Decomposition (Order of Growth & Linked List Practice)

Announcements

Fast Doubling

Double a List and a Linked List (Last Lecture)

```
double(      cycle(      5, 100000), 3): 302ms
double_link(cycle_link(5, 100000), 3):  15ms
```

```
def double(s, v):
    """Insert another v after each v.

    >>> s = [2, 7, 1, 8, 2, 8]
    >>> double(s, 8)
    >>> s
    [2, 7, 1, 8, 8, 2, 8, 8]
    """
    i = 0
    while i < len(s):
        if s[i] == v:
            s.insert(i+1, v)
            i += 2
        else:
            i += 1
```

Quadratic Growth

Shift over everything after i+1

```
def double_link(s, v):
    """Insert another v after each v.

    >>> end = Link(1, Link(8, Link(2, Link(8))))
    >>> t = Link(2, Link(7,end))
    >>> double_link(t, 8)
    >>> print(t)
    (2 7 1 8 8 2 8 8)
    """
    while s is not Link.empty:
        if s.first == v:
            s.rest = Link(v, s.rest)
            s = s.rest.rest
        else:
            s = s.rest
```

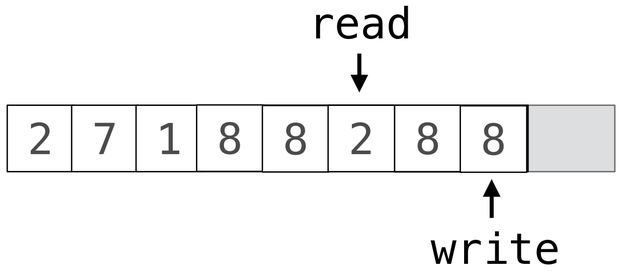
Linear Growth

Make 1 Link

Double a List with Linear Growth

Could you insert another v after each v in a Python list s with linear growth?

```
def double_fast(s, v):
    """Insert another v after each v in s.

    >>> s = [2, 7, 1, 8, 2, 8]
    >>> double_fast(s, 8)
    >>> s
    [2, 7, 1, 8, 8, 2, 8, 8]
    """
    read = len(s) - 1
    vs = s.count(v)
    s.extend([0 for _ in range(vs)]) # Make space
    write = len(s) - 1
    while write > read:
        if s[read] == v:
            s[write] = v
            s[write - 1] = v
            write -= 2
        else:
            s[write] = s[read]
            write -= 1
        read -= 1
```




```
double(      cycle(      5, 100000), 3): 302ms
double_link(cycle_link(5, 100000), 3):  15ms
double_fast(cycle(      5, 100000), 3):   8ms
```

Order of Growth Practice

pollev.com/cs61a

Match each function to its order of growth

Exponential growth. E.g., recursive fib

Incrementing *n* multiplies *time* by a constant

Quadratic growth.

Incrementing *n* increases *time* by *n* times a constant

Linear growth.

Incrementing *n* increases *time* by a constant

Logarithmic growth.

Doubling *n* only increments *time* by a constant

Constant growth. Increasing *n* doesn't affect time

Definition. A *prefix sum* of a sequence of numbers is the sum of the first n elements for some positive length n.

(1 pt) What is the order of growth of the time to run prefix(s) in terms of the length of s? Assume append and + take one step.

```
def prefix(s):
    """Return a list of all prefix
    sums of list s.
    """
    t = 0
    result = []
    for x in s:
        t = t + x
        result.append(t)
    return result
```

pollev.com/cs61a

Match each function to its order of growth

Exponential growth. E.g., recursive fib

Incrementing *n* multiplies *time* by a constant

Quadratic growth.

Incrementing *n* increases *time* by *n* times a constant

Linear growth.

