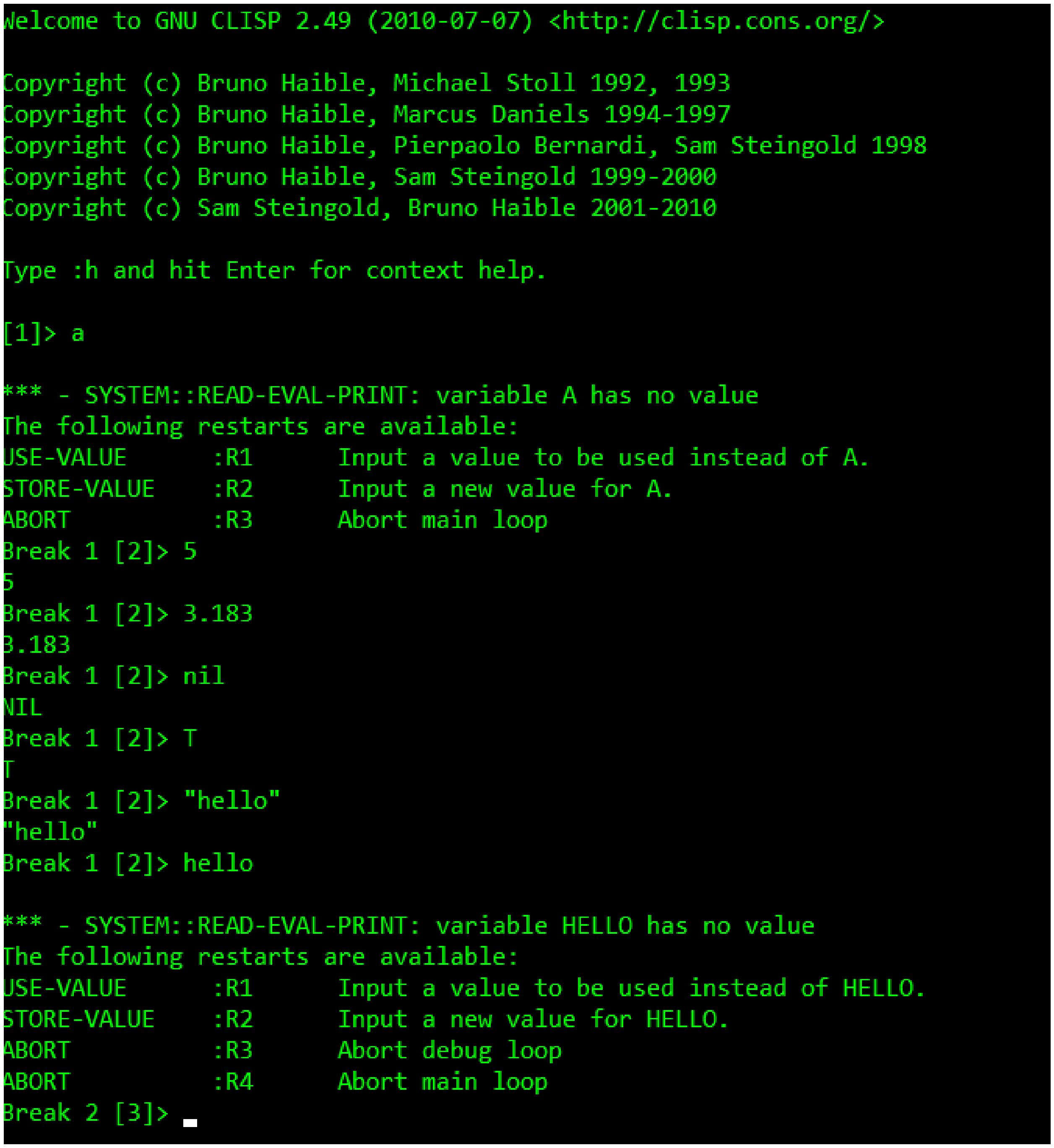
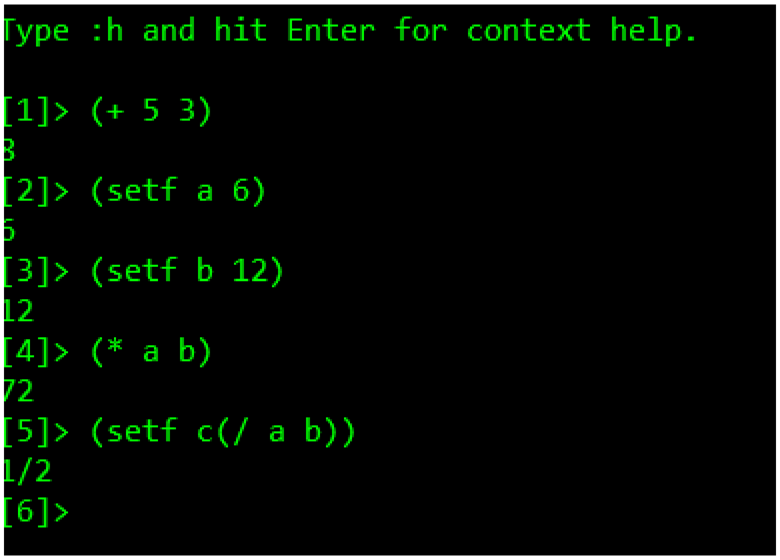
1. Start it and play around with it

