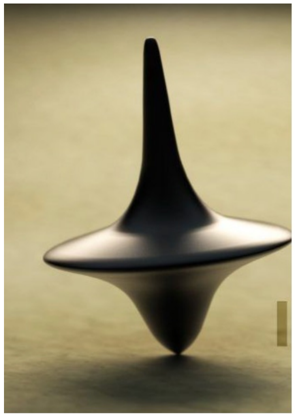
# Lecture 14: Recursion

CS 51P

October 24, 2022

Tom Yeh
he/him/his



# Class News

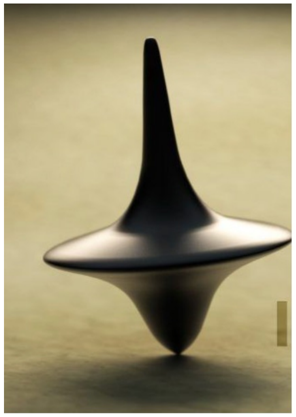
- Image manipulations lab deadline extended to Tue 10/25

# Learning Goals

- Recursion

# What is recursion?

- Wikipedia: ”Recursion occurs when a thing is defined in terms of itself.”
- A technique for tackling large or complicated problems by taking 1 “bite” of the problem at a time
  - Divide and conquer



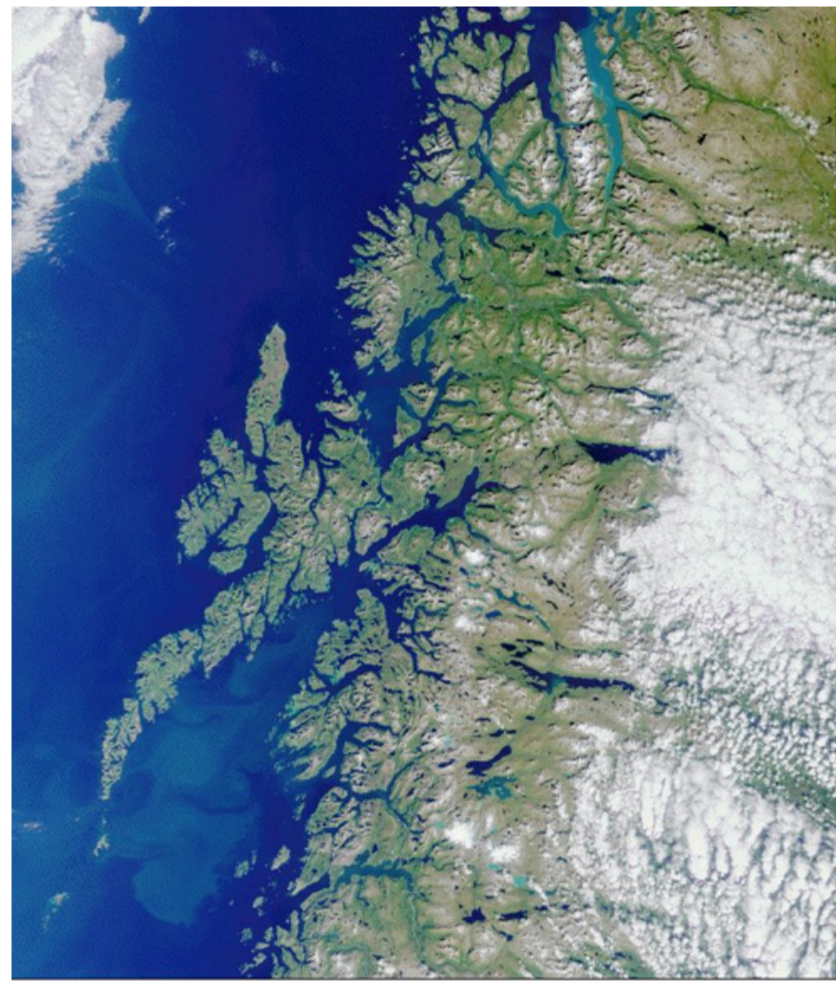
# What is recursion?

