Sequence Iteration
def count(s, value):
    total = 0
    for elem in s:
        if elem == value:
            total = total + 1
    return total
For Statement Execution Procedure
For Statement Execution Procedure

for <name> in <expression>:
    <suite>
For Statement Execution Procedure

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

1. Evaluate the header <expression>, which must yield an iterable value.
For Statement Execution Procedure

```python
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

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 first frame of the current environment.
For Statement Execution Procedure

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 first frame of the current environment.

   B. Execute the <suite>.
Sequence Unpacking in For Statements
Sequence Unpacking in For Statements

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

>>> same_count = 0
```
Sequence Unpacking in For Statements

A sequence of fixed-length sequences

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

>>> same_count = 0
```
Sequence Unpacking in For Statements

A sequence of fixed-length sequences

```python
>>> pairs = [(1, 2), (2, 2), (2, 3), (4, 4)]

>>> same_count = 0

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

>>> same_count
2
```
Sequence Unpacking in For Statements

A sequence of fixed-length sequences

```python
>>> 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
```
Sequence Unpacking in For Statements

A sequence of fixed-length sequences

```python
>>> 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
```

A name for each element in a fixed-length sequence

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

A range is a sequence of consecutive integers.*
The Range Type

A range is a sequence of consecutive integers.*

* Ranges can actually represent more general integer sequences.
The Range Type

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

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

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

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

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

A range is a sequence of consecutive integers.*

\[ ..., -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, ... \]

Length: ending value - starting value

* Ranges can actually represent more general integer sequences.
The Range Type

A range is a sequence of consecutive integers.*

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

Length: ending value − starting value

Element selection: starting value + index

* Ranges can actually represent more general integer sequences.
The Range Type

A range is a sequence of consecutive integers.*

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

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

A range is a sequence of consecutive integers.*

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

Range: 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

A range is a sequence of consecutive integers.*

..., −5, −4, −3, \(\text{range}(-2, 2)\), −2, −1, 0, 1, 2, 3, 4, 5, ...

Length: ending value − starting value

Element selection: starting value + index

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

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

* Ranges can actually represent more general integer sequences.
The Range Type

A range is a sequence of consecutive integers.*

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

Range: range(-2, 2)

Length: ending value - starting value

Element selection: starting value + index

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

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

* Ranges can actually represent more general integer sequences.
Membership & Slicing

The Python sequence abstraction has two more behaviors!
Membership & Slicing

The Python sequence abstraction has two more behaviors!

Membership.
Membership & Slicing

The Python sequence abstraction has two more behaviors!

**Membership.**

```python
>>> digits = (1, 8, 2, 8)
>>> 2 in digits
True
>>> 1828 not in digits
True
```
Membership & Slicing

The Python sequence abstraction has two more behaviors!

Membership.

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

Slicing.
Membership & Slicing

The Python sequence abstraction has two more behaviors!

**Membership.**

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

**Slicing.**

```python
>>> digits[0:2]
(1, 8)
>>> digits[1:]
(8, 2, 8)
```
Strings are an Abstraction
Strings are an Abstraction

Representing data:

'200'  '1.2e-5'  'False'  '(1, 2)'
Strings are an Abstraction

Representing data:

'200' '1.2e-5' 'False' '(1, 2)'

Representing language:

""""""And, as imagination bodies forth
The forms of things to unknown, and the poet's pen
Turns them to shapes, and gives to airy nothing
A local habitation and a name.""""
Strings are an Abstraction

Representing data:

'200'    '1.2e-5'    'False'    '(1, 2)'

Representing language:

""""And, as imagination bodies forth
The forms of things to unknown, and the poet's pen
Turns them to shapes, and gives to airy nothing
A local habitation and a name.
"""

Representing programs:

'curry = lambda f: lambda x: lambda y: f(x, y)'
Strings are an Abstraction

Representing data:

'200'    '1.2e-5'    'False'    '(1, 2)'

Representing language:

"""And, as imagination bodies forth
The forms of things to unknown, and the poet's pen
Turns them to shapes, and gives to airy nothing
A local habitation and a name.
"""

Representing programs:

'curry = lambda f: lambda x: lambda y: f(x, y)'

Demo
String Literals Have Three Forms

>>> 'I am string!'  
'I am string!'

>>> "I've got an apostrophe"
"I've got an apostrophe"

>>> '您好'
'您好'
String Literals Have Three Forms

>>> 'I am string!'
'I am string!'

>>> "I've got an apostrophe"
"I've got an apostrophe"

>>> '您好'
'您好'

Single- and double-quoted strings are equivalent
String Literals Have Three Forms

```python
>>> 'I am string!'  
'I am string!'

>>> "I've got an apostrophe"  
"I've got an apostrophe"

>>> '您好'  
'您好'

>>> """"""The Zen of Python  
claims, Readability counts.  
Read more: import this."""

