

# فصل ۶

## برنامه نویسی در MATLAB

در این فصل خواهیم دید که چگونه یک برنامه MATLAB قسمتی از کد را اجرا کند

- اگر (if) چیزی درست باشد
- زمانی که (while) چیزی درست باشد
- برای (for) دفعات مشخص

همچنین خواهیم دید که چطور قسمتهای مختلف کد اجرا شود بسته به این که

- مقدار یک متغیر چقدر است
- کدام شرط مشخص برقرار است
- کدام ترکیب از شرطها برقرار است
- اگر این و (and) آن درست باشد
- اگر این یا (or) آن درست باشد
- چه رابطه ای میان دو چیز برقرار است
- مثلاً چیزی کمتر از، بیشتر از، مساوی با یا نامساوی با دیگری است

## عملگرهای رابطه ای:

<u>Relational operator</u>	<u>Description</u>
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
~=	Not Equal to

- بین عملگرهایی که دو کاراکتر دارند نمیتوان فاصله قرار داد
- مساوی برای مقایسه دو عدد با دو علامت مساوی نوشته میشود (=)، نه یکی.
- میدانیم که "=" یعنی قرار بده یا اختصاص بده



- نتیجه مقایسه با یک عملگر رابطه ای همیشه درست (true) یا غلط (false) است
- اگر درست باشد، MATLAB به مقایسه مقدار ۱ را نسبت می دهد
- اگر غلط باشد، MATLAB به مقایسه مقدار ۰ را نسبت می دهد

## در مقایسه آرایه ها

- باید دارای ابعاد برابر باشند
- مقایسه به صورت درایه به درایه انجام خواهد شد
- نتیجه آرایه ای با ابعاد دو آرایه مورد مقایسه خواهد بود که تنها حاوی ۰ و ۱ است

در مقایسه آرایه ها با اعداد

- عدد با همه درایه های آرایه مقایسه خواهد شد

- نتیجه آرایه ای با ابعاد آرایه مورد مقایسه خواهد بود که تنها حاوی ۰ و ۱ است

# مثال

```
>> x=8:12
```

x =	8	9	10	11	12
-----	---	---	----	----	----

```
>> x>10
```

ans =	0	0	0	1	1
-------	---	---	---	---	---

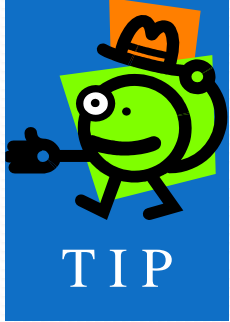
```
>> x==11
```

ans =	0	0	0	1	0
-------	---	---	---	---	---

```
>> x>=7
```

ans =	1	1	1	1	1
-------	---	---	---	---	---





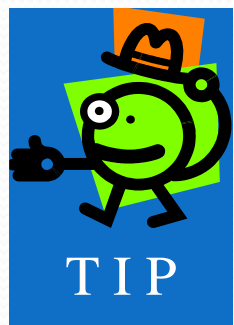
در واقع نتیجه یک مقایسه منطقی

1. یک بردار است

2. نظیر هر درایه بردار اصلی ۰ یا ۱ در آن وجود دارد

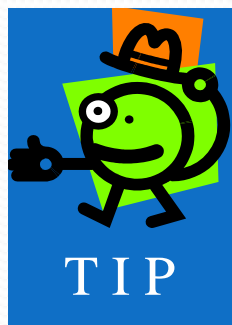
```
>> x=8:12
```

x =	8	9	10	11	12
>> x>10	↓	↓	↓	↓	↓
ans =	0	0	0	1	1



اگر نتیجه یک مقایسه رابطه ای در یک بردار ذخیره شود، تعداد درایه هایی که در مقایسه نتیجه درست دارند را به راحتی میتوان با دستور sum که جمع درایه های یک بردار را میدهد به دست آورد

- این به آن علت است که درایه هایی که مقدار درست دارند برابر 1 و غلطها برابر 0 هستند



## مثال

چه تعداد از اعداد ۱ تا ۲۰ اول هستند؟

- از دستور `isprime` استفاده کنید که برای اعداد اول ۱ و در غیر این صورت ۰ میدهد

```
>> numbers = 1:20;
```

```
>> sum( isprime(numbers) )
```

```
ans =
```

```
8
```



میتوان عملگرهای رابطه ای و حسابی را در یک عبارت ترکیب کرد

- عملگرهای حسابی از اولویتهای معمول پیروی میکنند و نسبت به عملگرهای رابطه ای دارای اولویت هستند
- عملگرهای رابطه ای دارای اولویت یکسان هستند و از چپ به راست ارزیابی میشوند

```
>> 3+4<16/2
ans =
    1
```

+ and / are executed first.  
The answer is 1 since 7 < 8 is true.

```
>> 3+(4<16)/2
ans =
    3.5000
```

4 < 16 is executed first, and is equal to 1, since it is true.  
3.5 is obtained from 3 + 1/2.

یک آرایه منطقی آرایه ای است که تنها حاوی ۰ و ۱ منطقی باشد

- ۰ و ۱ های ناشی از عملیات ریاضی منطقی محسوب نمی شوند

- تنها ۰ و ۱ های ناشی از مقایسه، منطقی هستند

- به محض استفاده از یک آرایه منطقی در محاسبات، MATLAB آن را به یک آرایه عددی تبدیل می کند

میتوان از بردارهای منطقی برای بررسی مقادیری که به ازای آنها رابطه ای درست است، و نه فقط درستی کلی رابطه استفاده کرد. به این عمل اندیس گذاری منطقی میگویند.

- این کار را با استفاده از بردار منطقی به عنوان نشانی درایه های بردار مقادیر انجام دهید. نتیجه مقادیری است که در رابطه صدق میکنند، یعنی مقادیری که اندیس آنها ۱ است.

## مثال

کدام اعداد از ۱ تا ۱۰ مضرب ۳ هستند؟

```
>> numbers = 1:10
```

```
numbers      = 1   2   3   4   5   6   7   8   9
               10
```

```
>> multiples = rem( numbers, 3 ) == 0
```

```
multiples = 0   0   1   0   0   1   0   0   1
              0
```

```
>> multiplesOf3 = numbers(multiples)
```

```
multiplesOf3 =
              3   6   9
```

# مثال

numbers (multiples) را به عنوان بیرون  
کشیدن درایه هایی از numbers که درایه نظیر آنها در  
multiples است در نظر بگیرید

numbers	=	1	2	3	4	5	6	7	8	9	10
				↕			↕			↕	
multiples	=	0	0	1	0	0	1	0	0	1	0
				↘			↓			↙	
numbers(multiples)	=			3			6			9	



## مثال

کدام یک از اعداد ۱ تا ۲۰ اول هستند؟

```
>> numbers = 1:20;
```

```
>> numbers( isprime(numbers) )
```

```
ans =
```

```
2    3    5    7   11   13   17   19
```

اندیس گذاری منطقی به خصوص وقتی مفید خواهد بود که با عملگرهای منطقی استفاده شود. در ادامه در مورد آن صحبت میشود

عملگرهای منطقی:

منطق بولی سیستمی است برای ترکیب عباراتی که درست یا غلط هستند

- MATLAB عملگرها و دستورهایی برای انجام بسیاری از عملیات بولی دارد

- عملیات بولی در ترکیب با دستورات رابطه ای امکان انجام انواع خاصی از محاسبات را به آسانی و روشنی میدهند

جدول صحت سنجی قوانین منطق بولی را تعریف میکند. این جدول نتیجه عملیات منطقی را برای هر ترکیب ممکن از ورودی ها میدهد. جدول صحت سنجی مرتبط با MATLAB به این صورت است

INPUT		OUTPUT				
A	B	AND A&B	OR A B	XOR (A,B)	NOT ~A	NOT ~B
false	false	false	false	false	true	true
false	true	false	true	true	true	false
true	false	false	true	true	false	true
true	true	true	true	false	false	false

## جدول صحت منجی به صورت تشریحی یعنی

- AND درست است اگر هر دو ورودی درست باشند، وگرنه غلط است
- OR درست است اگر حداقل یک ورودی درست باشد، وگرنه غلط است
- XOR (exclusive OR) درست است اگر تنها یک ورودی درست باشد، وگرنه غلط است
- NOT درست است اگر ورودی غلط باشد، وگرنه غلط است

یک عملگر حسابی مثل  $+$  یا  $-$  نمادی است که باعث میشود MATLAB یک محاسبه با اعداد یا عبارات دو طرف آن انجام دهد

به طور مشابه، یک عملگر منطقی کاراکتری است که باعث میشود MATLAB یک عملیات منطقی روی یک یا دو عدد یا عبارت انجام دهد

MATLAB سه عملگر منطقی دارد:  $\&$ ,  $|$ ,  $\sim$

- $a \& b$  عمل و (AND) منطقی را روی  $a$  و  $b$  انجام میدهد
- $a | b$  عمل یا (OR) منطقی را روی  $a$  و  $b$  انجام میدهد
- $\sim a$  عمل نقیض (NOT) را روی  $a$  انجام میدهد
- ورودی های همه عملگرهای منطقی عدد هستند
- صفر غلط (false) است
- هر عدد به جز صفر درست (true) است
- نتیجه یا خروجی یک عملگر منطقی ۱ یا ۰ منطقی است

زمانی که عملگرهای منطقی برای آرایه ها استفاده میشوند

- باید دارای ابعاد برابر باشند
- MATLAB ارزیابی عملگر را درایه به درایه انجام میدهد
- نتیجه آرایه ای با همان ابعاد ولی تنها حاوی ۰ و ۱ خواهد بود
- not (نقیض) تنها روی یک آرایه کار میکند

- زمانی که از عدد و آرایه استفاده شود
- MATLAB عملیات را با هر درایه آرایه و عدد انجام میدهد
  - نتیجه آرایه ای با ابعاد آرایه اولیه ولی تنها حاوی ۰ و ۱ خواهد بود



میتوان عملگرهای حسابی، رابطه ای و منطقی را ترکیب کرد. اولویتها به این صورت هستند

<u>Precedence</u>	<u>Operation</u>
1 (highest)	Parentheses (if nested parentheses exist, inner ones have precedence)
2	Exponentiation
3	Logical NOT (~)
4	Multiplication, division
5	Addition, subtraction
6	Relational operators (>, <, >=, <=, ==, ~=)
7	Logical AND (&)
8 (lowest)	Logical OR ( )

# مثال

کودی - ۱۲ سال یا کمتر

نوجوان - بیش از ۱۲ و کمتر از ۲۰ سال

بزرگسال - بیش از ۲۰ سال

```

>> age=[45 47 15 13 11]

```

```

age = 45      47      15      13      11

```

# مثال

نوجوانها کدامها هستند؟

```
>> age=[ 45  47  15  13  11];
```

```
>> age>=13
```

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

```
>> age<=19
```

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

```
>> age>=13 & age<=19
```

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

دو نوجوان اینجا مشخص میشوند

# مثال

```
>> age=[45 47 15 13 11]
```

```
age = 45      47      15      13      11
```

چه کسی نوجوان نیست؟

```
>> ~(age>=13 & age<=19)
```

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

چه کسی بزرگسال یا کودکی است؟

```
>> age>19 | age<13
```

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

توابع منطقی داخلی:

MATLAB دارای چند تابع داخلی یا دستور برای انجام عملیات منطقی و محاسبات مربوطه است. سه تای آنها معادل عملگرهای منطقی هستند

- $\text{and}(A, B)$  – معادل  $A \& B$
- $\text{or}(A, B)$  – معادل  $A | B$
- $\text{not}(A)$  – معادل  $\sim A$

# MATLAB توابع بولی دیگری نیز دارد

Function	Description	Example
<code>xor(a,b)</code>	Exclusive or. Returns true (1) if one operand is true and the other is false.	<pre>&gt;&gt; xor(7,0) ans =      1 &gt;&gt; xor(7,-5) ans =      0</pre>
<code>all(A)</code>	Returns 1 (true) if all elements in a vector A are true (non-zero). Returns 0 (false) if one or more elements are false (zero). If A is a matrix, treats columns of A as vectors, and returns a vector with 1s and 0s.	<pre>&gt;&gt; A=[6 2 15 9 7 11]; &gt;&gt; all(A) ans =      1 &gt;&gt; B=[6 2 15 9 0 11]; &gt;&gt; all(B) ans =      0</pre>
<code>any(A)</code>	Returns 1 (true) if any element in a vector A is true (nonzero). Returns 0 (false) if all elements are false (zero). If A is a matrix, treats columns of A as vectors, and returns a vector with 1s and 0s.	<pre>&gt;&gt; A=[6 0 15 0 0 11]; &gt;&gt; any(A) ans =      1 &gt;&gt; B = [0 0 0 0 0 0]; &gt;&gt; any(B) ans =      0</pre>
<code>find(A)</code> <code>find(A&gt;d)</code>	If A is a vector, returns the indices of the nonzero elements. If A is a vector, returns the address of the elements that are larger than d (any relational operator can be used).	<pre>&gt;&gt; A=[0 9 4 3 7 0 0 1 8]; &gt;&gt; find(A) ans =      2     3     4      5     8     9 &gt;&gt; find(A&gt;4) ans =      2     5     9</pre>


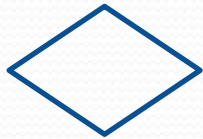

یک جمله شرطی دستوری است که به  
MATLAB امکان تصمیم گیری برای اجرا یا  
عدم اجرای دستوراتی که پس از آن می آید را  
می دهد

- جملات شرطی تقریباً در تمام کدها یا توابع وجود دارند

- این جملات دارای سه صورت کلی هستند:

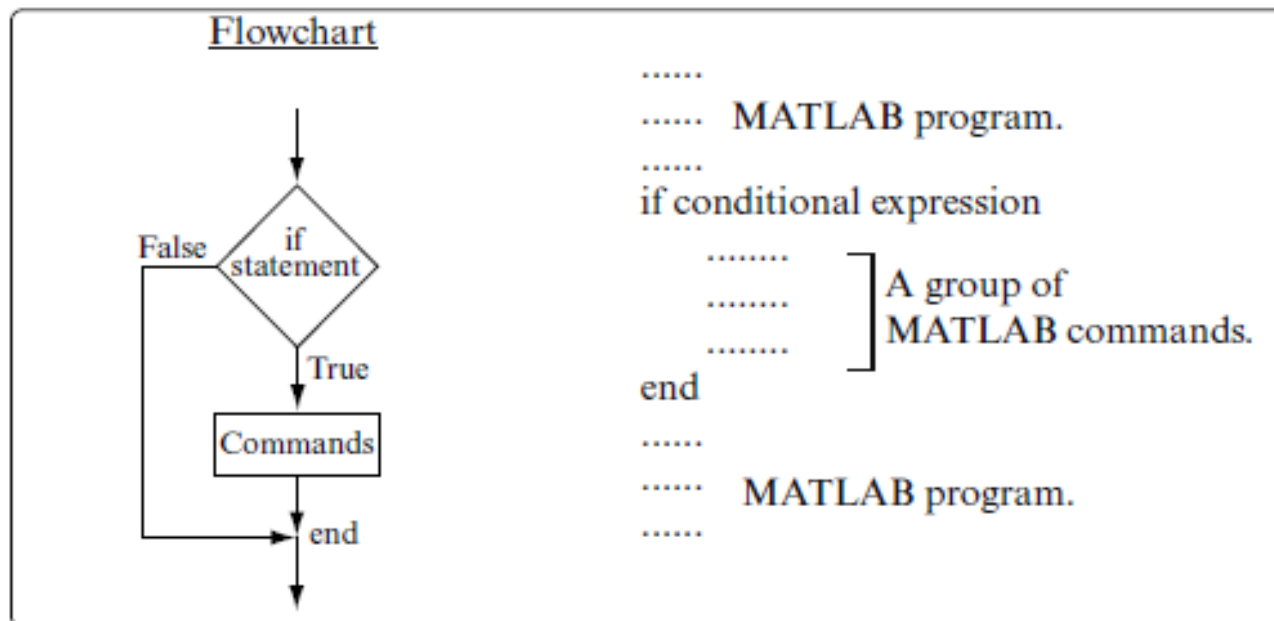
- if-end
- if-else-end
- if-elseif-else-end

فلوپارت نموداری است که روند اجرای کد را نشان میدهد. به خصوص برای نشان دادن نحوه عملکرد دستورات شرطی مناسب است. برخی از علائم رایج در فلوپارتهای عبارتند از:

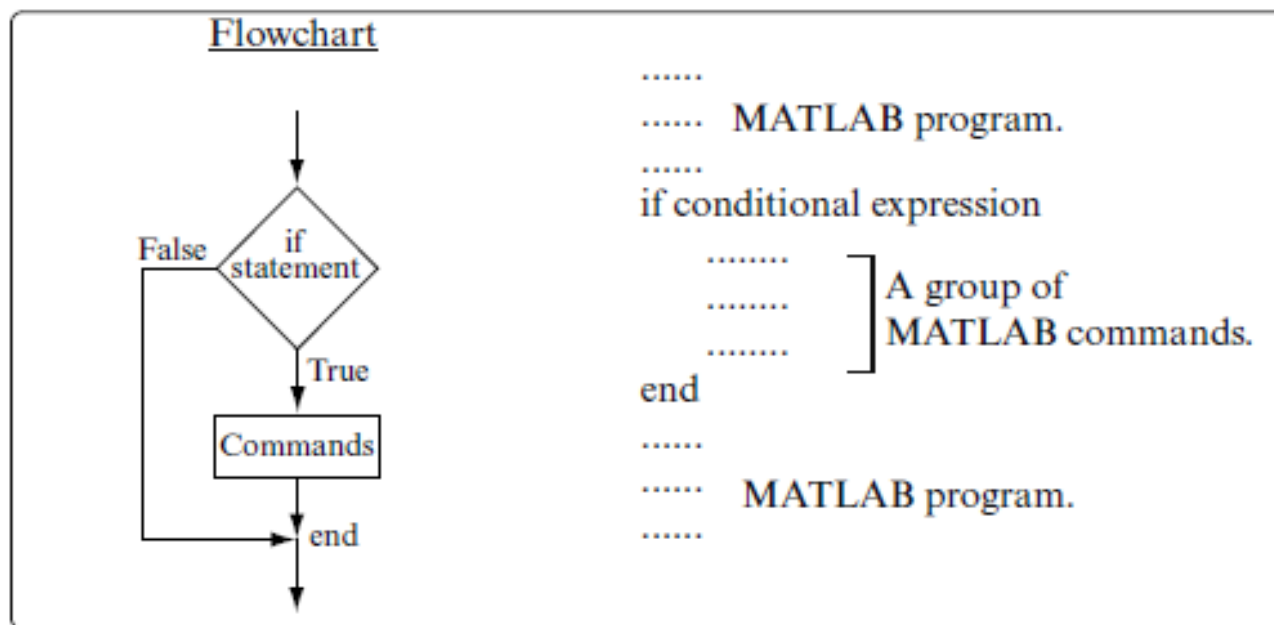
-  نشان دهنده اجرای دستورات به ترتیب است
-  نشان دهنده یک جمله if است
-  مسیر اجرای کد را نشان میدهد



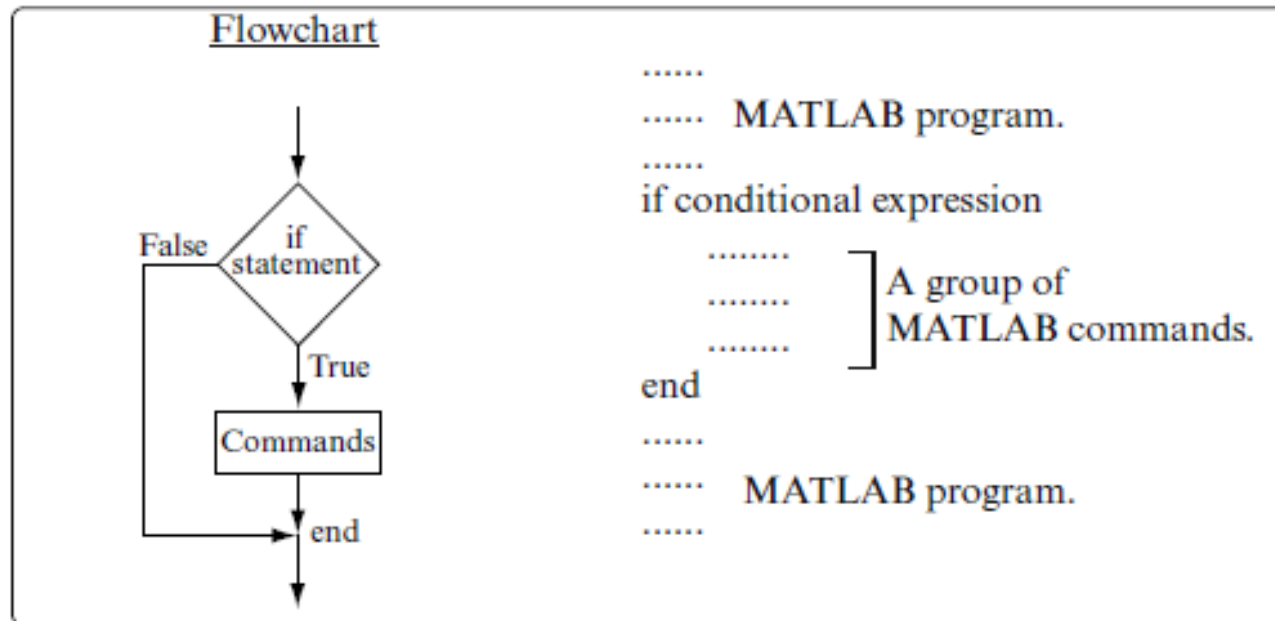
اگر عبارت شرطی درست باشد، MATLAB  
خطوطی از کد که بین خط حاوی جمله `if` و خط  
حاوی `end` است را اجرا میکند. سپس روند کد  
با اجرای دستورات قرار گرفته پس از خط حاوی  
`end` دنبال می شود.



