



# Computational Structures in Data Science

---



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

## Lecture 7: Lambda & Abstract Data Types



# 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

## Mother Tongues

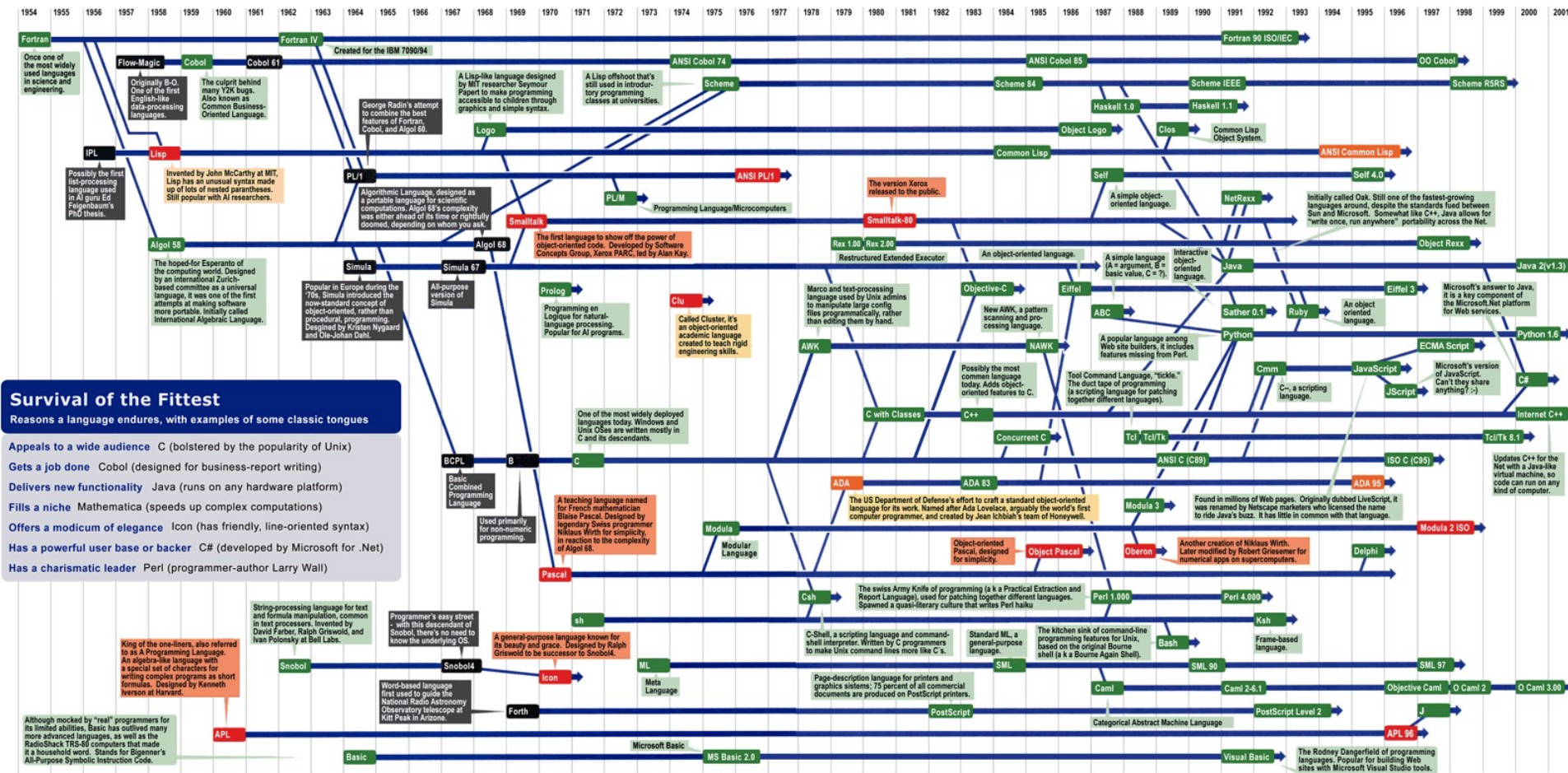
Tracing the roots of computer languages through the ages

Just like half of the world's spoken tongues, most of the 2,300-plus computer programming languages are either endangered or extinct. As powerhouses C/C++, Visual Basic, Cobol, Java and other modern source codes dominate our systems, hundreds of older languages are running out of life.