- Type in some symbols and numbers (‘ a, 5, 3.183, nil, T, “hello”, etc.) and see what results you get.



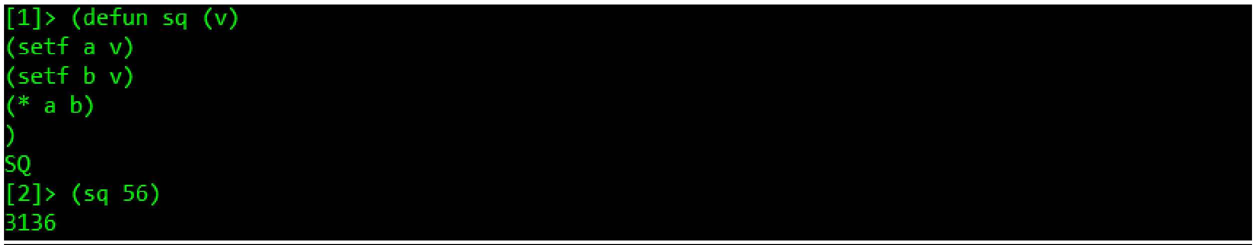
```
Welcome to GNU CLISP 2.49 (2010-07-07) <http://clisp.cons.org/>

Copyright (c) Bruno Haible, Michael Stoll 1992, 1993
Copyright (c) Bruno Haible, Marcus Daniels 1994-1997
Copyright (c) Bruno Haible, Pierpaolo Bernardi, Sam Steingold 1998
Copyright (c) Bruno Haible, Sam Steingold 1999-2000
Copyright (c) Sam Steingold, Bruno Haible 2001-2010

Type :h and hit Enter for context help.

[1]> a

*** - SYSTEM::READ-EVAL-PRINT: variable A has no value
The following restarts are available:
USE-VALUE      :R1      Input a value to be used instead of A.
STORE-VALUE    :R2      Input a new value for A.
ABORT          :R3      Abort main loop
Break 1 [2]> 5
5
Break 1 [2]> 3.183
3.183
Break 1 [2]> nil
NIL
Break 1 [2]> T
T
Break 1 [2]> "hello"
"hello"
Break 1 [2]> hello

*** - SYSTEM::READ-EVAL-PRINT: variable HELLO has no value
The following restarts are available:
USE-VALUE      :R1      Input a value to be used instead of HELLO.
STORE-VALUE    :R2      Input a new value for HELLO.
ABORT          :R3      Abort debug loop
ABORT          :R4      Abort main loop
Break 2 [3]>
```

- Call some basic functions (+ 5 3), (setf a 6), (setf b 12), (* a b), (set c(/ a b)), etc.

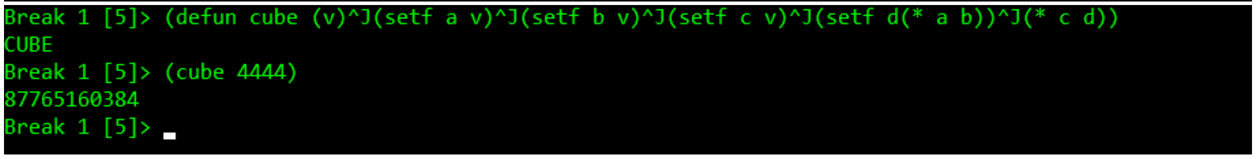


```
Type :h and hit Enter for context help.

[1]> (+ 5 3)
8
[2]> (setf a 6)
6
[3]> (setf b 12)
12
[4]> (* a b)
72
[5]> (setf c(/ a b))
1/2
[6]>
```

- Write some simple functions like square, cube, etc.