اگر عبارت شرطی غلط باشد، MATLAB  
خطوطی از کد که بین خط حاوی if و خط حاوی  
end است را ندیده میگیرد. سپس روند کد با  
اجرای دستورات قرار گرفته پس از خط حاوی  
end دنبال می شود.



عبارت شرطی زمانی درست است که حاصل آن  
۱ منطقی یا یک عدد غیر صفر باشد. عبارت  
شرطی زمانی غلط است که حاصل آن ۰ منطقی  
یا صفر عددی باشد

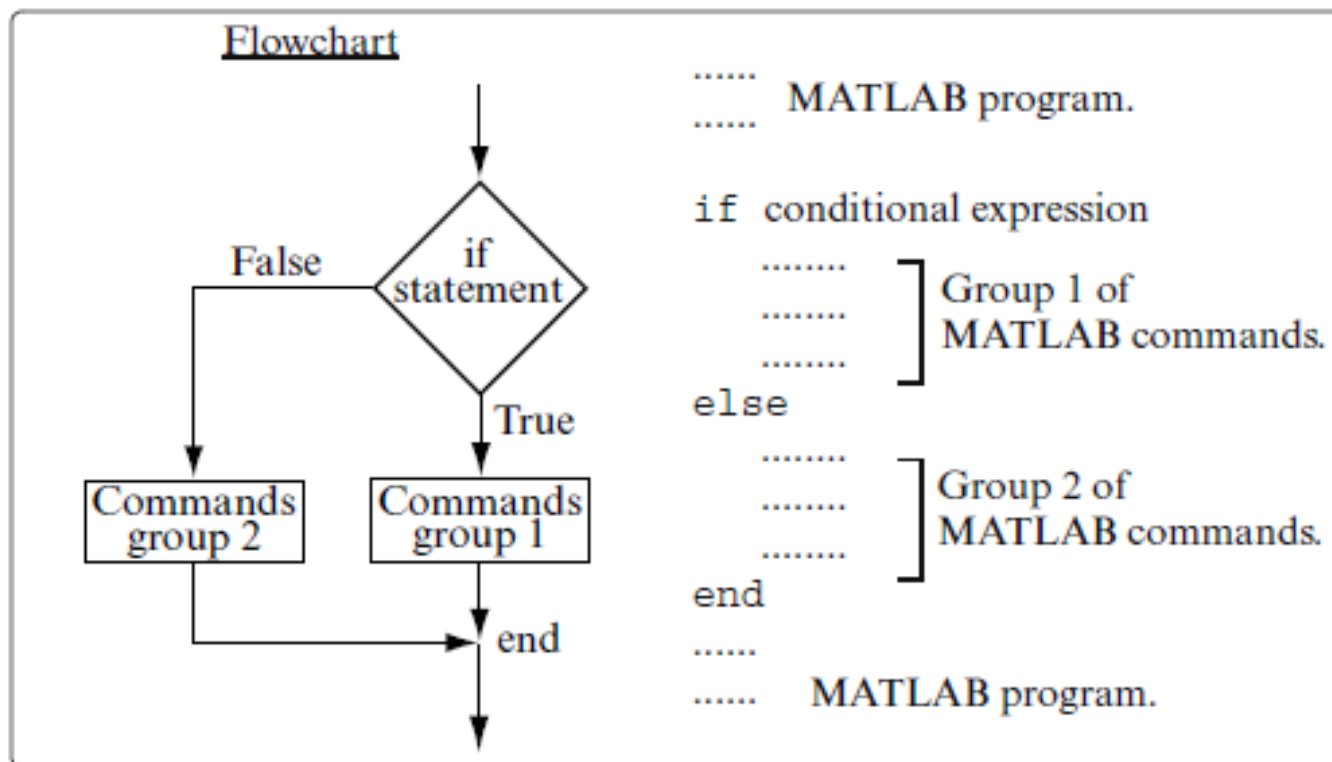


ساختار if-else-end این امکان را میدهد که قسمتی از کد زمانی که شرط درست است اجرا شود و قسمت دیگری از آن زمانی که شرط غلط است.

مثال - جواب دادن تلفن

```
if تماس گیرنده دوست شماست
    مدت زمان طولانی صحبت کن
else
    مدت زمان کوتاه صحبت کن
end
```

