Classic Programming Job Interview Challenges:

A. Fizz-Buzz

B. Find nth Fibonacci Number

C. Check for duplicates in array

A. Fizz-Buzz

Output to the console all integers from 1-100 along with "Fizz" and/or "Buzz" according to the following rules:

- If the number is evenly divisible by both 3 and 5, output the number followed by ' Fizz-Buzz'.
- If the number is evenly divisible by 3 (but not 5), output the number followed by ' Fizz'.
- If the number is evenly divisible by 5 (but not 3), output the number followed by ' Buzz'.
- If the number is divisible by neither 3 nor 5, output just the number.

One aim of the challenge is to see if the candidate figures out that the **modulo** operator is the key to the solution. Before tackling "Fizz-Buzz", let's try an example of modulo (%), which returns the remainder of the first number divided by the second number:

```
let remainder = 17 % 5;
console.log('remainder', remainder); // 2
```

And now for "Fizz-Buzz"...

1. Set up a for loop that goes from 1-100:

```
for(let i = 1; i <= 100; i++) {
}
```

Another aim of the challenge is to see if the candidate can figure out to start by checking if the number is divisible by both and and 5, as opposed to starting by seeing if it's divisible by one or the other. This test requires the **&&** (AND) operator.

2. Use the **&&** operator to see if the current number yields a remainder of 0 when divided by 3 and 5:

```
for(let i = 1; i <= 100; i++) {
    if(i % 3 == 0 && i % 5 == 0) {
        console.log(i + ' Fizz BUzz');
    }
}
```

3. If the number is not divisible by both 3 and 5, check if it's divisible by 3 only. If it is, log the number followed by 'Fizz':

```
for(let i = 1; i <= 100; i++) {
    if(i % 3 == 0 && i % 5 == 0) {
        console.log(i + ' Fizz BUzz');
    } else if(i % 3 == 0) {
        console.log(i + ' Fizz');
    }
}
```

4. Add another *else if* to see if the number is divisible by just 5. If it is, log the number followed by 'Buzz':

```
for(let i = 1; i <= 100; i++) {
    if(i % 3 == 0 && i % 5 == 0) {
        console.log(i + ' Fizz BUzz');
    } else if(i % 3 == 0) {
        console.log(i + ' Fizz');
    } else if(i % 5 == 0) {
        console.log(i + ' Buzz');
    }
}
```

5. Finally, if the number is divisible by neither 3 nor 5, log just the number:

```
for(let i = 1; i <= 100; i++) {
    if(i % 3 == 0 && i % 5 == 0) {
        console.log(i + ' Fizz BUzz');
    } else if(i % 3 == 0) {
        console.log(i + ' Fizz');
    } else if(i % 5 == 0) {
        console.log(i + ' Buzz');
    } else {
        console.log(i);
    }
}
```

B. Find nth Fibonacci number

Another popular coding challenge is to produce a sequence of Fibonacci numbers. Part of the point is to see if the candidate even knows what a Fibonacci sequence is. The wording can vary, but is typically something like:

"Find the 20th number in the Fibonacci sequence."

In a Fibonacci sequence, each number is the sum of the previous two. A starter array **[0, 1]** is usually given. The sequence continues as: **[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, etc.]**.

In programming challenges, it is often required that solutions be implemented as a function, with any givens provided as arguments. Our function will be called **makeFibos** and will take two arguments: the starter array and **n**, the target, which is this case is 20.

The **makeFibos(arr, n)** function will do the following:

- iterate the array, which to start off is just **[0, 1]**; each iteration adds a new item to keep the loop going
- our loop runs **n-2** (18) times, because we have a 2-item head start
- each time the loop runs, get the current item: **fibos[i]**
- we also need the next item: **fibos[i+1]**, which is also the last item
- add the numbers: **fibos[i] + fibos[i+1]**. The sum is the next number in the sequence.
- **push()** the sum into the array. The array just got longer by 1, so the loop can run one more time. This repeats until the loop has run **n-2** (18) times.
- when the loop is over, log the last item of the 20-item array, which is the item at index **n-1** (19).

6. Start by declaring an array, **fibosArr**, containing the two starter values. Also declare a number variable to store the target, which is 20:

```
const fibosArr = [0, 1];
let n = 20;
```

In the interview challenge, the givens will not likely be provided as variables; you have to declare them, as we just did.

7. Set up the function with parameters **arr** and **n**:

```
function makeFibos(fibos, n) {
}
```

8. Set up a loop that runs **n-2** times:

```
function makeFibos(fibos, n) {
    for(let i = 0; i < n-2; i++) {
    }
}
```

9. Add the current and next item, which are the last two numbers: **fibos[i] + fibos[i+1]**. Push the sum into the array:

```
function makeFibos(fibos, n) {
    for(let i = 0; i < n-2; i++) {
        let nextFibo = fibos[i] + fibos[i+1];
        fibos.push(nextFibo);
    }
}
```

10. Return the answer. Console log the full array inside the function as a test, and return the last item, which is the nth (20th) item:

```
function makeFibos(fibos, n) {
    for(let i = 0; i < n-2; i++) {
        let nextFibo = fibos[i] + fibos[i+1];
        fibos.push(nextFibo);
    }
    console.log(fibos);
    return fibos[fibos.length-1];
}
```

11. Call the function, saving the call to a variable which captures the return value:

```
let fibo20th = makeFibos(fibosArr, n);
```

12. Log the return value, which is the 20th fibo:

```
console.log(fibo20th);
```

One refinement: to make the loop more dynamic, we should not hard-code **n-2** (18). After all, what if the provided starter array is **[0, 1, 1, 2]**? Then we would need to run the loop **n-4** (16) times to get to the 20th item. The answer is to save **n-arr.length** to a variable above the loop.

