Data Examples
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`?

```python
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`?

```python
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)
```
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.
    """
    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))
```

```
>>> 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))
```
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]

[ 1, 2, 3, 4, 5, 6] → [0]
```

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?

```python
[-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)

```python
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
    """
```

[-4, -3, -2, 3, 2, 4] → 6
-7 -5 1 5 6

[-4, 3, -2, -3, 2, -4] → 1
-1 1 -5 -1 -2
Exercise: Largest sum (Solution)

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

\[
\begin{array}{cccccc}
-4 & -3 & -2 & 3 & 2 & 4 \\
-7 & -5 & 1 & 5 & 6 \\
\end{array}
\]

\[
\begin{array}{cccccc}
-4 & 3 & -2 & -3 & 2 & -4 \\
-1 & 1 & -5 & -1 & -2 \\
\end{array}
\]

def largest_adj_sum(s):
   
   
   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)))
Exercise: Digits dictionary

Create a dictionary mapping each digit d to the lists of elements in s that end with d.

```
[5, 8, 13, 21, 34, 55, 89] → {1: [21], 3: [13], 4: [34], 5: [5, 55], 8: [8], 9: [89]}
```

```
def digit_dict(s):
    """Map each digit d to the lists of elements in s that end with d.

    >>> digit_dict([5, 8, 13, 21, 34, 55, 89])
    {1: [21], 3: [13], 4: [34], 5: [5, 55], 8: [8], 9: [89]}
    """
```
Exercise: Digits dictionary (Solution)

Create a dictionary mapping each digit \(d\) to the lists of elements in \(s\) that end with \(d\).

\[
[5, 8, 13, 21, 34, 55, 89] \rightarrow \{1: [21], 3: [13], 4: [34], 5: [5, 55], 8: [8], 9: [89]\}
\]

```python
def digit_dict(s):
    """Map each digit \(d\) to the lists of elements in \(s\) that end with \(d\)."

    >>> digit_dict([5, 8, 13, 21, 34, 55, 89])
    \{1: [21], 3: [13], 4: [34], 5: [5, 55], 8: [8], 9: [89]\}
    """
    return {i: [x for x in s if x % 10 == i]
             for i in range(10) if any([x % 10 == i for x in s])}
    # OR
    last_digits = list(map(lambda x: x % 10, s))
    return {i: [x for x in s if x % 10 == i]
             for i in range(10) if i in last_digits}
```
Exercise: Element comparer

Does every element equal some other element in s?

[-4, -3, -2, 3, 2, 4] → False
[4, 3, 2, 3, 2, 4] → True

def all_have_an_equal(s):
    """Does every element equal some other element in s?"

    >>> all_have_an_equal([-4, -3, -2, 3, 2, 4])
    False
    >>> all_have_an_equal([4, 3, 2, 3, 2, 4])
    True
    """
Exercise: Element comparer (Solution)

Does every element equal some other element in s?

[[-4, -3, -2, 3, 2, 4] → False
[4, 3, 2, 3, 2, 4] → True

```python
def all_have_an_equal(s):
    """Does every element equal some other element in s?"

    >>> all_have_an_equal([-4, -3, -2, 3, 2, 4])
    False
    >>> all_have_an_equal([4, 3, 2, 3, 2, 4])
    True
    """
    return min([sum([1 for y in s if x == y]) for x in s]) > 1
    # OR
    return all([s[i] in s[:i] + s[i+1:] for i in range(len(s))])
    # OR
    return all(map(lambda x: s.count(x) > 1, s))
```
Lists in environment diagrams
List operations

Starting from:

\[ s = [2, 3] \]
\[ t = [5, 6] \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
</table>
| append          | adds one element to a list | \( s\text{.append}(t) \)
                 |                           | \( t = 0 \)          |
| extend          | adds all elements in one list to another list | \( s\text{.extend}(t) \)
                 |                           | \( t[1] = 0 \)        |

addition & slicing create new lists containing existing elements

\[ a = s + [t] \]
\[ b = a[1:] \]
\[ a[1] = 9 \]
\[ b[1][1] = 0 \]
# List operations

## Starting from:

\[ s = [2, 3] \]
\[ t = [5, 6] \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>append</td>
<td>[ s.append(t) ]</td>
<td>[ s \rightarrow [2, 3, [5, 6]] ]</td>
</tr>
<tr>
<td></td>
<td>[ t = 0 ]</td>
<td>[ t \rightarrow 0 ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>extend</td>
<td>[ s.extend(t) ]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[ t[1] = 0 ]</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>addition &amp; slicing</td>
<td>[ a = s + [t] ]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[ b = a[1:] ]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[ a[1] = 9 ]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[ b[1][1] = 0 ]</td>
<td></td>
</tr>
</tbody>
</table>
# List operations

