Computational Structures in Data Science

UC Berkeley EECS
Adj. Ass. Prof.
Dr. Gerald Friedland

Lecture #5: More HOF, Abstract Data Types

February 16, 2018 http://inst.eecs.berkeley.edu/~cs88

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: 02/22. Lecture = Study Session
- Next lecture: Research lecture, not part of midterm
- Today's lecture relevant for project!

  Lots of code after the last lecture slide to look up and try out.

Recap: Higher Order Functions (cont)

- A function that returns (makes) a function

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

```
>>> leq_maker(3)
<function leq_maker.<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]
>>>
```


Three super important HOFS

`map(function_to_apply, list_of_inputs)`

Applies function to each element of the list

`filter(condition, list_of_inputs)`

Returns a list of elements for which the condition is true

`reduce(function, list_of_inputs)`

Reduces the list to a result, given the function

Recursion with Higher Order Fun

```
def map(f, s):
    if
    else:
def square(x):
    return x**2

>>> map(square, [2,4,6])
[4, 16, 36]
```

Base Case

Recursive Case

- Divide and conquer

Using HOF to preserve interface

```
def sum_of_squares(n):
    def sum_upper(i, accum):
        if i > n:
            return accum
        else:
            return sum_upper(i+1, accum + i*i)

    return sum_upper(1,0)
```

- What are the globals and locals in a call to `sum_upper`?
  - Try python tutor
- Lexical (static) nesting of function def within def - vs
- Dynamic nesting of function call within call

Recap: Quicksort

- Break the problem into multiple smaller sub-problems, and Solve them recursively

```
def split(x, s):
    return [i for i in s if i <= x], [i for i in s if i > x]

def qsort(s):
    """Sort a sequence - split it by the first element,
    sort both parts and put them back together."""
    if not s:
        return []
    else:
        pivot = first(s)
        lessor, more = split(pivot, rest(s))
        return qsort(lessor) + [pivot] + qsort(more)

>>> qsort([3,3,1,4,5,4,3,2,1,17])
[1, 1, 2, 3, 3, 3, 4, 4, 5, 17]
```


Quicksort with HOF

```
def qsort(s):
    """Sort a sequence - split it by the first element,
    sort both parts and put them back together."""

    if not s:
        return []
    else:
        pivot = first(s)
        lessor, more = split_fun(leq_maker(pivot), rest(s))
        return qsort(lessor) + [pivot] + qsort(more)

>>> qsort([3,3,1,4,5,4,3,2,1,17])
[1, 1, 2, 3, 3, 3, 4, 4, 5, 17]
```


How much ???

- "Time" is required to compute `quicksort(s)`?
- "Space" is required?

Logarithmic to len(s)
c*log(len(s)) for some c

- Name of this recursion scheme?
  - Tree recursion

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

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

```
inc = lambda v : v + 1
```

```
def inc(v):
    return v + 1
```

Lambda Examples

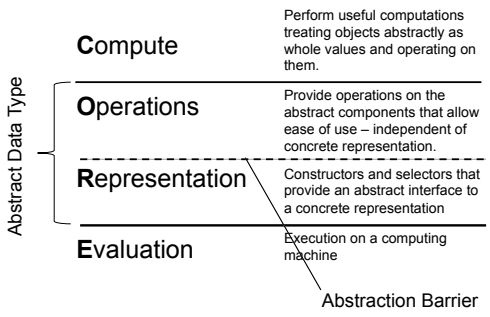
```
>>> msort([1,2,3,4,5], lambda x: x)
    [1, 2, 3, 4, 5]

>>> 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:x[1])
    [(7, 'I'), (5, 'goes'), (2, 'hi'), (1, 'how')]

>>> 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

Lambdas

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

>>> inc_maker(3)(4)
7
>>> 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]
>>>
```


C.O.R.E concepts



Creating an Abtract 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

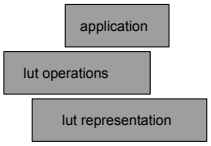
- **Lists**
  - **Constructors:**
    - **`list( … )`**
    - **`[ <exps>,… ]`**
    - **`[<exp> for <var> in <list> [ if <exp> ] ]`**
  - **Selectors: `<list> [ <index or slice> ]`**
  - **Operations: `in, not in, +, *, len, min, max`**
    - **Mutable ones too (but not yet)**
- **Tuples**
  - **Constructors:**
    - **`tuple( … )`**
    - **`( <exps>,… )`**
  - **Selectors: `<tuple> [ <index or slice> ]`**
  - **Operations: `in, not in, +, *, len, min, max`**

Examples You have seen

- **Lists**
- **Tuples**
- **Strings**
  - **Constructors:**
    - **`str( … )`**
    - **`"<chars>", '<chars>'`**
  - **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`**

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**

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

lut ADT

- **Constructors**
  - **`lut()`, `lut_add(lut, key, value)`, `lut_del(lut, key)`**
- **Selectors**
  - **`lut_get(lut, key)`, `lut_keys(lut)`, `lut_values(lut)`, `lut_items(lut)`**
- **Operations**
  - **`lut_with_bindings(bindings)` - Return a lut of bindings**
  - **`lut_len(lut)` - Return the number of bindings in lut.**
  - **`lut_print(lut)` - Print a representation of bindings in lut.**
  - **`lut_map_values(lut, fun)`**
  - **`lut_sorted(lut, fun)`**
  - **`lut_update(lut, key, value)`**
  - **`lut_fuzzy_get(lut, fuzz_key, dist_fun)`**
    - » **Return (key, value) for the key closest to fuzz_key under dist_fun.**

