Lists

Working with Lists

The number of elements

An element selected by its index

Concatenation and repetition

Nested lists

Concatenation and repetition

Containers

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

For Statements

Sequence Iteration

Ranges

For Statement Execution Procedure

Sequence Unpacking in For Statements
The Range Type

A range is a sequence of consecutive integers. 

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

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

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 \(<\text{iter exp}>\):
   A. Bind \(<\text{name}>\) to that element in the new frame from step 1
   B. If \(<\text{filter exp}>\) evaluates to a true value, then add the value of \(<\text{map exp}>\) to the result list

List Comprehensions

\[ \text{letters} = ['a', 'b', 'c', 'd', 'e', 'f', 'm', 'n', 'o', 'p'] \]
\[ \text{[letters}\[i\] \text{for } i \text{ in } [3, 4, 6, 8]} \]
\[ \text{['d', 'e', 'm', 'o']} \]

Strings

Strings are an Abstraction

Representing data:

'200'  '1.2e-5'  'False'  '[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)' 

Strings Literals Have Three Forms

--- 'I am string'
'I am string'
--- 'I've got an apostrophe'
'I've got an apostrophe'
--- '఍అ'
'఍అ'
--- "The Zen of Python
claims, Readability counts.
Read more: import this." 

'Delimiting "quotes" the following character'
'single quote' represents a new line
'A backslash "escapes" the following character'

Dictionaries

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

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

Rational Numbers

\[ \text{numerator} \]
\[ \text{denominator} \]

Exact representation of fractions

A pair of integers

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

\[ x = \text{rational}(n, d) \] returns a rational number \( x \)
\[ \text{numer}(x) \] returns the numerator of \( x \)
\[ \text{denom}(x) \] returns the denominator of \( x \)

Representing Rational Numbers

\[ \text{numerator}(x) \]
\[ \text{denominator}(x) \]

These functions implement an abstract representation for rational numbers

Reducing to Lowest Terms

\[ \frac{\text{numerator}(x)}{\text{denominator}(x)} \]

Example:

\[ \frac{3}{2} \times \frac{5}{5} = 1 \]
\[ \frac{2}{3} + \frac{1}{18} = 1 \]

\[ \frac{18}{18} + \frac{1}{18} = \frac{19}{18} \]

From fractions \( \text{gcd} \): Greatest common division

\[ \text{rational}(x, y) \] generates a rational that represents \( x/y \) in lowest terms

\[ \text{gcd}(n, d) \] returns \( \text{gcd}(n, d) \)

Abstraction Barriers

Rational Number Arithmetic

\[ \frac{3}{2} + \frac{3}{5} = \frac{15}{10} \]

\[ \frac{5}{3} - \frac{2}{5} = \frac{13}{15} \]

\[ \frac{1}{18} + \frac{1}{18} = \frac{2}{18} \]

\[ \frac{18}{18} + \frac{1}{18} = \frac{19}{18} \]

Example

General Form

\[ \frac{nx}{dx} + \frac{ny}{dy} = \frac{nx dy + ny dx}{dx dy} \]

Example

Pairs

\[ \text{pair} = [1, 2] \]

\[ \text{pair}[1] \]

\[ \text{pair}[0] \]

\[ \text{getitem}(\text{pair}, 0) \]

\[ \text{getitem}(\text{pair}, 1) \]

Representing Pairs Using Lists

\[ \text{pair} = [1, 2] \]

\[ \text{getitem}(\text{pair}, 0) \]

\[ \text{getitem}(\text{pair}, 1) \]

Element selection function

\[ \text{rational}(n, d) \]

Greatest common divisor

\[ \text{gcd}(n, d) \]
Abstraction Barriers

<table>
<thead>
<tr>
<th>Topic</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parts of the program that...</td>
<td>Treat rationals as...</td>
</tr>
<tr>
<td>Use rational numbers to perform computation</td>
<td>whole data values</td>
</tr>
<tr>
<td>Create rational or implement rational operations</td>
<td>numerator and denominator</td>
</tr>
<tr>
<td>Implementation of lists</td>
<td>two-element lists</td>
</tr>
</tbody>
</table>

Treat rationals as whole data values |

Using... |

Create rational or implement rational operations |

Rationals implemented as functions |

Violating Abstraction Barriers

```python
add_rational( [1, 2], [1, 4] )
def divide_rational(x, y):
    return [ x[0] * y[1], x[1] * y[0] ]
```

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 numer(x)/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 abstract data representations by its behavior.

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.

```python
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')
x = rational(3, 8)
numer(x)
```