CSC148-Section:L0301/L0401 Week#6-Friday

Instructed by

AbdulAziz Al-Helali

a.alhelali@mail.utoronto.ca

Office hours: Wednesday 11-1, BA2230.

Slides adapted from Professor Danny Heap course material winter17

Computer Science
UNIVERSITY OF TORONTO

Outline

- Recursion
  - Examples
    - factorial
    - sum_list of nested lists
    - depth of nested lists

Computer Science
UNIVERSITY OF TORONTO

Example 1: factorial

Trace this call: factorial(1)

```
def factorial(num: int)->int:

    if num == 1:
        return 1
    else:
        return num * factorial(num-1)
```

Trace this call: factorial(2)

```
def factorial(num: int)->int:

    if num == 1:
        return 1
    else:
        return num * factorial(num-1)
```

Computer Science
UNIVERSITY OF TORONTO

Trace this call: factorial(3)

```
def factorial(num: int)->int:

    if num == 1:
        return 1
    else:
        return num * factorial(num-1)
```

Computer Science
UNIVERSITY OF TORONTO

Trace this call: factorial(3)

```
3 * factorial(2)
3 * 2 * factorial(1)
3 * 2 * 1
3 * 2
6
```

Computer Science
UNIVERSITY OF TORONTO

Trace this call: factorial(5)

```
5 * factorial(4)
5 * (4 * factorial(3))
5 * (4 * (3 * factorial(2)))
5 * (4 * (3 * (2 * factorial(1))))
5 * (4 * (3 * (2 * 1)))
5 * (4 * (3 * 2))
5 * (4 * 6))
5 * 24
120
```

