CS 61A Lecture 9

Friday, September 14

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 type (in Python or in general)`

The Sequence Abstraction

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

There isn't just one sequence type (in Python or in general)

This abstraction is a collection of behaviors:

The Sequence Abstraction

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

There isn't just one sequence type (in Python or in general)

This 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 type (in Python or in general)

This 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 type (in Python or in general)

This abstraction is a collection of behaviors:

Length. A sequence has a finite length.

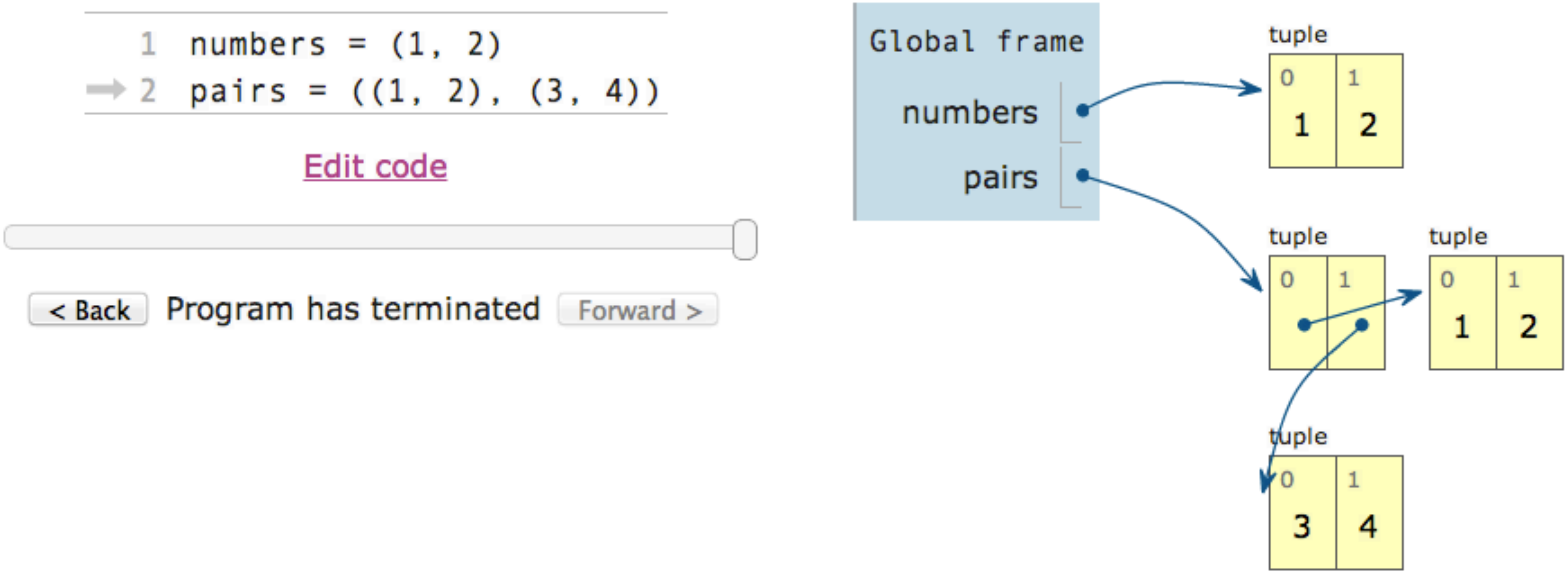
Element selection. A sequence has an element corresponding to any non-negative integer index less than its length, starting at 0 for the first element.

The sequence abstraction is shared among several types.

Tuples are Sequences

(Demo)

Box-and-Pointer Notation



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

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

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.

