Practice Exam 1

Please read and study the "Notes on Exam 1" document prior to spending time with this practice exam, being sure not to violate the one inviolable rule pertaining to this practice exam.

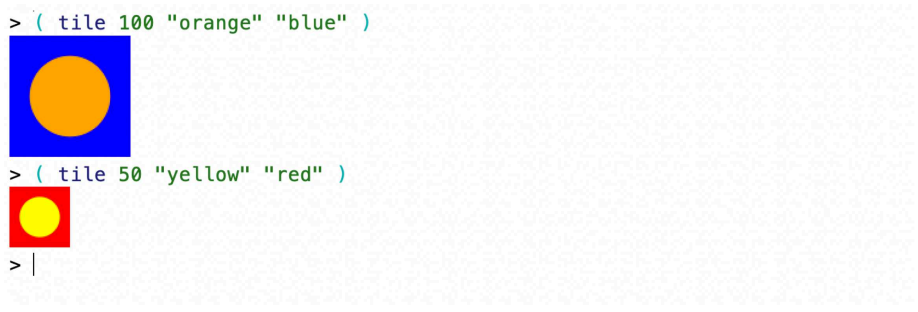
Please note that this exam is intended for practice. That is, it is intended to help you to strengthen your knowledge of the subject matter, while providing you with a feel for what the actual exam will be like. You might think of this exam as an opportunity to train, by analogy with the training that a distance runner might do. The first time that you take this practice exam, you will probably find it hard to comfortably complete in one hour. Each successive time that you take this practice exam, you will likely come closer to finishing it in one hour, provided you have worked on understanding the questions and their answers in the intervening time. Eventually, once you no longer have to actually read the questions, you might find yourself completing it in less than one hour. At that point, your relationship to the knowledge of the subject matter of the course will, most likely, have changed considerably. You will have learned something. You will have consolidated and refined your knowledge. That is the goal of this practice exam.

The real exam will be structured just like this practice exam. The questions will, perhaps with an exception or two, be different, but not wildly different.

Question 1 - Simple function definition requiring 2htdp/image

Task, in brief: Define a Racket function that precisely mimics the function which is illustrated in the following demo.

Demo:



```
> ( tile 100 "orange" "blue" )
> ( tile 50 "yellow" "red" )
> |
```

Details:

1. Name of the function: `tile`
2. Three parameters:
   (a) The first parameter is a positive integer representing the side length of the tile.
   (b) The second parameter is the string representation of a color used to paint the decorative dot located within the tile.
   (c) The third parameter is the string representation of a color used to paint the background of the tile.
3. Value of the function: a square of side length equal to the first parameter, with interior dot of radius equal to one third of the side of the tile, with the dot colored according to the second parameter, and with the background colored with the third parameter.

Constraints:

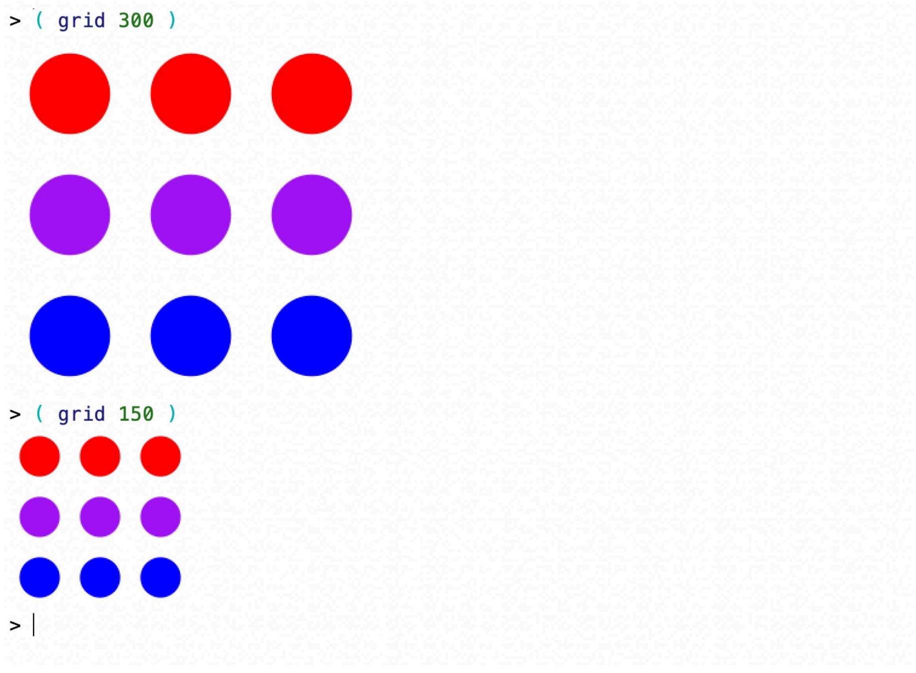
1. Use the `square` and `circle` functionality from the `2htdp/image` library to help you to render the image.
2. Solve the problem of creating the image by placing the dot on top of the square.
3. Do not call any helper functions of your own creation from within the function that you are asked to define.