# تصویر ۶-۲ کد و فلوچارت ساختار if-else-end را نشان میدهد.



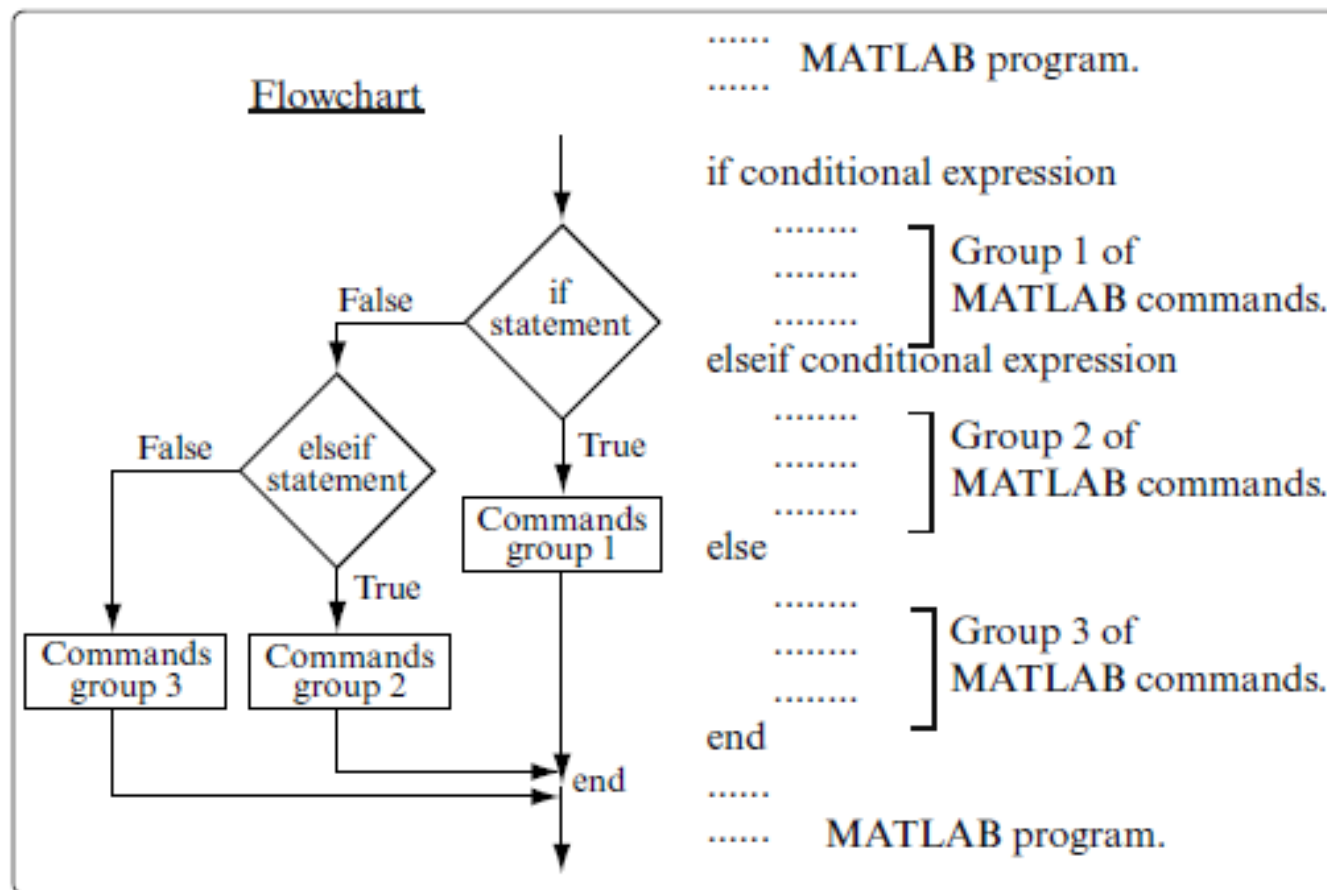
ساختار if-elseif-else-end امکان انتخاب  
بین سه (یا چند) قسمت از کد را برای اجرا می دهد  
مثال - جواب دادن تلفن

```
if تماس گیرنده دوست شماست  
    مدت زمان طولانی صحبت کن  
elseif تماس گیرنده هم گروه درسی شماست  
    تا زمانی که پاسخ اشکالها را بفهمی صحبت کن  
else  
    بگو بعداً تماس میگیری  
end
```

## میتوانید به تعداد دلخواه جمله elseif داشته باشید مثال

```
if تماس گیرنده دوست شماست
    مدت زمان طولانی صحبت کن
elseif تماس گیرنده دوست صمیمی شماست
    کمی صحبت کن و قرار ملاقات بگذار
elseif تماس گیرنده هم گروه درسی شماست
    تا زمانی که پاسخ اشکالها را بفهمی صحبت کن
else
    بگو بعداً تماس میگیری
end
```

تصویر ۳-۶ کد و فلوچارت مربوط به ساختار  
if-elseif-else-end را نشان میدهد





- میتوان جمله `else` را حذف کرد  
در این حالت، اگر شرایط جملات `if` یا `elseif` برقرار نباشد، کدی اجرا نخواهد شد

ساختار `if-elseif-else-end` در صورت استفاده از تعداد زیادی `elseif` نامفهوم خواهد شد. یک راه حل بهتر استفاده از ساختار `switch-case` است

- ساختار `switch-case` قدری متفاوت است زیرا در این حالت انتخاب کد برای اجرا بر اساس مقدار یک عدد یا رشته است، نه درست یا غلط

## اساس کار به این صورت است

```
switch name
```

```
case 'Bobby'
```

مدت زمان طولانی صحبت کن

```
case 'Susan'
```

تا زمانی که پاسخ اشکالها را بفهمی صحبت کن

```
case 'Hubert'
```

کمی صحبت کن و قرار ملاقات بگذار

```
case 'David'
```

مدت زمان کوتاه صحبت کن

```
otherwise
```

بگو بعداً تماس میگیری

```
end
```

```

.....    MATLAB program.
.....

switch switch expression
    case value1
        .....
        .....    ] Group 1 of commands.
        .....
    case value2
        .....
        .....    ] Group 2 of commands.
        .....
    case value3
        .....
        .....    ] Group 3 of commands.
        .....
    otherwise
        .....
        .....    ] Group 4 of commands.
end
.....
.....    MATLAB program.

```

```

..... MATLAB program.
.....
switch switch expression
case value1
..... ] Group 1 of commands.
case value2
..... ] Group 2 of commands.
case value3
..... ] Group 3 of commands.
otherwise
..... ] Group 4 of commands.
end
.....
..... MATLAB program.

```

# switch عبارت سوئیچ

## را بررسی میکند

- اگر مقدار آن برابر

Value1 باشد، همه

دستورات را تا case، otherwise یا end  
 بعدی اجرا میکند، یعنی دستورات گروه ۱. سپس  
 کد بعد از end اجرا میشود

- اگر مقدار آن برابر value2 باشد دستورات  
 گروه ۲ را اجرا میکند

- و به همین ترتیب

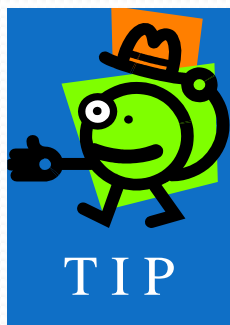
```

..... MATLAB program.
.....
switch switch expression
case value1
.....
..... ] Group 1 of commands.
case value2
.....
..... ] Group 2 of commands.
case value3
.....
..... ] Group 3 of commands.
otherwise
.....
..... ] Group 4 of commands.
end
.....
..... MATLAB program.

```

• اگر عبارت سوییچ برابر با هیچ  
کدام از مقادیر مشخص شده در  
جملات case نباشد، دستورات  
بعد از otherwise اجرا میشود  
اگر otherwise وجود نداشته  
باشد، دستوری اجرا نخواهد شد

• اگر عبارت سوییچ برابر با بیش از یک مقدار case باشد،  
تنها اولین case اجرا خواهد شد



در بررسی رشته های متنی، حروف کوچک و بزرگ مهم هستند. از دستورات `upper` و `lower` میتوانید برای بزرگ یا کوچک کردن حروف یک عبارت استفاده کنید

```
caller = lower( name );
switch caller
case 'bobby'
    some code
case 'susan'
    some code
case 'mom'
    some code
end
```

حلقه روش دیگری برای کنترل روند اجرای کد است. حلقه گروهی از دستورات را پشت سر هم اجرا میکند. MATLAB دو روش برای کنترل تعداد دفعات اجرای دستورات توسط حلقه دارد

