CSC148H Week 4

January 28, 2015

Announcements

- Assignment 1 - due next week, February 4 at 22:00
  - What is an anagram?
  - Can you redo before an undo?
  - Note: All assignments will be checked for plagiarism.
- Quiz 1 - this Friday during lecture. Please bring a pen!
  - 15 minutes
  - Content covered: week 1 up to and including week 3
  - Short answer questions (correction, no multiple choice)

Summing a Nested List

Let's write a recursive function to sum all items in a list with *n* nested lists. We're going solve this without using `flatten`.

- What is the base case?
- What is the recursive structure?

```
def sumlist(lst):
  '''(list of int) -> int
  lst is arbitrarily nested.
  Return the sum of lst.

  >>> sumlist([1, 2, [3, [4]], 5])
  15
  '''
```

Summming a Nested List, Base Case

```
def sumlist(lst):
    '''(list of int) -> int
    lst is arbitrarily nested.
    Return the sum of lst.

    >>> sumlist([1,2,[3,[4]],5])
    15
    '''

    total = 0

    for i in lst:
        total += i
    return total
```

Summing a Nested List...

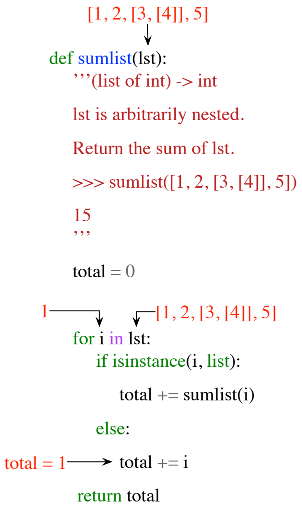
```
def sumlist(lst):
  '''(list of int) -> int
  lst is arbitrarily nested.
  Return the sum of lst.

  >>> sumlist([1,2,[3,[4]],5])
  15
  '''

  total = 0

  for i in lst:
    if isinstance(i, list):
      total += sumlist(i)
    else:
      total += i

  return total
```

Summing a Nested List...

Evaluate when lst = [1, 2, [3, [4]], 5]

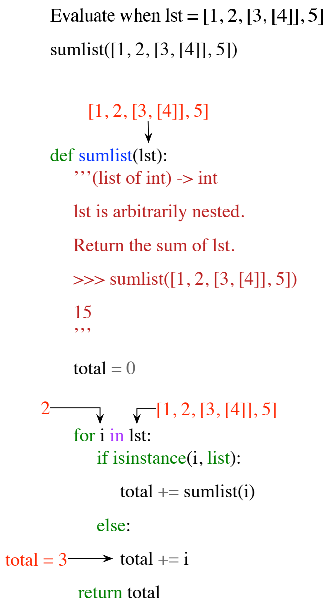
sumlist([1, 2, [3, [4]], 5])

```
def sumlist(lst):
    ’’’(list of int) -> int

    lst is arbitrarily nested.

    Return the sum of lst.

    >>> sumlist([1, 2, [3, [4]], 5])

    15
    ’’’

    total = 0


    for i in lst:
        if isinstance(i, list):

            total += sumlist(i)

        else:

            total += i

     return total
```



```
         [1, 2, [3, [4]], 5]
def sumlist(lst):
    ’’’(list of int) -> int

    lst is arbitrarily nested.

    Return the sum of lst.

    >>> sumlist([1, 2, [3, [4]], 5])

    15
    ’’’

    total = 0

1                 [1, 2, [3, [4]], 5]
    for i in lst:
        if isinstance(i, list):

            total += sumlist(i)

        else:

total = 1   total += i

     return total
```

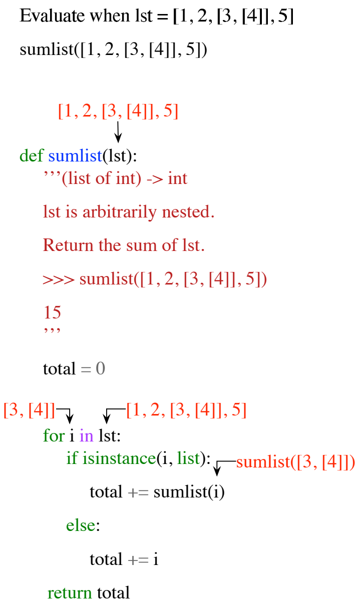
Summing a Nested List...