- A powerful substitute for iteration (loops)
  - Start by seeing the difference between iteration vs recursion
  - Some problems can only be solved using recursion
- Results in elegant, shorter code when used well
- Often applied to sorting and searching problems

# What is recursion?

- Can be used to express patterns seen in nature
- Object containing smaller copies of itself



# How many students are in class?

- If I want to find out how many people are in class today, but I don't want to walk around and count each person.

- I am recruiting you to help, but I also want to minimize each student's amount of work.

# How many students are in class?

- If I want to find out how many people are in class today, but I don't want to walk around and count each person.

- I am recruiting you to help, but I also want to minimize each student's amount of work.

- We can solve this problem recursively!

# How many students are in class?

- Let’s focus on solving the problem for a single column of students.

# How many students are in class?

- Let's focus on solving the problem for a single column of students.

- I will ask the first person in the front row: "How many people are sitting directly behind you in your column?"

# How many students are in class?

- Student's algorithm:
  - If there is no one behind me, answer 0.
  - If someone is sitting behind me:
    - Ask that person: "How many people are sitting directly behind you in your column?"
    - When they respond with a value N, respond (N + 1) to the person who asked me.

# How many students are in class?

- Student's algorithm:
  - If there is no one behind me, answer 0.
  - If someone is sitting behind me:
    - Ask that person: "How many people are sitting directly behind you in your column?"
    - When they respond with a value N, respond (N + 1) to the person who asked me.

- Can generalize to the entire classroom!

# 2 main components of recursion

- 1) Base case
  - The simplest version of your problem that all other cases reduce to
  - An occurrence that can be answered directly

# 2 main components of recursion

- 1) Base case
  - The simplest version of your problem that all other cases reduce to
  - An occurrence that can be answered directly

  - What’s the base case for the demo?

# 2 main components of recursion

- 1) Base case
  - The simplest version of your problem that all other cases reduce to
  - An occurrence that can be answered directly
  - What’s the base case for the demo?
- 2) Recursive case
  - The step where you break down more complex versions of the task into smaller occurrences
  - Cannot be answered directly
  - What is the recursive case for the demo?

# Recursion overview

- Reduce problem into repeated, smaller tasks of the same form
- Recursion has 2 main parts: **base case** and **recursive case**
- Solution is built up as you come back up the call stack
- When solving recursively, look for self-similarity and think about what info is stored in each stack frame
- Take the “recursive leap of faith” and trust the smaller tasks will solve the problem for you!

# Factorial example

- The number n factorial, n! in math notation, is
  - n x ( n – 1) x … x 3 x 2 x 1

# Factorial example

- The number n factorial, n! in math notation, is
  - n x ( n – 1) x … x 3 x 2 x 1
- For example:
  - 3! = 3 x 2 x 1 = 6
  - 5! = 5 x 4 x 3 x 2 x 1 = 120
  - 0! = 1 ( by definition)

# Factorial example

- The number n factorial, n! in math notation, is
  - n x ( n – 1) x … x 3 x 2 x 1
- For example:
  - 3! = 3 x 2 x 1 = 6
  - 5! = 5 x 4 x 3 x 2 x 1 = 120
  - 0! = 1 ( by definition)
- Let’s implement the factorial function!

# Factorial function

- 5! = 5 x 4 x 3 x 2 x 1

# Math view of factorials

- n! = 1 if n = 0
- n! = n x (n – 1)! Otherwise

- Convert to code:

# Recursion in action

- Stack frame – one gets created each time a function is called
- Stack is where information is stored in computer’s memory
- Every time we call factorial(), we get a new copy of the local variable n
- Stack frames go away once they return

# Recursion review

- Reduce problem into repeated, smaller tasks of the same form
- Recursion has 2 parts: **base case** and **recursive case**
  - Each part may have multiple cases
- Solution is built up as you come back up the call stack
- When solving recursively, look for self-similarity and think about what info is stored in each stack frame

