne write-onyldir, yani değer atanamadıkları gibi elde edilemezler de; sadece ilgili Item'a ulaşmada kullanılırlar.
* Collectionlar sıralıdır, Dictionaryler’de sıra yoktur. Bu yüzden Collectionlar'a indeks numarası ile erişebilirken Dictionaryler’de indeks ile erişilemez. Bununla beraber Dictionaryler’in Keys ve Items metodları bunları 0 tabanlı bir dizi olarak döndürür, yani Key ve Item'lardan oluşan bir dizi 0 indekslidir. Ancak Collectionlar’da indeks 1'den başlar.
* Her iki yapıda da elemanlarda dolaşmak için For Each yapısı kullanılır. Direkt ilgili nesnenin adı verilerek dolaşılmaya çalışıldığında, Collection’da itemlarda dolaşılırken Dictionaryler’de Keylerde dolaşılır.

Collectionları diziye atamak için döngüsel bir yapıyı içeren birkaç satırlık koda ihtiyaç duyulurken Dictionaryler’de **Items** ve **Keys** metodları bize doğrudan dizi verirler.

Dictionaryler’de eleman eklemek için **Add** metoduna ek olarak implicit(üstü kapalı) ekleme yöntemi de varken, Collectionlar’da yalnızca Add metodu kullanılır.

Dictionaryler’de tüm elemanlar tek seferde(**RemoveAll** ile) silinebilirken Collectionlar’da dolaylı yöntemler izlenir.

* Keyler her ikisinde de benzersiz olmalıdır.
* Keyler Dictionary’de zorunlu iken Collection’da seçimliktir.
* Keyler Collectionlar’da String olmalıyken, Dictionaryler’de ise dizi dışında herşey olabilirler.

Dictionaryler’de bir elamanın var olup olmadığı **Exists** metodu ile kontrol edilebilirken Collectionlar’da bu kontrol için birkaç satırlık kod yazmak gerekir.

* Collectionlar küçük/büyük harf ayrımına duyarlı değilken Dictionaryler duyarlıdır, istenirse duyarsız hale getirilebilir.
* Genel olarak bakıldığında, Dictionaryler Collectionlara göre daha hızlıdır

|  |  |  |
| --- | --- | --- |
|  | Dictionary | Collection |
| Parametreler | İkisi de zorunlu | Sadece Item zorunlu |
| Vurgu | **Key**, Item | **Item**, (Key) |
| Erişim | Item(«key adı»)-->Item Keys(indeks)-->Key | İndeks, Key-->Sadece item elde edilir. Key elde edilemez |

Bu farkları bir kısmını küçük bir kodda inceleyelim:

Sub coll\_vs\_dict()

Dim dict As New Dictionary 'library

Dim coll As New Collection 'no library

dict.Add "yüz", "hundred"

coll.Add "hundred", "yüz"

Debug.Print dict.Keys(0), dict.Items(0) 'iki değer de elde edilebilir

Debug.Print coll(1), coll("yüz") 'Key yani "yüz" değeri elde edilemez

dict.Item("yüz") = 100 'key'in lookup değeri olan item'ı değiştirebiliriz

Debug.Print dict.Keys(0), dict.Items(0)

coll(1) = 100 'hata. collectionlar readonlydir, itemlar dğeiştirilemez, keylere zaten ulaşamıyoruz bile

Debug.Print dict(0) 'Dictionary’de indeks yoktur

Debug.Print coll(1) 'Collectionlar’da indeks var ve 1'den başlar

End Sub

Büyük üstat Cpearson der ki:

Her iki obje de benzer datayı gruplamak için çok faydalıdır ancak herşey eşit olduğunda ben Dictionary kullanmayı tercih ederim.

Gerekçe olarak da yukarda belirttiğim maddelerden bazılarını dile getirmiş.

Benim de naçizane bir tavsiyem var: Eğer sadece arka arkaya bir şeyler eklemek istiyorsanız ve kümeden bir şeyler çıkarma veya Varmı kontrolü gibi şeyler yapmayacaksanız Collection kullanın, hem de Key'siz haliyle. Ama bir lookup değeri de olacaksa Key ve Item ikilisine yani bir Dictionary'ye ihtiyacınız var demektir.

#### İçiçe Dictionary(Dictionary of Dictionary)

Dizi Dizisi ve Collection Collection’ı gibi Dictionaryler’in de içiçe geçmiş formları vardır. Hatta yeri gelir, Dictionary of Collection, Collection of Dictionary veya Array of Collection gibi çapraz formlar da kullanmamız gerekebilir.

Benim şahsen çok ihtiyacım olmadı ancak internette bol miktarda örnek bulunmaktadır.

Bununla birlikte gözünüzde canlanması için aşağıdaki gibi bir örnek kod yazabiliriz.