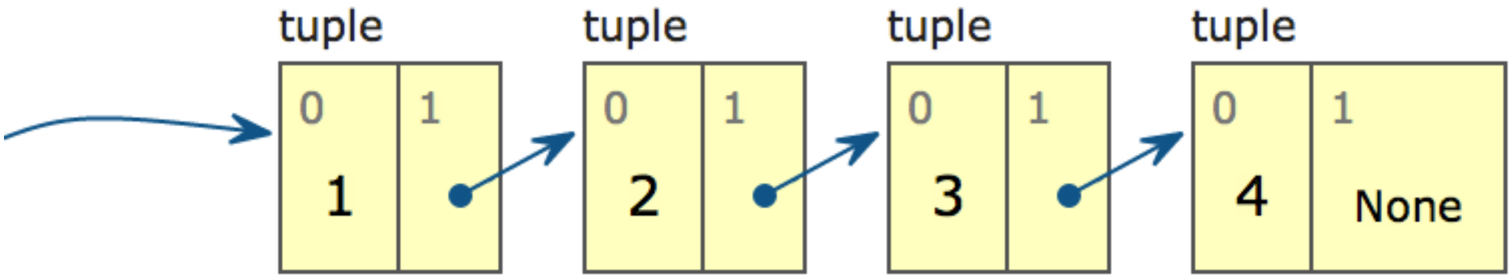
Implementing Recursive Lists with Pairs

Implementing Recursive Lists with Pairs

```
1 , 2 , 3 , 4
```

Implementing Recursive Lists with Pairs

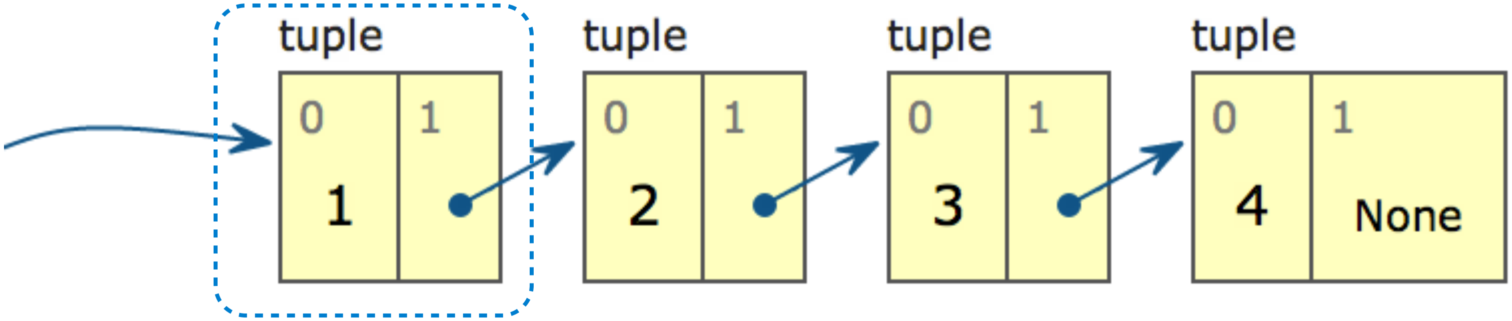
1 , 2 , 3 , 4



Implementing Recursive Lists with Pairs

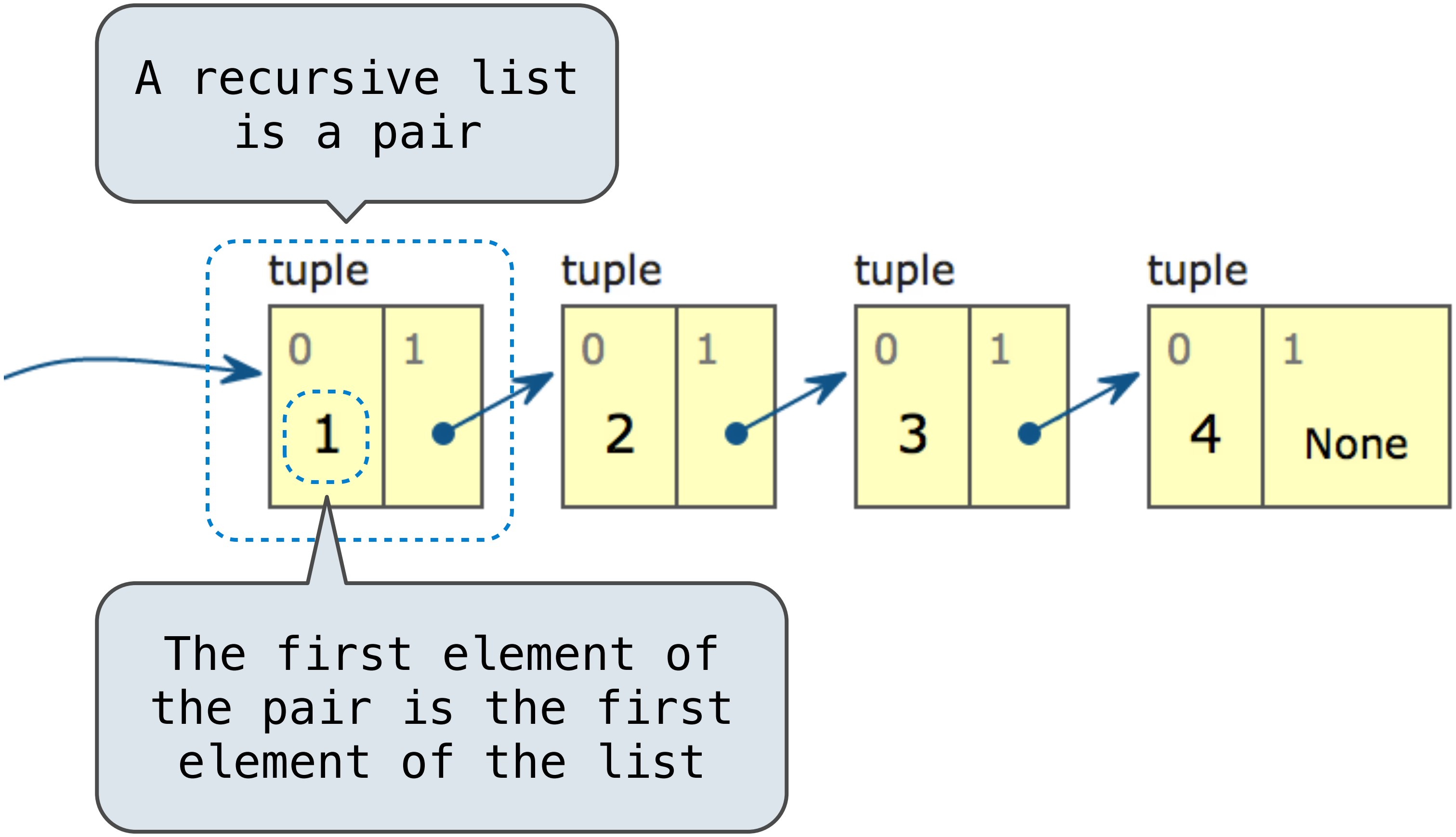
1 , 2 , 3 , 4

A recursive list is a pair



Implementing Recursive Lists with Pairs

