# [REAL, GRADED] Quiz 3- Requires Respondus LockDown Browser

**Due** 20 Nov at 13:55 **Points** 100 **Questions** 16
**Available** until 20 Nov at 13:55 **Time limit** 50 Minutes
Requires Respondus LockDown Browser

## Instructions

This exam is ONLY to be taken in Tech Auditorium during your scheduled exam time. Any other testing environment without explicit permission from the instructor will be considered a violation of our academic honesty policy.

- Please remember that you only get 1 attempt on this exam. You will have 50 minutes to complete it from the time that you start unless you have testing accommodations that allow for extra time.
- **Quiz Glossary (THIS LINK WORKS IN THE LOCKDOWN BROWSER) (https://bain-cs111.github.io/course-files/quizzes/q3_glossary_compact.pdf)**

In part 1, if you get the answer *exactly* right, canvas will mark it as CORRECT automatically. If there is any slight variation it will mark it as INCORRECT. After the exam is over, we will be going through each question and doing manual grading. For example, if you write (list of str) and Canvas expected (listof string), canvas will automatically mark your answer wrong. However, when we go in to manually grade, we would mark the above answer as fully correct.

In part 2 and part 3, we will be grading the answers completely by hand. What we are looking for is that you understand what was wrong with the code and know how to fix it. We can utilize line number(s) as a reference point, so that you can indicate where the error occurred, but more important will be your rewrite. We will grade these with the same leniency and partial credit that we would in a printed exam.

If you encounter any issues while taking the exam, please raise your immediately!

This quiz was locked 20 Nov at 13:55.

## Attempt history

| | Attempt | Time | Score |
|---|---|---|---|
| **LATEST** | Attempt 1 | 45 minutes | 95 out of 100 |

Score for this quiz: **95** out of 100
Submitted 20 Nov at 13:55
This attempt took 45 minutes.

## Question 1

**0 / 0 pts**

1) You may not communicate with anyone (electronically or in person) during this exam.  You must turn off your phone and other nearby devices after successfully logging into Canvas.

2) You must be under the direct supervision of the course staff in one of the approved testing rooms (either Tech Auditorium or the Testing Accommodations room) in order to complete this Quiz. Any other testing environments are a violation of our Academic Honesty policy and you will be reported to your Dean of Students for an official Academic Violation which may result in expulsion from Northwestern.

2) You may not use any resources during this exam. You may only use your computer in the Lockdown browser. You may not use any printed/written notes/books or any other devices/tools while taking this exam with TWO EXCEPTIONS:

- You may use a printed Quiz glossary (remember you can use the electronic version by clicking on the link provided in each set of instructions and opening it in a new tab).
- You may use scratch paper and a pen/pencil to take notes and sketch out solutions.

**********

The following statement is an excerpt from the Northwestern Provost's Office "Academic Integrity: A Basic Guide"

**A. Basic Standards of Academic Integrity**

Registration at Northwestern requires adherence to the University's standards of academic integrity. These standards may be intuitively understood, and cannot in any case be listed exhaustively; the following examples represent some basic types of behavior that are unacceptable:

1. **Cheating:** using unauthorized notes, study aids, or information on an examination; altering a graded work after it has been returned, then

submitting the work for regrading; allowing another person to do one's work and submitting that work under one's own name; submitting identical or similar papers for credit in more than one course without prior permission from the course instructors.

