1. Take input text and split it into an array of the characters' ASCII codes

'A Test'

[A, , T, e, s, t]

```
function sha1(text) {
  const asciiText = text.split('')
    .map((letter) => utils.charToASCII(letter));
```

[65, 32, 84, 101, 115, 116]

2. Convert ASCII codes to binary

3. Pad zeros to the front of each until they are 8 bits long

[65, 32, 84, 101, 115, 116]

```
let binary8bit = asciiText
                 .map((num) => utils.asciiToBinary(num))
                 .map((num) => utils.padZero(num, 8));
```

[01000001, 00100000, 01010100, 01100101, 01110011, 01110100]

4. join and append a 1

```
let numString = binary8bit.join('') + '1';
```

0100000100100000010101000110 01010111001101110100 1

5. pad the binary message with zeros until its length is 512 mod 448

```
while (numString.length % 512 !== 448) {
    numString += '0';
  }
```

01000001001000000101010001
10010101110011011101001

6. take binary 8-bit ASCII code array from step 3, get its length in binary
7. pad with zeros until it is 64 characters
8. append to your previously created binary message from step 5

```
const length = binary8bit.join('').length;
const binaryLength = utils.asciiToBinary(length);
```

48

110000

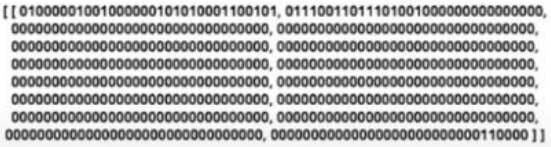
```
const paddedBinLength = utils.padZero(binaryLength, 64);
numString += paddedBinLength;
```

0000000000000000000000-
0000000000000000000000-
0000000000000110000

9. break the message into an array of 'chunks' of 512 characters
10. break each chunk into a subarray of sixteen 32-bit 'words'

```
const chunks = utils.stringSplit(numString, 512);
```

```
const chunkWords = chunks
            .map((chunk) =>
            utils.stringSplit(chunk, 32));
```

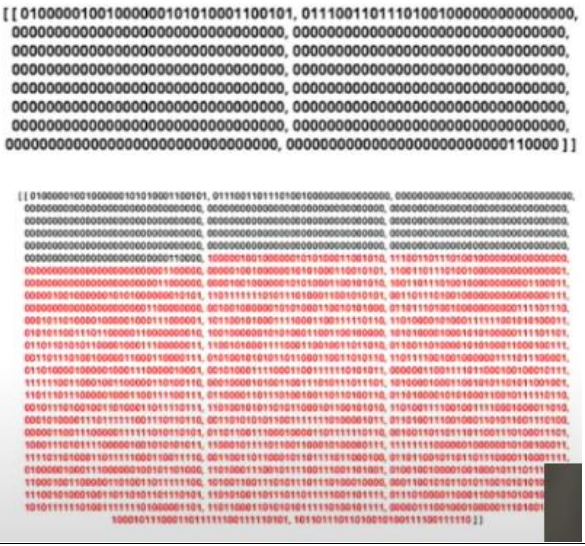


11. loop through each 'chunk' array of sixteen 32-bit 'words' and extend each array to 80 'words' using bitwise operations

```
      //chunkWords is an array of 'chunk' subarrays
const words80 = chunkWords.map((chunk) => {
//we start with a 'chunk' array of 16 32-bit 'words'
    for (let i = 16; i <= 79; i++) {
      //take four words from that chunk using
      //your current i in the loop
      const wordA = chunk[i - 3];
      const wordB = chunk[i - 8];
      const wordC = chunk[i - 14];
      const wordD = chunk[i - 16];

      //perform consecutive XOR bitwise
      //operations going through each word
      const xorA = utils.xOR(wordA, wordB);
      const xorB = utils.xOR(xorA, wordC);
      const xorC = utils.xOR(xorB, wordD);

      //left rotate by one
      const newWord = utils.leftRotate(xorC, 1);
      //append to the array and continue the loop
      chunk.push(newWord);
    }
    return array;
  });
```



12. initialize some variables

```
let h0 = '01100111010001010010001100000001';
let h1 = '11101111110011011010101110001001';
let h2 = '10011000101110101101110011111110';
let h3 = '00010000001100100101010001110110';
let h4 = '11000011110100101110000111110000';
let a = h0;
let b = h1;
let c = h2;
let d = h3;
let e = h4;
```

13. looping through each chunk: bitwise operations and variable reassignment

```
for (let i = 0; i < words80.length; i++) {
  for (let j = 0; j < 80; j++) {
    let f;
    let k;
    if (j < 20) {
      const BandC = utils.and(b, c);
      const notB = utils.and(utils.not(b), d);
      f = utils.or(BandC, notB);
      k = '01011010100000100111100110011001';
    } else if (j < 40) {
      const BxorC = utils.xOR(b, c);
      f = utils.xOR(BxorC, d);
      k = '01101110011011001110101110100001';
    } else if (j < 60) {
      const BandC = utils.and(b, c);
      const BandD = utils.and(b, d);
      const CandD = utils.and(c, d);
      const BandCorBandD = utils.or(BandC, BandD);
      f = utils.or(BandCorBandD, CandD);
      k = '10001111000110111011110011011100';
    } else {
      const BxorC = utils.xOR(b, c);
      f = utils.xOR(BxorC, d);
      k = '11001010011000101100000111010110';
    }
    const word = words80[i][j];
    const tempA = utils.binaryAddition(utils.leftRotate(a, 5), f);
    const tempB = utils.binaryAddition(tempA, e);
    const tempC = utils.binaryAddition(tempB, k);
    let temp = utils.binaryAddition(tempC, word);

    temp = utils.truncate(temp, 32);
    e = d;
    d = c;
    c = utils.leftRotate(b, 30);
    b = a;
    a = temp;
  }
  h0 = utils.truncate(utils.binaryAddition(h0, a), 32);
  h1 = utils.truncate(utils.binaryAddition(h1, b), 32);
  h2 = utils.truncate(utils.binaryAddition(h2, c), 32);
  h3 = utils.truncate(utils.binaryAddition(h3, d), 32);
  h4 = utils.truncate(utils.binaryAddition(h4, e), 32);
}
```

h0 = 01100111010001010010001100000001
h1 = 11101111110011011010101110001001
h2 = 10011000101110101101110011111110
h3 = 00010000001100100101010001110110
h4 = 11000011110100101110000111110000

h0 = 10001111000011000000100001010101
h1 = 10010001010101100011001111100100
h2 = 10100111110111100001100101000110
h3 = 10001011001110000111010011001000
h4 = 10010000000111011111000001000011

14. convert each of the five resulting variables to hexadecimal
15. join them together and return it!

h0 = 10001111000011000000100001010101
h1 = 10010001010101100011001111100100
h2 = 10100111110111100001100101000110
h3 = 10001011001110000111010011001000
h4 = 10010000000111011111000001000011

```
return [h0, h1, h2, h3, h4]
       .map((string) => utils.binaryToHex(string))
       .join('');
```

h0 = 8f0c0855
h1 = 915633e4
h2 = a7de1946
h3 = 8b3874c8
h4 = 901df043

Your hash value!

8f0c0855915633e4a7de19468b3874c8901df043