Evaluate when lst = [1, 2, [3, [4]], 5]

sumlist([1, 2, [3, [4]], 5])

```
          [1, 2, [3, [4]], 5]

def sumlist(lst):
    '''(list of int) -> int

    lst is arbitrarily nested.

    Return the sum of lst.

    >>> sumlist([1, 2, [3, [4]], 5])

    15
    '''

    total = 0

2                  [1, 2, [3, [4]], 5]
    for i in lst:
        if isinstance(i, list):

            total += sumlist(i)

        else:

total = 3   total += i

    return total
```



Evaluate when lst = [1, 2, [3, [4]], 5]

sumlist([1, 2, [3, [4]], 5])

```
          [1, 2, [3, [4]], 5]

def sumlist(lst):
    '''(list of int) -> int

    lst is arbitrarily nested.

    Return the sum of lst.

    >>> sumlist([1, 2, [3, [4]], 5])

    15
    '''

    total = 0

[3, [4]]           [1, 2, [3, [4]], 5]
    for i in lst:
        if isinstance(i, list):   sumlist([3, [4]])

            total += sumlist(i)

        else:

            total += i

    return total
```

Summing a Nested List...

Evaluate when lst = [1, 2, [3, [4]], 5]

sumlist([1, 2, [3, [4]], 5])

```
        [3, [4]]
def sumlist(lst):
    ’’’(list of int) -> int

    lst is arbitrarily nested.

    Return the sum of lst.

    >>> sumlist([1, 2, [3, [4]], 5])

    15
    ’’’

    total = 0

3               [3, [4]
    for i in lst:
        if isinstance(i, list):

            total += sumlist(i)

        else:

            total += i

    return total    total = 3
```

Evaluate when lst = [1, 2, [3, [4]], 5]

sumlist([1, 2, [3, [4]], 5])

```
        [3, [4]]
def sumlist(lst):
    ’’’(list of int) -> int

    lst is arbitrarily nested.

    Return the sum of lst.

    >>> sumlist([1, 2, [3, [4]], 5])

    15
    ’’’

    total = 0

[4]             [3, [4]
    for i in lst:
        if isinstance(i, list):   sumlist([4])

            total += sumlist(i)

        else:

            total += i

    return total
```

Summing a Nested List...

Evaluate when lst = [1, 2, [3, [4]], 5]

sumlist([1, 2, [3, [4]], 5])

```
                [4]
def sumlist(lst):
    '''(list of int) -> int

    lst is arbitrarily nested.

    Return the sum of lst.

    >>> sumlist([1, 2, [3, [4]], 5])

    15
    '''

    total = 0

4               [4]
    for i in lst:
        if isinstance(i, list):

            total += sumlist(i)

        else:

total = 4   total += i

     return total   total = 4
```

Evaluate when lst = [1, 2, [3, [4]], 5]

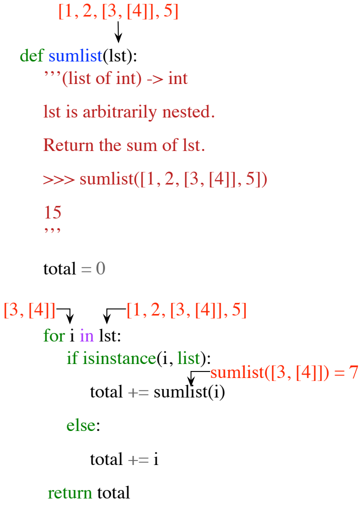
sumlist([1, 2, [3, [4]], 5])

```
                [3, [4]]
def sumlist(lst):
    '''(list of int) -> int

    lst is arbitrarily nested.

    Return the sum of lst.

    >>> sumlist([1, 2, [3, [4]], 5])

    15
    '''

    total = 0

[4]             [3, [4]
    for i in lst:
        if isinstance(i, list):   sumlist([4]) = 4

            total += sumlist(i)

        else:

            total += i

     return total
```

Summing a Nested List...

