STUDENTS OFFERING SUPPORT

CS116 Final Exam-AID Session

'RAISING MARKS, RAISING MONEY, RAISING ROOFS'

Students Offering Support is a nationally recognized Canadian Charity.

Today's Agenda

- Review of CS115: What should you already know? (*Design Recipe*)
- Functional Abstraction in Scheme
- Generative Recursion
- Accumulative Recursion
- Mutation in Scheme: Our Memory Model

Today's Agenda (cont.)

- Basic Programming in Python
- Strings, Lists and Structures
- Iteration, Dictionaries and Classes
- File Input/Output
- Closing Out
- Questions

Example 1

- On the midterm, you'll be asked to supply certain parts of the **design recipe**
- Example: Can we suggest a contract, examples, and tests for a function that consumes a number n and prints out a list that prints out a list of the first n squares?

Example 1

- Contract:
- int[>=0] -> (listof int[>=0])[non-empty]
- Examples: (list-o-sqrs 4) -> (listof 0 1 4 9 16)
- Tests:
- (check-expect (list-o-sqrs 0) (list 0))
- (check-expect (list-o-sqrs 1) (list 0 1))
- (check-expect (list-o-sqrs 5) (list 1 4 9 16 25))

Module 2

- Functional Abstraction
- You can use functions as values!
- (map f list)
  - map applies f to every item in the list.
- Contract: (X -> Y) (listof X) -> (listof Y)
- (build-list num f)
- Ex. (build-list 3 double) -> (list 0 2 4)

map, filter, foldr

- (filter f list)
  – return list of values where function is true
- Ex. ((filter odd? (list 2 4 1 6 8 3 5)) -> (list 1 3 5)

- (foldr f base list)
  – applies function recursively on list
- Ex. (foldr + 0 (list 5 6 7)) -> 5+6+7+0=18

The keyword lambda

- The **lambda** keyword allows definition of anonymous functions
- You should use **lambda** if they're one time use
- Ex: (lambda (x) (= (/ x 7) 0))
- Ex: (lambda (a b) (sqrt (+ (* a a) (* b b))))
- Ex: (map (lambda (x) (+ x 2)) (list 1 4 7))

Question

- Determine if two functions are inverses between 2 numbers inclusively.

- Ex. (is-inverse? abs sqr -1 1)
- -1 ≠ abs(sqr -1)
- 0 = abs (sqr 0)
- 1 = abs (sqr 1)
- Returns: False

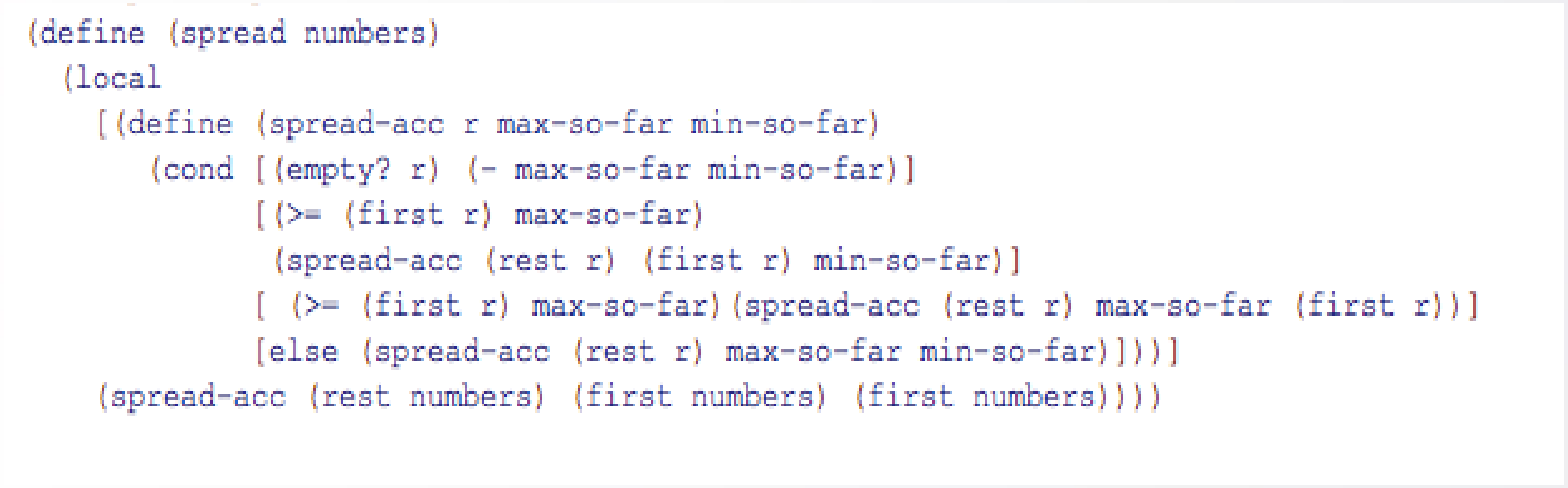
```
(define (is-inverse? f g lo hi)
  (andmap (lambda (x) (equal? x (g (f x))))
                  (build-list (- hi lo -1) (lambda (x) (- hi x)) )))
```

