Computational Structures in Data Science

Lecture #5: Abstract Data Types

You can now register to vote by sending a text
9/23/16
UCB CS88 Fa16 L5

Computational Concepts Toolbox

- Data type: values, literals, operations,
  - Expressions, Call expression
  - Variables
  - Assignment Statement
  - Sequences: tuple, list
  - Data structures
  - Tuple assignment
  - Call Expressions
  - Function Definition Statement
  - Conditional Statement
  - Iteration: list comp, for, while

- Higher Order Functions
  - Functions as Values
  - Functions with functions as argument
  - Assignment of function values

- Higher order function patterns
  - Map, Filter, Reduce
  - Function factories – create and return functions

- Recursion
  - Linear, Tail, Tree

Administrative Issues

- Midterm: 10/07. Lecture = Study Session
- Next lecture (09/30): Research lecture, not part of midterm
- Plan for lectures online now.
- Today’s lecture relevant for project! Lots of code that I am going to skim over in lecture – useful to look up.

Errata: Higher Order Functions (cont)

- A function that returns (makes) a function

```python
def leq_maker(c):
    def leq(val):
        return val <= c
    return leq
```

```python
>>> leq_maker(3)
<function leq_maker.<locals>.<locals>.<locals>.leq at 0x1019d8c80>
>>> leq_maker(3)(4)
False
>>> filter(leq_maker(3), [0,1,2,3,4,5,6,7])
[0, 1, 2, 3]
```

Recap: Universality

- Everything that can be computed, can be computed with what you know since lecture 1.
- Well
- or poorly

Aside: lambda

- Function expression
  - “Anonymous” function creation
  - Expression, not a statement, no return or any other statement

```python
lambda <arg or arg_tuple> : <expression using args>
```

```python
inc = lambda v : v + 1
def inc(v):
    return v + 1
```
Lambda Examples

```python
>>> msort([1,2,3,4,5], lambda x: x)
[1, 2, 3, 4, 5]
>>> msort([1,2,3,4,5], lambda x: -x)
[5, 4, 3, 2, 1]
>>> msort(((2, 'hi'), (1, 'how'), (5, 'goes'), (7, 'I')), lambda x: x[0])
[(1, 'how'), (2, 'hi'), (5, 'goes'), (7, 'I')]
>>> msort(((2, 'hi'), (1, 'how'), (5, 'goes'), (7, 'I')), lambda x: len(x[1]))
[(7, 'I'), (2, 'hi'), (1, 'how'), (5, 'goes')]
```

http://cs88-website.github.io/assets/slides/adt/mersort.py

9/23/16 UCB CS88 Fa16 L5

Lambdas

```python
>>> def inc_maker(i):
...     return lambda x: x+i
...     return lambda x: x+i
... >>> inc_maker(3)
<function inc_maker.<locals>.<lambda> at 0x10073c510>
>>> inc_maker(3)(4)
7
```

```python
>>> map(lambda x: x*x, [1,2,3,4])
<map object at 0x1020950b8>
>>> list(map(lambda x: x*x, [1,2,3,4]))
[1, 4, 9, 16]
```

9/23/16 UCB CS88 Fa16 L5

C.O.R.E concepts

Compute
Operations
Provide operations on the abstract components that allow ease of use – independent of concrete representation.

Representation
Constructors and selectors that provide an abstract interface to a concrete representation.

Evaluation
Execution on a computing machine

Abstraction Barrier

Creating an Abstract Data Type

- Operations
  - Express the behavior of objects, invariants, etc
    - Implemented (abstractly) in terms of Constructors and Selectors for the object
- Representation
  - Constructors & Selectors
    - Implement the structure of the object
- An abstraction barrier violation occurs when a part of the program that can use the higher level functions uses lower level ones instead
  - At either layer of abstraction
- Abstraction barriers make programs easier to get right, maintain, and modify
  - Few changes when representation changes

Examples You have seen

- Lists
  - Constructors:
    - list(...)
  - Selectors: list[<index or slice>]
  - Operations: in, not in, +, *, len, min, max
    - Mutable ones too (but not yet)
- Tuples
  - Constructors:
    - tuple(...)
  - Selectors: tuple[<index or slice>]
  - Operations: in, not in, +, *, len, min, max
- Strings
  - Constructors:
    - str(...)
    - selectors: str[<index or slice>]
  - Operations: in, not in, +, *, len, min, max
- Range
  - Constructors:
    - range(<end>), range(<start>,<end>), range(<start>,<end>,<step>)
    - Selectors: range[<index or slice>]
  - Operations: in, not in, +, *, len, min, max

9/23/16 UCB CS88 Fa16 L5

Examples You have seen

- Lists
- Tuples
- Strings
- Range

9/23/16 UCB CS88 Fa16 L5
Example ADT: lookup table (lut)

- Unordered collection of unique key => value bindings
  - "lookup, i.e. get, the value associated with a key"
- Where does this occur?
  - Phonebook
  - Facebook friends
  - Movie listings
  - Restaurant ratings
  - Roster
  - …

lut ADT

- Constructors
  - lut(): Return an empty lut
  - lut_add(lut, key, value): Return a lut with new key => value binding
  - lut_del(lut, key): Return a lut without a binding for key
- Selectors
  - lut_get(lut, key): Return value in lut bound to key or None if none exists.
  - lut_keys(lut): Return a list of keys for bindings in lut
  - lut_values(lut): Return a list of values for bindings in lut
  - lut_items(lut): Return a list of (key, value) for bindings in lut
- Operations