# Exercise: isPalindrome

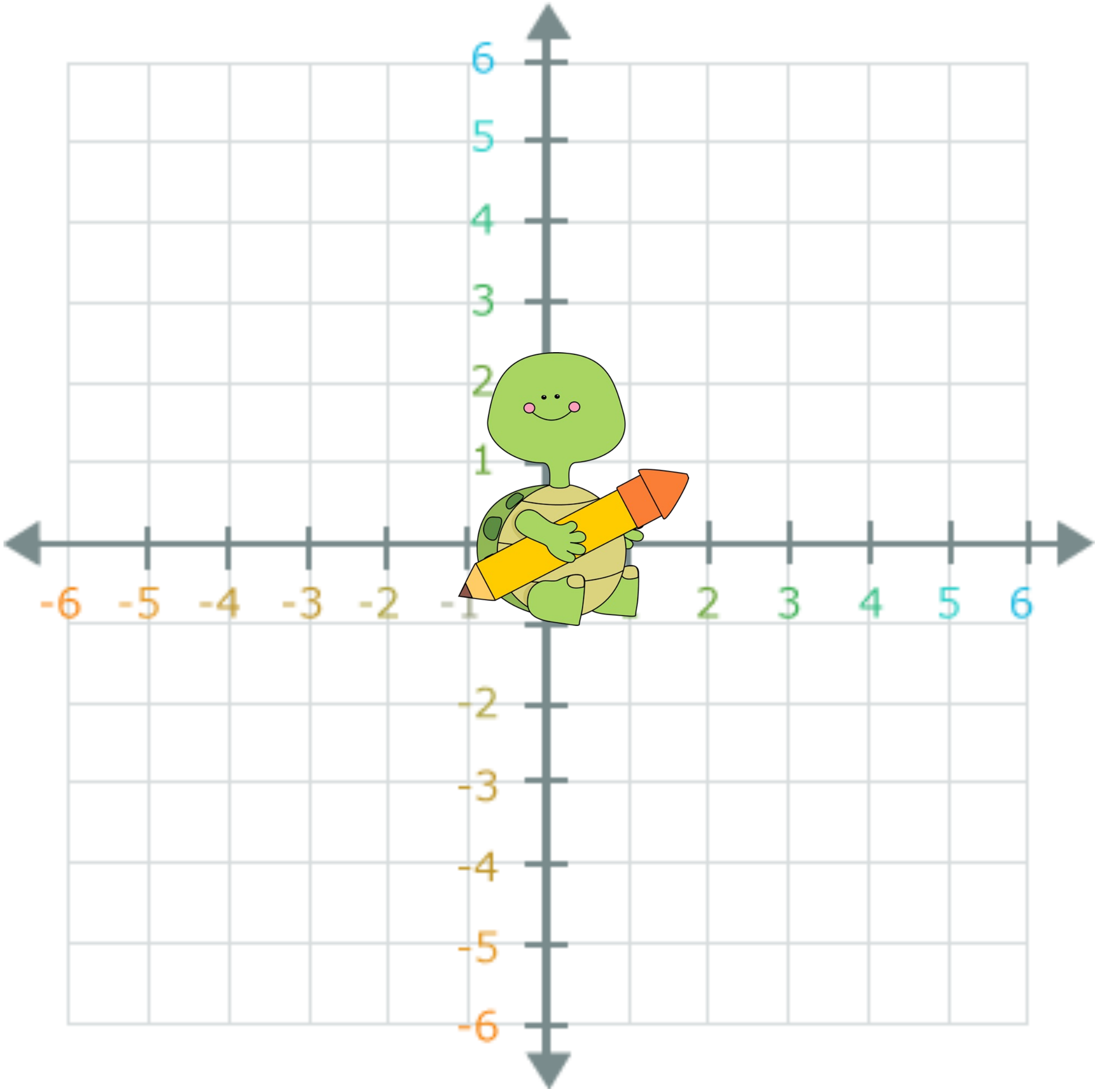
- Write a recursive function to check if a string is a palindrome
- Palidrome is word, number, phrase or other sequence of symbols that reads the same backwards and forwards.

