CS 61A Lecture 11

Friday, September 27

Announcements

Announcements

- Midterm 1 has been graded...

Announcements

- Midterm 1 has been graded...
  - Many of you did very well!

Announcements

- Midterm 1 has been graded...
  - Many of you did very well!
  - High scores on homework and projects balance out exam scores.

Announcements

- Midterm 1 has been graded...
  - Many of you did very well!
  - High scores on homework and projects balance out exam scores.
  - Typically, more than 75% of students receive A's & B's in 61A.

Announcements

- Midterm 1 has been graded...
  - Many of you did very well!
  - High scores on homework and projects balance out exam scores.
  - Typically, more than 75% of students receive A's & B's in 61A.
  - If you are falling behind, come to class (lecture, discussion, lab, & office hours)!

Announcements

- Midterm 1 has been graded...
  - Many of you did very well!
  - High scores on homework and projects balance out exam scores.
  - Typically, more than 75% of students receive A's & B's in 61A.
  - If you are falling behind, come to class (lecture, discussion, lab, & office hours)!
- Homework 3 due Tuesday 10/1 @ 11:59pm

Announcements

- Midterm 1 has been graded...
  - Many of you did very well!
  - High scores on homework and projects balance out exam scores.
  - Typically, more than 75% of students receive A's & B's in 61A.
  - If you are falling behind, come to class (lecture, discussion, lab, & office hours)!
- Homework 3 due Tuesday 10/1 @ 11:59pm
- Optional Hog Contest due Thursday 10/3 @ 11:59pm

Sequences

The Sequence Abstraction

The Sequence Abstraction

red, orange, yellow, green, blue, indigo, violet.

The Sequence Abstraction

red, orange, yellow, green, blue, indigo, violet.

There isn't just one sequence class or abstract data type (in Python or in general).

The Sequence Abstraction

red, orange, yellow, green, blue, indigo, violet.

There isn't just one sequence class or abstract data type (in Python or in general).

The sequence abstraction is a collection of behaviors:

The Sequence Abstraction

red, orange, yellow, green, blue, indigo, violet.

There isn't just one sequence class or abstract data type (in Python or in general).

The sequence abstraction is a collection of behaviors:

Length. A sequence has a finite length.

Element selection. A sequence has an element corresponding to any non-negative integer index less than its length, starting at 0 for the first element.

The Sequence Abstraction

red, orange, yellow, green, blue, indigo, violet.

0 , 1 , 2 , 3 , 4 , 5 , 6 .

There isn't just one sequence class or abstract data type (in Python or in general).

The sequence abstraction is a collection of behaviors:

Length. A sequence has a finite length.

Element selection. A sequence has an element corresponding to any non-negative integer index less than its length, starting at 0 for the first element.

The Sequence Abstraction

red, orange, yellow, green, blue, indigo, violet.

0 , 1 , 2 , 3 , 4 , 5 , 6 .

There isn't just one sequence class or abstract data type (in Python or in general).

The sequence abstraction is a collection of behaviors:

> **Length.** A sequence has a finite length.
>
> **Element selection.** A sequence has an element corresponding to any non-negative integer index less than its length, starting at 0 for the first element.

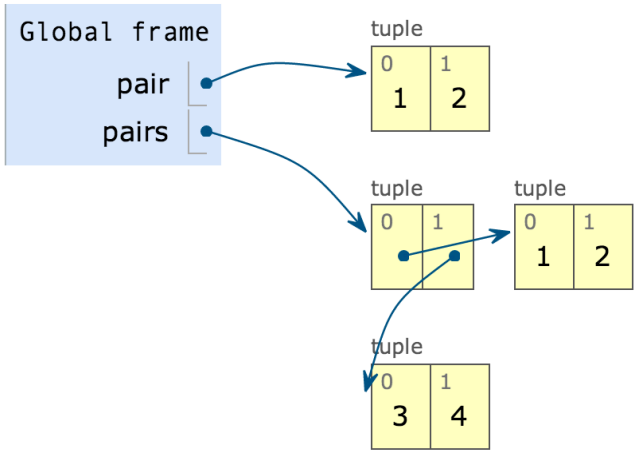
There is built-in syntax associated with this behavior, or we can use functions.

