For-loop comparisons (1)

Matlab

```
for k = 1:n
    % ...
end
```

Python

```
for k in range(1, n + 1):
    # ...
```

C99, C++, Java

```
for (int k = 1; k <= n; ++k) {
    // ...
}
```

Fortran 77

```
      INTEGER k
      DO 10 k = 1, n
C         ...
   10 CONTINUE
```

Ada

```
for k in 1 .. n loop
   -- ...
end loop;
```

For-loop comparisons (2)

Matlab

```
for k = 1:n
    % ...
end
```

Scala

```
for (k ← 1 to n) {
    // ...
}
```

Rust

```
for k in 1..=n {
    // ...
}
```

OCaml

```
for k = 1 to n do
    (* ... *)
done
```

LISP

```
(loop for k from 1 to n
   do ; ...
)
```

Perl

```
foreach my $k (1..n) {
    # ...
}
```

Announcements/Agenda

- Assignment 1 posted; due Sep 13

Su	M	Tu	W	Th	F	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

- (review) How to make decisions
  - `if`/`elseif`/`else`, relational & Boolean operators
- How to repeat until something happens
  - `while`
- How to see what you're doing
  - `plot`
- How to make lists
  - Vectors

`fprintf()`

- Format specifiers: `%f`, `%e`, `%s`
- Fixed point: `%8.3f`
  - 8 columns, right-aligned
  - Tenths, hundredths, & thousandths decimal places
  - Fits up to -999.999
- Floating-point: `%.3e`
  - 4 sig-figs
- New line: `\n`

If your output will be read by both people and machines, *always* use

`%.17g`

Otherwise, *Chaos* could ensue.

Boolean expressions: relational operators

- A boolean value is either true (1) or false (0)
- Obtain boolean values by comparing things
- Operators only act on two things at once – don't try to chain them

Symbol	Comparison
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
~=	Not equal to

`a < x < b` does *not* do what it looks like

Logical operators "short-circuit"

a > b && c > d

true

Go on

a > b && c > d

false

Stop

Entire expression is false since the first part is false

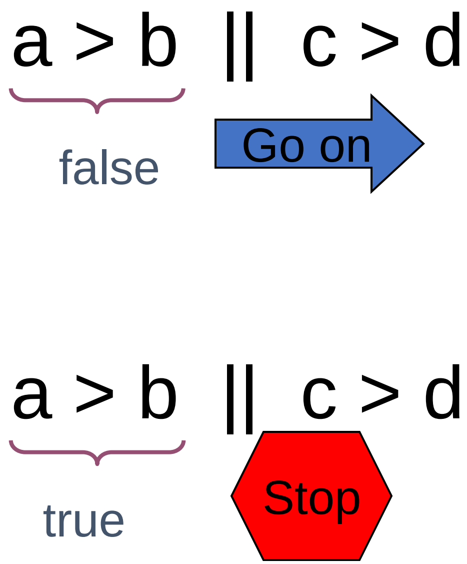
A **&&** expression short-circuits to false if the left operand evaluates to *false*.

A || expression short-circuits to ____________________ if

____

____

Logical operators "short-circuit"



Entire expression is true since the first part is true

A **&&** expression short-circuits to false if the left operand evaluates to *false*.

A **||** expression short-circuits to true if the left operand evaluates to *true*.

Why short-circuit?

- Right-hand Boolean expression may be *expensive* or potentially *invalid*
- Much clearer than alternatives

```
if (x < 0.5) || (tan(x) < 1)
    % ...
end

if (x ~= 0) && (y/x > 1e-8)
    % ...
end
```

Last time: Monte Carlo estimator for π

for N_{darts} trials:

generate random dart location

if dart is in circle:

count as a hit

estimate π as $4\ N_{hits} / N_{darts}$

- Goal: draw blue hits, red misses

if dart is in circle:

draw blue dot

otherwise:

draw red dot

Application 1: Draw blue and red darts

- *Draw red star:* `plot(x, y, 'r*')`
- *Draw blue star:* `plot(x, y, 'b*')`
- *Don't erase old points:* `hold on`
- *Preserve geometry:* `axis equal`