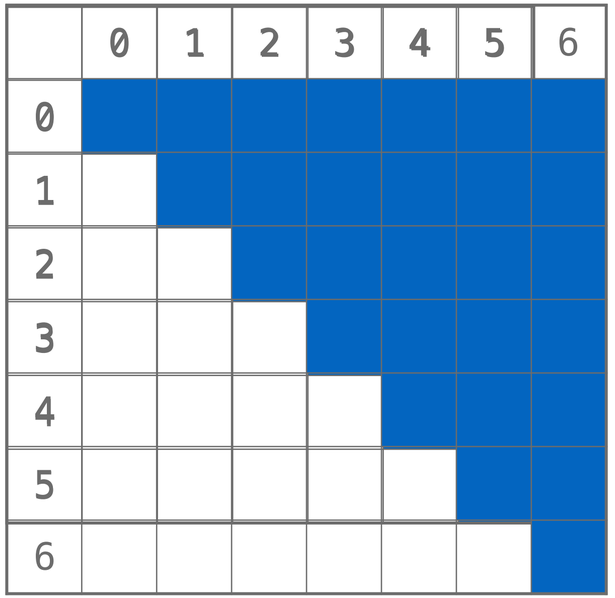
Incrementing *n* increases *time* by a constant

Logarithmic growth.

Doubling *n* only increments *time* by a constant

Constant growth. Increasing *n* doesn't affect time

```
def max_sum(s):
    """Return the largest sum of a contiguous
    subsequence of s.
    >>> max_sum([3, 5, -12, 2, -4, 4, -1, 4, 2, 2])
    11
    """
    largest = 0
    for i in range(len(s)):
        total = 0
        for j in range(i, len(s)):
            total += s[j]
            largest = max(largest, total)
    return largest
```



Match each function to its order of growth

Exponential growth. E.g., recursive fib

Incrementing *n* multiplies *time* by a constant

Quadratic growth.

Incrementing *n* increases *time* by *n* times a constant

Linear growth.

Incrementing *n* increases *time* by a constant

Logarithmic growth.

Doubling *n* only increments *time* by a constant

Constant growth. Increasing *n* doesn't affect time

```
def count_tree(n):
    """Return a count tree with n leaves.
    >>> print(count_tree(10))
    10
      5
        2
          1
          1
        3
          2
            1
            1
          1
      5
        2
          1
          1
        3
          2
            1
            1
          1
    """
    if n == 1:
        return Tree(1)
    left = count_tree(n//2)
    if n % 2 == 0:
        right = left
    else:
        right = Tree(left.label + 1, [left, Tree(1)])
    return Tree(left.label+right.label, [left,right])
```

Definition. A *count tree* is a tree whose labels are counts of the leaves below each node

(1 pt) What is the order of growth of the time to run count_tree(n) in terms of n?

pollev.com/cs61a

pollev.com/cs61a

Match each function to its order of growth

Exponential growth. E.g., recursive fib

Incrementing *n* multiplies *time* by a constant

Quadratic growth.

Incrementing *n* increases *time* by *n* times a constant

Linear growth.

Incrementing *n* increases *time* by a constant

Logarithmic growth.

Doubling *n* only increments *time* by a constant

Constant growth. Increasing *n* doesn't affect time

(1 pt) What is the order of growth of the time to run duplicate(s) in terms of the length of list s?

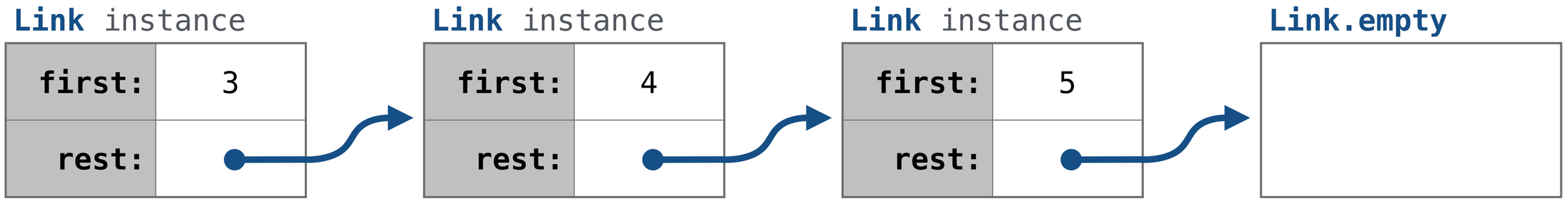
```
def duplicate(s):
    """Return a list containing the list s twice.

    >>> duplicate([2, 5, 8, 11, 14, 17])
    [[2, 5, 8, 11, 14, 17], [2, 5, 8, 11, 14, 17]]
    """
    return [s, s]
```