The Layered Design Process

- Build the application based entirely on the ADT interface
  - Operations, Constructors and Selectors
- Build the operations entirely in ADT
  - Constructors and Selectors
  - Not the implementation of the representation
- Build the constructors and selectors on some concrete representation

A lut application (lut_app.py)

```python
from lut import *

phone_book_data = [
    ('Christine Strauch', '510-842-9235'),
    ('Frances Catal Bulosan', '932-567-3241'),
    ('Jack Chow', '617-547-0923'),
    ('Joy De Rosario', '310-912-6483'),
    ('Casey Casem', '415-432-9232'),
    ('Lydia Lu', '707-341-1254')
]

phone_book = lut_with_bindings(phone_book_data)
lut_print(phone_book)
print("Jack Chow's Number: ", lut_get(phone_book, "Jack Chow"))

print("Area codes")
area_codes = lut_map_values(phone_book, lambda x: x[0:3])
lut_print(area_codes)
```

Apps (cont)

```python
lut_sorted(new_phone_book, lambda k,v:k[0])
```

http://cs68-website.github.io/assets/slides/adt/lut_app.py
Apps (cont)

```python
def name_dist(name1, name2):
    count = max(len(name1), len(name2)) - min(len(name1), len(name2))
    for i in range(min(len(name1), len(name2))):
        if name1[i] != name2[i]:
            count += 1
    return count

lut_fuzzy_get(phone_book, "Jack", name_dist)
```

Friends App

```python
friend_data = ["Christine Strauch", "Jack Chow"],
"Christine Strauch", "Lydia Lu"],
"Jack Chow", "Christine Strauch"],
"Casey Cases", "Jack Chow"],
"Casey Cases", "Frances Catal Buloan"],
"Casey Cases", "Joy De Rosario"],
"Casey Cases", "Casey Cases"],
"Frances Catal Buloan", "Jack Chow"],
"Jack Chow", "Frances Catal Buloan"],
"Joy De Rosario", "Lydia Lu"],
"Joy De Lydia", "Jack Chow"
```

More Friends

```python
def make_friends(friends):
    friend_lut = lut()
    for (der, dee) in friends:
        old_friends = lut_get(friend_lut, der)
        new_friends = old_friends + [dee] if old_friends is not None else [dee]
        friend_lut = lut_update(friend_lut, der, new_friends)
    return friend_lut
```

Above Abstraction Barrier – lut.py

```python
def lut_with_bindings(bindings):
    """Construct lookup table with (key,val) bindings."""
    new_lut = lut()
    for k,v in bindings:
        new_lut = lut_add(new_lut, k, v)
    return new_lut
```

```python
def lut_sorted(lut, fun):
    """Return a list of (k,v) for bindings in lut sorted by <= over fun(k, v)."""
    return msort(lut_items(lut), lambda b: fun(b[0],b[1]))
```

```python
def lut_print(lut):
    """Print a representation of bindings in lut."""
    for k,v in lut_sorted(lut, lambda k,v:k):
        print(k,">" ,v)
```
Above Abstraction Barrier – lut.py

```python
def lut_with_bindings(bindings):
def lut_sorted(lut, fun):
def lut_print(lut):
def lut_map_values(lut_to_map, fun):
    return lut_with_bindings({(k, fun(v)) for k, v in lut_items(lut_to_map)})
```

Beneath the Abstraction Barrier

• How to represent a lookup table?

Repr: list of tuples (lut_tuples.py)

```python
# Constructors
def lut():
    return []
def lut_add(lut, key, value):
def lut_del(lut, key):
# Selectors
def lut_get(lut, key):
def lut_keys(lut):
def lut_values(lut):
def lut_items(lut):
```

Representation: list of tuples

```python
# Constructors
def lut():
def lut_add(lut, key, value):
def lut_del(lut, key):
keys, values = lut
key_index = keys.index(key)
return (keys[0:key_index] + keys[key_index+1:],
values[0:key_index] + values[key_index+1:])
```

Repr: tuple of lists – lut_lists.py

```python
# Constructors
def lut():
def lut_add(lut, key, value):
def lut_del(lut, key):
```

http://cs88-website.github.io/assets/slides/adt/lut_tuples.py
http://cs88-website.github.io/assets/slides/adt/lut_lists.py
# Constructors

def lut():
    return ([], [])
def lut_add(lut, key, value):
    def lut_del(lut, key):
        # Selectors
        def lut_get(lut, key):
            for k, val in zip(lut[0], lut[1]):
                if k == key:
                    return val
            return None
def lut_keys(lut):
    """Return a list of keys in lookup table lut."""
    return lut[0]

# Constructors

def lut():
    return ([], [])
def lut_add(lut, key, value):
    def lut_del(lut, key):
        # Selectors
        def lut_get(lut, key):
            for k, val in zip(lut[0], lut[1]):
                if k == key:
                    return val
            return None
def lut_keys(lut):
    """Return a list of keys in lookup table lut."""
    return lut[0]

Dictionaries

• Lists, Tuples, Strings, Range
• Dictionaries
  – Constructors:
    » dict(<list of 2-tuples>)
    » dict(<key>=<val>, ... # like kwargs
    » {<key>;<val> for <iteration expression>}
    >>> {x:y for x,y in zip(["a","b"], [1,2])}
    {"a": 1, "b": 2}
  – Selectors: <dict>.keys(), .items(), .values()
  – Operations:
    » Key in, not in, len, min, max
    » <dict>[<key>]=<val>

In Lab

• Dictionaries
• Lambdas
• Abstract Data Types
• Go build things...