2. **Plagiarism:** submitting material that in part or whole is not entirely one's own work without attributing those same portions to their correct source.
3. **Fabrication:** falsifying or inventing any information, data or citation; presenting data that were not gathered in accordance with standard guidelines defining the appropriate methods for collecting or generating data and failing to include an accurate account of the method by which the data were gathered or collected.
4. **Obtaining an Unfair Advantage:** (a) stealing, reproducing, circulating or otherwise gaining access to examination materials prior to the time authorized by the instructor; (b) stealing, destroying, defacing or concealing library materials with the purpose of depriving others of their use; (c) unauthorized collaborating on an academic assignment (d) retaining, possessing, using or circulating previously given examination materials, where those materials clearly indicate that they are to be returned to the instructor at the conclusion of the examination; (e) intentionally obstructing or interfering with another student's academic work (f) recycling one’s own work done in previous classes without obtaining permission from one’s current instructor or (g) otherwise undertaking activity with the purpose of creating or obtaining an unfair academic advantage over other students' academic work.
5. **Aiding and Abetting Academic Dishonesty:** (a) providing material, information, or other assistance to another person with knowledge that such aid could be used in any of the violations stated above; (b) providing false information in connection with any inquiry regarding academic integrity; or (c) providing(including selling) class materials to websites that sell or otherwise share such materials – including homework, exams and exam solutions, submitted papers or projects, as well as original course materials (for example, note packets, power point decks, etc.). In addition to violating Northwestern’s policies on academic integrity, such conduct may also violate University policies related to copyright protection.
6. **Falsification of Records and Official Documents:** altering documents affecting academic records; forging signatures of authorization or falsifying information on an official academic

document, grade report, letter of permission, petition, drop/add form, ID card, or any other official University document.

7. **Unauthorized Access to computerized academic or administrative records or systems:** viewing or altering computer records, modifying computer programs or systems, releasing or dispensing information gained via unauthorized access, or interfering with the use or availability of computer systems or information.

If you agree to follow these rules, please type "I agree" in the following blank

Correct!

I agree

orrect Answers

"I agree

I agree"

I agree

"I agree"

Give the type of each of the following expressions. If more than one expression is given, assume that each one is executed, in order, and give the type of the value for the **last** expression. You may assume that all related libraries have been correctly loaded and that we are working in the **Advanced Student Language.**

- If it is a **primitive type** such as a number, string, Boolean or image (picture), just give the type.
- If it is a **record type (struct)**, just give the name of the record type. For example, if it's an album object, just say "album"
- If it is a **list**
  - If all the elements of the list are the same type, say "(listof *type*)" where *type* is the type of data in the list. For example (list 1 2 3) is a `(listof number)`.
  - If it is a with different types of data, say `(listof any)`
  - If you know the result is specifically the empty list, which has no elements and therefore no element type, just say empty list.
  - If you know the result is a **list** but you don't know the type of data in it, just say "list" and we will give you partial credit.

- If the result is a **function**, give its argument and return types. That is, write the type(s) of its argument(s) followed by an arrow and the type of its result. If the function accepts any type of value for an argument, just say “any”. For example:
  - The type of the **sin** function is: `number -> number`
  - The type of the **integer?** function is: `any -> boolean`
  - The type of the **<** function is: `number number -> boolean`
  - The type of the **square** function is: `number string color -> image`
- If you know the expression’s value is a function but don’t know its argument or return types, just say **function**, and we will give you partial credit.
- If it returns **no value,** say "(void)"
- If executing it would produce an **exception**, say “Exception." You don't need to specify the type of exception. This includes a program that will run out of memory due to infinite recursion.

**Quiz Glossary (THIS LINK WORKS IN THE LOCKDOWN BROWSER) (https://bain-cs111.github.io/course-files/quizzes/q3_glossary_compact.pdf)**

## Question 2

**0 / 5 pts**

```
(define lst (for-each (lambda (x)
                        (string-append x x))
                      (list "turkey " "time " "is " "here")))
lst
```

ou Answered

-> void

orrect Answers

void

(void)

## Question 3

5 / 5 pts

```
(define current-sound "silence")
(define (quack)
  (set! current-sound "quack"))
quack
```

ou Answered

-> void

orrect Answers

-> (void)

ok

## Question 4

5 / 5 pts

```
(define yip-yip
  (lambda (s)
    (local [(define appa 0)]
      (begin (for-each (λ (a)
                         (set! appa (+ appa (* 2 a))))
                       s)
             appa))))
yip-yip
```

Correct!

(listof number) -> number

orrect Answers

(listof number) -> number

## Question 5

5 / 5 pts

```
(cons false (list true true))
```

Correct!

