FIZ219_EST_UygulamaNotlari_04_for_while_if

November 10, 2019

1 Uygulama Notları: 4

1.1 FİZ219 - Bilgisayar Programlama I | 10/11/2019

- Listelere ekleme yapmak
- Döngü ve karar prosedürleri
- for.. döngüsü
- while.. döngüsü
- if.. elseif.. else karar mekanizması
- break döngü "kırıcısı"

Emre S. Tasci emre.tasci@hacettepe.edu.tr

1.2 For döngüsü

Grafik çizerken, x değerlerimiz için bir liste oluşturmayı görmüştük. Örneğin, x = [-3,3] aralığı için 0.5 artan şekilde bir listeyi:

[19]:
```
x = -3:0.5:3
```

```
x =

 Columns 1 through 8:

   -3.0000   -2.5000   -2.0000   -1.5000   -1.0000   -0.5000    0.0000    0.5000

 Columns 9 through 13:

    1.0000    1.5000    2.0000    2.5000    3.0000
```

diyerek oluşturabiliyorduk. Aralık büyüklüğü ile uğraşmak istemeyip, doğrudan *"-3 ile 3 arasında 100 noktayı düzenli şekilde alayım"* dediğimizde ise yardımımıza `linspace` komutu yetişiyordu:

[20]:
```
x = linspace(-3,3,100)
```

```
x =
```

```
Columns 1 through 7:

 -3.000000  -2.939394  -2.878788  -2.818182  -2.757576  -2.696970  -2.636364

Columns 8 through 14:

 -2.575758  -2.515152  -2.454545  -2.393939  -2.333333  -2.272727  -2.212121

Columns 15 through 21:

 -2.151515  -2.090909  -2.030303  -1.969697  -1.909091  -1.848485  -1.787879

Columns 22 through 28:

 -1.727273  -1.666667  -1.606061  -1.545455  -1.484848  -1.424242  -1.363636

Columns 29 through 35:

 -1.303030  -1.242424  -1.181818  -1.121212  -1.060606  -1.000000  -0.939394

Columns 36 through 42:

 -0.878788  -0.818182  -0.757576  -0.696970  -0.636364  -0.575758  -0.515152

Columns 43 through 49:

 -0.454545  -0.393939  -0.333333  -0.272727  -0.212121  -0.151515  -0.090909

Columns 50 through 56:

 -0.030303   0.030303   0.090909   0.151515   0.212121   0.272727   0.333333

Columns 57 through 63:

  0.393939   0.454545   0.515152   0.575758   0.636364   0.696970   0.757576

Columns 64 through 70:

  0.818182   0.878788   0.939394   1.000000   1.060606   1.121212   1.181818

Columns 71 through 77:

  1.242424   1.303030   1.363636   1.424242   1.484848   1.545455   1.606061

Columns 78 through 84:

  1.666667   1.727273   1.787879   1.848485   1.909091   1.969697   2.030303
```

```
 Columns 85 through 91:

   2.090909   2.151515   2.212121   2.272727   2.333333   2.393939   2.454545

 Columns 92 through 98:

   2.515152   2.575758   2.636364   2.696970   2.757576   2.818182   2.878788

 Columns 99 and 100:

   2.939394   3.000000
```

for… döngüsü belirlediğimiz bir listenin elemanlarını **sıra ile** ve **tek tek** işleme sokar.

