Composition
Announcements
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
Linked List Structure

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

\[ 3, 4, 5 \]

**Link instance**

<table>
<thead>
<tr>
<th>first:</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>rest:</td>
<td></td>
</tr>
</tbody>
</table>
A linked list is either empty or a first value and the rest of the linked list.

\[ 3, 4, 5 \]
Linked List Structure

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

3, 4, 5
A linked list is either empty or a first value and the rest of the linked list.

3, 4, 5
A linked list is either empty or a first value and the rest of the linked list.

A linked list is a pair.

A linked list instance:

<table>
<thead>
<tr>
<th>first</th>
<th>rest</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Link instance:

<table>
<thead>
<tr>
<th>first</th>
<th>rest</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Link instance:

<table>
<thead>
<tr>
<th>first</th>
<th>rest</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

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

A linked list is a pair:

- The first (zeroth) element is an attribute value.
A linked list is either empty or a first value and the rest of the linked list.

A linked list is a pair

The first (zeroth) element is an attribute value

The rest of the elements are stored in a linked list

3, 4, 5
A linked list is either empty or a first value and the rest of the linked list.

A linked list is a pair.

The first (zeroth) element is an attribute value.

A class attribute represents an empty linked list.

The rest of the elements are stored in a linked list.
A linked list is either empty or a first value and the rest of the linked list.

A linked list is a pair.

The first (zeroth) element is an attribute value.

The rest of the elements are stored in a linked list.

A class attribute represents an empty linked list.

```
Link(3, Link(4, Link(5, Link.empty)))
```
A linked list is either empty or a first value and the rest of the linked list.

```
Link(3, Link(4, Link(5, Link.empty)))
```
Linked List Structure

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

```
3, 4, 5
```

```
Link instance
first: 3
rest: |

Link instance
first: 4
rest: |

Link instance
first: 5
rest: |

Link.empty
```

```
Link(3, Link(4, Link(5, Link.empty)))
```
A linked list is either empty or a first value and the rest of the linked list.

```
3, 4, 5

Link instance
| first: | 3 |
| rest:   |

Link instance
| first: | 4 |
| rest:   |

Link instance
| first: | 5 |
| rest:   |

Link.empty
```

`Link(3, Link(4, Link(5, Link.empty)))`
A linked list is either empty or a first value and the rest of the linked list.

```
3, 4, 5
```

```
Link instance
first: 3
rest:  

Link instance
first: 4
rest:  

Link instance
first: 5
rest:  

Link.empty
```

```
Link(3, Link(4, Link(5, Link.empty)))
```
A linked list is either empty or a first value and the rest of the linked list.

\[ \text{Link(3, Link(4, Link(5, Link.empty)))} \]
A linked list is either empty or a first value and the rest of the linked list:

\[3, 4, 5\]

\[
\text{Link instance} \\
\begin{array}{|c|}
\hline
\text{first:} & 3 \\
\text{rest:} & \text{Link instance} \\
\hline
\end{array}
\]

\[
\text{Link instance} \\
\begin{array}{|c|}
\hline
\text{first:} & 4 \\
\text{rest:} & \text{Link instance} \\
\hline
\end{array}
\]

\[
\text{Link instance} \\
\begin{array}{|c|}
\hline
\text{first:} & 5 \\
\text{rest:} & \text{Link(3, Link(4, Link(5)))} \\
\hline
\end{array}
\]
Linked List Class

\[
\text{Link}(3, \text{Link}(4, \text{Link}(5)))
\]
Linked List Class

Linked list class: attributes are passed to __init__

```
Link(3, Link(4, Link(5)))
```
Linked List Class

Linked list class: attributes are passed to `__init__`

class Link:

```
Link(3, Link(4, Link(5)
```
Linked List Class

Linked list class: attributes are passed to `__init__`

```python
class Link:
    def __init__(self, first, rest=empty):
        Link(3, Link(4, Link(5)))
```
Linked List Class

