Announcements

- Homework 9 due Tuesday 11/19 @ 11:59pm
- Project 4 due Thursday 11/21 @ 11:59pm

Data Processing

Processing Sequential Data

Many data sets can be processed sequentially:
- The set of all Twitter posts
- Votes cast in an election
- Sensor readings of an airplane
- The positive integers: 1, 2, 3, ...

However, the sequence interface we used before does not always apply.
- A sequence has a finite, known length.
- A sequence allows element selection for any element.

Important ideas in big data processing:
- Implicit representations of streams of sequential data
- Declarative programming languages to manipulate and transform data
- Distributed and parallel computing

Implicit Sequences

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

Example: The built-in 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 long sequences.
Iterators

The Iterator Interface

An iterator is an object that can provide the next element of a sequence.

The \_\_next\_\_ method of an iterator returns the next element.

The built-in 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.

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

```
iter(range(-2, 2))  \____\ returns \____\ next(<range_iterator object>)
Is invokes \_\_iter\_\_ on its argument \____\ \____\ \____\ [Demo]
```

Iterables and Iterators

Iterator: Mutable object that tracks a position in a sequence, advancing on \_\_next\_\_.

Iterable: Represents a sequence and returns a new iterator on \_\_iter\_\_.

```

LetterIter is an iterator:  LetterIter('a', 'e')  \____\
LetterIter('a', 'e')  \____\

Letters is iterable:  Letters['a', 'e']  'a'  'b'  'c'  'd'

```

[Demo]

Iterable Objects

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\_.

When executing a for statement, \_\_iter\_\_ returns an iterator and \_\_next\_\_ provides each item.

```

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

```

For Statements
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(next_letter, end):
    while next_letter < end:
        yield next_letter
        next_letter = chr(ord(next_letter) + 1)

>>> for letter in letters_generator('a', 'd'):
    print(letter)
a b c d
```

Generator Examples

fib_generator(): "Yield Fibonacci numbers."
all_pairs(s): "Yield pairs of elements from iterable s."
Letters.__iter__(): "Yield sequential letters."
powerset(): "Yield all subsets of iterator t."