# FIZ219_EST_UygulamaNotlari_08_Fonksiyonlar_Giris_ve_Basit_Uygulam

December 23, 2019

## 1 Uygulama Notları: 8

### 1.1 FİZ219 - Bilgisayar Programlama I | 23/12/2019

**Fonksiyonlar I: Giriş ve basit uygulamalar** * Giriş * Ortalama fonksiyonu * Fonksiyonumuzu geliştirelim * Çok işlevli fonksiyonlar * Fonksiyonlar hakkında yardım * Fonksiyon çağıran fonksiyonlar * Fibonacci fonksiyonu (tekrarlı) * Fonksiyon yazarken dikkat edilecek birkaç şey

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

## 2 Fonksiyonlar

### 2.1 Giriş

Fonksiyonlar, kendilerine verilen değerleri (hiç değer verilmiyor da olabilir) alıp, onları kullanarak birtakım işler yapan özel betiklerdir (betik: *script*). O kadar özeldirler ki, GNU Octave'da, fonksiyon yazarken, normal bir betik yazarkenkinden daha fazla şeye dikkat etmemiz gerekir.

Bir fonksiyonu yazmak için öncelikle üç bileşenine karar vermeliyiz: * Fonksiyonun adı * Fonksiyonun girdi değerleri * Fonksiyonun çıktı değeri (/değerleri)

Fonksiyonlar, kendisi ile aynı addaki bir dosyaya yazılır: örneğin, fonksiyonumuzun adı `ortalama` ise, saklayacağımız dosyanın adı da `ortalama.m` olacaktır.

### 2.2 Ortalama fonksiyonu

Verilen iki sayının aritmetik ortalamasını hesaplayan bir fonksiyon yazalım. Bunun için öncelikle `ortalama.m` adında bir dosya açıyoruz (Windows kullanıcısı iseniz, ve Octave-GUI kullanmıyorsanız dikkat! Dosyanızı kaydederken türünü "txt dosyası" olarak kaydederseniz, siz adını `ortalama.m` olarak yazsanız bile Windows onu aslında `ortalama.m.txt` şeklinde "txt" uzantılı kaydedecek, Octave da tanımayacaktır. Bu nedenle, tür olarak "tüm dosya türleri"ni seçip, öyle kaydedin).

Dosyamızın içeriği şu şekilde olabilir:

function f=ortalama(a,b) f = (a + b) / 2; endfunction

daha ilk satırda, fonksiyonumuza dair pek çok şey belirtmiş olduk: fonksiyonumuzun adını ("ortalama") belirledik, iki tane girdi parametresi (a,b) olduğunu ve cevabın (fonksiyondan döndürülecek değerin) f değişkeninde tutulduğunu.

Fonksiyonun bitişini işaret eden `endfunction` satırına gelindiğinde, fonksiyon, ilk satırda fonksiyonun adının solundaki (bizim örneğimizde "f" değişkeni) değişkenin dönüş değeri olduğunu bildiğinden, o değişkendeki değer neyse onu çağıran tarafa geri döndürür.

Fonksiyon dosyamızı bu şekilde tanımlayıp, Octave'ın gördüğü bir klasöre "ortalama.m" adıyla kayıt ettikten sonra fonksiyonu çağıralım:

[1]:
```
ortalama(3,9)
```

```
ans =  3
```

Dikkat edecek olursanız, fonksiyonu, dosya adına istinaden "ortalama.m" olarak değil, doğrudan adıyla çağırdık. Dahası, fonksiyonumuzu parametreleri ile çağırdık (girdisi olmayan bir fonksiyonu bile çağırsak, mutlaka fonksiyon olduğunu belirtmek için parantez kullanırız). Aslında işin başından beri türlü türlü fonksiyonu çağırmaktayız (`plot(), sin(), sum(), ...`), bizimkinin hazırda gelenlerden yapı olarak hiçbir farkı yok. Nasıl ki `sin` demek tek başına bir şey ifade etmiyorsa, `ortalama` yazıp enter'a basmak da hatadan başka bir şeye sebep olmaz.

### 2.2.1 Fonksiyonumuzu geliştirelim

Peki ya 2 değil de, 3 sayının ortalamasını almak istersek? Ya da 5?.. Fonksiyonumuz mevcut haliyle sadece 2 sayının ortalamasını alıyor maalesef. Bunu nasıl geliştirebiliriz peki?

