In Class Exercise on loops and if statements.

These are unrealistic problems, but they help to practice to completion.

**Problems #1:**

```javascript
start
    var firstNum = 100;
    var secondNum = 5;
    var thirdNum = 12;
    var theResult = 0;
    do while secondNum <= thirdNum
        if firstNum > 500
            firstNum = firstNum - 2;
        secondNum = secondNum + 1;
    else
        firstNum = firstNum + 100;
        thirdNum = thirdNum - 1;
    end if
    theResult = firstNum + secondNum + thirdNum;
    display theResult
end
```

**Problems #2:**

```javascript
start
    var firstNum = 25;
    var secondNum = 50;
    var thirdNum = 75;
    var fourthNum = 100;
    var ct = 0;
    var workAns = 0;
    do while ct < 6
        workAns = firstNum + secondNum;
    end do
```

Better to use var when you define the variables.

Using the while - depending on the values assigned - you might not enter the loop.
This uses the do with the while at the bottom which means the loop always gets executed at least once.
Safer to use parseFloat when you do an Add.

Problem #2:

```javascript
start

firstNum = 25;
secondNum = 50;
thirdNum = 75;
fourthNum = 100;
ct = 0;
workAns = 0;
theAns = 0;
do while ct < 6
    workAns = firstNum + secondNum
    if workAns > thirdNum
        fourthNum = fourthNum / 2;
        firstNum = firstNum * 3;
        secondNum = firstNum - secondNum;
        thirdNum = thirdNum * 2;
    else
        firstNum = firstNum + 10;
        secondNum = secondNum + 10;
        thirdNum = thirdNum + 10;
    end if
    ct = ct + 1;
end do

theAns = firstNum + secondNum * thirdNum + fourthNum;
display theAns;
end
```
Check Department

Division III has four departments.

A look at arrays.

Please enter the department number

2

OK  Cancel
I define the array in a script I put in the head. That script could also be within the body.

Here I go to the array where every dept has the same name of deptArray. However when I use an index in square braces then it makes the name unique.
Check Department

Division III has four department.
The department is CI

The end!

```html
1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<meta charset="utf-8">`
5. `<title>Basic Array</title>`
6. `<style type="text/css">`
7. `body`
8. `  {`
9. `    background-color: beige;
10. `    color: navy;`
11. `  }`
12. `</style>`
13. `<script type="text/javascript">`
14. `deptArray = new Array(4);`
15. `deptArray[0] = "No Dept 0";`
16. `deptArray[1] = "BU";`
17. `deptArray[2] = "CI";`
18. `deptArray[3] = "CU";`
20. `// I have an array that starts with 0 and ends with 4 so the number when I define`
21. `//the array is 4 event though there are 5 elements. I could eliminate the`
22. `// 0th element and it will work the same.`
23. `</script>`
24. `</head>`
25. `</body>`
26. `<h1>Check Department</h1>`
27. `<p>Division III has four department.</p>`
28. `<script type="text/javascript">`
29. `deptNo = prompt("Please enter the department number", 2);`
30. `if (deptNo < 5)`
31. `{`
32. `    document.write("The department is " + deptArray[deptNo]);`
33. `}
34. `else`
35. `{`
36. `    document.write("Invalid dept number");`
37. `}`
38. `</script>`
39. `<h1>The end!</h1>`
40. `</body>`
41. `</html>`
```
Decimal Numbering Systems:
The decimal numbering system is a base 10 numbering system (this means there are 10 digits we can use - these digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9). When we talk about a number, we understand what the number is because of its face value and its positional value. Thus, the digit 5 has a different meaning when it is in the tens position than when it is in the ones position (i.e. when it is in the tens position, we express it as 50 and when it is in the ones position, we express it as 5). In this case, the face value of the digit is 5 and the positional value of a number is based on the position it occupies. In decimal, the positional value of a number is based on the powers of 10 (remember, we are in base 10):

\[
\begin{array}{cccc}
5 & 7 & 2 & 4 \\
10^3 & 10^2 & 10^1 & 10^0 \\
1000 & 100 & 10 & 1 \\
\end{array}
\]

Face value
Positional Value (powers of 10)
Resolved positional value

To figure out the value of 5724, we do the following:

\[
\begin{align*}
5 \times 10^3 &= 5 \times 1000 = 5000 \\
7 \times 10^2 &= 7 \times 100 = 700 \\
2 \times 10^1 &= 2 \times 10 = 20 \\
4 \times 10^0 &= 4 \times 1 = 4 \\
\hline
5724
\end{align*}
\]

Binary Numbering systems:
The binary numbering system works much the same way as the decimal numbering system except that now we are in base 2 so we only have 2 digits (0, 1). The value of the number is still determined by the face value times the positional value, but since we are in base 2, the positional values are the powers of 2. Since the face values can only be 0 or 1, this means that the 0 or 1 is multiplied by the positional place in which it is found.

