This Course was About...

- Fundamentals of programming
  - From specification to implementation
  - Software engineering principles
This Course was...

Not about...

- A particular programming language (e.g., Java, C++, Scheme)
- A particular programming tool (e.g., gcc, DrScheme)
- Specific libraries or protocols (e.g., Gtk, XML, HTTP)
- How programs get translated into electronic signals
Theme 1: Data Structures

Atomic data

num
1

string
"apple"
Theme 1: Data Structures

Compound data

; A posn is
; (make-posn num num)

(make-posn 1 2)

class Snake {
    String name;
    double weight;
    String food;
    ...
}

new Snake("slinky", 10, "rats")
Theme 1: Data Structures

Inductively defined data

• Lists

; A list-of-num is either
; - empty
; - (cons num list-of-num)

(cons 1 (cons 2 empty))
Theme 1: Data Structures

Inductively defined data

• Lists

```java
abstract class Pizza { ... }
class Crust extends Pizza {
    boolean wheat;
    ... 
}
class topping extends Pizza {
    String top;
    Pizza bottom;
    ... 
}

new Topping("tomato", 2, new Crust(false))
```
A rumor-mill is either
- empty
- (make-gossip string rumor-mill rumor-mill)

(make-gossip "Amir"
  (make-gossip "Joe"
    empty
    empty)
  (make-gossip "Linsey"
    empty
    empty))
Theme 1: Data Structures

• And more:

  ; A dir is
  ;   (make-dir sym lofd)

  ; A file is
  ;   (make-file sym num)

  ; A lofd is either
  ;   - empty
  ;   - (cons file lofd)
  ;   - (cons dir lofd)

  (make-dir 'tmp
       (list (make-file 'preview.ps 10)
            (make-dir 'build
                 (list
                    (make-file 'x.c 30)
                    (make-file 'a.out 10))))))
Theme 1: Data Structures

• And more:

```java
class Room {
    Door left;
    Door right;
    ... }
abstract class Door {
    ... }
class Escape extends Door {
    ... }
class Into extends Door {
    Room next;
    ... }
... 

new Into(new Room(new Escape("mars"),
    new Escape("venus")))
```
Theme 2: Data Drives Design

Data

- Understand the input data

Contract, Purpose, and Header

- Describe (but don't write) the function

Examples

- Show what will happen when the function is done

Template

- Set up the body based on the input data (and only the input)

Body

- The most creative step: implement the function body

Test

- Run the examples
Theme 2: Data Drives Design

The template is a pivotal implementation step:

- Programs that match the shape of the data tend to work, and they can be understood by others.
- Programs that do not match the shape of the data tend to fail in incomprehensible ways.

```scheme
; A list-of-num is either
;   - empty
;   - (cons num list-of-num)

; func : list-of-num -> ...
(define (func l)
  (cond
   [(empty? l) ...]
   [else (first l) ... (func (rest l)) ...]])
```
Theme 2: Data Drives Design

; A dir is
;  (make-dir sym lofd)

; A file is
;  (make-file sym num)

; A lofd is either
;  - empty
;  - (cons file lofd)
;  - (cons dir lofd)

; dir-func : dir -> ...
(define (dir-func d)
  ...
  (dir-name d)
  ...
  (lofd-func (dir-content d)) ...)

; file-func : file -> ...
(define (file-func f)
  ...
  (file-name f) ...
  (file-size f))

; lofd-func : lofd -> ...
(define (lofd-func l)
  (cond
    [(empty? l) ...]
    [(file? (first l))
      ...
      (file-func (first l))
      ...
      (lofd-func (rest l))]
    [(dir? (first l))
      ...
      (dir-func (first l))
      ...
      (lofd-func (rest l))]]))
Theme 2: Data Drives Design

class Room {
    Door left;
    Door right; ...
    Path escapePath(Person p) {
        ... left.escapePath(p)
        ... right.escapePath(p) ...
    }
}

abstract class Door {
    abstract Path escapePath(Person p);
}
class Escape extends Door {
    Path escapePath(Person p) {
        ... }
}
class Into extends Door {
    Room next; ...
    Path escapePath(Person p) {
        ... next.escapePath(p) ...
    }
}
Theme 2: Data Drives Design

Good Java style essentially forces you to follow the template
Following the template essentially forces good Java style
Theme 3: Contracts

A contract specifies, *in advance*

- Obligations of a producer
- Restrictions for a consumer

```scheme
; disk-usage : dir -> num

(define (disk-usage d)
  (foldr (lambda (f n)
           (+ n (file-size f)))
          0
          (dir-content d)))
```