A tuple is a kind of built-in sequence (demo)

Box-and-Pointer Notation

Box-and-Pointer Notation

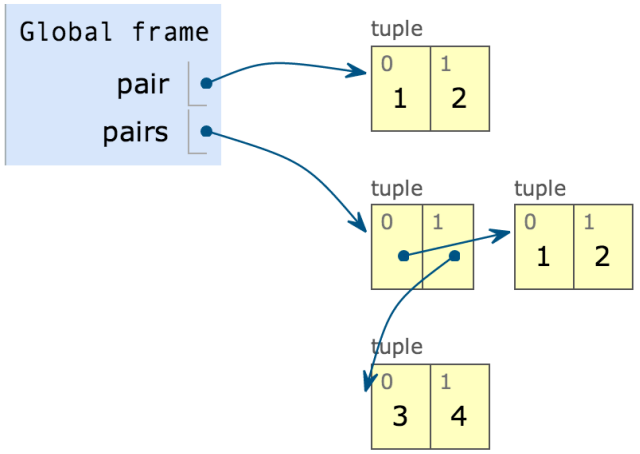
```
1  pair = (1, 2)
2  pairs = ((1, 2), (3, 4))
```



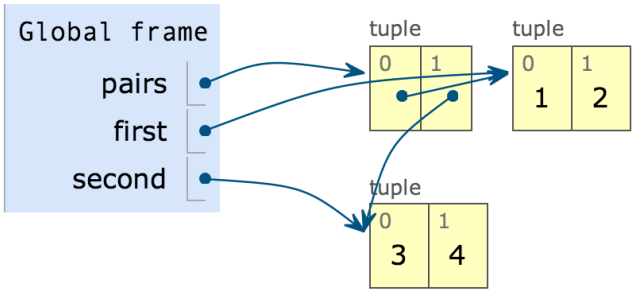
Examples: http://goo.gl/nyQfWG http://goo.gl/ovgSWT

Box-and-Pointer Notation

```
1  pair = (1, 2)
2  pairs = ((1, 2), (3, 4))
```




```
1  pairs = ((1, 2), (3, 4))
2  first, second = pairs
```



Examples: http://goo.gl/nyQfWG http://goo.gl/ovgSWT

The Closure Property of Data Types

The Closure Property of Data Types

- A method for combining data values satisfies the *closure property* if:

The Closure Property of Data Types

- A method for combining data values satisfies the *closure property* if:
- The result of combination can itself be combined using the same method.

The Closure Property of Data Types

- A method for combining data values satisfies the *closure property* if:
- The result of combination can itself be combined using the same method.
- Closure is the key to power in any means of combination because it permits us to create hierarchical structures.

The Closure Property of Data Types

- A method for combining data values satisfies the *closure property* if:
- The result of combination can itself be combined using the same method.
- Closure is the key to power in any means of combination because it permits us to create hierarchical structures.
- Hierarchical structures are made up of parts, which themselves are made up of parts, and so on.

The Closure Property of Data Types

- A method for combining data values satisfies the *closure property* if:
- The result of combination can itself be combined using the same method.
- Closure is the key to power in any means of combination because it permits us to create hierarchical structures.
- Hierarchical structures are made up of parts, which themselves are made up of parts, and so on.

Tuples can contain tuples as elements

Recursive Lists

Recursive Lists

Recursive Lists

Constructor:

```
def rlist(first, rest):
    """Return a recursive list from its first element and the rest."""
```

Recursive Lists

Constructor:

```
def rlist(first, rest):
    """Return a recursive list from its first element and the rest."""
```

Selectors:

```
def first(s):
    """Return the first element of a recursive list s."""


def rest(s):
    """Return the rest of the elements of a recursive list s."""
```

Recursive Lists

Constructor:

```
def rlist(first, rest):
    """Return a recursive list from its first element and the rest."""
```

Selectors:

```
def first(s):
    """Return the first element of a recursive list s."""
```

```
def rest(s):
    """Return the rest of the elements of a recursive list s."""
```

Behavior condition(s):

Recursive Lists

Constructor:

```
def rlist(first, rest):
    """Return a recursive list from its first element and the rest."""
```

Selectors:

```
def first(s):
    """Return the first element of a recursive list s."""


def rest(s):
    """Return the rest of the elements of a recursive list s."""
```

Behavior condition(s):

> If a recursive list `s` is constructed from a first element `f` and a recursive list `r`, then

Recursive Lists

Constructor:

```
def rlist(first, rest):
    """Return a recursive list from its first element and the rest."""
```