https://pythontutor.com/cp/composingprograms.html#code=def%20duplicate%28s%29%3A%0A%20%20%20%20return%20%5Bs,%20s%5D%0A%20%20%20%20%0Aresult%20%3D%20duplicate%28%5B2,%205,%208,%2011,%2014,%2017%5D%29&cumulative=true&curInstr=1&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Linked Lists Practice

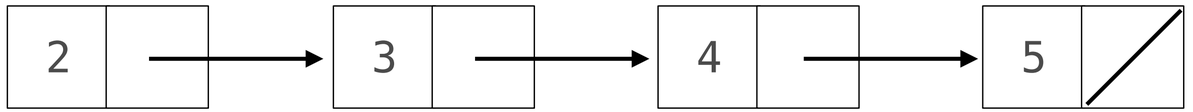
Linked List Notation

```
s = Link(3, Link(4, Link(5)))
```

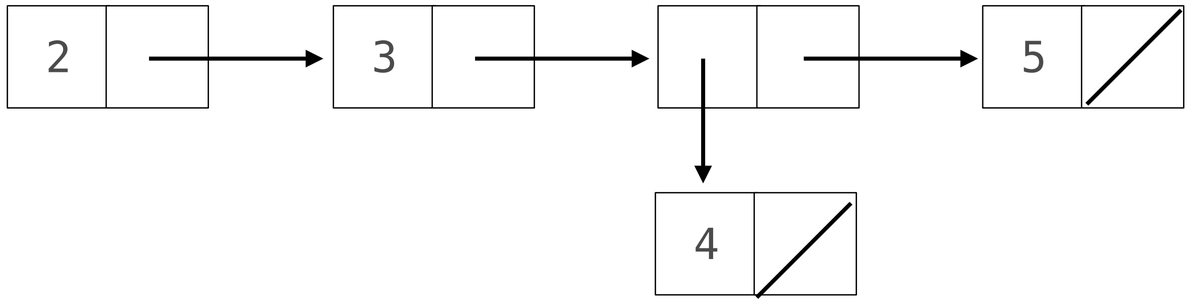


Nested Linked Lists

```
>>> s = Link(2, Link(3, Link(    4     , Link(5))))

>>> t = Link(2, Link(3, Link(  Link(4) , Link(5))))

>>> print(s)
(2 3 4 5)
```