13. Above the loop, save the target **n** minus the length of the array argument to a variable, **loopTimes**. This lets the starter array can have more than two items:

```
function makeFibos(fibos, n) {
    let loopTimes = n-fibos.length;
    for(let i = 0; i < loopTimes; i++) {
        let nextFibo = fibos[i] + fibos[i+1];
        fibos.push(nextFibo);
    }
```

```
        console.log(fibos);
        return fibos[fibos.length-1];
    }
```

14. Make a new starter array with a few more "fibos" in it:

```
    const fibos = [0, 1];
    let n = 20;
```

15. Call the function, passing in the five-item starter array, saving the return value to a new variable:

```
    let twentiethFibo = makeFibos(fiveFibos, n);
    console.log(twentiethFibo);
```

C. Find duplicate items in an array

"Given an array of numbers or strings, find all duplicates and return it/them as a new array containing one instance of each duplicate. If there are no duplicates, return 'none'"

The solution function takes an array as its argument. An example array may be given, but make sure your function works on any array of numbers or strings.

16. Assuming **nums1** is given, declare three more arrays, to give us more testing scenarios:

```
    const nums1 = [2, 4, 5, 7, 8, 9, 4, 11, 6, 8, 7, 9, 11, 6, 7];
    const nums2 = [2, 4, 5, 7, 1, 9, 11, 3, 6, 8, 10];
    const fruits1 = ['apple', 'banana', 'apple', 'cherry', 'kiwi',
'banana', 'mango', 'pear', 'peach', 'kiwi', 'grape'];
    const fruits2 = ['apple', 'banana', 'cherry', 'mango'];
```

17. Define a function that takes an array as its argument:

```
    function findDuplicates(arr) {
    }
```

18. Iterate the array:

```
    function findDuplicates(arr) {
        for(let i = 0; i < arr.length; i++) {
        }
    }
```

The key to the solution is to keep track of which values have come up so far. We can do this by saving each unique item as a key of an object, which we declare inside the function.

- check if the current array item is already an object key.
- if the current item is already a key, it is a duplicate.
- if the current item is not yet a key, make a key for it. The syntax uses square brackets for both array index and dynamic object property accessor.

19. Define an object above the loop. Inside the loop, check if the current array item is **not !** a key. If it is not, make a key for that item:

```
function findDuplicates(arr) {
    let obj = {};
    for(let i = 0; i < arr.length; i++) {
        if(!obj[arr[i]]) {
            obj[arr[i]] = arr[i];
        }
    }
}
```

20. But if the current array item, **arr[i]** is already a key, we have a duplicate item, so push it into the array of duplicates. Also, declare a new empty array, **dups**, above the loop for storing the duplicates:

```
function findDuplicates(arr) {
    let obj = {};
    let dups = [];
    for(let i = 0; i < arr.length; i++) {
        if(!obj[arr[i]]) {
            obj[arr[i]] = arr[i];
        } else {
            dups.push(arr[i]);
        }
    }
}
```

avoid looking up array items again and again

Here's a refinement that interviewers will be sure to appreciate. The function has to repeatedly look up the value of the current array item (**arr[i]**). More memory-efficient would be to save **arr[i]** to a variable and just refer to the variable. This also makes the code cleaner by avoiding nested square brackets: **obj[arr[i]]**.

21. Above the if-else, save arr[i] to a variable::

```
function findDuplicates(arr) {
    let obj = {};
    let dups = [];
    for(let i = 0; i < arr.length; i++) {
```

```
            let item = arr[i];
            if(!obj[item]) {
                obj[item] = item;
            } else {
                dups.push(item);
            }
        }
    }
```

22. After the loop ends, check if the **dups** array has any items it it. If it doesn't, return the array. Else, return "none":

```
    function findDuplicates(arr) {
        let obj = {};
        let dups = [];
        for(let i = 0; i < arr.length; i++) {
            if(!obj[item]) {
                obj[item] = item;
            } else {
                dups.push(item);
            }
        }
        if(dups.length > 0) {
            return dups;
        } else {
            return "none";
        }
    }
```

23. Call the function four times, once per array. Save the function call to a variable to store the return value, and log the variable:

```
    let n1 = findDuplicates(nums1);
    console.log('duplicates:', n1);

    let n2 = findDuplicates(nums2);
    console.log('duplicates:', n2);

    let f1 = findDuplicates(fruits1);
    console.log('duplicates:', f1);

    let f2 = findDuplicates(fruits2);
    console.log('duplicates:', f2);
```

24. We are getting duplicates of duplicates, such as: ['apple', 'kiwi', 'banana', 'kiwi']). Let's add logic so that the duplicate gets pushed into the array only if it is *not* already there:

```
    } else {
        if(!dups.includes(item)) {
            dups.push(item);
        }
    }
```

- **END: Lesson 05.06**
- **NEXT: Lesson 05.07**