Sub dictofdict()

Dim m As New Scripting.Dictionary

Dim s As New Scripting.Dictionary

Dim dict As New Scripting.Dictionary

m.Add "elma", "apple"

m.Add "erik", "plum"

s.Add "salatalık", "cucumber"

s.Add "domates", "tomato"

dict.Add "meyveler", m

dict.Add "sebzeler", s

Debug.Print dict("meyveler")("elma") 'veya dict("meyveler").Item("elma")

End Sub

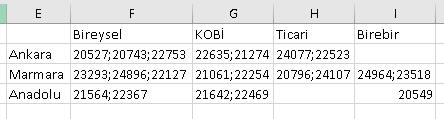
Bu arada  ilk başta kulağa sanki Dictionary of Dictionary ile çözülebilirmiş gibi gelen bir problemi ben aşağıdaki gibi Dictionary ve 3 boyutlu dizi ile hallettim. Biraz üzerine düşününce farklı çözümler de üretilebiliyor. Mesela listenizi farklı bir formata getirip arkasından Dictionary tipli bir dizi ile de sorun çözülebilir.

#### Dictionary ve Dizi bir arada

Aşağıdaki gibi bir listemiz var ve bunu bir alttaki resimdeki hale getirmek istiyoruz. Bu liste hergün güncellenen bir personel dosyası ve bir alttaki hale gelmeli, yoksa burdan beslenen formüllerde hatalar olacağı gibi bölgelere giden otomatik maillerde yanlış kişilere yanlış mailler gidebilir. Böyle bir listenin hergün manuel bir şekilde işlenmesi de oldukça zahmetli olurdu. O yüzden aşağıdaki gibi bir kod yazmalıyız.



Getirmek istediğim hal ise şu. Bu hale geldikten sonra burdan beslenen birçok lookup formülü var.



**İzlenecek yol**: Blg ve Sgm isminde iki Dictionary tanımlarız. Bir de sicilleri atayacağımız bir dizi. Dizimiz çok boyutlu olabileceği gibi içiçe dizi şeklinde de olabilir. Ben burada çok boyutlu dizi yöntemini seçtim.

Sub dictvedizi()

Dim Blg As New Scripting.Dictionary

Dim Sgm As New Scripting.Dictionary

Dim Siciller() As String

ReDim Siciller(0 To 2, 0 To 3, 0 To 3) 'boyutlar sırayla şöyle: bölge sayısı, segment sayısı, kişi sayısı

Set alanBolge = Range(Range("a2"), Range("a2").End(xlDown))

Set alanSegment = Range(Range("c2"), Range("c2").End(xlDown))

i = 0

For Each d In alanBolge

If Not Blg.Exists(d.Value) Then

Blg.Add Key: =d.Value, Item: =i

i = i + 1

End If

Next d

k = 0

For Each d In alanSegment

If Not Sgm.Exists(d.Value) Then

Sgm.Add Key: =d.Value, Item: =k

k = k + 1

End If

Next d

'data okuma

For Each d In alanBolge

Siciller(Blg(d.Value), Sgm(d.Offset(0, 2).Value), dolusay(Siciller, Blg(d.Value), Sgm(d.Offset(0, 2).Value)) + 1) = d.Offset(0, 1).Value

Next d

'hedefe yazma

For i = 0 To Blg.Count - 1

Cells(i + 2, 5).Value = Blg.Keys(i)

Next i

For i = 0 To Sgm.Count - 1

Cells(1, i + 6).Value = Sgm.Keys(i)

Next i

For x = 1 To 4

For y = 1 To 3

Set h = Cells(1 + y, 5 + x)

h.Select

h.Value = sonucgetir(Siciller, Blg(h.Offset(0, -x).Value), Sgm(h.Offset(-y, 0).Value))

Next y

Next x

End Sub

Public Function dolusay(ByVal Data As Variant, ByVal i1 As Integer, ByVal i2 As Integer) As Integer

Dim Count As Integer

Count = 0

For j = 0 To UBound(Data, 3) - 1

If Len(Data(i1, i2, j)) > 0 Then

Count = Count + 1

End If

Next j

dolusay = Count

End Function

Public Function sonucgetir(ByVal Data As Variant, ByVal i1 As Integer, ByVal i2 As Integer) As String

sonucgetir = ""

For i = 0 To UBound(Data, 3)

If Len(Data(i1, i2, i)) > 0 Then

x = Data(i1, i2, i) & ";" & x

sonucgetir = Left(x, Len(x) - 1)

End If

Next i

End Function

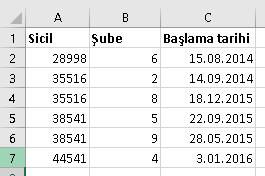
Kodu biraz inceleyin, başka türlü nasıl yapılabilirdi, onu düşünün. Şükür ki, gerek Excel'de gerek VBA'de bir işi yapmanın birden çok yolu olabilmektedir. Artık aklınıza hangisi gelirse onun üzerine yoğunlaşın.