Linked list class: attributes are passed to \texttt{\_\_init\_}

\begin{verbatim}
class Link:

    def \_\_init\_(self, first, rest=empty):
        assert rest is Link.empty or isinstance(rest, Link)

    Link(3, Link(4, Link(5)))
\end{verbatim}
Linked List Class

Linked list class: attributes are passed to __init__

class Link:

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

      Link(3, Link(4, Link(5)))
Linked List Class

Linked list class: attributes are passed to __init__

```python
class Link:

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

```
Link(3, Link(4, Link(5)))
```
Linked List Class

Linked list class: attributes are passed to `__init__`

class Link:

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

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

Link(3, Link(4, Link(5)))
Linked List Class

Linked list class: attributes are passed to __init__

```python
class Link:
    empty = ()

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

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

```
Link(3, Link(4, Link(5 )))
```
Linked List Class

Linked list class: attributes are passed to __init__

```python
class Link:
    empty = ()  # Some zero-length sequence

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

    Link(3, Link(4, Link(5)))  # Returns whether rest is a Link
```

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

Linked list class: attributes are passed to `__init__`

```python
class Link:
    empty = ()  # Some zero-length sequence

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

```python
Demonstration:
Link(3, Link(4, Link(5)))
```

`help(isinstance)`: Return whether an object is an instance of a class or of a subclass thereof.
Linked List Processing
Example: Range, Map, and Filter for Linked Lists
Example: Range, Map, and Filter for Linked Lists

square, odd = lambda x: x * x, lambda x: x % 2 == 1
Example: Range, Map, and Filter for Linked Lists

```
square, odd = lambda x: x * x, lambda x: x % 2 == 1
list(map(square, filter(odd, range(1, 6))))  # [1, 9, 25]
```
Example: Range, Map, and Filter for Linked Lists

\[
\text{square, odd = } \lambda x: x \times x, \lambda x: x \mod 2 == 1 \\
\text{list(map(square, filter(odd, range(1, 6))))} \quad \# \ [1, 9, 25] \\
\text{map_link(square, filter_link(odd, range_link(1, 6)))} \quad \# \ \text{Link(1, Link(9, Link(25)))}
\]
Example: Range, Map, and Filter for Linked Lists

```
square, odd = lambda x: x * x, lambda x: x % 2 == 1
list(map(square, filter(odd, range(1, 6))))  # [1, 9, 25]
map_link(square, filter_link(odd, range_link(1, 6)))  # Link(1, Link(9, Link(25)))

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)))
```

Example: Range, Map, and Filter for Linked Lists

```
square, odd = lambda x: x * x, lambda x: x % 2 == 1
list(map(square, filter(odd, range(1, 6))))  # [1, 9, 25]
map_link(square, filter_link(odd, range_link(1, 6)))  # Link(1, Link(9, Link(25)))
```

```python
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)))
    """

def map_link(f, s):
    """Return a Link that contains f(x) for each x in Link s."
    >>> map_link(square, range_link(3, 6))
    Link(9, Link(16, Link(25)))
    """
```
Example: Range, Map, and Filter for Linked Lists

```
square, odd = lambda x: x * x, lambda x: x % 2 == 1
list(map(square, filter(odd, range(1, 6))))  # [1, 9, 25]
map_link(square, filter_link(odd, range_link(1, 6)))  # Link(1, Link(9, Link(25)))

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)))
    """

def map_link(f, s):
    """Return a Link that contains f(x) for each x in Link s."
    >>> map_link(square, range_link(3, 6))
    Link(9, Link(16, Link(25)))
    """

def filter_link(f, s):
    """Return a Link that contains only the elements x of Link s for which f(x)
    is a true value."
    >>> filter_link(odd, range_link(3, 6))
    Link(3, Link(5))
    """
```
Linked Lists Mutation
Linked Lists Can Change

Attribute assignment statements can change first and rest attributes of a Link
Linked Lists Can Change

Attribute assignment statements can change first and rest attributes of a Link.

The rest of a linked list can contain the linked list as a sub-list.
Linked Lists Can Change

Attribute assignment statements can change first and rest attributes of a Link

The rest of a linked list can contain the linked list as a sub-list

