Introduction to Number Systems

History of the Decimal System (*Optional?*)

We sometimes take our present day number system for granted, but there was a time when fractions, decimals, negative numbers, and zero were not recognized as commonplace numbers!

Read the articles on the websites below and create a timeline of the history of the decimal system. (You may want to use other websites to help you in your search!) Try to find the following information:

- When/where was the decimal (base 10) system created? India around the 6th/7th century.
- When/where was the number 0 first used? Babylon around 400 to 300 BC.
- When/where were negative numbers first used? China (although a time period is not specified.) Also, India around 200 BC.
- When/where were decimals used to represent fractions? Ancient China (date not specified), Babylonians (around 1000 to 500 BC), medieval Arabia. Modern system of writing decimals introduced around 1585 in Europe.

Add any other information that you find interesting to your timeline!

https://www.britannica.com/topic/Hindu-Arabic-numerals
http://www.csun.edu/~hbund408/math%20history/math.htm
https://nrich.maths.org/5747
https://www.scientificamerican.com/article/history-of-zero/
http://www.mathpages.com/home/kmath298.htm
https://www.britannica.com/biography/Simon-Stevin

Decimal System Basics

Let's go back to the basics of numbers. Our number system is called the **decimal** or **base 10** system. It has **10 single digits**: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. After 9, we run out of single digits and have to use **two digits** instead. So we add a space to the left to represent a unit of ten and put a 1 in that space and a 0 to the right to represent zero units of one. We count the way we normally would, and after reaching a digit of 9 in each of these two spaces (99), we add a third space to the left with a 1 to represent a unit of a hundred and put a 0 in the tens and the ones spaces. We continue doing this to represent larger and larger numbers.

When we look at a number like 7342, we know that each column represents a different power of ten:

$1000 = 10^3$	$100 = 10^2$	$10 = 10^1$	$1 = 10^0$
7	3	4	2

There are seven 1000s, three 100s, four 10s, and two 1s. We could even write an expression using addition and multiplication to show this number:

$$7(10^3) + 3(10^2) + 4(10^1) + 2(10^0)$$

Amazingly, we can use the rules mentioned above to create a number system using any number base we want!

Here are rules again (now for any number base other than base 10):

1. The number of the base is equal to the number of **single** digits (single digits can be numbers, letters, symbols...you'll see why in a moment). For example, in base 10, there are 10 single digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. In base 4, we would use the 4 single digits: 0, 1, 2, 3. In base 7, we would use the 7 single digits: 0, 1, 2, 3, 4, 5, 6. **What would we use for the single digits in base 2?** 0 and 1 **In base 5?** 0, 1, 2, 3, 4 **What if we wanted to use a base higher than 10? What issue do we have?** In a base higher than 10, we run out of single

2. Once we have our list of single digits, we use a space system similar to our regular base 10 system. We place a single digit in each space. The rightmost space represents **units of ones**.

   The space to the left of that represents **units of the base**. In base 10, that would be units of ten. In base 4, that would be units of 4. In base 18, that would be units of 18.

   The space to the left of that represents **units of the base squared**. In base 10, that would be units of a hundred. In base 4, that would be units of 16. In base 18, that would be units of 324.

   The space to the left of that represents **units of the base cubed**. And as we move further to the left, the power of the base increases.

So why did the decimal system become the most popular and widely-used number system? It likely comes from the fact that we are used to counting to units of ten on our fingers. **And how are other number systems useful?** We are going to be exploring two different number systems that are used extensively in the field of computer science: Binary and Hexadecimal.

In the next several sections, we will learn how to convert between Binary, Decimal, Hexadecimal, and other number systems. We will program several converters, which will automatically take numbers and convert them to different number systems. We will learn how computers use binary and hexadecimal to store information, and the basics of image storing and processing.

Binary

Binary, as you may have already have guessed, is a number system with **base 2**. Since it is base 2, it uses the two single digits 0 and 1. Each column/space in a binary number will contain either a 1 or a 0 and will represent a power of the base 2.

Converting from Binary to Decimal: suppose we wanted to see what the binary number 1100100 is as a decimal number, we could write out the following table:

$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
1	1	0	0	1	0	0

Using our table, we see that 1100100 = $1(2^6) + 1(2^5) + 0(2^4) + 0(2^3) + 1(2^2) + 0(2^1) + 0(2^0)$ = 64 + 32 + 4 = 100.

Converting from Decimal to Binary: suppose we want to convert a decimal number like 219 into a binary number. Discuss with your group members how you might do this.