Basit bir örnek olarak 1'den 5'e kadar olan sayıları yazdıralım:

[21]:
```
a = 1
a = 2
a = 3
a = 4
a = 5
```

```
a =  1
a =  2
a =  3
a =  4
a =  5
```

Görüleceği üzere, hemen hemen aynı satırı 5 kere küçük bir değişiklikle yazıp çalıştırıyoruz, ki bu oldukça sıkıcı ve zahmetli. Satırlardaki tek ***değişen*** şeyin a'yı eşitlediğimiz sayı olduğunu görüyoruz. Demek ki `a = i` diye bir şey düşünsek, ve i'nin değerini 1'den başlayıp, her seferinde 1 arttırarak 5'e kadar getirsek işler hayli kolaylaşacak.

i'nin değerinin 1'den 5'e birer birer arttırıldığı halini yapmayı biliyoruz:

[22]:
```
n = 1:5
```

```
n =

   1   2   3   4   5
```

Burada listemize "i" yerine "n" adını verdik çünkü n, i'nin alacağı bütün değerleri barındıran bir kutu; i ise bu değerlerin her biri. for döngüsünü şu şekilde tanımlıyoruz:

for = *yapılacak iş #1… yapılacak iş #2… yapılacak iş #3… …* endfor

for … endfor arasında yer alan ve yapılacak işleri tutan kısma **blok** diyoruz.

Şu halde, a'nın değerini 1 ile 5 arasında değiştirecek ilk döngü örneğimiz şöyle olur:

[23]:
```
for i = 1:5
    a = i
endfor
```

```
a =  1
a =  2
a =  3
a =  4
a =  5
```

Açık açık `1:5` yazmak yerine, önceden tanımlamış olduğumuz ve aynı değerleri tutan n'yi de kullanabilirdik:

[24]:
```
n = 1:5;
for i = n
    a = i
endfor
```

```
a =  1
a =  2
a =  3
a =  4
a =  5
```

(unutmayın ki Octave sembolik bir dil değil, yani n gördüğü yere *n neye karşılık geliyorsa* onu alıp yerleştiriyor: `i = 1:5` yazmakla `i = n` yazmak arasında işlemsel hiçbir fark yok (pratik açıdansa programlarınızı değişkenler üzerinden yazmak yüksek esneklik sağlamakta).

Biraz daha ilginç bir örnek yapalım, örneğin x'in değerini -3'den 3'e 0.5'e kadar arttırıp, her bir değerin karesini s diye bir değişkende toplayalım:

[25]:
```
s = 0; % Değerleri s'de toplayacağımız için ilk olarak s'yi tanımlayıp␣
 ↪sıfırlıyoruz.
xler = -3:0.5:3; % 'xler' ile 'x' arasındaki farka dikkat edin!
for x = xler
    s = s + x*x;
endfor
s
```

```
s =  45.500
```

Başta da belirttiğimiz üzere, for döngüsündeki değerler bir listeden sıralı ve tek tek olarak alınır. Listenin kendisinin sıralı olmasına gerek yok. Sınıftan 5 kişiye akıllarına gelen bir sayıyı söylemelerini isteyelim ve onlar da 5,-3,19,12,-2 sayılarını söylemiş olsunlar. Bir önceki örneği biraz değiştirerek bu sayıların karelerini toplayan bir program yazalım:

[26]:
```
s = 0;
xler = [5,-3,19,12,-2];
for x = xler
    printf("x'in şimdiki değeri: %3d\n",x) % printf() ekrana düzgün bir şekilde
```

```
                                                % yazdırmak için kullanılır,␣
↪yoksayabilirsiniz.
    s = s + x*x;
endfor
s
```

```
x'in şimdiki değeri:   5
x'in şimdiki değeri:  -3
x'in şimdiki değeri:  19
x'in şimdiki değeri:  12
x'in şimdiki değeri:  -2
s =  543
```

Peki ya verdiğimiz sayıların karelerinin toplamını bulmak yerine, her bir sayının karesini bir listede toplamak istersek? Bunu doğrudan liste üzerinde işlem yaparak bulabileceğimizi görmüştük, örneğin:

[27]:
```
xler = [5,-3,19,12,-2];
xkareler = xler.^2
```

```
xkareler =

    25     9   361   144     4
```

Ama şimdi bunu for döngüsü ile gerçekleştirmek istiyoruz (ileride böyle kolaylıkla aritmetik yoldan yapamayacağımız durumlara hazırlık olarak).