Answer Area for Question 1

Question 2 - Function definition requiring 2htdp/image and a bit of design

Task, in brief: Define a Racket function that precisely mimics the function which is illustrated in the following demo.

Demo:



```
> ( grid 300 )
> ( grid 150 )
>
```

Details:

1. Name of the function: `grid`
2. Parameter: A positive integer representing the side length of the grid when conceived as a 3x3 grid of square tiles.
3. Value of the function: An image that represents the 3x3 grid of appropriately spaced dots, with three red dots in the top row, three purple dots in the middle row, and three blue dots in the bottom row.

Constraints:

1. Feature the `tile` function from the first question on this exam within the function that you are now being asked to define.
2. Do not call any helper functions of your own creation from within the function that you are asked to define, other than the `tile` function that you are obliged to call.

Answer Area for Question 2

Question 3 - BNF Basics

Consider the following BNF description of a lambda function:

```
<lambda-function> ::= ( lambda <lambda-list> <body> )
<lambda-list> ::= ( <id-seq> )
<body> ::= <form-seq>
<id-seq> ::= <empty> | <id> <id-seq>
<form-seq> ::= <empty> | <form> <form-seq>
```

1. Short questions about the BNF description:
   (a) I left three **nonterminal** symbols unrefined in the BNF description of a lambda function that I presented. What were these three nonterminals?
   (b) Altogether, including the three nonterminals that I left unrefined in specifying the grammar, how many **nonterminal** symbols are present in the BNF grammar?
   (c) How many **tokens** (**terminal** symbols) are present in the BNF grammar?
   (d) What is the **start symbol** for this BNF grammar?
   (e) How many **productions** appear in this BNF grammar?
   (f) Why didn't I refine the <`id`> nonterminal?
      i. Because the definition of identifier varies some from Lisp to Lisp.
      ii. Because identifiers are a lexical level construct, and it is generally best to separate the lexical grammar from the phrase structure grammar.
      iii. Because it would have distracted from what I was trying to feature in this BNF, that being the nature of a lambda function.
      iv. All of the above.
   (g) Why didn't I refine the <`form`> nonterminal?
      i. Because forms are insignificant in Lisp.
      ii. Because it would have distracted from what I was trying to feature in this BNF, that being the nature of a lambda function.
      iii. Both of the above.

2. Draw the parse tree for the following lambda function, assuming that `a` and `b` are identifiers, and that `( + a b )` is a form:

   `( lambda ( a b ) ( + a b ) )`

   Note: (1) When it is time to expand an <`id`> node in the parse tree, you can do so with the appropriate identifier, since the <`id`> nonterminal was not refined in the grammar. (2) When it is time to expand the <`form`> node in the parse tree, you will simply write down `( + a b )` as the child node, since the <`form`> nonterminal was not refined in the grammar.

Answer Area for Question 3

1. Short questions about the BNF description:

   (a)

   (b)

   (c)

   (d)

   (e)

   (f)

   (g)

2. The parse tree:

Question 4 - BNF Grammar Writing

Write a BNF grammar, by listing just the productions, for the language consisting of two visual rendering commands, `draw` and `paint`, which render four sorts of shapes, `circle` and `square` and `triangle` and `rectangle`, that come in three sizes, `large` and `medium` and `small`, and in six colors, `red` and `yellow` and `blue` and `brown` and `green` and `purple`. Furthermore, assume that the commands may be of four varieties:

1. Variety 1, the "shape" variety, in which a command is followed by the indefinite article which is followed by a shape. For example:

   ```
   draw a square
   paint a triangle
   paint a rectangle
   ```

2. Variety 2, the "size-shape" variety, in which a command is followed by the indefinite article which is followed by a size which is followed by a shape. For example:

   ```
   draw a small circle
   draw a medium square
   paint a large triangle
   ```

