## Data Examples

## Announcements

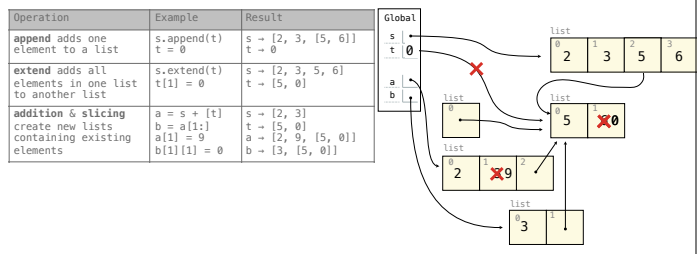
## Lists

## Lists in Environment Diagrams

**Assume that before each example below we execute:**

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

| Operation | Example | Result |
|---|---|---|
| **append** adds one element to a list | s.append(t)<br>t = 0 | s → [2, 3, [5, 6]]<br>t → 0 |
| **extend** adds all elements in one list to another list | s.extend(t)<br>t[1] = 0 | s → [2, 3, 5, 6]<br>t → [5, 0] |
| **addition** & **slicing** create new lists containing existing elements | a = s + [t]<br>b = a[1:]<br>a[1] = 9<br>b[1][1] = 0 | s → [2, 3]<br>t → [5, 0]<br>a → [2, 9, [5, 0]]<br>b → [3, [5, 0]] |

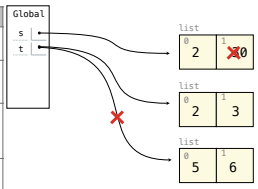



## Lists in Environment Diagrams

**Assume that before each example below we execute:**

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

| Operation | Example | Result |
|---|---|---|
| **append** adds one element to a list | s.append(t)<br>t = 0 | s → [2, 3, [5, 6]]<br>t → 0 |
| **extend** adds all elements in one list to another list | s.extend(t)<br>t[1] = 0 | s → [2, 3, 5, 6]<br>t → [5, 0] |
| **addition** & **slicing** create new lists containing existing elements | a = s + [t]<br>b = a[1:]<br>a[1] = 9<br>b[1][1] = 0 | s → [2, 3]<br>t → [5, 0]<br>a → [2, 9, [5, 0]]<br>b → [3, [5, 0]] |
| The **list** function also creates a new list containing existing elements | t = list(s)<br>s[1] = 0 | s → [2, 0]<br>t → [2, 3] |

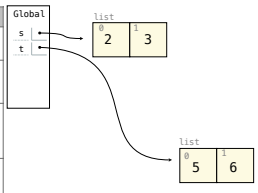



## Lists in Environment Diagrams

**Assume that before each example below we execute:**

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

| Operation | Example | Result |
|---|---|---|
| **append** adds one element to a list | s.append(t)<br>t = 0 | s → [2, 3, [5, 6]]<br>t → 0 |
| **extend** adds all elements in one list to another list | s.extend(t)<br>t[1] = 0 | s → [2, 3, 5, 6]<br>t → [5, 0] |
| **addition** & **slicing** create new lists containing existing elements | a = s + [t]<br>b = a[1:]<br>a[1] = 9<br>b[1][1] = 0 | s → [2, 3]<br>t → [5, 0]<br>a → [2, 9, [5, 0]]<br>b → [3, [5, 0]] |
| The **list** function also creates a new list containing existing elements | t = list(s)<br>s[1] = 0 | s → [2, 0]<br>t → [2, 3] |
| **slice assignment** replaces a slice with new values | s[0:0] = t<br>s[3:] = t<br>t[1] = 0 | |

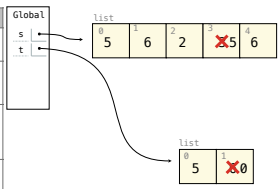



## Lists in Environment Diagrams

**Assume that before each example below we execute:**

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

| Operation | Example | Result |
|---|---|---|
| **append** adds one element to a list | s.append(t)<br>t = 0 | s → [2, 3, [5, 6]]<br>t → 0 |
| **extend** adds all elements in one list to another list | s.extend(t)<br>t[1] = 0 | s → [2, 3, 5, 6]<br>t → [5, 0] |
| **addition** & **slicing** create new lists containing existing elements | a = s + [t]<br>b = a[1:]<br>a[1] = 9<br>b[1][1] = 0 | s → [2, 3]<br>t → [5, 0]<br>a → [2, 9, [5, 0]]<br>b → [3, [5, 0]] |
| The **list** function also creates a new list containing existing elements | t = list(s)<br>s[1] = 0 | s → [2, 0]<br>t → [2, 3] |
| **slice assignment** replaces a slice with new values | s[0:0] = t<br>s[3:] = t<br>t[1] = 0 | s → [5, 6, 2, 5, 6]<br>t → [5, 0] |




