# Final Study Guide

1. When a script is loaded on a page, and a variable is declared via `var` at the top level in that script, what visibility does this variable have, e.g. in other scripts that were loaded before or after this script?
2. How many different types of numbers does Javascript have? (e.g. are integers different from floating point numbers? Are there different kinds of integers or different kinds of floating point numbers?)
3. True or False: A function must be called with the correct number of arguments. What happens if it is not?
4. True or False: In order to have objects in Javascript, we must create a class instead.
5. Explain the distinct roles of HTML, CSS, and Javascript in the development of a web application.
6. There are two different equality tests in Javascript. Explain their differences, with specific examples demonstrating the interesting behavior of each case.
7. Describe what "immediate function invocation" is, what its syntax is like, and why we would use it. Demonstrate with at least one example.
8. Have a solid understanding of the scoping rules for local variables and their visibility.
9. Describe what the array methods `map`, `filter` and `reduce` do, and what arguments they take (and if those arguments are functions, what inputs those functions take and what they are supposed to do).
10. Describe how to do the following with an object:

    - Set a value for a key. Both in the case where the key's name is fixed and known beforehand, and in the case where the key's name is given as the value of a variable.
    - Get the value for a given key.
    - Test if an object has a given key.
    - Get an array of all an object's keys.

11. Describe the DRY principle, its importance, and some examples of its usage.
12. What are the four different ways that a function may be called? How is the `this` object determined in each case?
13. Explain how object prototypes work.
14. There are two ways we learned for creating classes:

    - Write a constructor function that also represents the class, so something like `function Card(...) { ... use this ... }`. Then call this as `new Card(...)`.
    - Write an object with a `new` method, which in turn creates and returns an object via `Object.create`. Then call this as `Card.new(...)`.

    Write using both techniques (so two different versions of this question) a `Sum` class whose constructor takes as input two numbers `a`, `b` and stores them in instance variables with the same name. The prototype of the class needs to also contain a `getSum` method that returns the sum of the two numbers. Your constructor does *not* need to validate its inputs for correctness.

15. Describe how Object.defineProperty works, and what specific parameters you can use to specify a property's behavior (enumerable etc).
16. Describe how setTimeout and setInterval work.
17. Describe how the basic event mechanism works and what purpose it serves. What are its advantages over other methods of inter-module communication?
18. The visitor pattern concerns itself with a situation where we have many different classes, all part of the same superclass, and we want to implement many different operations on them. Consider the general/abstract class Shape, with specific/concrete subclasses Triangle, Rectangle, Circle, Line, and some possible operations draw(canvas), computeArea(), rescale(factor), etc.

    - If we do not use a visitor pattern, describe where these operations would be implemented, and what problems this causes.
    - If we do use a visitor pattern, describe what methods we would implement in these subclasses, and also what other classes we need to set up. In particular, describe how the above operations may be implemented, also accounting for the state that they need to maintain (e.g. the canvas, the scaling factor etc).
    - The problem space can grow in two ways: by adding more shapes or by adding more operations. Describe how the visitor approach and the non-visitor approach handle these two ways, and in particular which is suited for which way.

19. Describe the AMD module approach and the Node module approach, and their differences and similarities. What are the advantages of these approaches over "using globals"?