Working with Lists

```python
>>> digits = [1, 8, 2, 8]
>>> digits = [2//2, 2*2*2, 2, 2*2*2]

The number of elements
>>> len(digits)
4

An element selected by its index
>>> digits[3]
8

Concatenation and repetition
>>> [2, 7] + digits + 2
[2, 7, 1, 8, 2, 8, 1, 8, 2, 8]
>>> add([2, 7], mul(digits, 2))
[7, 7, 1, 8, 1, 8, 1, 8, 2, 8]

Nested lists
>>> pairs = [[10, 20], [30, 40]]
>>> pairs[1]
[30, 40]
>>> pairs[1][0]
30
```

Containers

Built-in operators for testing whether an element appears in a compound value

```python
>>> digits = [1, 8, 2, 8]
>>> 1 in digits
True
>>> 8 in digits
True
>>> 5 not in digits
True
>>> not(5 in digits)
True
```

Containers

For Statements

```python
def count(s, value):
    total = 0
    for element in s:
        if element == value:
            total += 1
    return total
```
For Statement Execution Procedure

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

1. Evaluate the header <expression>, which must yield an iterable value (a sequence)
2. For each element in that sequence, in order:
   A. Bind <name> to that element in the current frame
   B. Execute the <suite>

Sequence Unpacking in For Statements

A sequence of fixed-length sequences

>>> pairs = [[1, 2], [2, 2], [3, 3], [4, 4]]

A name for each element in a fixed-length sequence

>>> same_count = 0

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

>>> 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.

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

Length: ending value - starting value

Element selection: starting value + index

>>> list(range(-2, 2))
[-2, -1, 0, 1]

>>> list(range(4))
[0, 1, 2, 3]

Ranges can actually represent more general integer sequences.

Recursive Sums

Sum (recursively)

def mysum(L):
    if L == []:
        return 0
    else:
        return L[0] + mysum(L[1:])**

# ——— DRILL ———
# Write an iterative function that takes as input
# integer “n” and returns the sum of the first “n”
# integers: sum(5) returns 1+2+3+4+5

def sum_iter(n):
    sum = 0
    for i in range(0, n+1):
        sum = sum + i
    return( sum )

# ——— DRILL ———
# Write an iterative function that takes as input
# integer “n” and returns the sum of the first “n”
# integers: sum(5) returns 1+2+3+4+5

def sum_iter(n):
    sum = 0
    for i in range(0, n+1):
        sum = sum + i
    return( sum )
# ——— DRILL ———
# Write a recursive function that takes as input 
# integer “n” and returns the sum of the first “n” 
# integers: sum(5) returns 1+2+3+4+5

def sum_rec(n):
    if(n == 0):
        return(0)
    else:
        return n + sum_rec(n-1)

List Comprehensions

A combined expression that evaluates to a list using this evaluation procedure:
1. Add a new frame with the current frame as its parent
2. Create an empty result list that is the value of the expression
3. For each element in the iterable value of <iter exp>:
   A. Bind <name> to that element in the new frame from step 1
   B. If <filter exp> evaluates to a true value, then add the value of <map exp> to the result list

Strings are an Abstraction

Representing data:

'200'  
'1.2e-5'  
'value'  
'[1, 2]'

Representing language:

"And, as imagination bodies forth
The forms of things unknown, 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

---

 repayment

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

(Demo)

Reversing a String
Reversing a List (recursively)

\[
\text{reverse("ward") = "draw"}
\]
\[
\text{reverse("ward") = reverse("ard") + "w"}
\]
\[
\text{reverse("ard") = reverse("rd") + "a"}
\]
\[
\text{reverse("rd") = reverse("d") + "r"}
\]
\[
\text{reverse("d") = "d"}
\]

def reverse(s):
    if len(s) == 1:
        return s
    else:
        return reverse(s[1:]) + s[0]