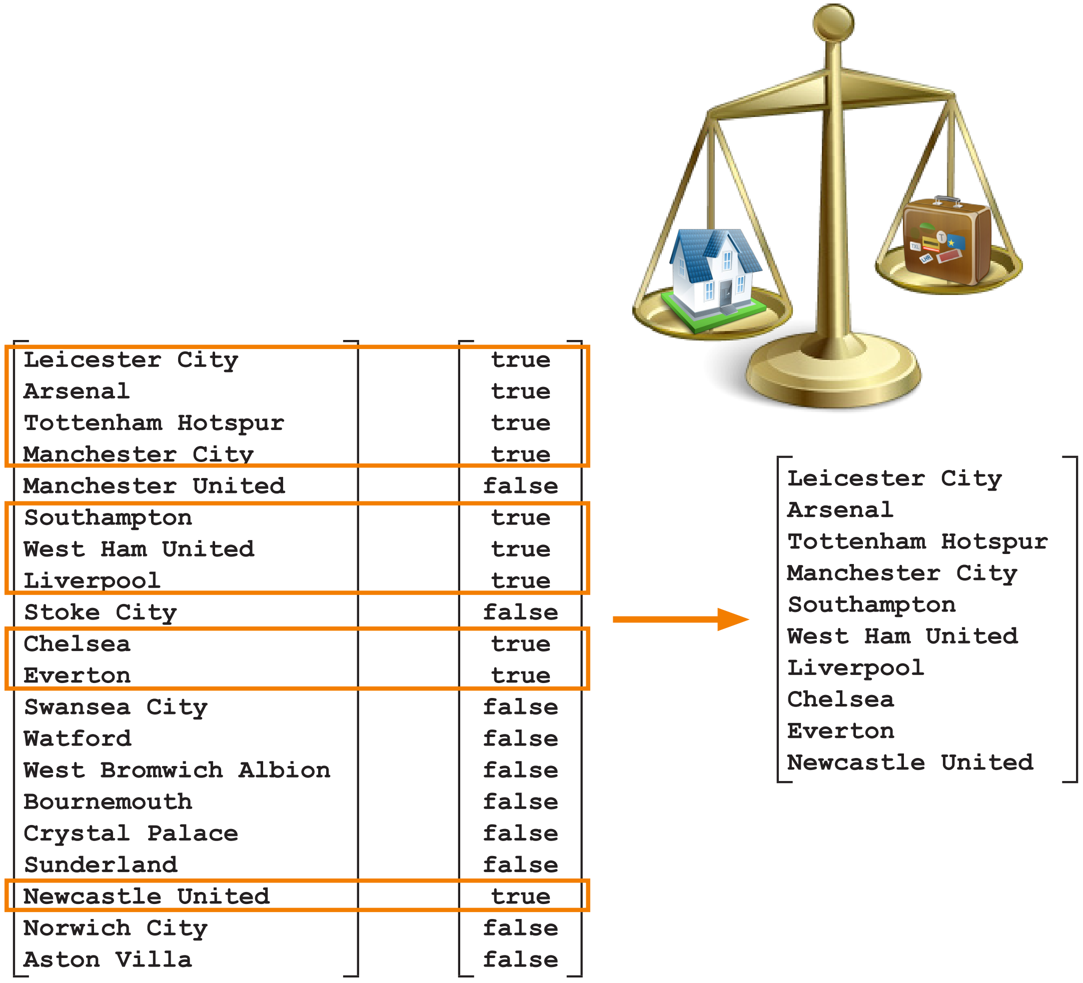
Conditional Data Selection

MATLAB® Fundamentals for Aerospace Applications

Outline

- Logical operations and variables
- Finding and counting
- Logical indexing



Leicester City	true
Arsenal	true
Tottenham Hotspur	true
Manchester City	true
Manchester United	false
Southampton	true
West Ham United	true
Liverpool	true
Stoke City	false
Chelsea	true
Everton	true
Swansea City	false
Watford	false
West Bromwich Albion	false
Bournemouth	false
Crystal Palace	false
Sunderland	false
Newcastle United	true
Norwich City	false
Aston Villa	false

```
Leicester City
Arsenal
Tottenham Hotspur
Manchester City
Southampton
West Ham United
Liverpool
Chelsea
Everton
Newcastle United
```

Chapter Learning Outcomes

The attendee will be able to:

- Perform logical operations on variables and create logical variables.
- Access and manipulate the data stored in variables using logical indexing.