Bir çözüm, sayıları tek tek girmek yerine, bir listeyle kabul etmek olabilir. Örneğin 5 sayılık bir listeyi `x = [4 2 9 -3 4]` olarak tanımlayıp, fonksiyonumuza o şekilde verebiliriz. Fonksiyonumuz şu anda bunu yapamıyor, o halde tekrardan içine girip düzenleyip, geliştirelim:

function f=ortalama(x) eleman_sayisi = length(x); toplam = sum(x); f = toplam / eleman_sayisi; endfunction

[2]:
```
x = [4 2 9 -3 4];
ortalama(x)
```

```
ans =  3.2000
```

### 2.2.2 Çok işlevli fonksiyonlar

Bu haliyle fonksiyonumuz aritmetik ortalama ($\frac{\sum_{i=1}^{N} x_i}{N}$) alıyor, peki bunun yanında geometrik ortalama ( $\sqrt[N]{\prod_{i=1}^{N} x_i}$) almasını da isteyebiliriz.

Bahsettiğimiz şey şu: verilen $N$ boyutlu bir $\vec{x}$ girdisine karşılık, iki farklı işlemin sonucunu istiyoruz. Birçok dilin aksine, Octave'da birden fazla dönüş değeri üreten fonksiyon tanımlamak mümkün, tek yapmamız gereken, fonksiyonumuzu tanımlarken, dönüş değerlerini dizi olarak tanıtmak:

function [a,g] = ortalama(x) eleman_sayisi = length(x); toplam = sum(x); carpim = prod(x); a = toplam / eleman_sayisi; g = carpim^(1/eleman_sayisi); % benzer şekilde nthroot(carpim,N) % de kullanilabilir.. endfunction

Fonksiyon çalışırken, `endfunction` yönergesine geldiğinde, ilk satırda tanımla dönüş değişkenlerinin değerlerini döndürdüğünü unutmayın.

[3]:
```
dizi = [1 2 3];
ortalama(dizi)
```

```
ans =  2
```

Yukarıda gördüğünüz üzere, fonksiyon hem aritmetik, hem de geometrik ortalamayı hesaplamasına rağmen, bize sadece aritmetik ortalamayı verdi!

Bunun sebebi, bizim ondan ikinci dönüş değişkenini istemediğimizi sanmış olmasıdır. Eğer ilk dönüş değerinden başka, diğer değişkenleri de istiyorsak, fonksiyonumuzu çağırırken, bunu özellikle belirtmeliyiz:

[4]:
```
[a_ort g_ort] = ortalama(dizi)
```

```
a_ort =  2
g_ort =  1.8171
```

### 2.2.3 Fonksiyonlar hakkında yardım

Bildiğiniz üzere, bir fonksiyon hakkındaki yardımı, o fonksiyonun adının başına `help` yazarak alabiliyoruz. Örneğin:

Açıklamanın ilk satırında komutun ait olduğu fonksiyon dosyasının yerini söylüyor (benim durumumda: /snap/octave/current/usr/share/octave/5.1.0/m/elfun/sind.m klasörü imiş). Sonrasında da fonksiyonun ne yaptığını ve ilgili fonksiyonları.

Kendi elimizle yazdığımız fonksiyonlara da bu türden yardımcı bilgiler koyabiliriz - fonksiyon tanımından sonra yorum olarak yazacağımız her satır, `help` komutu ile bilgi istendiğinde yazdırılır:

function [a,g] = ortalama(x) % [a,g] = ortalama(x) fonksiyonu % % verilen x dizisinin elemanlarının % aritmetik ve geometrik ortalamalarını % hesaplar. % % ilk dönüş değeri olarak aritmetik ortalamayı, % ikinci dönüş değeri olarak da geometrik ortalamayı % döndürür. % % ayrıca bkz. mean(x,“a”), mean(x,“g”) eleman_sayisi = length(x); toplam = sum(x); carpim = prod(x); a = toplam / eleman_sayisi; g = carpim^(1/eleman_sayisi); % benzer şekilde nthroot(carpim,N) % de kullanilabilir.. endfunction

