MT2 Review Pt 1

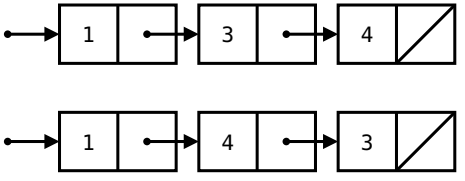
Class outline:

- Linked lists
- Lists
- Objects

Linked lists

Exercise: Is it ordered?

Is a linked list ordered from least to greatest?

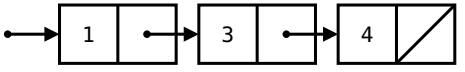


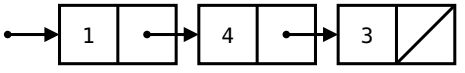

```
def ordered(s):
    """Is Link s ordered?

    >>> ordered(Link(1, Link(3, Link(4))))
    True
    >>> ordered(Link(1, Link(4, Link(3))))
    False
    >>> ordered(Link(1, Link(-3, Link(4))))
    False
    """
```

Exercise: Is it ordered? (Solution)

Is a linked list ordered from least to greatest?



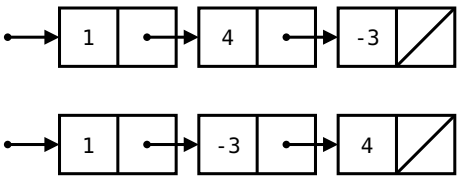



```
def ordered(s, key=lambda x: x):
    """Is Link s ordered?

    >>> ordered(Link(1, Link(3, Link(4))))
    True
    >>> ordered(Link(1, Link(4, Link(3))))
    False
    >>> ordered(Link(1, Link(-3, Link(4))))
    False
    """
    if s is Link.empty or s.rest is Link.empty:
        return True
    elif s.first > s.rest.first:
        return False
    else:
        return ordered(s.rest)
```

Exercise: Is it ordered? Part 2

Is it ordered when a key function is applied, like `abs`?

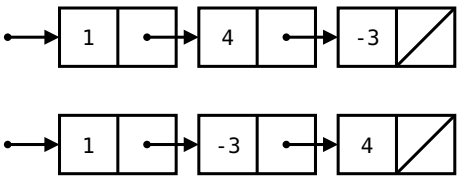



```
def ordered(s, key=lambda x: x):
    """Is Link s ordered?

    >>> ordered(Link(1, Link(3, Link(4))))
    True
    >>> ordered(Link(1, Link(4, Link(3))))
    False
    >>> ordered(Link(1, Link(-3, Link(4))))
    False
    >>> ordered(Link(1, Link(-3, Link(4))), key=abs)
    True
    >>> ordered(Link(-4, Link(-1, Link(3))))
    True
    >>> ordered(Link(-4, Link(-1, Link(3))), key=abs)
    False
    """
```

Exercise: Is it ordered? Part 2 (Solution)

Is it ordered when a key function is applied, like `abs`?

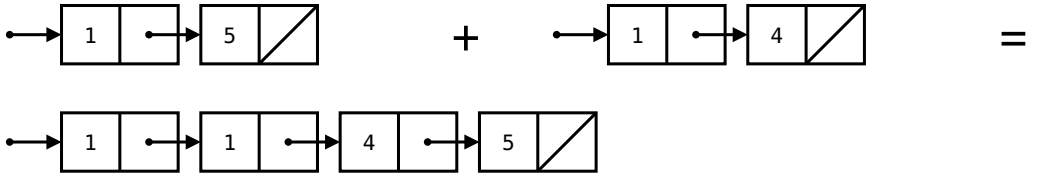



```
def ordered(s, key=lambda x: x):
    """Is Link s ordered?

    >>> ordered(Link(1, Link(3, Link(4))))
    True
    >>> ordered(Link(1, Link(4, Link(3))))
    False
    >>> ordered(Link(1, Link(-3, Link(4))))
    False
    >>> ordered(Link(1, Link(-3, Link(4))), key=abs)
    True
    >>> ordered(Link(-4, Link(-1, Link(3))))
    True
    >>> ordered(Link(-4, Link(-1, Link(3))), key=abs)
    False
    """
    if s is Link.empty or s.rest is Link.empty:
        return True
    elif key(s.first) > key(s.rest.first):
        return False
    else:
        return ordered(s.rest, key)
```

Exercise: Sorted merged list

Create a sorted Link containing all the elements of two sorted Links.

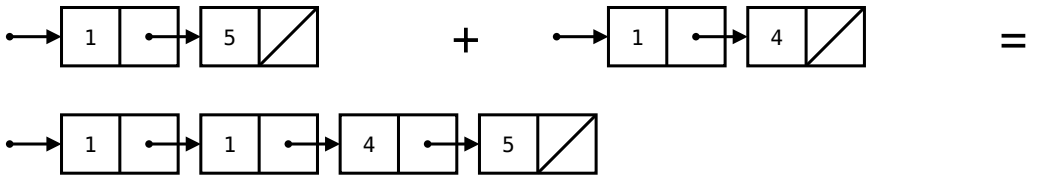



```
def merge(s, t):
    """Return a sorted Link containing the elements of sorted s & t.

    >>> a = Link(1, Link(5))
    >>> b = Link(1, Link(4))
    >>> merge(a, b)
    Link(1, Link(1, Link(4, Link(5))))
    >>> a
    Link(1, Link(5))
    >>> b
    Link(1, Link(4))
    """
```

Exercise: Sorted merged list (Solution)

Create a sorted Link containing all the elements of two sorted Links.