### Çeşitli örnekler

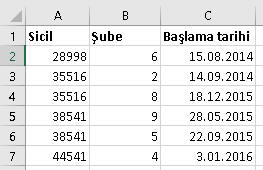
#### Her değer için min/maks değeri alıp depolama

Kişi isimlerinin birkaç kez geçtiği yerde herkesin en büyük satışını aldırma, herkese ait en küçük tarih buldurma gibi örnekler de dictionarylerin pratik uygulamaları arasındadır. Burdaki mantık şu şekilde işler. For döngüsü ile tüm değerler taranır ve sırayla eklenir, ancak ekeme yapılırken Exists ile "daha önce eklenmiş mi kontrolü" yapılır. Tabiki datanın ya manuel ya da kod içinde sıralanmış olması gerekir.

Aşağıdaki örnekte, belirli sicil numaralı kişilerin belirli şubelere başlama tarihleri var. Bir kişi zaman içinde bir şubeden başka şubeye tayin olabilmekte, o yüzden bazı kişilerin birden fazla satırda geçtiğini anlamak zor olmayacaktır. Biz burada bir kişinin bankaya en erken giriş tarihin bulmaya çalışacağız. Ör: 35516 için 14.09.2014 tarihini bulmalıyız, 38541 için de 28.05.2015.



Kodu uzatmamak adına listeyi manuel olarak sıralayalım



Kodumuz ise aşağıdaki gibi olacaktır.

Sub enbuyuktarih()

Dim st As New Scripting.Dictionary

Dim alan As Range, a As Range

Set alan = Range("A2: A7")

For Each a In alan

If Not st.Exists(a.Value2) Then

st.Add a.Value2, a.Offset(0, 2).Value

End If

Next a

For Each s In st

Debug.Print s, st(s)

Next s

End Sub

Eğer ki bu ilk şubenin hangisi olduğunu da öğrenmek isteseydik farklı bişeyler daha yazmamız gerekirdi. Ben şube kodunu Collection'a ekleyerek bulma yöntemini denedim. Dizi kullanarak da çözülebilirdi.

Sub enbuyuktarih2()

Dim st As New Scripting.Dictionary

Dim col As New Collection

Dim alan As Range, a As Range

Set alan = Range("A2: A7")

For Each a In alan

If Not st.Exists(a.Value2) Then

st.Add a.Value2, a.Offset(0, 2).Value

col.Add a.Offset(0, 1).Value, CStr(a.Value) 'Keyler Collectionlar’da string olmalı

End If

Next a

For Each s In st

Debug.Print s, st(s), col(CStr(s))

Next s

End Sub

#### Dictionary tipli bir dizi

Gerçek bir sözlük uygulaması(İngilizce dışındakiler uydurmadır 😊)



Sub arraydict()

Dim dict(2) As New Scripting.Dictionary

dict(0).Add "ekmek", "bread" 'ingilizce

dict(1).Add "ekmek", "brot" 'almanca

dict(2).Add "ekmek", "brotti" 'italyanca

'1000 satır boyunca 1000 kelimeyi okuyup atadık diyelim

'almancada ekmek ne demek

Debug.Print dict(1)("ekmek")

End Sub

Datamızı bu şekle getirdikten sonra bu yukardaki örnekteki ve benzer örneklerdeki sorunu da bu yöntemle çözebiliriz

* Akdeniz bölgesinin Ticari müdürü kim(diğer 3 segment müdürü de var)
* Ankarada Patates fiyatı(diğer 10 sebze fiyatı da var)
* Almanyada erkeklerin yaş ortalaması kaç(kadınların ortalaması da var)
* v.s v.s

Mesela bir üstteki sicil-tarih örneğini de bu yöntemle yazabiliriz.

Sub arraydict2()

Dim dict(1) As New Scripting.Dictionary

Dim alan As Range, a As Range

Set alan = Range("A2: A7")

For Each a In alan

If Not dict(0).Exists(a.Value2) Then 'herhangi birine bakılabilir

dict(0).Add a.Value, a.Offset(0, 1).Value 'şb

dict(1).Add a.Value, a.Offset(0, 2).Value 'tarih

End If

Next a

For Each s In dict(0)

Debug.Print s, dict(0)(s), dict(1)(s)

Next s

End Sub

## Neler Öğrendik

Benzer gruptaki elemanların farklı değişkenler yerine bir dizi veya collection’da nasıl tutulabileceğini öğrendik.

Key-Value çiftlerinin dictionary’lerde nasıl depolanacağını gördük.

Bu veri yapılarının birlikte nasıl kullanılabileceğini öğrendik.