## Lists in Environment Diagrams

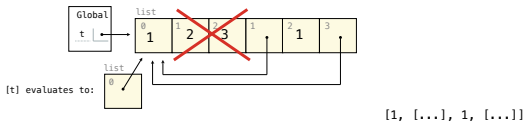
**Assume that before each example below we execute:**

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

| Operation | Example | Result |
|---|---|---|
| **pop** removes & returns the last element | t = s.pop() | s → [2]<br>t → 3 |
| **remove** removes the first element equal to the argument | t.extend(t)<br>t.remove(5) | s → [2, 3]<br>t → [6, 5, 6] |
| **slice assignment** can remove elements from a list by assigning [] to a slice. | s[:1] = []<br>t[0:2] = [] | s → [3]<br>t → [] |

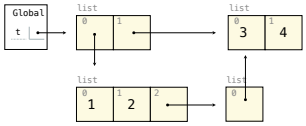
## Lists in Lists in Lists in Environment Diagrams

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




```
[1, [...], 1, [...]]
```

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




```
[[1, 2, [[3, 4]]], [3, 4]]
```

## Objects

## Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'

jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'
```

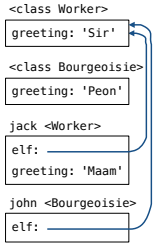
```
>>> Worker().work()
'Sir, I work'

>>> jack
Peon

>>> jack.work()
'Maam, I work'

>>> john.work()
Peon, I work
'I gather wealth'

>>> john.elf.work(john)
'Peon, I work'
```



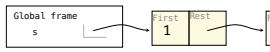

## Mutable Linked Lists

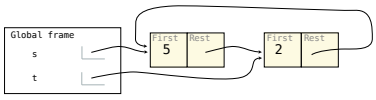
## Recursive Lists Can Change

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

```
>>> 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
```

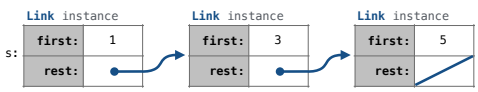



Note: The actual environment diagram is much more complicated.

## Linked List Mutation Example

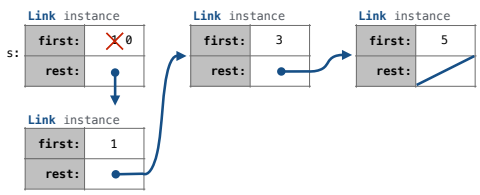
## Adding to an Ordered List




```
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)
```

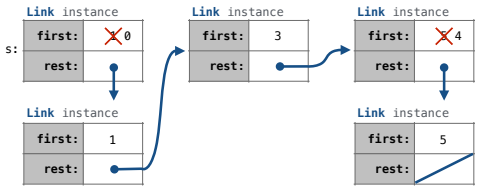
## Adding to an Ordered List




```
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)
```

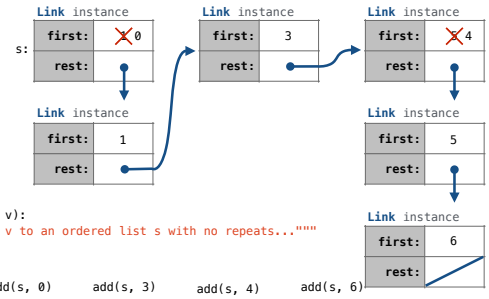
## Adding to an Ordered List




```
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)

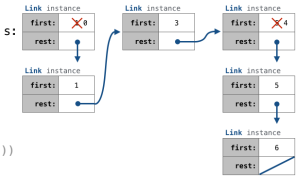
## Adding to an Ordered List




```
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



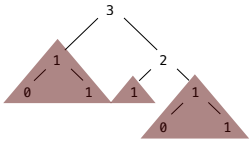

```
def add(s, v):
    """Add v to s, returning modified s."""
    >>> s = Link(1, Link(3, Link(5)))
    >>> add(s, 0)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 3)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 4)
    Link(0, Link(1, Link(3, Link(4, Link(5)))))
    >>> add(s, 6)
    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 = ____v____ , ____Link(s.first, s.rest)____
    elif s.first < v and empty(s.rest):
        s.rest = ____Link(v)____
    elif s.first < v:
        ____add(s.rest, v)____
    return s
```

## Tree Mutation

## 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____, ____n____)
```