'The Zen of Python
claims, Readability counts.  
Read more: import this.'
```

Single- and double-quoted strings are equivalent
String Literals Have Three Forms

```python
>>> 'I am string!'
'I am string!'

>>> "I've got an apostrophe"
"I've got an apostrophe"

>>> '您好'
'您好'

>>> """The Zen of Python claims, Readability counts.
Read more: import this."""
'The Zen of Python claims, Readability counts.
Read more: import this.'
```

A backslash "escapes" the following character

Single- and double-quoted strings are equivalent
String Literals Have Three Forms

>>> 'I am string!'
'I am string!'

>>> "I've got an apostrophe"
"I've got an apostrophe"

>>> '您好'
'您好'

>>> """"The Zen of Python
cclaims, Readability counts.
Read more: import this."""
'The Zen of Python claims, Readability counts.\nRead more: import this.'

A backslash "escapes" the following character

"Line feed" character represents a new line
Strings are Sequences
Strings are Sequences

**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.
Strings are Sequences

```python
>>> city = 'Berkeley'
>>> len(city)
8
>>> city[3]
'k'
```

**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.
Strings are Sequences

```python
>>> city = 'Berkeley'
>>> len(city)
8
>>> city[3]  # An element of a string is itself a string!
'k'
```

**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.
Strings are Sequences

```python
>>> city = 'Berkeley'
>>> len(city)
8
>>> city[3]
'k'
```

An element of a string is itself a string!

**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.

```python
>>> 'Berkeley' + ', CA'
'Berkeley, CA'
>>> 'Shabu ' * 2
'Shabu Shabu '
```
Strings are Sequences

```plaintext
>>> city = 'Berkeley'
>>> len(city)
8
>>> city[3]
'k'
```

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.

```plaintext
>>> 'Berkeley' + ', CA'
'Berkeley, CA'
>>> 'Shabu ' * 2
'Shabu Shabu '
```

String arithmetic is similar to tuple arithmetic.
String Membership Differs from Other Sequence Types
String Membership Differs from Other Sequence Types

The "in" and "not in" operators match substrings
String Membership Differs from Other Sequence Types

The "in" and "not in" operators match substrings

```python
>>> 'here' in "Where's Waldo?"
True
```
String Membership Differs from Other Sequence Types

The "in" and "not in" operators match substrings

```python
>>> 'here' in "Where's Waldo?"
True
```

Why? Working with strings, we care about words, not characters.
String Membership Differs from Other Sequence Types

The "in" and "not in" operators match substrings

```python
>>> 'here' in "Where's Waldo?"
True
```

Why? Working with strings, we care about words, not characters

The count method also matches substrings
String Membership Differs from Other Sequence Types

The "in" and "not in" operators match substrings

```python
>>> 'here' in "Where's Waldo?"
True
```

Why? Working with strings, we care about words, not characters

The count method also matches substrings

```python
>>> 'Mississippi'.count('i')
4
>>> 'Mississippi'.count('issi')
1
```
String Membership Differs from Other Sequence Types

The "in" and "not in" operators match substrings

```python
>>> 'here' in "Where's Waldo?"
True
```

Why? Working with strings, we care about words, not characters

The count method also matches substrings

```python
>>> 'Mississippi'.count('i')
4
>>> 'Mississippi'.count('issi')
1
```

the number of non-overlapping occurrences of a substring
String Membership Differs from Other Sequence Types

The "in" and "not in" operators match substrings

```python
>>> 'here' in "Where's Waldo?"
True
```

Why? Working with strings, we care about words, not characters

The count method also matches substrings

