1. (8 points) Write a Racket function `multicons` that takes an item `x`, a non-negative integer `n`, and a list `xs`; and returns a new list with `n` occurrences of `x` followed by `xs`. You don’t need to handle bad inputs. Examples:

```
(multicons 'z 3 '(a b c)) => (z z z a b c)
(multicons '(a b) 3 '(x y z)) => ((a b) (a b) (a b) x y z)
(multicons 'z 0 '(a b c)) => (a b c)
```
2. (10 points) Suppose we have the following class in Ruby:

```ruby
class Toy
  def initialize(size, type)
    @size = size
    @type = type
  end

  # The goodness method returns a number indicating how good
  # the toy is. The default is that bigger is better!
  def goodness
    size
  end
end
```

(a) Add code to `Toy` to define public getters for `size` and `type` (but not setters), and to mix in the `Comparable` module. Write your extra code in the blank spaces in the `Toy` definition above. To compare two toys, compare their goodness. You should define in `Toy` the method needed by `Comparable`; this should then automatically let you compare two toys `t1` and `t2` using `t1 > t2`, `t1 < t2`, `t1 == t2`, and so on, without needing to define `<`, `>`, and so on in `Toy`. Hint: `3 <= 10` evaluates to `-1`.

(b) Write a subclass of `Toy` called `StuffedAnimal`.

- `StuffedAnimal` has the same fields as `Toy` plus `numHugsGiven`.
- `numHugsGiven` should be 0 on initialization.
- The new `initialize` method for `StuffedAnimal` should still take 2 arguments: `size` and `type`. For full credit, when possible reuse relevant methods inherited from `Toy`.
- Redefine `goodness` for `StuffedAnimal` to be the product of its size and the number of hugs given.
- Add a public method `hug` that increments `numHugsGiven` by 1 when called.
- Add a public getter for `numHugsGiven`.
3. (10 points) Write a Haskell function `indices` that takes a item and a list of that same type of item, and returns a list of the positions of that item in the list. You can use a helper function if needed. Also give the most general type of the `indices` function. Examples:

```haskell
indices 'b' "ababb" => [1,3,4]
indices true [false,false,true] => [2]
indices 'x' "abc" => []
```
4. (6 points) What is the output from the following Ruby program? Write the output on the numbered lines. Hint: `puts` for a hash prints like this: {"x"=>100}.

```ruby
def test1(a,b)
    a["x"] = "squid"
    b["x"] = "clam"
end

def test2(a,b)
    a["x"] = "tuna"
    b = {"x"=>"starfish"}
end

a = Hash.new
test1(a,a)
puts a

b = Hash.new
c = Hash.new
putst2(b,c)
puts b

b = Hash.new
c = Hash.new
putst2(b,c)
puts b
```

1. ______________________________
2. ______________________________
3. ______________________________
4. ______________________________
5. ______________________________
6. ______________________________

5. (6 points) Suppose that Ruby passed parameters by reference. What would the output be in that case for the program in Question 4?

1. ______________________________
2. ______________________________
3. ______________________________
4. ______________________________
5. ______________________________
6. ______________________________
6. (10 points) Write a Prolog rule \texttt{index\_of(X,Xs,N)} that finds the element at a given position in a list. You can assume that \(N\) is an integer in the goal. However, either \(X\) or \(Xs\) or both could be variables. Use \texttt{is} for arithmetic. Examples:

\begin{itemize}
\item \texttt{index\_of(X, [a, b, c, d], 2)} should succeed with \(X=c\)
\item \texttt{index\_of(X, [a, b, c, d], 10)} should fail
\item \texttt{index\_of(X, [], 0)} should fail
\end{itemize}

7. (6 points) Using your rule from Question 6, what are all the answers returned for the following goals? If there are infinitely many, give the first three. Write \texttt{false} if the derivation fails. If your answer involves variables generated by Prolog, make up names like this: \texttt{\_42} (the exact number you use in the name doesn't matter).

\begin{itemize}
\item (a) \texttt{index\_of(b, [a, b, c, d], 3)}
\item (b) \texttt{index\_of(a, Xs, 0)}
\item (c) \texttt{index\_of(a, Xs, 2)}
\end{itemize}
8. (10 points) Rewrite your Prolog rule from Question 6 to use constraints on integers, using the clpfd library. Hint: to remind you of the syntax for constraints in clpfd, here are examples of constraints on K: K#>5, K#=J+4, K#>=0.