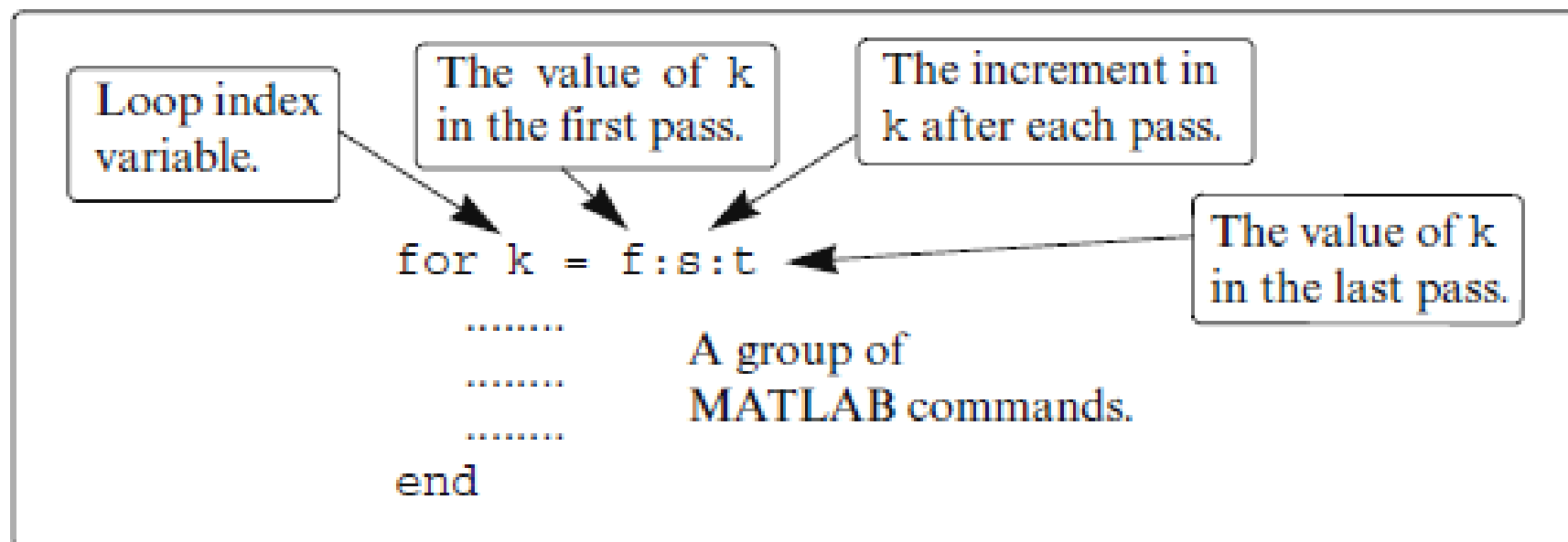
- روش ۱ - حلقه دستورات را به تعداد دفعات مشخصی اجرا میکند

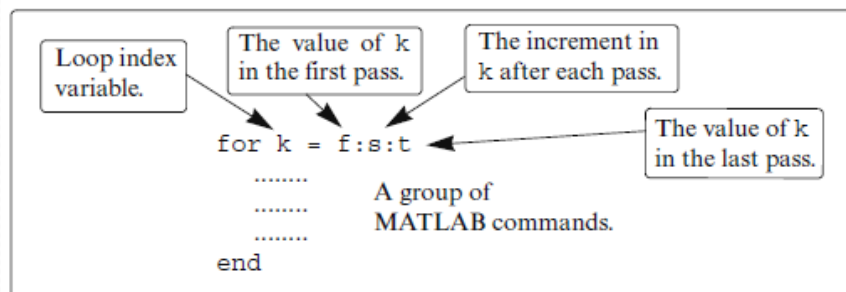
- روش ۲ - حلقه دستورات را تا زمانی که شرط مشخصی برقرار باشد اجرا میکند



یک حلقه for-end (که اغلب حلقه for نامیده میشود) گروهی از دستورات را به تعداد دفعات مشخص اجرا میکند. گروه دستورات بدنه حلقه نامیده میشود

- متغیر اندیس حلقه میتواند هر نامی داشته باشد (معمولاً از  $i, j, k, m, n$  استفاده میشود)
- زمانی که با اعداد مختلف کار میکنید از  $i$  و  $j$  استفاده نکنید. ( $ii$  و  $jj$  جایگزینهای مناسبی هستند)





1. حلقه  $k$  را برابر  $f$  قرار داده

و دستورات بین `for` و

`end`، یعنی بدنه حلقه را

اجرا میکند

2. حلقه  $k$  را برابر  $f+s$  قرار داده و بدنه را اجرا میکند

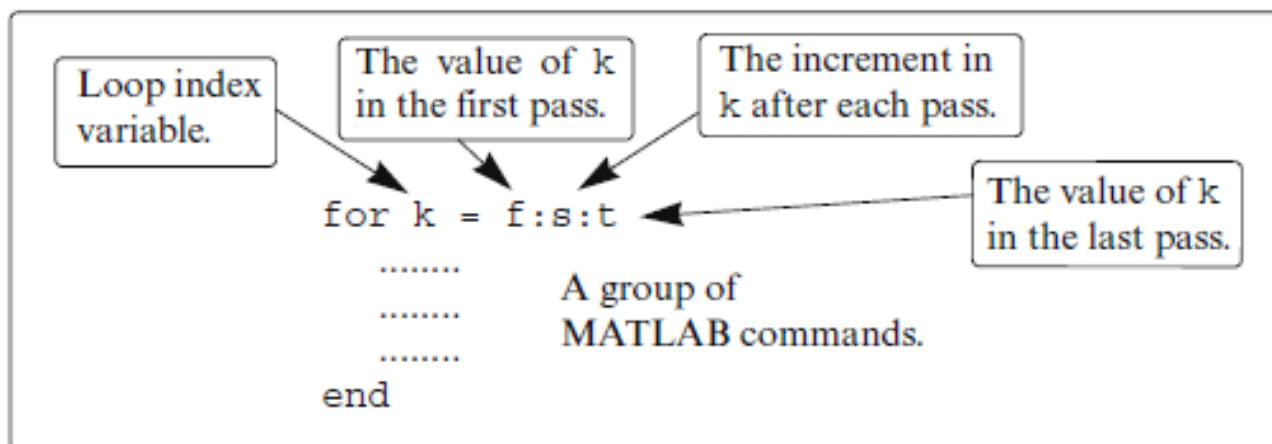
3. روند تا زمانی که  $k > t$  اجرا میشود

4. سپس برنامه با اجرای دستورات پس از `end` ادامه می یابد

•  $f$  و  $t$  معمولاً اعداد صحیح هستند

•  $s$  معمولاً استفاده نمیشود. در این حالت حلقه با گام 1 تکرار میشود

- گام  $s$  میتواند منفی باشد
- مثلاً،  $k=25:-5:10$  چهار بار با مقادیر  $k=25, 20, 15, 10$  تکرار میشود
- اگر  $f=t$  حلقه یک بار اجرا میشود
- اگر  $f>t$  و  $s>0$ ، یا اگر  $f<t$  و  $s<0$ ، حلقه اجرا نمیشود

