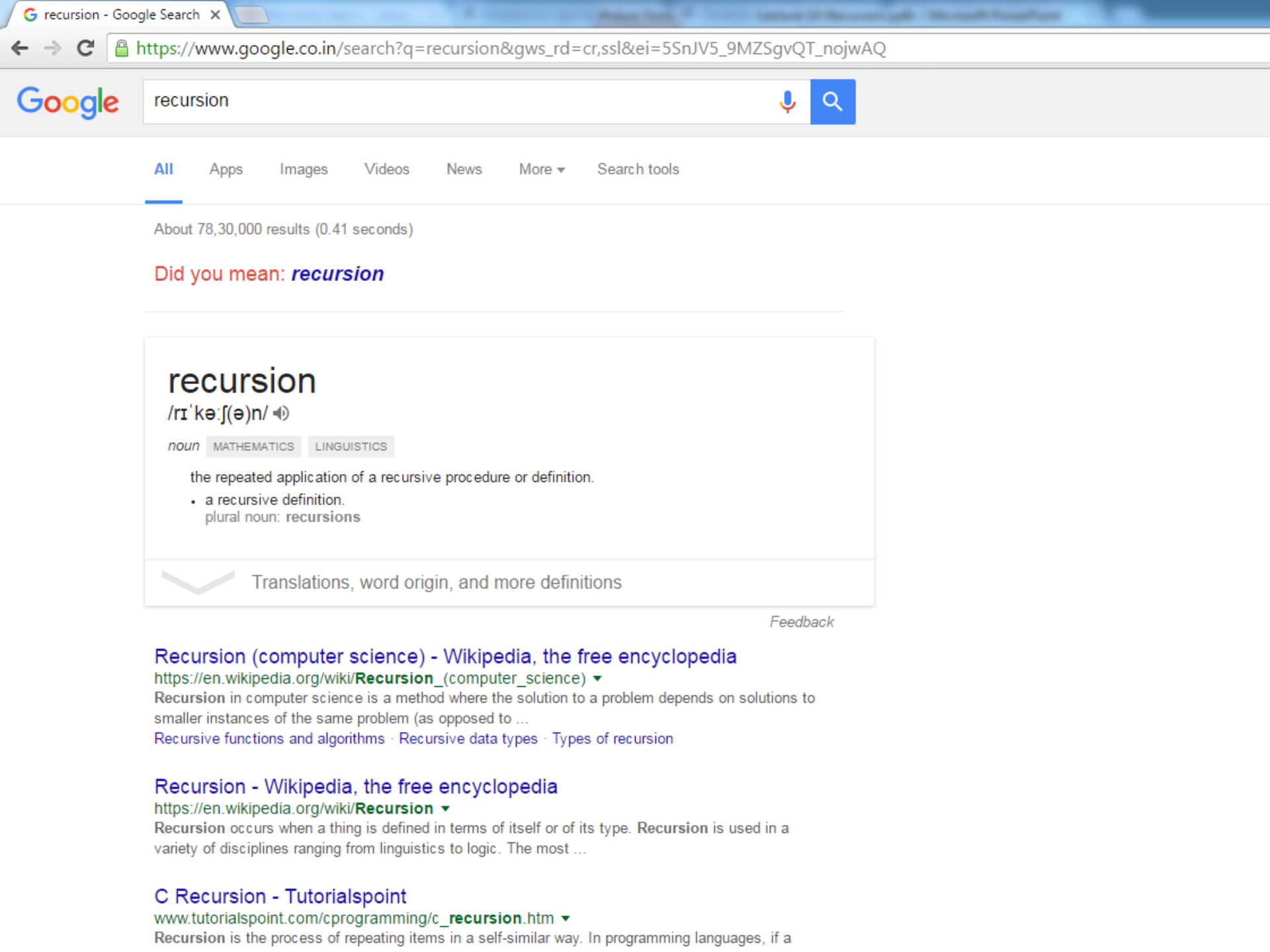

recursion - Google Search
https://www.google.co.in/search?q=recursion&gws_rd=cr,ssl&ei=5SnJV5_9MZSgvQT_nojwAQ
Google
recursion
All
Apps
Images
Videos
News
More
Search tools
About 78,30,000 results (0.41 seconds)
Did you mean: recursion
recursion
/rɪˈkəːʃ(ə)n/
noun
MATHEMATICS
LINGUISTICS
the repeated application of a recursive procedure or definition.
a recursive definition.
plural noun: recursions
Translations, word origin, and more definitions
Feedback
Recursion (computer science) - Wikipedia, the free encyclopedia
https://en.wikipedia.org/wiki/Recursion_(computer_science)
Recursion in computer science is a method where the solution to a problem depends on solutions to smaller instances of the same problem (as opposed to ...
Recursive functions and algorithms · Recursive data types · Types of recursion
Recursion - Wikipedia, the free encyclopedia
https://en.wikipedia.org/wiki/Recursion
Recursion occurs when a thing is defined in terms of itself or of its type. Recursion is used in a variety of disciplines ranging from linguistics to logic. The most ...