Starting from:

\[
\begin{align*}
  s &= [2, 3] \\
  t &= [5, 6]
\end{align*}
\]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>append</strong></td>
<td>(s.append(t))</td>
<td>(s \rightarrow [2, 3, [5, 6]])</td>
</tr>
<tr>
<td></td>
<td>(t = 0)</td>
<td>(t \rightarrow 0)</td>
</tr>
<tr>
<td><strong>extend</strong></td>
<td>(s.extend(t))</td>
<td>(s \rightarrow [2, 3, 5, 6])</td>
</tr>
<tr>
<td></td>
<td>(t[1] = 0)</td>
<td>(t \rightarrow [5, 0])</td>
</tr>
<tr>
<td>Addition &amp; Slicing</td>
<td>(a = s + [t])</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(b = a[1:])</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(a[1] = 9)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(b[1][1] = 0)</td>
<td></td>
</tr>
</tbody>
</table>
# List operations

Starting from:

\[
\begin{align*}
  s &= [2, 3] \\
  t &= [5, 6]
\end{align*}
\]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>append</strong></td>
<td><code>s.append(t)</code></td>
<td><code>s \rightarrow [2, 3, [5, 6]]</code></td>
</tr>
<tr>
<td></td>
<td><code>t = 0</code></td>
<td><code>t \rightarrow 0</code></td>
</tr>
<tr>
<td><strong>extend</strong></td>
<td><code>s.extend(t)</code></td>
<td><code>s \rightarrow [2, 3, 5, 6]</code></td>
</tr>
<tr>
<td></td>
<td><code>t[1] = 0</code></td>
<td><code>t \rightarrow [5, 0]</code></td>
</tr>
<tr>
<td><strong>addition &amp; slicing</strong></td>
<td><code>a = s + [t]</code></td>
<td><code>a \rightarrow [2, 3]</code></td>
</tr>
<tr>
<td></td>
<td><code>b = a[1:]</code></td>
<td><code>b \rightarrow [3, [5, 0]]</code></td>
</tr>
<tr>
<td></td>
<td><code>a[1] = 9</code></td>
<td><code>a \rightarrow [2, 9, [5, 0]]</code></td>
</tr>
<tr>
<td></td>
<td><code>b[1][1] = 0</code></td>
<td><code>b \rightarrow [3, [5, 0]]</code></td>
</tr>
</tbody>
</table>
List operations

Starting from:

\[ s = [2, 3] \]
\[ t = [5, 6] \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>The <code>list</code> constructor also creates a new list containing existing elements</td>
<td><code>t = list(s)</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>s[1] = 0</code></td>
<td></td>
</tr>
<tr>
<td>slice assignment replaces a slice with new values</td>
<td><code>s[0:0] = t</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>s[3:] = t</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>t[1] = 0</code></td>
<td></td>
</tr>
</tbody>
</table>
# List operations

## Starting from:

\[
\begin{align*}
    s &= [2, 3] \\
    t &= [5, 6]
\end{align*}
\]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
</table>
| The **list** constructor also creates a new list containing existing elements | \[
    t = \text{list}(s) \\
    s[1] = 0
\] | \[
    s \rightarrow [2, 0] \\
    t \rightarrow [2, 3]
\] |
| slice assignment replaces a slice with new values | \[
    s[0:0] = t \\
    s[3:] = t \\
    t[1] = 0
\] | |
# List operations

## Starting from:

\[
\begin{align*}
    s &= [2, 3] \\
    t &= [5, 6]
\end{align*}
\]

## Operation

The **list** constructor also creates a new list containing existing elements

\[
\begin{align*}
    t &= \text{list}(s) \\
    s[1] &= 0 \\
    t[1] &= 0
\end{align*}
\]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
</table>
| slice assignment replaces a slice with new values | \[
\begin{align*}
    s[0:0] &= t \\
    s[3:] &= t \\
    t[1] &= 0
\end{align*}
\] | \[
\begin{align*}
    s &= [5, 6, 2, 5, 6] \\
    t &= [5, 0]
\end{align*}
\] |
Lists in lists

t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)

View in PythonTutor

t = [[1, 2], [3, 4]]
t[0].append(t[1:2])

View in PythonTutor
Objects
Santa's helpers

class Elf:
    greeting = 'Boss'
    def __init__(self):
        self.shelf = Elf
    def work(self):
        return self.greeting + ', I toil all day'
    def __repr__(self):
        return Santa.greeting

class Santa(Elf):
    greeting = 'Elfie'
    def work(self):
        print(Elf.work(self))
        return 'My job is to break into kid\'s homes!'