```python
 s = Link(1, Link(2, Link(3)))
```
Linked Lists Can Change

Attribute assignment statements can change first and rest attributes of a Link

The rest of a linked list can contain the linked list as a sub-list

```python
>>> s = Link(1, Link(2, Link(3)))
```
Linked Lists Can Change

Attribute assignment statements can change first and rest attributes of a Link

The rest of a linked list can contain the linked list as a sub-list

```python
>>> s = Link(1, Link(2, Link(3)))
```

Note: The actual environment diagram is much more complicated.
Linked Lists Can Change

Attribute assignment statements can change first and rest attributes of a Link

The rest of a linked list can contain the linked list as a sub-list

```python
>>> s = Link(1, Link(2, Link(3)))
```

Note: The actual environment diagram is much more complicated.
Linked Lists Can Change

Attribute assignment statements can change first and rest attributes of a Link

The rest of a linked list can contain the linked list as a sub-list

```python
>>> s = Link(1, Link(2, Link(3)))
>>> s.first = 5
```
Linked Lists Can Change

Attribute assignment statements can change first and rest attributes of a Link

The rest of a linked list can contain the linked list as a sub-list

```python
>>> s = Link(1, Link(2, Link(3)))
>>> s.first = 5
>>> t = s.rest
```

Note: The actual environment diagram is much more complicated.
Linked Lists Can Change

Attribute assignment statements can change first and rest attributes of a Link

The rest of a linked list can contain the linked list as a sub-list

```python
>>> s = Link(1, Link(2, Link(3)))
>>> s.first = 5
>>> t = s.rest
>>> t.rest = s
```

Note: The actual environment diagram is much more complicated.
Linked Lists Can Change

Attribute assignment statements can change first and rest attributes of a Link

The rest of a linked list can contain the linked list as a sub-list

```python
>>> s = Link(1, Link(2, Link(3)))
>>> s.first = 5
>>> t = s.rest
>>> t.rest = s
>>> s.first
```

Note: The actual environment diagram is much more complicated.
Linked Lists Can Change

Attribute assignment statements can change first and rest attributes of a Link

The rest of a linked list can contain the linked list as a sub-list

```python
>>> s = Link(1, Link(2, Link(3)))
>>> s.first = 5
>>> t = s.rest
>>> t.rest = s
>>> s.first
5
```

Note: The actual environment diagram is much more complicated.
**Linked Lists Can Change**

Attribute assignment statements can change first and rest attributes of a Link

The rest of a linked list can contain the linked list as a sub-list

```python
>>> s = Link(1, Link(2, Link(3)))
>>> s.first = 5
>>> t = s.rest
>>> t.rest = s
>>> s.first
5
>>> s.rest.rest.rest.rest.rest.first
```

Note: The actual environment diagram is much more complicated.
**Linked Lists Can Change**

Attribute assignment statements can change first and rest attributes of a Link

The rest of a linked list can contain the linked list as a sub-list

```python
>>> s = Link(1, Link(2, Link(3)))
>>> s.first = 5
>>> t = s.rest
>>> t.rest = s
>>> s.first
5
>>> s.rest.rest.rest.rest.first
2
```

Note: The actual environment diagram is much more complicated.
Linked Lists Can Change

Attribute assignment statements can change first and rest attributes of a Link

The rest of a linked list can contain the linked list as a sub-list

```python
>>> s = Link(1, Link(2, Link(3)))
>>> s.first = 5
>>> t = s.rest
>>> t.rest = s
>>> s.first
5
>>> s.rest.rest.rest.rest.rest.first
2
```

![Environment Diagram]

Note: The actual environment diagram is much more complicated.
Linked List Mutation Example
Adding to an Ordered List

- **Link** instance
  - first: 1
  - rest: 

- **Link** instance
  - first: 3
  - rest: 

- **Link** instance
  - first: 5
  - rest: 

s:
Adding to an Ordered List