```python
>>> 'Mississippi'.count('i')
4
>>> 'Mississippi'.count('issi')
1
```

the number of non-overlapping occurrences of a substring
Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NUL</td>
<td>SOH</td>
<td>STX</td>
<td>ETX</td>
<td>EOT</td>
<td>ENQ</td>
<td>ACK</td>
<td>BEL</td>
<td>BS</td>
<td>HT</td>
<td>LF</td>
<td>VT</td>
<td>FF</td>
<td>CR</td>
<td>SO</td>
<td>SI</td>
</tr>
<tr>
<td>1</td>
<td>DLE</td>
<td>DC1</td>
<td>DC2</td>
<td>DC3</td>
<td>DC4</td>
<td>NAK</td>
<td>SYN</td>
<td>ETB</td>
<td>CAN</td>
<td>EM</td>
<td>SUB</td>
<td>ESC</td>
<td>FS</td>
<td>GS</td>
<td>RS</td>
<td>US</td>
</tr>
<tr>
<td>2</td>
<td>!</td>
<td>&quot;</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>’</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>-</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>:</td>
<td>;</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
</tr>
<tr>
<td>4</td>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>5</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>[</td>
<td>\</td>
<td>]</td>
<td>^</td>
<td>_</td>
</tr>
<tr>
<td>6</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
<td></td>
<td></td>
<td>}</td>
<td>~</td>
</tr>
</tbody>
</table>
### American Standard Code for Information Interchange

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0</strong></td>
<td>NUL</td>
<td>SOH</td>
<td>STX</td>
<td>ETX</td>
<td>EOT</td>
<td>ENQ</td>
<td>ACK</td>
<td>BEL</td>
<td>BS</td>
<td>HT</td>
<td>LF</td>
<td>VT</td>
<td>FF</td>
<td>CR</td>
<td>SO</td>
</tr>
<tr>
<td><strong>1</strong></td>
<td>DLE</td>
<td>DC1</td>
<td>DC2</td>
<td>DC3</td>
<td>DC4</td>
<td>NAK</td>
<td>SYN</td>
<td>ETB</td>
<td>CAN</td>
<td>EM</td>
<td>SUB</td>
<td>ESC</td>
<td>FS</td>
<td>GS</td>
<td>RS</td>
</tr>
<tr>
<td><strong>2</strong></td>
<td>!</td>
<td>&quot;</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>.</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td><strong>3</strong></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>:</td>
<td>;</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
</tr>
<tr>
<td><strong>4</strong></td>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
</tr>
<tr>
<td><strong>5</strong></td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>[</td>
<td>\</td>
<td>]</td>
<td>^</td>
</tr>
<tr>
<td><strong>6</strong></td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td>o</td>
</tr>
<tr>
<td><strong>7</strong></td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
<td></td>
<td>}</td>
<td>~</td>
</tr>
</tbody>
</table>
Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUL</td>
<td>SOH</td>
<td>STX</td>
<td>ETX</td>
<td>EOT</td>
<td>ENQ</td>
<td>ACK</td>
<td>BEL</td>
<td>BS</td>
<td>HT</td>
<td>LF</td>
<td>VT</td>
<td>FF</td>
<td>CR</td>
<td>SO</td>
<td>SI</td>
</tr>
<tr>
<td>DLE</td>
<td>DC1</td>
<td>DC2</td>
<td>DC3</td>
<td>DC4</td>
<td>NAK</td>
<td>SYN</td>
<td>ETB</td>
<td>CAN</td>
<td>EM</td>
<td>SUB</td>
<td>ESC</td>
<td>FS</td>
<td>GS</td>
<td>RS</td>
<td>US</td>
</tr>
<tr>
<td>!</td>
<td>&quot;</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>'</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>.</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>:</td>
<td>;</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
</tr>
<tr>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>[</td>
<td>\</td>
<td>]</td>
<td>^</td>
<td>_</td>
</tr>
<tr>
<td>`</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td>o</td>
</tr>
<tr>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
<td></td>
<td>}</td>
<td>~</td>
<td>DEL</td>
</tr>
</tbody>
</table>

8 rows: 3 bits
16 columns: 4 bits
**Representing Strings: the ASCII Standard**

American Standard Code for Information Interchange

<p>| | | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NUL</td>
<td>SOH</td>
<td>STX</td>
<td>ETX</td>
<td>EOT</td>
<td>ENQ</td>
<td>ACK</td>
<td>BEL</td>
<td>BS</td>
<td>HT</td>
<td>LF</td>
<td>VT</td>
<td>FF</td>
<td>CR</td>
<td>SO</td>
</tr>
<tr>
<td>1</td>
<td>DLE</td>
<td>DC1</td>
<td>DC2</td>
<td>DC3</td>
<td>DC4</td>
<td>NAK</td>
<td>SYN</td>
<td>ETB</td>
<td>CAN</td>
<td>EM</td>
<td>SUB</td>
<td>ESC</td>
<td>FS</td>
<td>GS</td>
<td>RS</td>
</tr>
<tr>
<td>2</td>
<td>!</td>
<td>&quot;</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>.</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>;</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
</tr>
<tr>
<td>4</td>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
</tr>
<tr>
<td>5</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>[</td>
<td>\</td>
<td>]</td>
<td>^</td>
</tr>
<tr>
<td>6</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td>o</td>
</tr>
<tr>
<td>7</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
<td></td>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

16 columns: 4 bits