jack = Elf()
klaus = Santa()
jack.greeting = 'Your Jollyness'

>>> Elf().work()

>>> jack
>>> jack.work()

>>> klaus.work()

>>> klaus.shelf.work(klaus)
```python
class Elf:
    greeting = 'Boss'
    def __init__(self):
        self.shelf = Elf
    def work(self):
        return self.greeting + ', I toil all day'
    def __repr__(self):
        return Santa.greeting

class Santa(Elf):
    greeting = 'Elfie'
    def work(self):
        print(Elf.work(self))
        return 'My job is to break into kid\'s homes!'

jack = Elf()
klaus = Santa()
jack.greeting = 'Your Jollyness'

>>> Elf().work()
'Boss, I toil all day'
>>> jack
```
>>> jack.work()

>>> klaus.work()

>>> klaus.shelf.work(klaus)
Santa's helpers

class Elf:
    greeting = 'Boss'
    def __init__(self):
        self.shelf = Elf
    def work(self):
        return self.greeting + ', I toil all day'
    def __repr__(self):
        return Santa.greeting

class Santa(Elf):
    greeting = 'Elfie'
    def work(self):
        print(Elf.work(self))
        return 'My job is to break into kid\'s homes!'

jack = Elf()
klaus = Santa()
jack.greeting = 'Your Jollyness'

>>> Elf().work()
'Boss, I toil all day'
>>> jack
Elfie
>>> jack.work()

>>> klaus.work()

>>> klaus.shelf.work(klaus)
class Elf:
    greeting = 'Boss'
    def __init__(self):
        self.shelf = Elf
    def work(self):
        return self.greeting + ', I toil all day'
    def __repr__(self):
        return Santa.greeting

class Santa(Elf):
    greeting = 'Elfie'
    def work(self):
        print(Elf.work(self))
        return 'My job is to break into kid\'s homes!'

jack = Elf()
klaus = Santa()
jack.greeting = 'Your Jollyness'

>>> Elf().work()
'Boss, I toil all day'
>>> jack
Elfie
>>> jack.work()
'Your Jollyness, I toil all day'

>>> klaus.work()

>>> klaus.shelf.work(klaus)
Santa's helpers

class Elf:
    greeting = 'Boss'
    def __init__(self):
        self.shelf = Elf
    def work(self):
        return self.greeting + ', I toil all day'
    def __repr__(self):
        return Santa.greeting

class Santa(Elf):
    greeting = 'Elfie'
    def work(self):
        print(Elf.work(self))
        return 'My job is to break into kid\'s homes!'

jack = Elf()
klaus = Santa()
jack.greeting = 'Your Jollyness'

>>> Elf().work()
'Boss, I toil all day'
>>> jack
Elfie
>>> jack.work()
'Your Jollyness, I toil all day'
>>> klaus.work()
Elfie, I toil all day

>>> klaus.shelf.work(klaus)
class Elf:
    greeting = 'Boss'
    def __init__(self):
        self.shelf = Elf
    def work(self):
        return self.greeting + ', I toil all day'
    def __repr__(self):
        return Santa.greeting

class Santa(Elf):
    greeting = 'Elfie'
    def work(self):
        print(Elf.work(self))
        return 'My job is to break into kid\'s homes!'

jack = Elf()
klaus = Santa()
jack.greeting = 'Your Jollyness'

>>> Elf().work()
'Boss, I toil all day'
>>> jack
Elfie
jack.work()
'Your Jollyness, I toil all day'
>>> klaus.work()
Elfie, I toil all day
"My job is to break into kid's homes!"
>>> klaus.shelf.work(klaus)
Santa's helpers

class Elf:
    greeting = 'Boss'
    def __init__(self):
        self.shelf = Elf
    def work(self):
        return self.greeting + ', I toil all day'
    def __repr__(self):
        return Santa.greeting

class Santa(Elf):
    greeting = 'Elfie'
    def work(self):
        print(Elf.work(self))
        return 'My job is to break into kid\'s homes!'

jack = Elf()
klaus = Santa()
jack.greeting = 'Your Jollyness'

>>> Elf().work()
'Boss, I toil all day'
>>> jack
Elfie
>>> jack.work()
'Your Jollyness, I toil all day'
>>> klaus.work()
Elfie, I toil all day
"My job is to break into kid's homes!"
>>> klaus.shelf.work(klaus)
'Elfie, I toil all day'
Python Project of The Day!
Outreachy

Outreachy: An organization that provides internships in open source to people subject to systemic bias and impacted by underrepresentation in the technical industry where they are living.

Website written in Django, a popular Python web framework.

Github repository