Example: $n$-gon → circle

Inscribed hexagon: $(n/2) \sin(2\pi/n)$
Circumscribed hexagon: $n \tan(\pi/n)$

As $n$ approaches infinity, the inscribed and circumscribed areas approach the area of a circle. How big should $n$ be?

---

Find $n$ such that $\text{outerA}$ and $\text{innerA}$ converge

First, itemize the tasks:
- define how close is close enough
- select an initial $n$
- calculate $\text{innerA}$, $\text{outerA}$ for current $n$
- $\text{diff} = \text{outerA} - \text{innerA}$
- close enough?
- if not, increase $n$, repeat above tasks

---

Find $n$ such that $\text{outerA}$ and $\text{innerA}$ converge

Now organize the tasks → algorithm:

$n$ gets initial value

Repeat until tolerance is reached:
- calculate $\text{innerA}$, $\text{outerA}$ for current $n$
- $\text{diff} = \text{outerA} - \text{innerA}$
- increase $n$

---

Common loop patterns

<table>
<thead>
<tr>
<th>Do something $n$ times</th>
<th>Do something an indefinite number of times</th>
</tr>
</thead>
<tbody>
<tr>
<td>for $k = 1:n$</td>
<td>%Initialize loop variables</td>
</tr>
<tr>
<td>% Do something</td>
<td>while ( not stopping signal )</td>
</tr>
<tr>
<td>end</td>
<td>% Do something</td>
</tr>
<tr>
<td></td>
<td>% Update loop variables</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
</tbody>
</table>

---

Important Features of Iteration

- A task can be accomplished if some steps are repeated; these steps form the loop body
- Need a starting point
- Need to know when to stop
- Need to keep track of (and measure) progress

---
for-loop or while-loop: that is the question

- for-loop: loop body repeats a fixed (predetermined) number of times. The "increment" is fixed.

- while-loop: loop body repeats an indefinite number of times under the control of the "loop condition."

Example: Times Table

Write a script to print a times table for a specified range.

Row headings: 3 4 5 6 7

Column headings: 3 | 12 | 15 | 18 | 21
4 | 12 | 16 | 20 | 24 | 28
5 | 15 | 20 | 25 | 30 | 35
6 | 18 | 24 | 30 | 36 | 42
7 | 21 | 28 | 35 | 42 | 49

Developing the algorithm for the times table

```
3 4 5 6 7
3 9 12 15 18 21
4 12 16 20 24 28
5 15 20 25 30 35
6 18 24 30 36 42
7 21 28 35 42 49
```

Find the min value in a sequence

- Find the min in a sequence of positive numbers. User enters the numbers one-by-one and indicates the end of the sequence by entering a negative value.

Example:

```
Enter a positive # (negative # to stop): 5
Enter a positive # (negative # to stop): 3
Enter a positive # (negative # to stop): 25
Enter a positive # (negative # to stop): 3.4
Enter a positive # (negative # to stop): -1
Min value entered was 3.000000
```
Is it prime?
- Write a program fragment to determine whether a given integer \( n \) is prime. Assume \( n > 1 \).
- Hint: \( \text{mod}(x, y) \) returns the remainder of \( x \) divided by \( y \).

Are they prime?
- Given integers \( a \) and \( b \), sketch a program that lists all the prime numbers in the range \([a, b]\).
- Assume \( a, b > 1 \) and \( a < b \).

Find the mode in a sequence
- A mode is the number in a sequence that appears the most number of times.
- Develop an algorithm for calculating the mode of a user-entered sequence that is
  - Non-negative
  - Entered one-by-one in non-decreasing order
  - Terminated by a negative number
- E.g., sequence 87, 92, 92, 98, 98, 98, 100 has a mode...
- Write the algorithm and then the code on your own for practice! Do this after section this week.

The savvy programmer...
- learns useful programming patterns and use them where appropriate.
- Seeks inspiration by working through test data “by hand”
  - Asks, “What am I doing?” at each step
  - Declares a variable for each piece of information maintained when working the problem by hand
- Decomposes the problem into manageable subtasks
  - Refines the solution iteratively, solving simpler subproblems first
- Remembers to check the problem’s boundary conditions
- Validates the solution (program) by trying it on test data