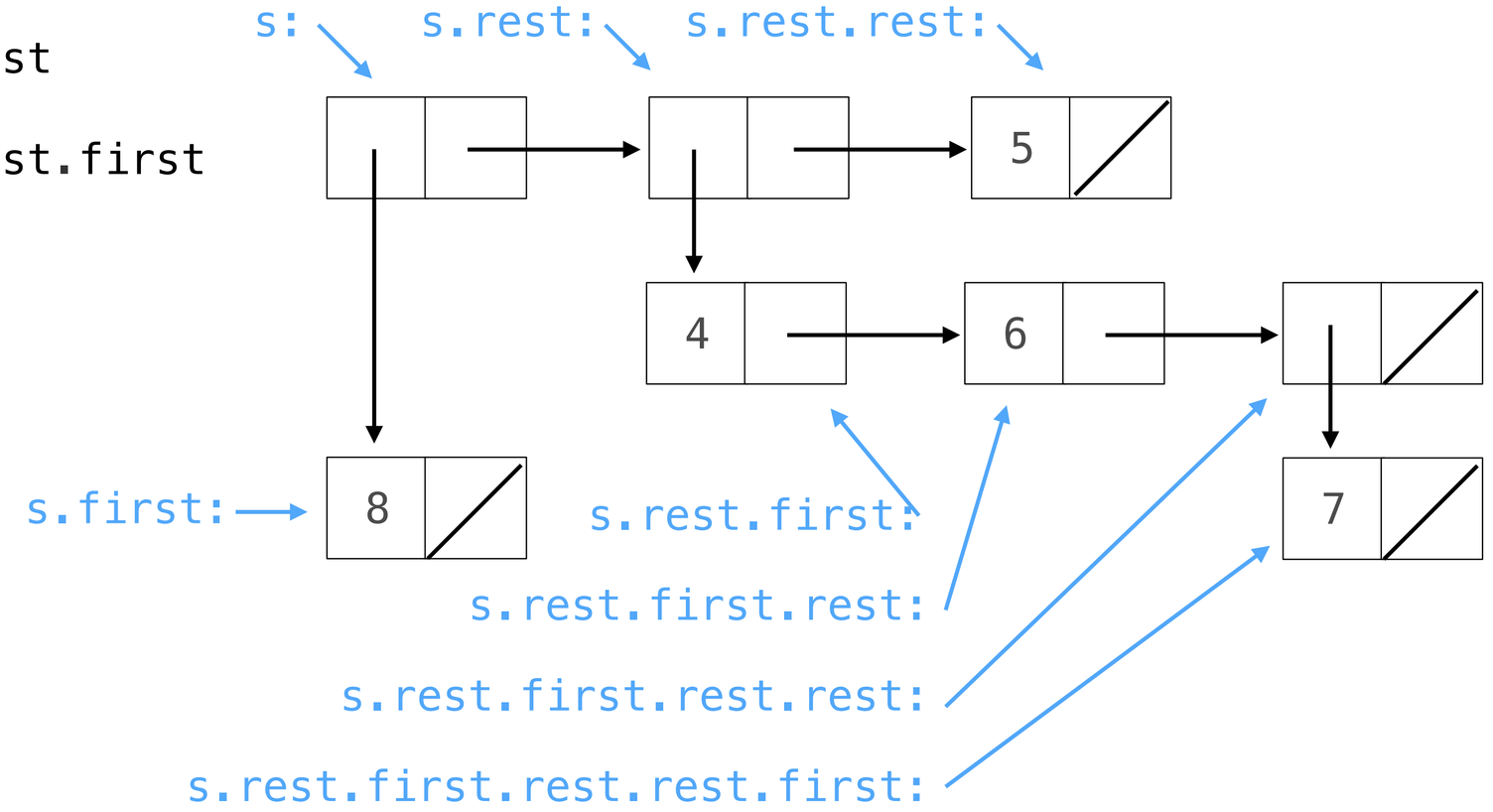



```
>>> print(t)
(2 3 (4) 5)
```



Nested Linked Lists

```
>>> s = Link(Link(8), Link(Link(4, Link(6, Link(Link(7)))), Link(5)))
>>> print(s)
((8) (4 6 (7)) 5)
>>> s.first.first
8
>>> s.rest.first.rest.rest.first
Link(7)
>>> s.rest.first.rest.rest.first.first
7
```



Recursion and Iteration

Many linked list processing functions can be written both iteratively and recursively

Recursive approach:

- What recursive call do you make?
- What does this recursive call do/return?
- How is this result useful in solving the problem?

```
def length(s):
    """The number of elements in s.

    >>> length(Link(3, Link(4, Link(5))))
    3
    """
    if s is Link.empty:
        return 0
    else:
        return 1 + length(s.rest)
```

Iterative approach:

- Describe a process that solves the problem.
- Figure out what additional names you need to carry out this process.
- Implement the process using those names.

```
def length(s):
    """The number of elements in s.

    >>> length(Link(3, Link(4, Link(5))))
    3
    """
    k = 0
    while s is not Link.empty:
        s, k = s.rest, k + 1
    return k
```