Öncesinde, bir listeye değerleri tek tek nasıl toplayacağımızı düşünelim... Mesela, xler listesini tek tek oluşturalım (listenin içine önce 5'i, sonra -3'ü, ardından 19'u, vs. atarak):

[28]:
```
xler = []
xler = [5]
xler = [5 -3]
xler = [5 -3 19]
```

```
xler = [](0x0)
xler =  5
xler =

   5  -3

xler =

    5   -3   19
```

Burada bir duralım zira saçmasapan bir iş yapıyoruz. Tamam, ilk iş olarak, değerleri toplayacağımız kutumuz olan `xleri` boş olarak tanımladık, sonra da ilk eleman 5'i ekledik, bu da tamam, ama daha sonra ikinci eleman -3'ü ve üçüncü eleman 19'u eklerken aslında ekleme yapmayıp, tekrar tekrar

öncekileri yazdık (bir başka deyişle sadece `xler = [5 -3 19]` deseydik de yeterdi – öncekileri yazmaya hiç gerek yoktu)!..

Burada yine Octave'ın sembolik bir dil olmayışının nimetlerinden faydalanalım:

Üçüncü satırda `xler = [5 -3]` derken biz <enter>'a basana kadar (yani o satırı yazdığımız sürece) xler'in değeri [5] – o halde aynı satırı `xler = [xler -3]` olarak da yazabilirdik. <enter>'a basıp, 4. satıra geçtiğimizde xler'in değeri artık [5 -3] olmuş durumda, o halde bu satırı da `xler = [5 -3 19]` yerine `xler = [xler 19]` olarak yazabilirdik. Özetle:

[29]:
```
xler = []
xler = [xler 5]
xler = [xler -3]
xler = [xler 19]
xler = [xler 12]
xler = [xler -2]
```

```
xler = [](0x0)
xler =  5
xler =

   5  -3

xler =

    5   -3   19

xler =

    5   -3   19   12

xler =

    5   -3   19   12   -2
```

Bu yöntem, döngülerde çok işimize yarayacak. Artık verilmiş bir xler listesindeki x değerlerinin karelerini *tek tek* bir başka listede toplayacak bilgimiz var:

[30]:
```
xler = [5,-3,19,12,-2];
xkareler = [];
for x = xler
    xkareler = [xkareler x*x];
endfor
xkareler
```

```
xkareler =

    25     9   361   144     4
```

1.3 while döngüsü

while döngüsü, döngüyü kurarken belirttiğimiz önerme **doğru** (bilgisayar dilinde konuşursak '1') olduğu sürece, döngüyü devam ettirir. Önergenin doğru olması ne anlama geliyor? Birkaç sayısal önerme yazalım:

- 3, 1'den büyüktür: $3 > 1$
- 5, 4'ten küçüktür: $5 < 4$
- 12, 13'ten küçük veya eşittir: $12 <= 13$
- 12, 12'den büyük veya eşittir: $12 >= 12$
- 3, 7'ye eşit değildir: $3 != 7$
- 7, 7'ye eşittir: $7 == 7$
- 3, 5'e eşittir: $3 == 5$

Özellikle son iki önermeye dikkat edin: onlaraa gelinceye kadar, aşağı yukarı matematikte kullandığımız işaretlerin benzerlerini yazdık ama iş "eşittir"e gelince, tek bir "=" işareti yerine, "==" olarak, iki tane yazdık. Bunun sebebi, "=" işaretinin her zaman için **değer atama** anlamına gelmesi. Bu nedenle, "eşittir" önermesini "==" şeklinde, çift "=" işareti ile karşılıyoruz.

Bu önermeleri Octave'a soralım, bakalım ne cevap verecek:

[51]:
```
3 > 1
5 < 4
12 <= 13
12 >= 12
3 != 7
7 == 7
3 == 5
```

```
ans = 1
ans = 0
ans = 1
ans = 1
ans = 1
ans = 1
ans = 0
```

"1 : Doğru" ve "0 : Yanlış" olmak üzere yorumlarsak, cevapları anlamış oluruz.

