

 Computational Structures in Data Science

Lecture 6: Recursion (cont)

October 14, 2019 <http://cs88.org>

 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

 10/14/19 UCB CS88 Fa19 L4 1

 Today's Lecture

- Recursion
 - More practice
 - Some tips & tricks
- Abstract Data Types
 - More use of functions!
 - Value in documentation and clarity
 - Not on next Monday's midterm

10/14/19 UCB CS88 Fa19 L4 3

 Announcements

- MIDTERM MOVED!
- Monday 10/21 7-9pm
- No Assignments Due this week
- Saturday October 19th 5 - 8:30 pm

4

 Recall: Iteration

```
def sum_of_squares(n):
    accum = 0
    for i in range(1, n+1):
        accum = accum + i*i
    return accum
```

1. Initialize the "base" case of no iterations
2. Starting value
3. Ending value
4. New loop variable value

 Recursion Key concepts – by example

```
def sum_of_squares(n):
    if n < 1:
        return 0
    else:
        return sum_of_squares(n-1) + n**2
```

1. Test for simple "base" case
2. Solution in simple "base" case
3. Assume recursive solution to simpler problem
4. Transform soln of simpler problem into full soln

Oct 7, 2019 UCB CS88 Fall 2019 L5 6

Recursion

- Base Case:**
 - What stops the recursion?
- Recursive Case:**
 - Divide
 - Invoke
 - Combine2

```
def sum_of_squares(n):
    if n < 1:
        return 0
    else:
        return sum_of_squares(n-1) + n**2
```

Why does it work

```
sum_of_squares(3)
# sum_of_squares(3) => sum_of_squares(2) + 3**2
#                         => sum_of_squares(1) + 2**2 + 3**2
#                         => sum_of_squares(0) + 1**2 + 2**2 + 3**2
#                         => 0 + 1**2 + 2**2 + 3**2 = 14
```

How does it work?

- Each recursive call gets its own local variables
 - Just like any other function call
- Computes its result (possibly using additional calls)
 - Just like any other function call
- Returns its result and returns control to its caller
 - Just like any other function call
- The function that is called happens to be itself
 - Called on a simpler problem
 - Eventually bottoms out on the simple base case
- Reason about correctness “by induction”
 - Solve a base case
 - Assuming a solution to a smaller problem, extend it

Questions

- In what order do we sum the squares ?
- How does this compare to iterative approach ?

```
def sum_of_squares(n):
    accm = 0
    for i in range(1,n+1):
        accm = accm + i**2
    return accm
```

```
def sum_of_squares(n):
    if n < 1:
        return 0
    else:
        return sum_of_squares(n-1) + n**2
```

```
def sum_of_squares(n):
    if n < 1:
        return 0
    else:
        return n**2 + sum_of_squares(n-1)
```

Local variables

```
def sum_of_squares(n):
    n_squared = n**2
    if n < 1:
        return 0
    else:
        return n_squared + sum_of_squares(n-1)
```

- Each call has its own “frame” of local variables
- What about globals?

<https://goo.gl/CiFaUJ>

Oct 7, 2019 UCB CS68 Fall 2019 L5 11

Fibonacci Sequence

$$\begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ F_n &= F_{n-1} + F_{n-2} \end{aligned}$$

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89,

Go Bears!

13

How many calls?

- How many calls of fib() does executing fib(4) make?

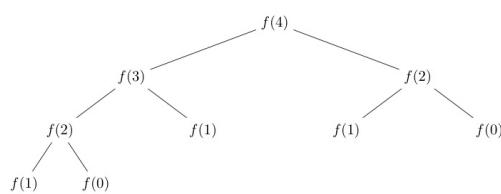
E.g. Calling fib(1) makes 1 call.

- A) 4
- B) 5
- C) 8
- D) 9
- E) 16

14

Answer 9

- Fib(4) → Fib(3), Fib(2)
- Fib(3) → Fib(2), Fib(1)
- Fib(2) → Fib(1), Fib(0)



15

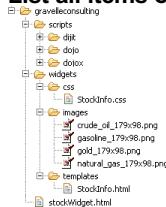
Trust ...

- The recursive “leap of faith” works as long as we hit the base case eventually

What happens if we don't?