```
def add(s, v):
    """Add v to an ordered list s with no repeats, returning modified s."""
```
def add(s, v):
    """Add v to an ordered list s with no repeats, returning modified s."""
    (Note: If v is already in s, then don't modify s, but still return it.)
def add(s, v):
    """Add v to an ordered list s with no repeats, returning modified s."""
    (Note: If v is already in s, then don't modify s, but still return it.)

    add(s, 0)
Adding to an Ordered List

```
def add(s, v):
    """Add v to an ordered list s with no repeats, returning modified s."""
    (Note: If v is already in s, then don't modify s, but still return it.)

    add(s, 0)
```
Adding to an Ordered List

```python
def add(s, v):
    """Add v to an ordered list s with no repeats, returning modified s."""
    (Note: If v is already in s, then don't modify s, but still return it.)

    add(s, 0)    add(s, 3)
```
Adding to an Ordered List

```
def add(s, v):
    """Add v to an ordered list s with no repeats, returning modified s."""
    (Note: If v is already in s, then don't modify s, but still return it.)

    add(s, 0)  add(s, 3)  add(s, 4)
```
Adding to an Ordered List

```python
def add(s, v):
    """Add v to an ordered list s with no repeats..."""

    add(s, 0)  # add(0)
    add(s, 3)  # add(3)
    add(s, 4)  # add(4)
```
Adding to an Ordered List

```python
def add(s, v):
    """Add v to an ordered list s with no repeats..."""
```

```
add(s, 0)      add(s, 3)      add(s, 4)      add(s, 6)
```
Adding to an Ordered List

```python
def add(s, v):
    """Add v to an ordered list s with no repeats..."""
    add(s, 0)  add(s, 3)  add(s, 4)  add(s, 6)
```
Adding to a Set Represented as an Ordered List

\[ s:\]

\[
\begin{array}{|c|c|}
\hline
\text{Link instance} & \\
\text{first} & 0 \\
\text{rest} & \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
\text{Link instance} & \\
\text{first} & 1 \\
\text{rest} & \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
\text{Link instance} & \\
\text{first} & 3 \\
\text{rest} & \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
\text{Link instance} & \\
\text{first} & 4 \\
\text{rest} & \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
\text{Link instance} & \\
\text{first} & 5 \\
\text{rest} & \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
\text{Link instance} & \\
\text{first} & 6 \\
\text{rest} & \\
\hline
\end{array}
\]
Adding to a Set Represented as an Ordered List

def add(s, v):

```python
s:
    Link instance
    +------------------+
    | first: | rest: |
    +------------------+
    |   0     |      |
    +------------------+

    Link instance
    +------------------+
    | first: | rest: |
    +------------------+
    |   1     |      |
    +------------------+

    Link instance
    +------------------+
    | first: | rest: |
    +------------------+
    |   3     |      |
    +------------------+

    Link instance
    +------------------+
    | first: | rest: |
    +------------------+
    |   4     |      |
    +------------------+

    Link instance
    +------------------+
    | first: | rest: |
    +------------------+
    |   5     |      |
    +------------------+