An ad hoc collection of engineers-electronic lexicographers, if you will-aim to save, or at least document the lingo of classic software. They're combing the globe's 9 million developers in search of coders still fluent in these nearly forgotten lingua frangas. Among the most endangered are Ada, APL, B (the predecessor of C), Lsp, Oberon, Smalltalk, and Simula.

Code-raker Grady Booch, Rational Software's chief scientist, is working with the Computer History Museum in Silicon Valley to record and, in some cases, maintain languages by writing new compilers so our ever-changing hardware can grok the code. Why bother? "They tell us about the state of software practice, the minds of their inventors, and the technical, social, and economic forces that shaped history at the time," Booch explains. "They'll provide the raw material for software archaeologists, historians, and developers to learn what worked, what was brilliant, and what was an utter failure." Here's a peek at the strongest branches of programming's family tree. For a nearly exhaustive rundown, check out the Language List at [HTTP://www.informatik.uni-freiburg.de/Java/misc/lang\\_list.html](http://www.informatik.uni-freiburg.de/Java/misc/lang_list.html). - Michael Mendeno

Key	
1954	Year Introduced
Green	Active: thousands of users
Orange	Protected: taught at universities; compilers available
Yellow	Endangered: usage dropping off
Red	Extinct: no known active users or up-to-date compilers
Blue arrow	Lineage continues



### Survival of the Fittest

Reasons a language endures, with examples of some classic tongues

- Appeals to a wide audience C (bolstered by the popularity of Unix)
- Gets a job done Cobol (designed for business-report writing)
- Delivers new functionality Java (runs on any hardware platform)
- Fills a niche Mathematica (speeds up complex computations)
- Offers a modicum of elegance Icon (has friendly, line-oriented syntax)
- Has a powerful user base or backer C# (developed by Microsoft for .Net)
- Has a charismatic leader Perl (programmer-author Larry Wall)

Sources: Paul Boutin; Brent Hallgren, associate director of computer science at IBM Research; The Retrocomputing Museum; Todd Proebsting, senior researcher at Microsoft; Gio Wiederhold, computer scientist, Stanford University



# Today's Lecture

---

<http://bit.ly/cs88-fa18-L07>

- **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**

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



# 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

---

```
>>> 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[0])  
[(1, 'how'), (2, 'hi'), (5, 'goes'), (7, 'I')]
```

```
>>> 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')]
```