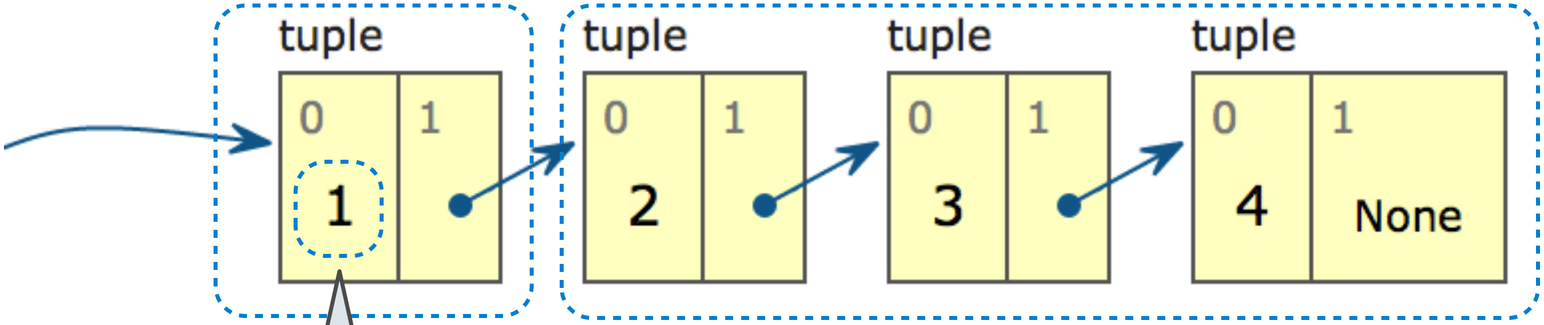
1 , 2 , 3 , 4


A recursive list is a pair
tuple
0
1
1
tuple
0
1
2
tuple
0
1
3
tuple
0
1
4
None
The first element of the pair is the first element of the list

Implementing Recursive Lists with Pairs

1 , 2 , 3 , 4

A recursive list is a pair



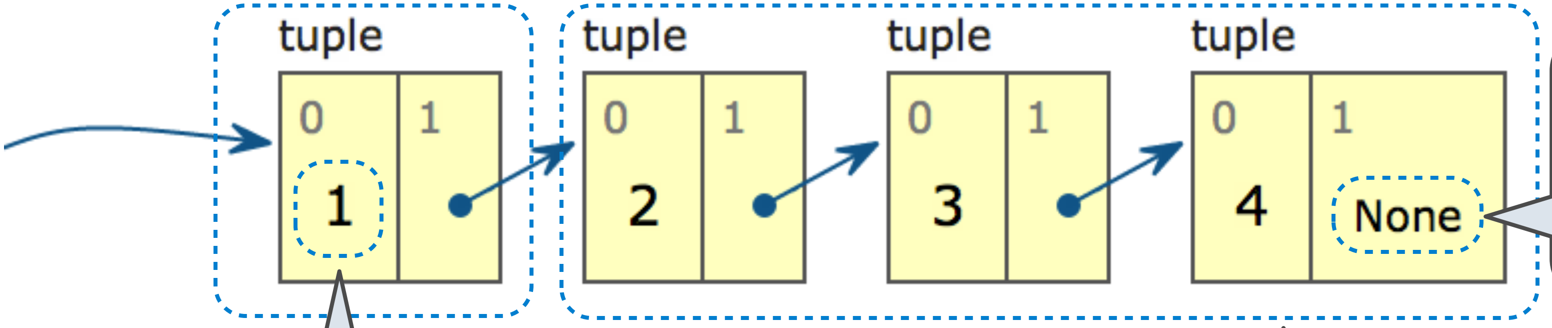
The first element of the pair is the first element of the list