O zaman -3'ten 3'e kadar olan sayıların karelerin toplamını bir kez daha, bu sefer while döngüsü ile hesaplayalım:

[33]:
```
kareler_toplami = 0;
x = -3; % başlangıç değerimiz
while (x <= 3)
    kareler_toplami = kareler_toplami + x*x;
    x = x + 1;
endwhile
kareler_toplami
```

```
kareler_toplami =  28
```

Bloğumuz, *önermemiz doğru olduğu müddetçe* çalıştırılıyor.

1. while'a ilk gelindiğinde, x'in değeri -3 –> `-3 <= 3` doğru olduğundan blok çalışıyor,
2. değerler yerlerine yerleştirildiğinde `kareler_toplami = 0 + (-3)*(-3)` işleminden 9 oluyor.
3. Sonraki satıra gelindiğinde -3 değerindeki x'in yeni değeri ona 1 eklenerek (-2) yapılıp, döngünün başına gidiliyor.
4. `-2 <= 3` de doğru olduğundan, blok yine çalıştırılıyor.
5. Bu sefer `kareler_toplami = 9 + (-2)*(-2)` işleminden 13 oluyor.
6. Bu şekilde x'in değeri +3'e kadar yükseltiliyor.
7. x = +3 için while'a gelindiğinde önerme hala doğru zira 3, 3'ten küçük veya ona eşit.
8. Blok çalıştırılıyor, `kareler_toplami = 19 + (3)*(3)` işleminden 28 oldu.
9. Bir sonraki satırda x'in değeri 1 eklenerek 4 oluyor, yine while'a gönderiliyor.
10. 4 <= 3 mü? Hayır. O halde döngüye girilmeden döngünün sonuna ışınlanılıyor, burada kareler_toplami'nın yazılması var, onu yazıp, bitiriyoruz.

İlginç bir detay olarak, x'in döngü bittiğindeki değerini düşünelim… 9. adımda 4 olmuştu, bunu teyit etmek istersek:

[34]:
```
x
```

```
x =  4
```

1.4 if.. elseif.. else karar mekanizması

Aslında an itibarı ile karar mekanizması hakkında bir bilgimiz var zira while döngüsünü işlerken, önermenin -hâlâ- doğru olup olmadığına bakarken karar mekanizmasını kullandık. Tek başına hallerde ise karar vermek(/verdirmek) için if.. karar mekanizmasını kullanıyoruz.

[35]:
```
a = 5
if (a == 5)
    disp("a, 5'e eşit")
endif

if(a > 6)
    disp("a, 6'dan büyük")
endif

if(a < 9)
    disp("a, 9'dan küçük")
endif
```

```
a =  5
a, 5'e eşit
a, 9'dan küçük
```

Yukarıdaki kodu çalıştırdığımızda, sadece doğru önermelere sahip if bloklarının işleme konduğunu görüyoruz. Bu örnekte a'nın değerine bağlı olarak üç olasılıktan biri olmak zorunda:

a: * ya 5'e eşit olacak * ya 5'ten büyük olacak * ya da 5'ten küçük olacak

Üç ayrı if bloğu yazmak yerine, bunları tek bir if bloğunda toplayabiliriz:

[36]:
```
a = 5
if(a == 5)
    disp("a 5'e eşit")
elseif(a > 5)
    disp("a 5'ten büyük")
else
    disp("a 5'ten küçük")
endif
```

```
a =  5
a 5'e eşit
```

a'nın değerinin 5'ten küçük olmasını kontrol ettirmeyip *"a, 5'e eşit değilse, 5'ten büyük de değilse, o zaman 5'ten küçük olmak zorundadır."* mantığıyla hareket ettiğimize dikkat edin. Tek bir değer üzerinden gittiğimizden kod biraz sıkıcı oldu, o nedenle bir for döngüsü içinde çağıralım:

[41]:
```
alar = [7   -9    3   -7    8   5    6    -2     2    5    0]
for a = alar
    a
    if(a==5)
        disp("a 5'e eşit!")
    elseif(a>5)
        disp("a 5'ten büyük.")
    else
        disp("a 5'ten küçük")
    endif
    disp("----------------------")
endfor
```

```
alar =

   7  -9   3  -7   8   5   6  -2   2   5   0

a =  7
a 5'ten büyük.
----------------------
a = -9
a 5'ten küçük
----------------------
a =  3
a 5'ten küçük
----------------------
a = -7
a 5'ten küçük
----------------------
a =  8
a 5'ten büyük.
```

```
--------------------
a =  5
a 5'e eşit!
--------------------
a =  6
a 5'ten büyük.
--------------------
a = -2
a 5'ten küçük
--------------------
a =  2
a 5'ten küçük
--------------------
a =  5
a 5'e eşit!
--------------------
a = 0
a 5'ten küçük
--------------------
```

İşleri biraz geliştirelim. Verilmiş sayıları tek/çift olarak ayıralım:

[42]:
```
alar = [7   -9    3   -7    8   5    6    -2     2    5    0]
tekler = [];
ciftler = [];
for a = alar
    if(mod(a,2) == 0)
        % a, 2 ile bölündüğünde kalanı sıfır oluyor (mod)
        % o zaman çift demektir
        ciftler = [ciftler a];
    else
        tekler = [tekler a];
    endif
endfor
disp("Tek sayılar:")
tekler

disp("Çift sayılar:")
ciftler
```

```
alar =

   7  -9   3  -7   8   5   6  -2   2   5   0

Tek sayılar:
tekler =

   7  -9   3  -7   5   5
```

```
Çift sayılar:
ciftler =

   8   6  -2   2   0
```

1.4.1 Örnek: Asal sayıların bulunması:

Asal sayılar, tanımları gereği kendileri ve 1 dışında hiçbir sayıya tam olarak bölünemeyen sayılar. 2 ile 10 arasındaki asal sayıları bulmak için biri kötü, diğeri iyi iki yoldan gideceğiz.

Kötü (verimsiz) yaklaşım Kötü olan metot, akla ilk gelen: elimizdeki sayı n olsun. 2'den başlayarak (n-1)'e kadar sayıları tek tek deneyip, n'i tam olarak bölüp bölemediklerine bakacağız, bir yandan da koddaki eksiklikleri tespit edip, adım adım geliştireceğiz:

[69]:
```
asallar = []; % bulduğumuz asalları bu listenin içine koyacağız
for sayi = 2:10
    % bu bizim asallığını kontrol etmek istediğimiz sayı
    disp("Asallığını kontrol ettiğimiz sayı:")
    sayi
    disp("") % Bir boş satır ekleyelim
    for bolen = 2:sayi-1
        % bölenlerin döngüsünün bitişini
        % sayi'ya bağlı olarak tanımladığımıza dikkat edin!
        bolen
        kalan = mod(sayi,bolen)
        if(kalan == 0)
            % Bu sayı asal olamaz.
            disp("Sayı asal değil!")
            disp("") % Bir boş satır ekleyelim
        else
            % Buraya sadece asal sayılar mı ulaşabiliyor acaba?
            % Hayır! 8)
            % Örneğin sayi 9, bolen 2,4,5,6,7,8 olduğunda
            % kalan 0 olmayacak ama biliyoruz ki sayı asal değil
        endif
        disp("") % Bir boş satır ekleyelim
    endfor
    disp("--------------------")
endfor
```

```
Asallığını kontrol ettiğimiz sayı:
sayi =  2

--------------------
Asallığını kontrol ettiğimiz sayı:
```

```
sayi =  3

bolen =  2
kalan =  1

-------------------
Asallığını kontrol ettiğimiz sayı:
sayi =  4

bolen =  2
kalan = 0
Sayı asal değil!


bolen =  3
kalan =  1

-------------------
Asallığını kontrol ettiğimiz sayı:
sayi =  5

bolen =  2
kalan =  1

bolen =  3
kalan =  2

bolen =  4
kalan =  1

-------------------
Asallığını kontrol ettiğimiz sayı:
sayi =  6

bolen =  2
kalan = 0
Sayı asal değil!


bolen =  3
kalan = 0
Sayı asal değil!


bolen =  4
kalan =  2

bolen =  5
```

```
kalan =  1

-------------------
Asallığını kontrol ettiğimiz sayı:
sayi =  7

bolen =  2
kalan =  1

bolen =  3
kalan =  1

bolen =  4
kalan =  3

bolen =  5
kalan =  2

bolen =  6
kalan =  1

-------------------
Asallığını kontrol ettiğimiz sayı:
sayi =  8

bolen =  2
kalan = 0
Sayı asal değil!


bolen =  3
kalan =  2

bolen =  4
kalan = 0
Sayı asal değil!


bolen =  5
kalan =  3

bolen =  6
kalan =  2

bolen =  7
kalan =  1

-------------------
```

```
Asallığını kontrol ettiğimiz sayı:
sayi =  9

bolen =  2
kalan =  1

bolen =  3
kalan = 0
Sayı asal değil!


bolen =  4
kalan =  1

bolen =  5
kalan =  4

bolen =  6
kalan =  3

bolen =  7
kalan =  2

bolen =  8
kalan =  1

--------------------
Asallığını kontrol ettiğimiz sayı:
sayi =  10

bolen =  2
kalan = 0
Sayı asal değil!


bolen =  3
kalan =  1

bolen =  4
kalan =  2

bolen =  5
kalan = 0
Sayı asal değil!


bolen =  6
kalan =  4
```

```
bolen =  7
kalan =  3

bolen =  8
kalan =  2

bolen =  9
kalan =  1

------------------
```

sayi = 2 ve 3 için güzel gitti ama sayi = 4 durumunu inceleyelim:

```
Asallığını kontrol ettiğimiz sayı:
sayi =  4

bolen =  2
kalan = 0
Sayı asal değil!

bolen =  3
kalan =  1
```

buradaki sıkıntı, 2'ye bölündüğünü bulduğumuz halde (yani sayının asal olmadığını anladığımız halde) hala kontrole devam ediyor oluşumuz. Bunu engellemek için döngüyü *kırmalıyız* (`break`)