(listof boolean)

orrect Answers

(listof bool)

(listof boolean)

(listof booleans)

(listof bools)

## Question 6

5 / 5 pts

```
(define y "hello")
(local [(define y 1)]
  (begin (set! y (+ y 1))
         y))
```

Correct!

number

orrect Answers

number

## Question 7

5 / 5 pts

```
(define (simply-walk into mordor)
  (begin (when (number? into)
           (set! mordor "impossible"))
         mordor))
(simply-walk 5 "just do it")
```

Correct!

exception

orrect Answers

exception

error

For the rest of Part 1, use the following definitions (note: there are both struct and variable definitions below). Note that the order in which any imperatives are executed will not affect the type of any of these expressions.

```
(define-struct event (title location date)
  #:methods
  ; add-to-wish-list!: event -> (void)
  ; adds an event to a wish-list
  ; effect: updates your wish-list
  (define (add-to-wish-list! e)
    (set! wish-list (cons e wish-list)))

  ; purchase-ticket!: event -> number
  ; these events are free so we don't deduct any money from our
  ;   account but do remove it from our wish-list. Returns your
  ;   bank account balance.
  ; effect: remove the event from your wish-list
  (define (purchase-ticket! e)
    (begin (set! wish-list (remove-all e wish-list))
           bank-account)))

(define-struct (concert event) (artist ticket-price)
  #:methods
  ; purchase-ticket!: event -> (void)
  ; as long as you have enough money in your bank-account,
  ; purchases a ticket and removes it from your wish-list,
  ; effect: might remove money from your bank account
  ;   and also the event from your wish list
  (define (purchase-ticket! e)
    (when (< (concert-ticket-price e) bank-account)
      (begin (set! bank-account (- bank-account
                                   (concert-ticket-price e)))
             (set! wish-list (remove-all e wish-list))))))

(define-struct (tswift-concert concert) (surprise-song))

(define saturday-event (make-event "Holiday Parade"
                                   "Michigan Ave."
                                   "11/18"))
(define weekend-plans (make-concert "Acidrap"
                                    "United Center"
                                    "8/19"
                                    "Chance the Rapper"
                                    50))
(define eras-chicago (make-tswift-concert "Eras Tour!"
                                          "Soldier Field"
                                          "10/18"
                                          "Taylor Swift"
                                          900
                                          "You All Over Me"))

(define bank-account 1000)
(define wish-list (list saturday-event weekend-plans))
```

## Question 8

5 / 5 pts

```
(map event-title wish-list)
```

Correct!

(listof string)

orrect Answers

listof strings

listof string

(listof strings)

(listof string)

## Question 9

5 / 5 pts

```
(add-to-wish-list! eras-chicago)
```

Correct!

void

orrect Answers

(void)

void

## Question 10

5 / 5 pts

```
set-tswift-concert-surprise-song!
```

ou Answered

tswift-concert any -> void (though intended 2nd input is probably string)

orrect Answers

tswift-concert string -> (void)

tswift-concert string -> void

ok

## Question 11

5 / 5 pts

```
(purchase-ticket! saturday-event)
```

Correct!

number

orrect Answers

number

Each of the following questions shows some code being executed (with a line number on the left of each line) at the Racket prompt, along with the output or error it generated, and the intended output that the programmer wanted. Give the **correction** to the code to produce the desired result. Remember, **you are not allowed to completely rewrite the given program.**

**It is highly recommended that you copy and paste the code and then correct it as your answer. If you correctly identify the error and fix it you will get full credit**. When copy and pasting, you can highlight just the code without the line numbers.

If you'd like your code to look more like Racket code, you can use the Canvas Toolbar to select the PREFORMATTED style. In the toolbar for the answer box, you'll see a button that says "Paragraph" with a down arrow (right next to the font size). Click on that then select Preformatted which will format your Text to look more like Racket code.

**If you are on a Chromebook that doesn't allow Ctrl-C then Ctrl-V,** you can highlight the code and then "drag it" to the box below in order to copy.

What we are looking for is that you understand what was wrong with the code and know how to fix it. If you are unsure of what the problem is, then you can provide an explanation of your thoughts. We will accept the line number(s) as a reference point, so that you can indicate where the error occurred, but more important will be your rewrite. We will grade these with the same leniency and partial credit that we would in a printed exam.

