Exercise 2

All course content was manageable and most elements were familiar with a little extra insight. There was the occasional problem officially completing the task to receive the completed status despite both the input and the solution being identical. This happened twice with the rest posing no issues.

Exercise 3

a)

Integer values are commonly used in everyday calculations but computers can only understand information in binary form so integers are coded in this form to make them understandable for computers. Integer values are often represented in the decimal format which has a base of 10, the binary form understood by computers has a base of 2 and represents the same number as a sequence of 0s and 1s. The base is the amount by which the numbers are multiplied and the base has an exponent that is relative to their position. The 0s and 1s represent bits which are the smallest piece of information that can be created but computers often handle multiple bits as a unit to make it understandable for them. The smallest of these is the 8-bit format, known as a byte. An example highlights the principle:

17 - represented as a decimal number:

$7 * 10^0 = 7$
$1 * 10^1 = 10$
17

1 0001 - is the representation in binary, we start from the right and move to the left.

$1 * 2^0 = 1$
$0 * 2^1 = 0$
$0 * 2^2 = 0$
$0 * 2^3 = 0$
$1 * 2^4 = 16$
17

This can be represented as a whole byte by placing three 0s in front of the last one on the left to make it 8 bits long.

b)

When representing numbers with decimal places in binary we use the same logic as for decimal integers but in the other direction and the base takes on negative exponents. Each number to the right of the decimal point (referred to as a binary point for binary) undergoes the same process of being multiplied by the base to the power of its position from the decimal point, as we move each time to the right the position is represented as a minus number. So one position to the right is -1 (position and power), two positions to the right is -2, three is -3 etc. The example below illustrates the concept:

17.625 - decimal number - 6, 2 and 5 are respectively -1, -2 and -3 positions from the decimal point

$5 * 10^{-3} =$	0.5
$2 * 10^{-2} =$	0.2
$6 * 10^{-1} =$	0.6
$7 * 10^{0} =$	7
$1 * 10^{1} =$	10
	17.625

10001.101 is the representation in binary:

$1 * 2^{-3} =$	0.125
$0 * 2^{-2} =$	0
$1 * 2^{-1} =$	0.5
$1 * 2^{0} =$	1
$0 * 2^{1} =$	0
$0 * 2^{2} =$	0
$0 * 2^{3} =$	0
$1 * 2^{4} =$	16
	17.625

Because numbers with decimal places have to be represented as a sum of numbers using the base 2 with a negative exponent we often have problems of accuracy. Numbers that are quite simple in decimal notation often require very long binary representation as the values created using 2^{-k} do not easily add up to certain decimal values and sometimes it is impossible, as with 0.1. When we try to reach 0.1 using binary we find that to reach the exact number we must increase by smaller and smaller iterations of numbers to the base of 2. Eventually we see that we can never get to 0.1 exactly, we can only get infinitely closer to it due to its format and the recurring nature of the numbers as we increase the number of decimal places the number has. This can be seen in **c).**

c)

17.1 --> IEEE representation:

- decimal representation = 17.1_{10}

- binary representation = 10001.0001100110011001100110011.....

17 - 10001
0.1 - .000**1100**..... (bold digits represent the recurring part of the number)

10001.0001100... = 1.**00010001100**... * 2^4 (the exponent is for because we move the binary point 4 positions to the left)
mantissa = digits in bold

sign (1 = -, 0 = +)	exponent 127 = bias 4 = exponent	mantissa
0	127 + 4 = 131_{10} 1000 0011_2	0001000**11001100** single precision requires 32 bits so the remaining spaces are filled with the recurring section in bold

solution (not rounded) = 0 1000 0011 0001 0001 1001 1001 1001 100

solution (rounded) = 0 1000 0011 0001 0001 1001 1001 1001 101