Producer error: **disk-usage** should work on any **dir**
Theme 3: Contracts

A contract specifies, *in advance*

- Obligations of a producer
- Restrictions for a consumer

```
; disk-usage : dir -> num
```

```
... (disk-usage (make-snake 'Slinky 10 'rats))
```

Consumer error: `disk-usage` accepts only dirs
Theme 3: Contracts

A contract identifies the relevant data definition

- for examples
- for the implementation (template)
- for testing — helps ensure coverage

```scheme
; disk-usage : dir -> num
(define (disk-usage d)
  ... (dir-name d)
  ... (lofd-usage (dir-content d)) ...)
...
(disk-usage (make-dir 'home empty))
"should be" 0
```
Theme 3: Contracts

A contract identifies the relevant data definition

- for examples
- for the implementation (template)
- for testing – helps ensure coverage

Incorrect and abused contracts were the primary source of homework difficulties
Armed with data definitions and templates, you can write most things from scratch...

...but you shouldn't

If nothing else, cut and paste (or deja vu) should trigger reuse
Theme 4: Reuse

Data Representation and Contract

Examples

Maybe Abstract
  Use Existing

Template
  Body

Trivial Cases
  Recur on Smaller

Test
Reuse from abstraction:

; sum : list-of-num -> num
(define (sum l)
    (cond
        [(empty? l) 0]
        [(cons? l)
            (+ (first l)
                (sum (rest l)))]))

; product : list-of-num -> num
(define (product l)
    (cond
        [(empty? l) 1]
        [(cons? l)
            (* (first l)
                (product (rest l)))]))

; combine-nums : list-of-num
; (num num -> num) -> num
(define (combine-nums l base-n COMB)
    (cond
        [(empty? l) base-n]
        [(cons? l)
            (COMB (first l)
                (combine-nums (rest l)
                    base-n
                    COMB))]))

; sum : list-of-num -> num
(define (sum l)
    (combine-nums l 0 +))

; product : list-of-num -> num
(define (product l)
    (combine-nums l 1 *)
Reuse from existing abstractions:

; sum : list-of-num -> num
(define (sum l)
  (foldr + l 0))

; product : list-of-num -> num
(define (product l)
  (foldr * l 1))
Reuse from existing abstractions:

```java
int sum(List l) {
    Enumerator e = l.elements();
    int s = 0;
    while (e.hasMoreElements()) {
        Integer i = (Integer)e.nextElement();
        s = s + i.intValue();
    }
    return s;
}
```
Theme 4: Reuse

Reuse by class extension:

class Into extends Door {
    ...
    Path escapePath(Person p) {
        return this.next.escapePath(p);
    }
}

class Short extends Into {
    ...
    Path escapePath(Person p) {
        if (p.height <= this.height)
            return super.escapePath(p);
        else
            return new Fail();
    }
    // everything else is like Into
}
A good design process focuses your energy on two deeply creative problems:

- choosing and defining a data representation
- implementing the body of a function/method
Theme 5: Creativity

Problem: choose a data definition for mazes

class Room {
    Door left;
    Door right;
    ... }
abstract class Door { ... }
class Escape extends Door { ... }
class Into extends Door {
    Room next;
    ...
}
Theme 5: Creativity

Problem: combine images to check for disguises

; same-person-maybe-disguised? : ; image image image image image -> bool
(define (same-person-maybe-disguised? p p2 g b)
  (or (image=? p p2)
      (wearing-glasses? p p2 g)
      (wearing-beard? p p2 b)
      (image=? p (add-beard (add-glasses p2 g) b)))))

- Which part was automatic from contracts?
- Which part required creativity?
Theme 5: Creativity

Problem: produce an image's negative

; photo-negative : image -> image
(define (photo-negative i)  
  (color-list->image
    (negate-colors (image->color-list i))
    (image-width i)
    (image-height i)))

- Which part was automatic from contracts?
- Which part required creativity?
Theme 5: Creativity

Data Representation and Contract

Examples

Maybe Abstract

Use Existing

Template

Body

Trivial Cases

Recur on Smaller

Test

38
Theme 6: Programming Tools

• Structures
• Functions
• Classes
• Methods
• Contracts in comments and code
• Local declarations
• Assignment
• Computational complexity
Themes in the Final Exam

Expect the final exam to hit all of these themes:

- Data Structures
- Data Drives Design
- Contracts
- Reuse
- Creativity
- Programming Tools

More details next time