Lists

Working with Lists

```python
>>> digits = [1, 8, 2, 8]
The number of elements
>>> len(digits)
4
An element selected by its index
>>> digits[3]
8
Concatenation and repetition
[2, 7, 1, 8, 2, 8, 1, 8, 2, 8, 2, 8]
Nested lists
>>> pairs = [[10, 20], [30, 40]]
>>> pairs[1]
[30, 40]
>>> pairs[0][0]
10
>>> [2, 7] + digits * 2
[2, 7, 1, 8, 2, 8, 1, 8, 2, 8]
```

Containers

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

Sequence iteration

For Statements

```python
def count(s, value):
    total = 0
    for element in s:
        if element == value:
            total = 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

```python
>>> pairs = [[1, 2, 3, 2], [1, 2], [4, 6]]
>>> same_count = 0
>>> for x, y in pairs:
...     if x == y:
...         same_count += 1

>>> same_count
type: int
value: 2
```

A sequence of fixed-length sequences

A name for each element in a fixed-length sequence

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

Ranges

A range is a sequence of consecutive integers:

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

The Range Type

Length: ending value - starting value
Element selection: starting value + index

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

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

A range is a sequence of consecutive integers.

Ranges can actually represent more general integer sequences.

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

```python
>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p']

>>> [letters[i] for i in [3, 4, 6, 8]]
['d', 'e', 'm', 'o']
```
Strings are an Abstraction

Representing data:

```python
'200'   \"1.2e-5\"   \"False\"   \"[1, 2]\"
```

Representing language:

```python
"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:

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

String Literals Have Three Forms

```python
>>> 'I am string!
'I am string!'
>>> "I've got an apostrophe"
"I've got an apostrophe"
>>> '\n'
\\
>>> "The Zen of Python
claims, Readability counts.
Read more: import this."
'The Zen of Python
claims, Readability counts.
Read more: import this.'
```

Line feed character represents a new line
A backslash "escapes" the following character
"Line feed" character represents a new line

Single-quoted and double-quoted strings are equivalent

Dictionaries

```python
{'Dem': 0}
```

Limitations on Dictionaries

Dictionaries are unordered collections of key-value pairs.

Dictionary keys do have two restrictions:

* A key of a dictionary cannot be a list or a dictionary (or any mutable type)

* Two keys cannot be equal; There can be at most one value for a given key

This first restriction is tied to Python's underlying implementation of dictionaries

The second restriction is part of the dictionary abstraction

If you want to associate multiple values with a key, store them all in a sequence value

Data Abstraction
Data Abstraction

- Compound values combine other values together
- A date: a year, a month, and a day
- A geographic position: latitude and longitude

Data abstraction lets us manipulate compound values as units

- Isolate two parts of any program that uses data:
  - How data are represented (as parts)
  - How data are manipulated (as units)

Data abstraction: A methodology by which functions enforce an abstraction barrier between representation and use

Rational Numbers

<table>
<thead>
<tr>
<th>numerator</th>
<th>denominator</th>
</tr>
</thead>
</table>

Exact representation of fractions

A pair of integers

As soon as division occurs, the exact representation may be lost! (Demo)

Assume we can compose and decompose rational numbers:

- Constructor:
  - rational(n, d) returns a rational number x
- Selectors:
  - numer(x) returns the numerator of x
  - denom(x) returns the denominator of x

Rational Number Arithmetic

\[
\begin{array}{c|c|c|c|c}
\text{num} & \times & \text{num} & \times & \text{den} \\
\hline
\text{num} & \times & \text{num} & \times & \text{den} \\
\hline
\end{array}
\]

Example

Rational Number Arithmetic Implementation

```python
def mul_rational(x, y):
    return rational(numer(x) * numer(y), denom(x) * denom(y))

def add_rational(x, y):
    nx, dx = numer(x), denom(x)
    ny, dy = numer(y), denom(y)
    return rational(nx * dy + ny * dx, dx * dy)

print_rational(rational(3, 4) + rational(2, 5))
```

Representing Pairs Using Lists

```python
>>> pair = [1, 2]
A list literal: 
Comma-separated expressions in brackets
```

```python
>>> x, y = pair
“Unpacking” a list
```

```python
>>> x
1
```

```python
>>> y
2
```

```python
>>> pair[0]
Element selection using the selection operator
```

```python
>>> pair[1]
```

```python
>>> from operator import getitem
```

```python
>>> getitem(pair, 0)
```

```python
>>> getitem(pair, 1)
Element selection function
```
Representing Rational Numbers

```python
def rational(x):
    """Construct a rational number that represents n/d."""
    return (n, d)

def numer(x):
    """Return the numerator of rational number X."""
    return x[0]

def denom(x):
    """Return the denominator of rational number X."""
    return x[1]
```

Reducing to Lowest Terms

```python
def rational(n, d):
    """Construct a rational that represents n/d in lowest terms."""
    g = gcd(n, d)
    return [n // g, d // g]
```

Example:

```
3   5   7
--- * --- * ---
2   3   2
```

```
15   1
--- * ---
6   1
```

```
15   1/25   1
--- * --- = ---
58   1/26   2
```

Abstraction Barriers

```plaintext
<table>
<thead>
<tr>
<th>Parts of the program that...</th>
<th>Treat rationals as...</th>
<th>Using...</th>
</tr>
</thead>
<tbody>
<tr>
<td>use rational numbers to perform computation</td>
<td>whole data values</td>
<td>add/rational, mul_rational, rationalize, eq, repr_rational</td>
</tr>
</tbody>
</table>

Create rationals or implement rational operations

Implement selectors and constructor for rationals

Implementation of lists

Does not use constructors

No selectors!

And no constructor!
```

Data Representations
What are Data?

- We need to guarantee that constructor and selector functions work together to specify the right behavior.
- Behavior condition: If we construct rational number \( x \) from numerator \( n \) and denominator \( d \), then \( \text{num}(x)/\text{denom}(x) \) must equal \( n/d \).
- Data abstraction uses selectors and constructors to define behavior.
- If behavior conditions are met, then the representation is valid.

You can recognize an abstract data representation by its behavior.

```
def rational(n, d):
    def select(name):
        if name == 'n':
            return n
        elif name == 'd':
            return d
    return select

def numer(x):
    return x('n')

def denom(x):
    return x('d')
```

This function represents a rational number.

Rationals Implemented as Functions

Dictionaries

Dictionaries are unordered collections of key-value pairs.

Dictionaries have two restrictions:

- A key of a dictionary cannot be a list or a dictionary (or any mutable type).
- Two keys cannot be equal; there can be at most one value for a given key.

This first restriction is tied to Python's underlying implementation of dictionaries.

The second restriction is part of the dictionary abstraction.

If you want to associate multiple values with a key, store them all in a sequence value.