It might be easier to think about smaller numbers first. For example, let's try counting in binary. The first 20 decimal numbers are listed. What are their binary representations? What multiples of powers of 2 are we adding together? The first several are done for you.

Decimal	Binary	Mutiples of Powers of 2
0	0	0
1	1	1(1)
2	10	1(2) + 0(1)
3	11	1(2) + 1(1)
4	100	1(4) + 0(2) + 0(2)
5	101	1(4) + 0(2) + 1(1)
6	110	1(4) + 1(2) + 0(1)
7	111	1(4) + 1(2) + 1(1)
8	1000	1(8) + 0(4) + 0(2) + 0(1)
9	1001	1(8) + 0(4) + 0(2) + 1(1)

10	1010	1(8) + 0(4) + 1(2) + 0(1)
11	1011	1(8) + 0(4) + 1(2) + 1(1)
12	1100	1(8) + 1(4) + 0(2) + 0(1)
13	1101	1(8) + 1(4) + 0(2) + 1(1)
14	1110	1(8) + 1(4) + 1(2) + 0(1)
15	1111	1(8) + 1(4) + 1(2) + 1(1)
16	10000	1(16) + 0(8) + 0(4) + 0(2) + 0(1)
17	10001	1(16) + 0(8) + 0(4) + 0(2) + 1(1)
18	10010	1(16) + 0(8) + 0(4) + 1(2) + 0(1)
19	10011	1(16) + 0(8) + 0(4) + 1(2) + 1(1)
20	10100	1(16) + 0(8) + 1(4) + 0(2) + 0(1)

To convert 219 into a binary number, think about how we write numbers in decimal. We find the largest power of ten that goes into the number, write that first, and then deal with whatever is left over. For example, 100 is the largest power of ten that goes into 219. It goes in twice, so we write a 2 in the hundreds place. What is left over is 19, and 10 is the largest power of ten that goes into 19. It goes in once, so we write a 1 in the tens place. What is leftover is 9, and we write that single digit in the ones place.

We can do the same thing for converting a decimal to binary. We think of the **largest** power of 2 that goes into 219, which would be $2^7 = 128$. Then we divide 219 by 128 and get 1 with a remainder of 91 left over. This means we want to put a 1 in the 2^7 column as shown in the table. Then we continue with our remainder.

The largest power of 2 that goes into 91 is $2^6 = 64$. 91 divided by 64 is 1 with 27 left over. We put a 1 in the 2^6 place and continue with 27. The largest power of 2 that goes into 27 is $2^4 = 16$. 27 divided by 16 is 1 with 11 left over. The largest power of 2 that goes into 11 is $2^3 = 8$. 11 divided by 8 is 1 with remainder 3. The largest power of 2 that goes into 3 is $2^1 = 2$. 3 divided by 2 is 1 with remainder 1. The largest power of 2 that goes into 1 is $2^0 = 1$, and 1 divided by 1 is one with remainder 0. Once we reach a remainder of 0 in this process we are done! Now we want to list the digits in order. Notice that we skipped some powers of 2. There will need to be 0s in these spaces. Note that in all the other cases, the powers of 2 only go into our numbers **once**, so all the other places have digits of 1. So we get

$2^7 = 128$	$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
1	1	0	1	1	0	1	1

And we see that 219 in decimal is the same as 11011011 in binary!

With your group members complete the "Binary to Decimal" and "Decimal to Binary" portions of your worksheet.

Hexadecimal

The word "**hexadecimal**" might have some roots that you recognize: "hex" means six and "dec" means ten, so hexadecimal is a number system with **base 16**. Since it is base 16, we need 16 single digits in our number system. We already have ten if we use 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9, but we still need six more! Any single symbol will work for the remaining six, but let's use something familiar, like our alphabet. So in hexadecimal the sixteen single digits we use are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F in that order. In hexadecimal, A represents the decimal number 10, B is 11, C is 12, D is 13, E is 14, and F is 15. (*In hexadecimal, you can use either capital letters or lowercase letters for A, B, C, D, E, and F. Just be consistent with what you use.)*

In a hexadecimal number we can have a string of any of those digits, and each space in the number will represent a power of 16.