- Layout was chosen to support sorting by character code

**Bonus Material**
Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUL</td>
<td>SOH</td>
<td>STX</td>
<td>ETX</td>
<td>EOT</td>
<td>ENQ</td>
<td>ACK</td>
<td>BEL</td>
<td>BS</td>
<td>HT</td>
<td>LT</td>
<td>FF</td>
<td>CR</td>
<td>SO</td>
<td>SI</td>
<td></td>
</tr>
<tr>
<td>DLE</td>
<td>DC1</td>
<td>DC2</td>
<td>DC3</td>
<td>DC4</td>
<td>NAK</td>
<td>SYN</td>
<td>ETB</td>
<td>CAN</td>
<td>EM</td>
<td>SUB</td>
<td>ESC</td>
<td>FS</td>
<td>GS</td>
<td>RS</td>
<td>US</td>
</tr>
<tr>
<td>!</td>
<td>&quot;</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>'</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>-</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>:</td>
<td>;</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
</tr>
<tr>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>[</td>
<td>\</td>
<td>]</td>
<td>^</td>
<td>_</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
<td></td>
<td></td>
<td>}</td>
<td>DEL</td>
</tr>
</tbody>
</table>

16 columns: 4 bits
- Layout was chosen to support sorting by character code
- Rows indexed 2–5 are a useful 6-bit (64 element) subset
Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

<table>
<thead>
<tr>
<th>ASCII Code Chart</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NUL</td>
<td>SOH</td>
<td>STX</td>
<td>ETX</td>
<td>EOT</td>
<td>ENQ</td>
<td>ACK</td>
<td>BEL</td>
<td>BS</td>
<td>HT</td>
<td>LF</td>
<td>VT</td>
<td>FF</td>
<td>CR</td>
<td>SO</td>
<td>SI</td>
</tr>
<tr>
<td>1</td>
<td>DLE</td>
<td>DC1</td>
<td>DC2</td>
<td>DC3</td>
<td>DC4</td>
<td>NAK</td>
<td>SYN</td>
<td>ETB</td>
<td>CAN</td>
<td>EM</td>
<td>SUB</td>
<td>ESC</td>
<td>FS</td>
<td>GS</td>
<td>RS</td>
<td>US</td>
</tr>
<tr>
<td>2</td>
<td>!</td>
<td>&quot;</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>'</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>-</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>4</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>[</td>
<td>\</td>
<td>]</td>
<td>^</td>
<td>_</td>
</tr>
<tr>
<td>5</td>
<td>\</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td>o</td>
</tr>
<tr>
<td>6</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
<td></td>
<td></td>
<td>}</td>
<td>~</td>
</tr>
</tbody>
</table>
| 7                | DEL | 16 columns: 4 bits

- Layout was chosen to support sorting by character code
- Rows indexed 2–5 are a useful 6-bit (64 element) subset
- Control characters were designed for transmission
Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

<table>
<thead>
<tr>
<th>ASCII Code Chart</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>NUL</td>
</tr>
<tr>
<td>DLE</td>
</tr>
<tr>
<td>!</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>@</td>
</tr>
<tr>
<td>P</td>
</tr>
<tr>
<td>`</td>
</tr>
<tr>
<td>p</td>
</tr>
</tbody>
</table>

16 columns: 4 bits

- Layout was chosen to support sorting by character code
- Rows indexed 2–5 are a useful 6-bit (64 element) subset
- Control characters were designed for transmission
Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUL</td>
<td>SOH</td>
<td>STX</td>
<td>ETX</td>
<td>EOT</td>
<td>ENQ</td>
<td>ACK</td>
<td>BEL</td>
<td>BS</td>
<td>HT</td>
<td>LF</td>
<td>VT</td>
<td>FF</td>
<td>CR</td>
<td>SO</td>
<td>SI</td>
</tr>
<tr>
<td>DLE</td>
<td>DC1</td>
<td>DC2</td>
<td>DC3</td>
<td>DC4</td>
<td>NAK</td>
<td>SYN</td>
<td>ETB</td>
<td>CAN</td>
<td>EM</td>
<td>SUB</td>
<td>ESC</td>
<td>FS</td>
<td>GS</td>
<td>RS</td>
<td>US</td>
</tr>
<tr>
<td>!</td>
<td>&quot;</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>'</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>-</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>:</td>
<td>;</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
</tr>
<tr>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>[</td>
<td>\</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td>o</td>
<td>p</td>
</tr>
<tr>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 16 columns: 4 bits
- Layout was chosen to support sorting by character code
- Rows indexed 2–5 are a useful 6-bit (64 element) subset
- Control characters were designed for transmission

Bonus Material
Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUL</td>
<td>SOH</td>
<td>STX</td>
<td>ETX</td>
<td>EOT</td>
<td>ENQ</td>
<td>ACK</td>
<td>BEL</td>
<td>BS</td>
<td>HT</td>
<td>LF</td>
<td>VT</td>
<td>FF</td>
<td>CR</td>
<td>SO</td>
<td>SI</td>
</tr>
<tr>
<td>DLE</td>
<td>DC1</td>
<td>DC2</td>
<td>DC3</td>
<td>DC4</td>
<td>NAK</td>
<td>SYN</td>
<td>ETB</td>
<td>CAN</td>
<td>EM</td>
<td>SUB</td>
<td>ESC</td>
<td>FS</td>
<td>GS</td>
<td>RS</td>
<td>US</td>
</tr>
<tr>
<td>!</td>
<td>&quot;</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>'</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>-</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>:</td>