Course Example: Investigating Premier League Scoring

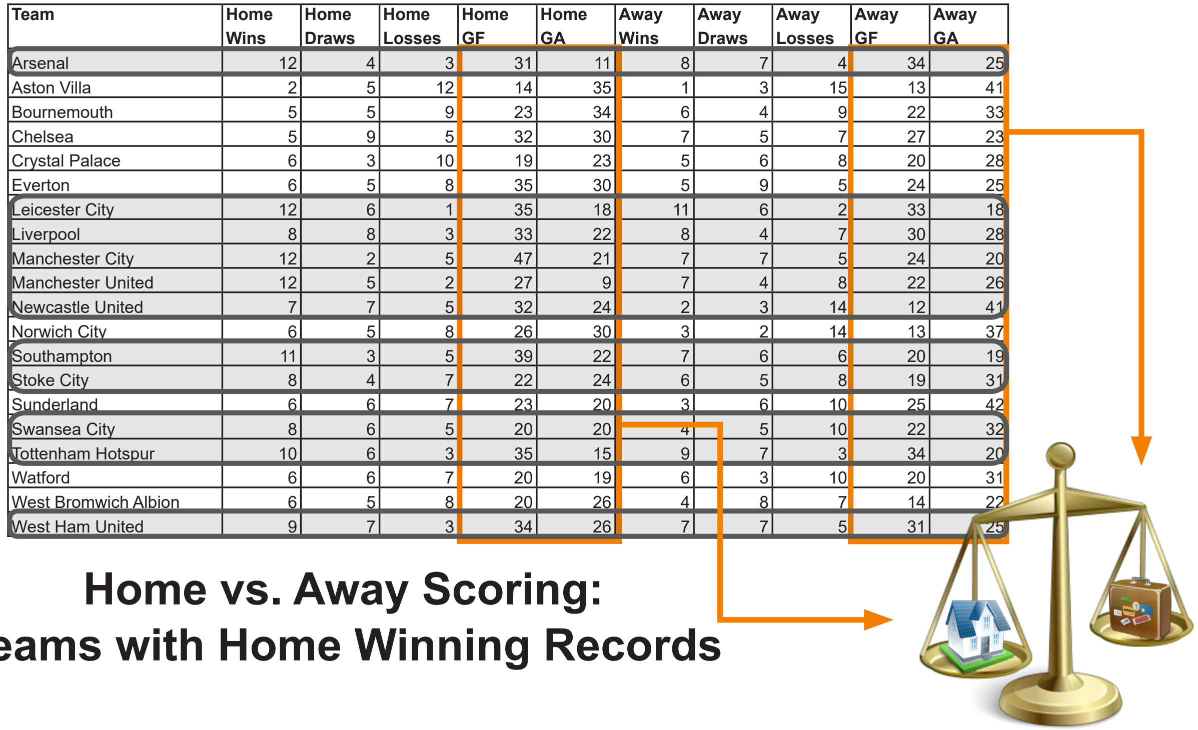
Which teams in the 2015-2016 season of the English Premier League had a home winning record, meaning they won more games at home than they lost? How many of those teams also had an away winning record?

Are there any teams that performed better away than at home?

Do teams with home winning records win more at home because they score more goals or give up fewer goals?

To answer these kinds of questions about a data set, you need to be able to extract portions of the data according to a given criterion.

This chapter illustrates how to conditionally select data. One of the most elegant constructs in the MATLAB® language is logical indexing, where a logical condition is used to index into a variable.