**Quiz Glossary (THIS LINK WORKS IN THE LOCKDOWN BROWSER) (https://bain-cs111.github.io/course-files/quizzes/q3_glossary_compact.pdf)**

For the first 2 questions in Part 2, continue to refer to the same struct definitions. **You can assume all values are defined as below and don't need to consider whether or not any imperatives from Part 1 Questions have been run.**

```
(define-struct event (title location date)
  #:methods
  ; add-to-wish-list!: event -> (void)
  ; adds an event to a wish-list
  ; effect: updates your wish-list
  (define (add-to-wish-list! e)
    (set! wish-list (cons e wish-list)))

  ; purchase-ticket!: event -> number
  ; these events are free so we don't deduct any money from our
  ;   account but do remove it from our wish-list. Returns your
  ;   bank account balance.
  ; effect: remove the event from your wish-list
  (define (purchase-ticket! e)
    (begin (set! wish-list (remove-all e wish-list))
           bank-account)))

(define-struct (concert event) (artist ticket-price)
  #:methods
  ; purchase-ticket!: event -> (void)
  ; as long as you have enough money in your bank-account,
  ; purchases a ticket and removes it from your wish-list,
  ; effect: might remove money from your bank account
  ;   and also the event from your wish list
  (define (purchase-ticket! e)
    (when (< (concert-ticket-price e) bank-account)
      (begin (set! bank-account (- bank-account
                                   (concert-ticket-price e)))
             (set! wish-list (remove-all e wish-list))))))

(define-struct (tswift-concert concert) (surprise-song))

(define saturday-event (make-event "Holiday Parade"
                                   "Michigan Ave."
                                   "11/18"))
(define weekend-plans (make-concert "Acidrap"
```

```
                                    "United Center"
                                    "8/19"
                                    "Chance the Rapper"
                                    50))
(define eras-chicago (make-tswift-concert "Eras Tour!"
                                          "Soldier Field"
                                          "10/18"
                                          "Taylor Swift"
                                          900
                                          "You All Over Me"))

(define bank-account 1000)
(define wish-list (list saturday-event weekend-plans))
```

## Question 12

**10 / 10 pts**

### Interaction

```
1  (define (date-list a-lst)
2    (if (empty? a-lst)
3        ""
4        (string-append (concert-date (first a-lst))
5                       (string-append ","
6                                      (date-list (rest a-lst))))))
7
8  (date-list (list eras-chicago weekend-plans saturday-event))
```

### Actual Output:

```
concert-date: this function is not defined
```

### Desired Output:

```
"10/18,8/19,11/18," ; the list of dates for all 3 things in the inputted lis
t
```

Your answer:

```
1  (define (date-list a-lst)
2    (if (empty? a-lst)
3        ""
4        (string-append (event-date (first a-lst))
5                       (string-append ","
6                                      (date-list (rest a-lst))))))
7
8  (date-list (list eras-chicago weekend-plans saturday-event))
```

date is a property of the parent-type event:

```
(define (date-list a-lst)
  (if (empty? a-lst)
      ""
      (string-append (event-date (first a-lst))
                     (string-append ","
                                    (date-list (rest a-lst))))))

(date-list (list eras-chicago weekend-plans saturday-event))
```

## Question 13

**10 / 10 pts**

### Interaction

```
1 (begin (set! (concert-ticket-price eras-chicago) 350)
2        (concert-ticket-price eras-chicago))
```

### Actual Output:

```
set!: expected a variable after set!, but found a part
```

### Desired Output:

```
350 ; the updated price of the eras-chicago ticket
```

Your answer:

```
1 (begin (set-concert-ticket-price! eras-chicago 350)
2        (concert-ticket-price eras-chicago))
```

If we want to mutate a property of a struct, we have to use its automatically created mutator!

```
(begin (set-concert-ticket-price! eras-chicago 350)
       (concert-ticket-price eras-chicago))
```

## Question 14

10 / 10 pts

### Interaction

```
(define result empty)
(define q (list 1 2 3 4 5 6 7 8 9))
(for-each (lambda (x)
            (set! result (append result (list (* 10 x)))))
          q)
```