Module 3

- **Accumulative Recursion**
- It’s basically recursion, but with memory.
- We store a value for the next recursive call
- **Template**
  - – Describe how to store the knowledge
  - – Use structural recursion to compute
  - – Use Local to recurse
  - – Figure out the base case

Example

- Create a function that produces the difference between the greatest and smallest numbers in a list.



```
(define (spread numbers)
  (local
    [(define (spread-acc r max-so-far min-so-far)
       (cond [(empty? r) (- max-so-far min-so-far)]
             [(>= (first r) max-so-far)
              (spread-acc (rest r) (first r) min-so-far)]
             [ (>= (first r) max-so-far)(spread-acc (rest r) max-so-far (first r))]
             [else (spread-acc (rest r) max-so-far min-so-far)]))]
    (spread-acc (rest numbers) (first numbers) (first numbers))))
```

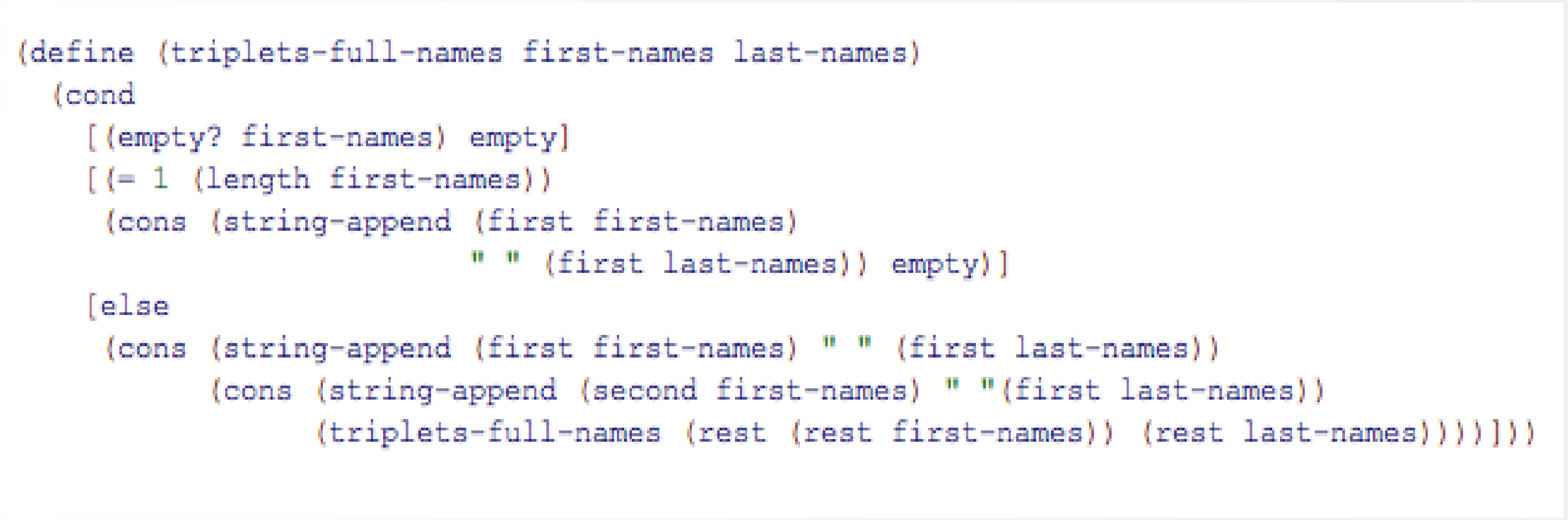
Testing

- Test all calls to the accumulator helper
- For accumulator helper test base case, near base case and non-base base.
- Remembering what values your variables hold during a given call will make debugging easier