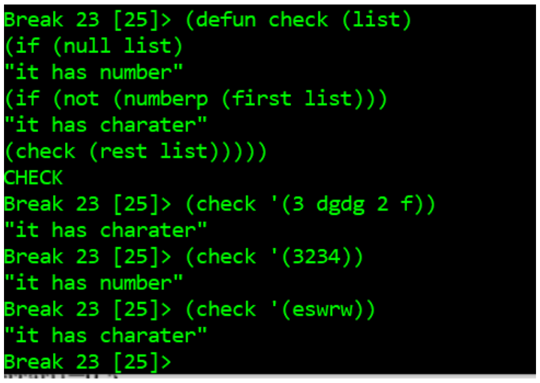


```
[1]> (defun sq (v)
(setf a v)
(setf b v)
(* a b)
)
SQ
[2]> (sq 56)
3136
```



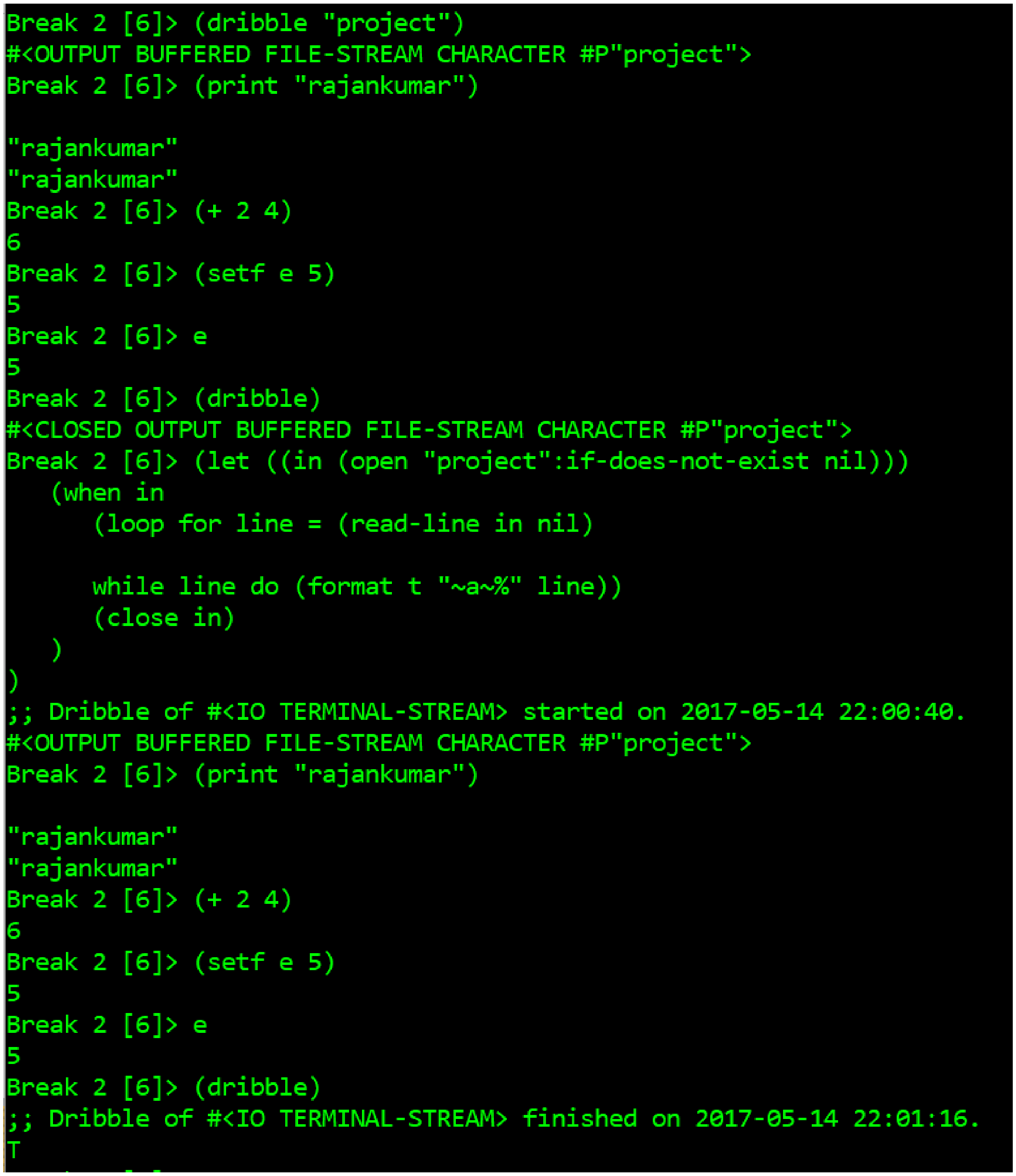
```
Break 1 [5]> (defun cube (v)^J(setf a v)^J(setf b v)^J(setf c v)^J(setf d(* a b))^J(* c d))
CUBE
Break 1 [5]> (cube 4444)
87765160384
Break 1 [5]>
```

