

Question 2

```
let dividend = +prompt("Insert a dividend.");
let divisor = +prompt("Insert a divisor.");
if ((dividend % divisor) === 0) {
  alert(The quotient is ${dividend / divisor}.);
} else {
  alert(The quotient is ${Math.floor(dividend /
  divisor)} and the remainder is ${dividend %
  divisor}.);
}
```

Write a script that takes in a dividend and a divisor. With those, alert either the quotient by itself OR the quotient and the remainder if applicable.

Question 3

```
function cuts(input) {
  let cuts = 0;
  while (input > 1) {
    input /= 2;
    cuts++;
  }
  return cuts;
}
```

Write a function that takes in a number an integer and divides it in half until it becomes 1 or less.

Question 4

```
function listElements(a, n) {
  if (a.length > n) {
    let result = [];
    for (let i = 0; i < n; i++) {
      result.push(a[i]);
    }
    return result;
  } else {
    return a;
  }
}
```

Question 4 (cont)

```
}
```

Write a function that takes in array a and returns a new array consisting of the first n elements of the original array. If n is greater than a, then simply return a copy of array a.

Question 5

```
function compareArrays(a, b) {
  if (a.length !== b.length) {
    return false;
  }
  for (let i = 0; i < a.length; i++) {
    if (a[i] !== b[i]) {
      return false;
    }
  }
  return true;
}
```

Write a function that takes in two arrays, a and b, and returns whether or not the listed elements are equal. Assume that all values are primitive (and not Objects) and that the values must be in the same order. It should also output false if the parameters are not arrays.

Loops

```
"for" loop      for (begin; condition; step) {
                // ... loop body ... }
                }
```

```
"while" loop    while (condition) {
                //code
                //loop body (i++)
                }
```



Question 6

```
function capitalizeMiddleCharacter(inputString) {
  let middleLetterPosition =
    Math.ceil(inputString.length / 2);
  return inputString.substring(0, middleLetterPosition)
  +
  inputString.charAt(middleLetterPosition).toUpperCase()
  + inputString.substring(middleLetterPosition + 1,
    inputString.length);
}
```

Write a function which takes in a String and capitalizes the middle letter of the string. If the string is even, then capitalize the character to the right of the middle.

Question 7

```
function suffixes(inputWord) {
  let result = [""];
  let lastSuffix = "";
  for (let i = inputWord.length - 1; i >= 0; i--) {
    lastSuffix = inputWord.charAt(i) + lastSuffix;
    result.push(lastSuffix);
  }
  return result;
}
```

Write a function which takes in a word and returns an array containing successive suffixes of the word, starting with the last character.

Question 8

```
function student(name, birthday, email, ID, SSID,
  Major, Minor, enrolled, graduate) {
  this.name = name;
  this.birthday = birthday;
  this.email = email;
  this.ID = ID;
  this.SSID = SSID;
  this.Major = Major;
  this.Minor = Minor;
  this.enrolled = enrolled;
  this.graduate = graduate;
}
```

Write an Object constructor or Class of a human being. It would also be helpful to draw the Object diagram of a few of these.

Precedence

| Operator | Operator Use | Operator Associativity | Operator Precedence |
|--|---|------------------------|---------------------|
| () | Method/function call, grouping | Left to right | Highest - 1 |
| [] | Array access | Left to right | 1 |
| . | Object property access | Left to right | 1 |
| ++ | Increment | Right to left | 2 |
| -- | Decrement | Right to left | 2 |
| - | Negation | Right to left | 2 |
| ! | Logical NOT | Right to left | 2 |
| ~ | Bitwise NOT | Right to left | 2 |
| delete | Removes array value or object property | Right to left | 2 |
| new | Creates an object | Right to left | 2 |
| typeof | Returns data type | Right to left | 2 |
| void | Specifies no value to return | Right to left | 2 |
| / | Division | Left to right | 3 |
| * | Multiplication | Left to right | 3 |
| % | Modulus | Left to right | 3 |
| + | Plus | Left to right | 4 |
| + | String Concatenation | Left to right | 4 |
| - | Subtraction | Left to right | 4 |
| >> | Bitwise right-shift | Left to right | 5 |
| << | Bitwise left-shift | Left to right | 5 |
| >= | Greater than, greater than or equal to | Left to right | 6 |
| <= | Less than, less than or equal to | Left to right | 6 |
| === | Equality | Left to right | 7 |
| != | Inequality | Left to right | 7 |
| === | Identity operator — equal to (and same data type) | Left to right | 7 |
| !== | Non-identity operator — not equal to (or don't have the same data type) | Left to right | 7 |
| & | Bitwise AND | Left to right | 8 |
| ^ | Bitwise XOR | Left to right | 9 |
| | Bitwise OR | Left to right | 10 |
| && | Logical AND | Left to right | 11 |
| | Logical OR | Left to right | 12 |
| ? | Conditional branch | Left to right | 13 |
| = | Assignment | Right to left | 14 |
| *=, /=, %=, +=, -=, <<=, >>=, &=, ^=, = | Assignment according to the preceding operator | Right to left | 14 |
| , | Multiple evaluation | Left to right | Lowest: 15 |