Converting from Hexadecimal to Decimal: suppose we want to see what the hexadecimal number A8F2 is in decimal. Again, we create a table:

$16^3 = 4096$	$16^2 = 256$	$16^1 = 16$	$16^0 = 1$
A	8	F	2

We know that A is 10 in the decimal system, and F is 15, so A8F2 = $10(16^3) + 8(16^2) + 15(16^1) + 2(16^0)$ = 10(4096) + 8(256) + 15(16) + 2(1) = 43250 in the decimal system! **Test yourselves by converting the number 5DE0 to decimal.** 24032

Converting from Decimal to Hexadecimal: converting from decimal to hexadecimal is very similar to converting binary to decimal. Let's again try some smaller numbers first.

The first 20 decimal numbers are listed. What are their hexadecimal representations? What powers of 16 are we adding together? Several are done for you.

Decimal	Hexadecimal	Multiples of Powers of 16
0	0	0(1)
1	1	1(1)
2	2	2(1)
3	3	3(1)
4	4	4(1)
5	5	5(1)
6	6	6(1)
7	7	7(1)
8	8	8(1)
9	9	9(1)
10	A	10(1)
11	B	11(1)
12	C	12(1)
13	D	13(1)
14	E	14(1)
15	F	15(1)
16	10	1(16) + 0(1)
17	11	1(16) + 1(1)
18	12	1(16) + 2(1)
19	13	1(16) + 3(1)
20	14	1(16) + 4(1)

Now let's look at the decimal number 987. We want to find the largest power of 16 that goes into 987. 4096 is too large, so the largest power of 16 is $16^2 = 256$.

Dividing 987 by 256, we get 3 with a remainder of 219. We put a 3 in the 16^2 to show that there are three units of 256 in our number 987. Then we look at the remainder. The largest power of 16 which goes into 219 is 16 itself. 219 divided by 16 is 13 with a remainder of 11. Normally we would want to put 13 in the 16^1 column, but remember that we have to use a **single digit**, so we instead use the hexadecimal representation for 13, which is D. Now looking at the remainder 11, we see that the largest power of 16 that goes into 11 is $16^0 = 1$. 11 divided by 1 is 11 with remainder 0. Since we have a remainder of 0 already, we are done, and we place a B to represent 11 in the 16^0 column.

$16^2 = 256$	$16^1 = 16$	$16^0 = 1$
3	D	B

And we see that 987 in decimal is the same as 3DB in hexadecimal! **Test yourselves by converting the decimal number 12122 to hexadecimal.** 2F5A

With your group members, complete the "Hexadecimal Conversions" portion of your worksheet.

Other Number Systems

Using the rules similar to the ones for binary and hexadecimal, we can represent numbers in almost any number system we like. For example if we wanted to write numbers in base 4, we would use the single digits 0, 1, 2, 3, and each place in a number would represent a power of 4. If we wanted to write numbers in base 20, we would use the single digits 0 through 9, A, B, C, D, E, F, G, H, I, and J. Each number place would represent a power of 20.

Eventually though, we are limited by the number of letters in the alphabet. **Base 36** uses the single digits 0 through 9 AND all the letters in the English alphabet. If we wanted to continue to higher bases, we would have to start adding other symbols to use as single digits. Base 36 is fun to work with because every word in the English language can be converted to a decimal number!

With your group members, convert the following four decimal numbers into base 36 to find the sentence that is written below. Write your answer in the box.

"27749440 1366760 1442151747 63054156628."

GIRLS TALK NUMBER SYSTEMS.

With your group members, try the following challenges. Write sentences in which each word is a base 36 number and see if you can use ONLY words that are

- **Even numbers when converted to decimal.** Answers may vary. Check that each word ends in the following letters: A, C, E, G, etc. Each word must end in a letter starting with A and alternating every other letter. This ensures the word is even.
- **Odd numbers when converted to decimal.** Same as above, but words must end in every other letter starting with B (B, D, F, H, etc.)
- **Multiples of 3 when converted to decimal.** Same as above, but words must end in every three letters starting with C (C, F, I, etc.)

Conversion Programs (*Note: this Mathematica notebook requires Mathematica 10; if students do not have access to this, the code will have to be changed)*

Open the Mathematica notebook named "Converter.nb". The notebook begins with some guided exercises to help you understand how to use Mathematica and some Mathematica functions. Then there are three functions: "DecToLowBase", "DecToHighBase", and "BaseToDec". The commented sections in blue (surrounded by "(*" on the right and "*)" on the left) above each function describe what the function does. The commented sections inside the functions tell you what kind of code to write.

For each numbered comment, write code or perform the action given by the instructions in the comment. If you are not sure what a function does, run a Google search for the function along with the word "Mathematica" to find the answer.