    Link instance
    +------------------+
    | first: | rest: |
    +------------------+
    |   6     |      |
    +------------------+
```
Adding to a Set Represented as an Ordered List

def add(s, v):
    """Add v to s, returning modified s."""

s:
Adding to a Set Represented as an Ordered List

def add(s, v):
    """Add v to s, returning modified s.""

>>> s = Link(1, Link(3, Link(5)))
Adding to a Set Represented as an Ordered List

def add(s, v):
    """Add v to s, returning modified s."""

>>> s = Link(1, Link(3, Link(5)))
>>> add(s, 0)
Link(0, Link(1, Link(3, Link(5)))))
Adding to a Set Represented as an Ordered List

def add(s, v):
    """Add v to s, returning modified s."""

    >>> s = Link(1, Link(3, Link(5)))
    >>> add(s, 0)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 3)
    Link(0, Link(1, Link(3, Link(5)))))
Adding to a Set Represented as an Ordered List

def add(s, v):
    """Add v to s, returning modified s."""

    >>> s = Link(1, Link(3, Link(5)))
    >>> add(s, 0)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 3)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 4)
    Link(0, Link(1, Link(3, Link(4, Link(5)))))


![Diagram showing the process of adding elements to a set represented as an ordered list.](image)
Adding to a Set Represented as an Ordered List

def add(s, v):
    '''Add v to s, returning modified s.'''

>>> s = Link(1, Link(3, Link(5)))
>>> add(s, 0)
Link(0, Link(1, Link(3, Link(5)))))
>>> add(s, 3)
Link(0, Link(1, Link(3, Link(5)))))
>>> add(s, 4)
Link(0, Link(1, Link(3, Link(4, Link(5)))))
>>> add(s, 6)
Link(0, Link(1, Link(3, Link(4, Link(5, Link(6)))))

s:
Adding to a Set Represented as an Ordered List

def add(s, v):
    """Add v to s, returning modified s."""

    >>> s = Link(1, Link(3, Link(5)))
    >>> add(s, 0)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 3)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 4)
    Link(0, Link(1, Link(3, Link(4, Link(5)))))
    >>> add(s, 6)
    Link(0, Link(1, Link(3, Link(4, Link(5, Link(6))))))

    assert s is not List.empty
Adding to a Set Represented as an Ordered List

```python
def add(s, v):
    """Add v to s, returning modified s."""

>>> s = Link(1, Link(3, Link(5)))
>>> add(s, 0)
Link(0, Link(1, Link(3, Link(5))))
>>> add(s, 3)
Link(0, Link(1, Link(3, Link(5))))
>>> add(s, 4)
Link(0, Link(1, Link(3, Link(4, Link(5)))))
>>> add(s, 6)
Link(0, Link(1, Link(3, Link(4, Link(5, Link(6))))))

assert s is not List.empty
if s.first > v:
    s.first, s.rest = ___________________________ , ___________________________
```

---

```
Link instance
  first: 0
  rest:  

Link instance
  first: 1
  rest:  

Link instance
  first: 3
  rest:  

Link instance
  first: 4
  rest:  

Link instance
  first: 5
  rest:  

Link instance
  first: 6
  rest:  
```

---
def add(s, v):
    """Add v to s, returning modified s."""
    s = Link(1, Link(3, Link(5)))
    add(s, 0)
    Link(0, Link(1, Link(3, Link(5))))
    add(s, 3)
    Link(0, Link(1, Link(3, Link(5))))
    add(s, 4)
    Link(0, Link(1, Link(3, Link(4, Link(5))))))
    add(s, 6)
    Link(0, Link(1, Link(3, Link(4, Link(5, Link(6))))))

    assert s is not List.empty
    if s.first > v:
        s.first, s.rest = __________________________ , _____________________________
    elif s.first < v and empty(s.rest):
        s.rest = _____________________________

assert s is not List.empty
if s.first > v:
    s.first, s.rest = __________________________ , _____________________________
elif s.first < v and empty(s.rest):
    s.rest = _____________________________
Adding to a Set Represented as an Ordered List

def add(s, v):
    """Add v to s, returning modified s."""
    >>> s = Link(1, Link(3, Link(5)))
    >>> add(s, 0)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 3)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 4)
    Link(0, Link(1, Link(3, Link(4, Link(5))))))
    >>> add(s, 6)
    Link(0, Link(1, Link(3, Link(4, Link(5, Link(6))))))
    assert s is not List.empty
    if s.first > v:
        s.first, s.rest = __________________________ , __________________________
    elif s.first < v and empty(s.rest):
        s.rest = __________________________
    elif s.first < v:
        __________________________
    return s
Adding to a Set Represented as an Ordered List

def add(s, v):
    """Add v to s, returning modified s."""

    >>> s = Link(1, Link(3, Link(5)))
    >>> add(s, 0)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 3)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 4)
    Link(0, Link(1, Link(3, Link(4, Link(5)))))
    >>> add(s, 6)
    Link(0, Link(1, Link(3, Link(4, Link(5, Link(6)))))
    
    assert s is not List.empty
    if s.first > v:
        s.first, s.rest = __________________________ , __________________________
    elif s.first < v and empty(s.rest):
        s.rest = ___________________________________________________________________
    elif s.first < v:
        __________________________________________________________________________
    return s
Adding to a Set Represented as an Ordered List