<td>;</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
</tr>
<tr>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>[</td>
<td>\</td>
<td>]</td>
<td>^</td>
<td>_</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
<td></td>
<td></td>
<td>}</td>
<td>-</td>
</tr>
</tbody>
</table>

16 columns: 4 bits

- Layout was chosen to support sorting by character code
- Rows indexed 2–5 are a useful 6-bit (64 element) subset
- Control characters were designed for transmission

Demo
Representing Strings: the Unicode Standard
Representing Strings: the Unicode Standard

http://ian-albert.com/unicode_chart/unichart-chinese.jpg
• **109,000 characters**

![Image of Chinese characters and their Unicode values](http://ian-albert.com/unicode_chart/unichart-chinese.jpg)
Representing Strings: the Unicode Standard

- 109,000 characters
- 93 scripts (organized)

http://ian-albert.com/unicode_chart/unichart-chinese.jpg
Representing Strings: the Unicode Standard

• 109,000 characters
• 93 scripts (organized)
• Enumeration of character properties, such as case
Representing Strings: the Unicode Standard

- 109,000 characters
- 93 scripts (organized)
- Enumeration of character properties, such as case
- Supports bidirectional display order

http://ian-albert.com/unicode_chart/unichart-chinese.jpg
Representing Strings: the Unicode Standard

- 109,000 characters
- 93 scripts (organized)
- Enumeration of character properties, such as case
- Supports bidirectional display order
- A canonical name for every character

http://ian-albert.com/unicode_chart/unichart-chinese.jpg
Representing Strings: the Unicode Standard

- 109,000 characters
- 93 scripts (organized)
- Enumeration of character properties, such as case
- Supports bidirectional display order
- A canonical name for every character

U+0058 LATIN CAPITAL LETTER X

Bonus Material

http://ian-albert.com/unicode_chart/unichart-chinese.jpg
Representing Strings: the Unicode Standard

- 109,000 characters
- 93 scripts (organized)
- Enumeration of character properties, such as case
- Supports bidirectional display order
- A canonical name for every character

U+0058 LATIN CAPITAL LETTER X

U+263a WHITE SMILING FACE
Representing Strings: the Unicode Standard

- 109,000 characters
- 93 scripts (organized)
- Enumeration of character properties, such as case
- Supports bidirectional display order
- A canonical name for every character

U+0058 LATIN CAPITAL LETTER X

U+263a WHITE SMILING FACE

U+2639 WHITE FROWNING FACE
Representing Strings: the Unicode Standard

- 109,000 characters
- 93 scripts (organized)
- Enumeration of character properties, such as case
- Supports bidirectional display order
- A canonical name for every character

U+0058 LATIN CAPITAL LETTER X

U+263a WHITE SMILING FACE

U+2639 WHITE FROWNING FACE

http://ian-albert.com/unicode_chart/unichart-chinese.jpg
Representing Strings: the Unicode Standard

- 109,000 characters
- 93 scripts (organized)
- Enumeration of character properties, such as case
- Supports bidirectional display order
- A canonical name for every character

U+0058 LATIN CAPITAL LETTER X

U+263a WHITE SMILING FACE

U+2639 WHITE FROWNING FACE

http://ian-albert.com/unicode_chart/unichart-chinese.jpg
Representing Strings: the Unicode Standard

- 109,000 characters
- 93 scripts (organized)
- Enumeration of character properties, such as case
- Supports bidirectional display order
- A canonical name for every character

U+0058 LATIN CAPITAL LETTER X
U+263a WHITE SMILING FACE
U+2639 WHITE FROWNING FACE

http://ian-albert.com/unicode_chart/unichart-chinese.jpg
Representing Strings: UTF-8 Encoding
Representing Strings: UTF-8 Encoding

UTF (UCS (Universal Character Set) Transformation Format)
Representing Strings: UTF-8 Encoding

UTF (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and integers
Representing Strings: UTF-8 Encoding

UTF (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and integers

UTF-8: Correspondence between numbers and bytes
Representing Strings: UTF-8 Encoding

UTF (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and integers

UTF-8: Correspondence between numbers and bytes

A byte is 8 bits and can encode any integer 0–255
Representing Strings: UTF-8 Encoding

UTF (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and integers

UTF-8: Correspondence between numbers and bytes

A byte is 8 bits and can encode any integer 0–255
Representing Strings: UTF-8 Encoding

UTF (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and integers

UTF-8: Correspondence between numbers and bytes

A byte is 8 bits and can encode any integer 0–255
Representing Strings: UTF-8 Encoding

UTF (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and integers

UTF-8: Correspondence between numbers and bytes

A byte is 8 bits and can encode any integer 0–255