Module 4

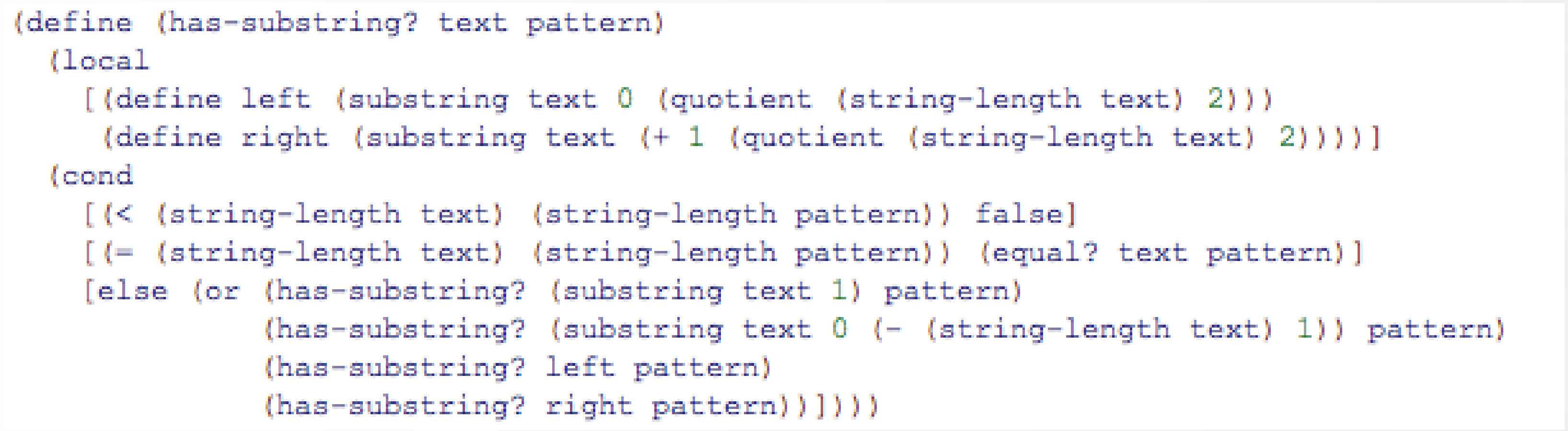
- What is generative recursion?
  - Recursion that isn’t based on the data structure
- Example: Create a function which determines the GCD of 2 numbers.
-
```
(define (GCD divisor remain)
  (cond
    [(zero? remain) divisor]
    [else (GCD remain (remainder divisor remain))]))
```

- Example: Create a function which consumes a list of first names and a list of last names of triplets. The function will produce a list of the triplets full names.



```
(define (triplets-full-names first-names last-names)
  (cond
    [(empty? first-names) empty]
    [(= 1 (length first-names))
     (cons (string-append (first first-names)
                          " " (first last-names)) empty)]
    [else
     (cons (string-append (first first-names) " " (first last-names))
           (cons (string-append (second first-names) " "(first last-names))
                 (triplets-full-names (rest (rest first-names)) (rest last-names))))]))
```

- Create a function that consumes two strings, text and pattern. If the pattern is in text it will produce true, and false otherwise!



```
(define (has-substring? text pattern)
  (local
    [(define left (substring text 0 (quotient (string-length text) 2)))
     (define right (substring text (+ 1 (quotient (string-length text) 2))))]
  (cond
    [(< (string-length text) (string-length pattern)) false]
    [(= (string-length text) (string-length pattern)) (equal? text pattern)]
    [else (or (has-substring? (substring text 1) pattern)
              (has-substring? (substring text 0 (- (string-length text) 1)) pattern)
              (has-substring? left pattern)
              (has-substring? right pattern))])))
```

Sorting Algorithms

- You learned about three sorting algorithms;
  - Insertion sort
  - Selection sort
  - Quick sort

Insertion Sort

- Insertion sort is the first sorting algorithm you learned
- Idea: Select the first element in the list
- Then sort the rest of the list, and place the first element in the correct position
- Example – (list 2 3 6 5 1 4)

Selection Sort

- The next algorithm you learned was selection sort; it’s a little different from insertion sort
- Idea: select the smallest element from the list
- Then sort the other elements, and place the element at the front of the sorted list
- Example - (list 2 3 6 5 1 4)

