1. Encapsulation

Encapsulation refers to the idea of hiding details inside an outer container, details that would otherwise be distracting or unnecessarily complicated for us to think about.

Example of Encapsulation
People drive a car without knowing exactly how the steering works, the engine works, or the brakes work. The details of a car's mechanical operation are hidden away from us.

Example of Encapsulation
What is the square root of 138384? To get this answer, most people use a calculator or their smart phone to get the answer. The details of how to calculate the square root of a number are hidden away.

Example of Encapsulation
Sending a text on a cellphone is easy for us to do, but it's a very complex process that involves the digital electronics in your phone, encoding of your text message into binary, a radio transmitter, the cellphone company's antennas... The details of how a text gets sent are *encapsulated*.

Write your own example of encapsulation

Write two examples of your own in which the idea of encapsulation occurs. Explain briefly what is encapsulated or "hidden away" in the examples you chose.

1.

2.

2. The Binary System

The binary System is a number system in which there are only two symbols, 0 and 1.

You are certainly already familiar with the "base-10" number system that we all use—*decimal*—with its "ones place," its "tens place," its "hundreds place," and so on. You probably don't think about it all the time because it's been long time since you learned that. Here's how it was probably explained to you.

2.a. Base 10 = Decimal

Decimal numbers in base-10, have 10 symbols (0-9), and are written and thought of in terms of "places":

10^6	10^5	10^4	10^3	10^2	10^1	10^0
millions	hundred-thousands	ten-thousands	thousands	hundreds	tens	ones

We're already familiar with base-10 so there's not much explaining that needs to happen, although it's helpful to note that the number 437_{10} is:
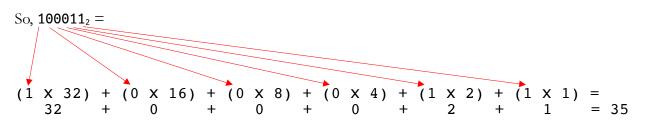
```
(4 X 100) + (3 X 10) + (7 X 1) =
   400     +    30    +    7    =   437
```

2.b. Base 2 = Binary

This same process is how numbers in other bases can be analyzed.

2^6	2^5	2^4	2^3	2^2	2^1	2^0
sixty-fours	thirty-twos	sixteens	eights	fours	twos	ones

So, 100011_2 =

```
(1 X 32) + (0 X 16) + (0 X 8) + (0 X 4) + (1 X 2) + (1 X 1) =
   32     +    0     +    0    +    0    +    2    +    1    = 35
```

To convert a decimal number to binary, find the largest power of 2 that will divide into that number. Write down a 1 in that position, and then work you way down (to the right) to see if any additional powers of 2 will divide into that number.

Example: Convert the number 45 from base-10 to base-2 (binary)
Starting with a pretty high power of 2, 64 doesn't divide into 45. The next smallest power of 2, though, does. 32 divides into 45, so that gives us "1" thirty-two.

0	1	?	?	?	?	?
sixty-fours	thirty-twos	sixteens	eights	fours	twos	ones

Subtract the 32 from the 45 to get 13 left over, and keep on moving down. 16 doesn't go into 13, so that's a 0, but 8 does…

0	1	0	1	?	?	?
sixty-fours	thirty-twos	sixteens	eights	fours	twos	ones

Subtract the 8 from 13 and we have 5 left over.

4 goes into 5 once, leaving us just one left over, so the binary version of 45 is: **101101**

0	1	0	1	1	0	1
sixty-fours	thirty-twos	sixteens	eights	fours	twos	ones

These 0s and 1s are used to operate the digital hardware, the memory locations, and the logic gates in a digital computer that solve problems and produce output. While these digits are hidden ("encapsulated") well beneath the keyboards, mice, and monitors that we use to interact with our computers, ultimately, everything that a computer does is represented by and manipulated by the binary numbers that encode those processes

In fact, before we had keyboards, mice, and monitors, the personal computer was simply a set of switches with lights above them. Computer code was entered using the individual switches—a long, painful, error-prone process—and output was read from the computer as a series of flashing lights on a display panel.

Most people who use a computer don't need to know the details of machine language, and how specifically the 0s and 1s—those "binary digits," or bits—are used to manipulate those machines. But some people do. Some data is managed as a series of bits, and "bit-wise operations" can be use to process certain types of data very efficiently.

Calculating Binary Numbers

Convert each of these binary numbers to decimal.

1. `00010101 =`

2. `00101101 =`

3. `00011000 =`

Convert each of these decimal numbers to binary.

1. `32 =`

2. `13 =`

3. `78 =`

3. ASCII and UTF

Every letter that gets typed into a computer or cellphone, and every letter that gets displayed on a screen or printed on a piece of paper, is encoded as a numeric value.

The first system for encoding characters into number was called the American Standard Code for Information Interchange (ASCII). It was a simple system that had numeric codes for characters, punctuation, numbers, etc., but it only included the English alphabet and only allowed for 256 codes.

The Unicode Transformation Format (UTF) was developed later to allow for a greater variety of encodings, including codes for non-English letters and symbols. UTF is backwards compatible with the first 128 characters of ASCII—codes 0-127—but codes above 127 are different, and should be identified using a Unicode table (which you can find online).

Using ASCII codes

Look online to find an ASCII or UTF-8 table that displays information for characters, the equivalent ASCII code in decimal, and the equivalent binary code for that character. Use that table to solve the following problems.

1. What does the ASCII decimal sequence `72, 69, 76, 76, 79, 33` mean in English?

2. What does the ASCII binary sequence `01101101, 01111001, 00100000, 01100100, 01101111, 01100111` mean in English?

3. What does the UTF-8 decimal sequence `161, 115, 101, 241, 111, 114, 105, 116, 97, 33` mean in Spanish?