Constructing a Linked List

Build the rest of the linked list, then combine it with the first element.


```
s = Link.empty
s = Link(5, s)
s = Link(4, s)
s = Link(3, s)
```

```
def range_link(start, end):
    """Return a Link containing consecutive
    integers from start up to end.

    >>> range_link(3, 6)
    Link(3, Link(4, Link(5)))
    """
    if start >= end:
        return Link.empty
    else:
        return Link(start, range_link(start + 1, end))
```

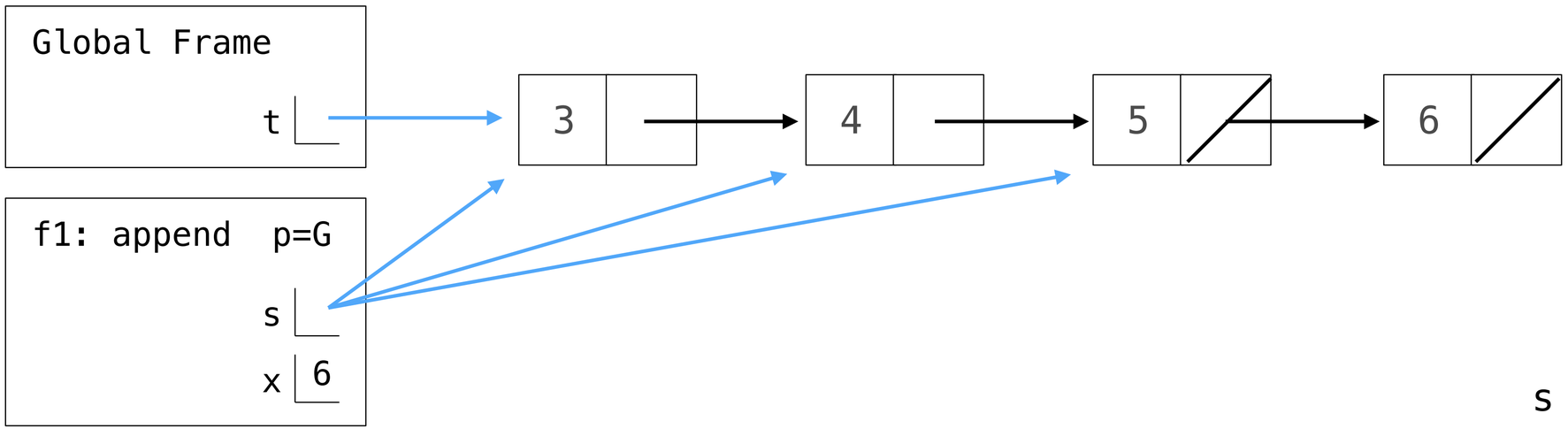
```
def range_link(start, end):
    """Return a Link containing consecutive
    integers from start to end.

    >>> range_link(3, 6)
    Link(3, Link(4, Link(5)))
    """
    s = Link.empty
    k = end - 1
    while k >= start:
        s = Link(k, s)
        k -= 1
    return s
```

Linked List Mutation

To change the contents of a linked list, assign to first and rest attributes

Example: Append x to the end of non-empty s

```
>>> t = Link(3, Link(4, Link(5)))
>>> append(t, 6)
>>> t
Link(3, Link(4, Link(5, Link(6))))
```




```
s = s.rest

s.rest = Link(x)
```

Recursion and Iteration

Many linked list processing functions can be written both iteratively and recursively

Recursive approach:

- What recursive call do you make?
- What does this recursive call do/return?
- How is this result useful in solving the problem?

```
def append(s, x):
    """Append x to the end of non-empty s.
    >>> append(s, 6)  # returns None!
    >>> print(s)
    (3 4 5 6)
    """
    if ___s.rest is not Link.empty___:
        append(___s.rest___, ___x___)
    else:
        ___s.rest = Link(x)___
```

Iterative approach:

- Describe a process that solves the problem.
- Figure out what additional names you need to carry out this process.
- Implement the process using those names.

