CS61A Lecture 9
Immutable Data Structures

Jom Magrotker
UC Berkeley EECS
July 2, 2012

TODAY

• Review: Tuples.
• Review: Data abstraction.
• New sequences and data structures:
  Ranges, Pairs, Immutable recursive lists.

SEQUENCES

A sequence is an ordered collection of data values.

There are many kinds of sequences, and all share certain properties.

Length: A sequence has a finite length.
Element selection: A sequence has an element for any non-negative integer less than its length.

REVIEW: TUPLES

A tuple is a built-in type that represents a sequence.

>>> triplet = (1, 2, 3)
>>> len(triplet)
3
>>> triplet[0]
1
>>> from operator import itemgetter
>>> itemgetter(triplet, 0)
1

REVIEW: TUPLES

A tuple is an example of a data structure.

A data structure is a type of data that exists primarily to hold other pieces of data in a specific way.
**Review: Working with Tuples**

Write the higher order function `map`, which takes a function `fn` and a tuple of values `vals`, and returns a tuple of results of applying `fn` to each value in `vals`.

```python
>>> map(square, (1, 2, 3, 4, 5))
(1, 4, 9, 16, 25)
```

```python
>>> map(lambda x: x+1, (1, 2, 3, 4, 5))
(2, 3, 4, 5, 6)
```

**Review: Working with Tuples**

Write the higher order function `filter`, which takes a predicate function `pred` and a tuple of values `vals`, and returns a tuple of values that satisfy the predicate.

```python
>>> filter(lambda x: x%2==0, (1, 2, 3, 4, 5))
(2, 4)
```

```python
>>> filter(isprime, (2, 3, 4, 5, 6))
(2, 3, 5)
```

**Ranges**

A `range` is another built-in type that represents a sequence. It represents a range of integers.

```python
>>> range(0, 10)
range(0, 10)
```

```python
>>> range(0, 10)
range(0, 10)
```

```python
>>> range(0, 10)
range(0, 10)
```

```python
>>> range(0, 10)
range(0, 10)
```

```python
>>> range(0, 10)[3]
4
```

**Announcements**

- Homework 4 is due July 3.
- Homework 5 is released, due July 6.
- Project 2 is released, due July 13.
- No class on Wednesday, July 4.
- Project 1 contest is on!
  - How to submit: Submit a file with your `final_strategy` to `proj1-contest`.
  - Deadline: Friday, July 6 at 11:59pm.
  - Prize: One of 3 copies of Feynman and 1 extra credit point.
  - Metric: We will simulate your strategy against everyone else's, and tally your win rate. Draws count as losses.

**Announcements: Midterm 1**

- Midterm 1 is on July 9.
  - Where? 2050 VLSB.
  - When? 7PM to 9PM.
- Closed book and closed electronic devices.
- One 8.5" x 11" 'cheat sheet' allowed.
- Group portion is 15 minutes long.
- Post-midterm potluck on Wednesday, July 11.
**REVIEW: DATA ABSTRACTION**

We want to think about data in terms of its **meaning**, not its **representation**. Programs should operate on **abstract data**.

We use functions to create a **division** between manipulation and representation. Functions can be **constructors** or **selectors**.

---

**EXAMPLE: STUDENT RECORDS**

We would like to work with **student records**.

- `make_student(name, id, grades)` creates a new record.
- `name(student)` returns the name of student.
- `calid(student)` returns the ID of student.
- `grades(student)` returns a tuple of grades of student.

---

**EXAMPLE: STUDENT RECORDS**

Write a function `names_start_with` that takes in a tuple of student records, `records`, and a letter, and returns a tuple of the IDs of the students whose name starts with `letter`.

```python
def names_start_with(records, letter):
    results = ()
    for record in records:
        if name(record).startswith(letter):
            results = results + (calid(record),)
    return results
```

---

**EXAMPLE: STUDENT RECORDS**

Can use anything to construct the student record, as long as the selectors are consistent.

```python
def make_student(name, id, grades):
    return (name, id, grades)

def name(student):
    return student[0]

def calid(student):
    return student[1]

def grades(student):
    return student[2]
```

---

**RESPECT THE DATA ABSTRACTION!**

Louis Reasoner wrote the following code to count the number of As for a given student. However, he has a data abstraction violation. Correct his code so that it respects the data abstraction.

```python
def count_as(student):
    number_of_as = 0
    for grade in student[2]:
        if grade == "A":
            number_of_as = number_of_as + 1
    return number_of_as
```
RESPECT THE DATA ABSTRACTION!