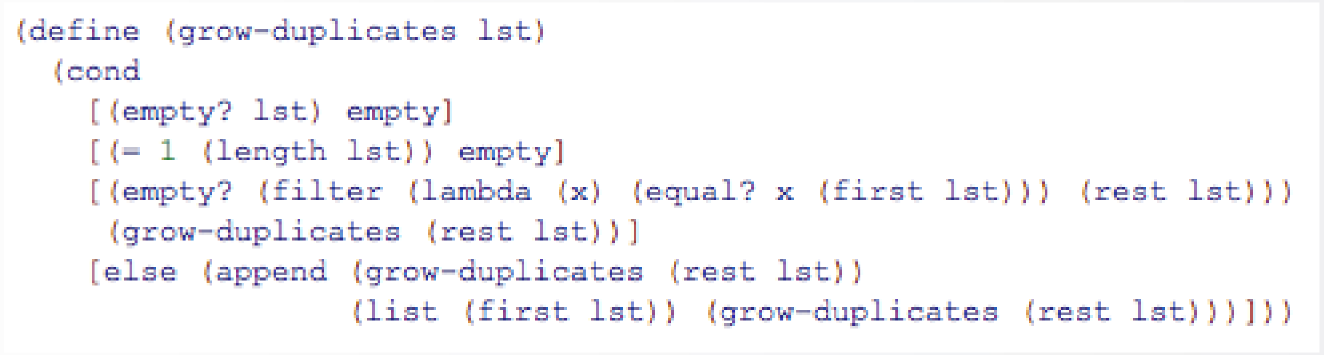
Quick Sort

- Quick Sort is unlike insertion/selection sort
- It’s based on a divide and conquer concept
- We’ll use the first element as a pivot, and divide the list into two smaller lists.
- Then we’ll sort both lists and merge the sorted lists
- Example - (list 3 2 6 5 1 4)

Worst Case Runtimes

- With sorting algorithms, we can compare the worst case running times
- You learned about constant, linear, quadratic, and exponential running times
- What are the worst case runtimes of insertion sort, selection sort, and quick sort?

- What is the best and worst case running time of grow duplicates?



```
(define (grow-duplicates lst)
  (cond
    [(empty? lst) empty]
    [(= 1 (length lst)) empty]
    [(empty? (filter (lambda (x) (equal? x (first lst))) (rest lst)))
     (grow-duplicates (rest lst))]
    [else (append (grow-duplicates (rest lst))
                  (list (first lst)) (grow-duplicates (rest lst)))]))
```

The worst case running time happens when we repeatedly have to call the fourth case. It has an exponential running time. The best case is when the list is empty. It has constant running time.

Module 5

- Mutation: changing a variable’s value during a program’s runtime
- In Scheme, we use **set!** to mutate variables
- In Python, we’ll use mutation a lot
- **set!** returns the value **(void)**

Scheme’s Memory Model

- **Aliasing**: when more than one variable points to the same thing in memory
- Set! Breaks the alias and creates a new item

- **Exceptions**:
- **Functions**
- Ex. (define (f1 x) (* x 2)) (define f2 f1) (define (f3 x) (f1 x))
- What are the values of f1, f2, f3 after (set! f1 sqr)
- (define (f1 x) (sqr x)) (define (f2 x) (* x 2)) (define (f3 x) (f1 x))

- **Structures**
- If we use set!, we point to a (possibly new) entry in the table.
- If we use the structure field mutators, it will affect the entry in the table, and so it will affect everything that points to it.

Example

- What is the output of the following code?

```
(define x 3)
(define y x)
(begin
    (set! y 5)
    y ;; what is y? 5
    x ;; what is x? 3
) ;; what is the return value? (void)
```

Example w/ Structures

- What’s the final value of each person?

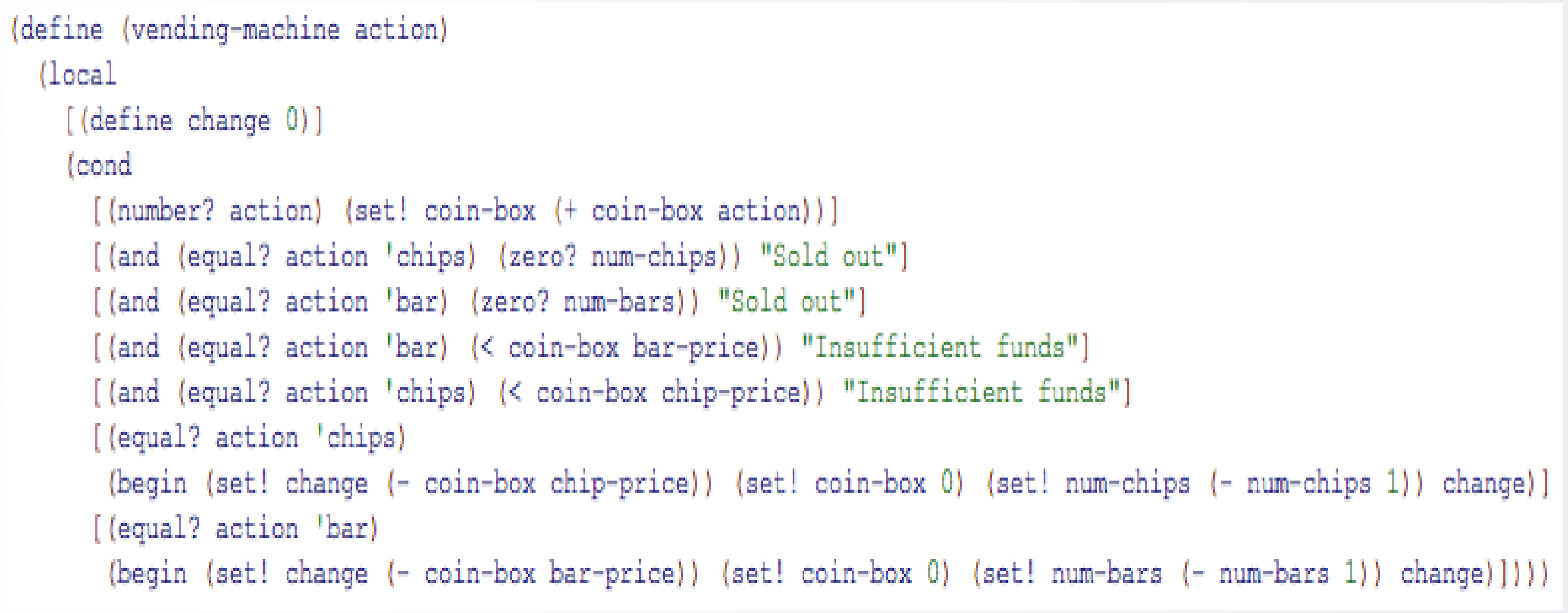
```
(define-struct person (height weight))
(define bob (make-person 150 200))
(define john bob)
(define dave (make-person 150 200))
(set-person-height! john 100)
;; bob, john, dave? (what’s this look like in
   memory?)
```