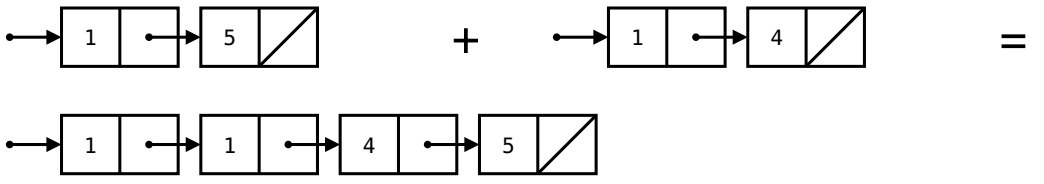



```
def merge(s, t):
    """Return a sorted Link containing the elements of sorted s & t.

    >>> a = Link(1, Link(5))
    >>> b = Link(1, Link(4))
    >>> merge(a, b)
    Link(1, Link(1, Link(4, Link(5))))
    >>> a
    Link(1, Link(5))
    >>> b
    Link(1, Link(4))
    """
    if s is Link.empty:
        return t
    elif t is Link.empty:
        return s
    elif s.first <= t.first:
        return Link(s.first, merge(s.rest, t))
    else:
        return Link(t.first, merge(s, t.rest))
```

Exercise: Sorted merged list II

This time, do it without creating any new `Link` objects.

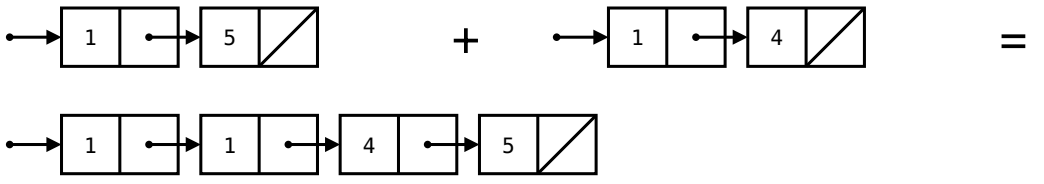



```
def merge_in_place(s, t):
    """Return a sorted Link containing the elements of sorted s & t.

    >>> a = Link(1, Link(5))
    >>> b = Link(1, Link(4))
    >>> merge_in_place(a, b)
    Link(1, Link(1, Link(4, Link(5))))
    >>> a
    Link(1, Link(1, Link(4, Link(5))))
    >>> b
    Link(1, Link(4, Link(5)))
    """
```

Exercise: Sorted merged list II (Solution)

This time, do it without creating any new `Link` objects.




```
def merge_in_place(s, t):
    """Return a sorted Link containing the elements of sorted s & t.

    >>> a = Link(1, Link(5))
    >>> b = Link(1, Link(4))
    >>> merge_in_place(a, b)
    Link(1, Link(1, Link(4, Link(5))))
    >>> a
    Link(1, Link(1, Link(4, Link(5))))
    >>> b
    Link(1, Link(4, Link(5)))
    """
    if s is Link.empty:
        return t
    elif t is Link.empty:
        return s
    elif s.first <= t.first:
        s.rest = merge_in_place(s.rest, t)
        return s
    else:
        t.rest = merge_in_place(s, t.rest)
        return t
```

Iterables & Iterators

Exercise: Find indices

What are the indices of all elements in a list that have the smallest absolute value?

```
[-4, -3, -2,  3,  2,  4]  → [2, 4]
  0   1   2   3   4   5
```

```
[ 1,  2,  3,  4,  5,  6]  → [0]
  0   1   2   3   4   5
```

```
def min_abs_indices(s):
    """Indices of all elements in list s that have the smallest absolute value.

    >>> min_abs_indices([-4, -3, -2, 3, 2, 4])
    [2, 4]
    >>> min_abs_indices([1, 2, 3, 4, 5])
    [0]
    """
```

Exercise: Find indices (Solution)

What are the indices of all elements in a list that have the smallest absolute value?

```
[-4, -3, -2,  3,  2,  4]  → [2, 4]
  0   1   2   3   4   5
```

```
[ 1,  2,  3,  4,  5,  6]  → [0]
  0   1   2   3   4   5
```

```
def min_abs_indices(s):
    """Indices of all elements in list s that have the smallest absolute value.

    >>> min_abs_indices([-4, -3, -2, 3, 2, 4])
    [2, 4]
    >>> min_abs_indices([1, 2, 3, 4, 5])
    [0]
    """
    min_abs = min(map(abs, s))
    return list(filter(lambda i: abs(s[i]) == min_abs, range(len(s))))
    # OR
    return [i for i in range(len(s)) if abs(s[i]) == min_abs]
```

Exercise: Largest sum

What's the largest sum of two adjacent elements in a list? (Assume length > 1)

```
[-4,  -3,  -2,   3,   2,   4]  → 6
   -7    -5    1    5    6
```

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

```
def largest_adj_sum(s):
    """Largest sum of two adjacent elements in a list s.

    >>> largest_adj_sum([-4, -3, -2, 3, 2, 4])
    6
    >>> largest_adj_sum([-4, 3, -2, -3, 2, -4])
    1
    """
```

Exercise: Largest sum (Solution)

What's the largest sum of two adjacent elements in a list? (Assume length > 1)

```
[-4,  -3,  -2,   3,   2,   4]  → 6
   -7    -5    1    5    6
```

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

```
def largest_adj_sum(s):
    """Largest sum of two adjacent elements in a list s.

    >>> largest_adj_sum([-4, -3, -2, 3, 2, 4])
    6
    >>> largest_adj_sum([-4, 3, -2, -3, 2, -4])
    1
    """
    return max([x + y for x, y in zip(s[:-1], s[1:])])
    # OR
    return max([s[i] + s[i + 1] for i in range(len(s) - 1)])
    # OR
    return max(map(lambda i: s[i] + s[i + 1], range(len(s) - 1)))
```