3. Variety 3, the "color-shape" variety, in which a command is followed by the indefinite article which is followed by a color which is followed by a shape. For example:

   ```
   draw a red triangle
   draw a brown square
   paint a purple rectangle
   ```

4. Variety 4, the "size-color-shape" variety, in which a command is followed by the indefinite article which is followed by a size which is followed by a color which is followed by a shape. For example:

   ```
   draw a small blue circle
   paint a large purple triangle
   paint a medium yellow square
   ```

Answer Area for Question 4

Question 5 - Reading a Recursive Program

Please study the accompanying `mystery` program. Your task is to describe, in English, what the program does for given parameters, the first assumed to be a non-negative integer, and the second assumed to be a valid color within the `2htdp/image` library. For example, the following would be a valid function call: `( mystery 4 "blue" )`, as would `( mystery 15 "purple" )`.

The Program

```
#lang racket

( require 2htdp/image )

( define ( mystery n color )
  ( mystery-helper 1 ( odd-number n ) color )
)

( define ( mystery-helper k n c )
  ( define color ( select-color k c ) )
  ( define disk ( circle ( * k 10 ) "solid" color ) )
  ( cond
    ( ( < k n )
      ( overlay disk ( mystery-helper ( + k 1 ) n c ) )
    )
    ( ( = k n )
      disk
    )
  )
)

( define ( select-color n c )
  ( cond
    ( ( odd? n ) c )
    ( else "white" )
  )
)

( define ( odd-number n )
  ( cond
    ( ( = n 1 ) 1 )
    ( else ( + 2 ( odd-number ( - n 1 ) ) ) )
  )
)
```

Answer Area for Question 5

Question 6 - Writing a Recursive Program

Task, in brief: Define a **recursive** function in Racket (no iteration allowed) that precisely mimics the function which is illustrated in the following demo.

```
> ( alternation "a" "b" 3 )
"aba"
> ( alternation "b" "a" 3 )
"bab"
> ( alternation "0" "1" 6 )
"010101"
> ( alternation "1" "0" 6 )
"101010"
> ( alternation "|" "-" 9 )
"|-|-|-|-|"
> ( alternation "<" ">" 12 )
"<><><><><><>"
> ( alternation "x" "y" 0 )
""
```

Details:

1. Name of the function: `alternation`
2. Three parameters:
   (a) The first parameter is string of length 1
   (b) The second parameter is string of length 1
   (c) The third parameter is a nonnegative integer
3. Value of the function: A string whose length equals the third parameter, which alternates between the string represented by the first parameter and the string represented by the second parameter, beginning (in the case of a nonempty result) with the string represented by the first parameter.

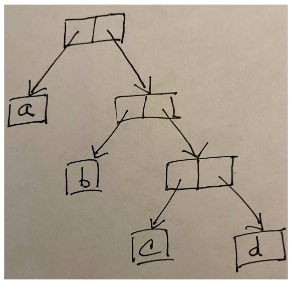
Constraints:

1. Use the the primitive function `string-append` for gluing strings together.
2. Do not call any helper functions of your own creation from within the function that you are asked to define.
3. And don't forget that this function must be recursive.

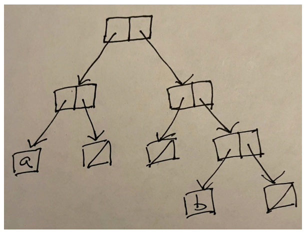
Answer Area for Question 6

Question 7 - Lisp: S-expressions

1. True/False: The following S-expressions are equivalent:
   (a) (a . b)
   (b) (a b)
2. True/False: The following S-expressions are equivalent:
   (a) (a . (b c))
   (b) (a b c)
3. True/False: The following S-expressions are equivalent:
   (a) (a b ())
   (b) (a . (b . ()))
4. True/False: The following S-expressions are equivalent:
   (a) (a . (() . (b . ())))
   (b) (a () (b))
5. Short answer: Rewrite `( ( ( blue ) ( red ) ) . ( green . ( purple ) ) )` as the equivalent dot free S-expression (i.e. the S-expression which contains no dots but which is equal to the given S-expression).
6. Short answer: Write down the **pure dotted-pair textual** representation of the S-expression which has the following internal representation:



7. Short answer: Write down a **pure list textual** representation of the S-expression which has the following internal representation:



Answer Area for Question 7

1.

2.

3.

4.

5.

6.

7.

Question 8 - Lisp: Historical Lisp

The following redacted Racket Interactions session incorporates forms that feature 6 of the 10 functions of Historical Lisp. Respond as Racket would to each of the forms by filling in the "blanks" in the answer area below of the redacted session.