- John=Bob=(make-person 100 200)

Question

- Create a function vending-machine which consumes a number or a symbol. If money is put in the vending machine then the value of the coin box is updated. If bar or chips are entered then: Sold Out is produced if there are no more of that thing left. Insufficient funds if there is not enough money in the coin box to buy the item, or the amount of change if the item is bought.

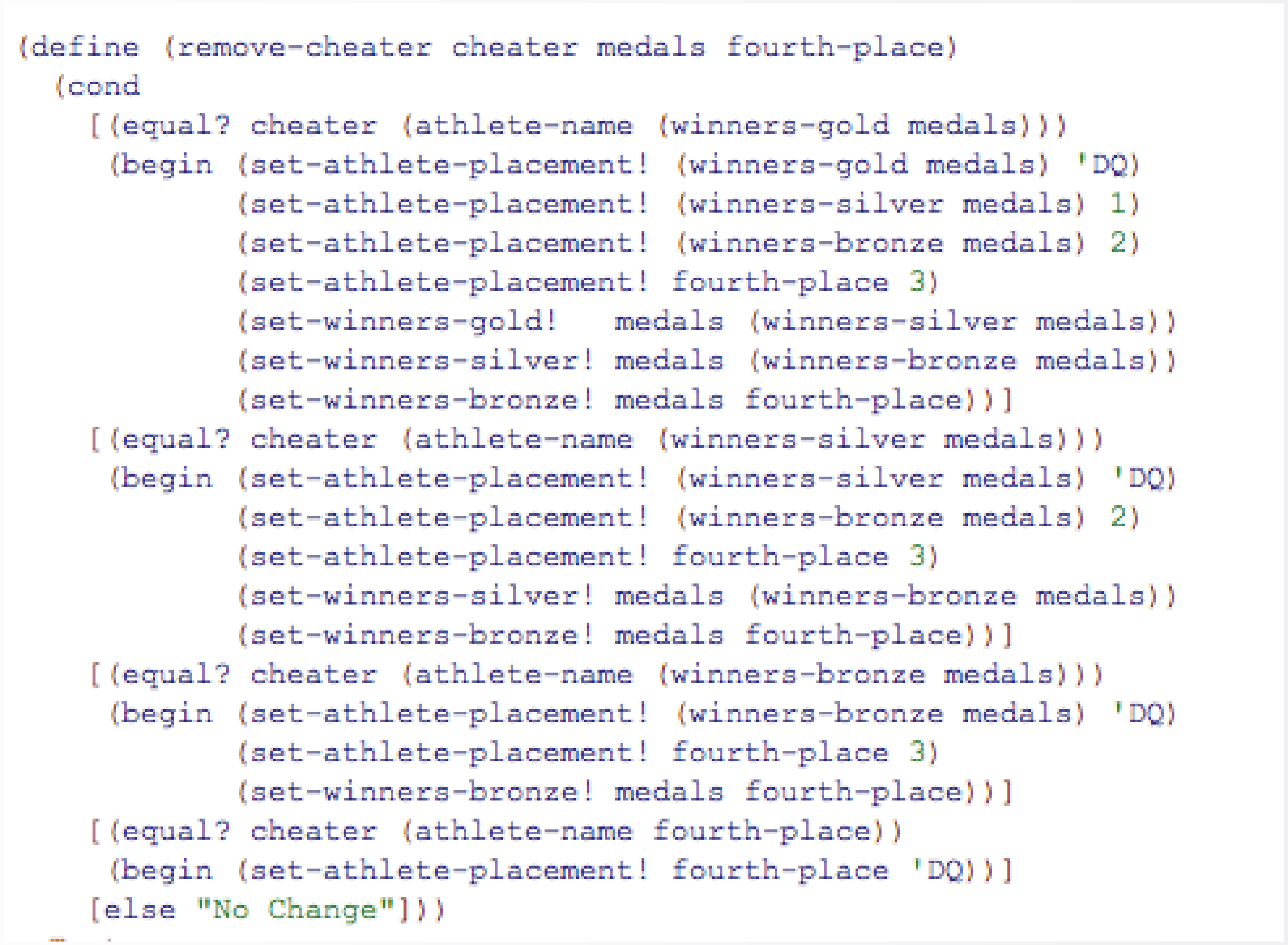
Answer



```
(define (vending-machine action)
  (local
    [(define change 0)]
    (cond
      [(number? action) (set! coin-box (+ coin-box action))]
      [(and (equal? action 'chips) (zero? num-chips)) "Sold out"]
      [(and (equal? action 'bar) (zero? num-bars)) "Sold out"]
      [(and (equal? action 'bar) (< coin-box bar-price)) "Insufficient funds"]
      [(and (equal? action 'chips) (< coin-box chip-price)) "Insufficient funds"]
      [(equal? action 'chips)
       (begin (set! change (- coin-box chip-price)) (set! coin-box 0) (set! num-chips (- num-chips 1)) change)]
      [(equal? action 'bar)
       (begin (set! change (- coin-box bar-price)) (set! coin-box 0) (set! num-bars (- num-bars 1)) change)])))
```