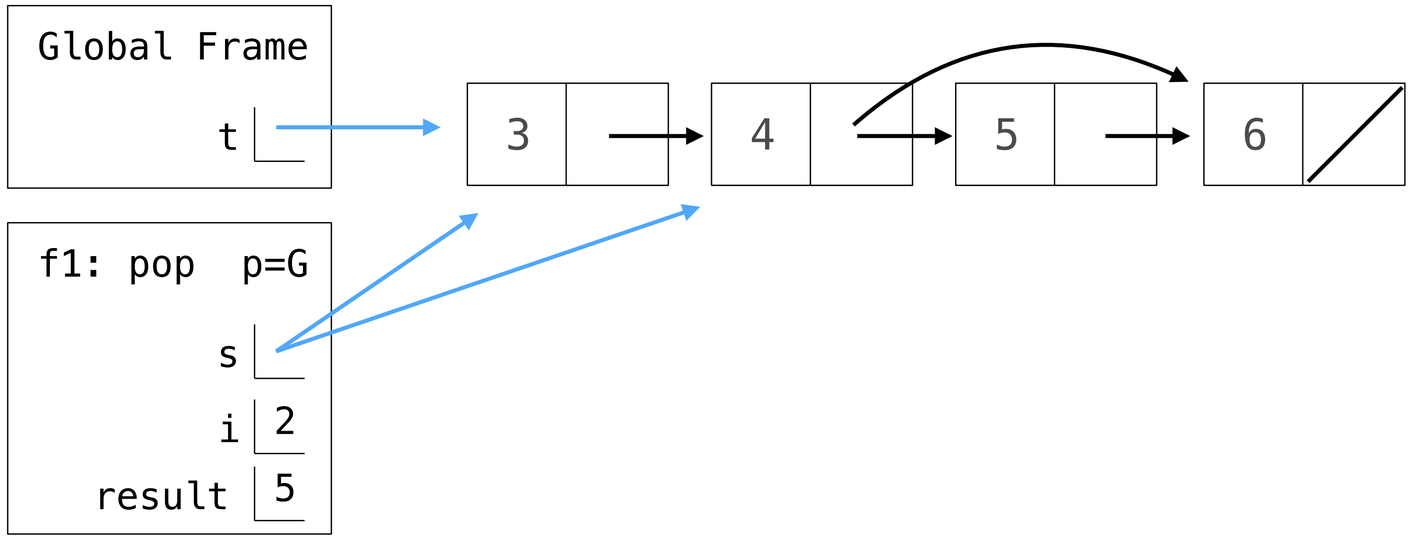
```
def append(s, x):
    """Append x to the end of non-empty s.
    >>> append(s, 6)  # returns None!
    >>> print(s)
    (3 4 5 6)
    """
    while ___s.rest is not Link.empty___:
        ___s = s.rest___
    ___s.rest = Link(x)___
```