The second element of the pair is the rest of the list

Implementing Recursive Lists with Pairs

1 , 2 , 3 , 4

A recursive list is a pair



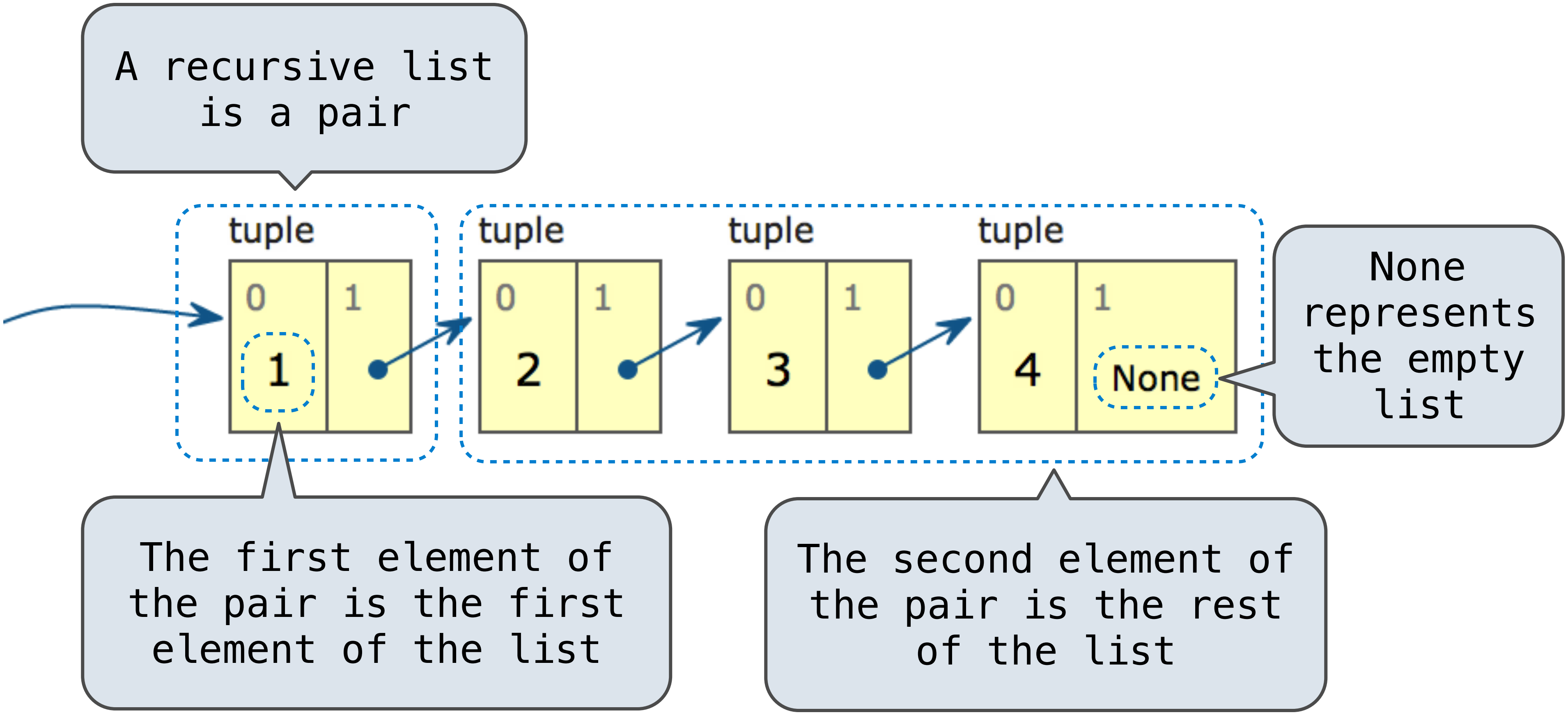
None represents the empty list

The first element of the pair is the first element of the list

The second element of the pair is the rest of the list

Implementing Recursive Lists with Pairs

1 , 2 , 3 , 4



A recursive list is a pair

None represents the empty list

The first element of the pair is the first element of the list

The second element of the pair is the rest of the list

(Demo)

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
```

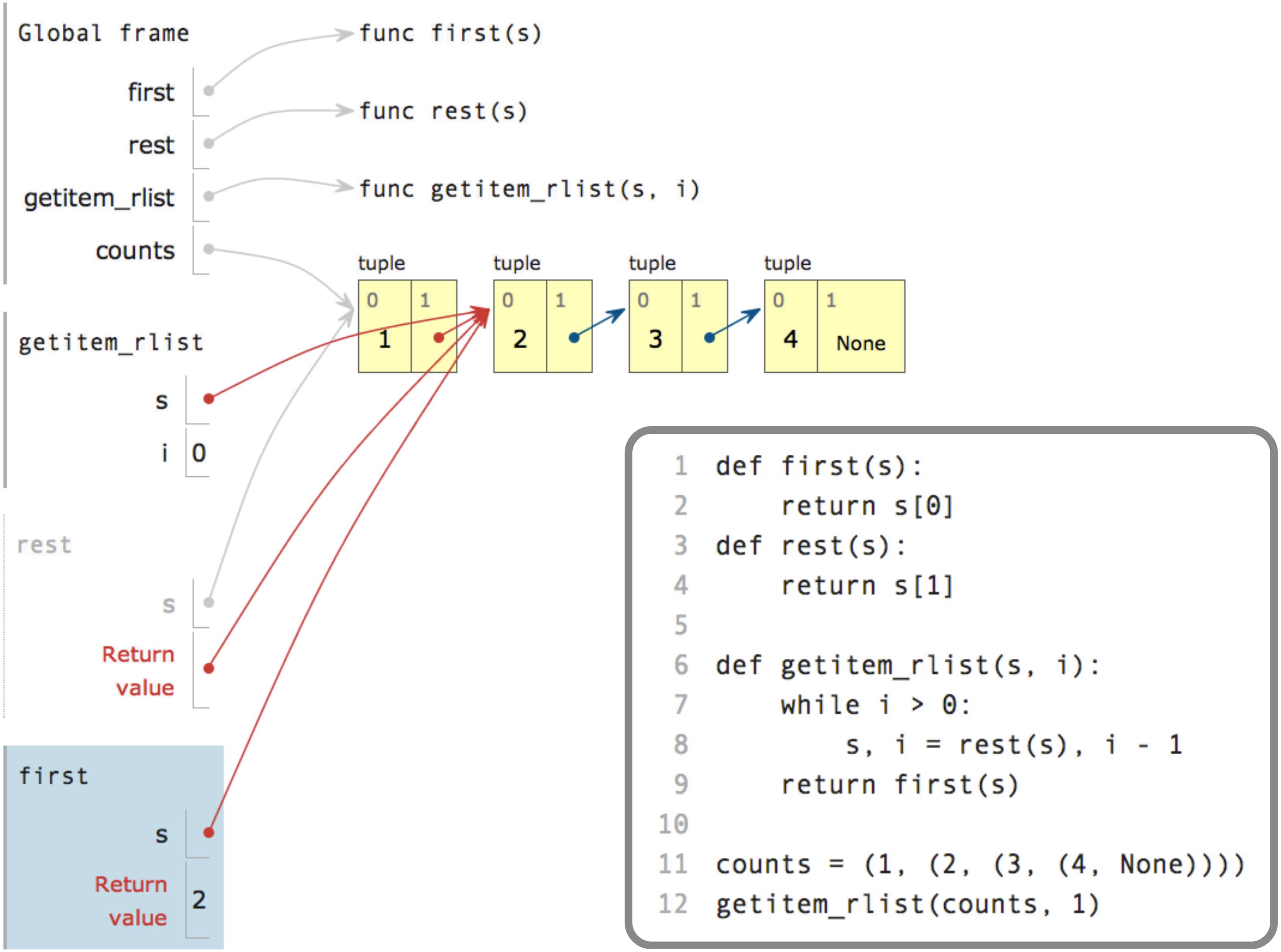
(Demo)

