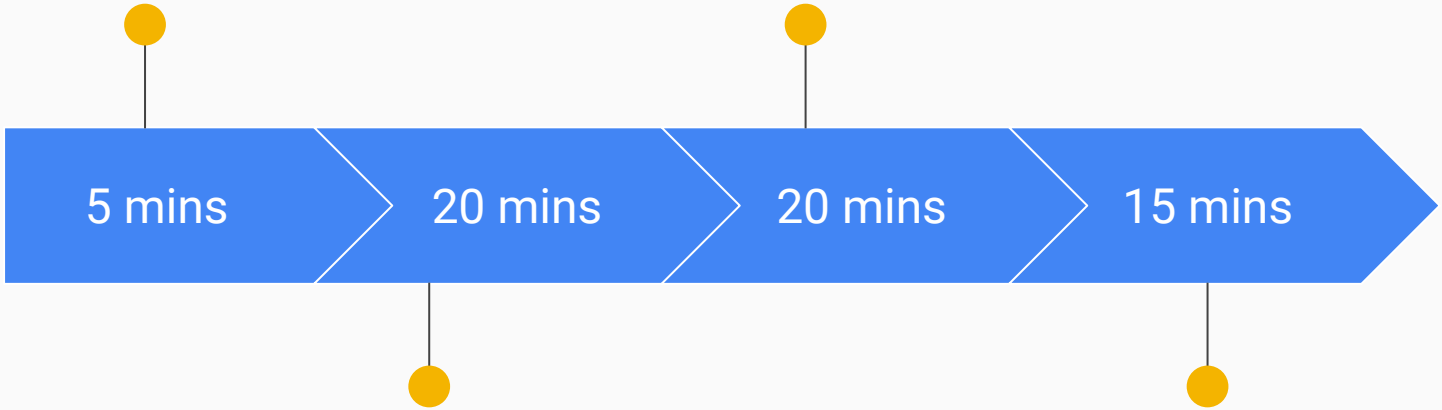
Introduction: what is recursion?

Guided practice



Lecture with demonstration: recursion vs iteration, lecture demos(factorial, fibonacci), more complicated examples

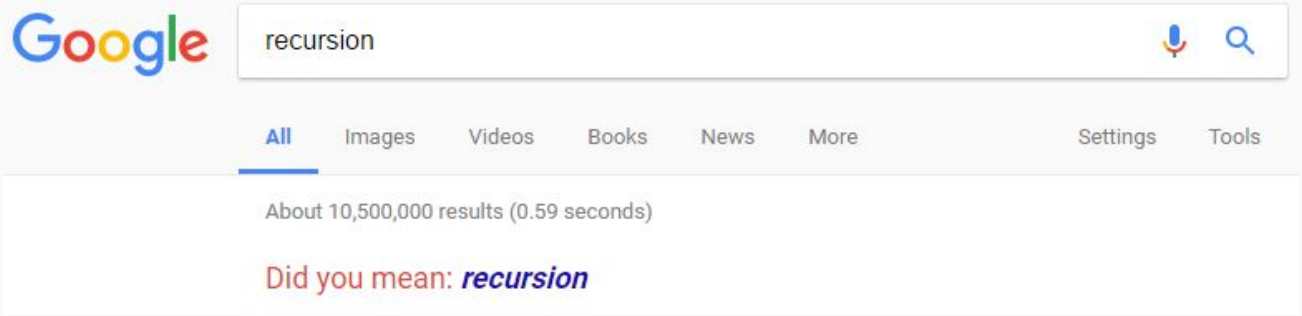
Answers, review, and questions

Recursion (with Python)

Kevin Wang

https://github.com/kwang21093/Recursion-Demos

What is Recursion?



- Process in which a function calls itself
- Has a base case, or termination case, or else the function would go on forever!

Recursion vs. Iteration

- Recursion often leads to cleaner code, but can be trickier to understand/prove correctness
- Recursion take up more memory than iteration, since each function call creates a new *activation record*
- *Tail-recursion*: when the function calling itself is the last action of the method
  - Every tail-recursive method can and should be converted to a *while*-loop

- ```
  def factorial(n):
      if n <= 1:
          return n
      else:
          return n * factorial(n-1)
  ```

- ```
  def iterative_factorial(n):
      result = 1
      for i in range(2,n+1):
          result *= i
      return result
  ```

The Call Stack

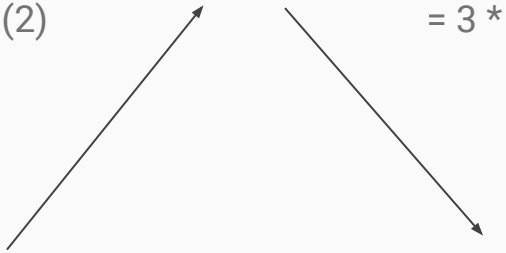
n = 1; factorial(1) = 1

n = 2; factorial(2) = 2 * factorial(1)

n = 3; factorial(3) = 3 * factorial(2)

n = 4; factorial(4) = 4 * factorial(3)

n = 5; factorial(5) = 5 * factorial(4)



= 2 * (1)

= 3 * (2 * (1))

= 4 * (3 * (2 * (1)))

= 5 * (4 * (3 * (2 * (1))))

Try These!

- Write a recursive function that returns the number of digits a positive integer has.
  - Hint: Dividing by 10 removes the rightmost digit
- Write a recursive function that returns the number of steps it takes for the *hailstone sequence* to converge to 1 from a given starting point.
  - A *hailstone sequence*: start at any positive integer **n**, if **n** is even, divide it by 2 and get **n**/2; else triple it and add one to get 3**n**+1; then repeat with the new number
  - hailstone(4) returns 8 because the sequence starting at 40, {40, 20, 10, 5, 16, 8, 4, 2, 1}, converges in 8 steps