Example: binary number 1011011

\[
\begin{array}{cccc}
1 & 0 & 1 & 1 \\
2^6 & 2^5 & 2^4 & 2^3 \\
\end{array}
\]

Face value
Positional value
Binary Numbering systems:
The binary numbering system works much the same way as the decimal numbering system except that now we are in base 2 so we only have 2 digits (0, 1). The value of the number is still determined by the face value times the positional value, but since we are in base 2, the positional values are the powers of 2. Since the face values can only be 0 or 1, this means that the 0 or 1 is multiplied by the positional place in which it is found.
Example: binary number 1011011

<table>
<thead>
<tr>
<th>Face value</th>
<th>2⁶</th>
<th>2⁵</th>
<th>2⁴</th>
<th>2³</th>
<th>2²</th>
<th>2¹</th>
<th>2⁰</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>64</td>
<td>32</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

The positional values are first shown in the powers of 2 and then as the resolved number - in other words, 2 to the 6th is equal to 64.

Converting binary to decimal:
In the previous example to find the decimal equivalent for the number 1011011, we do the following:

\[
\begin{align*}
1 \times 2^6 &= 1 \times 64 = 64 \\
0 \times 2^5 &= 0 \times 32 = 0 \\
1 \times 2^4 &= 1 \times 16 = 16 \\
1 \times 2^3 &= 1 \times 8 = 8 \\
0 \times 2^2 &= 0 \times 4 = 0 \\
1 \times 2^1 &= 1 \times 2 = 2 \\
1 \times 2^0 &= 1 \times 1 = 1 \\
\end{align*}
\]

Converting decimal to binary:
Before doing this it is important that we review the decimal equivalent for the frequently used powers of 2:

\[
\begin{align*}
2^0 &= 1 \\
2^1 &= 2 \\
2^2 &= 4 \\
2^3 &= 8 \\
2^4 &= 16 \\
2^5 &= 32 \\
2^6 &= 64 \\
2^7 &= 128 \\
2^8 &= 256 \\
2^9 &= 512 \\
2^{10} &= 1024 \\
\end{align*}
\]
Dec  Bin
0   0
1   1
2   10
3   11
4   100
5   101
6   110
7   111
8   1000
\[
125_{10} = \frac{64}{61} - \frac{32}{29} - \frac{16}{13} - \frac{8}{5} - 1
\]

\[
2 \quad 64 \quad 32 \quad 16 \quad 8 \quad 4 \quad 2 \quad 1 = \frac{125}{10}
\]
Resolved positional value

The positional values are first shown in the powers of 2 and then as the resolved number - in other words, 2 to the 6th is equal to 64.

Converting binary to decimal:
In the previous example to find the decimal equivalent for the number 1011011, we do the following:

\[
\begin{align*}
1 \times 2^6 &= 1 \times 64 = 64 \\
0 \times 2^5 &= 0 \times 32 = 0 \\
1 \times 2^4 &= 1 \times 16 = 16 \\
1 \times 2^3 &= 1 \times 8 = 8 \\
0 \times 2^2 &= 0 \times 4 = 0 \\
1 \times 2^1 &= 1 \times 2 = 2 \\
1 \times 2^0 &= 1 \times 1 = 1
\end{align*}
\]

Converting decimal to binary:
Before doing this it is important that we review the decimal equivalent for the frequently used powers of 2:

\[
\begin{align*}
2^0 &= 1 & 2^1 &= 2 & 2^2 &= 4 & 2^3 &= 8 & 2^4 &= 16 & 2^5 &= 32 \\
2^6 &= 64 & 2^7 &= 128 & 2^8 &= 256 & 2^9 &= 512 & 2^{10} &= 1024 & \text{etc.}
\end{align*}
\]

To convert 91 from decimal to binary, you can follow the following steps:

1. Look at 91 and see what power of 2 can be taken from it. The highest power that can be subtracted is 2 to the 6th which is 64. Therefore we put a 1 in the 2 to the 6th position. Then we subtract: 91 - 64 = 27

\[
\begin{array}{cccccc}
1 & 0 & 1 & 1 & 0 & 1 \\
| & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
64 & 32 & 16 & 8 & 4 & 2 & 1
\end{array}
\]
<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
</tr>
<tr>
<td>16</td>
<td>10000</td>
</tr>
</tbody>
</table>

This means that when we count, we get the following:

- | Binary |
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>100</td>
</tr>
<tr>
<td>101</td>
</tr>
<tr>
<td>110</td>
</tr>
<tr>
<td>111</td>
</tr>
<tr>
<td>1000</td>
</tr>
<tr>
<td>1001</td>
</tr>
<tr>
<td>1010</td>
</tr>
<tr>
<td>1011</td>
</tr>
<tr>
<td>1100</td>
</tr>
<tr>
<td>1101</td>
</tr>
<tr>
<td>1110</td>
</tr>
<tr>
<td>1111</td>
</tr>
<tr>
<td>10000</td>
</tr>
</tbody>
</table>

8 binary bits: 4 4
Hex: 1000 1111
ASCII: 8 F
<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>101</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>110</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>111</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>1010</td>
<td>10</td>
<td>A</td>
</tr>
<tr>
<td>1011</td>
<td>11</td>
<td>B</td>
</tr>
<tr>
<td>1100</td>
<td>12</td>
<td>C</td>
</tr>
<tr>
<td>1101</td>
<td>13</td>
<td>D</td>
</tr>
<tr>
<td>1110</td>
<td>14</td>
<td>E</td>
</tr>
<tr>
<td>1111</td>
<td>15</td>
<td>F</td>
</tr>
</tbody>
</table>

Hexadecimal base 16:

\[
\begin{align*}
\frac{9}{10} & + \frac{1}{E} \\
\frac{A}{B} & + \frac{1}{F} \\
\frac{C}{D} & + \frac{1}{10}
\end{align*}
\]

Conversion from binary to hexadecimal:

<table>
<thead>
<tr>
<th>Binary</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>10</td>
</tr>
<tr>
<td>10001</td>
<td>11</td>
</tr>
<tr>
<td>10010</td>
<td>12</td>
</tr>
<tr>
<td>Binary</td>
<td>Decimal</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>101</td>
<td>5</td>
</tr>
<tr>
<td>110</td>
<td>6</td>
</tr>
<tr>
<td>111</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>1010</td>
<td>10</td>
</tr>
<tr>
<td>1011</td>
<td>11</td>
</tr>
<tr>
<td>1100</td>
<td>12</td>
</tr>
<tr>
<td>1101</td>
<td>13</td>
</tr>
<tr>
<td>1110</td>
<td>14</td>
</tr>
<tr>
<td>1111</td>
<td>15</td>
</tr>
<tr>
<td>10000</td>
<td>16</td>
</tr>
<tr>
<td>10001</td>
<td>17</td>
</tr>
<tr>
<td>10010</td>
<td>18</td>
</tr>
</tbody>
</table>

Conversion from binary to hexadecimal:

- \(1100_2 = 14_{10} = E_{16}\)
- \(8421_{10} = 10_{16} = A_{16}\)
As you can see from the chart above, every hexadecimal digit can be represented by 4 binary digits (bits) and every combination of 4 binary digits can be represented by a single hexadecimal digit. Because of this, any string of 4 binary digits can be converted to its hexadecimal equivalent by either checking the chart above or doing the conversion. If I have a string of binary digits, it can be divided into groups of four starting at the far right and each group can be converted to its hexadecimal equivalent.

Examples:

1. 11001111 can be divided into groups of 4
   1100/1111 1100 is C and 1111 is F
   \[ \text{C} \quad \text{F} \]
   therefore the hexadecimal equivalent of 11001111 is CF

2. the hexadecimal equivalent of 1010101110101 is 3575
   1010101110101 = 11/ 0101/ 0111/ 0101
   \[ 3 \quad 5 \quad 7 \quad 5 \]

Note that if there are not enough digits to make groups of 4, then you start at the right and the group with less than 4 digits is the leftmost group D in translating that group, assume leading 0s.

**Conversion from hexadecimal to binary:**
To convert from hexadecimal to binary, you express each hexadecimal character as 4 binary digits D either use the chart above or figure out the equivalent.
As you can see from the chart above, every hexadecimal digit can be represented by 4 binary digits (bits) and every combination of 4 binary digits can be represented by a single hexadecimal digit. Because of this, any string of 4 binary digits can be converted to its hexadecimal equivalent by either checking the chart above or doing the conversion. If I have a string of binary digits, it can be divided into groups of four starting at the far right and each group can be converted to its hexadecimal equivalent.

Examples:

1. 11001111 can be divided into groups of 4
   1100/ 1111  1100 is C and 1111 is F
   C     F
   therefore the hexadecimal equivalent of 11001111 is CF

2. the hexadecimal equivalent of 11010101110101 is 3575
   11010101110101 = 11/ 0101/ 0111/ 0101
   3      5      7      5

Note that if there are not enough digits to make groups of 4, then you start at the right and the group with less than 4 digits is the leftmost group D in translating that group, assume leading 0s.

**Conversion from hexadecimal to binary.**
To convert from hexadecimal to binary, you express each hexadecimal character as 4 binary digits D either use the chart above or figure out the equivalent.