# Q1

1. a closure is a function that contains another function. they are often used to manipulate a variable without making that variable global

```
function counterHandler() {
  let x = 0;
  return function () {
    return x++;
  };
}

let handler = counterHandler(); // returns a Function object
console.log(handler()); // 0
console.log(handler()); // 1
console.log(handler()); // 2
console.log(handler()); // 3
```

2. they are functions passed as an argument to another function. for example a function that times another function:

```
function timeFunc(func) {
  let start = performance.now(); // Get current high-resolution time
(number)
  func(); // call the function
  return performance.now() - start; // find the time difference
}

console.log(
  timeFunc(() => {
    for (let i = 0; i < 10_000; i++);
  })
);
```

3. recursion is when a function calls itself under a certain condition and stops when it hits a base condition.

```
function countDown(n) {
  if (n == 0) {
    console.log(0);
    return;
  }
  console.log(n);
  return countDown(n - 1);
}
```

4. array destructuring is to bring the inner elements of the array into the outer scope. often used in assignments

```
let [a, b, c] = [1, 2, 3];
console.log(a, b, c); // 1 2 3

[a, ...b] = [1, 2, 3, 4, 5, 6];
console.log(a, b); // 1 [2,3,4,5,6]

[a, , b] = [1, 0, 2];
console.log(a, b); // 1 2

[a, , , ...b] = [1, 2, 3, 4, 5, 6, 7, 8, 9];
console.log(a, b); // 1 [4,5,6,7,8,9]

let arr = [1, 2, 3, 4];
a = { ...arr, 0: 56 };
console.log(a); // { '0': 56, '1': 2, '2': 3, '3': 4 }

let arr1 = [1, 2, 3];
let arr2 = [4, 5, 6];
let arr3 = [...arr1, ...arr2];
console.log(arr3); // [ 1, 2, 3, 4, 5, 6 ]
```

5. strict mode runs the code in a stricter environment by turning silent errors into exceptions and allow bugs to be spotted early

# Q2

1. 2 12
2. true false
3. John Doe
4. `[10, 10, 15]`
5. `[1,2,3,4] [5,4,3,2,1]`

# Q3

1.

```
<!DOCTYPE html>
<html lang="en">

<head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-
scale=1.0">
```

```
	<title>Document</title>
</head>

<body>

	<script>
		function confirmWild(name1, name2) {
			confirm("Are you wild?") ? alert(name1) : alert(name2)
		}

		if (confirm("Do you fly?")) {
			confirmWild('Eagle', "Parrot")
		}
		else {
			if (confirm("Do you live under the sea?")) {
				confirmWild('Shark', 'Dolphin')
			}
			else {
				confirmWild('Lion', 'Cat')
			}
		}
	</script>
</body>

</html>
```

2.

```
function objToArr(obj) {
	let arr = []
	for (let prop in obj) {
		let innerArr = [prop]
		if(typeof obj[prop] === 'object'){
			innerArr.push(objToArr(obj[prop]))
		}
		else{
			innerArr.push(obj[prop])
		}
		arr.push(innerArr)
	}
	return arr
}
```

3.

```
class Circle{
	constructor(radius){
		this.radius = radius
	}
```

```
        get area(){
                return Math.PI * this.radius * this.radius
        }
        get circumference(){
                return 2 * Math.PI * this.radius
        }
}
```

4.

```
function getBudgets(arrObj){
        let initial = 0
        return arrObj.reduce((initial, current) => initial + current.budget,
initial)
}
```

5.

```
function makePlusFunction(num1){
        return function(num2){
                return num1 + num2
        }
}
```

# Q4

1. Encapsulation is to prevent access to some data members within a class. Abstraction is to hide the implementations of functions and data members from the users when it is not necessary and only show what the class should do and not what it contains or how it works; Encapsulation is to lock a door, Abstraction is to see what is behind the door as one whole unit (a bedroom. Not a bed, a nightstand and a window)

2.

```
// Helper function
function reverseStr(s){
        let rev = ''
        for(let i = s.length - 1; i > -1; i--){
                rev += s[i]
        }
        return rev
}

// Main function
function specialReverse(letter, str){
```

```
        let splitStr = str.split(" ")
        for(let i = 0; i < splitStr.length; i++){
              if(splitStr[i][0] == letter.toLowerCase()){
                    splitStr[i] = reverseStr(splitStr[i])
              }
        }
        return splitStr.join(' ')
  }
```