C Recursion - Tutorialspoint
www.tutorialspoint.com/cprogramming/c_recursion.htm
Recursion is the process of repeating items in a self-similar way. In programming languages, if a

CBCT: CS 535

Introduction to *Scientific Computing*

Lecture 12

Zubin Bhuyan

Department of CSE,
Tezpur University
http://tezu.ernet.in/~zubin

Outline

- What is recursion?
- Simple recursion: an example
- Demo.

Factorial

- n! = n.(n-1).(n-2)…3.2.1

Factorial

- $n! = n.(n-1).(n-2)...3.2.1$
  - $5! = 5 \times 4 \times 3 \times 2 \times 1$

Factorial

- $n! = n.(n-1).(n-2)\ldots3.2.1$

$(n-1)!$

  - $5! = 5 \times 4 \times 3 \times 2 \times 1$

- $n! = n.(n-1)!$

Factorial

- $n! = n.(n-1).(n-2)\ldots3.2.1$

  $(n-1)!$

  - $5! = 5 \times 4 \times 3 \times 2 \times 1$

- $$n! = \begin{cases} n.(n-1)! & \text{if } n > 0 \\ 1 & \text{if } n = 0 \end{cases}$$

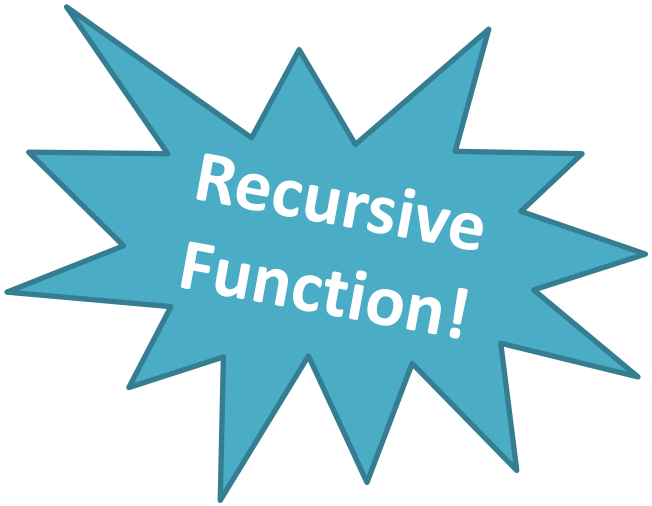
Factorial

- $n! = n.(n-1).(n-2)\ldots3.2.1$

  $(n-1)!$

  - $5! = 5 \times 4 \times 3 \times 2 \times 1$

- $$n! = \begin{cases} n.(n-1)! & \text{if } n > 0 \\ 1 & \text{if } n = 0 \end{cases}$$