Louis Reasoner wrote the following code to count the number of As for a given student. However, he has a data abstraction violation. Correct his code so that it respects the data abstraction.

```python
def count_as(student):
    number_of_as = 0
    for grade in grades(student):
        if grade == "A":
            number_of_as = number_of_as + 1
    return number_of_as
```

IMMUTABILITY

Numbers, Booleans, strings, tuples, and ranges are examples of **immutable** data structures.

```python
def map(fn, vals):
    results = ()
    for val in vals:
        results = results + (fn(val),)
    return results
```

DATA STRUCTURE: PAIRS

A **pair** is an ADT that can hold two elements. It can be implemented using tuples.

(But it can be implemented in other ways, including using functions.)

```python
make_pair(x, y) creates a new pair.
first(x) returns the first element of the pair.
second(x) returns the second element of the pair.
```

NESTED PAIRS

For simplicity, we will represent pairs as two-element tuples. Pairs can contain other pairs as elements.

```
(1, 2)
((1, 2), 3)
((1, 2), (3, 4))
((1, (2, 3)), 4)
```
NESTED PAIRS: BOX-AND-POINTER DIAGRAM

(1, 2)

NESTED PAIRS: BOX-AND-POINTER DIAGRAM

((1, 2), 3)

NESTED PAIRS: BOX-AND-POINTER DIAGRAM

((1, 2), (3, 4))

NESTED PAIRS: BOX-AND-POINTER DIAGRAM

(1, (2, 3))
((1, (2, 3)), 4)
(1, (2, (3, 4)))

NESTED PAIRS: BOX-AND-POINTER DIAGRAM

Nested pair (1, (2, (3, ()))))

NEW DATA STRUCTURE

Immutable Recursive List <1, 2, 3>
One possible (and useful) representation of a list with elements 1, 2 and 3
An immutable recursive list (or an IRList) is a pair such that:
- The first element of the pair is the first element of the list.
- The second element of the pair is the rest of the list – another immutable recursive list. The rest of the list could be empty.

```
empty_irlist = ()
def make_irlist(first, rest=empty_irlist):
    return (first, rest)
def irlist_first(irlist):
    return irlist[0]
def irlist_rest(irlist):
    return irlist[1]
```

Why are they useful?
- They are defined recursively. Functions that operate on IRLists are usually best and easily defined recursively.
- They are the basis for linked lists, a versatile data structure in computer science.

```
def irlist_len(irlist):
    if irlist == empty_irlist:
        return 0
    return 1 + irlist_len(irlist_rest(irlist))
```
IMMUTABLE RECURSIVE LISTS

Write the function `irlist_select` that returns the element at position `index` of the `irlist`. (Assume the inputs are valid.)

```python
def irlist_select(irlist, index):
    if index == 0:
        return ______________
    return irlist_select(irlist_rest(irlist), index - 1)
```

```
def irlist_select(irlist, index):
    if index == 0:
        return irlist_first(irlist)
    return irlist_select(irlist_rest(irlist), index - 1)
```

IMMUTABLE RECURSIVE LISTS

Write the function `irlist_map` that takes a function `fn` and an `irlist`, and returns an IRList of the results of applying `fn` to the elements of `irlist`.

```python
def irlist_map(fn, irlist):
    if irlist == empty_irlist:
        return make_irlist(____________)
    return make_irlist(fn(irlist_first(irlist)), irlist_map(fn, irlist_rest(irlist)))
```

```
def irlist_map(fn, irlist):
    if irlist == empty_irlist:
        return empty_irlist
    return make_irlist(fn(irlist_first(irlist)), irlist_map(fn, irlist_rest(irlist)))
```
CONCLUSION

• Data abstraction allows us to separate the meaning of abstract data from its implementation.
• A sequence is an ordered collection of data with certain properties.
• There are many useful ADTs in computer science, some of which are immutable.
• One example of a useful ADT is the immutable recursive list, built from pairs.
• Preview: Immutable dictionaries.