- Write a few list functions like determining if all elements of a list are numbers, characters, etc.



```
Break 23 [25]> (defun check (list)
(if (null list)
"it has number"
(if (not (numberp (first list)))
"it has charater"
(check (rest list)))))
CHECK
Break 23 [25]> (check '(3 dgdg 2 f))
"it has charater"
Break 23 [25]> (check '(3234))
"it has number"
Break 23 [25]> (check '(eswrw))
"it has charater"
Break 23 [25]>
```

2. Start a dribble session. Your first command should be (print “name”) with your name in quotes.



```
Break 2 [6]> (dribble "project")
#<OUTPUT BUFFERED FILE-STREAM CHARACTER #P"project">
Break 2 [6]> (print "rajankumar")

"rajankumar"
"rajankumar"
Break 2 [6]> (+ 2 4)
6
Break 2 [6]> (setf e 5)
5
Break 2 [6]> e
5
Break 2 [6]> (dribble)
#<CLOSED OUTPUT BUFFERED FILE-STREAM CHARACTER #P"project">
Break 2 [6]> (let ((in (open "project":if-does-not-exist nil)))
   (when in
      (loop for line = (read-line in nil)

      while line do (format t "~a~%" line))
      (close in)
   )
)
;; Dribble of #<IO TERMINAL-STREAM> started on 2017-05-14 22:00:40.
#<OUTPUT BUFFERED FILE-STREAM CHARACTER #P"project">
Break 2 [6]> (print "rajankumar")

"rajankumar"
"rajankumar"
Break 2 [6]> (+ 2 4)
6
Break 2 [6]> (setf e 5)
5
Break 2 [6]> e
5
Break 2 [6]> (dribble)
;; Dribble of #<IO TERMINAL-STREAM> finished on 2017-05-14 22:01:16.
T
```

3. Create a list that stores the courses that you are taking this semester where the courses are stored in sublists as (subjectnumber) such as (CSCI 755) so that your list will look like this: ((CSCI 755) (EENG 641) (ACCT 801) (CHEM 601) (ENGY 720)) and then create 3 other lists for fictitious CSCI and/or EENG majors. All students should have at least one CSCI class. They should all have at least 2 courses, no more than 7

Write the following functions.

```
acct = ['ACCT 840', 'ACCT 765', 'ACCT 715', 'ACCT 665'];
chem = ['CHEM 820', 'CHEM 785', 'CHEM 725', 'CHEM 625'];
engy = ['ENGY 805', 'ENGY 790', 'ENGY 720', 'ENGY 685'];
csci = ['CSCI 755', 'CSCI 640', 'CSCI 610', 'CSCI 710'];
eeng = ['EENG 740', 'EENG 635', 'EENG 825', 'EENG 720'];

all_courses = [acct, chem, engy, csci, eeng];

student1 = ['CSCI 640', 'CSCI 610', 'ENGY 685', 'EENG 635'];
student2 = ['ENGY 805', 'ENGY 720', 'CSCI 755', 'EENG 740'];
student3 = ['CHEM 820', 'CHEM 725', 'CSCI 710', 'EENG 635'];
student4 = ['ACCT 765', 'ACCT 665', 'CSCI 610', 'EENG 720'];

all_students = [student1, student2, student3, student4];
```

a. Given a subject, print all courses of that subject

```
def get_Course(subject):
        for course in all_courses:
                if (subject in course):
                        print (subject, "exists in this course : ", course, "\n")
```

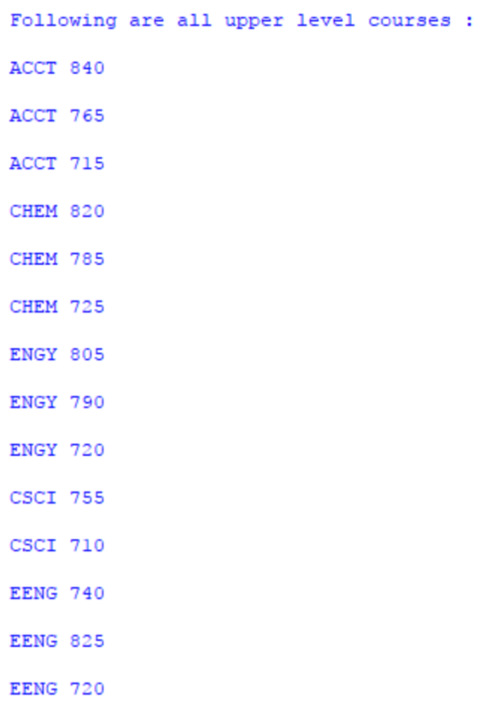
get_Course('CSCI 640')

```
CSCI 640 exists in this course :  ['CSCI 755', 'CSCI 640', 'CSCI 610', 'CSCI 710
']
```

b. Print all upper level courses (7xx and 8xx courses)

