SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 120: Introduction to Computing Science and Programming 1

Recursion: Functions That Call Themselves

Reminders

One-Stop Access To Course Information

- **Course website**: **One-stop access** to all course information.

  http://www2.cs.sfu.ca/CourseCentral/120/liaqata/WebSite/index.html

- Course Outline
- Exam Schedule
- Python Info
- CourSys/Canvas link
- Learning Outcomes
- Office Hours
- Textbook links
- and more…
- Grading Scheme
- Lab/Tutorial Info
- Assignments

- **Canvas**: Discussions forum - https://canvas.sfu.ca/courses/39187
- **CourSys**: Assignments submission, grades - www.coursys.sfu.ca

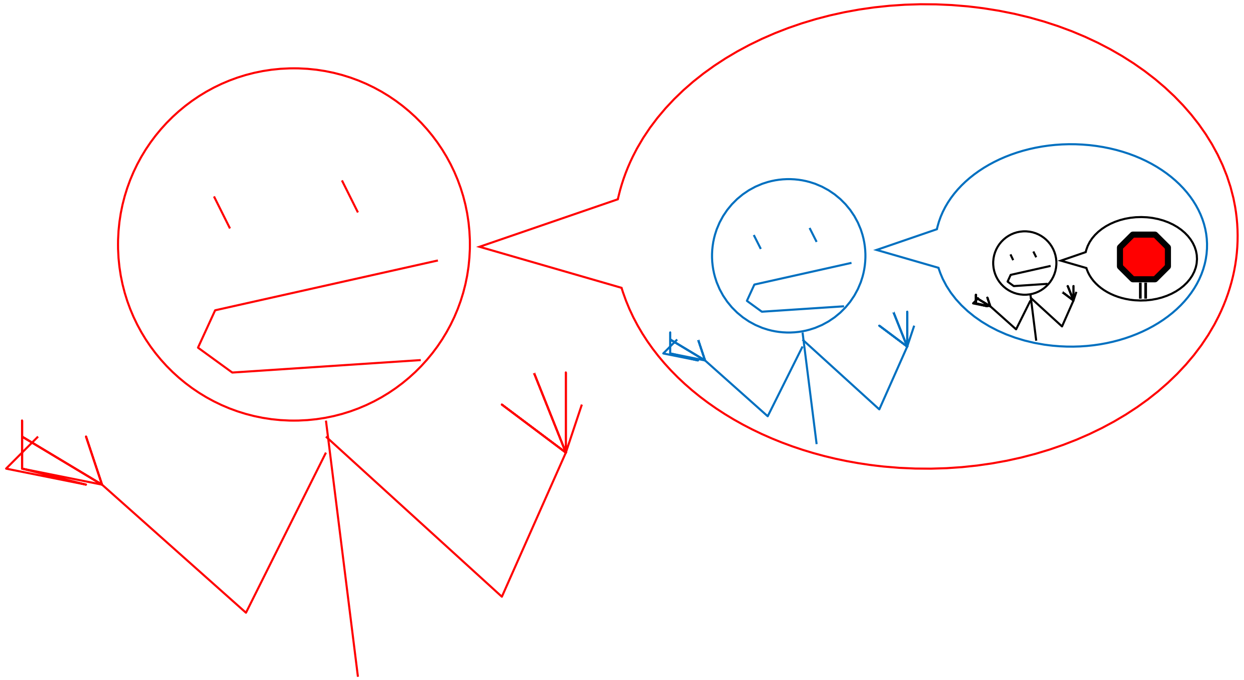
Course Topics

1. General introduction
2. Algorithms, flow charts and pseudocode
3. Procedural programming in Python
4. Data types and Control Structures
5. Binary encodings
6. Fundamental algorithms
7. Basics of (Functions and) Recursion (Turtle Graphics)
8. Basics of computability and complexity
9. Subject to time availability:
   - Basics of Data File management

Today's Topics

1. Introduction to Recursion
2. Problem Solving with Recursion
3. Examples of Recursive Algorithms

Recursion



Introduction to Recursion

- **Recursion**: A way of coding an algorithm without needing loops.
  - Instead, a problem is solved by recursively calling the function itself.
- **Recursive function**: a function that calls itself.
- A recursive function:
  1. **must** have a function call to itself.
  2. **must** have a **way to control** the number of times it calls itself/repeats.
     - Usually involves an **if-else** statement which defines:
       - when the function should end (base case), and
       - when it should call itself.
- **Depth of recursion**: The number of times a function calls itself.
- Read: The Three Laws of Recursion

Introduction to Recursion: Factorial Example

- Let's consider a factorial problem.

0! = 1
1! = 1 x 1
2! = 2 x 1 x 1
3! = 3 x 2 x 1 x 1
4! = 4 x 3 x 2 x 1 x 1

0! = 1
1! = 1 x 0!
2! = 2 x 1!
3! = 3 x 2!
4! = 4 x 3!