For the **first function** we are going to be programming the following algorithm. Divide the decimal number by the base of the system we are converting to. Record the remainder. Take the quotient and repeat this process until the quotient is 0. Then write the remainders backwards. This will give you the correct number in the new number system. Convince yourself that this algorithm works by trying it on paper with one of the decimal to binary exercises you did above. See if the algorithm works for another base, like base 7. Try to convert 143 to base 7 using the algorithm. Does it work?

For the **second function**, we are going to be using the same algorithm as in the first, but this time, some of the remainders that we want may actually be two digit numbers, and we will have to use their letter counterparts. Think about how this should be done.

For the **third function**, we want to take a number input as a string, figure out the number represented by each character, multiply by the power of the base in that column and then add everything together.

Some vocabulary that might be helpful for you to know:

- **Character**: a single letter, number, or symbol that can be encoded using 1 byte (8 bits). Characters are usually entered inside quotes (“ “)
- **String**: an expression containing one or more characters. String are also usually entered inside quotes. The **empty string** is given by two quotes with nothing inside: “”.

Computer Data Storage and Data Transfer

Read the following article and answer the questions below with your group members.
https://www.lifewire.com/the-difference-between-bits-and-bytes-816248

Questions:

1. What are bits and bytes? Standard units of digital data transmitted over network connections.
2. What is “bit” short for? Binary digit
3. What numeric values can a bit have? 0 and 1
4. How many bits are in a byte? 8
5. How many bits are in an IPv4 address? How many bytes? 32 bits, 4 bytes
6. How many bits are represented in one hexadecimal digit? 4
7. How many hexadecimal digits are needed to represent the information in 1 byte? 2
8. What is one thing hexadecimal is used for? Wireless security keys
9. Based on your reading, what do you think is the advantage of using hexadecimal over binary? Hexadecimal sequences are shorter than binary sequences because you can encode 4 bits in 1 hexadecimal digit.
10. Complete the section of your worksheet entitled “Numbering Systems Exam.” If there is a term you have not encountered in your reading, look it up!

How do computers store numbers?

So far, we have seen how to write decimal numbers in binary and how to write binary numbers as decimals, BUT all the decimal numbers we used were **positive whole numbers and zero**. In the field of scientific computing, we use computers to perform very complex calculations with **ALL** different kinds of numbers: whole numbers, negative numbers, and numbers with long decimals. So, you might wonder, how does a computer use binary to store numbers that are negative or numbers with long decimals?

In many widely-used computer languages (like C/C++ and Java), you have to actually tell the computer what type of number you’re using before you use it. This allows the computer to

understand how to read it and how to store it and how to perform calculations. Also, computers have a **finite** amount of storage, meaning we won't be able to store all the numbers all the way up to infinity. The largest number we are able to store will depend on the number of bits we are allowed to use.

Below, read about the different types of numbers and answer the questions with your group members.

Integers:
Integers include positive and negative whole numbers and zero. Most of the time, a computer uses either 32 bits to store an integer (usually called an "int" or a "long") or 64 bits (usually called a "long long").

In the case of 32 bits, the first bit represents the sign: "0" indicates a positive number, and "1" indicates a negative number. The remaining 31 bits are used to indicate a whole number in binary. **What is the largest integer we could represent using this system?** 2,147,483,647 **About how many digits are this in decimal?** 10

We often say that the 32-bit representation can encode numbers from $-(2^{31} - 1)$ to $2^{31} - 1$. **Explain where these expressions come from.** If the first bit is used to indicate sign, we know the largest number we can form with 31 bits has a 1 in each place. If we add 1 to this, that would change all of those bits to 0 and and add another bit on the left equal to 1. This bit represents a power of 2^{31} (the other bits represent 2^0 all the way to 2^{30}), so $2^{31} - 1$ should represent 31 bits with a 1 in each position.

If we use 64 bits, again the first bit represents the sign, and the remaining 63 bits are used to indicate the whole number in binary. **What is the largest integer we could represent using this system?** 9,223,372,036,854,775,807 **About how many digits are this in decimal?** 19 **What is the range of numbers we can represent (use the exponential - 1 format used above)?** $-(2^{63} - 1)$ to $2^{63} - 1$

Numbers with Decimals:

To encode ALL types of numbers (positive/negative/decimals), computers use something called **floating point representation**. Think about a scientific calculator and how it displays answers. Each time it shows an answer, it only uses a certain number of digits. Depending on how big the answer is and how many decimal places it has, the decimal point is going to be in different locations (it is going to "float" around). Imagine trying to display 3.123435 or 213335.23435 using only 8 digits (the decimal point is also considered a digit in this case). For the first number, we'll be able to display all the digits (3.123435), but for the second, we'll have to cut it off after the first decimal place (213335.2). In each of these two representations, the decimal is located in different places.