```python
def add(s, v):
    """Add v to s, returning modified s."""
    >>> s = Link(1, Link(3, Link(5)))
    >>> add(s, 0)
    Link(0, Link(1, Link(3, Link(5)))))
    >>> add(s, 3)
    Link(0, Link(1, Link(3, Link(5)))))
    >>> add(s, 4)
    Link(0, Link(1, Link(3, Link(4, Link(5)))))
    >>> add(s, 6)
    Link(0, Link(1, Link(3, Link(4, Link(5, Link(6)))))

    assert s is not List.empty
    if s.first > v:
        s.first, s.rest = __________________________ , _____________________________
    elif s.first < v and empty(s.rest):
        s.rest = ___________________________________________________________________
    elif s.first < v:
        __________________________________________________________________________
    return s
```
Adding to a Set Represented as an Ordered List

def add(s, v):
    """Add v to s, returning modified s."""

>>> s = Link(1, Link(3, Link(5)))
>>> add(s, 0)
Link(0, Link(1, Link(3, Link(5))))
>>> add(s, 3)
Link(0, Link(1, Link(3, Link(5))))
>>> add(s, 4)
Link(0, Link(1, Link(3, Link(4, Link(5)))))
>>> add(s, 6)
Link(0, Link(1, Link(3, Link(4, Link(5, Link(6)))))

assert s is not List.empty
if s.first > v:
    s.first, s.rest = __________________________, ___________________________
elif s.first < v and empty(s.rest):
    s.rest = ___________________________
else:
    Link(v)
return s
Adding to a Set Represented as an Ordered List

def add(s, v):
    """Add v to s, returning modified s."""
    >>> s = Link(1, Link(3, Link(5)))
    >>> add(s, 0)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 3)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 4)
    Link(0, Link(1, Link(3, Link(4, Link(5)))))
    >>> add(s, 6)
    Link(0, Link(1, Link(3, Link(4, Link(5), Link(6)))))
    
    assert s is not List.empty
    if s.first > v:
        s.first, s.rest = __________________________ , __________________________
    elif s.first < v and empty(s.rest):
        s.rest = __________________________________________________________________
    elif s.first < v:
        _______________________________________________________________________
        add(s.rest, v)
    return s
Adding to a Set Represented as an Ordered List

def add(s, v):
    """Add v to s, returning modified s."""

>>> s = Link(1, Link(3, Link(5)))
>>> add(s, 0)
Link(0, Link(1, Link(3, Link(5)))))
>>> add(s, 3)
Link(0, Link(1, Link(3, Link(5)))))
>>> add(s, 4)
Link(0, Link(1, Link(3, Link(4, Link(5)))))
>>> add(s, 6)
Link(0, Link(1, Link(3, Link(4, Link(5), Link(6))))

assert s is not List.empty
if s.first > v:
    s.first, s.rest = __________________________, __________________________
elif s.first < v and empty(s.rest):
    s.rest = __________________________________________________________________
elif s.first < v:
    __________________________________________________________________________
return s
Tree Class
Tree Abstraction (Review)
Recursive description (wooden trees):  
Relative description (family trees):
Tree Abstraction (Review)

Recursive description (wooden trees):
A tree has a root label and a list of branches

Relative description (family trees):
Recursive description (wooden trees):
A tree has a root label and a list of branches

Relative description (family trees):
Recursive description (wooden trees): A tree has a root label and a list of branches

Relative description (family trees):
Tree Abstraction (Review)

Recursive description (wooden trees):
A tree has a root label and a list of branches
Each branch is a tree

Relative description (family trees):
Root label
Branch
**Tree Abstraction (Review)**

Recursive description (wooden trees):
A tree has a root label and a list of branches
Each branch is a tree

Relative description (family trees):
Tree Abstraction (Review)

Recursive description (wooden trees):
A tree has a root label and a list of branches
Each branch is a tree
A tree with zero branches is called a leaf

Relative description (family trees):
Recursive description (wooden trees):
A tree has a root label and a list of branches.
Each branch is a tree.
A tree with zero branches is called a leaf.

Relative description (family trees):
