Announcements
Announcements

• Homework 3 deadline extended to Wednesday 10/2 @ 11:59pm.
Announcements

• Homework 3 deadline extended to Wednesday 10/2 @ 11:59pm.

• Optional Hog strategy contest due Thursday 10/3 @ 11:59pm.
Announcements

• Homework 3 deadline extended to Wednesday 10/2 @ 11:59pm.

• Optional Hog strategy contest due Thursday 10/3 @ 11:59pm.

• Homework 4 due Tuesday 10/8 @ 11:59pm.
Announcements

• Homework 3 deadline extended to Wednesday 10/2 @ 11:59pm.

• Optional Hog strategy contest due Thursday 10/3 @ 11:59pm.

• Homework 4 due Tuesday 10/8 @ 11:59pm.

• Project 2 due Thursday 10/10 @ 11:59pm.
Announcements

• Homework 3 deadline extended to Wednesday 10/2 @ 11:59pm.

• Optional Hog strategy contest due Thursday 10/3 @ 11:59pm.

• Homework 4 due Tuesday 10/8 @ 11:59pm.

• Project 2 due Thursday 10/10 @ 11:59pm.

• Guerrilla Section 2 this Saturday 10/5 & Sunday 10/6 10am–1pm in Soda.
Announcements

• Homework 3 deadline extended to Wednesday 10/2 @ 11:59pm.

• Optional Hog strategy contest due Thursday 10/3 @ 11:59pm.

• Homework 4 due Tuesday 10/8 @ 11:59pm.

• Project 2 due Thursday 10/10 @ 11:59pm.

• Guerrilla Section 2 this Saturday 10/5 & Sunday 10/6 10am–1pm in Soda.

  ▪ Topics: Data abstraction, sequences, and non-local assignment.
Announcements

• Homework 3 deadline extended to Wednesday 10/2 @ 11:59pm.

• Optional Hog strategy contest due Thursday 10/3 @ 11:59pm.

• Homework 4 due Tuesday 10/8 @ 11:59pm.

• Project 2 due Thursday 10/10 @ 11:59pm.

• Guerrilla Section 2 this Saturday 10/5 & Sunday 10/6 10am–1pm in Soda.

  • Topics: Data abstraction, sequences, and non-local assignment.

  • Please RSVP on Piazza!
Announcements

• Homework 3 deadline extended to Wednesday 10/2 @ 11:59pm.

• Optional Hog strategy contest due Thursday 10/3 @ 11:59pm.

• Homework 4 due Tuesday 10/8 @ 11:59pm.

• Project 2 due Thursday 10/10 @ 11:59pm.

• Guerrilla Section 2 this Saturday 10/5 & Sunday 10/6 10am–1pm in Soda.
  ▪ Topics: Data abstraction, sequences, and non-local assignment.
  ▪ Please RSVP on Piazza!

• Guest lecture on Wednesday 10/9, Peter Norvig on Natural Language Processing in Python.
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

```python
>>> '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-quoted 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
claims, Readability counts.
Read more: import this."""
'The Zen of Python\nclaims, Readability counts.\nRead more: import this.'
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
claims, Readability counts.
Read more: import this."""
'The Zen of Python
claims, Readability counts.
Read more: import this.'
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
claims, Readability counts.
Read more: import this.""
'The Zen of Python
claims, Readability counts.
Read more: import this.'

A backslash "escapes" the following character

"Line feed" character represents a new line

Single-quoted and double-quoted strings are equivalent
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

>>> 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.

An element of a string is itself a string, but with only one character!
Strings are Sequences

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

An element of a string is itself a string, but with only one character!

**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.

(Demo)
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

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

>>> 234 in (1, 2, 3, 4, 5)
False
String Membership Differs from Other Sequence Types

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

```python
>>> 'here' in "Where's Waldo?"
True
>>> 234 in (1, 2, 3, 4, 5)
False
```

Why? Working with strings, we usually care about words more than characters.
String Membership Differs from Other Sequence Types

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

```python
>>> 'here' in "Where's Waldo?"
True
>>> 234 in (1, 2, 3, 4, 5)
False
```

Why? Working with strings, we usually care about words more than 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
>>> 234 in (1, 2, 3, 4, 5)
False
```

Why? Working with strings, we usually care about words more than 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
>>> 234 in (1, 2, 3, 4, 5)
False
```

Why? Working with strings, we usually care about words more than 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
>>> 234 in (1, 2, 3, 4, 5)
False
```

Why? Working with strings, we usually care about words more than 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
Encoding Strings
### Representing Strings: the ASCII Standard

**American Standard Code for Information Interchange**

#### ASCII Code Chart

<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>DEL</td>
</tr>
</tbody>
</table>


## 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>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
</tbody>
</table>

8 rows: 3 bits
## Representing Strings: the ASCII Standard

**American Standard Code for Information Interchange**

### ASCII Code Chart

<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>`</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>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>

8 rows: 3 bits

16 columns: 4 bits
# 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>

- Layout was chosen to support sorting by character code
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>`</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>
<td></td>
</tr>
</tbody>
</table>

- 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>
</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>

- 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></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>

- 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

"Line feed" (\n)
### 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>&quot;Bell&quot; (\a)</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>DEL</td>
<td></td>
</tr>
</tbody>
</table>