9. (6 points) Using your improved rule from Question 8, what are all the answers returned for the following goals? If there are infinitely many, give the first three. Write false if the derivation fails. If your answer involves variables generated by Prolog, make up names like this: _42 (the exact number you use in the name doesn’t matter).

(a) index_of(b, [a,b,c,d,a,b,c,d], N)

(b) index_of(X, [a,b,c], N)

(c) index_of(a,Xs,N)
10. (10 points) Here are some groups of statements about Java types. Circle all statements that are correct as far as the Java compiler is concerned. In addition, write an E on the line next to each statement if that statement is correct as far as the Java compiler is concerned, but that could result in a runtime exception due to a type error.

For example, suppose that one group of statements is

Rectangle2D is a subtype of RectangularShape  ______
RectangularShape is a subtype of Rectangle2D  ______
Neither is a subtype of the other

You would circle “Rectangle2D is a subtype of RectangularShape” because that statement is correct as far as the Java compiler is concerned. You would not write an E next to it, since this could never result in a runtime exception due to a type error. You would not circle the other two statements.

Hint: note that any type $T$ is a subtype of itself.

(a) Integer is a subtype of Object  ______
Object is a subtype of Integer  ______
Neither is a subtype of the other

(b) Integer[] is a subtype of Object[]  ______
Object[] is a subtype of Integer[]  ______
Neither is a subtype of the other

(c) LinkedList<Integer> is a subtype of LinkedList<Object>  ______
LinkedList<Object> is a subtype of LinkedList<Integer>  ______
Neither is a subtype of the other

(d) LinkedList<? extends RectangularShape> is a subtype of LinkedList<? extends RectangularShape>  ______
LinkedList<? extends RectangularShape> is a subtype of LinkedList<?>  ______
Neither is a subtype of the other

(e) LinkedList<? extends Object> is a subtype of LinkedList<?>  ______
LinkedList<? extends Object> is a subtype of LinkedList<?>  ______
Neither is a subtype of the other
11. (10 points) True or false? Write T or F on the line in front of the question.

(a) ______ Racket’s `eq?` function could be added to OCTOPUS as a new primitive function.
(b) ______ Adding support for floating-point numbers to OCTOPUS would require changes to the lexer and/or parser, in addition to changes to the interpreter.
(c) ______ The class `Class` in Ruby is a subclass of itself.
(d) ______ The class `Class` in Ruby is an instance of itself.
(e) ______ Any two Haskell lists can be tested for equality, since the list type is in the `Eq` type class.
(f) ______ Any `let*` expression in Racket can be rewritten as a set of nested `let` expressions.
(g) ______ Adding a cut to a Prolog program may change the number of answers that are returned, but will never result in different answers.
(h) ______ Java methods can be contravariant in the return type.
(i) ______ Java methods can be overloaded based on the declared types of the method arguments.
(j) ______ In Ruby, a singleton class has only one superclass, but other classes may have multiple superclasses.

12. (12 points) Consider the following Ruby class definitions.

```ruby
class Book
  attr_reader :author, :title
  def initialize(author, title)
    @author = author
    @title = title
  end
  def description
    title + " by " + author + "."
  end
end

class Textbook < Book
  attr_reader :subject
  def initialize(author, title, subject)
    super(author, title)
    @subject = subject
  end
  def description
    return super + " A textbook about " + subject + "."
  end
end

class AnonymouslyWrittenBook < Book
  def initialize(title)
    @title = title
  end
  def author
    "anonymous"
  end
end
```
Suppose we make three objects \( b, t, \) and \( a \) by evaluating these statements:

\[
b = \text{Book.new("J.K. Rowling", "Harry Potter and the Deathly Hallows")}
\]
\[
t = \text{Textbook.new("James Stewart", "Calculus", "mathematics")}
\]
\[
a = \text{AnonymouslyWrittenBook.new("Haskell Good")}
\]

Then what is the result of evaluating each of these expressions? Hint: the \text{instance_variables} method returns an array of instance variable names, like this: \([:x, :y]\).

\[b.\text{description}\]

\[t.\text{description}\]

\[a.\text{description}\]

\[b.\text{instance_variables}\]

\[t.\text{instance_variables}\]

\[a.\text{instance_variables}\]