Recursive description (wooden trees):
A tree has a root label and a list of branches
Each branch is a tree
A tree with zero branches is called a leaf
A tree starts at the root

Relative description (family trees):
Tree Abstraction (Review)

Recursive description (wooden trees):

A tree has a root label and a list of branches
Each branch is a tree
A tree with zero branches is called a leaf
A tree starts at the root

Relative description (family trees):
Recursive description (wooden trees):

A tree has a root label and a list of branches.
Each branch is a tree.
A tree with zero branches is called a leaf.
A tree starts at the root.

Relative description (family trees):
Tree Abstraction (Review)

Recursive description (wooden trees):
- A **tree** has a **root label** and a list of **branches**
- Each **branch** is a **tree**
- A **tree** with zero **branches** is called a **leaf**
- A **tree** starts at the **root**

Relative description (family trees):
- Each location in a tree is called a **node**
Recursive description (wooden trees):
A tree has a root label and a list of branches
Each branch is a tree
A tree with zero branches is called a leaf
A tree starts at the root

Relative description (family trees):
Each location in a tree is called a node
Each node has a label that can be any value
**Recursive description (wooden trees):**

A tree has a root label and a list of branches.

Each branch is a tree.

A tree with zero branches is called a leaf.

A tree starts at the root.

**Relative description (family trees):**

Each location in a tree is called a node.

Each node has a label that can be any value.
Tree Abstraction (Review)

Recursive description (wooden trees):
A tree has a root label and a list of branches
Each branch is a tree
A tree with zero branches is called a leaf
A tree starts at the root

Relative description (family trees):
Each location in a tree is called a node
Each node has a label that can be any value
One node can be the parent/child of another
Tree Abstraction (Review)

Recursive description (wooden trees):
A tree has a root label and a list of branches
Each branch is a tree
A tree with zero branches is called a leaf
A tree starts at the root

Relative description (family trees):
Each location in a tree is called a node
Each node has a label that can be any value
One node can be the parent/child of another
The top node is the root node
Tree Abstraction (Review)

**Recursive description (wooden trees):**
- A tree has a root label and a list of branches.
- Each branch is a tree.
- A tree with zero branches is called a leaf.
- A tree starts at the root.

**Relative description (family trees):**
- Each location in a tree is called a node.
- Each node has a label that can be any value.
- One node can be the parent/child of another.
- The top node is the root node.
Recursive description (wooden trees):
A tree has a root label and a list of branches
Each branch is a tree
A tree with zero branches is called a leaf
A tree starts at the root

People often refer to labels by their locations: "each parent is the sum of its children"
Tree Abstraction (Review)

Recursive description (wooden trees):
- A tree has a root label and a list of branches.
- Each branch is a tree.
- A tree with zero branches is called a leaf.
- A tree starts at the root.

Relative description (family trees):
- Each location in a tree is called a node.
- Each node has a label that can be any value.
- One node can be the parent/child of another.
- The top node is the root node.

People often refer to labels by their locations: "each parent is the sum of its children"
Tree Class

A Tree has a label and a list of branches; each branch is a Tree.
Tree Class

A Tree has a label and a list of branches; each branch is a Tree

```python
class Tree:
```
A Tree has a label and a list of branches; each branch is a Tree

class Tree:
    def __init__(self, label, branches=[]):
Tree Class

A Tree has a label and a list of branches; each branch is a Tree

class Tree:
    def __init__(self, label, branches=[]):
        self.label = label
A Tree has a label and a list of branches; each branch is a Tree