C
ANNA
CIVIC
RACECAR
STEP ON NO PETS
STRESSED DESSERTS

# isPalindrome

- Base cases:
- Recursive case:

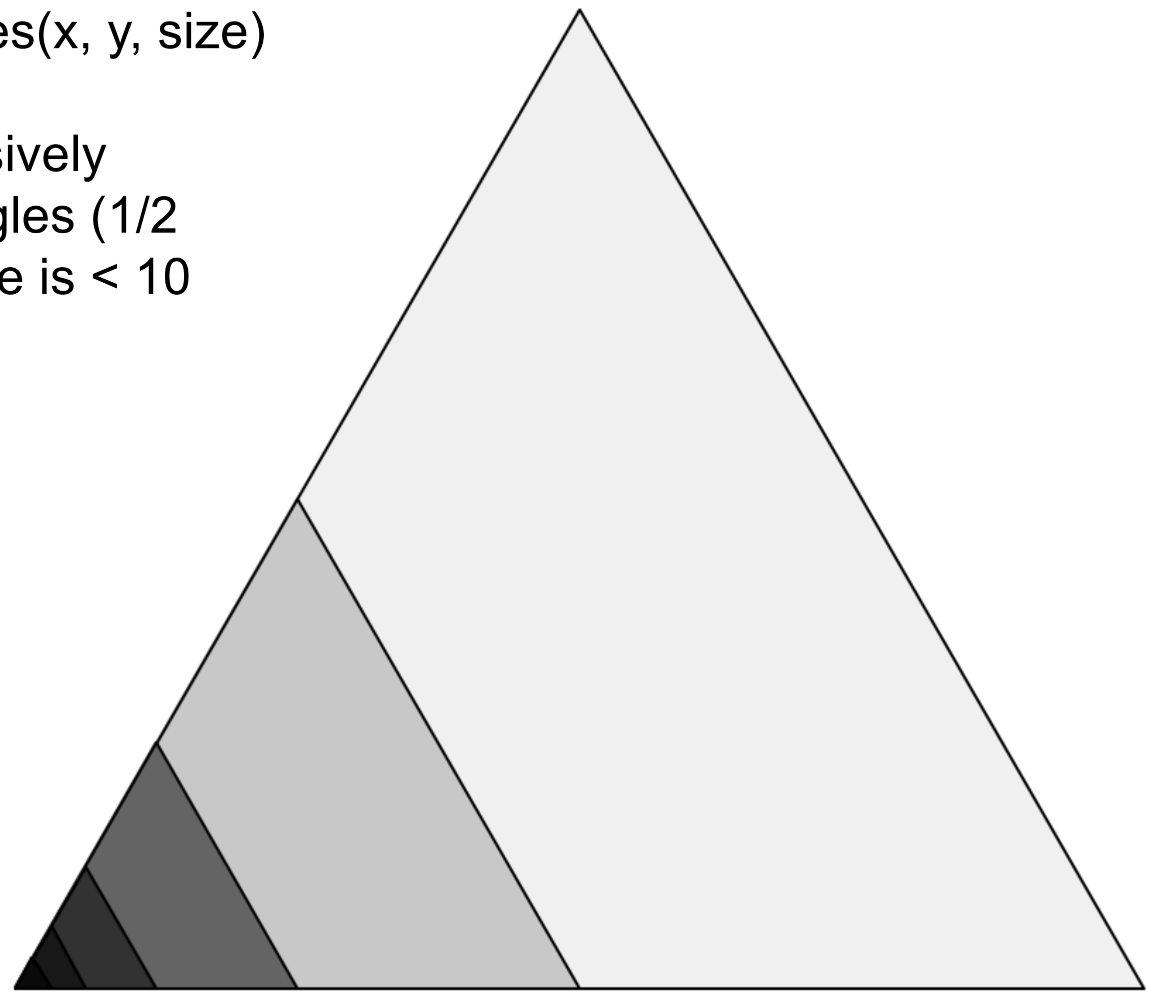
# Turtle graphics



# Example - Recursive Graphics

draw_triangles(x, y, size)

Draws recursively smaller triangles (1/2 size) until size is < 10



# Counting Triangles