1. Consider the initial configuration of a world containing two penguins and a beach ball, as shown above left. Based on this initial world, describe the result of the code depicted above right. Do not describe each instruction individually; rather, describe at a high level the action that would take place once you start the world.

The chicken turns to face the beach ball. The chicken then jumps onto the beach ball. The penguin and the monkey then simultaneously turn toward the beach ball. The beach ball and the chicken on top of it then turn to face the monkey and move toward it (because orient to the penguin means that the beach ball and the chicken both now have the same forward direction as the penguin). The beach ball and the chicken on top of it then turn to face the penguin and move toward it (because orient to the monkey means that the beach ball and the chicken both now have the same forward direction as the monkey).

2. Briefly explain the difference in effect between changing the property for an Alice object in the “Add Objects” window versus changing the property in the program code editor. Give an example to support your answer. (Note that I am not asking you how to do it; I’m asking you what the resulting effect is when you run your program.)

If you change the property in the “Add Objects” window, the property will be set starting at the beginning of the program execution and persist until the property is changed in the program code. When the property is set using the program code, the property is not changed until the program executes the code that sets it.

As an example, consider the color property of an object. If you set the object's color to yellow in the “Add Objects” window, the object will start yellow and remain yellow (unless otherwise changed during execution). If, however, you set the object's color to yellow using an instruction in a method, the object will start with its default color, and will turn to yellow only when your instruction to change the color is executed.

3. (a) In Alice, what is a method? Give an example.
   (b) In Alice, what is a function? Give an example.

   A method performs an action; a built-in function returns a value (whether number, object, Boolean, etc.). The move to method causes an object to move a distance in a scene. The distance to built-in function returns a number that specifies how far two objects are from one another. You might use the value returned by distance to as an argument to move to.

4. Briefly explain why some numbers cannot be represented precisely in a computer.

   Regardless of the word size in a computer, there are only a finite number of bits that can be used to represent any number. This means that some numbers (π for example) whose decimal representations have an infinite number of digits, cannot possibly be represented exactly in a computer. One can also use a counting argument: there are an infinite number of real numbers between 0 and 1, so since the computer can only represent a finite number of these exactly, there are in fact infinitely many numbers that cannot be represented precisely on a computer.

5. Consider the binary sequence 00011000. Give three possible interpretations (not representations) for this sequence.

   We can interpret this sequence as an integer (i.e., with value 24), an ASCII character, or an instruction (i.e., Woody’s CopyFrom location 24).
6. Suppose you are creating a set of instructions for a new machine. The total number of instructions in this machine is 16. The word size for the machine is 12 bits. Some instructions may specify a memory address as an argument.

   (a) In a 12-bit word, how many bits are necessary for specifying an instruction code? Explain.

   Since we want to encode 16 instructions, we need 4 bits to do so because \(2^4 = 16\).

   (b) How many bits will be used for specifying a memory-address argument? Explain.

   That would leave \(12 - 4 = 8\) bits for specifying memory locations.

   (c) How many memory cells will the computer have? Explain.

   This means we can specify up to \(2^8 = 256\) different memory cells.

7. By default, the CPU executes machine language instructions in the exact order they appear in the program. Briefly explain how this behavior can be altered. (Use terms like IR, PC, and DR as necessary.)

   The CPU executes machine language instructions in the exact order they appear in the program because in the typical fetch-execute cycle, the PC is incremented by one. There are some instructions, however, such as the `IfNegGoto` instruction in Woody, that cause the contents of the PC to be altered. In this case, instructions may not be executed in the order in which they appear.

8. Why would it be a very bad idea to omit a stop instruction from a machine language program?

   If there is no stop machine language instruction, then a program can potentially execute through the entire contents of memory (since the CPU does not distinguish between data and instructions — if something is pointed to by the PC, the CPU considers it an instruction and executes it). This can be especially bad if a hacker type person has placed executable machine instructions in some data file (in which case the machine can be compromised by the execution of these malicious instructions).

9. (a) Give the 8-bit sign magnitude representation for the decimal number –101.

   (b) Using your 8-bit result from part (a), give the corresponding hexadecimal representation.

   First, we need to find the binary representation of +101. To do this we look at powers of 2:

   The largest power of 2 less than or equal to 101 is \(2^6 = 64\). The remainder when 64 is subtracted from 101 is 37. The largest power of 2 that is less than or equal to 37 is \(2^5 = 32\). The remainder when 32 is subtracted from 37 is 5. The largest power of 2 that is less than or equal to 5 is \(2^2 = 4\). The remainder when 4 is subtracted from 5 is 1. The largest power of 2 that is less than or equal to 1 is \(2^0 = 1\). Thus the binary representation of 101 is

   \[
   64\ 32\ 16\ 8\ 4\ 2\ 1\\
   1\ 1\ 0\ 0\ 1\ 0\ 1
   \]

   Since the number we want is -101, we add the one bit to the left of this, to get the 8-bit sign magnitude representation 11100101. To get the hexadecimal representation of this, we divide it into groups of 4 (with groupings made from right to left). That is, we need the hexadecimal digits corresponding to 1110 and 0101. These are E and 5, so the hex representation is E5.
10. Consider the assembly language program shown below:

```
Loop:  CopyFrom  C
      Add       A
      CopyTo    C
      CopyFrom  Counter
      Add       One
      CopyTo    Counter
      Subtract  B
      IfNegGoTo Loop
      CopyFrom  C
      Print
      Stop

A:  11
B:  2
C:  0
Counter: 0
One:  1
```

(a) What value(s) is(are) sent to the output unit? (You do not have to show the fetch-execute cycle of the program.)

(b) If memory location A initially contains 3 and B initially contains 2, what value(s) is(are) sent to the output unit?

(c) Very briefly describe the purpose of this program.

The program is too complicated to write out all of the individual steps (though anyone who does not understand it is encouraged to come see me for a more detailed explanation). However, at a high level, what is happening is that the content of location A is repeatedly added to itself (and this sum stored in location C). How many times is it added to itself? Well, the variable Counter is increased by one each time through the loop. We continue looping until B subtracted from Counter is no longer negative (e.g., we loop exactly B times). Thus A is added to itself B times (so if A is 11 and B is 2, we get $11 + 11 = 22$; if A is 3 and B is 2, we get $3 + 3 = 6$; if A is 4 and B is 3, we get $4 + 4 + 4 = 12$). Very briefly, the program multiplies A by B and prints the product.