[70]:
```
asallar = []; % bulduğumuz asalları bu listenin içine koyacağız
for sayi = 2:10
    % bu bizim asallığını kontrol etmek istediğimiz sayı
    disp("Asallığını kontrol ettiğimiz sayı:")
    sayi
    disp("") % Bir boş satır ekleyelim
    for bolen = 2:sayi-1
        % bölenlerin döngüsünün bitişini
        % sayi'ya bağlı olarak tanımladığımıza dikkat edin!
        bolen
        kalan = mod(sayi,bolen)
        if(kalan == 0)
            % Bu sayı asal olamaz.
            disp("Sayı asal değil! - döngüden burada çıkıyorum.")
            disp("") % Bir boş satır ekleyelim
            break; % "bolen = 2:sayi-1" döngüsünü kırıyoruz
        endif
        disp("") % Bir boş satır ekleyelim
    endfor

    % Buraya iki yoldan gelmiş olabiliriz:
    %   ya sayi kalansız bolundu ve döngü break ile kırıldı
```

```
    %     (bu durumda kalan == 0)
    %   ya da hiçbir bolen kalansız bölemedi ve 2:sayi-1 döngüsü
    %     normal yoldan sona erdi (bu durumda kalan != 0)
    if(kalan != 0)
        disp("Sayı asal!")
        asallar = [asallar sayi];
    endif
    disp("-------------------")
endfor
asallar
```

```
Asallığını kontrol ettiğimiz sayı:
sayi =  2

Sayı asal!
-------------------
Asallığını kontrol ettiğimiz sayı:
sayi =  3

bolen =  2
kalan =  1

Sayı asal!
-------------------
Asallığını kontrol ettiğimiz sayı:
sayi =  4

bolen =  2
kalan = 0
Sayı asal değil! - döngüden burada çıkıyorum.

-------------------
Asallığını kontrol ettiğimiz sayı:
sayi =  5

bolen =  2
kalan =  1

bolen =  3
kalan =  2

bolen =  4
kalan =  1

Sayı asal!
-------------------
Asallığını kontrol ettiğimiz sayı:
```

```
sayi =  6

bolen =  2
kalan = 0
Sayı asal değil! - döngüden burada çıkıyorum.

-------------------
Asallığını kontrol ettiğimiz sayı:
sayi =  7

bolen =  2
kalan =  1

bolen =  3
kalan =  1

bolen =  4
kalan =  3

bolen =  5
kalan =  2

bolen =  6
kalan =  1

Sayı asal!
-------------------
Asallığını kontrol ettiğimiz sayı:
sayi =  8

bolen =  2
kalan = 0
Sayı asal değil! - döngüden burada çıkıyorum.

-------------------
Asallığını kontrol ettiğimiz sayı:
sayi =  9

bolen =  2
kalan =  1

bolen =  3
kalan = 0
Sayı asal değil! - döngüden burada çıkıyorum.

-------------------
Asallığını kontrol ettiğimiz sayı:
sayi =  10
```

```
bolen =  2
kalan = 0
Sayı asal değil! - döngüden burada çıkıyorum.

-------------------
asallar =

   2   3   5   7
```

break komutu o an içerisinde bulunulan döngüyü sonlandırmakta kullanılır.

Örneğin, satır ve sütunu eşit oluncaya kadar (o eleman dahil) değerleri 1 olan bir matris oluşturalım:

[59]:
```
m = zeros(5,5) % bütün elemanları 0 olan, 5x5 bir matris oluşturur.
for satir=1:5
    for sutun = 1:5
        m(satir,sutun) = 1;
        if(satir == sutun)
            break;
            % şu anda sutun=1:5 döngüsünün içinde olduğumuzdan
            % break ile sadece bu bloktan çıkarız.
        endif
    endfor
endfor
m
```

```
m =

   0   0   0   0   0
   0   0   0   0   0
   0   0   0   0   0
   0   0   0   0   0
   0   0   0   0   0

m =

   1   0   0   0   0
   1   1   0   0   0
   1   1   1   0   0
   1   1   1   1   0
   1   1   1   1   1
```

İyi (verimli) yaklaşım Yukarıdaki yaklaşımda, 7'nin asal olup olmadığını kontrol ederken şu aşamalardan geçtik:

```
Asallığını kontrol ettiğimiz sayı:
```

```
sayi =  7

bolen =  2
kalan =  1

bolen =  3
kalan =  1

bolen =  4
kalan =  3

bolen =  5
kalan =  2

bolen =  6
kalan =  1

Sayı asal!
```

Buradaki sıkıntı, 2'ye bölünmeyen bir sayının 4'e ve 6'ya bölünüp bölünmediğini kontrol etmemiz. Halbuki 2'ye bölünmüyorsa, 2'nin katlarına da bölünemez. Benzer şekilde, 3'e bölünmüyorsa, 6'ya ve 9'a da bölünemez, vs.. O halde boşu boşuna kontrol ediyoruz.

Asal sayıların tanımını biraz geliştirelim: *kendilerinden ve 1'den başka hiçbir* ***asal*** *sayıya kalansız olarak bölünmeyen sayılara asal sayı denir.* diyelim. Sonuçta asal olmayan bir sayıyı her zaman için asal çarpanlarının çarpımı şeklinde yazabiliriz (12 = 2 x 6 = 2 x 2 x 3 = 2^2 x 3 gibi).