Introduction to Recursion: Factorial Example

- In mathematics, the $n!$ notation represents the factorial of a number n
  - For $n = 0$, $n! = 1$
  - For $n > 0$, $n! = 1 \times 2 \times 3 \times \ldots \times n$
- The above definition lends itself to recursive programming
  - $n = 0$ is the base case
  - $n > 0$ is the recursive case
    - factorial(n) = n x factorial(n-1)

Introduction to Recursion

```
# Factorial by loop
# setup variables
number = 4
repeat = 1
factorial = 1

# find factorial using a loop method
while repeat <= number:
    factorial = factorial * repeat
    repeat+=1

print(factorial)
```

```
# Factorial by recursion
# define a recursive function
def get_factorial(number):
    # Base case and recursive calls
    if number < 1:     # base case
        factorial= 1
        return factorial
    else:
        factorial = number * get_factorial(number - 1)
        return factorial

# Main
factorial = get_factorial(number)
print(factorial)
```

Introduction to Recursion: Tree Example

```
import turtle
# A recursive function to draw a tree
def draw_tree(level, branch_length):
    #As long as we are not at the leaf level
    if level>0:
        #1. Draw a branch
        turtle.forward(branch_length)
        #2. Turn left and make a mini tree
        turtle.left(40)
        draw_tree(level-1, branch_length/1.61)
        #3. Turn back
        turtle.right(40)
        #4. Turn righ and make a mini tree
        turtle.right(40)
        draw_tree(level-1, branch_length/1.61)
        #4. Go back
        turtle.left(40)
        turtle.back(branch_length)
    # Otherwise
    else:
        # Stop the leaf
        turtle.color("green")
        turtle.stamp()
        turtle.color("brown")
```

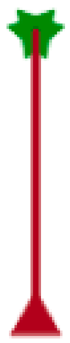
A recursive function

The function call that **starts it all!**

```
# Main program

# Move the turtle
turtle.speed(0)
turtle.penup()
turtle.goto(0, -180)
turtle.left(90)
turtle.pendown()

# Setup drawing
turtle.color("brown")
turtle.width(3)
turtle.shape("triangle")

# Call the draw tree function
draw_tree(1, 120)
```

Introduction to Recursion: Tree Example

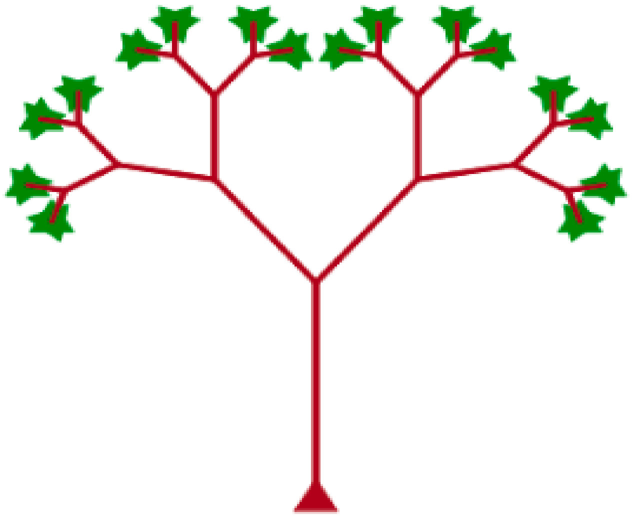
```
# Call the draw tree function
draw_tree(1, 120)
```



```
# Call the draw tree function
draw_tree(2, 120)
```

Introduction to Recursion: Tree Example

```
# Call the draw tree function
draw_tree( , 120)
```



```
# Call the draw tree function
draw_tree( , 120)
```

How would you modify the function to get this tree?

Direct and Indirect Recursion

- **Direct recursion**: when a function directly calls itself
  - All the examples shown so far were of direct recursion
- **Indirect recursion**: when function A calls function B, which in turn calls function A

Problem Solving with Recursion

- Recursion is a powerful tool for solving repetitive problems
- Recursion is never required to solve a problem.
  - Any problem that can be solved recursively can be solved with a loop.
  - Recursive algorithms usually less efficient than iterative ones.
    - Due to overhead of each function call.

Recursion versus Looping

- **Reasons not to use recursion**:
  - Less efficient: entails function calling overhead that is not necessary with a loop
  - Usually a solution using a loop is more evident than a recursive solution
- Some problems are more easily solved with recursion than with a loop
  - Example: Fibonacci, where the mathematical definition lends itself to recursion.

More Examples of Recursive Algorithms

- Summing a range of list elements with recursion
  - Function receives a list containing range of elements to be summed, index of starting item in the range, and index of ending item in the range
  - Base case:
    - `if start index > end index return 0`
  - Recursive case:
    - `return current_number + sum(list, start+1, end)`

More Examples of Recursive Algorithms (cont'd.)

```
# The range_sum function returns the sum of a specified
# range of items in num_list. The start parameter
# specifies the index of the starting item. The end
# parameter specifies the index of the ending item.
def range_sum(num_list, start, end):
    if start > end:
        return 0
    else:
        return num_list[start] + range_sum(num_list, start + 1, end)
```

The Fibonacci Series

- Fibonacci series: has two base cases
  - `if n = 0 then Fib(n) = 0`
  - `if n = 1 then Fib(n) = 1`
  - `if n > 1 then Fib(n) = Fib(n-1) + Fib(n-2)`
- Corresponding function code:

```
# The fib function returns the nth number
# in the Fibonacci series.
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)
```

Finding the Greatest Common Divisor

- Calculation of the greatest common divisor (GCD) of two positive integers
  - If x can be evenly divided by y, then
  - gcd(x,y) = y
  - Otherwise, gcd(x,y) = gcd(y, remainder of x/y)
- Corresponding function code:

```
# The gcd function returns the greatest common
# divisor of two numbers.
def gcd(x, y):
    if x % y == 0:
        return y
    else:
        return gcd(x, x % y)
```

?

Questions?