Processing Sequential Data
Processing Sequential Data

Many data sets can be viewed and processed sequentially:
Processing Sequential Data

Many data sets can be viewed and processed sequentially:

• The set of all Twitter posts
Processing Sequential Data

Many data sets can be viewed and processed sequentially:
  • The set of all Twitter posts
  • Votes cast in a presidential election
Processing Sequential Data

Many data sets can be viewed and processed sequentially:

- The set of all Twitter posts
- Votes cast in a presidential election
- Sensor readings of an airplane
Processing Sequential Data

Many data sets can be viewed and processed sequentially:
• The set of all Twitter posts
• Votes cast in a presidential election
• Sensor readings of an airplane
• The set of all positive integers
Processing Sequential Data

Many data sets can be viewed and processed sequentially:

- The set of all Twitter posts
- Votes cast in a presidential election
- Sensor readings of an airplane
- The set of all positive integers

However, the sequence interface we developed previously does not always apply.
Many data sets can be viewed and processed sequentially:

- The set of all Twitter posts
- Votes cast in a presidential election
- Sensor readings of an airplane
- The set of all positive integers

However, the sequence interface we developed previously does not always apply.

- A sequence has a finite, known length
Processing Sequential Data

Many data sets can be viewed and processed sequentially:
• The set of all Twitter posts
• Votes cast in a presidential election
• Sensor readings of an airplane
• The set of all positive integers

However, the sequence interface we developed previously does not always apply.
• A sequence has a finite, known length
• A sequence support element selection for any element
Many data sets can be viewed and processed sequentially:

- The set of all Twitter posts
- Votes cast in a presidential election
- Sensor readings of an airplane
- The set of all positive integers

However, the sequence interface we developed previously does not always apply.

- A sequence has a finite, known length
- A sequence support element selection for any element

In most cases, satisfying the sequence interface requires storing the entire sequence in a computer's memory.
Processing Sequential Data

Many data sets can be viewed and processed sequentially:

- The set of all Twitter posts
- Votes cast in a presidential election
- Sensor readings of an airplane
- The set of all positive integers

However, the sequence interface we developed previously does not always apply.

- A sequence has a finite, known length
- A sequence support element selection for any element

In most cases, satisfying the sequence interface requires storing the entire sequence in a computer's memory.

**Today:** Efficient representations of sequential data
Implicit Sequences
Implicit Sequences

An implicit sequence is a representation of sequential data that does not explicitly store each element.
Implicit Sequences

An implicit sequence is a representation of sequential data that does not explicitly store each element.

Example: The range class represents consecutive integers.
Implicit Sequences

An implicit sequence is a representation of sequential data that does not explicitly store each element.

Example: The range class represents consecutive integers.
  • The range is represented by two values: \textit{start} and \textit{end}.
Implicit Sequences

An implicit sequence is a representation of sequential data that does not explicitly store each element.

**Example:** The range class represents consecutive integers.
- The range is represented by two values: *start* and *end*.
- The length and elements are computed on demand.
Implicit Sequences

An implicit sequence is a representation of sequential data that does not explicitly store each element.

**Example:** The range class represents consecutive integers.

- The range is represented by two values: *start* and *end*.
- The length and elements are computed on demand.
- Constant space for arbitrarily large sequences.
Implicit Sequences

An implicit sequence is a representation of sequential data that does not explicitly store each element.

**Example:** The range class represents consecutive integers.
- The range is represented by two values: *start* and *end*.
- The length and elements are computed on demand.
- Constant space for arbitrarily large sequences.

Demo
The Iterator Interface
The Iterator Interface

An iterator is an object that can provide the next element of a (possibly implicit) sequence.
The Iterator Interface

An iterator is an object that can provide the next element of a (possibly implicit) sequence.

The iterator interface has two methods:
The Iterator Interface

An iterator is an object that can provide the next element of a (possibly implicit) sequence.

The iterator interface has two methods:

- `__next__(self)` returns the next element in the sequence
The Iterator Interface

An iterator is an object that can provide the next element of a (possibly implicit) sequence.

The iterator interface has two methods:

• `__next__(self)` returns the next element in the sequence
• `__iter__(self)` returns an equivalent iterator (Why?)
The Iterator Interface

An iterator is an object that can provide the next element of a (possibly implicit) sequence.

The iterator interface has two methods:

• \texttt{\_\_next\_\_}(self) returns the next element in the sequence
• \texttt{\_\_iter\_\_}(self) returns an equivalent iterator (Why?)

The \texttt{next} function invokes the \texttt{\_\_next\_\_} method on its argument.
The Iterator Interface

An iterator is an object that can provide the next element of a (possibly implicit) sequence.

The iterator interface has two methods:
• __next__(self) returns the next element in the sequence
• __iter__(self) returns an equivalent iterator (Why?)