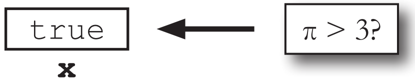
Team	Home Wins	Home Draws	Home Losses	Home GF	Home GA	Away Wins	Away Draws	Away Losses	Away GF	Away GA
Arsenal	12	4	3	31	11	8	7	4	34	25
Aston Villa	2	5	12	14	35	1	3	15	13	41
Bournemouth	5	5	9	23	34	6	4	9	22	33
Chelsea	5	9	5	32	30	7	5	7	27	23
Crystal Palace	6	3	10	19	23	5	6	8	20	28
Everton	6	5	8	35	30	5	9	5	24	25
Leicester City	12	6	1	35	18	11	6	2	33	18
Liverpool	8	8	3	33	22	8	4	7	30	28
Manchester City	12	2	5	47	21	7	7	5	24	20
Manchester United	12	5	2	27	9	7	4	8	22	26
Newcastle United	7	7	5	32	24	2	3	14	12	41
Norwich City	6	5	8	26	30	3	2	14	13	37
Southampton	11	3	5	39	22	7	6	6	20	19
Stoke City	8	4	7	22	24	6	5	8	19	31
Sunderland	6	6	7	23	20	3	6	10	25	42
Swansea City	8	6	5	20	20	4	5	10	22	32
Tottenham Hotspur	10	6	3	35	15	9	7	3	34	20
Watford	6	6	7	20	19	6	3	10	20	31
West Bromwich Albion	6	5	8	20	26	4	8	7	14	22
West Ham United	9	7	3	34	26	7	7	5	31	25

Home vs. Away Scoring: Teams with Home Winning Records

Logical Operations and Variables

The outcome of a comparison for equality or inequality is either true or false. MATLAB uses a distinct *logical* data type to represent the results of such comparisons. Logical variables have one of only two possible values: `true` and `false`, which are displayed as `1` and `0`, respectively:

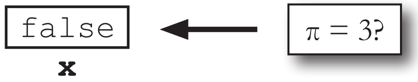
```
x = pi > 3

 x = 1
```



The equals sign is (as always) an assignment: the result of the comparison (π > 3, which is true) is assigned to the variable `x`. Although `x` appears to be numeric with value 1, the icon and the Class column in the workspace both show that `x` is actually a logical variable. (If used in a calculation, however, `x` will be converted to a numerical value of 1.)

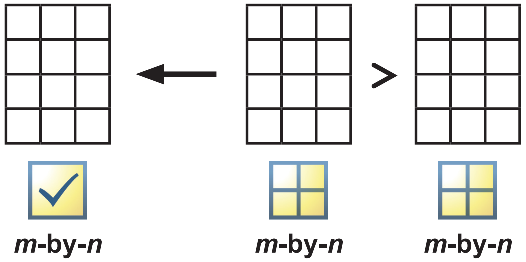
To test for equality, use a double equal sign:

```
x = pi == 3

 x = 0
```



As always, logical variables in MATLAB are assumed to be arrays, and logical operations are vectorized. Therefore,

```
z = x > y
```



compares all values of the array `x` to the corresponding values of the array `y`. The result is a logical array of the same size as `x` and `y`.

Try

Compare scoring and the number of wins and losses for each team. Note the size and type of the result.

```
load EPLresults
homegf = EPL.HomeGF;
awaygf = EPL.AwayGF;
morehomegoals = homegf > awaygf;
homewinning = EPL.HomeWins >= EPL.HomeLosses;
```

As with arithmetic operators, if `y` is a scalar, MATLAB will compare each element of `x` to `y`, again resulting in a logical array the same size as `x`.

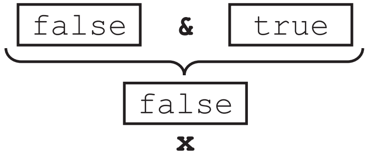
The relational operators available in MATLAB are listed in the table below. Note that these operators are designed for numeric comparison. To compare text, use the appropriate string comparison function.

Relational Operators	
`==`	Equal
`>`	Greater than
`<`	Less than
`>=`	Greater than or equal
`<=`	Less than or equal
`~=`	Not equal
String Comparison Functions	
`strcmp`	String comparison
`strcmpi`	Case insensitive comparison
`strncmp`	Partial (n-character) comparison
`strncmpi`	Case insensitive partial comparison

Combining Logical Conditions

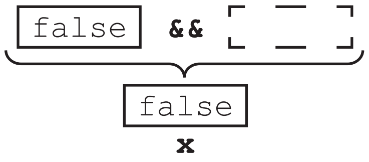
Logical relations can be combined with operators:

```
x = (pi > 5) & (0 < 6)
```



In this example, because `pi` is not greater than 5, the result must be `false`, regardless of the outcome of the second comparison. In this situation, you can use the *short-circuit* version of `&`:

```
x = (pi > 5) && (0 < 6)
```



Now the second comparison is performed only if necessary.