Application 2: Estimate π via annulus

- New math

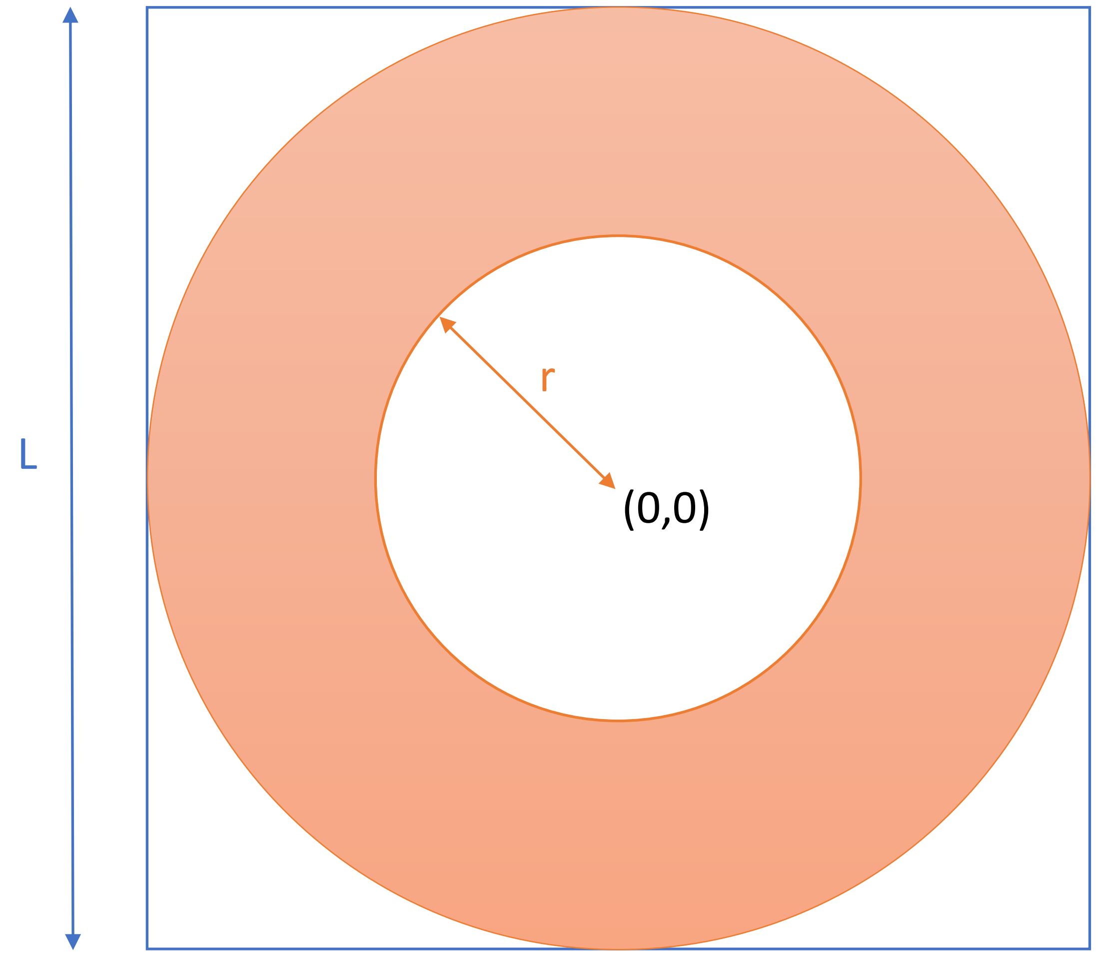
$$P \approx N_{hits}/N_{darts}$$

$$\pi = P/(¼ - (r/L)^2)$$

- New condition

```
(x^2 + y^2 < (L/2)^2) && ...
(x^2 + y^2 > r^2)

~((x^2 + y^2 > (L/2)^2) || ...
  (x^2 + y^2 < r^2))
```



Application 3: Stop when we're close

- A `for`-loop always repeats a fixed number of times
  - There are ways to leave a loop early, but they're not used in this class
- Want to stop repeating when a Boolean expression changes value
  - *"Are we there yet?"*
  - Matlab can do this: `while`-loop
- BUT a `for`-loop gave us a counter for free
  - Need to make our own

While-loops in place of for-loops

```
N= ___; L= ___; hits= 0;

for k= 1:N
   % Throw kth dart
   x = rand*L – L/2;
   y = rand*L – L/2;
   % Count if in circle
   if x^2 + y^2 <= (L/2)^2
      hits= hits + 1;
   end

end
myPi= 4*hits/N;
```

```
N= ___; L= ___; hits= 0;
k= 1;
while k <= N
   % Throw kth dart
   x = rand*L – L/2;
   y = rand*L – L/2;
   % Count if in circle
   if x^2 + y^2 <= (L/2)^2
      hits= hits + 1;
   end
   k= k + 1;
end
myPi= 4*hits/N;
```

