Here are examples of the *kinds of questions* that might be asked. This is not intended to be a sample exam; the topics covered below are not intended to be an exhaustive review. In particular, knowing the answers to all the questions below will not guarantee a good grade on the exam!

1. True or False: This demonstrates the fact that Python is statically typed. **FALSE.** The type of variable `a` is determined at the time the function is called with a particular argument; a compiler cannot determine `a`'s type.

2. Variable `boo` is unchanged—still 42. Variable `ghost` is equal to 21 since its address is in `phantom`.

3. (a) and (c) are illegal—`mummy` can't call functions that are “hidden” inside `wolfman`. But (b) and (d) are legal, since function `A` can call function `B` if `A` is inside the scope of `B`.

4. ```
   "bat" ++ "boris" = "bat","boris"
   ```

5. The correct answer is (d). There is an infinite recursion in (a) (no base case tested); (b) is just wrong (makes no use of the digits); (c) returns only the last digit, not the sum of the digits.

6. When a variable can be simultaneously viewed as more than one type, we have **polymorphism**.

7. Location = 0 + 4(3 * r + c). For instance, `fang[2][1]` is in location `0 + 4(3 * 2 + 1) = 28`.

8. Orthogonality is a property that measures how well different language features work together. And I think I told you in class that I would not ask for definitions of terms whose meanings are somewhat vague!

9. Functions that rely on “tail recursion” can be efficiently implemented without actually performing any recursive calls.

10. It depends on the program. If the program tends to process data a column at a time, column-major is likely more efficient because of the principle of “locality of reference”—successive memory accesses will tend to be in the same block of memory, reducing the need for many memory accesses. Similarly, row-major order is better for accessing data a row at a time.

11. In Haskell, currying is defining a function by partially specifying the parameters of another function. E.g., in the following, `g` is defined as `f` with the first parameter of `f` set to 10:

```haskell
Prelude> let f x y z = x+y+z
Prelude> f 1 2 3
6
Prelude> f 5 2 9
```
12. Types provide an implicit context for certain operations, freeing the programmer from having to explicitly list all the assumptions and requirements necessary for an operation to take place. Types limit the set of operations that may be performed on values, preventing invalid or nonsensical operations from being carried out.

13. Primitive types include things like int, char, boolean. Composite types include things like arrays, records, lists, and sets.

14. An enumerated, or enumeration, type is just a collection of named elements, with no other properties (other than the ability to be compared with each other and tested for equality). In C we can write:

   ...
   typedef enum {michelangelo, donatello, raphael, leonardo} ninjaturtle;
   int main() {
     ninjaturtle x = raphael, y = leonardo;
     if (x != y) printf("different\n");
     else printf("same\n");
   ...  

   In Java we can write:

   enum Turtle {
     mike,don,ralph,leo;
   }

   public class Ninja {
     public static void main(String[] args) {
       Turtle a = Turtle.mike, b = Turtle.don;
     ...  

15. Since head[black] = [1,2,3] and tail lagoon = [8,9], it will print [1,2,3,8,9].

16. For instance, false += 1 is meaningless in Java; 10+'a' is disallowed in Haskell; etc. Note that in both Java and C, the expression 10+'a' is legal, so examples of this are not universally valid in every language.

17. 🍣