- **8 rows: 3 bits**
- **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

![ASCII Code Chart](image-url)
Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0 | NULL | START | END | SPACE | RETURN | NL | HT | LF | VT | FF | CR | SO | SI | DLE | DC1 | DC2 | DC3 | DC4 | NAK | SYN | ETB | CAN | EM | SUB | ESC | FS | GS | RS | US |
| 1 | ! | " | # | $ | % | & | ' | ( | ) | * | + | , | - | . | / | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 2 | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | [ | ] | ^ | _ |
| 3 | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | { | | | |

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
Representing Strings: the Unicode Standard

- 109,000 characters
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
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
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

[Image: 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
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: 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 those integers and bytes
Representing Strings: UTF-8 Encoding

UTF (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and integers

UTF-8: Correspondence between those integers 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 those integers and bytes

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

bytes
Representing Strings: UTF-8 Encoding

UTF (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and integers

UTF-8: Correspondence between those integers 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 those integers 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 those integers 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>
</tbody>
</table>
Representing Strings: UTF-8 Encoding

UTF (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and integers

UTF-8: Correspondence between those integers 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 those integers 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 those integers 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 those integers and bytes

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

\[
\begin{array}{c|c}
\text{bytes} & \text{integers} \\
0000000 & 0 \\
00000001 & 1 \\
00000010 & 2 \\
00000011 & 3 \\
\end{array}
\]

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

In Python: `string` length is measured 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 those integers 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 is measured in characters, `bytes` length in bytes.

(Demo)
Sequence Processing
Sequence Processing
Sequence Processing

Consider two problems:
Consider two problems:

- Sum the even members of the first n Fibonacci numbers.
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.
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.
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:
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.
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:

Sequence Processing
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.
Sequence Processing

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 even:
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.

**Sequence Processing**

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 even:
Sequence Processing

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 even: 0, 2, 8, 34, .
Sequence Processing

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 even: 0, 2, 8, 34, .

accumulate sum:
Sequence Processing

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 even: 0, 2, 8, 34, .
accumulate sum: ,, ,, ,, ,, =44
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.
Sequence Processing

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.
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.

e numerate words:
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'
Sequence Processing

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.

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

filter capitalized:
```
Sequence Processing

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 capitalized:
Sequence Processing

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 capitalized: 'University', 'California', 'Berkeley'
Sequence Processing

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 capitalized: 'University', 'California', 'Berkeley'

map first:
Sequence Processing

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 capitalized:  
'University', 'California', 'Berkeley'

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

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 capitalized:  
'University', 'California', 'Berkeley'

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

accumulate tuple:
Sequence Processing

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 capitalized:  '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)
>>> 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)
```

```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

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)
Iteration and Accumulation
Iterable Values and Accumulation
Iterable Values and Accumulation

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

*Iterable* objects give access to their elements in order.

Similar to a sequence, but does not always allow element selection or have finite length.
Iterable Values and Accumulation

*Iterable* objects give access to their elements in order.

Similar to a sequence, but does not always allow element selection or have finite length.

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

*Iterable* objects give access to their elements in order.

Similar to a sequence, but does not always allow element selection or have finite length.

Many built-in functions take iterable objects as argument.

```
tuple  Return a tuple containing the elements
```
Iterable Values and Accumulation

*Iterable* objects give access to their elements in order.

Similar to a sequence, but does not always allow element selection or have finite length.

Many built-in functions take iterable objects as argument.

- **tup**le: Return a tuple containing the elements
- **sum**: Return the sum of the elements
Iterable Values and Accumulation

*Iterable* objects give access to their elements in order.

Similar to a sequence, but does not always allow element selection or have finite length.

Many built-in functions take iterable objects as argument.

- **tuple**: Return a tuple containing the elements
- **sum**: Return the sum of the elements
- **min**: Return the minimum of the elements
**Iterable Values and Accumulation**

*Iterable* objects give access to their elements in order.

Similar to a sequence, but does not always allow element selection or have finite length.

Many built-in functions take iterable objects as argument.

- **tuple**: Return a tuple containing the elements
- **sum**: Return the sum of the elements
- **min**: Return the minimum of the elements
- **max**: Return the maximum of the elements
Iterable Values and Accumulation

*Iterable* objects give access to their elements in order.

Similar to a sequence, but does not always allow element selection or have finite length.

Many built-in functions take iterable objects as argument.

- **tuple**  
  Return a tuple containing the elements

- **sum**  
  Return the sum of the elements

- **min**  
  Return the minimum of the elements

- **max**  
  Return the maximum of the elements

For statements also operate on iterable values.
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.

>>> 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))
18
```
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
```

Similar to accumulate from Homework 2, but with iterable objects.
Generator Expressions

One large expression that evaluates to an iterable object
Generator Expressions

One large expression that evaluates to an iterable object

$$\left(\langle\text{map exp}\rangle\ \text{for}\ \langle\text{name}\rangle\ \text{in}\ \langle\text{iter exp}\rangle\ \text{if}\ \langle\text{filter exp}\rangle\right)$$
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.
Generator Expressions

One large expression that evaluates to an iterable object

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

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

One large expression that evaluates to an iterable object

(\texttt{<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.
Generator Expressions

One large expression that evaluates to an iterable object

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

• 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: \[
\left(\text{<map exp> for <name> in <iter exp>}\right)
\]
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>})

    (Demo)