Evaluate when lst = [1, 2, [3, [4]], 5]

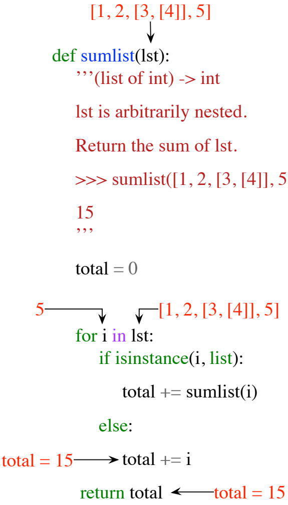
sumlist([1, 2, [3, [4]], 5])



```
def sumlist(lst):
    '''(list of int) -> int

    lst is arbitrarily nested.

    Return the sum of lst.

    >>> sumlist([1, 2, [3, [4]], 5])

    15
    '''

    total = 0

    for i in lst:
        if isinstance(i, list):

            total += sumlist(i)

        else:

            total += i

    return total
```

Evaluate when lst = [1, 2, [3, [4]], 5]

sumlist([1, 2, [3, [4]], 5])



```
def sumlist(lst):
    '''(list of int) -> int

    lst is arbitrarily nested.

    Return the sum of lst.

    >>> sumlist([1, 2, [3, [4]], 5])

    15
    '''

    total = 0

    for i in lst:
        if isinstance(i, list):

            total += sumlist(i)

        else:

            total += i

    return total
```

Summing a Nested List...Let's Use Exceptions

```python
def sumlist(lst):
  '''(list of int) -> int
  lst is arbitrarily nested.
  Return the sum of lst.

  >>> sumlist([1,2,[3,[4]],5])
  15
  '''

  total = 0

  for i in lst:
    try:
      total += i
    except TypeError:
      total += sumlist(i)
  return total
```

Removing 3s

Let's write a recursive function to return a new list with all elements of a list except the 3s.

- What is the base case?
- What is the recursive structure?

```
def remove_three(lst):
  '''(list of int) -> list of int

  Return a new list with all elements of lst except the 3s.
  lst has no nesting.

  >>> remove_three([1, 2, 3, 3, 4])
  [1, 2, 4]
  '''
```

Removing 3s, Base Case

```
def remove_three(lst):
  '''(list of int) -> list of int

  Return a new list with all elements of lst except the 3s.
  lst has no nesting.

  >>> remove_three([1, 2, 3, 3, 4])
  [1, 2, 4]
  '''
  if len(lst) == 0:
    return []
```

Removing 3s...

```
def remove_three(lst):
  '''(list of int) -> list of int

  Return a new list with all elements of lst except the 3s.
  lst has no nesting.

  >>> remove_three([1, 2, 3, 3, 4])
  [1, 2, 4]
  '''
  if len(lst) == 0:
    return []

  if lst[0] == 3:
    return remove_three(lst[1:])
  else:
    return [lst[0]] + remove_three(lst[1:])
```

Removing 3s In A Nested List

Let's revisit our previous example, except this time our list can contain nested lists. The goal is to write a recursive function to return a new list with all elements of a list except the 3s.

- What is the base case?
- What is the recursive structure?

```
def remove_three_nested(lst):
  '''(list of int) -> list of int

  Return a new list with all elements of lst except the 3s.
  lst may have nesting to arbitrary depth.

  >>> remove_three_nested([1, [2, 3], 3, 4])
  [1, [2], 4]
  '''
```

Removing 3s In A Nested List, Base Case

```
def remove_three_nested(lst):
  '''(list of int) -> list of int

  Return a new list with all elements of lst except the 3s.
  lst may have nesting to arbitrary depth.

  >>> remove_three_nested([1, [2, 3], 3, 4])
  [1, [2], 4]
  '''
  no_three = []
  for element in lst:
    if isinstance(element, int):
      if element != 3:
        no_three.append(element)
  return no_three
```

Removing 3s In A Nested List...

```python
def remove_three_nested(lst):
  '''(list of int) -> list of int

  Return a new list with all elements of lst except the 3s.
  lst may have nesting to arbitrary depth.

  >>> remove_three_nested([1, [2, 3], 3, 4])
  [1, [2], 4]
  '''
  no_three = []
  for element in lst:
    if isinstance(element, int):
      if element != 3:
        no_three.append(element)
    else:
      no_three.append(remove_three_nested(element))
  return no_three
```

Removing 3s In A Nested List...

