Sequence Iteration

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

Name bound in the first frame of the current environment

```python
if elem == value:
    total = total + 1
return total
```

For Statement Execution Procedure

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

```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
```

The Range Type

A range is a sequence of consecutive integers,*

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

Length: ending value - starting value

Element selection: starting value + index

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

```python
>>> digits = (1, 8, 2, 8)
>>> 2 in digits
True
>>> 182B not in digits
True
```

Slicing.

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

---

String Literals Have Three Forms

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

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

A backslash "escapes" the following character

"Line feed" character represents a new line

String Membership Differs from Other Sequence Types

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

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

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

The count method also matches substrings

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

The number of non-overlapping occurrences of a substring

---

Strings are Sequences

```python
>>> city = 'Berkeley'
>>> len(city)
8
```  

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'
```  

String arithmetic is similar to tuple arithmetic

---

Representing Strings: the ASCII Standard

- **American Standard Code for Information Interchange**
- **"Bell"**
- **"Line feed"**
- **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 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**

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

---

Demo
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

Demonstration

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`: ▲ ▲ ▲ ▲ ▲

`accumulate sum`: ,, ,, ,, ,, ,, 44.

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

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
- `min`: Return the minimum of the elements
- `max`: Return the maximum of the elements

For statements also operate on iterable values.

Generator Expressions

One large expression that evaluates to an iterable object

```
(<map exp> for <name> in <iter exp> if <filter exp>)
```

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

Short version: `(<map exp> for <name> in <iter exp>)`

Precise evaluation rule introduced in Chapter 4.
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
```

Like accumulate from Homework 2, but with iterable objects