Example

- A function consumes an athlete who is a cheater in a race. A winners structure and the fourth-place athlete. The function will change around the winners structure if the cheater placed and produces “No Change” if the cheater did not place.



```
(define (remove-cheater cheater medals fourth-place)
  (cond
    [(equal? cheater (athlete-name (winners-gold medals)))
     (begin (set-athlete-placement! (winners-gold medals) 'DQ)
            (set-athlete-placement! (winners-silver medals) 1)
            (set-athlete-placement! (winners-bronze medals) 2)
            (set-athlete-placement! fourth-place 3)
            (set-winners-gold!   medals (winners-silver medals))
            (set-winners-silver! medals (winners-bronze medals))
            (set-winners-bronze! medals fourth-place))]
    [(equal? cheater (athlete-name (winners-silver medals)))
     (begin (set-athlete-placement! (winners-silver medals) 'DQ)
            (set-athlete-placement! (winners-bronze medals) 2)
            (set-athlete-placement! fourth-place 3)
            (set-winners-silver! medals (winners-bronze medals))
            (set-winners-bronze! medals fourth-place))]
    [(equal? cheater (athlete-name (winners-bronze medals)))
     (begin (set-athlete-placement! (winners-bronze medals) 'DQ)
            (set-athlete-placement! fourth-place 3)
            (set-winners-bronze! medals fourth-place))]
    [(equal? cheater (athlete-name fourth-place))
     (begin (set-athlete-placement! fourth-place 'DQ))]
    [else "No Change"]))
```

Python

- Assignment in Python is simple.
- X = “Bob”
- We can print text on the screen
- print X -> “Bob”
- Python uses Order of Operations for arithmetic, just like in math.
- Y = 4 + 3 * 7

Programming Pitfalls

- Indentation of Code
  - Also new in Python, indentation now matters
- Locally Defined Variables
  - This is not new; we saw this in Scheme also
- Integer Division
  - This is new in Python, Scheme was very exact

Example

- Float and Int are two types of numbers
- Consider the following operations:

A) 7 / 2 -> 3

B) 7 / 2.0 -> 3.5

C) int(0.6) -> 0

D) int(7.0) / 2 -> 3

- **Creating Functions:**
- def fname (arguments):

```
    statements
return
```

- **Conditionals:**
- if test:

```
    outcome
else
    alternative
```

Functions

- Consider the following code:

```
def add_three(x):
    x = x + 3
    print x
x = 4
add_three(2) -> Prints 5
print x -> 4
```

Keyboard Input

- We've seen how to output to the screen using **print**
- If we wanted to accept input, then we would use the **raw_input(prompt)** function
- It places the prompt on the screen and waits for user input until a termination character

Example

- Output, and input

```
greeting = “Hi there. What is your name? ”
name = raw_input(greeting)
print “Hello, “ + name + “. Nice to meet you.”
```

Mutation

- Variables must be defined first with a value, then you can mutate them

```
X = ‘I Love Finals!’
X = ‘Just Kidding!’
X -> ‘Just Kidding!’
```

Python’s Design Recipe

- Even though Python is different from Scheme, we’ll still use the **design recipe** when designing programs
- It will follow the same format as the recipe in Scheme