Logical operators, excluding the short-circuit operators, are vectorized. The command

```
z = x > y
```

compares the values of `x` to the values of `y` element-by-element, resulting in an *m*-by-*n* logical array. If this array is a vector, you can use the logical functions `all` and `any` to determine if all or any of the values are `true`:

```
any(z)
```

If `z` is a matrix, the `any` and `all` functions work columnwise, in the same manner as statistical functions such as `mean`, returning a vector of logical values.

Try

Are there any teams with both home and away winning records?

```
awaywinning = EPL.AwayWins >= EPL.AwayLosses;
winning = homewinning & awaywinning;
any(winning)
```

Did every team have a home field advantage, by either having a winning home record or scoring more goals at home than away?

```
HFA = homewinning | morehomegoals;
all(HFA)
```

Did any team score 30 or more goals at home and twice as many goals at home than away?

```
bigHFA = (homegf >= 30) & (homegf >= 2*awaygf);
any(bigHFA)
```

Logical Operators	
`&`	AND
`|`	OR
`~`	NOT
`&&`	AND (short-circuit)
`||`	OR (short-circuit)

Logical Functions	
`any`	Multiple OR
`all`	Multiple AND

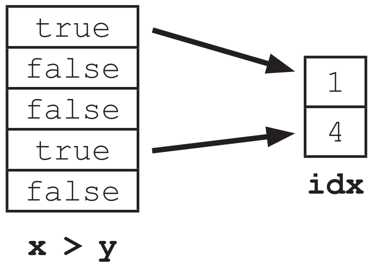
Finding and Counting

Two common operations to perform on an array are finding and counting elements that meet certain criteria. Both cases require writing a logical expression to specify the elements of interest.

The `find` function returns the indices of `true` values in a logical array. The logical array is often passed directly to `find` as a comparison that returns a logical result:

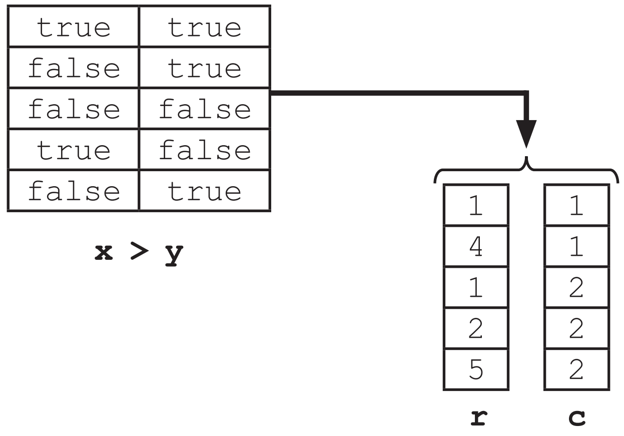
```
z = x > y
idx = find(z)
```

```
idx = find(x > y)
```



When the input to `find` is a matrix, you can ask for the locations of the `true` elements as row-column indices:

```
[r,c] = find(x > y)
```



Try

How many teams had a winning home record? How many only had a winning record at home?

```
nnz(homewinning)
nnz(homewinning & ~awaywinning)
```

What are the ranks of teams with both home and away winning records?

```
find(winning)
```

What are the ranks of the top three teams with a losing home record?

```
lhr = find(~homewinning);
lhr(1:3)
```

Because `true` and `false` values convert to `1` and `0`, respectively, one way to count the number of `true`s in a logical array is to apply the `sum` function:

```
sum(x)
```

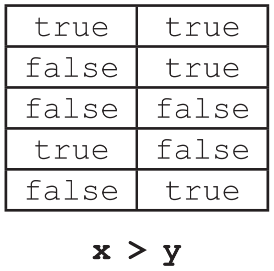
You can also use the `nnz` function, which returns the number of nonzero elements in an array:

```
nnz(x)
```

Unlike statistical functions (such as `sum`), `nnz` automatically reshapes all data into a vector, rather than acting columnwise:

```
sum(x > y)

ans =
     2    3
```