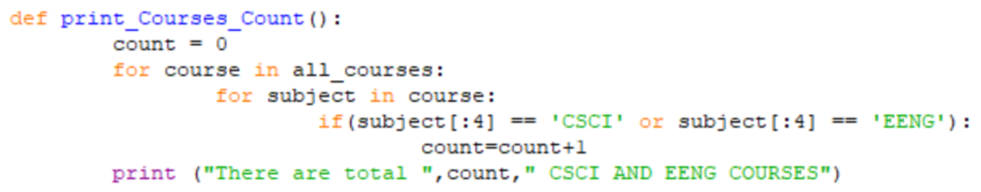
```
def print_Upper_Courses():
        print ("Following are all upper level courses :\n")
        for course in all_courses:
                for subject in course:
                        if(int(subject[-3:-2]) > 6):
                                print (subject,"\n")
```

Print_Upper_Courses()



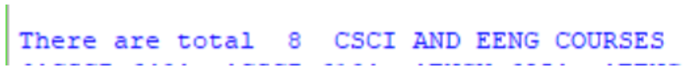
```
Following are all upper level courses :

ACCT 840

ACCT 765

ACCT 715

CHEM 820

CHEM 785

CHEM 725

ENGY 805

ENGY 790

ENGY 720

CSCI 755

CSCI 710

EENG 740

EENG 825

EENG 720
```

c. Count the number of courses that are either CSCI or EENG and return that number



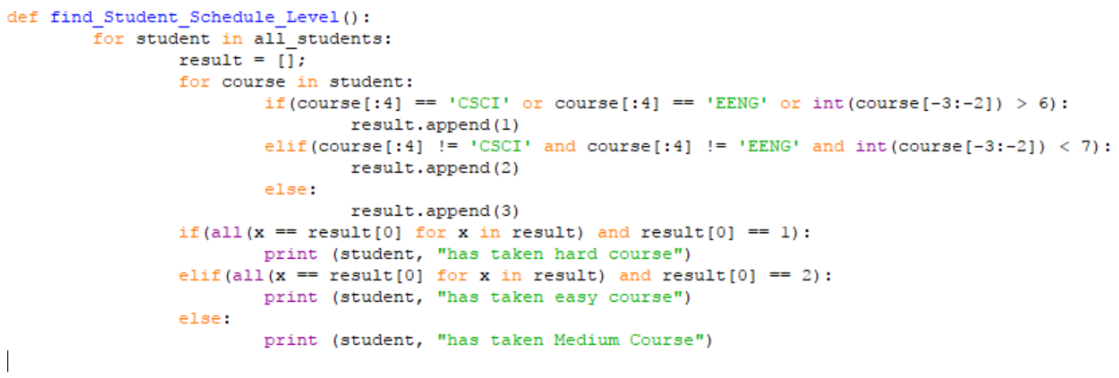
```
def print_Courses_Count():
        count = 0
        for course in all_courses:
                for subject in course:
                        if(subject[:4] == 'CSCI' or subject[:4] == 'EENG'):
                                count=count+1
        print ("There are total ",count," CSCI AND EENG COURSES")
```

print_Courses_Count()



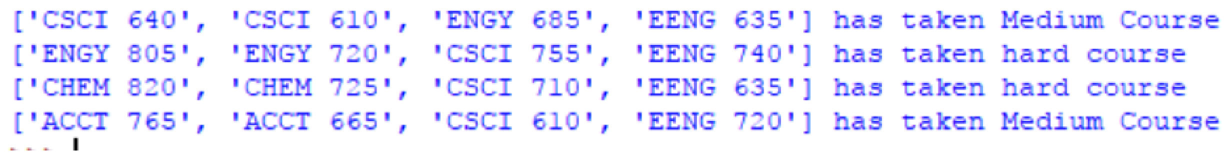
```
There are total  8  CSCI AND EENG COURSES
```

d. Given a course list, determine whether a given schedule is “hard”, “easy” or “inbetween”. Where a “hard” schedule is one where the student is taking only upper level courses or only CSCI/EENG courses; an “easy” schedule is one where the student is taking only lower level courses or is one with no CSCI/EENG courses; and “inbetween” is everything else.



```
def find_Student_Schedule_Level():
        for student in all_students:
                result = [];
                for course in student:
                        if(course[:4] == 'CSCI' or course[:4] == 'EENG' or int(course[-3:-2]) > 6):
                                result.append(1)
                        elif(course[:4] != 'CSCI' and course[:4] != 'EENG' and int(course[-3:-2]) < 7):
                                result.append(2)
                        else:
                                result.append(3)
                if(all(x == result[0] for x in result) and result[0] == 1):
                        print (student, "has taken hard course")
                elif(all(x == result[0] for x in result) and result[0] == 2):
                        print (student, "has taken easy course")
                else:
                        print (student, "has taken Medium Course")
```

find_Student_schedule_Level()



```
['CSCI 640', 'CSCI 610', 'ENGY 685', 'EENG 635'] has taken Medium Course
['ENGY 805', 'ENGY 720', 'CSCI 755', 'EENG 740'] has taken hard course
['CHEM 820', 'CHEM 725', 'CSCI 710', 'EENG 635'] has taken hard course
['ACCT 765', 'ACCT 665', 'CSCI 610', 'EENG 720'] has taken Medium Course
```