Selectors:

```
def first(s):
    """Return the first element of a recursive list s."""


def rest(s):
    """Return the rest of the elements of a recursive list s."""
```

Behavior condition(s):

If a recursive list `s` is constructed from a first element `f` and a recursive list `r`, then

- `first(s)` returns `f`, and

Recursive Lists

Constructor:

```
def rlist(first, rest):
    """Return a recursive list from its first element and the rest."""
```

Selectors:

```
def first(s):
    """Return the first element of a recursive list s."""


def rest(s):
    """Return the rest of the elements of a recursive list s."""
```

Behavior condition(s):

If a recursive list `s` is constructed from a first element `f` and a recursive list `r`, then

- `first(s)` returns `f`, and
- `rest(s)` returns `r`, which is a recursive list.

Implementing Recursive Lists with Pairs

```
We can implement recursive lists as pairs.  We'll use two-element tuples to encode pairs.
```

Implementing Recursive Lists with Pairs

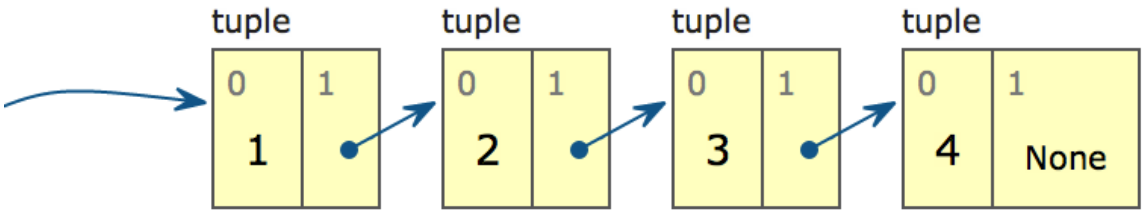
We can implement recursive lists as pairs. We'll use two-element tuples to encode pairs.

1 , 2 , 3 , 4

Implementing Recursive Lists with Pairs

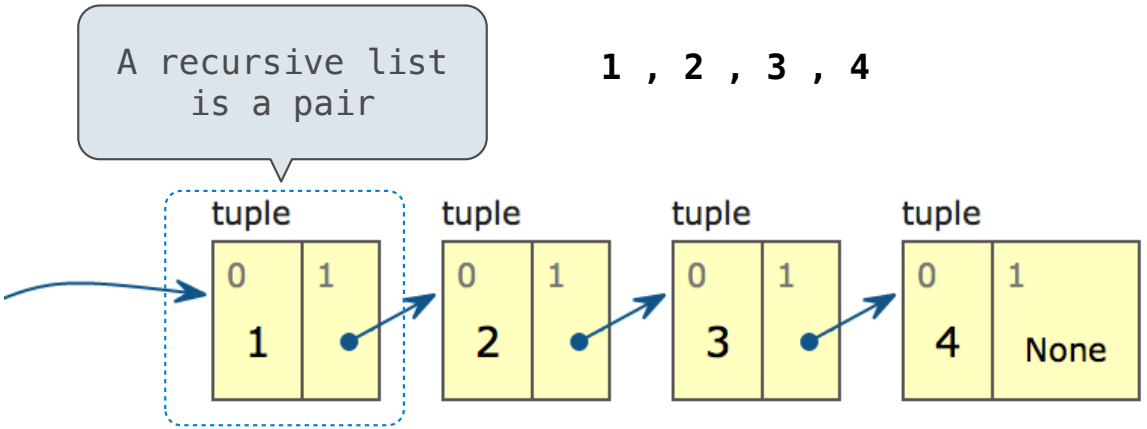
We can implement recursive lists as pairs. We'll use two-element tuples to encode pairs.

