

Composition

Announcements

List Efficiency

List Efficiency

Appending, assigning, and list comprehensions are fast:



Inserting (beginning/middle), slicing, and adding lists are slow:



Example: building long lists of perfect squares (numbers that are an integer times itself)

```
def using_list_comprehension(n):  
    return [k * k for k in range(n)]
```

```
def using_append(n):  
    s = []  
    for k in range(n):  
        s.append(k * k)  
    return s
```

```
def using_assign(n):  
    s = [0 for k in range(n)]  
    for k in range(n):  
        s[k] = k * k  
    return s
```

```
def using_insert(n):  
    s = []  
    for k in range(n):  
        s.insert(0, k * k)  
    return s
```

```
def using_add(n):  
    s = []  
    for k in range(n):  
        s = s + [k*k]  
    return s
```

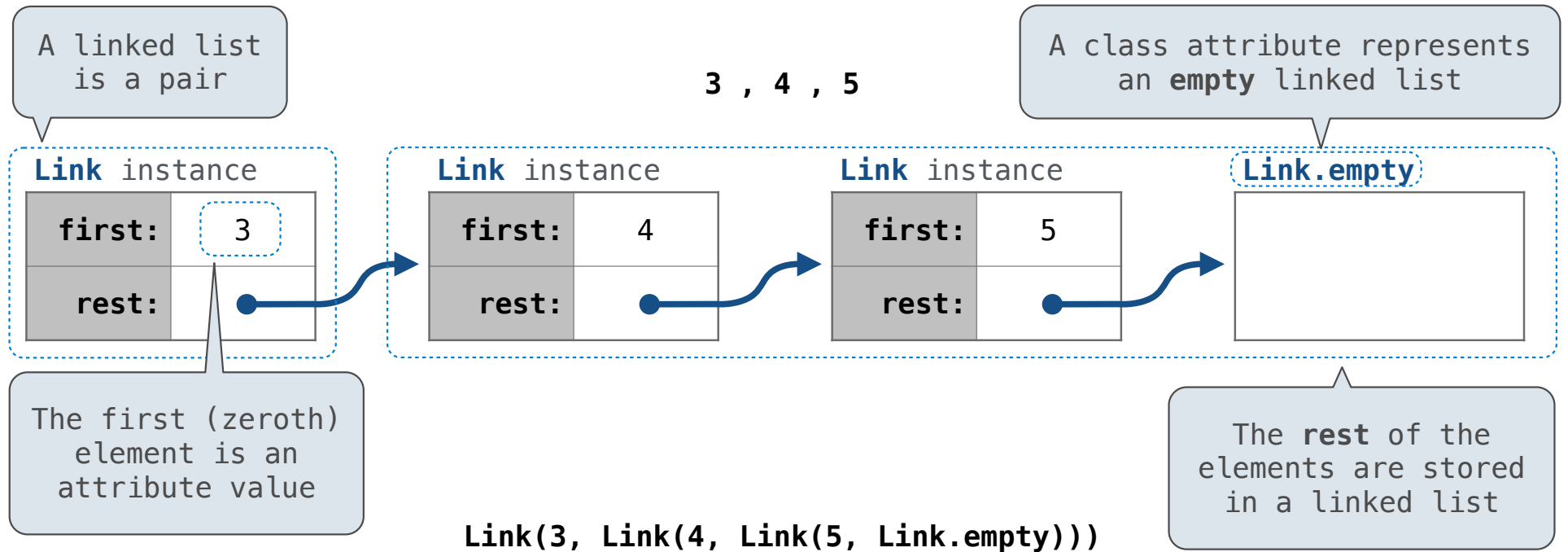
When n = 100,000

using_list_comprehension:	1.58 ms
using_append:	1.76 ms
using_assign:	2.58 ms
using_insert:	1,470 ms
using_add:	9,210 ms

Linked Lists

Linked List Structure

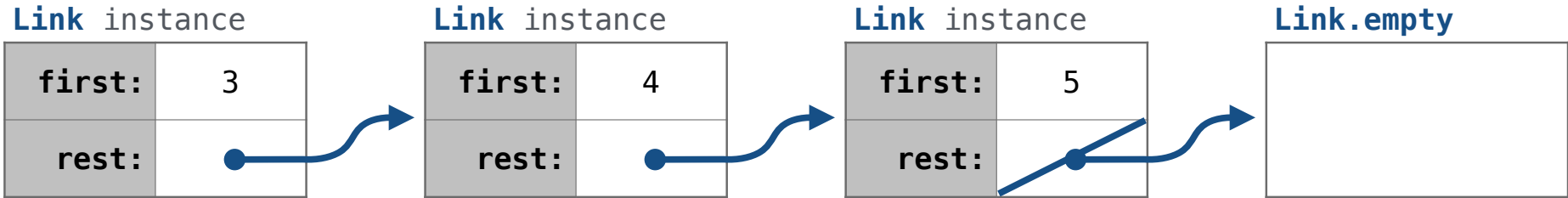
A linked list is either empty or a first value and the rest of the linked list



Linked List Structure

A linked list is either empty or a first value and the rest of the linked list

3 , 4 , 5



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

<3 4 5>



Linked List Class

Linked list class: attributes are passed to `__init__`

```
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        assert rest is Link.empty or isinstance(rest, Link)
        self.first = first
        self.rest = rest
```

Some zero-length sequence

Returns whether rest is a Link

`help(isinstance)`: Return whether an object is an instance of a class or of a subclass thereof.