Module 7

- Strings, Lists, Structures
- Like Scheme, Python uses strings, lists and structures
- We’ll have a look at each of them and the functionality in Python

Strings and Substrings

- Unlike Scheme, there's some arithmetic operations you can perform on strings
- You can add and multiply strings (by a number)
- You can also access substrings in Python
- Python also gives certain methods to types
- Examples

Examples (lots of them)

1. “ab” + “cd” -> “abcd”
2. “abcd” * 3 -> “abcdabcdabcd”
3. “xy” * “z” -> error
4. “far” in “farm” -> true
5. s = “abcdefg”
6. s[1:4] -> “bcd”
7. s[:3] -> “abc”
8. s[4:] -> “efg”
9. s[:] -> error
10. s = “hi my name is”
11. s.upper() -> “HI MY NAME IS”
12. s.split() -> [‘hi’, ‘my’, ‘name’, ‘is’]

Lists, Mutation and Aliasing

- Python also has lists, but unlike Scheme, lists are mutable
- We treat lists as a complex type, and if mutation occurs inside a function, we see it outside

- **Aliasing**
- If it modifies an existing list, then all objects that refer to that list will be changed.
- If it makes a new one, that new list is a new object, and it will break the alias.

Example

- What is the output of a after this code is run?

```
a = [1, 3, 5, 7]
a[2] = 10          a=[1,3,10,7]
a.append(5)        a=[1,3,10,7,5]
a.sort()           a=[1,3,5,7,10]
a = a[:3]          a=[1,3,5]
a -> ?
```

Example

```
Lunch = ['burger', 'fries', 'pop']
Dinner = Lunch
Dinner [1] = 'apple'
def dessert (x):
    Lunch.append('cookie')
Lunch = ['salad', 'water']
Lunch -> ['salad', 'water']
Dinner -> ['burger', 'apple', 'pop', 'cookie']
```

Functional Abstraction

- Python has the functions **map** and **filter**, which we can use

  map (function, list)

  filter (function, list)

- Python also has anonymous functions and **lambda**

  lambda x, y: expression

Examples

- Create a function which consumes a list of strings, subject_lines, and a string, junk. It produces a list of strings in which all of the strings with the word junk in them are removed from the list.
- def junk_mail(subject_lines, junk):
- return filter (lambda x: not(junk in x.split()), subject_lines)

- Design a function create_cheer which gets the user to input their favourite team and number of times they want to repeat their cheer. “Go Team, Go!” Each repetition will be printed on a new line and the number of ! After the cheer will be equal to the number of repetitions.

```
def cheer(gocgo, num):
    if num == 1:
        print gocgo
    else:
        print gocgo
        cheer(gocgo, (num - 1))
        return

def create_cheer():
    prompt1 = raw_input("Enter name of favourite country: ")
    prompt2 = int(raw_input("Enter times to repeat cheer >= 1: "))
    cheer1 = "Go, "+prompt1+", Go"+"!"*(prompt2)
    cheer(cheer1, prompt2)
    return
```

Module 8

- Where Python really differs from Scheme is in this module
- Python uses iteration extensively, and places limits on recursion
- Python also has the dictionary structure, which we'll see is very efficient
- Lastly, Python provides this notion of a class.

Iteration

- There are two types of iterations or loops in Python: **while** loops and **for** loops
- Python uses iteration heavily
- **while** loops use a condition to determine when to terminate
- **for** loops iterate over a list of things

Examples

- What happens in these two loops?

```
x = 3
while (x > 0)
    print x
    x = x – 1
Prints 3, 2, 1

sum = 0
    for i in [1, 5, 6]
    sum = sum + i
print sum
-> prints 12
```

- Asks the user how many employees are working tomorrow. For each employee it asks the user when the employee starts and ends. Print how many times no one is working and how many times more than one person is working.

```
def check_schedule():
    prompt1 = "How many employees are working tomorrow? "
    startprompt1 = "When does employee "
    startprompt2 = " start? "
    endprompt1 = "When does employee "
    endprompt2 = " end? "
    message1 = "The number of hours with more than one employee working is "
    message2 = "The number of hours with no employee working is "
    num_empl=int(raw_input(prompt1))
    hours = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
    no_empl=0
    lots_empl=0
    empl_count=1
    if num_empl==0:
        no_empl=24
    else:
        while empl_count<=num_empl:
            start=raw_input(startprompt1 + str(empl_count) + startprompt2)
            end=raw_input(endprompt1 + str(empl_count) + endprompt2)
            hours_worked = range(int(start), int(end))
            empl_count=empl_count + 1
            for h in hours_worked:
                hours[h]=hours[h]+1

        for hr in hours:
            if hr==0:
                no_empl=no_empl+1
            elif hr>1:
                lots_empl=lots_empl+1
    print message1 + str(lots_empl)
    print message2+ str(no_empl)
    return
```

