

# Logicals

# Logicals

---

- ▶ a logical expression is an expression that evaluates to either true or false
- ▶ a logical variable is a variable whose value is either true or false
- ▶ logical variables are usually called Boolean variables in computer science

# Logicals in MATLAB

---

- ▶ MATLAB uses the numbers 1 and 0 to represent true and false
  - ▶ that is, every logical expression will evaluate to either exactly 1 (true) or 0 (false), and the value of every logical variable will be either exactly 1 or 0
- ▶ however, MATLAB will convert any non-zero, non-NaN numeric value to logical 1
  - ▶ only values equal to exactly 0 are converted to logical 0

# Creating a logical variable

---

- ▶ the literals for logical true and false are the non-keywords **true** and **false**
  - ▶ however, these are rarely used (most people use 1 and 0)

```
>> x = true
```

```
x =
```

```
1
```

```
>> y = false
```

```
y =
```

```
0
```

# Relational operators

---

- ▶ more commonly, logical values arise from logical expressions usually involving a relational operator
- ▶ relational operators produce logical values by comparing two numbers

operator	name
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
==	equal to
~=	not equal to

# Relational operators

---

- ▶ the relational operators will operate in an element by element fashion for arrays
- ▶ you can also compare a scalar versus an array

```
>> ones(3, 3) > zeros(3, 3)
```

```
ans =
```

```
    1    1    1
    1    1    1
    1    1    1
```

```
>> 5 < ones(2, 2)
```

```
ans =
```

```
    0    0
    0    0
```

# Relational operators

---

- ▶ great care must be taken when comparing floating-point values for equality (or non-equality)

```
>> x = 0.1 + 0.1 + 0.1;
```

```
>> x == 0.3
```

```
ans =
```

```
0
```

```
>> x = 0.5 - 0.4 - 0.1;
```

```
>> x ~= 0
```

```
ans =
```

```
1
```



# Logical operators

---

- ▶ logical operators operate on logical values
- ▶ there are 5 logical operators and 1 function

Operator	name
<code>~</code>	<b>not</b>
<code>&amp;</code>	elementwise <b>and</b>
<code>&amp;&amp;</code>	short-circuit scalar <b>and</b>
<code> </code>	elementwise <b>or</b>
<code>  </code>	short-circuit scalar <b>or</b>
<code>xor</code>	elementwise <b>exclusive or</b>

# not ~

---

- ▶ ~ is Boolean negation (often called 'NOT')
  - ▶ NOT **true** is equal to **false**
  - ▶ NOT **false** is equal to **true**

expression	result
NOT <b>true</b>	<b>false</b>
NOT <b>false</b>	<b>true</b>

```
>> x = 1:5
```

```
x =
```

```
    1    2    3    4    5
```

```
>> ~(x > 2)           % parentheses needed (precedence)
```

```
ans =
```

```
    1    1    0    0    0
```

```
>> exist('x')
```

```
ans =
```

```
    1
```

```
>> ~exist('x')
```

```
ans =
```

```
    0
```

```
>> x = 1:5
```

```
x =
```

```
    1    2    3    4    5
```

```
>> any(x > 5)
```

```
ans =
```

```
    0
```

```
>> ~any(x > 5)
```

```
ans =
```

```
    1
```



# & elementwise and

---

- ▶ & is Boolean conjunction (often called 'AND') applied elementwise

expression	result
true AND true	true
true AND false	false
false AND true	false
false AND false	false

```
>> x = [1 1 0 0];
>> y = [1 0 1 0];
>> x & y
ans =
     1     0     0     0

>> I = imread('cameraman.tif');
>> imshow(I);
>> figure;
>> imhist(I);
>> figure;
>> imshow(I > 64 & I < 192);
```

# | elementwise or

---

- ▶ | is Boolean disjunction (often called 'OR') applied elementwise

expression	result
true OR true	true
true OR false	true
false OR true	true
false OR false	false

```
>> x = [1 1 0 0];
>> y = [1 0 1 0];
>> x | y
ans =
     1     1     1     0

>> I = imread('cameraman.tif');
>> imshow(I);
>> figure;
>> imhist(I);
>> figure;
>> imshow(I < 64 | I > 192);
```



# Scalar AND and OR

---

- ▶ the scalar versions of AND and OR try to minimize the number of comparisons that are computed
- ▶ consider the logical expression

**( x > 0 ) AND ( x < 10 )**

- ▶ if **( x > 0 )** is **false** then there is no need to evaluate **( x < 10 )**
  - ▶ because the overall expression must also be **false**

# Scalar AND and OR

---

- ▶ similiary, consider the logical expression

**(x < 0) OR (x > 10)**

- ▶ if **(x < 0)** is **true** then there is no need to evaluate **(x > 10)**
  - ▶ because the overall expresssion must also be **true**

# Scalar AND and OR

---

- ▶ the scalar versions of AND and OR ensure that the extra comparison is never performed

# Logical indexing

---

- ▶ you can use a logical array to perform indexing on another array
  - ▶ MATLAB extracts the array elements corresponding to the nonzero values in the logical array
    - ▶ the output is always in the form of a column vector unless the array is a vector

```
>> x = 1:5
```

```
x =
```

```
    1    2    3    4    5
```

```
>> I = x > 3
```

```
I =
```

```
    0    0    0    1    1
```

```
>> x(I)
```

```
ans =
```

```
    4    5
```

```
>> X = pascal(5)
```

```
X =
```

```
    1    1    1    1    1
    1    2    3    4    5
    1    3    6   10   15
    1    4   10   20   35
    1    5   15   35   70
```

```
>> I =
```

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

```
>> x(I)
```

```
ans =
```

```
15
```

```
20
```

```
35
```

```
15
```

```
35
```

```
70
```

```
>> % rectify a sine wave
>> t = 0:0.05:1;
>> y = sin(t);
>> plot(t, y, 'b'); hold on;
>> I = y < 0;
>> y(I) = -y(I);
>> plot(t, y, 'r');

>> % replace all spaces with -
>> s = 'a string with some spaces in it';
>> s(isspace(s)) = '-';

s =
a-string-with-some-spaces-in-it
```