Factorial

- $n! = \begin{cases} n.(n-1)! & \text{if } n > 0 \\ 1 & \text{if } n = 0 \end{cases}$

```
def computeFactorial(n):
if(n == 0):
        return 1
    return n * computeFactorial(n-1)
```

Factorial

- $n! = \begin{cases} n.(n-1)! & \text{if } n > 0 \\ 1 & \text{if } n = 0 \end{cases}$

	Return	State
F(4)	4× F(3)	P

```
def computeFactorial(n):
if(n == 0):
        return 1
    return n * computeFactorial(n-1)
```

Recursive call

Factorial

- $n! = \begin{cases} n.(n-1)! & \text{if } n > 0 \\ 1 & \text{if } n = 0 \end{cases}$

	Return	State
F(4)	4× F(3)	P
F(3)	3× F(2)	P

```
def computeFactorial(n):
if(n == 0):
        return 1
    return n * computeFactorial(n-1)
```

Recursive call

Factorial

- $n! = \begin{cases} n.(n-1)! & \text{if } n > 0 \\ 1 & \text{if } n = 0 \end{cases}$

	Return	State
F(4)	4× F(3)	P
F(3)	3× F(2)	P
F(2)	2× F(1)	P

```
def computeFactorial(n):
if(n == 0):
        return 1
    return n * computeFactorial(n-1)
```

Recursive call

Factorial

- $n! = \begin{cases} n.(n-1)! & \text{if } n > 0 \\ 1 & \text{if } n = 0 \end{cases}$

	Return	State
F(4)	4× F(3)	P
F(3)	3× F(2)	P
F(2)	2× F(1)	P
F(1)	1 × F(0)	P

```
def computeFactorial(n):
if(n == 0):
        return 1
    return n * computeFactorial(n-1)
```

Recursive call

Factorial

- $n! = \begin{cases} n.(n-1)! & \text{if } n > 0 \\ 1 & \text{if } n = 0 \end{cases}$

	Return	State
F(4)	4× F(3)	P
F(3)	3× F(2)	P
F(2)	2× F(1)	P
F(1)	1 × F(0)	P
F(0)	1	

```
def computeFactorial(n):
if(n == 0):
        return 1
    return n * computeFactorial(n-1)
```

Recursive call

Factorial

- $n! = \begin{cases} n.(n-1)! & \text{if } n > 0 \\ 1 & \text{if } n = 0 \end{cases}$

	Return	State
F(4)	4× F(3)	P
F(3)	3× F(2)	P
F(2)	2× F(1)	P
F(1)	1 × F(0)	R
F(0)	1	

```
def computeFactorial(n):
if(n == 0):
        return 1
    return n * computeFactorial(n-1)
```

Recursive call

Factorial

- $n! = \begin{cases} n.(n-1)! & \text{if } n > 0 \\ 1 & \text{if } n = 0 \end{cases}$

	Return	State
F(4)	4× F(3)	P
F(3)	3× F(2)	P
F(2)	2× F(1) = 2	R
F(1)	1	
F(0)		

```
def computeFactorial(n):
if(n == 0):
        return 1
    return n * computeFactorial(n-1)
```

Recursive call

Factorial

- $n! = \begin{cases} n.(n-1)! & \text{if } n > 0 \\ 1 & \text{if } n = 0 \end{cases}$

	Return	State
F(4)	4× F(3)	P
F(3)	3× F(2) = 6	R
F(2)	**2**	
F(1)		
F(0)		

```
def computeFactorial(n):
if(n == 0):
        return 1
    return n * computeFactorial(n-1)
```

Recursive call

Factorial

- $n! = \begin{cases} n.(n-1)! & \text{if } n > 0 \\ 1 & \text{if } n = 0 \end{cases}$

	Return	State
F(4)	4× F(3) = 24	R
F(3)	6	
F(2)		
F(1)		
F(0)		

```
def computeFactorial(n):
if(n == 0):
        return 1
    return n * computeFactorial(n-1)
```

Recursive call

- Demo