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**

Dictionaries

- **Lists, Tuples, Strings, Range**
- **Dictionaries**
  - **Constructors:**
    - » **`dict( <list of 2-tuples> )`**
    - » **`dict( <key>=<val>, ...) # like kwargs`**
    - » **`{ <key exp>:<val exp>, … }`**
    - » **`{ <key>:<val> for <iteration expression> }`**

```
>>> {x:y for x,y in zip(["a","b"],[1,2])}
{'a': 1, 'b': 2}
```

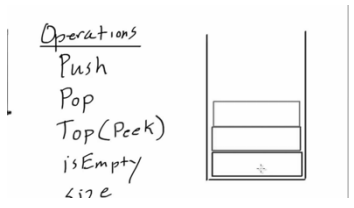
  - **Selectors: `<dict> [ <key> ]`**
    - » **`<dict>.keys(), .items(), .values()`**
    - » **`<dict>.get(key [, default] )`**
  - **Operations:**
    - » **`Key in, not in, len, min, max`**
    - » **`<dict>[ <key> ] = <val>`**

Dictionary Example

```
In [1]: text = "Once upon a time"
        d = {word : len(word) for word in text.split()}
        d
Out[1]: {'Once': 4, 'a': 1, 'time': 4, 'upon': 4}

In [2]: d['Once']
Out[2]: 4

In [3]: d.items()
Out[3]: [('a', 1), ('time', 4), ('upon', 4), ('Once', 4)]

In [4]: for (k,v) in d.items():
            print(k,"=>",v)

        ('a', '=>', 1)
        ('time', '=>', 4)
        ('upon', '=>', 4)
        ('Once', '=>', 4)

In [5]: d.keys()
Out[5]: ['a', 'time', 'upon', 'Once']

In [6]: d.values()
Out[6]: [1, 4, 4, 4]
```

In Lab

- **Dictionaries**
- **Lambdas**
- **Abstract Data Types**
- **Go build things…**



Questions?

A lut application (lut_app.py)

```
from lut import *

phone_book_data = [
    ("Christine Strauch", "510-842-9235"),
    ("Frances Catal Buloan", "932-567-3241"),
    ("Jack Chow", "617-547-0923"),
    ("Joy De Rosario", "310-912-6483"),
    ("Casey Casem", "415-432-9292"),
    ("Lydia Lu", "707-341-1254")]

phone_book = lut_with_bindings(phone_book_data)

lut_print(phone_book)

print("Jack Chows'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)

```
New_book = lut_update(phone_book, "Jack Chow", "805-962-0936")

lut_sorted(new_phone_book, lambda k,v:v)
```

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

Apps (cont)

```
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

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

More Friends

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


Above Abstraction Barrier – lut.py

```
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
```


Above Abstraction Barrier – lut.py

```
def lut_with_bindings(bindings):

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]))
```


Above Abstraction Barrier – lut.py

```
def lut_with_bindings(bindings):

def lut_sorted(lut, fun):

def lut_print(lut):
    """Print a representaion of bindings in lut."""
    for k,v in lut_sorted(lut, lambda k,v:k):
       print(k,"=>",v)
```


Above Abstraction Barrier – lut.py

```
def lut_with_bindings(bindings):

def lut_sorted(lut, fun):

def lut_print(lut):

def lut_map_values(lut_to_map, fun):
    """Return lut of bindings (k, fun(v))
       for k => v bindings in lut_to_map."""

    return lut_with_bindings([(k,fun(v)) for k,v in
                                    lut_items(lut_to_map)])
```


Above Abstraction Barrier – lut.py

```
def lut_with_bindings(bindings):

def lut_sorted(lut, fun):

def lut_print(lut):

def lut_map_values(lut_to_map, fun):

def lut_update(lut, key, value):
    """Return a new lut with new or updated
    key=>value binding."""

    if lut_get(lut, key) is None:
       return lut_add(lut, key, value)
    else:
        return lut_add(lut_del(lut, key), key, value)
```

Beneath the Abstraction Barrier

- **How to represent a lookup table?**

Representation: list of tuples

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

```
# Constructors

def lut():
    """Construct a lookup table."""
    return []

def lut_add(lut, key, value):
    """Return a new lut with (key,value) binding added."""
    assert key not in lut_keys(lut), "Duplicate key"

    return [(key, value)] + lut

def lut_del(lut, key):
    """Return a new lut with (key, *) binding removed."""
    assert key in lut_keys(lut), "Missing key"

    return [(k, v) for k,v in lut if k != key]
```


Repr: list of tuples (lut_tuples.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 lut:
       if k == key:
            return val
    return None

def lut_keys(lut):
    """Return a list of keys in lookup table lut."""
    return map(lambda x:x[0], lut)

def lut_values(lut):
def lut_items(lut):
```


Repr: tuple of lists – lut_lists.py

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

```
# Constructors

def lut():
    """Construct a lookup table."""
    return ([], [])

def lut_add(lut, key, value):
    """Return a new lut with (key,value) binding added."""
    assert key not in lut_keys(lut), "Duplicate key"
    return ([key] + lut_keys(lut), [value] + lut_values(lut))

def lut_del(lut, key):
    """Return a new lut with (key, *) binding removed."""
    assert key in lut_keys(lut), "Missing 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: list of tuples (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]
```


Repr: list of tuples (lut_lists.py)

```
# 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):
    """Return a list of values in lookup table lut."""
    return lut[1]

def lut_items(lut):
    """Return a list of (key,value) items in lut."""
    return list(zip(lut[0],lut[1]))
```