4. Write a lisp function ask-and-tell that processes simple sentences. It should allow you to enter lists that represent assertions, such as (Mary likes hiking) or (Steve hates pretzels), as well as questions such as (Does Mary like hiking) or (Does Steve like pretzels). The assertions should all have a proper name (e.g., Mary) in the first position, the word likes or hates in the second position, and either an activity (e.g., hiking) or and object (e.g., pretzels) in the last position. The questions should be similar, except that they begin with the word Does. The function should remember the assertions you give it, and answer your questions appropriately. If you repeat an assertion, it should let you know that; if you enter a contradictory assertion, it should alert you to that, and confirm that you're sure before making the change. E.g.,

> (ask-and-tell '(mary likes hiking))

"OK"

>(ask-and-tell '(steve hates pretzels))

"OK"

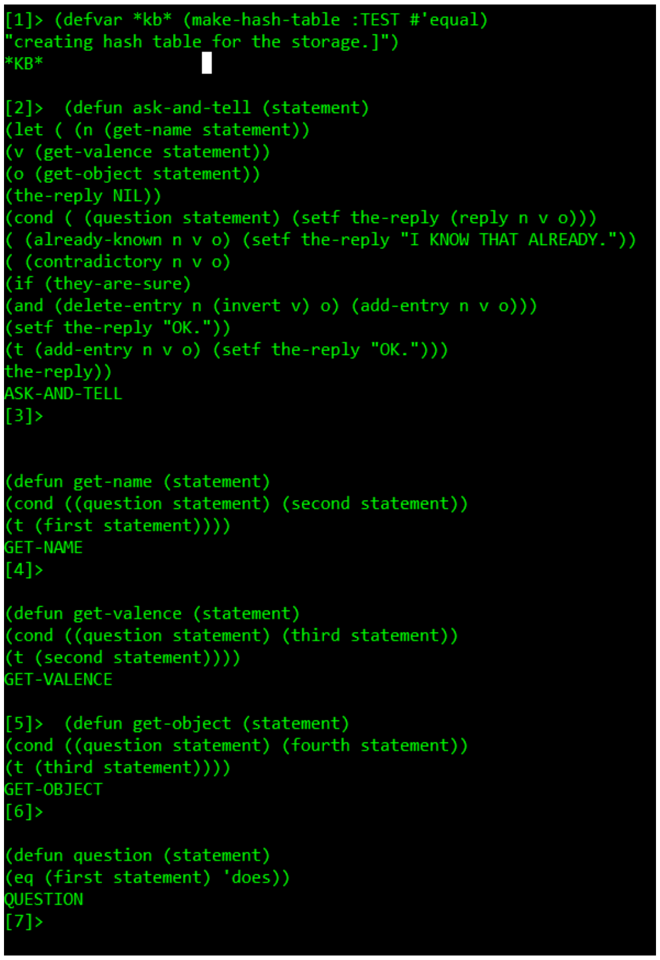
>(ask-and-tell '(does mary like hiking))

"YES"

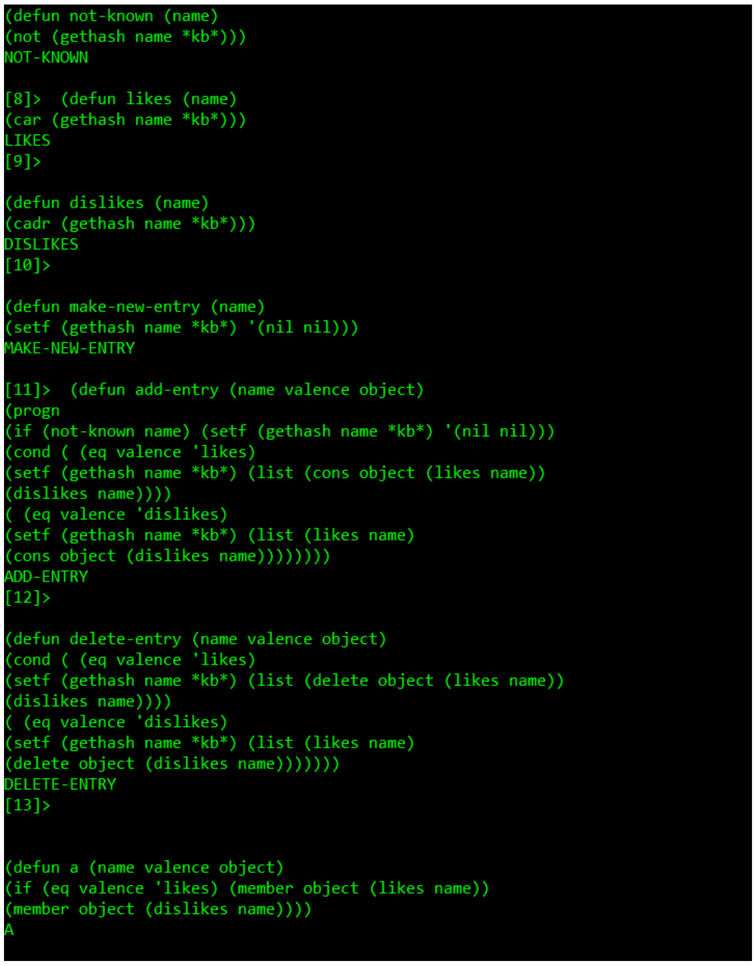
>(ask-and-tell '(does mary like pretzels))