```
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.

Environment Diagram for getitem_rlist



```
def first(s):
    return s[0]
def rest(s):
    return s[1]

def getitem_rlist(s, i):
    while i > 0:
        s, i = rest(s), i - 1
    return first(s)

counts = (1, (2, (3, (4, None))))
getitem_rlist(counts, 1)
```

Sequence Iteration

Not on
Midterm 1

(Demo)

Sequence Iteration

Not on Midterm 1

(Demo)

```
def count(s, value):
    total = 0
    for elem in s:
        if elem == value:
            total = total + 1
    return total
```

Name bound in the first frame of the current environment

For Statement Execution Procedure

Not on
Midterm 1

For Statement Execution Procedure

Not on
Midterm 1

```
for <name> in <expression>:
    <suite>
```

For Statement Execution Procedure

Not on Midterm 1

```
for <name> in <expression>:
    <suite>
```

1. Evaluate the header <expression>, which must yield an iterable value.

For Statement Execution Procedure

Not on Midterm 1

```
for <name> in <expression>:
    <suite>
```

1. Evaluate the header <expression>, which must yield an iterable value.

2. For each element in that sequence, in order:

For Statement Execution Procedure

Not on Midterm 1

```
for <name> in <expression>:
    <suite>
```

1. Evaluate the header <expression>, which must yield an iterable value.

2. For each element in that sequence, in order:

   A. Bind <name> to that element in the local environment.

For Statement Execution Procedure

Not on Midterm 1

```
for <name> in <expression>:
    <suite>
```

1. Evaluate the header <expression>, which must yield an iterable value.

2. For each element in that sequence, in order:

   A. Bind <name> to that element in the local environment.

   B. Execute the <suite>.

Sequence Unpacking in For Statements

Not on Midterm 1

Sequence Unpacking in For Statements

Not on
Midterm 1

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

>>> same_count = 0
```

Sequence Unpacking in For Statements

Not on Midterm 1

A sequence of fixed-length sequences

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

>>> same_count = 0
```

Sequence Unpacking in For Statements

Not on Midterm 1

A sequence of fixed-length sequences

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

>>> same_count = 0




>>> for x, y in pairs:
        if x == y:
            same_count = same_count + 1

>>> same_count
2
```

Not on Midterm 1

Sequence Unpacking in For Statements

A sequence of fixed-length sequences

```
>>> pairs = ((1, 2), (2, 2), (2, 3), (4, 4))
>>> same_count = 0
```

A name for each element in a fixed-length sequence

```
>>> for x, y in pairs:
        if x == y:
            same_count = same_count + 1

>>> same_count
2
```

Sequence Unpacking in For Statements

Not on Midterm 1

A sequence of fixed-length sequences

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

>>> same_count = 0
```

A name for each element in a fixed-length sequence

Each name is bound to a value, as in multiple assignment

```
>>> for x, y in pairs:
        if x == y:
            same_count = same_count + 1

>>> same_count
2
```

The Range Type

Not on Midterm 1

A range is a sequence of consecutive integers.*

The Range Type

Not on Midterm 1

A range is a sequence of consecutive integers.*

* Ranges can actually represent more general integer sequences.

The Range Type

Not on Midterm 1

A range is a sequence of consecutive integers.*

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

* Ranges can actually represent more general integer sequences.