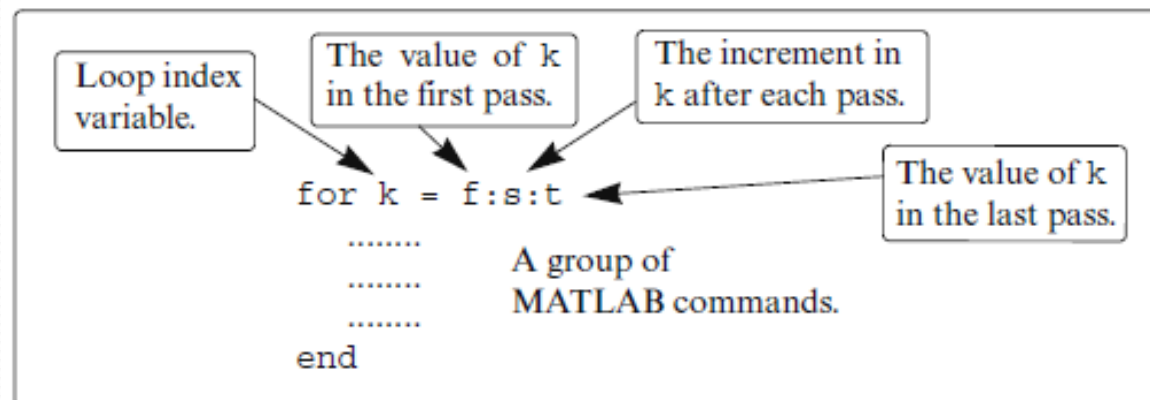


• اگر مقادیر  $k, s, t$  به گونه ای باشند که  $k$  نتواند برابر  $t$  باشد، آنگاه

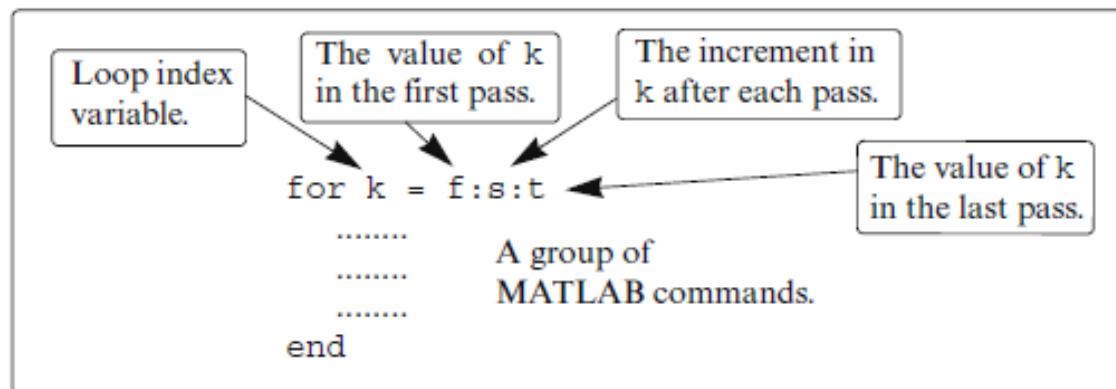
• اگر  $s$  مثبت باشد، آخرین تکرار زمانی است که  $k$  بزرگترین مقدار کمتر از  $t$  را داشته باشد

• مثلاً  $k=8:10:50$  پنج تکرار با  $k=8, 18, 28, 38, 48$  ایجاد میکند

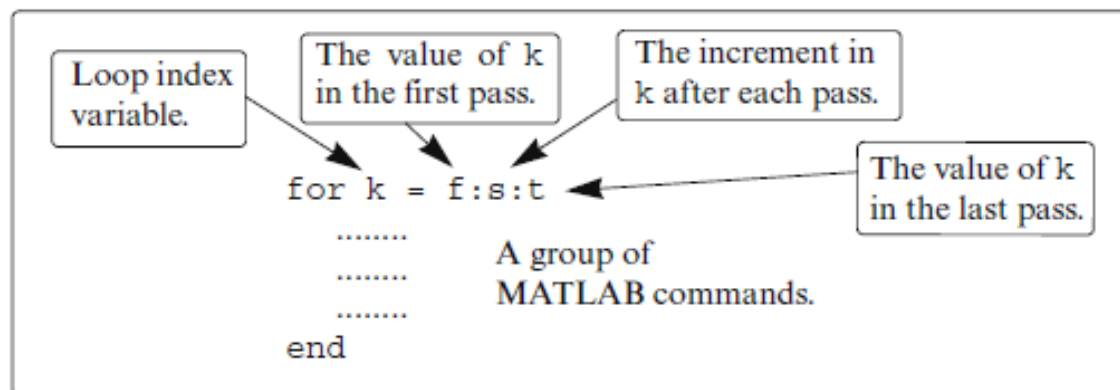
• اگر  $s$  منفی باشد، آخرین تکرار زمانی است که  $k$  کمترین مقدار بزرگتر از  $t$  را داشته باشد



- در دستور `for`، میتوان مقادیر مشخصی را به `k` اختصاص داد (که به صورت یک بردار وارد میشوند)
- مثلاً: `for k = [7 9 -1 3 3 5]`
- در حالت کلی، بدنه حلقه نباید مقدار `k` را تغییر دهد
- هر دستور `for` در یک برنامه باید حتماً یک دستور `end` داشته باشد



- مقدار متغیر اندیس حلقه ( $k$ ) به طور خودکار نمایش داده نمیشود
- مقدار  $k$  را در هر تکرار میتوان با نوشتن  $k$  به عنوان یکی از دستورات حلقه نمایش داد
- زمانی که حلقه پایان می یابد، متغیر اندیس حلقه ( $k$ ) برابر آخرین مقدار تخصیص داده شده به آن است



مثال

Script

```
for k=1:3:10
```

```
    k
```

```
    x = k^2
```

```
end
```

```
fprintf('After loop k = %d\n', k);
```

Output

```
k = 1
```

```
x = 1
```

```
k = 4
```

```
x = 16
```

```
k = 7
```

```
x = 49
```

```
k = 10
```

```
x = 100
```

```
After loop k = 10
```



اغلب برای محاسبات میتوان از یکی از دو روش حلقه for یا عملیات درایه به درایه استفاده کرد

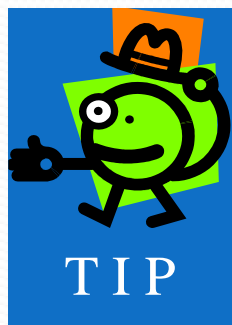
عملیات درایه به درایه:

- اغلب سریع تر است

- اغلب خوانا تر است

- برای MATLAB مناسب تر است

پیشنهاد کلی - از عملیات درایه به درایه هر زمان که میتوانید استفاده کنید. حلقه for تنها برای زمانی است که مجبور باشید.