The next function invokes the __next__ method on its argument.

If there is no next element, then the __next__ method of an iterator should raise a StopIteration exception.
The Iterator Interface

An iterator is an object that can provide the next element of a (possibly implicit) sequence.

The iterator interface has two methods:
• `__next__(self)` returns the next element in the sequence
• `__iter__(self)` returns an equivalent iterator (Why?)

The `next` function invokes the `__next__` method on its argument.

If there is no next element, then the `__next__` method of an iterator should raise a `StopIteration` exception.
The For Statement
The For Statement

for <name> in <expression>:
    <suite>
The For Statement

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

1. Evaluate the header <expression>, which yields an iterable object.
The For Statement

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

1. Evaluate the header `<expression>`, which yields an iterable object.
2. For each element in that sequence, in order:
The For Statement

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

1. Evaluate the header `<expression>`, which yields an iterable object.
2. For each element in that sequence, in order:
   A. Bind `<name>` to that element in the first frame of the current environment.
The For Statement

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

1. Evaluate the header `<expression>`, which yields an iterable object.
2. For each element in that sequence, in order:
   A. Bind `<name>` to that element in the first frame of the current environment.
   B. Execute the `<suite>`.
The For Statement

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

1. Evaluate the header `<expression>`, which yields an iterable object.
2. For each element in that sequence, in order:
   A. Bind `<name>` to that element in the first frame of the current environment.
   B. Execute the `<suite>`.

An iterable object has a method `__iter__` that returns an iterator.
The For Statement

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

1. Evaluate the header `<expression>`, which yields an iterable object.
2. For each element in that sequence, in order:
   A. Bind `<name>` to that element in the first frame of the current environment.
   B. Execute the `<suite>`.

An iterable object has a method `__iter__` that returns an iterator.

```python
>>> counts = [1, 2, 3]
>>> for item in counts:
    print(item)
1
2
3
```
The For Statement

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

1. Evaluate the header <expression>, which yields an iterable object.
2. For each element in that sequence, in order:
   A. Bind <name> to that element in the first frame of the current environment.
   B. Execute the <suite>.

An iterable object has a method __iter__ that returns an iterator.

```python
>>> counts = [1, 2, 3]
>>> for item in counts:
    print(item)
1
2
3
```

```python
>>> counts = [1, 2, 3]
>>> items = counts.__iter__()
>>> try:
    while True:
      item = items.__next__()
      print(item)
except StopIteration:
    pass
1
2
3
```
Generators and Generator Functions
Generators and Generator Functions

A generator is an iterator backed by a generator function.
Generators and Generator Functions

A generator is an iterator backed by a generator function.

A generator function is a function that yields values.
Generators and Generator Functions

A generator is an iterator backed by a generator function. A generator function is a function that *yields* values. When a generator function is called, it returns a generator.
Generators and Generator Functions

A generator is an iterator backed by a generator function.

A generator function is a function that yields values.

When a generator function is called, it returns a generator.

```python
>>> def letters_generator():
    current = 'a'
    while current <= 'd':
        yield current
        current = chr(ord(current)+1)
```
Generators and Generator Functions

A generator is an iterator backed by a generator function.

A generator function is a function that yields values.

When a generator function is called, it returns a generator.

```python
>>> def letters_generator():
    current = 'a'
    while current <= 'd':
        yield current
        current = chr(ord(current)+1)

>>> for letter in letters_generator():
    print(letter)
```
Generators and Generator Functions

A generator is an iterator backed by a generator function.

A generator function is a function that yields values.

When a generator function is called, it returns a generator.

```python
>>> def letters_generator():
    current = 'a'
    while current <= 'd':
        yield current
        current = chr(ord(current)+1)

>>> for letter in letters_generator():
    print(letter)

a
b
c
d
```
Streams
Streams

A stream is a recursive list with an explicit first element and an implicit rest of the list.
Streams

A stream is a recursive list with an *explicit* first element and an *implicit* rest of the list.

class Stream(object):
    """A lazily computed recursive list."""
A stream is a recursive list with an explicit first element and an implicit rest of the list.

class Stream(object):
    """A lazily computed recursive list."""
    class empty(object):
        def __repr__(self):
            return 'Stream.empty'
    empty = empty()
Streams

A stream is a recursive list with an explicit first element and an implicit rest of the list.

class Stream(object):
    """A lazily computed recursive list."""

class empty(object):
    def __repr__(self):
        return 'Stream.empty'

empty = empty()

def __init__(self, first, compute_rest=lambda: empty):
    assert callable(compute_rest), 'compute_rest must be callable.'
    self.first = first
    self._compute_rest = compute_rest
    self._rest = None
Streams

A stream is a recursive list with an *explicit* first element and an *implicit* rest of the list.

```python
class Stream(object):
    """A lazily computed recursive list."""
    class empty(object):
        def __repr__(self):
            return 'Stream.empty'
    empty = empty()

    def __init__(self, first, compute_rest=lambda: empty):
        assert callable(compute_rest), 'compute_rest must be callable.'
        self.first = first
        self._compute_rest = compute_rest
        self._rest = None

    @property
def rest(self):
        """Return the rest of the stream, computing it if necessary."""
        if self._compute_rest is not None:
            self._rest = self._compute_rest()
            self._compute_rest = None
        return self._rest
```
Integer Streams
An integer stream is a stream of consecutive integers.