Tekrar 7'nin asallık kontrolüne geldiğimizi düşünelim. O anda elimizdeki `asallar` listesinde kimler var? 2,3,5 –> o halde 7'nin sadece bu sayılara tam bölünüp bölünmediğini kontrol etmek yeterli. Kodu bu şekilde değiştirmek demek, bölenlerin listesini "2:sayi-1" yerine `asallar` listesine eşitlemek demek:

[72]:
```
asallar = [2]; % bulduğumuz asalları bu listenin içine koyacağız
for sayi = 3:10
    % bu bizim asallığını kontrol etmek istediğimiz sayı
    disp("Asallığını kontrol ettiğimiz sayı:")
    sayi
    disp("") % Bir boş satır ekleyelim
    for bolen = asallar
        % asallar listesinin dinamik bir liste olduğunu
        % yani mesela 4'e gelindiğinde [2 3];
        % 9'a gelindiğinde [2 3 5 7] olduğuna dikkat edin!
        bolen
        kalan = mod(sayi,bolen)
        if(kalan == 0)
            % Bu sayı asal olamaz.
            disp("Sayı asal değil! - döngüden burada çıkıyorum.")
            disp("") % Bir boş satır ekleyelim
            break; % "bolen = 2:sayi-1" döngüsünü kırıyoruz
```

```
        endif
        disp("") % Bir boş satır ekleyelim
    endfor

    % Buraya iki yoldan gelmiş olabiliriz:
    %   ya sayi kalansız bolundu ve döngü break ile kırıldı
    %     (bu durumda kalan == 0)
    %   ya da hiçbir bolen kalansız bölemedi ve 2:sayi-1 döngüsü
    %     normal yoldan sona erdi (bu durumda kalan != 0)
    if(kalan != 0)
        disp("Sayı asal!")
        asallar = [asallar sayi];
    endif
    disp("--------------------")
endfor
asallar
```

```
Asallığını kontrol ettiğimiz sayı:
sayi =  3

bolen =  2
kalan =  1

Sayı asal!
--------------------
Asallığını kontrol ettiğimiz sayı:
sayi =  4

bolen =  2
kalan = 0
Sayı asal değil! - döngüden burada çıkıyorum.

--------------------
Asallığını kontrol ettiğimiz sayı:
sayi =  5

bolen =  2
kalan =  1

bolen =  3
kalan =  2

Sayı asal!
--------------------
Asallığını kontrol ettiğimiz sayı:
sayi =  6
```

```
bolen =  2
kalan = 0
Sayı asal değil! - döngüden burada çıkıyorum.

-------------------
Asallığını kontrol ettiğimiz sayı:
sayi =  7

bolen =  2
kalan =  1

bolen =  3
kalan =  1

bolen =  5
kalan =  2

Sayı asal!
-------------------
Asallığını kontrol ettiğimiz sayı:
sayi =  8

bolen =  2
kalan = 0
Sayı asal değil! - döngüden burada çıkıyorum.

-------------------
Asallığını kontrol ettiğimiz sayı:
sayi =  9

bolen =  2
kalan =  1

bolen =  3
kalan = 0
Sayı asal değil! - döngüden burada çıkıyorum.

-------------------
Asallığını kontrol ettiğimiz sayı:
sayi =  10

bolen =  2
kalan = 0
Sayı asal değil! - döngüden burada çıkıyorum.

-------------------
asallar =
```

```
2   3   5   7
```

Dikkat ederseniz, bu sefer asallar listesini en başta 2 olacak şekilde başlattığımızı, sayi döngüsünü de 3'ten başlattığımızı göreceksiniz. Bunun sebebi "for bolen = asallar" döngüsünün asallar listesi boş olduğu zaman çalışmaması ve bu yüzden asal avına daha baştan çıkamamamız. En azından 2'yi koyuyoruz ki, 3'ü kontrol edecek bir bölenimiz olsun.