```
  00000000  0
```

bytes            integers
Representing Strings: UTF-8 Encoding

UTF (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and integers

UTF-8: Correspondence between numbers and bytes

A byte is 8 bits and can encode any integer 0–255

\[
\begin{array}{c|c}
\text{bytes} & \text{integers} \\
00000000 & 0 \\
00000001 & 1 \\
\end{array}
\]
Representing Strings: UTF-8 Encoding

UTF (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and integers

UTF-8: Correspondence between numbers and bytes

A byte is 8 bits and can encode any integer 0–255

<table>
<thead>
<tr>
<th>bytes</th>
<th>integers</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>0</td>
</tr>
<tr>
<td>00000001</td>
<td>1</td>
</tr>
<tr>
<td>00000010</td>
<td>2</td>
</tr>
</tbody>
</table>
Representing Strings: UTF-8 Encoding

UTF (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and integers

UTF-8: Correspondence between numbers and bytes

A byte is 8 bits and can encode any integer 0–255

<table>
<thead>
<tr>
<th>bytes</th>
<th>integers</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>0</td>
</tr>
<tr>
<td>00000001</td>
<td>1</td>
</tr>
<tr>
<td>00000010</td>
<td>2</td>
</tr>
<tr>
<td>00000011</td>
<td>3</td>
</tr>
</tbody>
</table>
Representing Strings: UTF-8 Encoding

UTF (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and integers

UTF-8: Correspondence between numbers and bytes

A byte is 8 bits and can encode any integer 0–255

<table>
<thead>
<tr>
<th>bytes</th>
<th>integers</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>0</td>
</tr>
<tr>
<td>00000001</td>
<td>1</td>
</tr>
<tr>
<td>00000010</td>
<td>2</td>
</tr>
<tr>
<td>00000011</td>
<td>3</td>
</tr>
</tbody>
</table>

Variable-length encoding: integers vary in the number of bytes required to encode them!
Representing Strings: UTF-8 Encoding

UTF (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and integers

UTF-8: Correspondence between numbers and bytes

A byte is 8 bits and can encode any integer 0–255

<table>
<thead>
<tr>
<th>bytes</th>
<th>integers</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>0</td>
</tr>
<tr>
<td>00000001</td>
<td>1</td>
</tr>
<tr>
<td>00000010</td>
<td>2</td>
</tr>
<tr>
<td>00000011</td>
<td>3</td>
</tr>
</tbody>
</table>

Variable-length encoding: integers vary in the number of bytes required to encode them!

In Python: string length in characters, bytes length in bytes
Representing Strings: UTF-8 Encoding

UTF (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and integers

UTF-8: Correspondence between numbers and bytes

A byte is 8 bits and can encode any integer 0–255

<table>
<thead>
<tr>
<th>bytes</th>
<th>integers</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>0</td>
</tr>
<tr>
<td>00000001</td>
<td>1</td>
</tr>
<tr>
<td>00000010</td>
<td>2</td>
</tr>
<tr>
<td>00000011</td>
<td>3</td>
</tr>
</tbody>
</table>

Variable-length encoding: integers vary in the number of bytes required to encode them!

In Python: string length in characters, bytes length in bytes

Demo
Sequences as Conventional Interfaces
Sequences as Conventional Interfaces

Consider two problems:
Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first n Fibonacci numbers.
Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first $n$ Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.
Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first $n$ Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.
Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first $n$ Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

enumerate naturals:
Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first $n$ Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

Enumerate naturals: $1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11$. 
Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first n Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

enumerate naturals: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11.

map fib:
Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first n Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

enumerate naturals: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11.

map fib: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55.
Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first n Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

enumerate naturals: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11.

map fib: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55.

filter iseven:
Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first $n$ Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

enumerate naturals:  1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11.

map fib:  0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55.

filter iseven:
Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first n Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

enumerate naturals: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11.

map fib: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55.

filter iseven: 0, 2, 8, 34, .
Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first n Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

enumerate naturals:  
1, 2, 3, 4, 5, 6, 7,  8,  9, 10, 11.

map fib: 
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55.

filter iseven: 
0, 2, 8, 34.

accumulate sum:
Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first n Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

enumerate naturals: \[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11.\]

map fib: \[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55.\]

filter iseven: \[0, 2, 8, 34, .\]

accumulate sum: \[., ., ., ., ., 44.\]
Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first $n$ Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.
Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first n Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.
Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first \( n \) Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

enumerate words:
Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first n Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

enumerate words: 'University', 'of', 'California', 'Berkeley'
Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first n Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

enumerate words: 'University', 'of', 'California', 'Berkeley'

filter iscap:
Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first \( n \) Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

