This is an individual-effort exam. You may read notes, textbooks, or the internet, but you may not post questions to online forums or share answers with classmates. Meaningful attempts must be made on all questions. Exams with any blank or perfunctory answers will be awarded zero Blugolds.
1. The Big Funzionale

Complete the following tasks using Haskell. All can and should be solved with non-complicated one-liners using the functional abstractions we discussed. Test your solutions on an actual computer.

(a) Write a function `decades` that accepts a list of integers. It returns a list like the parameter list, but in which each integer has been rounded down to the nearest multiple of 10. For example, `decades [14, 28, -15, 130, 38, 96] → [10, 20, -20, 130, 30, 90]`. No obscure library function is needed to solve this.

(b) Write a function `geomean` that accepts a list of numbers and returns the geometric mean of the list. The geometric mean of numbers \(n_1 through n_k\) is the \(k^{th}\) root of their product: \(\sqrt[k]{n_1 \cdot n_2 \cdot \ldots \cdot n_k}\). Recall that the \(k^{th}\) root of a value is that value raised to the \(\frac{1}{k}\) power. Use Haskell’s ** exponentiation operator. For example, `geomean [1, 4, 10] → (1 \cdot 4 \cdot 10)^{\frac{1}{3}} \approx 3.41995`.

(c) Write a function `nonempties` that accepts a list of lists and returns a list containing only the non-empty lists. For example: `nonempties ["abc", ",", "123", ","] → ["abc", "123"]`.

(d) Write a function `swap` that accepts a list of pairs. It returns a list in which the order of each pair has been swapped. For example, `swap [(1, 10), (3, 4)] → [(10, 1), (4, 3)]`.

\(^1\)Hint: how do you remove the ones-place from a number? And then what?
2. Joogle

Write in Java a static method `getMatching` to identify all the methods of a class that match a given signature. It accepts the following parameters, all of type `Class<?>:: a class whose methods are to be searched, a return type, and a varying number of parameter types. (For the parameter types, investigate and use Java’s ... syntax.) Return a `List<String>` of method names (just the names) in the specified class that have exactly the given return and parameter types. Bonus points if you use the streams API. Test your solution on a computer.
3. Namingway

(a) Every programming language with named abstractions has a set of rules describing legal identifiers. Ruby’s rules are a little different. Investigate Ruby’s rules and briefly summarize them in your own words.

(b) What would be the implications of allowing Java’s identifiers to begin with a number? Of allowing identifiers to contain non-alphanumeric characters?
4. Pass-by-foo

We discussed three different parameter-passing strategies: pass-by-copy (call-by-value), pass-by-closure (call-by-name), and pass-by-jit (call-by-need). There are a few other strategies; let's think about one of them.

Run this chunk of C++ code.

```cpp
void clamp(int &value) {
    if (value < 0) {
        value = 0;
    } else if (value > 255) {
        value = 255;
    }
}

int main(int argc, char **argv) {
    int foo = 500;
    clamp(foo);
    std::cout << foo << std::endl;
}
```

(a) What is the output?

(b) What parameter-passing strategy is at work here? (It's not one of the three mentioned above.) Investigate and use C++ parlance.

(c) Sometimes people claim that Java supports this same parameter-passing scheme. Does it? Why or why not? Why do people say it does?
5. i'll Be There

Eventually you’re going to need an abstraction that processes or visits all elements of a list—one that can be used with lambdas! But suppose the code that does the visiting needs to know each element’s index within the collection, as with Ruby’s each_with_index. Write a version of this in Java 8.

(a) Write a generic SAMI named IndexedVisitor that imposes a method named visit on its implementors. It accepts an element of type \( T \) and an index of type \( \text{int} \).

(b) Write a generic static method named eachWithIndex that accepts a List of \( T \) and an IndexedVisitor of \( T \). It visits each element in the list, passing the element and its index to the supplied visitor.

(c) Test your code on an actual computer with the following main:

```java
public static void main(String[] args) {
    eachWithIndex(Arrays.asList("Mercury", "Venus", "Earth"), (name, i) -> {
        System.out.printf("%d: %s%n", i, name);
    });
}
```

You should see this output:

```
0: Mercury
1: Venus
2: Earth
```

Does it work?