<http://cs88-website.github.io/assets/slides/ad/mersort.py>



# Lambdas

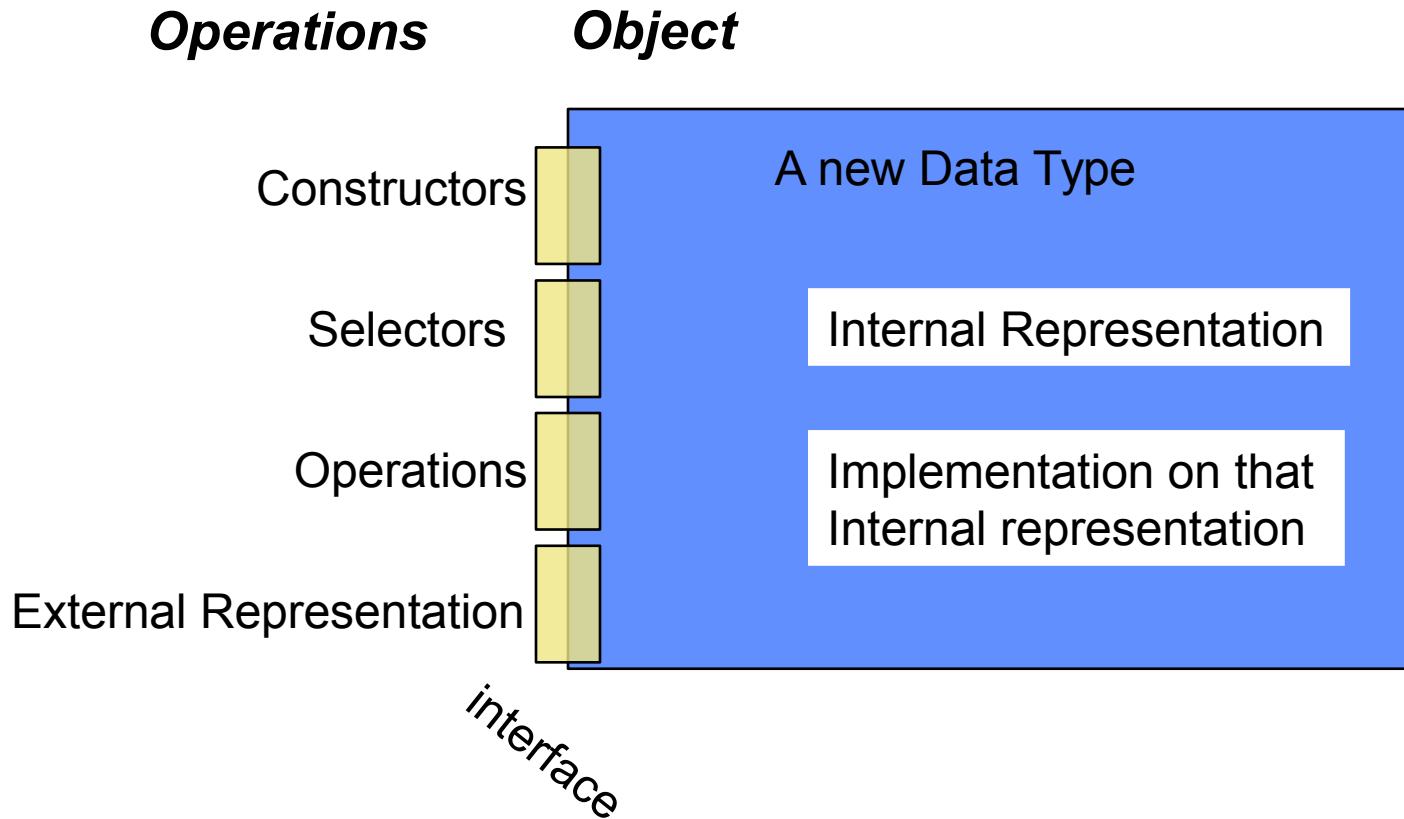
---

```
>>> 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]  
>>>
```





# Abstract Data Type





# Examples Data Types 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`



# More “Built-in” Examples

---

- **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`



# 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

---

```
from kv_pairs import *

phone_book_data = [
    ("Christine Strauch", "510-842-9235"),
    ("Frances Catal Bulloan", "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 = 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

```
In [19]: # introduction to zip
list(zip(['a', 'b', 'c'], [1, 2, 3]))
```

```
Out[19]: [('a', 1), ('b', 2), ('c', 3)]
```

```
In [20]: [a+b for (a,b) in zip([1, 2, 3], [10, 20, 30])]
```

```
Out[20]: [11, 22, 33]
```

```
In [16]: def zip2(a, b):
          return [(a[i], b[i]) for i in range(min(len(a), len(b)))]
```

```
In [18]: zip2(['a', 'b', 'c'], [1, 2, 3])
```

```
Out[18]: [('a', 1), ('b', 2), ('c', 3)]
```



# 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]
```

# Beware

---



shutterstock.com - 762803443

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



## Example 3

---

- **KV represented as dict**





# Building Apps over KV ADT

---

```
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 Bulloan"),  
    ("Casey Casem", "Joy De Rosario"),  
    ("Casey Casem", "Casey Casem"),  
    ("Frances Catal Bulloan", "Jack Chow"),  
    ("Jack Chow", "Frances Catal Bulloan"),  
    ("Joy De Rosario", "Lydia Lu"),  
    ("Joy De Lydia", "Jack Chow")  
]
```

