Clearly print your name as it appears on Canvas:

Clearly print your netid (ex. abc123):

COMP_SCI 111 Exam 2
DO NOT OPEN THIS TEST UNTIL INSTRUCTED TO

Part 1

Give the type of the value returned by 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 the image procedures, including **iterated-overlay** and **iterated-beside**, have been defined, and all the relevant cs111 libraries have been loaded and that we are working in "Advanced Student" language.

- If it is a **primitive type** such as a number, string, Boolean or image (picture), just give the name of the type. So if the result is a number, just say "number."
- If it is a **record/struct type**, 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 list 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 **procedure**, give its type signature, i.e. its argument and return types. In particular, write the type(s) of its argument(s) followed by an arrow and the type of its result. If the procedure accepts any type of value for an argument, just say "any".
  - If you know the expression's value is a **procedure**, but don't know its argument or return types, just say "procedure", and we will give you partial credit.
  - Examples:
    - The type of the **sin** procedure is
      $number \rightarrow number$
    - The type of the **integer?** procedure is:
      $any \rightarrow Boolean$
    - The type of the **<** procedure is:
      $number\ number \rightarrow Boolean$
    - The type of the **square** procedure is
      $number\ string\ color \rightarrow image$
- If it returns **no value**, say "void".
- If executing the expression would produce an **exception**, say "Exception". You do not have to explain what type of exception or why

_____ Clearly print your netid (ex. abc123):

(page total **FOR GRADING ONLY**)

1.
```
(apply max
       (map abs (list -5 10 365 -20)))
```

2.
```
(define x 10)
(local [(define x "Northwestern University")
        (define (help x)
           x)]
   (help x))
```

3.
```
(define val true)
(unless val 5)
```

4.
```
(define z (list "winter" "is" "great"))
(if (= (length z) 3)
    (cons "summer" (rest z)))
```

5.
```
(define r (list "a" "b" "c"))
(define res empty)
(begin (for-each (lambda (s)
                   (set! res
                         (cons (string-append s s)
                               res)))
                 r)
       res)
```

Clearly print your netid (ex. abc123): _____


```
; Refer to the following code for questions 6, 7, 8, 9 and 10...
; a furniture is a...
; (make-furniture string number string)
(define-struct furniture [name price owner]
  #:methods
  (define (sell! f new-owner)
    (set-furniture-owner! f new-owner)))

; a table is a...
; (make-table string number string string)
(define-struct (table furniture) [surface-material])

; a chair is a...
; (make-chair string number string string)
(define-struct (chair furniture) [color])

(define my-table (make-table "kitchen table"
                             400
                             "sara"
                             "wood"))
(define my-chair (make-chair "desk chair"
                             200
                             "sara"
                             "purple"))
```

6. `(apply + (map furniture-price (list my-table my-chair)))`

7. `sell!`

8. `(map (sell! f "ian") (list my-table my-chair))`

9. `(chair-color my-chair)`

10.
```
(begin (set-table-price! my-table 500)
       (table-price my-table))
```

_____ Clearly print your netid (ex. abc123):

(page total **FOR GRADING ONLY**)

Part 2

Each of the following questions shows some code being executed 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.

- Fix the code that's there; **don't rewrite it from scratch**. In grading, we're looking for evidence that you understand the bug in that particular code, not that you understand how to write new code.
- You do not need to provide an explanation, although you are free to do so if you like.
- It is sufficient to **write your correction on top of the existing code**; you **don't need to recopy** it.

```
; Refer to the following code for questions 1 and 2....
; a person is
; (make-person number string)
(define-struct person [ssn name])

; A person-node is either
; - empty
; - (make-person-node person person-node person-node)
(define-struct person-node [person left right])

(define sara (make-person 7 "sara"))
(define ian (make-person 4 "ian"))
(define sara-node (make-person-node sara empty empty))
(define ian-node (make-person-node ian empty sara-node))
```

Question 1 Score:

Interaction	Desired-output

`(define (list-contents pn)`
 	
`  (cond [(empty? pn)  empty]`
 	
`  [else (append (list-contents (person-node-left pn))`
 	
`  (person-ssn (person-node-person pn))`
 	
`  (list-contents (person-node-right pn)))]))`
 	
`(list-contents ian-node)`
	`> (list 4 7)`
	
	Actual output (exception):
	`append: expects a list, given 7`

Clearly print your netid (ex. abc123):

_____
(page total **FOR GRADING ONLY**)

Question 2 Score:

Interaction	Desired-output

`(define (find-max pn)`
 	
`  (cond [(empty? pn) 0]`
 	
`  [else (max (find-max (person-node-left pn))`
 	
`  (find-max (person-node-right pn)))]))`
 	
`(find-max ian-node)`
	`> 7`
	
	Actual output: `0`

Question 3 Score:

Interaction	Desired-output

`(define s 0)`
 	
`(map (lambda (x)`
 	
`  (set! s (+ s x)))`
 	
`  (list 1 2 3 4))`
	`> 10`
	
	Actual output:
	`(list (void) (void) (void) (void))`

_____
(page total **FOR GRADING ONLY**)

Clearly print your netid (ex. abc123):

Question 4 Score:

Interaction	Desired-output
(define (sumlist lst) (local [(define sum 0) (define remaining lst) (define (sum-help) (cond [(empty? remaining) sum] [else (begin (set! remaining (rest remaining)) (set! sum (+ sum (first remaining))) (sum-help))]))] (sum-help))) (sumlist (list 1 2 3))	> 6 Actual output (exception): first: expects a non-empty list; given: '()

Question 5 Score:

Interaction	Desired-output
(define odds empty) (define lst2 (list 1 2 3 4 5 6 7 8 9)) (begin (for-each (lambda (x) (when (odd? x) (set! odds (cons x odds)))) lst2) odds)	> (list 1 3 5 7 9) Actual output: (list 9 7 5 3 1)