1 , 2 , 3 , 4



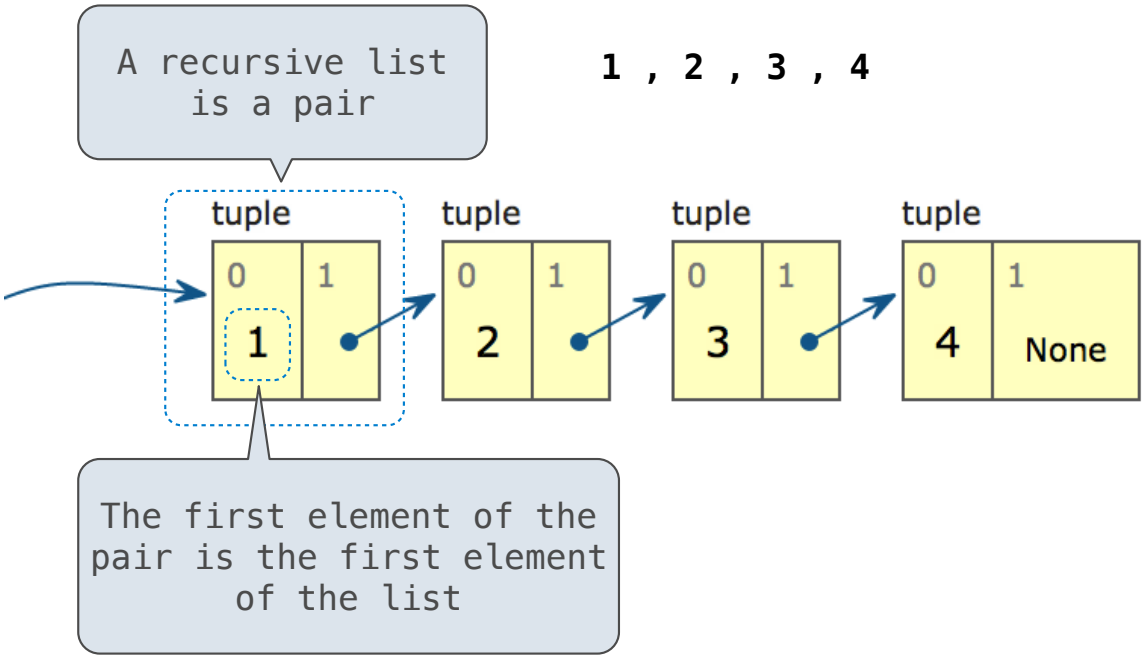
Implementing Recursive Lists with Pairs

We can implement recursive lists as pairs. We'll use two-element tuples to encode pairs.



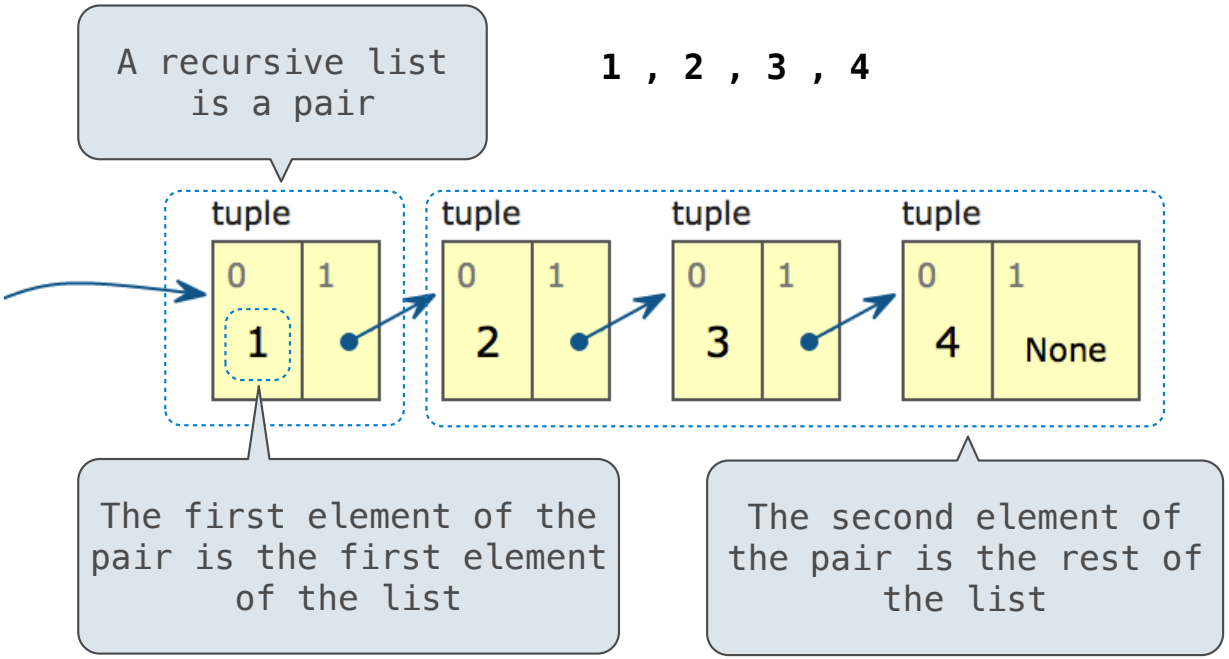
Implementing Recursive Lists with Pairs

We can implement recursive lists as pairs. We'll use two-element tuples to encode pairs.