"I DON'T KNOW"

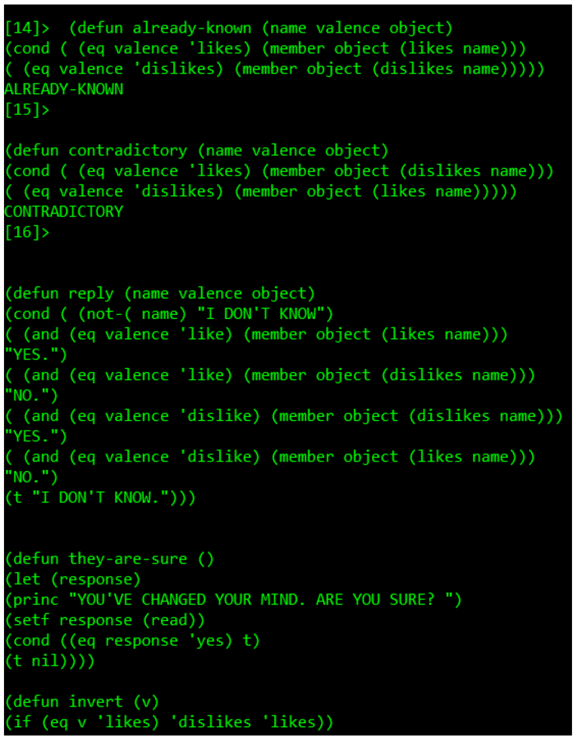
```
>(ask-and-tell ‘(mary likes hiking) )

“I KNOW THAT ALREADY”

>(ask-and-tell ‘(does steve like pretzels) )

“NO”

>(ask-and-tell ‘(mary hates hiking) )

“YOU’VE CHANGED YOUR MIND … ARE YOU SURE?”

no

“OK”

>(ask-and-tell ‘(does mary like hiking) )

“YES”

>(ask-and-tell ‘(mary hates hiking) )

“YOU’VE CHANGED YOUR MIND … ARE YOU SURE?”

yes

“OK”

>(ask-and-tell ‘(does mary like hiking) )

“NO”
```