حلقه while-end زمانی استفاده میشود که

- تعداد تکرار حلقه را نمی دانید
- شرطی دارید که میتوانید آن را امتحان کنید و هر زمان غلط بود اجرای حلقه را متوقف کنید. مثلاً
- داده ها را از یک فایل بخوانید تا زمانی که به انتهای فایل برسید
- جملاتی را به یک جمع اضافه کنید تا زمانی که تفاضل دو جمله انتهایی کمتر از مقدار مشخصی شود

```
while conditional expression
    .....
    .....      A group of
    .....      MATLAB commands.
end
```

1. حلقه عبارت  
شرطی را ارزیابی  
میکند

2. اگر عبارت شرطی درست باشد، کد بدنه را اجرا  
میکند، سپس به گام 1 باز میگردد

3. اگر عبارت شرطی غلط باشد، از کد بدنه عبور کرده  
و به کد بعد از جمله end می رود

جمله شرطی یک حلقه while-end

- دارای یک متغیر است
- بدنه حلقه باید مقدار آن متغیر را تغییر دهد
- باید مقادیری از متغیر وجود داشته باشد که به ازای آنها جمله شرطی غلط باشد

مثال

این کد

```
x = 1
```

```
while x <= 15
```

```
    x = 2*x
```

```
end
```

این خروجی را میدهد

```
x =
```

1

```
x =
```

2

```
x =
```

4

```
x =
```

8

```
x =
```

16

اگر عبارت شرطی هیچ گاه غلط نباشد، حلقه برای همیشه تکرار خواهد شد! این نوع حلقه در کتاب *حلقه نامحدود* نامیده شده است، ولی اغلب آن را *حلقه بینهایت* می نامند. برنامه شما همچنان اجرا خواهد شد، و اگر حلقه خروجی نداشته باشد (که اغلب اینطور است)، این گونه به نظر میرسد که MATLAB هنگ کرده است

علل شایع ایجاد حلقه های نامعین:

- متغیری در جمله شرطی وجود نداشته باشد

```
distance1 = 1;
```

```
distance2 = 10;
```

```
distance3 = 0;
```

```
while distance1 < distance2
```

```
    fprintf('Distance =
```

```
    %d\n', distance3);
```

```
end
```

distance2 و distance1  
هیچگاه تغییر نمیکنند

علل شایع ایجاد حلقه های نامعین:

- متغیر جمله شرطی هیچ گاه تغییر نمی‌کند

```
minDistance = 42;
```

```
distanceIncrement = 0; ← غلط تایپی - باید ۱۰ باشد
```

```
distance = 0;
```

```
while distance < minDistance
```

```
    distance = distance+distanceIncrement;
```

```
end
```



علل شایع ایجاد حلقه های نامعین:

- متغیر اشتباهی در جمله شرطی تغییر کند

```
minDistance = 42;
```

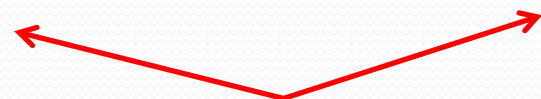
```
delta = 10;
```

```
distance = 0;
```

```
while distance < minDistance
```

```
    minDistance = minDistance + delta;
```

```
end
```



غلط تایپی-باید distance باشد

## علل شایع ایجاد حلقه های نامحدود:

- جمله شرطی هیچ گاه غلط نشود

```
minDistance = 42;
```

```
x = 0;
```

```
y = 0;
```

غلط تایپی - علامت منفی

نباید وجود داشته باشد

```
while -sqrt( x^2+y^2 ) < minDistance
```

```
    x = x + 1;
```

```
    y = y + x;
```

```
end
```



اگر برنامه در یک حلقه نامعین گیر افتاد،

- نشانگر را در پنجره فرمان قرار دهید
- CTRL+C را بزنید

اگر یک حلقه یا جمله شرطی درون یک حلقه یا  
جمله شرطی دیگر قرار گیرد، به آن حالت  
تو در تو گفته میشود

- حلقه های تو در تو رایج تر هستند
- اغلب در مسائل دوبعدی به کار می روند
- هر کدام از حلقه ها و جملات شرطی باید دارای یک  
جمله end باشند

```

n=input('Enter the number of rows ');
m=input('Enter the number of columns ');
A= [];
for k=1:n
    for h=1:m
        if k==1
            A(k,h)=h;
        elseif h==1
            A(k,h)=k;
        else
            A(k,h)=A(k,h-1)+A(k-1,h);
        end
    end
end
A

```

Define an empty matrix A.

Start of the first for-end loop.

Start of the second for-end loop.

Start of the conditional statement.

Assign values to the elements of the first row.

Assign values to the elements of the first column.

Assign values to other elements.

end of the if statement.

end of the nested for-end loop.

end of the first for-end loop.

The program is executed in the Command Window to create a  $4 \times 5$  matrix.

```

>> chap6_exp8
Enter the number of rows 4
Enter the number of columns 5

```

## دستور break:

- در داخل یک حلقه (for یا while) دستور break اجرای حلقه را متوقف میکند
- MATLAB از break به دستور end حلقه میرود، سپس اجرای برنامه با دستور بعدی ادامه پیدا میکند (به دستور for یا while آن حلقه باز نمیگردد)
- break کل اجرای حلقه را متوقف میکند، نه فقط دور جاری را.
- اگر break در یک حلقه تو در تو باشد، فقط حلقه داخلی متوقف میشود (نه حلقه های بیرونی)

- اگر دستور break در یک کد یا تابع و نه داخل یک حلقه باشد، اجرای فایل را متوقف میکند
- دستور break معمولاً همراه با یک دستور شرطی استفاده میشود
- این حالت در حلقه ها راهی برای توقف اجرای حلقه در صورت برقراری یک شرط خاص است

# مثال

نکته - ا همیشه درست است بنابراین باعث میشود حلقه همیشه تکرار شود

```
while( 1 )
    name = input( 'Type name or q to quit: ', 's' );
    if length( name ) == 1 && name(1) == 'q'
        break;
    else
        fprintf( 'Your name is %s\n', name );
    end
end
```

خروجی برای ورودی های "Greg", "quentin", "q"

```
Type name or q to quit: Greg
Your name is Greg
Type name or q to quit: quentin
Your name is quentin
Type name or q to quit: q
>>
```



### دستور continue:

از `continue` داخل یک حلقه (`while` و `for`) برای متوقف کردن تکرار جاری و رفتن به دور بعدی تکرار استفاده کنید

- `continue` معمولاً قسمتی از یک جمله شرطی است. زمانی که MATLAB به آن میرسد، بقیه دستورات حلقه را اجرا نمیکند، به `end` حلقه رفته و یک دور تکرار جدید را آغاز میکند

# مثال

```
for ii=1:100
    if rem( ii, 8 ) == 0
        count = 0;
        fprintf( 'ii=%d\n', ii );
        continue;
    end
    % code
    % more code
end
```

هر هشت تکرار count را صفر  
کن، شماره تکرار را بنویس، و از  
بقیه دستورات حلقه رد شو

# شماره تمرین های منتخب

۲۳ •

۲۴ •

۲۶ •

۲۷ •

۲۸ •

۳۰ •

۳۱ •

۳۲ •

۳۳ •

۳۴ •

۳۵ •

۷ •

۹ •

۱۱ •

۱۲ •

۱۴ •

۱۵ •

۱۷ •

۱۸ •

۱۹ •

۲۱ •

۲۲ •