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

(Demo)

Repeated Inserts

Double a List

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

    >>> 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
```

Double a Linked List

```
def double_link(s, v):  
    """Insert another v after each v in linked list s.  
  
    >>> s = Link(2, Link(7, Link(1, Link(8, Link(2, Link(8))))))  
    >>> double_link(s, 8)  
    >>> print(s)  
    <2 7 1 8 8 2 8 8>  
    """
```

```
while s is not Link.empty:
```

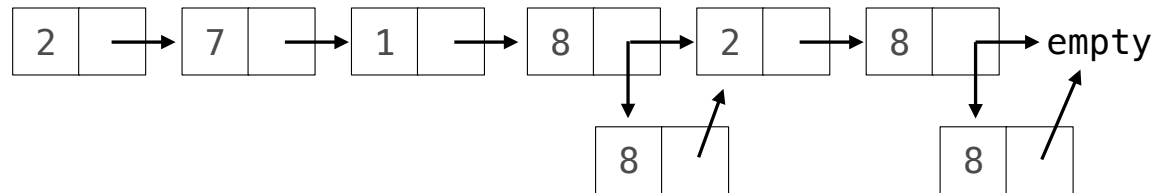
```
    if s.first == v:
```

```
        s.rest = Link(v, s.rest)
```

```
        s = s.rest.rest
```

```
    else:
```

```
        s = s.rest
```



Speed Comparison: Double a Cycle

```
def cycle(k, n):
    """Build an n-element list that cycles among range(k).

    >>> cycle(3, 10)
    [0, 1, 2, 0, 1, 2, 0, 1, 2, 0]
    """
    s = []
    for i in range(n):
        s.append(i % k)
    return s

def cycle_link(k, n):
    """Build an n-element linked list that cycles among range(k).

    >>> print(cycle_link(3, 10))
    <0 1 2 0 1 2 0 1 2 0>
    """
    first = Link.empty
    for i in range(n):
        new_link = Link(i % k)
        if first is Link.empty:
            first, last = new_link, new_link
        else:
            last.rest, last = new_link, new_link
    return first

double(cycle(5, 100000), 3): 299ms
double_link(cycle_link(5, 100000), 3): 14ms
```

Linked List Practice

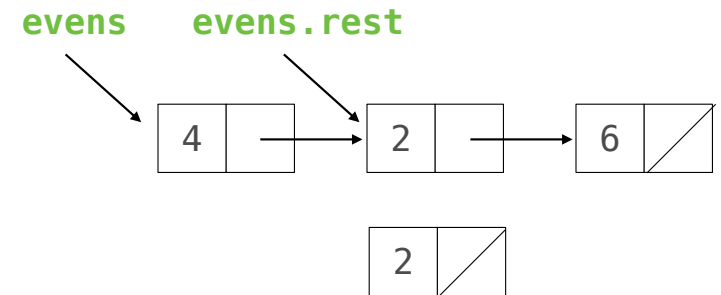
Slicing a Linked List

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 )
```

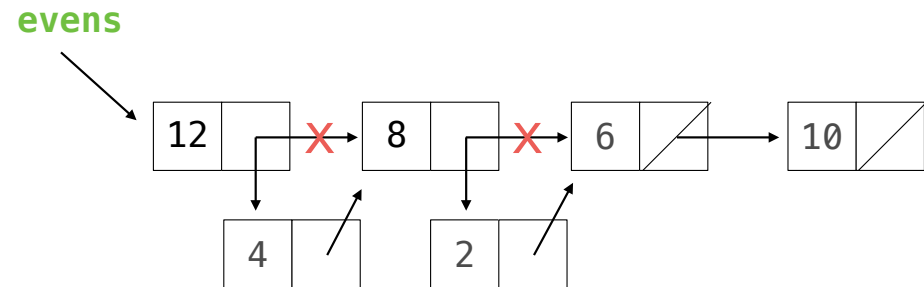


```
slice_link(evens, 1, 2) returns  
slice_link(evens.rest, 0, 1) links 2 to  
slice_link(evens.rest.rest, 0, 0) returns Link.empty
```

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)
```

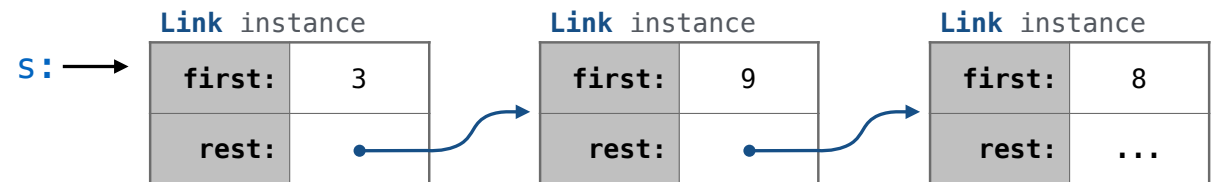


Spring 2023 Midterm 2 Question 3(b)

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

Implement `tens`, which takes a non-empty linked list of numbers `s` represented as a `Link` instance. It prints all of the prefix sums of `s` that are multiples of 10 in increasing order of the length of the prefix.

```
def tens(s):  
    """Print all prefix sums of Link s that are multiples of ten.  
    >>> tens(Link(3, Link(9, Link(8, Link(10, Link(0, Link(14, Link(6)))))))  
    20  
    30  
    30  
    50  
    .....
```



```
def f(suffix, total):  
    if total % 10 == 0:  
        print(total)  
  
    if suffix is not Link.empty:  
        f(suffix.rest, total + suffix.first)  
  
f(s.rest, s.first)
```