Evaluate when lst is [1, [2, 3], 3, 4]

remove_three_nested([1, [2, 3], 3, 4])

```
def remove_three_nested(lst):
    '''(list of int) -> list of int
     Return a new list with all elements of lst except the 3s.
    lst may have nesting to arbitrary depth.
    >>> remove_three_nested([1, [2, 3], 3, 4])
    [1, [2], 4]
    '''
    no_three = []

    for element in lst:
        if isinstance(element, int):
            if element != 3:
                no_three.append(element)
        else:
            no_three.append(remove_three_nested(element))
    return no_three
```

[1, [2, 3], 3, 4]

```
def remove_three_nested(lst):
    '''(list of int) -> list of int
     Return a new list with all elements of lst except the 3s.
    lst may have nesting to arbitrary depth.
    >>> remove_three_nested([1, [2, 3], 3, 4])
    [1, [2], 4]
    '''
    no_three = []
```

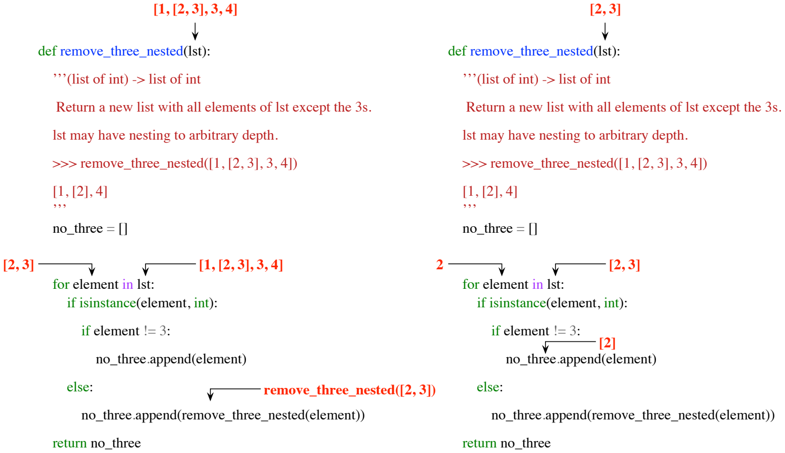
1 **[1, [2, 3], 3, 4]**

```
    for element in lst:
        if isinstance(element, int):
            if element != 3:
```

[1]

```
                no_three.append(element)
        else:
            no_three.append(remove_three_nested(element))
    return no_three
```

Removing 3s In A Nested List...



[1, [2, 3], 3, 4]

```
def remove_three_nested(lst):
    '''(list of int) -> list of int

     Return a new list with all elements of lst except the 3s.

    lst may have nesting to arbitrary depth.

    >>> remove_three_nested([1, [2, 3], 3, 4])

    [1, [2], 4]
    '''

    no_three = []

    for element in lst:
        if isinstance(element, int):

            if element != 3:

                no_three.append(element)

        else:

            no_three.append(remove_three_nested(element))

    return no_three
```

[2, 3]

[1, [2, 3], 3, 4]

remove_three_nested([2, 3])

[2, 3]

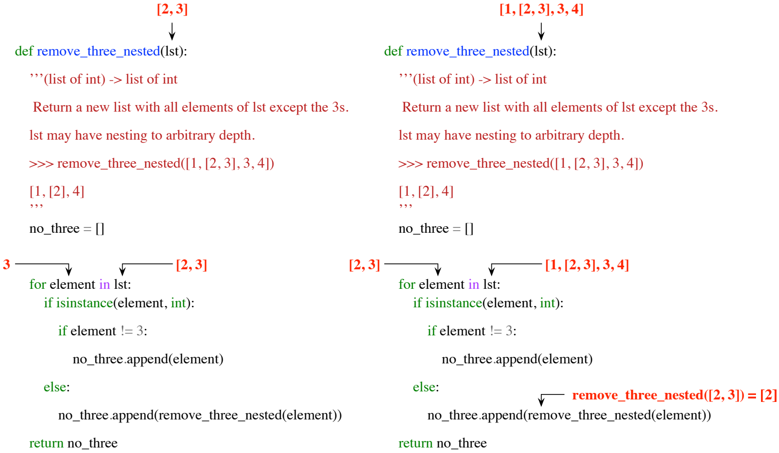
```
def remove_three_nested(lst):
    '''(list of int) -> list of int

     Return a new list with all elements of lst except the 3s.

    lst may have nesting to arbitrary depth.

    >>> remove_three_nested([1, [2, 3], 3, 4])

    [1, [2], 4]
    '''

    no_three = []

    for element in lst:
        if isinstance(element, int):

            if element != 3:

                no_three.append(element)

        else:

            no_three.append(remove_three_nested(element))

    return no_three
```