[5]:
```
help ortalama
```

```
'ortalama' is a function from the file
/home/sururi/ownCloud/Jupyter_notebooks/FIZ219/ortalama.m

 [a,g] = ortalama(x) fonksiyonu

 verilen x dizisinin elemanlarının
```

```
 aritmetik ve geometrik ortalamalarını
 hesaplar.

 ilk dönüş değeri olarak aritmetik ortalamayı,
 ikinci dönüş değeri olarak da geometrik ortalamayı
 döndürür.

 ayrıca bkz. mean(x,"a"), mean(x,"g")


Additional help for built-in functions and operators is
available in the online version of the manual.  Use the command
'doc <topic>' to search the manual index.

Help and information about Octave is also available on the WWW
at https://www.octave.org and via the help@octave.org
mailing list.
```

## 2.3 Fonksiyon çağıran fonksiyonlar

Bir fonksiyon, kendi tanımı içinde bir başka fonksiyonu çağırabilir. Örneğin, tanjant hesaplayan bir fonksiyonu şu şekilde yazabiliriz:

function t=tanjant(x) t = sin(x)/cos(x); endfunction

Bir fonksiyon iyi tanımlandığı takdirde, kendisini bile çağırabilir!!! Bu tür fonksiyonlara "tekrarlı" (*rekürsif / recursive*) fonksiyonlar denir. Bir örnek olarak Fibonacci Sayılarını bu yoldan hesaplayan bir fonksiyon yazalım.

### 2.3.1 Fibonacci fonksiyonu (tekrarlı)

İlk iki Fibonacci sayısı: $F_0 = 0$, $F_1 = 1$ şeklinde tanımlanıp, sonraki herhangi bir Fibonacci sayısı ise $F_{i+1} = F_i + F_{i-1}$ formülünden hesaplanır. Biz bu seriyi hesaplayan bir betiği 1. ara sınavın bonus sorusu olarak şu şekilde yazmıştık:

[6]:
```
fibonacci = [0 1];
for i=3:100
    fibonacci(i) = fibonacci(i-2) + fibonacci(i-1);
endfor
fibonacci(1:10) % ilk 10 terim
fibonacci(98:100) % son 3 terim
```

```
ans =

    0    1    1    2    3    5    8   13   21   34

ans =
```

```
   8.3621e+19    1.3530e+20    2.1892e+20
```

Şimdi de, biraz (epey! 8) daha yavaş çalışan ama tekrarlı özelliğinden ötürü hayli şık, biraz da üzerinde düşününce ufuk açan bir şekilde, fonksiyon olarak yazalım:

function f=fibo(n) if(n<2) f = n; else f = fibo(n-1) + fibo(n-2); endif endfunction

[7]:
```
for n=0:9
    printf("F_%d: %3d\n",n,fibo(n))
endfor
```

```
F_0:   0
F_1:   1
F_2:   1
F_3:   2
F_4:   3
F_5:   5
F_6:   8
F_7:  13
F_8:  21
F_9:  34
```

## 2.4 Fonksiyon yazarken dikkat edilecek birkaç şey

### 2.4.1 Varsayılan değerler

Eğer fonksiyonunuzda kullandığınız bir parametre belirtilmemişse varsayılan bir değer atamak istiyorsanız, fonksiyonunuzda tanımlarken "=" işareti ile bu değeri belirtin. Kullanıcı yazmadığı takdirde bu değer alınacaktır.

Örneğin:

[8]:
```
function g=arttir(a,b=4)
% ikinci parametre girilmediği takdirde, birinciyi 4 arttırır,
% girildiği takdirde ikinci kadar arttırır
g = a + b;
endfunction

arttir(3,9)
arttir(3)
```

```
ans =  12
ans =  7
```

### 2.4.2 Tekeri tekrardan icat etmeyin

(Sizler şu anda öğrenme aşamasında olduğunuz için aslında şimdilik bu kuraldan muafsınız 8)

Eğer yapmak istediğiniz bir işi yapan bir fonksiyon yazmak üzereyseniz, öncesinde bir durup, böyle bir fonksiyonun halihazırda mevcut olup olmadığını araştırın. Hazırtanımlı olarak gelen, veya geliştirici gruplar tarafından çıkartılmış ek kütüphanelerdeki fonksiyonlar sizin yazacağınız fonksiyonlardan genellikle daha hızlı çalışıp, daha hassas sonuçlar verir.

