

# Recursion

CSC 112  
Spring 2010

## Objectives

- To understand:
  - what recursion is
  - what a base case is
  - what a recursive case is
  - what causes a recursive algorithm to stop calling itself

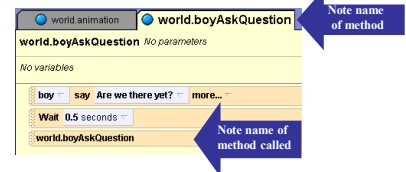
## Today's Agenda

1. Introduction to Recursion
2. Problem Solving with Recursion



## Introduction to Recursion

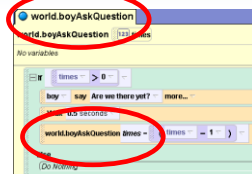
- Most methods call other methods
- Recursion** is when a method calls itself



- Similar to a loop

## Introduction to Recursion

- Note the call is made as long as the number of **times** it is called is  $> 0$
- Each time the call is made **times** is reduced by 1
- At some point, **times** is equal to 0 which means it is no longer  $> 0$  (If statement is no longer true and the “loop” stops)



## Are We There Yet?



## Tower of Hanoi

✦ <http://www.mazeworks.com/hanoi/index.htm>



## Problem Solving with Recursion

- ✦ Problems can be solved with recursion if...
  - it can be broken down into successive smaller problems that are identical to the overall problem
- ✦ Driving over the river and thru the woods to Grandmother's house (75 miles away).
  - Child asks every 5 miles: "are we there yet"
    - if miles to Grandma = 0
    - then child says we're here!!!
    - else
    - miles driven ahead is 5 (distance to Grandma's is 5 less)
    - check to see if we're at Grandma's!

## Problem Solving with Recursion

- ✦ Recursion is never absolutely required to solve a problem
  - any problem that can be solved with recursion can also be solved with a loop
  - Driving over the river and thru the woods to Grandmother's house (75 miles away).
    - Child asks every 5 miles: "are we there yet"
      - while miles to Grandma > 0
      - check to see if we're at Grandma's!
      - miles driven ahead is 5 (distance to Grandma's is 5 less)
      - end while
    - child says we're here!!!

## Recursive Methods

- ✦ **Base case**
  - where the problem can be solved WITHOUT recursion
- ✦ **Recursive case**
  - solving problems using recursion
  - the problems MUST always be reduced to smaller version of the original problem
- ✦ By reducing the problem with each recursive call, the base case will eventually be reached and the recursion will stop

## Recursive Methods

- ✦ A recursive method works as follows:
  - if the problem **can** be solved NOW, with recursion, then the method solves it and returns
  - if the problem **cannot** be solved NOW:
    - the method reduces it to a smaller, similar problem
    - and calls itself to solve the smaller problem

## Recursion in the KentuckyDerby

- ✦ What is the base case in the Horse Race?
- ✦ What is the recursion?

## Using Recursion in Mathematical Problems

✦  $n!$

✦ factorial!

- If  $n = 0$  then  $\text{factorial}(n)=1$
- If  $n > 0$  then  $\text{factorial}(n) = 1 \times 2 \times 3 \times \dots \times n$

✦ When  $n$  is 0, its factorial is 1

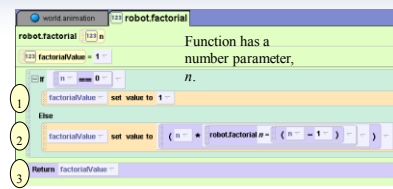
✦ When  $n$  is greater than 0, its factorial is the product of ALL positive integers from 1 up to  $n$

- $\text{factorial}(6) = 1 \times 2 \times 3 \times 4 \times 5 \times 6$

## Recursion in Factorial!

- ✦ What is the base case?
- ✦ What is the recursion?

## Using Recursion in Mathematical Problems



1. factorialValue is set at 1 and is executed if  $n = 0$

2. factorialValue variable is set to  $n$  times the factorial of  $n-1$ . This is the recursive case and is executed if  $n$  is  $\neq 0$

3. returns the value variable

## Using Recursion in Mathematical Problems

First call of the function
Value of $n$ : 3
Return value: 6

Third call of the function
Value of $n$ : 1
Return value: 1

Fourth call of the function
Value of $n$ : 0
Return value: 1

- ✦ Argument of 3 passed into  $n$
- ✦ Because  $n$  is  $\neq 0$ , the statement's else clause executes:

$\text{factorialValue.set value to } (n * \text{robot.factorial}(n - 1))$

- ✦ This instruction sets the value of factorialValue
- ✦  $\text{robot.factorial}(n - 1)$  value determines the value stored in a variable