### Actual Output:

```
; nothing is returned
```

### Desired Output:

```
(list 10 20 30 40 50 60 70 80 90); the list of numbers, each multiplied by 1
0
```

Your answer:

```
(define result empty)
(define q (list 1 2 3 4 5 6 7 8 9))
(begin (for-each (lambda (x)
                   (set! result (append resu
lt (list (* 10 x)))))
                 q)
       result)
```

For-each doesn't return anything, so if we want to return the resulting list, we need to sequence for-each with a return statement:

```
(define result empty)
(define q (list 1 2 3 4 5 6 7 8 9))
(
(begin (for-each (lambda (x)
                   (set! result (append result (list (* 10 x)))))
                 q)
       result)
```

## Question 15

**10 / 10 pts**

### Interaction

```
; my-reverse: (listof any) -> (listof any)
; returns the reversed list
(define (my-reverse lst)
  (local [(define rem lst)
          (define acc empty)
          (define (help)
            (cond [(empty? rem) acc]
                  [else         (begin (set! acc (cons (first rem)
                                                       acc))
                                       (rest rem)
                                       (help))]))]
    (help)))
(my-reverse (list "a" "b" "c"))
```

### Actual Output:

```
The program ran out of memory. (Ran forever...)
```

### Desired Output:

```
(list "c" "b" "a") ; the list reversed
```

Your answer:

```
; my-reverse: (listof any) -> (listof any)
; returns the reversed list
(define (my-reverse lst)
  (local [(define rem lst)
          (define acc empty)
          (define (help)
            (cond [(empty? rem) acc]
                  [else         (begin (set! acc (cons (first rem)
                                                       acc))
                                       (set! rem (rest rem))
                                       (help))]))]
    (help)))
(my-reverse (list "a" "b" "c"))
```

Throwback! We forgot to actually remove the first element of the list before we recursed!

```
(define (my-reverse lst)
  (local [(define rem lst)
          (define acc empty)
          (define (help)
            (cond [(empty? rem) acc]
                  [else         (begin (set! acc
                                             (cons (first rem) acc))
                                       (set! rem (rest rem))
                                       (help))]))]
    (help)))
(my-reverse (list "a" "b" "c"))
```

The following questions show a function definition. In the blank below you **may be asked to provide multiple tests** (`check-expect`) of the function. That is you, need to come up with and write valid tests that assesses whether or not the function works as the programmer intends **and that cover the specific cases referenced in the question**.

If you'd like your code to look more like Racket code, you can use the Canvas Toolbar to select the PREFORMATTED style. In the toolbar for the answer box, you'll see a button that says "Paragraph" with a down arrow (right next to the font size). Click on that then select Preformatted which will format your Text to look more like Racket code.

**Quiz Glossary (THIS LINK WORKS IN THE LOCKDOWN BROWSER) (https://bain-cs111.github.io/course-files/quizzes/q3_glossary_compact.pdf)**

## Question 16

**10 / 10 pts**

Consider the following atm code:

```
; deposit!: number -> number
; Deposits money into our bank account
; Effect: balance increases by deposit amount
(define deposit!
  (λ (amount)
    (begin (set! balance
                 (+ balance amount))
           balance)))

; withdraw!: number -> number
; Withdraws money from our account
; Effect: balance decreases by deposit amount unless
; balance is less than withdrawal amount
(define withdraw!
  (λ (amount)
    (begin (if (> amount balance)
               (error "Not enough money!")
               (set! balance (- balance amount)))
           balance)))
```

First write a definition of some global variable **balance** that starts at 100 and then write a **series of at least 3 check expects** that check to see the following transactions are processed correctly according to the code above (in this exact order):

- A withdrawal of 95
- A deposit of 25
- A withdrawal of 300

Hint: **check-error** allows you to check if you ran into a specific exception/error when running a function.

Your answer:

```
(define balance 100)

(check-expect (withdraw! 95)
              5)

(check-expect (deposit! 25)
              30)

(check-error (withdraw! 300)
             "Not enough money!")
```

```
(define balance 100)
(check-expect (withdraw! 95)
              5)
(check-expect (deposit! 25)
              30)
(check-error (withdraw! 300)
             "Not enough money!")
```

Quiz score: **95** out of 100