Answer Area for Question 8

```
Welcome to DrRacket, version 8.1 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( car '( 1 2 3 ) )



> ( cdr '( 1 2 3 ) )



> ( cons '( 1 2 3 ) '( 1 2 3) )



> ( quote quote )



> ( ( lambda ( x ) ( cdr x ) ) '( x y z ) )



> ( ( lambda ( x y ) ( cons y ( cons x '() ) ) ) 'two 'one )



> ( eval ( cons ( car ( cdr '( + - * / ) ) ) '( 15 10 ) ) )
```

Question 9 - Lisp: Basic List Processing Interactions

The following redacted Racket Interactions session incorporates some basic list processing forms. Respond as Racket would to each of the forms by filling in the "blanks", in the answer area below, of the redacted session.

Answer Area for Question 9

```
> ( define a '( 2 0 2 1 ) )
> ( define b '( ( 2 0 ) ( 2 1 ) ) )
> ( cons a b )



> ( list a b )



> ( append a b )



> ( list-ref ( append a ( cdr a ) ) 6 )



> ( and ( = ( length a ) 4 ) ( = ( length b ) 2 ) )



> ( or ( < ( car a ) 1 ) ( > ( car ( cdr a ) ) 1 ) )



> ( not ( equal? ( cdr ( cdr a ) ) ( car ( cdr b ) ) ) )
```

Question 10 - Lisp: Some Basic Lisp Processing Definitions

Question Area for Question 10

1. Write a parameterless Racket function called `noun` that randomly returns one of the following nouns: `kitty` or `robot` or `monster` or `professor` or `jabberwock` or `king`. *Please look at the demo for clarification.*
2. Write a parameterless Racket function called `adjective` that randomly returns one of the following adjectives: `ancient` or `insane` or `blue-eyed` or `silvery` or `methodical` or `unlucky` or `enormous`. *Please look at the demo for clarification.*
3. Write a Racket function called `adjective-list` that takes one numeric paramater that is guaranteed to be either `0` or `1` or `2`, and which randomly returns a list of adjectives of length given by the parameter. Constraints: (1) This function must reference the `adjective` function to get any needed adjectives. (2) It is required that you use the `cond` form in this function definition. *Please look at the demo for clarification.*
4. Write a parameterless Racket function called `noun-phrase` that returns a random noun phrase of length 2 or 3 or 4, represented as a list of atoms. Note that the first word of the noun phrase will always be the definite article. Constraint: This function must reference the `noun` function to get the noun, and the `adjective-list` function in conjunction with the `random` primitive to get an appropriate adjective list. *Please look at the demo for clarification.*
5. Write a parameterless Racket function called `bank-of-noun-phrases` that returns a list of exactly 6 noun-phrases, each represented as a list of atoms. *Please look at the demo for clarification.*

Demo Area for Question 10

```
Welcome to DrRacket, version 8.1 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( noun )
'robot
> ( noun )
'jabberwock
> ( noun )
'monster

> ( adjective )
'silvery
> ( adjective )
'insane
> ( adjective )
'silvery

> ( adjective-list 2 )
'(unlucky ragged)
> ( adjective-list 2 )
'(ragged unlucky)
> ( adjective-list 1 )
'(unlucky)
```

```
> ( adjective-list 1 )
'(methodical)
> ( adjective-list 0 )
'()

> ( noun-phrase )
'(the kitty)
> ( noun-phrase )
'(the insane insane robot)
> ( noun-phrase )
'(the blue-eyed robot)
> ( noun-phrase )
'(the kitty)
> ( noun-phrase )
'(the jabberwock)
> ( noun-phrase )
'(the unlucky silvery king)

> ( bank-of-noun-phrases )
'((the insane ragged king)
  (the ancient monster)
  (the blue-eyed jabberwock)
  (the insane silvery robot)
  (the methodical ancient professor)
  (the unlucky monster))
> ( bank-of-noun-phrases )
'((the unlucky ragged king)
  (the ancient king)
  (the methodical silvery professor)
  (the monster)
  (the jabberwock)
  (the ragged jabberwock))
>
```

Answer Area for Question 10

1.

2.

3.

4.

5.

Blank Page

This page was intentionally left blank. Rip it off and use it for “scratch” paper, if you like.

Blank Page

This page was intentionally left blank. Rip it off and use it for “scratch” paper, if you like.