Repeating something *N* times

```
for k= 1:N
    % Do something
    ...


end
```

```
% Initialize loop variables
k= 1;
while k <= N
    % Do something
    ...
    % Update loop variables
    k= k+1;
end
```

Common loop patterns

Do something N times

```
for k= 1:N
    % Do something
    ...
end
```

Do something an indefinite number of times

```
% Initialize loop variables

while not stopping signal
    % Do something
    ...
    % Update loop variables
    ...
end
```

Storing dart positions

- Don't want to declare *N* different variables
  - What if *N* changes? Comes from user input?
  - How to change variable name in each loop iteration?
- Need a list

Arrays

The basic variable in Matlab is a matrix

- Scalar: 1×1 matrix
- 1-D array of length 4:
  - 1×4 matrix (row vector) or 4×1 matrix (column vector)
- 2-D array: a matrix, naturally

Array indexing: starts at 1

x	5	0.4	.91	-4	-1	7
	1	2	3	4	5	6

Let x be a vector and k be an index. Then:

- k must be a positive integer
- `1 <= k && k <= length(x)`
- To access the k^th element: `x(k)`
  - Read: `y = x(k)`
  - Write: `x(k) = y`

Creating vectors

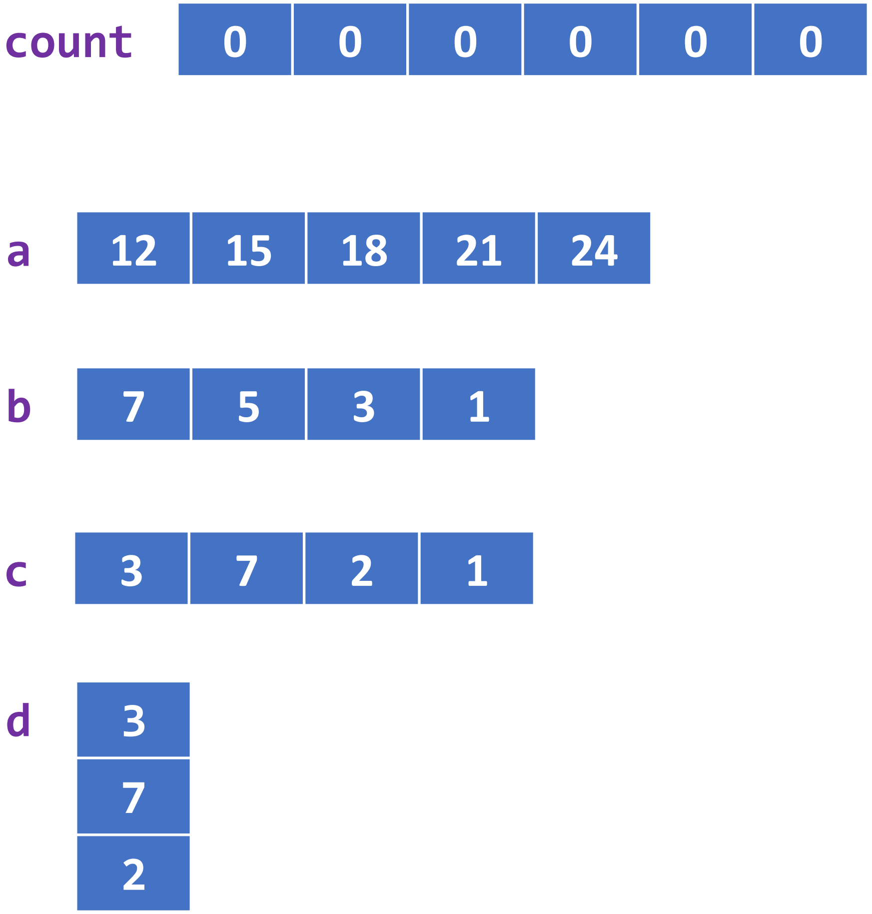
```
count= zeros(1,6)

a= linspace(12,24,5)

b= 7:-2:0

c= [3 7 2 1]

d= [3; 7; 2]
```



Example: cumulative sum

- Write a program fragment that calculates the cumulative sums of a given vector v.
- The cumulative sums should be stored in a vector of the same length as v.

```
1, 3, 5, 0  v
1, 4, 9, 9  cumulative sums of v
```

```
csum(1) = v(1);
csum(2) = ?
csum(k) = ?
```