Recursion (unwanted)

17

Example I**List all items on your hard disk**

18

Extra: List Files in Python

```
def listfiles(directory):
    content = [os.path.join(directory, x) for x in os.listdir(directory)]
    dirs = sorted([x for x in content if os.path.isdir(x)])
    files = sorted([x for x in content if os.path.isfile(x)])

    for d in dirs:
        print(d)
        listfiles(d)

    for f in files:
        print(f)
```

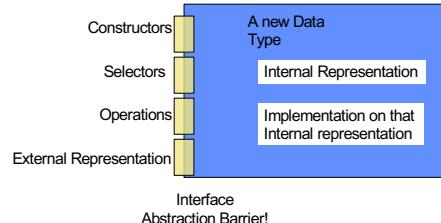
Iterative version about twice as much code and much harder to think about.

UCB CS88 Fall 2019 L5

19

Abstract Data Type

Operations Object



10/14/19

UCB CS88 Fa19 L4

20

Why ADTs?

- **"Self-Documenting"**
 - `contact_name(contact)`
 - » Vs `contact[0]`
 - "0" may seem clear now, but what about in a week? 3 months?
- **Change your implementation**
 - Maybe today it's just a Python List
 - Tomorrow: It could be a file on your computer; a database in web

21

Examples Data Types You have seen

- **Lists**
 - Constructors:
 - » `list(...)`
 - » `[<expr>, ...]`
 - » `[<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(...)`
 - » `(<expr>, ...)`
 - Selectors: `<tuple> [<index or slice>]`
 - Operations: `in`, `not in`, `+`, `*`, `len`, `min`, `max`

10/14/19

UCB CS88 Fa19 L4

21

An Abstract Data Type: Key-Value Pair

- **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?

10/14/19

UCB CS88 Fa19 L4

23

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

10/14/19

UCB CS88 Fa19 L4

24

A little application

```

phone_book_data = [
    ("Christine Strauch", "510-842-9235"),
    ("Frances Catal Bulaon", "932-567-3241"),
    ("Jack Chow", "617-547-0923"),
    ("Joy De Rosario", "310-912-9483"),
    ("Casey Casem", "415-432-9292"),
    ("Lydia Lu", "707-341-1254")
]

phone_book = pb_create(phone_book_data)

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

print("Area codes")
area_codes = list(map(lambda x:x[0:3], pb_numbers(phone_book)))
print(area_codes)

```

10/14/19 UCB CS88 Fa19 25

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

10/14/19 UCB CS88 Fa19 L4 26

Example 1

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

10/14/19 UCB CS88 Fa19 4 27

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



10/14/19 UCB CS88 Fa19 L4 28

Dictionary Example

```

In [1]: text = "Once upon a time"
d = {word : len(word) for word in text.split()}
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 (x,v) in d.items():
    print(x,">",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]

```

10/14/19 29

Beware

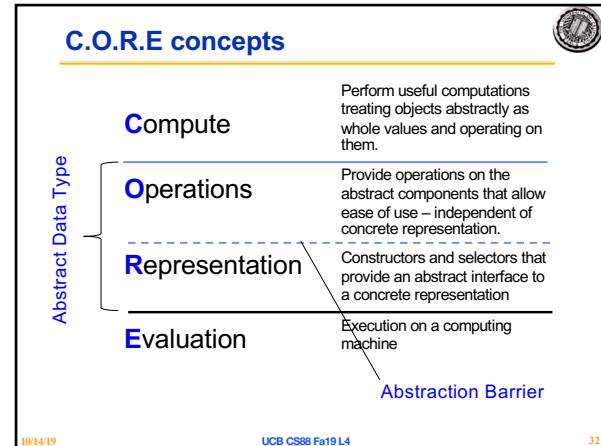
- Built-in data type dict relies on mutation
 - Clobbers the object, rather than “functional” – creating a new one
- Throws an errors of key is not present
- We will learn about mutation shortly

10/14/19 UCB CS88 Fa19 L4 30

Example 3

- KV represented as dict

10/14/19 UCB CS88 Fa19 L4 31



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

10/14/19 UCB CS88 Fa19 L4 33

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 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

10/14/19 UCB CS88 Fa19 L4 34

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
  
```

10/14/19 UCB CS88 Fa19 L4 35