The Range Type

Not on Midterm 1

A range is a sequence of consecutive integers.*

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

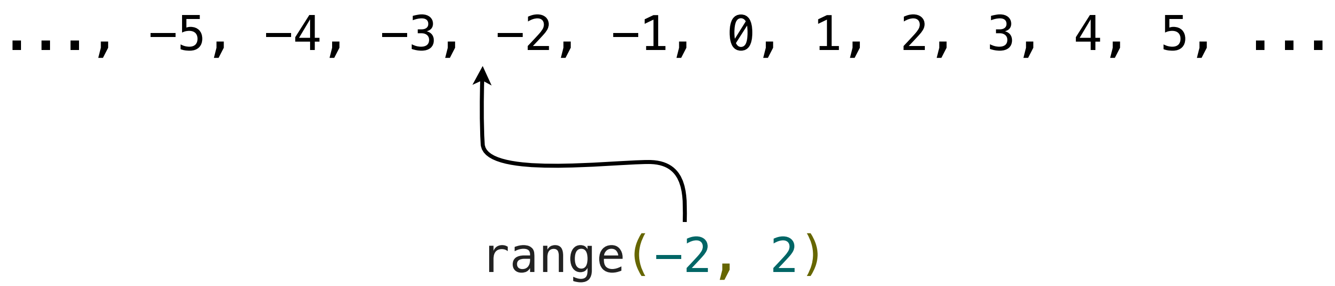
```
range(-2, 2)
```

* Ranges can actually represent more general integer sequences.

The Range Type

Not on Midterm 1

A range is a sequence of consecutive integers.*



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

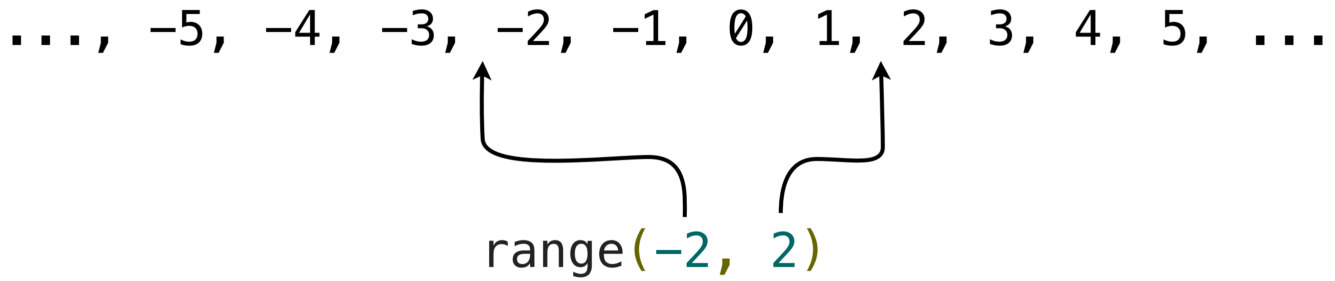
`range(-2, 2)`

* Ranges can actually represent more general integer sequences.

Not on Midterm 1

The Range Type

A range is a sequence of consecutive integers.*



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

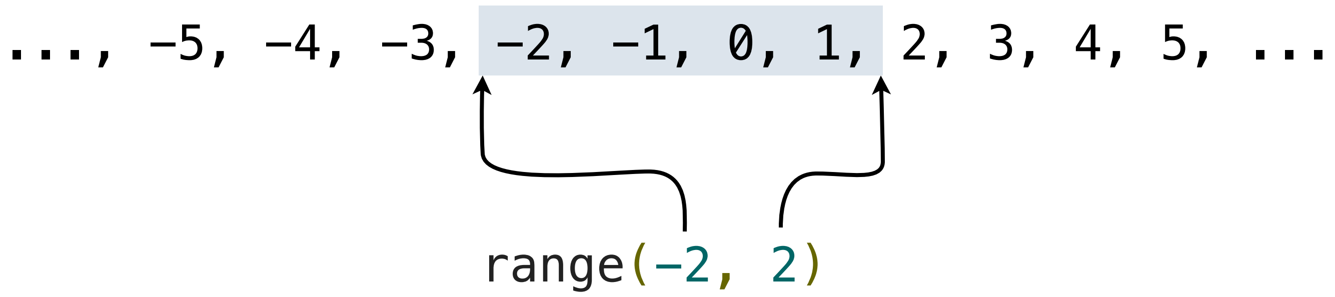
range(-2, 2)

* Ranges can actually represent more general integer sequences.