- **Construct a table of the friend list for each person**



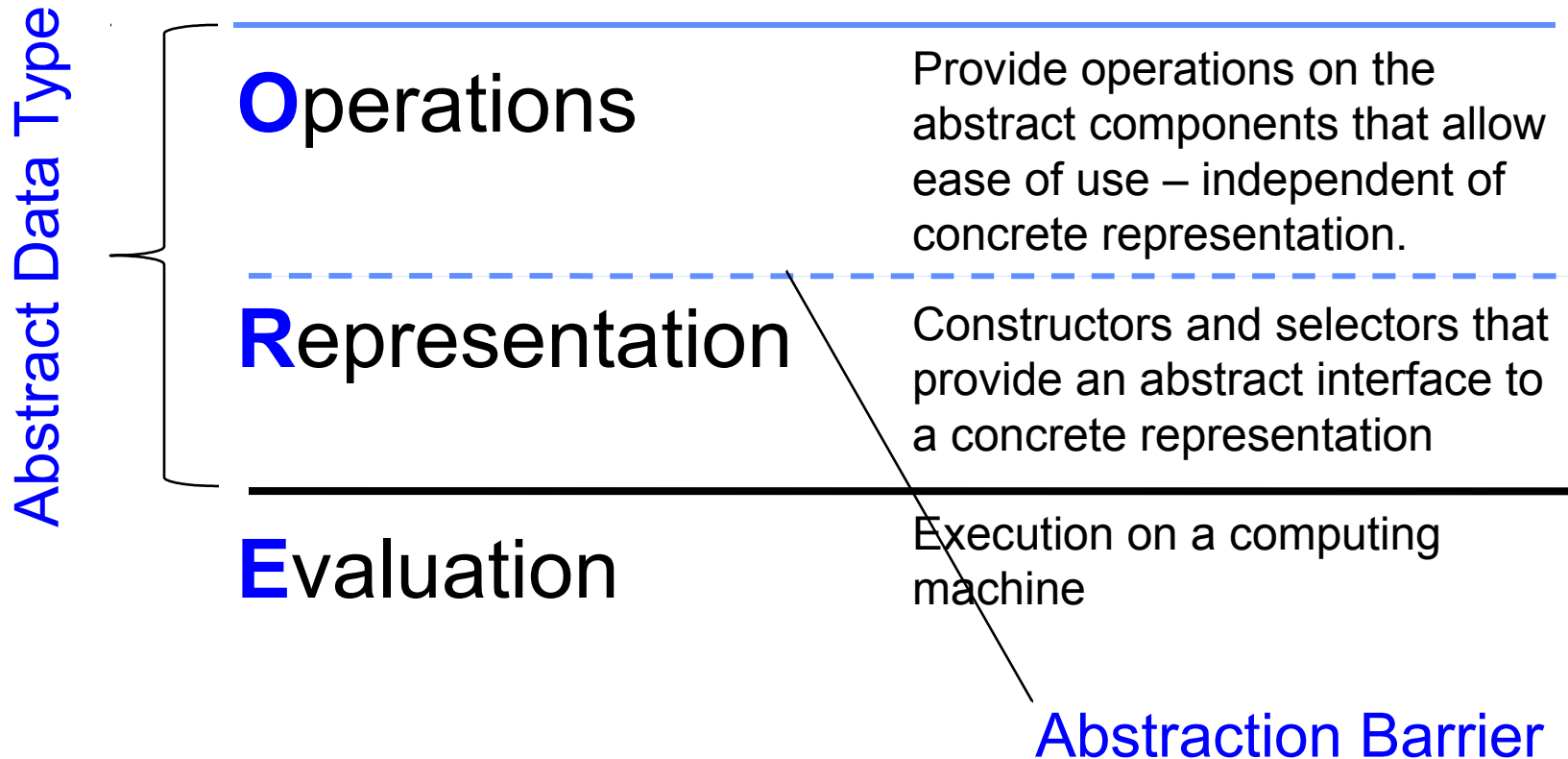
# Example: make\_friends

---

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



# C.O.R.E concepts





# 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 an ADT for a shopping cart containing a collection of products and their order count
  - `cart()` – creates an empty cart
  - `cart_add(ct, product)` – returns a new cart that includes an additional order of product, or the first one
  - `cart_print(ct)` – prints the contents of the cart
  - `cart_products(ct)` – returns the list of products ordered
  - `cart_items(ct)` – returns list of (product, count)
  - `cart_remove(ct, 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:**

```
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?