### 2.4.3 Fonksiyonlar kendi dünyalarında yaşarlar

Fonksiyon tanım bloğunda tanımladığınız her değişken, fonksiyon çalıştığı sürece vardırlar ve fonksiyon `endfunction` yönergesine gelip de değerini döndürdüğü anda yok olurlar. Fonksiyonların kendi evrenleri vardır ve bu evrendeki bir değişkenin ismi sizin programınızdaki bir başka değişkenle aynı olsa bile ikisi de birbirinden etkilenmeyecektir (`global` yönergesiyle istisnalar yapılabilse de, kesinlikle tavsiye edilmez). Örneğin:

[9]:
```
% a ve b'ye kodumuzda değer atayalım
a = 5
b = 7

function a=topla(x,y)
% fonksiyon içinde de a ve b geçiyor,
% hatta a, dönüş değerinin ta kendisi olarak tanımlı!
a = x + y;
% b = 3;
endfunction

topla(3,6)
% Fonksiyonu çalıştırdık
% bakalım a'nın değeri ne oldu?
a
% b ne oldu peki?
b
```

```
a =  5
b =  7
ans =  9
a =  5
b =  7
```

### 2.4.4 Fonksiyonlarınıza anlamlı adlar verin

Bu adlar, fonksiyonun yaptığı işi tanımlar şeyler olsun. Tek bir karakterden ibaret isimler vermeyin; “ı,ğ,İ,...” gibi Türkçe harfler içeren fonksiyon isimleri hem dosya isminden ötürü, hem de çalıştırılırken standard olmayan karakter setinizden dolayı sorun çıkarma potansiyeli yüksek seçimlerdir.

### 2.4.5 Fonksiyonlarınızı her zaman kendilerine ait dosyalarda tanımlayın

İki yukarıdaki `topla` örneğinde dikkat ettiyseniz fonksiyonumuzu doğrudan betiğimizin içinde tanımlayıp, sorunsuzca kullandık. Yanlış: sorun çıkacak çünkü fonksiyonda bir değişiklik yapamıyoruz, fonksiyonların en önemli özelliklerinden olan taşınabilirlik (yani başka programlarca da kullanılma, ki bu taşınabilirlik özelliğinin nimetlerinden gelecek dönemde Python'da nesne tabanlı (*object oriented*) kodlar yazarken çokça faydalanacağız) özelliğini kullanamıyoruz. Özetle, her zaman için fonksiyonunuzu ona ait bir dosyada tanımlayın.

### 2.4.6 Fonksiyonunuz farklı türlere kucak açsın

Verilen sayının karesini alan bir fonksiyon yazdığımızı varsayalım. İlk aklımıza gelen, doğal olarak:

function k = kare_al(x) k = x^2; endfunction

ama bu, diyelim ki 3'ün karesini alabilmekle beraber, [3 4 5] şeklinde verilen bir girdinin karesini alamaz. Hazır elimiz değmişken fonksiyonumuzu farklı türlere de -tabii ki yaptığı işlem onlarda da mana taşıyorsa- işlem yapacak şekilde yazalım, yani:

function k = kare_al(x) k = x.^2; endfunction

Unutmayın: Octave öncelikli ve tercihen sayısal bir programlama dilidir, sembolik değil. Bunun nimetlerinden sonuna kadar faydalanın, yazacağınız fonksiyonlarda da faydalandırın. Fonksiyonlarınız vektörleri de (hatta oluyorsa matrisleri de) skalerleri desteklediği kadar desteklesin, çağrıldıkları zaman tek satırda yüz binlerce elemanlı bir vektörün elemanlarını tek tek işleyebilsinler.

### 2.4.7 Aklınızda kalmasın: Girdisiz, çıktısız, girdisiz-çıktısız fonksiyonlar

Girdisi veya çıktısı ya da her ikisi de olmayan fonksiyonlar da yazmak mümkün, örneğin çağrıldığında, o andaki saati gösteren bir fonksiyon:

[10]:
```
function f=tarih_saat()
su_an = localtime(time);
printf("Saat: %02d:%02d\n",su_an.hour(),su_an.min());
endfunction

tarih_saat()
```

```
Saat: 21:08
```

## 2.5 (Bilerek) bahsetmediğimiz şeyler

Derste bahsetmedik ama ilginizi çekerse, bilginizi geliştirmek için fonksiyonlarla ilgili bakabileceğiniz birkaç başlık: * `global` yönergesi * tek satırlık (*inline*) fonksiyonlar