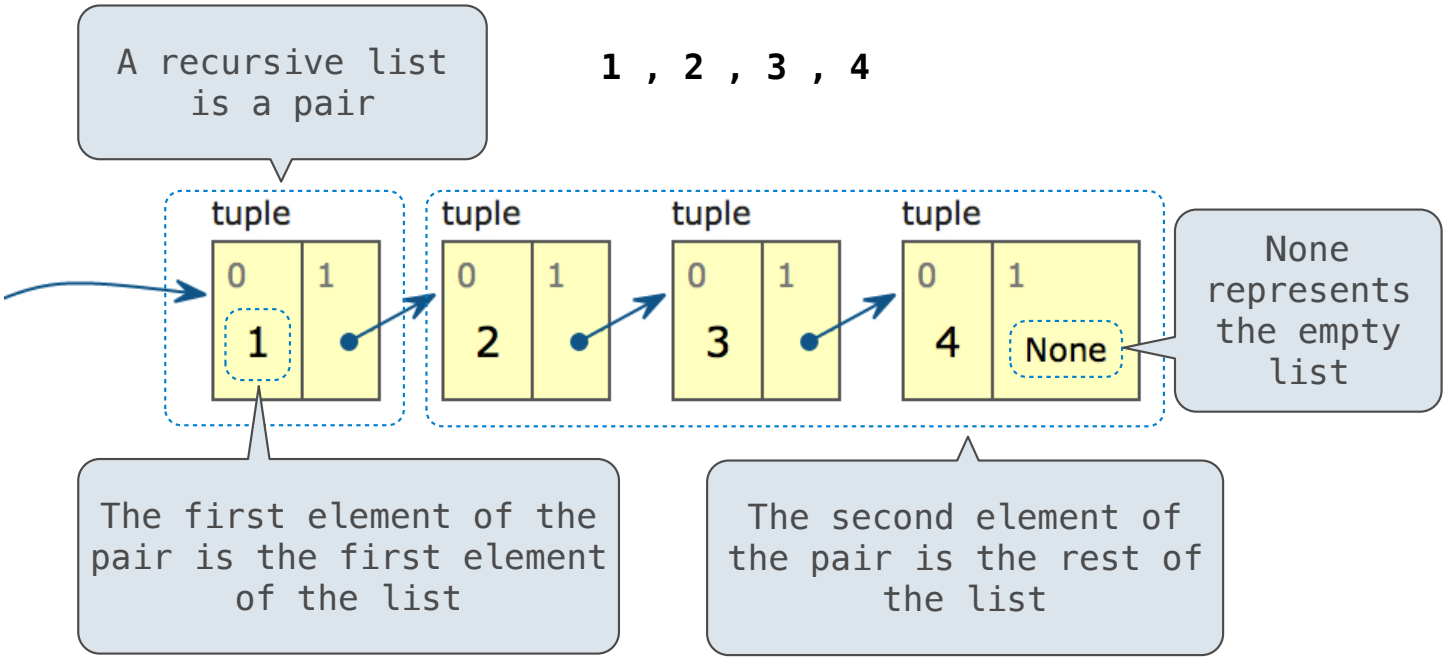
Implementing Recursive Lists with Pairs

We can implement recursive lists as pairs. We'll use two-element tuples to encode pairs.