```python
class Tree:
    def __init__(self, label, branches=[]):
        self.label = label
        for branch in branches:
            assert isinstance(branch, Tree)
```
A Tree has a label and a list of branches; each branch is a Tree

class Tree:
    def __init__(self, label, branches=[]):
        self.label = label
        for branch in branches:
            assert isinstance(branch, Tree)
        self.branches = list(branches)
Tree Class

A Tree has a label and a list of branches; each branch is a Tree

class Tree:
    def __init__(self, label, branches=[]):
        self.label = label
        for branch in branches:
            assert isinstance(branch, Tree)
        self.branches = list(branches)

def tree(label, branches=[]):
    for branch in branches:
        assert is_tree(branch)
    return [label] + list(branches)

def label(tree):
    return tree[0]

def branches(tree):
    return tree[1:]
A Tree has a label and a list of branches; each branch is a Tree

class Tree:
    def __init__(self, label, branches=[]):
        self.label = label
        for branch in branches:
            assert isinstance(branch, Tree)
        self.branches = list(branches)

def fib_tree(n):
    if n == 0 or n == 1:
        return Tree(n)
    else:
        left = fib_tree(n-2)
        right = fib_tree(n-1)
        fib_n = left.label + right.label
        return Tree(fib_n, [left, right])

def tree(label, branches=[]):
    for branch in branches:
        assert is_tree(branch)
    return [label] + list(branches)

def label(tree):
    return tree[0]

def branches(tree):
    return tree[1:]
Tree Class

A Tree has a label and a list of branches; each branch is a Tree.

class Tree:
    def __init__(self, label, branches=[]):
        self.label = label
        for branch in branches:
            assert isinstance(branch, Tree)
        self.branches = list(branches)

def fib_tree(n):
    if n == 0 or n == 1:
        return Tree(n)
    else:
        left = fib_tree(n-2)
        right = fib_tree(n-1)
        fib_n = left.label + right.label
        return Tree(fib_n, [left, right])

def tree(label, branches=[]):
    for branch in branches:
        assert is_tree(branch)
    return [label] + list(branches)

def label(tree):
    return tree[0]

def branches(tree):
    return tree[1:]
Tree Class

A Tree has a label and a list of branches; each branch is a Tree.

class Tree:
    def __init__(self, label, branches=[]):
        self.label = label
        for branch in branches:
            assert isinstance(branch, Tree)
        self.branches = list(branches)

def fib_tree(n):
    if n == 0 or n == 1:
        return Tree(n)
    else:
        left = fib_tree(n-2)
        right = fib_tree(n-1)
        fib_n = left.label + right.label
        return Tree(fib_n, [left, right])

(Demo)
Tree Mutation
Example: Pruning Trees

Removing subtrees from a tree is called *pruning*

Prune branches before recursive processing
Example: Pruning Trees

Removing subtrees from a tree is called *pruning*.

Prune branches before recursive processing.
Example: Pruning Trees

Removing subtrees from a tree is called *pruning*.

Prune branches before recursive processing.
Example: Pruning Trees

Removing subtrees from a tree is called *pruning*.

Prune branches before recursive processing.

```python
def prune(t, n):
    """Prune all sub-trees whose label is n."""
    t.branches = [______________ for b in t.branches if ________________]
    for b in t.branches:
        prune(______________________________, ________________)
```
Example: Pruning Trees

Removing subtrees from a tree is called pruning.

Prune branches before recursive processing.

```
def prune(t, n):
    """Prune all sub-trees whose label is n.""
    t.branches = [b for b in t.branches if b.label != n]
    for b in t.branches:
        prune(b.value, b.label)
```
Example: Pruning Trees

Removing subtrees from a tree is called *pruning*.

Prune branches before recursive processing.

```python
def prune(t, n):
    """Prune all sub-trees whose label is n."""
    t.branches = [b for b in t.branches if b.label != n]
    for b in t.branches:
        prune(b, n)
```