An integer stream starting at k consists of k and a function that returns the integer stream starting at k+1.
An integer stream is a stream of consecutive integers.

An integer stream starting at $k$ consists of $k$ and a function that returns the integer stream starting at $k+1$.

def make_integer_stream(first=1):
    """Return a stream of consecutive integers, starting with first."""
    """if first:
        s = make_integer_stream(first+1)
        return Stream(first, compute_rest())
    return Stream(first, compute_rest())

>>> s = make_integer_stream(3)
>>> s.first
3
>>> s.rest.first
4
"""
Mapping a Function over a Stream
Mapping a Function over a Stream

Mapping a function over a stream applies a function only to the first element at first, but computes the rest lazily.
Mapping a Function over a Stream

Mapping a function over a stream applies a function only to the first element at first, but computes the rest lazily.

```python
def map_stream(fn, s):
    """Map a function fn over the elements of a stream s."""
    if s is Stream.empty:
        return s
    def compute_rest():
        return map_stream(fn, s.rest)
    return Stream(fn(s.first), compute_rest)
```
Mapping a Function over a Stream

Mapping a function over a stream applies a function only to the first element at first, but computes the rest lazily.

```python
def map_stream(fn, s):
    """Map a function fn over the elements of a stream s."""
    if s is Stream.empty:
        return s
    def compute_rest():
        return map_stream(fn, s.rest)
    return Stream(fn(s.first), compute_rest)

Demo
```
Filtering a Stream
Filtering a Stream

When filtering a stream, processing continues until an element is kept in the output.
Filtering a Stream

When filtering a stream, processing continues until an element is kept in the output.

```python
def filter_stream(fn, s):
    """Filter stream s with predicate function fn."""
    if s is Stream.empty:
        return s
    def compute_rest():
        return filter_stream(fn, s.rest)
    if fn(s.first):
        return Stream(s.first, compute_rest)
    else:
        return compute_rest()
```
A Stream of Primes
A Stream of Primes

The stream of integers not divisible by any $k \leq n$ is:
A Stream of Primes

The stream of integers not divisible by any \( k \leq n \) is:

- The stream of integers not divisible by any \( k < n \),
A Stream of Primes

The stream of integers not divisible by any $k \leq n$ is:

- The stream of integers not divisible by any $k < n$,
- Filtered to remove any element divisible by $n$. 
A Stream of Primes

The stream of integers not divisible by any $k \leq n$ is:

- The stream of integers not divisible by any $k < n$,
- Filtered to remove any element divisible by $n$.
- Called the Sieve of Eratosthenes.
The stream of integers not divisible by any $k \leq n$ is:

- The stream of integers not divisible by any $k < n$,
- Filtered to remove any element divisible by $n$.
- Called the Sieve of Eratosthenes.

2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
A Stream of Primes

The stream of integers not divisible by any $k \leq n$ is:
- The stream of integers not divisible by any $k < n$,
- Filtered to remove any element divisible by $n$.
- Called the Sieve of Eratosthenes.

2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
A Stream of Primes

The stream of integers not divisible by any \( k \leq n \) is:
- The stream of integers not divisible by any \( k < n \),
- Filtered to remove any element divisible by \( n \).
- Called the Sieve of Eratosthenes.

\[ 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 \]
A Stream of Primes

The stream of integers not divisible by any $k \leq n$ is:
• The stream of integers not divisible by any $k < n$,
• Filtered to remove any element divisible by $n$.
• Called the Sieve of Eratosthenes.

2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
A Stream of Primes

The stream of integers not divisible by any $k \leq n$ is:
- The stream of integers not divisible by any $k < n$,
- Filtered to remove any element divisible by $n$.
- Called the Sieve of Eratosthenes.

2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
A Stream of Primes

The stream of integers not divisible by any $k \leq n$ is:
- The stream of integers not divisible by any $k < n$,
- Filtered to remove any element divisible by $n$.
- Called the Sieve of Eratosthenes.

$2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13$
A Stream of Primes

The stream of integers not divisible by any \( k \leq n \) is:
- The stream of integers not divisible by any \( k < n \),
- Filtered to remove any element divisible by \( n \).
- Called the Sieve of Eratosthenes.

\[
2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
\]