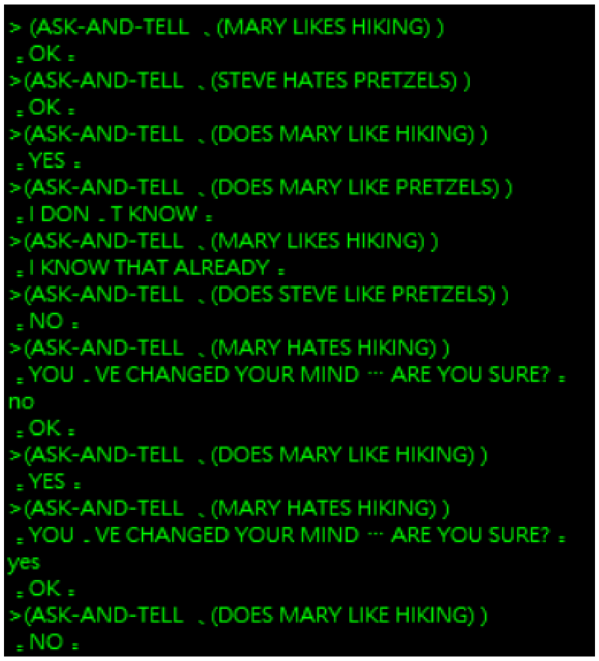
```
[1]> (defvar *kb* (make-hash-table :TEST #'equal)
"creating hash table for the storage.]")
*KB*

[2]>  (defun ask-and-tell (statement)
(let ( (n (get-name statement))
(v (get-valence statement))
(o (get-object statement))
(the-reply NIL))
(cond ( (question statement) (setf the-reply (reply n v o)))
( (already-known n v o) (setf the-reply "I KNOW THAT ALREADY."))
( (contradictory n v o)
(if (they-are-sure)
(and (delete-entry n (invert v) o) (add-entry n v o)))
(setf the-reply "OK."))
(t (add-entry n v o) (setf the-reply "OK.")))
the-reply))
ASK-AND-TELL
[3]>


(defun get-name (statement)
(cond ((question statement) (second statement))
(t (first statement))))
GET-NAME
[4]>

(defun get-valence (statement)
(cond ((question statement) (third statement))
(t (second statement))))
GET-VALENCE

[5]>  (defun get-object (statement)
(cond ((question statement) (fourth statement))
(t (third statement))))
GET-OBJECT
[6]>

(defun question (statement)
(eq (first statement) 'does))
QUESTION
[7]>
```



```
(defun not-known (name)
(not (gethash name *kb*)))
NOT-KNOWN

[8]>  (defun likes (name)
(car (gethash name *kb*)))
LIKES
[9]>

(defun dislikes (name)
(cadr (gethash name *kb*)))
DISLIKES
[10]>

(defun make-new-entry (name)
(setf (gethash name *kb*) '(nil nil)))
MAKE-NEW-ENTRY

[11]>  (defun add-entry (name valence object)
(progn
(if (not-known name) (setf (gethash name *kb*) '(nil nil)))
(cond ( (eq valence 'likes)
(setf (gethash name *kb*) (list (cons object (likes name))
(dislikes name))))
( (eq valence 'dislikes)
(setf (gethash name *kb*) (list (likes name)
(cons object (dislikes name))))))))
ADD-ENTRY
[12]>

(defun delete-entry (name valence object)
(cond ( (eq valence 'likes)
(setf (gethash name *kb*) (list (delete object (likes name))
(dislikes name))))
( (eq valence 'dislikes)
(setf (gethash name *kb*) (list (likes name)
(delete object (dislikes name)))))))
DELETE-ENTRY
[13]>


(defun a (name valence object)
(if (eq valence 'likes) (member object (likes name))
(member object (dislikes name))))
A
```



```
[14]>  (defun already-known (name valence object)
(cond ( (eq valence 'likes) (member object (likes name)))
( (eq valence 'dislikes) (member object (dislikes name)))))
ALREADY-KNOWN
[15]>

(defun contradictory (name valence object)
(cond ( (eq valence 'likes) (member object (dislikes name)))
( (eq valence 'dislikes) (member object (likes name)))))
CONTRADICTORY
[16]>


(defun reply (name valence object)
(cond ( (not-( name) "I DON'T KNOW")
( (and (eq valence 'like) (member object (likes name)))
"YES.")
( (and (eq valence 'like) (member object (dislikes name)))
"NO.")
( (and (eq valence 'dislike) (member object (dislikes name)))
"YES.")
( (and (eq valence 'dislike) (member object (likes name)))
"NO.")
(t "I DON'T KNOW.")))


(defun they-are-sure ()
(let (response)
(princ "YOU'VE CHANGED YOUR MIND. ARE YOU SURE? ")
(setf response (read))
(cond ((eq response 'yes) t)
(t nil))))

(defun invert (v)
(if (eq v 'likes) 'dislikes 'likes))
```



```
> (ASK-AND-TELL  ˛(MARY LIKES HIKING) )
 ₌OK ₌
>(ASK-AND-TELL  ˛(STEVE HATES PRETZELS) )
 ₌OK ₌
>(ASK-AND-TELL  ˛(DOES MARY LIKE HIKING) )
 ₌YES ₌
>(ASK-AND-TELL  ˛(DOES MARY LIKE PRETZELS) )
 ₌I DON ˍT KNOW ₌
>(ASK-AND-TELL  ˛(MARY LIKES HIKING) )
 ₌I KNOW THAT ALREADY ₌
>(ASK-AND-TELL  ˛(DOES STEVE LIKE PRETZELS) )
 ₌NO ₌
>(ASK-AND-TELL  ˛(MARY HATES HIKING) )
 ₌YOU ˍVE CHANGED YOUR MIND ⋯ ARE YOU SURE? ₌
no
 ₌OK ₌
>(ASK-AND-TELL  ˛(DOES MARY LIKE HIKING) )
 ₌YES ₌
>(ASK-AND-TELL  ˛(MARY HATES HIKING) )
 ₌YOU ˍVE CHANGED YOUR MIND ⋯ ARE YOU SURE? ₌
yes
 ₌OK ₌
>(ASK-AND-TELL  ˛(DOES MARY LIKE HIKING) )
 ₌NO ₌
```