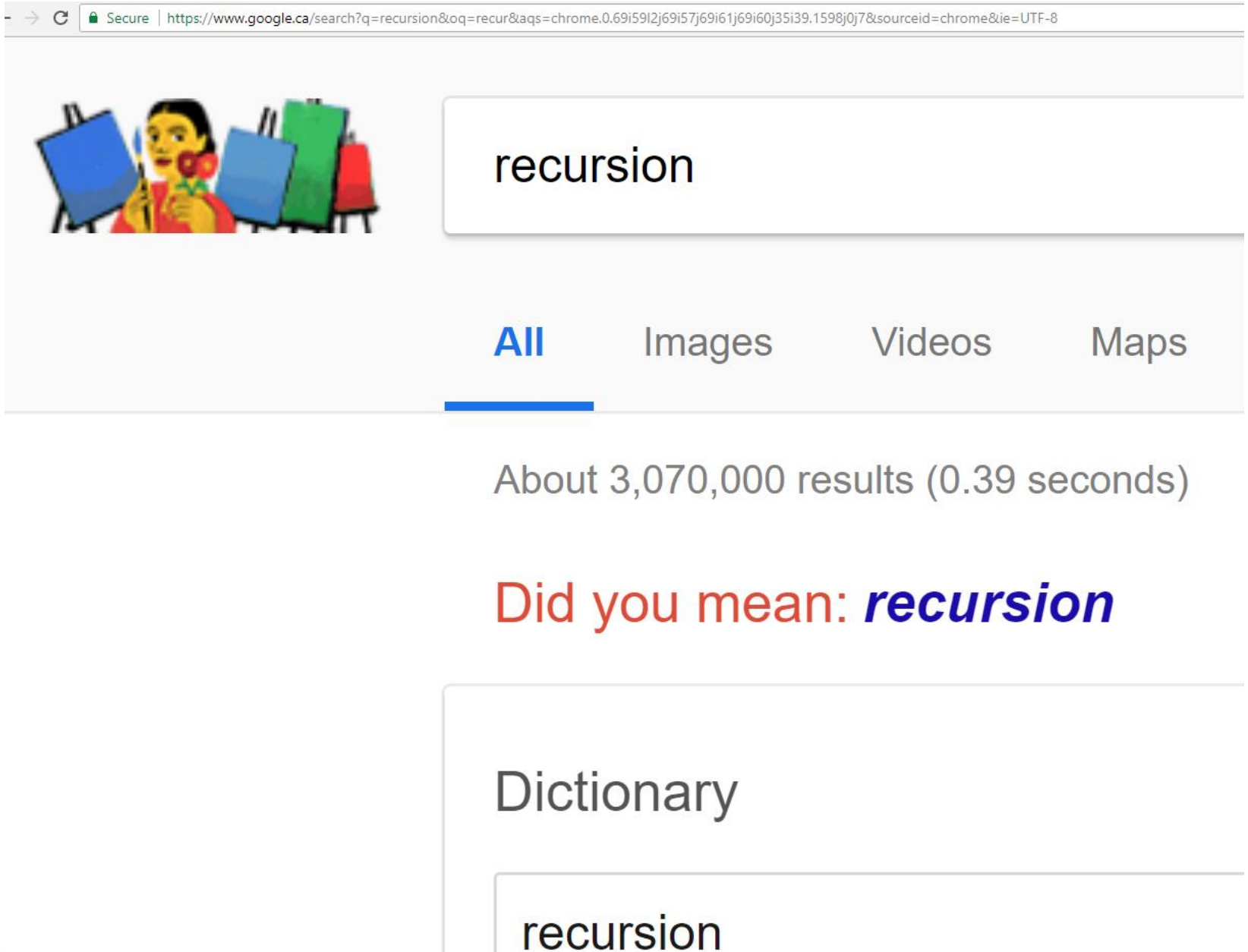
Recursion

- A method calls itself

```
def factorial(num: int)->int:
    if num == 1:
        return 1
    else:
        return num * factorial(num-1)
```

Base case

General case


Secure | https://www.google.ca/search?q=recursion&oq=recur&aqs=chrome.0.69i59l2j69i57j69i61j69i60j35i39.1598j0j7&sourceid=chrome&ie=UTF-8
recursion
All
Images
Videos
Maps
About 3,070,000 results (0.39 seconds)
Did you mean: recursion
Dictionary
recursion

Example 2: summing lists

- L1 = [1, 9, 8, 15]
- sum(L1) = ???
- L2 = [[1, 5], [9, 8], [1, 2, 3, 4]]
- sum([sum(row) for row in L2]) = ??
- L3 = [[1, 5], 9, [8, [1, 2], 3, 4]]
- How can we sum L3?

Build sum_list

- re-use built-in... recursion!
- a function sum_list that adds all the numbers in a nested list **shouldn't ignore** built-in sum
- sum wouldn't work properly on the nested lists, so make a list-comprehension of their sum_lists
- but wait, some of the list elements are numbers, not lists!
- write a denition of sum_list | don't look at next slide yet!

Write sum_list

```
def sum_list(list_: List[int]) -> int:
    """
    Return the sum of all ints in list_.

    >>> sum_list([1, [2, 3], [4, 5, [6, 7], 8]])
    36
    >>> sum([])
    0
    """
```

Computer Science
UNIVERSITY OF TORONTO

sum_list(L)

```
def sum_list(list_: List[int]) -> int:
    if isinstance(list_, list):
        return sum([sum_list(x) for x in list_])
    else:
        return list_
```

tracing recursion

- To understand recursion, **trace** from **simple** to **complex**:
  - trace **sum_list(27)**
  - trace **sum_list([1, 2, 3]).** Remember how the built-in sum works...
  - trace **sum_list([1, [2, 3], 4, [5, 6]]).** Immediately **replace calls you've already traced** (or traced something equivalent) **by their value**
  - trace **sum_list([1, [2, [3 ,4], 5], 6 [7, 8]])**. Immediately replace calls you've already traced by their value.

Computer Science
UNIVERSITY OF TORONTO

sum_list(L)

```
def sum_list(list_: List[int]) -> int:
    if isinstance(list_, list):
        return sum([sum_list(x) for x in list_])
    else:
        return list_
```

- What helper methods does this function call?

sum_list(L)

```
def sum_list(list_: List[int]) -> int:
    if isinstance(list_, list):
        return sum([sum_list(x) for x in list_])
    else:
        return list_
```

- What helper methods does this function call?

sum(...), isinstance(...)

sum_list(...)

sum_list(L)

```
def sum_list(list_: List[int]) -> int:
    if isinstance(list_, list):
        return sum([sum_list(x) for x in list_])
    else:
        return list_
```

- Trace this call: sum_list(27)

sum_list(L)

```
def sum_list(list_: List[int]) -> int:
    if isinstance(list_, list):
        return sum([sum_list(x) for x in list_])
    else:
        return list_
```