Dictionaries

- **Dictionaries** are structures that hold key-value pairs (the key is unique)
- We’ll use curly braces {} instead of [] to represent them
- The difference between dictionaries and lists is that lookup time in a dictionary is super fast
- Let’s take a look at some dictionary methods and an application. (Example)

Example

- Consider a dictionary named directory that has your UWID as the key and your name as the value.

```
directory = {201999999: ‘Bob’, 202000000:
    ‘John’}
directory[201999999] -> ‘Bob’
directory.has_key[201999999] -> True
directory.keys() -> [201999999, 20200000]
```

- Create a function which has two dictionaries of book titles where the key is the first letter of the book. Combine the two dictionaries so that there are no duplicates.

```
def join_book_collections(d1,d2):
    for books in d1:
        if d2.has_key(books):
            d1[books] = d1[books]+d2[books]
    for books in d2:
        if not(d1.has_key(books)):
            d1[books]=d2[books]
    for books in d1.keys():
        l=[]
        lst=[]
        l=d1[books]
        for ele in l:
            if not(ele in lst):
                lst=lst+[ele]
        lst.sort()
        d1[books]=lst
    return d1
```

Classes

- In Scheme, we used structures to hold compound data
- In Python, we'll use a **class**
- The class constructor is always ClassName()
- From there, we'll have to set each field on our own (we don't have to do this, but it's extra)

Example

- Code for creating a class object

```
class Card:
    'Fields: value, suit'


c = Card()
c.value = 4
c.suit = 'clubs'
```

Module 9

- In Python, the last feature we learn about is file input/output
- Formatting output, tuples, exception handling
- Opening/closing files, reading and writing data

Formatting Output

- When outputting to the screen, we saw the we could use **print**
- If we want to output multiple things on one line, we could just **add** them together
- But we can do something else: use formatting
- Formatting uses the % placeholder and then lists the variables after

Example

- Output, and input (cleaner)

```
greeting = “Hi there. What is your name? ”
name = raw_input(greeting)
print “Hello, %s. Nice to meet you.” % name
print “The first three odd numbers are %d, %d and %d.” % (1, 3, 5)
```

- Here, we’re using a “tuple” of (1, 3, 5)

Exception Handling

- When executing code that might fail, Python allows for trying code and catching exceptions
- In a **try/except** block, code is tried, and if it fails, enters the except block, otherwise it is executed normally
- We'll see that these blocks are used when reading/writing files

Example

- Note this code fails without the try/except block

```
def square (x):
    try:
        s = x * x
        print s
    except:
        print “That’s not a number!”
square(9) -> 81
square(‘Bob’) -> ‘That’s not a number!’
```

Files

- Finally, we see how to open/close/read/write files
- We open a file using **file()**, and close using **close()**
- There is a reading mode and a writing mode when opening a file
- There are multiple functions when reading/writing to files.

Example

```
F = file(‘input.txt’, ‘r’)
myStuff = F.readLines() ## myStuff is a list
F.close()
F = file(‘output.txt’, ‘w’)
F.writelines(myStuff)
F.close()
## What does this code do? Output.txt is a copy
    of input.txt
```

Example

- Create a function which takes in a file that is a book. Create a dictionary of all the words in the book and the number of times each word appears.

```
def rid_punct(st):
    return_str=""
    letters =['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z',' ']
    for i in st:
        i.lower()
        if i in letters:
            return_str=return_str+i
    return return_str
#Either one works
def rid_punct2(s):
    outS=""
    badchar="'-;.!/:,."+'"'
    for c in s:
        if not(c in badchar):
            outS=outS+c
    return outS

def book_reader(fn):
    try:
        F=file(fn, 'r')
        lines = F.readlines()
    except:
        print "Could not open file"
        return ({})
    lines = map(ridpunct, lines)
    lines=map(lambda x: x.lower(),lines)
    lines=map(lambda x: x.split(), lines)
    dic={}
    for line in lines:
        for s in line:
            if dic.has_key(s):
                dic[s]=dic[s]+1
            else:
                dic[s]=1
    return dic
```

Module 10

- CS116 ends off by discussing design choices in Python programs
- Lists are used when order of objects matter
- Dictionaries are great when you know you want to search often
- Classes are used for storing compound data
- There’s not always a clear answer

About the Final Exam

- There will be a "strong emphasis" on Modules 6-10 (The Python modules)
- You may get a reference sheet with Scheme and Python operations. Take some time to review this sheet before the exam; it will be invaluable!
- "Relax! Read this instruction as often as needed."

Closing Remarks

- Questions?
- Good luck on your final! =)