More Linked List Practice

Pop

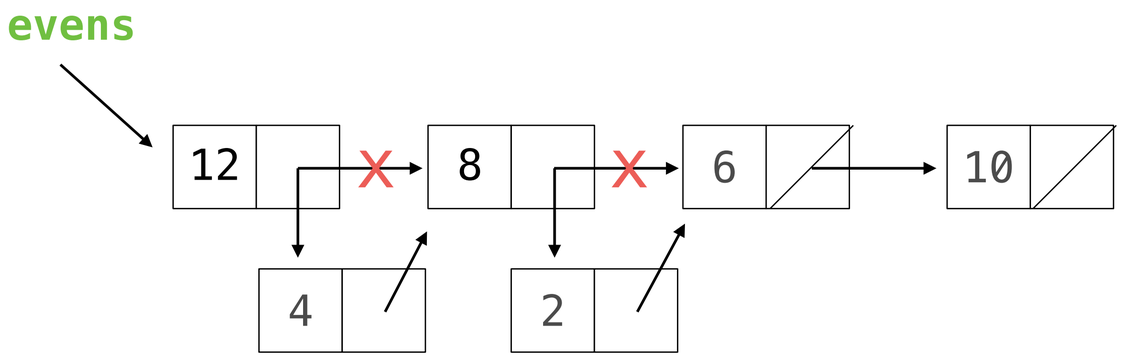
Implement pop, which takes a linked list s and positive integer i. It removes and returns the element at index i of s (assuming s.first has index 0).

```
def pop(s, i):
    """Remove and return element i from linked list s for positive i.
    >>> t = Link(3, Link(4, Link(5, Link(6))))
    >>> pop(t, 2)
    5
    >>> pop(t, 2)
    6
    >>> pop(t, 1)
    4
    >>> t
    Link(3)
    """
    assert i > 0 and i < length(s)
    for x in range(  i - 1  ):
        s = s.rest
    result = s.rest.first
    s.rest = s.rest.rest
    return   result
```



Inserting into a Linked List

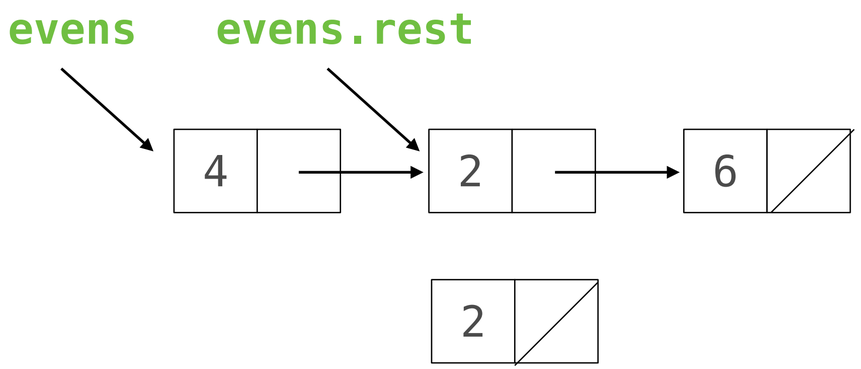
```
def insert_link(s, x, i):
    """Insert x into linked list s at index i.

    >>> evens = Link(4, Link(2, Link(6)))
    >>> insert_link(evens, 8, 1)
    >>> insert_link(evens, 10, 4)
    >>> insert_link(evens, 12, 0)
    >>> insert_link(evens, 14, 10)
    Index out of range
    >>> print(evens)
    (12 4 8 2 6 10)
    """
    if s is Link.empty:
        print('Index out of range')
    elif i == 0:
        second = Link(s.first, s.rest)
        s.first = x
        s.rest = second
    elif i == 1 and s.rest is Link.empty:
        s.rest = Link(x)
    else:
        insert_link(s.rest, x, i-1)
```



Slicing a Linked List

Normal slice notation (such as s[1:3]) doesn't work if s is a linked list.

```
def slice_link(s, i, j):
    """Return a linked list containing elements from i:j.

    >>> evens = Link(4, Link(2, Link(6)))
    >>> slice_link(evens, 1, 100)
    Link(2, Link(6))
    >>> slice_link(evens, 1, 2)
    Link(2)
    >>> slice_link(evens, 0, 2)
    Link(4, Link(2))
    >>> slice_link(evens, 1, 1) is Link.empty
    True
    """
    assert i >= 0 and j >= 0
    if  j == 0  or s is Link.empty:
        return Link.empty
    elif  i == 0 :
        return Link(s.first,  slice_link(s.rest, i, j-1) )
    else:
        return slice_link(s.rest, i-1 , j-1 )
```



slice_link(evens, 1, 2) returns

slice_link(evens.rest, 0, 1) links 2 to

slice_link(evens.rest.rest, 0, 0) returns Link.empty