2

[2, 3]

[2]

Removing 3s In A Nested List...



[2, 3]

```
def remove_three_nested(lst):
  '''(list of int) -> list of int
   Return a new list with all elements of lst except the 3s.
  lst may have nesting to arbitrary depth.
  >>> remove_three_nested([1, [2, 3], 3, 4])
  [1, [2], 4]
  '''
  no_three = []
```

3 [2, 3]

```
  for element in lst:
    if isinstance(element, int):
      if element != 3:
        no_three.append(element)
    else:
      no_three.append(remove_three_nested(element))
  return no_three
```

[1, [2, 3], 3, 4]

```
def remove_three_nested(lst):
  '''(list of int) -> list of int
   Return a new list with all elements of lst except the 3s.
  lst may have nesting to arbitrary depth.
  >>> remove_three_nested([1, [2, 3], 3, 4])
  [1, [2], 4]
  '''
  no_three = []
```

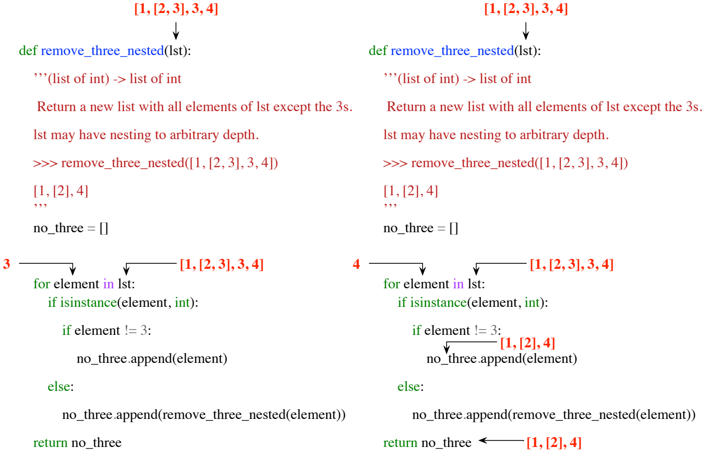
[2, 3] [1, [2, 3], 3, 4]

```
  for element in lst:
    if isinstance(element, int):
      if element != 3:
        no_three.append(element)
    else:
      no_three.append(remove_three_nested(element))
  return no_three
```

remove_three_nested([2, 3]) = [2]

Removing 3s In A Nested List...

remove_three_nested([1, [2, 3], 3, 4]) = [1, [2], 4]



```
                  [1, [2, 3], 3, 4]
def remove_three_nested(lst):
    '''(list of int) -> list of int
     Return a new list with all elements of lst except the 3s.
    lst may have nesting to arbitrary depth.
    >>> remove_three_nested([1, [2, 3], 3, 4])
    [1, [2], 4]
    '''
    no_three = []
3                         [1, [2, 3], 3, 4]
    for element in lst:
        if isinstance(element, int):
            if element != 3:
                no_three.append(element)
        else:
            no_three.append(remove_three_nested(element))
    return no_three
```

```
                  [1, [2, 3], 3, 4]
def remove_three_nested(lst):
    '''(list of int) -> list of int
     Return a new list with all elements of lst except the 3s.
    lst may have nesting to arbitrary depth.
    >>> remove_three_nested([1, [2, 3], 3, 4])
    [1, [2], 4]
    '''
    no_three = []
4                         [1, [2, 3], 3, 4]
    for element in lst:
        if isinstance(element, int):
            if element != 3:
                          [1, [2], 4]
                no_three.append(element)
        else:
            no_three.append(remove_three_nested(element))
    return no_three   [1, [2], 4]
```