```python
enumerate words: 'University', 'of', 'California', 'Berkeley'
```

```python
filter iscap:
```
Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first n Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

```python
enumerate words: 'University', 'of', 'California', 'Berkeley'
filter iscap: 'University', 'California', 'Berkeley'
```
Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first n Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

```python
enumerate words:  'University', 'of', 'California', 'Berkeley'

filter iscap:    'University', 'California', 'Berkeley'

map first:
```
Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first n Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

```python
enumerate words: 'University', 'of', 'California', 'Berkeley'

filter iscap: 'University', 'California', 'Berkeley'

map first: 'U', 'C', 'B'
```
Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first n Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

```python
enumerate words: 'University', 'of', 'California', 'Berkeley'

filter iscap: 'University', 'California', 'Berkeley'

map first: 'U', 'C', 'B'

accumulate tuple:
```
Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first n Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

enumerate words: 'University', 'of', 'California', 'Berkeley'

filter iscap: 'University', 'California', 'Berkeley'

map first: 'U', 'C', 'B'

accumulate tuple: ('U', 'C', 'B')
Mapping a Function over a Sequence

Apply a function to each element of the sequence
Mapping a Function over a Sequence

Apply a function to each element of the sequence

```python
>>> alternates = (-1, 2, -3, 4, -5)
```
Mapping a Function over a Sequence

Apply a function to each element of the sequence

```python
>>> alternates = (-1, 2, -3, 4, -5)
>>> tuple(map(abs, alternates))
```

Mapping a Function over a Sequence

Apply a function to each element of the sequence

```python
>>> alternates = (-1, 2, -3, 4, -5)
>>> tuple(map(abs, alternates))
(1, 2, 3, 4, 5)
```
Mapping a Function over a Sequence

Apply a function to each element of the sequence

```python
>>> alternates = (-1, 2, -3, 4, -5)

>>> tuple(map(abs, alternates))
(1, 2, 3, 4, 5)
```

The returned value of `map` is an iterable map object
Mapping a Function over a Sequence

Apply a function to each element of the sequence

```python
>>> alternates = (-1, 2, -3, 4, -5)
```

```python
>>> tuple(map(abs, alternates))
(1, 2, 3, 4, 5)
```

The returned value of `map` is an iterable map object

A constructor for the built-in map type
Mapping a Function over a Sequence

Apply a function to each element of the sequence

```python
>>> alternates = (-1, 2, -3, 4, -5)
>>> tuple(map(abs, alternates))
(1, 2, 3, 4, 5)
```

The returned value of `map` is an iterable map object

A constructor for the built-in map type

The returned value of `filter` is an iterable filter object
Mapping a Function over a Sequence

Apply a function to each element of the sequence