A **float** is the name for the 32 bit version of a floating point number. It is also sometimes called **single precision**. The encoding of floating point numbers is quite different, mainly because we are now dealing with fractions as well. In a float, there are three different groupings of bits: the first bit tells us the **sign** of the number, the next 8 bits are called the **characteristic** or the **exponent**, and the last 23 bits are called the **mantissa** or the **significand**.

The characteristic is a binary encoding of a whole number. The mantissa is a binary encoding of a fractional value. Once we know what these are, we can use the following formula to actually get the number:

$$\mathbf{(-1)^{sign}(2)^{characteristic\ -\ 127}(1 + mantissa)}$$

Let's look at a number as an example and then talk about where this formula comes from!

Sign	Characteristic	Mantissa
0	10000011	10111001000100000000000

Here we see that the sign is "0", so this is a positive number. The characteristic is a normal binary representation of a whole decimal number, so we see that it is $2^7 + 2^1 + 2^0 = 131$. The mantissa is a binary representation of *descending* powers of 2 (or increasing powers of ½). The first space on the left indicates $(½)^1$, the next space to the right is $(½)^2$, etc. So our mantissa is

$$1(½)^1 + 1(½)^3 + 1(½)^4 + 1(½)^5 + 1(½)^8 + 1(½)^{12} = \frac{1}{2} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{256} + \frac{1}{4096}$$

Now using our formula, we have

$$(-1)^0(2)^{131\ -\ 127}(1 + \frac{1}{2} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{256} + \frac{1}{4096}) = 27.56640625$$

We can get the binary representation for the next largest number by simply adding a one in the last column of the mantissa: 0 10000011 10111001000100000000000. **How would you represent the next smallest number from our example?**
0 10000011 10111001000011111111111

The actual number of bits used for the characteristic and the mantissa are actually standardized by the IEEE (Institute for Electrical and Electronic Engineers). The question is why don't we actually use the characteristic to represent the whole part of our number, and the mantissa to represent the decimal part? Why do we multiply by $2^{characteristic\ -\ 127}$ in our formula?

Can you think of reasons why we wouldn't just want to use the characteristic to represent the whole part of our number? Discuss with your group members.

The answer is that if we can only use 8 bits to represent the whole part of our number, that GREATLY reduces the numbers we can represent. The highest whole number we can encode with 8 bits is $2^8 - 1 = 255$. We use the power of two to increase our range.

Now, why subtract 127 from the characteristic? We know that 2 raised to a positive power gives numbers greater than 1, and 2 raised to a negative power gives numbers less than 1. Our characteristic has a maximum value of 255. Subtracting 127 means that half of the possible characteristic values will produce numbers greater than 1, and the other half of characteristic values will produce numbers less than 1. So doing this basically equalizes the large and small numbers that we are able to encode!

A **double** is the 64-bit version of a floating point number. It is also called **double precision**. For a double, the first bit encodes the sign, the next 11 bits are the characteristic, and the last 52 bits are the mantissa. The formula for finding the decimal representation of the number is:

$$\mathbf{(-1)^{sign}(2)^{characteristic\ -\ 1023}(1 + mantissa)}$$

Explain why we are now subtracting 1023. With 11 bits for our characteristic, the largest number we can represent is $2^{11} - 1 = 2047$. The range of possible numbers we can represent then runs from 0 to 2047, a total of 2048 numbers. We want to make it equally possible to represent numbers less than 1 and greater than 1, so we find the halfway point, which would be 1024. The lower half would extend over 0 to 1023, so we subtract 1023 from the characteristic to make half of the possibilities positive powers and half negative powers.

Questions:

1. Is it possible to encode 0 in the floating point format? No. Since we add 1 to the mantissa, it is only possible to represent very, very tiny numbers, but not 0.
2. What is the decimal representation of the 32-bit floating point number:
   0 10000000 11000000000000000000000
   Sign = positive
   Characteristic = 128
   Mantissa = 0.75
   3.5
3. What is the decimal representation of the 32-bit floating point number:
   1 01111110 10000000000000000000000
   Sign = negative
   Characteristic = 126
   Mantissa = 0.5
   -0.75
4. What is the decimal representation of the 32-bit floating point number:
   1 01111110 00000000000000000000001
   Sign = negative

Characteristic = 126
Mantissa = 0.5^{23}
-0.5000000596046 (approximately)