The Range Type

Not on Midterm 1

A range is a sequence of consecutive integers.*



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

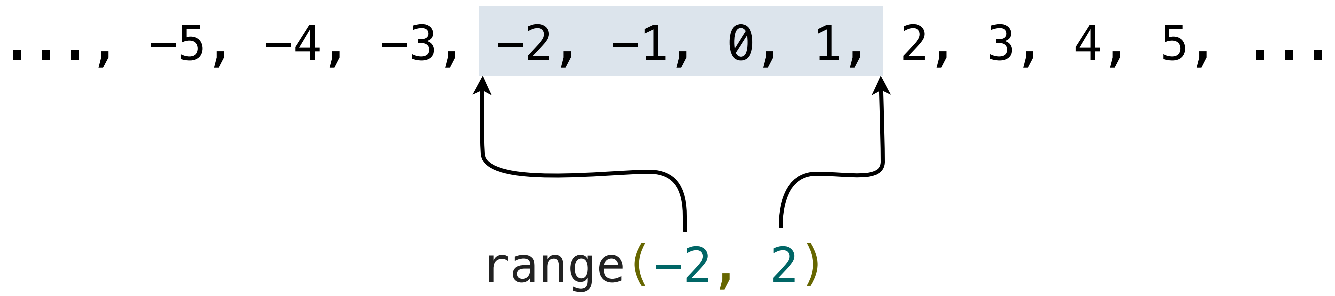
range(−2, 2)

* Ranges can actually represent more general integer sequences.

The Range Type

Not on Midterm 1

A range is a sequence of consecutive integers.*



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

range(-2, 2)

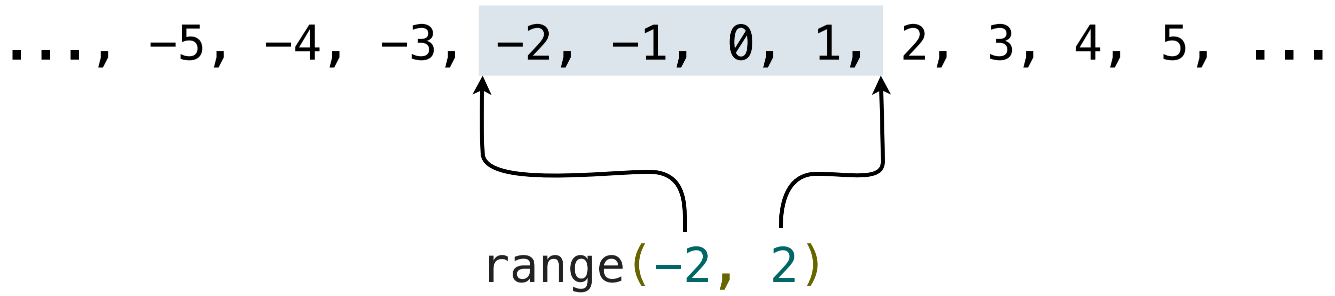
Length: ending value - starting value

* Ranges can actually represent more general integer sequences.

The Range Type

Not on Midterm 1

A range is a sequence of consecutive integers.*



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

range(−2, 2)

Length: ending value − starting value

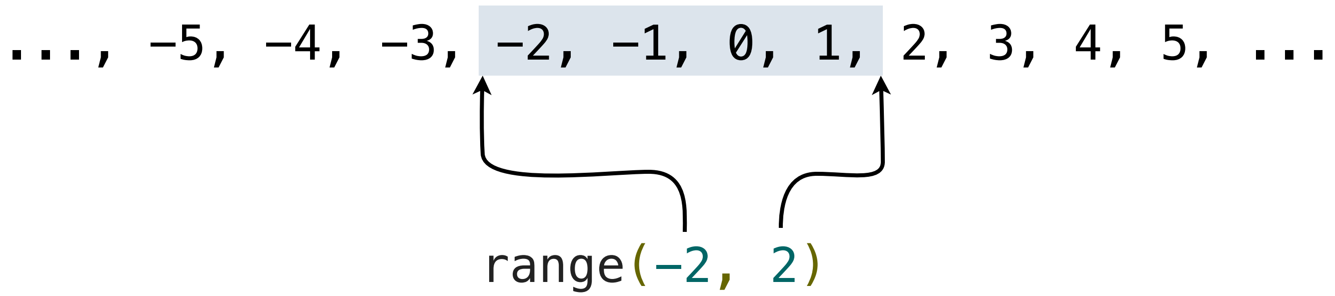
Element selection: starting value + index

* Ranges can actually represent more general integer sequences.

Not on Midterm 1

The Range Type

A range is a sequence of consecutive integers.*



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

range(-2, 2)

Length: ending value - starting value

Element selection: starting value + index

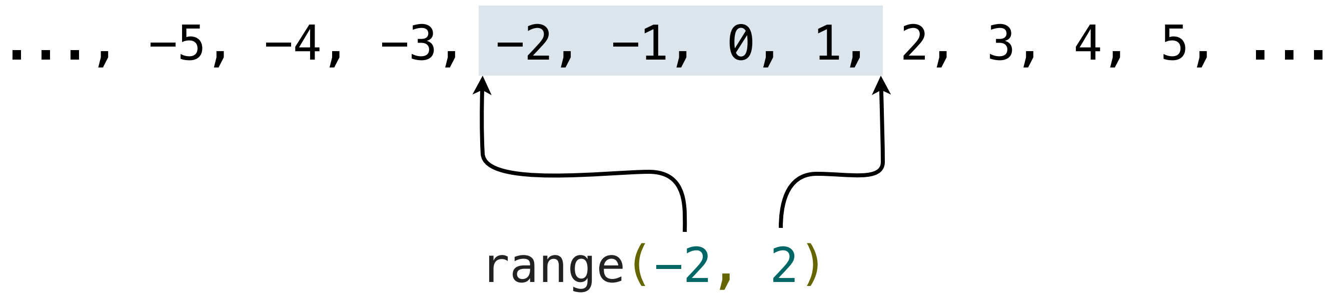
```
>>> tuple(range(-2, 2))
(-2, -1, 0, 1)

>>> tuple(range(4))
(0, 1, 2, 3)
```

* Ranges can actually represent more general integer sequences.