- Complete the following trace of this call: sum_list([4, 1, 8])

```
sum_list([4, 1, 8])  --> sum( [  sum_list(4),  sum_list(1),  sum_list(8)  ] )
                     -->| sum( [
                     -->
```

sum_list(L)

```
def sum_list(list_: List[int]) -> int:
    if isinstance(list_, list):
        return sum([sum_list(x) for x in list_])
    else:
        return list_
```

- Complete the following trace of this call: sum_list([4, 1, 8])

```
sum_list([4, 1, 8]) --> sum( [  sum_list(4),  sum_list(1),  sum_list(8)  ] )
                    --> sum( [   4,  1,  8] )
                    -->  13
```

sum_list(L)

```
def sum_list(list_: List[int]) -> int:
    if isinstance(list_, list):
        return sum([sum_list(x) for x in list_])
    else:
        return list_
```

4. Trace this call: `sum_list([4])`

5. Trace this call: `sum_list([])`

UNIVERSITY OF TORONTO

sum_list(L)

```
def sum_list(list_: List[int]) -> int:
    if isinstance(list_, list):
        return sum([sum_list(x) for x in list_])
    else:
        return list_
```

--> 13

4. Trace this call: `sum_list([4])` → sum([sum_list(4)])
→ sum([4])
→ 4

5. Trace this call: `sum_list([])` → sum([])
→ 0

```
def sum_list(list_: List[int]) -> int:
    if isinstance(list_, list):
        return sum([sum_list(x) for x in list_])
    else:
        return list_
```

6. Trace this call: `sum_list([4, [1, 2, 3], 8])`

```
def sum_list(list_: List[int]) -> int:
    if isinstance(list_, list):
        return sum([sum_list(x) for x in list_])
    else:
        return list_
```

lists of depth 2:

6. Trace this call: `sum_list([4, [1, 2, 3], 8])`

→ sum([sum_list(4), sum_list([1,2,3]), sum_list(8)])

→ sum([4, 6, 8])

→ 18

7. Trace this call: `sum_list([[1, 2, 3], [4, 5], 8])`

8. Trace this call: `sum_list([1, [2, 2], [2, [3, 3, 3], 2]])`

Computer Science
UNIVERSITY OF TORONTO

7. Trace this call: `sum_list([[1, 2, 3], [4, 5], 8])`

→ sum([sum_list([1,2,3]), sum_list([4,5]), sum_list(8)])

→ sum([6, 9, 8]) → ~~15~~ 23

lists of depth 2: sums all integers

8. Trace this call: `sum_list([1, [2, 2], [2, [3, 3, 3], 2]])`

depth 2

→ sum([sum_list(1), sum_list([2,2]), sum_list([2, [3,3,3], 2])])

→ sum([1, 4, 13]) → 18

lists of depth 3 sums all integers

9. Trace this call: `sum_list([1, [2, 2], [2, [3, [4, 4], 3, 3], 2]])`

10. Are you a believer yet?

depth 3

9. Trace this call: `sum_list([1, [2, 2], [2, [3, [4, 4], 3, 3], 2]])`

→ sum([sum_list(1), sum_list([2,2]), sum_list([2, [3, [4,4], 3, 3], 2])])

→ sum([1, 4, 21]) → 26

lists of depth 4: sums all ints

10. Are you a believer yet?

lets try depth 37...

Example 3: depth of a list

Define the `depth` of `list_` as 1 plus the maximum depth of `list_`'s elements if `list_` is a list, otherwise 0.

- the definition is almost exactly the Python code you write!
- start by writing `return` and pythonese for the definition:

```
if isinstance(list_, list):
    return 1 + max([depth(x) for x in list_])
else: # list_ is not a list
    return 0
# find the bug! (then fix it...)
```

- deal with the special case of a non-list

Computer S
UNIVERSITY OF TORONTO

Trace

Trace in increasing complexity; at each step fill in values for recursive calls that have (basically) **already been traced**

- Trace `depth([])`
- Trace `depth(17)`
- Trace `depth([3, 17, 1])`
- Trace `depth([5, [3, 17, 1], [2, 4], 6])`
- Trace
  `depth([14, 7, [5, [3, 17, 1], [2, 4], 6], 9])`

Computer Science
UNIVERSITY OF TORONTO