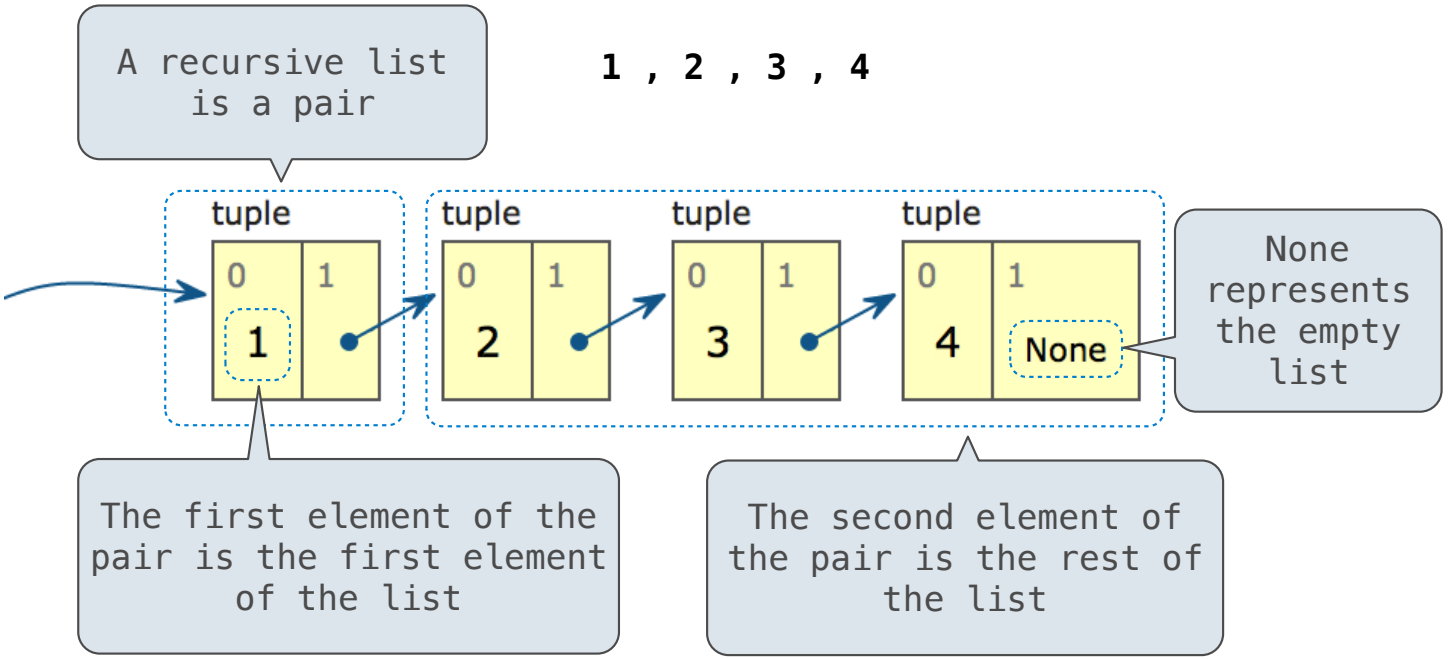
Implementing Recursive Lists with Pairs

We can implement recursive lists as pairs. We'll use two-element tuples to encode pairs.



Implementing Recursive Lists with Pairs

We can implement recursive lists as pairs. We'll use two-element tuples to encode pairs.



(Demo)

Sequence Abstraction Implementation

Implementing the Sequence Abstraction

Implementing the Sequence Abstraction

Length. A sequence has a finite length.

Element selection. A sequence has an element corresponding to any non-negative integer index less than its length, starting at 0 for the first element.

Implementing the Sequence Abstraction

```
def len_rlist(s):
    """Return the length of recursive list s."""
    length = 0
    while s != empty_rlist:
        s, length = rest(s), length + 1
    return length
```

Length. A sequence has a finite length.

Element selection. A sequence has an element corresponding to any non-negative integer index less than its length, starting at 0 for the first element.

Implementing the Sequence Abstraction

```
def len_rlist(s):
    """Return the length of recursive list s."""
    length = 0
    while s != empty_rlist:
        s, length = rest(s), length + 1
    return length


def getitem_rlist(s, i):
    """Return the element at index i of recursive list s."""
    while i > 0:
        s, i = rest(s), i - 1
    return first(s)
```

Length. A sequence has a finite length.

Element selection. A sequence has an element corresponding to any non-negative integer index less than its length, starting at 0 for the first element.

Implementing the Sequence Abstraction

```
def len_rlist(s):
    """Return the length of recursive list s."""
    length = 0
    while s != empty_rlist:
        s, length = rest(s), length + 1
    return length
```

(Demo)

```
def getitem_rlist(s, i):
    """Return the element at index i of recursive list s."""
    while i > 0:
        s, i = rest(s), i - 1
    return first(s)
```

Length. A sequence has a finite length.

Element selection. A sequence has an element corresponding to any non-negative integer index less than its length, starting at 0 for the first element.

Recursive implementations

(Demo)