CS150: Optional Final Exam
Turn In By Noon Monday May 04
via Email or physically to OLS 219

UVA ID: _____

Directions

Please be honorable. As long as everyone follows the honor code, take home exams benefit everyone. You are on your honor to follow the letter and spirit of these directions. You do not need to scribble a pledge on your exam to convince us you are honorable, but if you cheat please let us know so it does not unfairly impact other students. Please don't cheat.

Work alone. You may not discuss these problems or anything related to the material covered by this exam with anyone except for the course staff between receiving this exam and class Monday.

Open resources. You may use any books you want, lecture notes, slides, your notes, and problem sets. You may also use DrScheme or Python, but it is not necessary to do this. You may also use external non-human sources including books and web sites. **If you use anything other than the course books, slides, and notes, cite what you used.** You may not obtain any help from other humans other than the course staff.

Answer well. Answer all questions 1-9 (question 0 is writing your UVA ID on this page and the next, as well as *stapling or emailing the exam*, which hopefully everyone will receive full credit for), and optionally answer subsequent questions.

The questions are not necessarily in order of increasing difficulty, so if you get stuck on one question you should continue on to the next question. There is no time limit on this exam, but it should not take a well-prepared student more than a dozen hours to complete. It may take you longer, though, so please do not delay starting the exam. **There can be no extensions for this Final Exam. It is due at the time indicated above no matter what.**

Full credit depends on the clarity and elegance of your answer, not just correctness. Your answers should be as short and simple as possible, but not simpler. Your programs will be judged for correctness, clarity and elegance, but you will not lose points for trivial errors (such as missing a closing parenthesis).

The Final is optional, but if you turn it in, it counts toward your grade. **You should only turn it in if you think your score on the Final will be higher than your current class grade.**

NOTE: You will receive 30% of the credit for a (sub)question if you leave it totally blank (and thus do not waste our time). A wrong but present answer might receive as little as 0% of the credit. There is no need to guess.

UVA ID:

_____

Scores

0	1	2	3	4	5	6	7	8	9	EC	Tot
10	10	10	10	10	10	10	10	10	10	0	100

Formal Systems and Languages; Evaluation Rules

1a. Consider the following set of strings:

```
abnormal psychology
cognitive psychology
social psychology
psychology
cognitive psychology experiment
social psychology experiment
psychology experiment
behaviorism
```

Fill in the blanks below so that the grammar below produces exactly the strings above. Each English word is a single terminal. Fill in each blank with a single terminal or non-terminal.

```
S -> P
S -> ___
P -> ___ psychology G
P -> ___ psychology
E -> ___
E -> ___
E -> ___
G -> ___
G -> ___
T -> ___
C -> ___
```

1b. Consider the following Scheme definitions:

```
(define (wundt x y)
     (if   (<= x 0) y
           (wundt (- x 1) (+ y 1))))

(define (pavlov x y)
     (+ y x))
```

Either give inputs **x** and **y** such that **(wundt x y)** does not evaluate to the same value as **(pavlov x y)** or explain why no such inputs exist.

Recursive Definitions

2. Define a Scheme function **`magical`** that takes three arguments: a work function, an initial result value, and a list. The desired behavior of the function is specified by the examples below. Hint: three or four lines should suffice. You may use any procedure from class notes, the book, etc.

```
(magical work initial (list ))
     = initial
(magical work initial (list X))
     = (work initial X)
(magical work initial (list X Y))
     = (work (work initial X) Y)
(magical work initial (list X Y Z))
     = (work (work (work initial X) Y) Z)
;;; ... and so on, for longer lists

(magical + 0 (list 1 2 3 4)) = 10
(magical * 1 (list 2 3 4)) = 24
(magical (lambda (init elt) (+ 1 init)) 0 (list 7 8 9)) = 3
```

```
(define (magical work init lst)
```

Programming With Lists

3. Define a Scheme function **subseq** that takes one argument: a list of numbers. It should return true if there is a contiguous sublist of that list that sums to zero. It should return false otherwise. You may use any procedure from class notes, the book, etc.

```
(subseq (list 7 1 -3 2 8))
= #t  ;; 1 -3 2
(subseq (list 4 5 2 581))
= #f
(subseq (list 1 4 1 0 2))
= #t  ;; 0
(subseq (list 2 -3 4 -5 6 -7)
= #f
(subseq (list 2 -3 4 -6 7 -8 7)
= #t ;; -6 7 -8 7
(apply + (list 2 3 4))
= 9 ;; hint, but not necessary
(take (list 1 2 3 4 5) 2)
= (1 2) ;; hint, but not necessary
(drop (list 1 2 3 4 5) 2)
= (3 4 5) ;; hint, but not necessary
(subseq (list X Y Z)) =    ;; hint, but not necessary
     (or   (eq? X 0)
           (subseq (list Y Z))
           (subseq (list (+ X Y) Z)))
```

```
(define (subseq lst)
```