The Range Type

Not on Midterm 1

A range is a sequence of consecutive integers.*



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

range(-2, 2)

Length: ending value - starting value

Element selection: starting value + index

```
>>> tuple(range(-2, 2))
(-2, -1, 0, 1)

>>> tuple(range(4))
(0, 1, 2, 3)
```

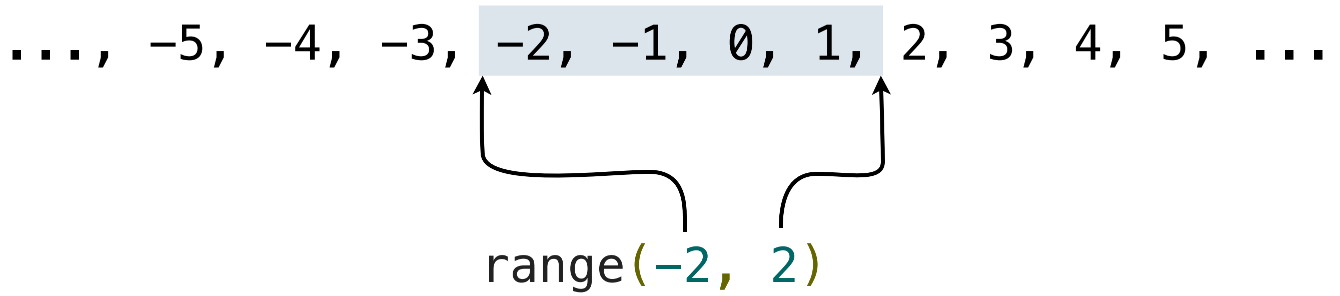
Tuple construction

* Ranges can actually represent more general integer sequences.

Not on Midterm 1

The Range Type

A range is a sequence of consecutive integers.*



Length: ending value - starting value

Element selection: starting value + index

```
>>> tuple(range(-2, 2))
(-2, -1, 0, 1)
```

Tuple construction

```
>>> tuple(range(4))
(0, 1, 2, 3)
```

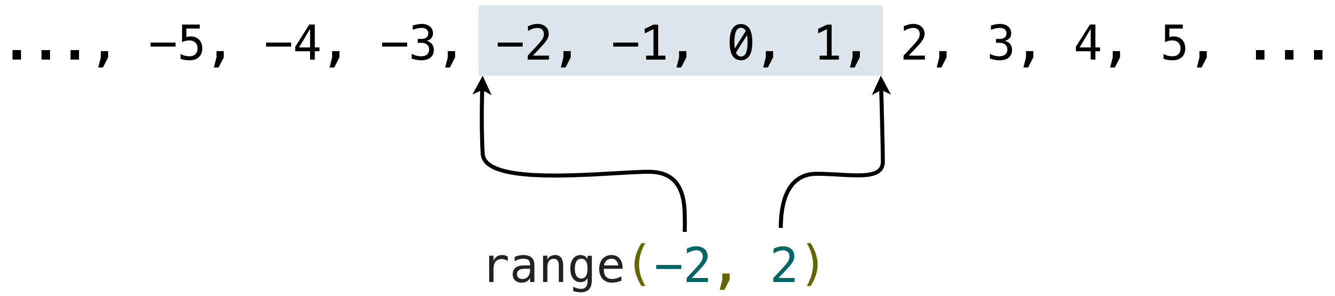
With a 0 starting value

* Ranges can actually represent more general integer sequences.

Not on Midterm 1

The Range Type

A range is a sequence of consecutive integers.*



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

range(−2, 2)

Length: ending value − starting value

(Demo)

Element selection: starting value + index

```
>>> tuple(range(-2, 2))
(-2, -1, 0, 1)
```

Tuple construction

```
>>> tuple(range(4))
(0, 1, 2, 3)
```

With a 0 starting value

* Ranges can actually represent more general integer sequences.

Membership & Slicing

Not on Midterm 1

The Python sequence abstraction has two more behaviors!

Membership & Slicing

Not on Midterm 1

The Python sequence abstraction has two more behaviors!

Membership.

Membership & Slicing

Not on Midterm 1

The Python sequence abstraction has two more behaviors!

Membership.

```
>>> digits = (1, 8, 2, 8)
>>> 2 in digits
True
>>> 1828 not in digits
True
```

Membership & Slicing

Not on Midterm 1

The Python sequence abstraction has two more behaviors!

Membership.

```
>>> digits = (1, 8, 2, 8)
>>> 2 in digits
True
>>> 1828 not in digits
True
```

Slicing.

Not on Midterm 1

Membership & Slicing

The Python sequence abstraction has two more behaviors!

Membership.

```
>>> digits = (1, 8, 2, 8)
>>> 2 in digits
True
>>> 1828 not in digits
True
```

Slicing.

```
>>> digits[0:2]
(1, 8)
>>> digits[1:]
(8, 2, 8)
```