Containers
Announcements
Box-and-Pointer Notation
The Closure Property of Data Types

• A method for combining data values satisfies the closure property if:
  The result of combination can itself be combined using the same method

• Closure is powerful because it permits us to create hierarchical structures

• Hierarchical structures are made up of parts, which themselves are made up of parts, and so on

Lists can contain lists as elements (in addition to anything else)
Box-and-Pointer Notation in Environment Diagrams

Lists are represented as a row of index-labeled adjacent boxes, one per element.

Each box either contains a primitive value or points to a compound value.

```
pair = [1, 2]
```
Box-and-Pointer Notation in Environment Diagrams

Lists are represented as a row of index-labeled adjacent boxes, one per element.

Each box either contains a primitive value or points to a compound value.

```
1 pair = [1, 2]
2 nested_list = [[1, 2], [],
3      [[3, False, None],
4      [4, lambda: 5]]]
```

```
Global frame
  pair
  nested_list

list
  0 1 2

list
  0 1 2

empty list

list
  0 1

list
  0 1

list
  0 1

list
  0 1

func λ() <line 5> [parent=Global]
```
Slicing

(Demo)
Slicing Creates New Values

digits = [1, 8, 2, 8]
start = digits[:1]
middle = digits[1:3]
end = digits[2:]
full = digits[:]

Global frame

digits
start
middle
digits
full

list
0 1 8 2 3
list
0
list
0 1 2
list
0 2 1 8
list
0 1 8 2 3

[Image of a code snippet and a Python Tutor diagram showing the slicing of a list and the creation of new values.]

python tutor.com/composingprograms.html?code=digits%20%3D%20%5B1,%208,%202,%208%5D%0Astart%20%3D%20digits%5B%3A1%5D%0Amiddle%20%3D%20digits%5B1:3%5D%0Aend%20%3D%20digits%5B2:%5D%0Afull%20%3D%20digits%5B%5D&cumulative%5D=true&curInstr%5D=5&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D
Processing Container Values
Aggregation

Several built-in functions take iterable arguments and aggregate them into a value

• **sum**(iterable[, start]) → value

  Return the sum of an iterable (not of strings) plus the value of parameter 'start' (which defaults to 0). When the iterable is empty, return start.

• **max**(iterable[, key=func]) → value
  max(a, b, c, ...[, key=func]) → value

  With a single iterable argument, return its largest item.
  With two or more arguments, return the largest argument.

• **all**(iterable) → bool

  Return True if bool(x) is True for all values x in the iterable. If the iterable is empty, return True.
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)'

(Demo)
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
Dictionaries

{"Dem": 0}
Limitations on Dictionaries

Dictionaries are 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
Dictionary Comprehensions

\{<\text{key exp}>: <\text{value exp}> \text{ for } <\text{name}> \text{ in } <\text{iter exp}> \text{ if } <\text{filter exp}>\}\n
Short version: \{<\text{key exp}>: <\text{value exp}> \text{ for } <\text{name}> \text{ in } <\text{iter exp}>\}\n
An expression that evaluates to a dictionary using this evaluation procedure:

1. Add a new frame with the current frame as its parent
2. Create an empty \textit{result dictionary} 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 to the result dictionary an entry that pairs the value of <\text{key exp}> to the value of <\text{value exp}>

\{x \times x: x \text{ for } x \text{ in } [1, 2, 3, 4, 5] \text{ if } x > 2\} \text{ evaluates to } \{9: 3, 16: 4, 25: 5\}
Example: Indexing

Implement the `index` function, which takes a sequence of `keys`, a sequence of `values`, and a two-argument `match` function. It returns a dictionary from `keys` to lists in which the list for a key `k` contains all `values v` for which `match(k, v)` is a true value.

```python
def index(keys, values, match):
    """Return a dictionary from keys k to a list of values v for which match(k, v) is a true value."

    >>> index([7, 9, 11], range(30, 50), lambda k, v: v % k == 0)
    {7: [35, 42, 49], 9: [36, 45], 11: [33, 44]}
    """
    return {k: [v for v in values if match(k, v)] for k in keys}
```