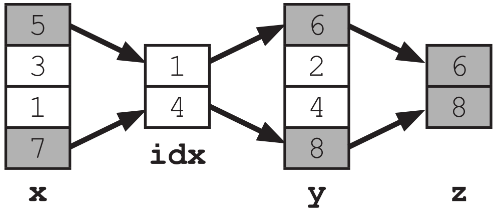
```
nnz(x > y)

ans =
     5
```



Logical Indexing

A common reason to use the `find` function is to index into other arrays:

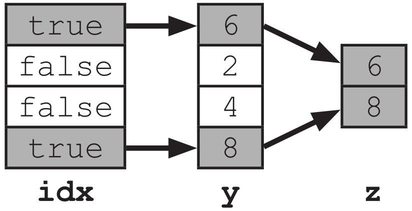
```
idx = find(x > 4)
z = y(idx)
```



If you do not actually need the indices (except to index into other variables), MATLAB provides an elegant way to index into arrays without having to use `find`: *logical indexing*.

Row, column indexing uses integer indices. However, you can use a logical variable as an array index, in which case MATLAB extracts the array elements where the index is `true`:

```
idx = [true,false,false,true]
z = y(idx)
```

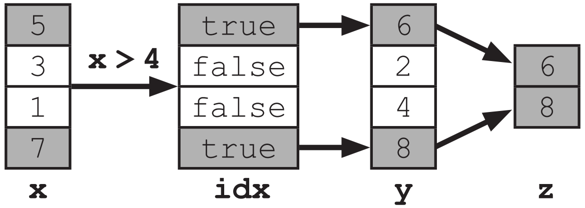


This is most commonly done by using a logical condition to create the logical index:

```
idx = (x > 4)
z = y(idx)
```

or even just

```
z = y(x > 4)
```



Try

Which teams have a winning home record?

```
EPL.Team(homewinning)
```

Get all the information on these teams.

```
EPL(homewinning,:)
```

What is the average number of home goals scored by teams with winning home records? By teams with losing home records?

```
mean(homegf(homewinning))
mean(homegf(~homewinning))
```

What gives the teams with winning home records their home advantage? Do they score more goals or concede fewer goals?

```
s = {'HomeGF','HomeGA','AwayGF','AwayGA'}
overallgoals = mean(EPL{:,s})
homewingoals = mean(EPL{homewinning,s})
homewingoals - overallgoals
```

Plot the home goal difference against the away goal difference.

```
HGD = EPL.HomeGF - EPL.HomeGA;
AGD = EPL.AwayGF - EPL.AwayGA;
scatter(HGD(homewinning),AGD(homewinning))
hold on
scatter(HGD(~homewinning),AGD(~homewinning))
plot([-30 30],[-30 30],'k:')
xlabel('Goal difference -- home')
ylabel('Goal difference -- away')
hold off
```

Summary

- Logical operations and variables
- Finding and counting
- Logical indexing

Try

Open and run the script `EPLscoring.mlx`.

Relational Operators	
`==`	Equal
`>`	Greater than
`<`	Less than
`>=`	Greater than or equal
`<=`	Less than or equal
`~=`	Not equal

String Comparison Functions	
`strcmp`	String comparison
`strcmpi`	Case insensitive comparison
`strncmp`	Partial (*n*-character) comparison
`strncmpi`	Case insensitive partial comparison

Logical Operators	
`&`	AND
`|`	OR
`~`	NOT
`&&`	AND (short-circuit)
`||`	OR (short-circuit)

Logical Functions	
`any`	Multiple OR
`all`	Multiple AND

Test Your Knowledge

Name:____

1. If `x` and `y` are both 20-by-1 (numeric) vectors and half of the elements of `y` are greater than 0.5, the command

```
y>0.5
```

will return:

A. A 10-by-1 numeric vector of `y` values

B. A 20-by-1 logical vector

C. A 20-by-1 numeric vector of `y` values

D. An error message

2. If `x` and `y` are both 20-by-1 (numeric) vectors and half of the elements of `y` are greater than 0.5, the command

```
x(y>0.5)
```

will return:

A. A 10-by-1 numeric vector of `x` values

B. A 10-by-1 numeric vector of `y` values

C. A 20-by-1 logical vector

D. A 20-by-1 numeric vector of `x` values

E. An error message