Computational Structures in Data Science

Lecture 7: Lambda & Abstract Data Types

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

March 4, 2019

Computational Concepts Toolbox

- Data type: values, literals, operations,
  - e.g., int, float, string
- Expressions, Call expression
- Variables
- Assignment Statement
- Sequences: tuple, list
  - indexing
- Data structures
- Tuple assignment
- Call Expressions
- Function Definition Statement
- Conditional Statement

- Iteration:
  - data-driven (list comprehension)
  - control-driven (for statement)
  - while statement
- Higher Order Functions
  - Functions as Values
  - Functions with functions as argument
  - Assignment of function values
- Recursion
- Lambda - function valued expressions

Universality

- Everything that can be computed, can be computed with what you know now.
- Poorly or Well

Evolution of Programming Languages

Today’s Lecture

- Lambda
- New Concept: Abstract Data Type
- Example Illustration: key-value store
  - Internal representation 1: list of pair
  - Internal representation 2: pair of lists (including zip intro)
- A simple application over the KV interface
- New language construct: dict
- Key-Value store 3: dict
- Optional Exercises

Lambda

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

\[
\text{lambda } \langle \text{arg or arg_tuple} \rangle : \langle \text{expression using args} \rangle
\]

\[
\text{inc} = \text{lambda } v : v + 1 \quad \text{def inc(v): return } v + 1
\]

http://inst.eecs.berkeley.edu/~cs88

lambda

http://datahub.berkeley.edu/user-redirect/interact?account=data-8&repo=cs-connector&branch=gh-pages&path=ADT

Lambda Examples

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

Lambdas

```python
>>> 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]
```

Abstract Data Type

- **Operations**
- **Object**
  - **Constructors**
  - **Selectors**
  - **Operations**

Examples Data Types You have seen

- **Lists**
  - Constructors:
    - `list(...)`
    - `[<expr>,..]`
  - Operations: `in, not in, +, *, len, min, max`
- **Tuples**
  - Constructors:
    - `tuple(...)`
  - Operations: `in, not in, +, *, len, min, max`
- **Strings**
  - Constructors:
    - `str(...)`
    - `<char>, <chars>`
  - Operations: `in, not in, +, *, len, min, max`
- **Range**
  - Constructors:
    - `range(<end>), range(<start>,<end>), range(<start>,<end>,<step>)`
  - Operations: `in, not in, +, *, len, min, max`

More “Built-in” Examples

- **Lists**
- **Tuples**
- **Strings**
- **Range**

A New Abstract Data Type: Key-Value

- **Collection of key-Value bindings**
  - Key: Value
- **Many real-world examples**
  - Dictionary, Directory, Phone book, Course Schedule, Facebook Friends, Movie listings, ...

Given some Key, What is the value associated with it?
Key-Value ADT

- Constructors
  - kv_empty: create an empty KV
  - kv_add: add a key:value binding to a KV
  - kv_create: create a KV from a list of key,value tuples

- Selectors
  - kv_items: list of (key,value) tuple in KV
  - kv_keys: list of keys in KV
  - kv_values: list of values in KV

- Operations
  - kv_len: number of bindings
  - kv_in: presence of a binding with a key
  - kv_display: external representation of KV

A little application

```python
from kv_pairs import *

phone_book_data = [(
    "Christine Strauch", "510-842-9235"),
    ("Francois Curnel", "932-567-3601"),
    ("Jack Chow", "415-947-0927"),
    ("Joy De Rosario", "330-812-6883"),
    ("Casey Casem", "415-832-9203"),
    ("Lydia Lu", "707-341-1250")
]

phone_book = kv_create(phone_book_data)

print("Jack Chows's Number: ", kv_get(phone_book, "Jack Chow"))

print("Area codes")

area_codes = list(map(lambda x:x[0:3], kv_values(phone_book)))

print(area_codes)
```