```python
>>> alternates = (-1, 2, -3, 4, -5)
>>> tuple(map(abs, alternates))
(1, 2, 3, 4, 5)
```

The returned value of `map` is an iterable map object

A constructor for the built-in map type

The returned value of `filter` is an iterable filter object

Demo
Accumulation and Iterable Values
Accumulation and Iterable Values

Iterable objects give access to some elements in order.
Accumulation and Iterable Values

Iterable objects give access to some elements in order.

Python-specific construct; less specific than a sequence
Accumulation and Iterable Values

Iterable objects give access to some elements in order.

Python-specific construct; less specific than a sequence

Many built-in functions take iterable objects as argument.
Accumulation and Iterable Values

Iterable objects give access to some elements in order.

Python-specific construct; less specific than a sequence

Many built-in functions take iterable objects as argument.

tuple Return a tuple containing the elements
Accumulation and Iterable Values

Iterable objects give access to some elements in order.
Python-specific construct; less specific than a sequence
Many built-in functions take iterable objects as argument.

tuple  Return a tuple containing the elements
sum    Return the sum of the elements
Accumulation and Iterable Values

Iterable objects give access to some elements in order.

Python-specific construct; less specific than a sequence

Many built-in functions take iterable objects as argument.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tuple</td>
<td>Return a tuple containing the elements</td>
</tr>
<tr>
<td>sum</td>
<td>Return the sum of the elements</td>
</tr>
<tr>
<td>min</td>
<td>Return the minimum of the elements</td>
</tr>
</tbody>
</table>
Accumulation and Iterable Values

Iterable objects give access to some elements in order.

Python-specific construct; less specific than a sequence

Many built-in functions take iterable objects as argument.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tuple</td>
<td>Return a tuple containing the elements</td>
</tr>
<tr>
<td>sum</td>
<td>Return the sum of the elements</td>
</tr>
<tr>
<td>min</td>
<td>Return the minimum of the elements</td>
</tr>
<tr>
<td>max</td>
<td>Return the maximum of the elements</td>
</tr>
</tbody>
</table>
Accumulation and Iterable Values

Iterable objects give access to some elements in order.

Python-specific construct; less specific than a sequence

Many built-in functions take iterable objects as argument.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tuple</td>
<td>Return a tuple containing the elements</td>
</tr>
<tr>
<td>sum</td>
<td>Return the sum of the elements</td>
</tr>
<tr>
<td>min</td>
<td>Return the minimum of the elements</td>
</tr>
<tr>
<td>max</td>
<td>Return the maximum of the elements</td>
</tr>
</tbody>
</table>

For statements also operate on iterable values.
Accumulation and Iterable Values

Iterable objects give access to some elements in order.

Python-specific construct; less specific than a sequence

Many built-in functions take iterable objects as argument.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tuple</code></td>
<td>Return a tuple containing the elements</td>
</tr>
<tr>
<td><code>sum</code></td>
<td>Return the sum of the elements</td>
</tr>
<tr>
<td><code>min</code></td>
<td>Return the minimum of the elements</td>
</tr>
<tr>
<td><code>max</code></td>
<td>Return the maximum of the elements</td>
</tr>
</tbody>
</table>

For statements also operate on iterable values.
Generator Expressions

One large expression that evaluates to an iterable object
Generator Expressions

One large expression that evaluates to an iterable object

\[
\langle\text{map exp}\rangle \text{ for } \langle\text{name}\rangle \text{ in } \langle\text{iter exp}\rangle \text{ if } \langle\text{filter exp}\rangle
\]
Generator Expressions

One large expression that evaluates to an iterable object

\[(\text{<map exp> for <name> in <iter exp> if <filter exp>})\]

- Evaluates to an iterable object.
Generator Expressions

One large expression that evaluates to an iterable object

\[
\langle\text{map exp}\rangle \text{ for } \langle\text{name}\rangle \text{ in } \langle\text{iter exp}\rangle \text{ if } \langle\text{filter exp}\rangle
\]

- Evaluates to an iterable object.
- \langle\text{iter exp}\rangle is evaluated when the generator expression is evaluated.
Generator Expressions

One large expression that evaluates to an iterable object

\[
(<\text{map exp}> \text{ for } <\text{name}> \text{ in } <\text{iter exp}> \text{ if } <\text{filter exp}>)
\]

- Evaluates to an iterable object.
- \(<\text{iter exp}>\) is evaluated when the generator expression is evaluated.
- Remaining expressions are evaluated when elements are accessed.
Generator Expressions

One large expression that evaluates to an iterable object

\[(\text{<map exp> for <name> in <iter exp> if <filter exp>})\]

- Evaluates to an iterable object.
- \texttt{<iter exp>} is evaluated when the generator expression is evaluated.
- Remaining expressions are evaluated when elements are accessed.

Short version: \[(\text{<map exp> for <name> in <iter exp>})\]
Generator Expressions

One large expression that evaluates to an iterable object

\[(\text{<map exp> for <name> in <iter exp> if <filter exp>})\]

- Evaluates to an iterable object.
- \(<\text{iter exp}>\) is evaluated when the generator expression is evaluated.
- Remaining expressions are evaluated when elements are accessed.

Short version: \((\text{<map exp> for <name> in <iter exp>})\)

Precise evaluation rule introduced in Chapter 4.
Generator Expressions

One large expression that evaluates to an iterable object

\[(\text{map exp} \text{ for } \text{name} \text{ in } \text{iter exp} \text{ if } \text{filter exp})\]

- Evaluates to an iterable object.
- \text{iter exp} is evaluated when the generator expression is evaluated.
- Remaining expressions are evaluated when elements are accessed.

Short version: \((\text{map exp} \text{ for } \text{name} \text{ in } \text{iter exp})\)

Precise evaluation rule introduced in Chapter 4.

Demo
Reducing a Sequence
Reducing a Sequence

Reduce is a higher-order generalization of max, min, & sum.
Reducing a Sequence

Reduce is a higher-order generalization of max, min, & sum.

```python
>>> from operator import mul
```
Reducing a Sequence

Reduce is a higher-order generalization of max, min, & sum.

```python
>>> from operator import mul
>>> from functools import reduce
```
Reducing a Sequence

Reduce is a higher-order generalization of max, min, & sum.

```python
>>> from operator import mul
>>> from functools import reduce
>>> reduce(mul, (1, 2, 3, 4, 5))
```

240
Reducing a Sequence

Reduce is a higher-order generalization of max, min, & sum.

```python
>>> from operator import mul
>>> from functools import reduce
>>> reduce(mul, (1, 2, 3, 4, 5))
120
```
Reducing a Sequence

Reduce is a higher-order generalization of max, min, & sum.

```python
>>> from operator import mul

>>> from functools import reduce

>>> reduce(mul, (1, 2, 3, 4, 5))
120
```

First argument:
A two-argument function
Reducing a Sequence

Reduce is a higher-order generalization of max, min, & sum.

```python
>>> from operator import mul
>>> from functools import reduce
>>> reduce(mul, (1, 2, 3, 4, 5))
120
```

First argument: A two-argument function
Second argument: an iterable object
Reducing a Sequence

Reduce is a higher-order generalization of max, min, & sum.

```python
>>> from operator import mul
>>> from functools import reduce

>>> reduce(mul, (1, 2, 3, 4, 5))
120
```

First argument: A two-argument function

Second argument: an iterable object

Like accumulate from Homework 2, but with iterable objects