Programming With Mutation And Objects

4. Define a Scheme function **`make-countdown`**. The function takes a single argument: an initial positive number value. It returns an object that simulates a rocket countdown: it counts down from the initial value to zero, at which point the rocket is launched. Your definition of **`make-countdown`** *must* be a subclass of make-counter (shown below) using the definition of **`make-sub-object`** (Chapter 11, Page 11). You may override **`'next!`** only.

```
> (define (make-counter)
  (let ((count 0))
    (lambda (message)
     (cond ((eq? message 'get) (lambda (self) count))
           ((eq? message 'set!) (lambda (self x)
                                  (set! count x)))
           ((eq? message 'next!) (lambda (self)
                 (ask self 'set! (+ 1 (ask self 'get)))))
           (true (error "Unrecognized message"))
     ))))

> (define (ask object message . args)
     (apply (object message) object args))

> (define tminus (make-countdown 4))
> (ask tminus 'next!)
> (ask tminus 'next!)
> (ask tminus 'get)
2
> (ask tminus 'next!)
> (ask tminus 'next!)
"Rocket launch!"
```

```
(define (make-countdown start)
```

Metalinguistic Abstraction

5. Give a definition for a StaticCharme program **`square`** that that correctly computes the square of its input *when viewed as a Charme program*. The input to **`square`** will always be a single number. Your program must work correctly in all cases if its type information is removed and it is interpreted by Charme. Your program must *fail* to typecheck in StaticCharme.

```
StaticCharme> (define square XYZ)
Some type error!
Charme> (define square XYZ-without-type-information)
Charme> (square 3)
9
Charme> (square -4)
16
```

```
(define square : ((Number) -> Number)
  (lambda (x : Number)
```

Programming For The Internet

6. Consider the following Python code from **`reviews.py`** for Problem Set 8 (using the standard code from **`ps8.zip`** available on the course website). Change the code below in any way you like so that **`getAll`** instead returns exactly those reviews *X* such that there also exists a review *Y* for the same restaurant as *X,* and with the same number of stars as *X*, but written by a different reviewer. The **`review`** class and the review database schema (i.e., the names of the database fields or columns) are the same as in Problem Set 8. You may change the text in the box only (i.e., do not change the definition of the **`review`** class or how data is stored in the database).

```
def getAll ():

    try:

        db = connect()

        c = db.cursor()

        c.execute ("SELECT * FROM reviews")

        all = c.fetchall()

        db.close ()

        res = []

        for el in all:

            val = review.Review (el)

            res.append (val)

        return res

    except:

        print "Error getting reviews!"

        import sys

        print "Error: %s / %s" % (sys.exc_type, sys.exc_value)

        return None
```

Measuring Complexity

7. For each function specification below, give the best possible bound in Big Theta notation. For each answer, define any new variables you introduce to explain the running time. Base your answer on the best way to address the problem, not necessarily on the particular code that you wrote. For example, the best answer for "sorting" is Theta(n log n), even though it is possible to write sorting routines that take longer. Give a one-sentence justification for each answer.

7a. The "magical" function from Question #2.

7b. The "subseq" function from Question #3.

7c. The "square" function from Question #5.

7d. The "getAll" function from Question #6.

Computability

8a. Consider the **Watson** problem below. Argue whether or not it is computable.

Problem **Watson**
Input: A program specification P.
Output: True if P halts *before* 10 seconds have passed. False otherwise (i.e., if P loops forever or if P halts after 11 or more seconds).

8b. Consider the **Crick** problem below. Argue whether or not it is computable.

Problem **Crick**
Input: A program specification P.
Output: True if P halts *after* 10 seconds have passed. False otherwise (i.e., if P loops forever or if P halts before 10 seconds have passed).

Models of Computation

9. A language *L* is a set of strings. A Turing Machine *M* is said to recognize a language *L* if that Turing Machine halts and accepts if and only if it is started with a string *s* from *L* written on its tape. A context-free grammar *G* with start symbol X is said to produce a language *L* if for every string *s* in *L*, *X* can be rewritten to *s* using some sequence of steps (productions, rewrite rules) from *G*.

Argue for or against each proposition below.

(1) For every context-free grammar *G* that produces a language *L* there exists a Turing Machine *M* such that *M* recognizes that same *L*.

(2) For every Turing Machine *M* that recognizes a language *L* there exists a context-free grammar *G* that produces that same language *L*.

(3) Given a Turing Machine *M*, a context-free grammar *G*, and the language *L* produced by *G*, is it computable to determine if *M* recognizes *L*?

End of Exam