A Layered Design Process

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

Example 1

- KV represented as list of (key, value) pairs

Example 2

- KV represented as pair of lists – (keys, values)

zip

- Zip (like a zipper) together k lists to form a list of k-tuples

```python
In [19]: # introduction to zip
    ...: list(zip('a', 'b', 'c', [1, 2, 3]))
Out[19]: [('a', 'b', 'c', 1), ('a', 'b', 'c', 2), ('a', 'b', 'c', 3)]
In [20]: # zip for (a,b) in zip(range(10), range(10))
Out[20]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
In [21]: def zip2(a, b):
    ...:     return [(a[i], b[i]) for i in range(min(len(a), len(b)))]
In [22]: zip2('a', 'b', 'c', [1, 2, 3])
Out[22]: [('a', 1), ('b', 2), ('c', 3)]
```
**Dictionaries**

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

**Dictionary Example**

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

**Beware**

- Built-in data type `dict` relies on mutation
  - Clobbers the object, rather than "functional" – creating a new one
- Throws an error if key is not present
- We will learn about mutation shortly

**Example 3**

- KV represented as dict

```
friend_data = [
    ("Christine Strauch", "Jack Chow"),
    ("Christine Strauch", "Lydia Lu"),
    ("Jack Chow", "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")
]

```

**Building Apps over KV ADT**

```
def make_friends(friendships):
    friends = kv_empty()
    for (der, dee) in friendships:
        if not kv_in(friends, der):
            friends = kv_add(friends, der, 
            else:
                der_friends = kv_get(friends, der)
                friends = kv_add(kv_delete(friends, der),
                return friends
```

```
friend_data = {
    ("Christine Strauch", "Jack Chow"),
    ("Christine Strauch", "Lydia Lu"),
    ("Jack Chow", "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")
}

```

- Construct a table of the friend list for each person
C.O.R.E concepts

### Compute
Perform useful computations treating objects abstractly as whole values and operating on them.

### 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

<table>
<thead>
<tr>
<th>Abstract Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Constructors &amp; Selectors</strong></td>
</tr>
<tr>
<td><strong>Operations</strong></td>
</tr>
<tr>
<td>Express the behavior of objects, invariants, etc</td>
</tr>
<tr>
<td>Implemented (abstractly) in terms of Constructors and Selectors for the object</td>
</tr>
<tr>
<td><strong>Representation</strong></td>
</tr>
<tr>
<td>Implement the structure of the object</td>
</tr>
<tr>
<td>An abstraction barrier violation occurs when a part of the program that can use the higher level functions uses lower level ones instead</td>
</tr>
<tr>
<td>At either layer of abstraction</td>
</tr>
<tr>
<td>Abstraction barriers make programs easier to get right, maintain, and modify</td>
</tr>
<tr>
<td>Few changes when representation changes</td>
</tr>
</tbody>
</table>

### Creating an Abstract Data Type

- Constructors & Selectors
- Operations
  - Express the behavior of objects, invariants, etc
  - Implemented (abstractly) in terms of Constructors and Selectors for the object
- Representation
  - 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

### Exercises
- Read 2.2, reread 2.3, esp 2.3.6
- Modify all three KV ADTs to avoid ever adding duplicate keys
- Create and ADT for a shopping cart containing a collection of products and their order count
  - \( \text{cart}() \) – creates an empty cart
  - \( \text{cart\_add}(\text{ct}, \text{product}) \) – returns a new cart that includes an additional order of product, or the first one
  - \( \text{cart\_print}(\text{ct}) \) – prints the contents of the cart
  - \( \text{cart\_products}(\text{ct}) \) – returns the list of products ordered
  - \( \text{cart\_items}(\text{ct}) \) – returns list of (product, count)
  - \( \text{cart\_remove}(\text{ct}, \text{product}) \) - returns a new cart with product removed
- Create an 1D array abstraction (like np.array) using lists as representation

### Thoughts for the Wandering Mind

Consider the following simple Python code:

```python
x = input('Enter a number between 0 and 1: ')
for i in range(10):
    x = -x**2 + 4*x
print x
```

- Plot the function implemented by the code.
  - Could you predict using sampling (e.g., interpolate from the results of inputs 0, 0.25, 0.5, 0.75, 1)?
  - Could you predict using calculus (e.g., using the derivative of \( f(x) = -x^2 + 4x \))?
  - Could a neural network learn the function, given enough (input, output) tuples as training data?