Data Structures & Unit Testing with Jest

Agenda

Data Structures

- Navigating nested arrays and objects
- Array methods (**forEach, map, filter, find, reduce**)

Unit Testing

- Why write tests?
- Jest
- Structure of a test (Arrange, Act, Assert)
- Jest matchers

Array Methods - `.forEach()`

- Accepts a function to be called with each item in the array.
- Returns undefined.

```
const  numbers = [1, 2, 3, 4]

numbers.forEach(num => {
   console.log(num)
})

// 1
// 2
// 3
// 4
```

Array Methods - `.map()`

- Accepts a function to be called with each item in the array
- Returns a new array of the same length
- Values are based on the result of calling the provided function on each item

```
const  numbers = [1, 2, 3, 4]

const numberObjects = numbers.map(num => {
   return { number: num * 2 }
})

console.log(numberObjects)
// [{ number: 2 }, { number: 4 }, { number: 6 }, { number: 8 }]
```

Array Methods - `.filter()`

- Accepts a function to be called with each item in the array
- This function must return a boolean
- Returns a new array of either shorter or the same length
- Values are those for which the provided function returns true

```
const  numbers = [14, 28, 3, 10, 9]

const  lessThanTen = numbers.filter(num =>  num < 10)

console.log(lessThanTen)
// [3, 9]
```

Array Methods - `.find()`

- Accepts a function to be called with each item in the array
- This function must return a boolean
- Returns the value of the FIRST item for which the provided function returns true
- If no item returns true, undefined is returned.

```
const  numbers = [1, 2, 3, 5, 7, 11 ]

const  middleNumber = numbers.find(num =>  {
  return num > 2 && num < 7
})

console.log(middleNumber)
// 3
```

Array Methods - `.reduce()`

- Accepts a function to be called with each item in the array
- This function should return an accumulating value based on the current item
- Returns a single item of any type
- The most flexible array method - you could create any of the previous methods with .reduce()

```
const  numbers = [1, 2, 3, 4 ]
const initialValue = 5

const  sum = numbers.reduce((accumulator, currentNum)  =>  {
  return accumulator + currentNum
}), initialValue)

console.log(sum)
//15
```

Why write tests?

- Critical component of high quality code
- Give yourself confidence to make changes to your codebase
- Automation! Especially for deployment
- Confirm and improve your understanding of the code you are writing - a great learning tool.

Jest & Structuring a Test

```
test('getSum returns the correct sum', ( ) => {
  // ARRANGE (setup)
  const numbers = [1, 6, 8, 4, 2, 9]
  // ACT (run the code being tested)
  const sum = getSum(numbers)
  // ASSERT (verify it did the right thing)
  expect(sum).toBe(30)
})
```

Once the test is passing, break it to ensure you aren't getting a false positive
More Jest matchers can be found in the docs: https://jestjs.io/docs/expect

Takeaways

- TOP PRIORITY MISSION: being comfortable working with objects and arrays (including array methods)
- Write tests!
- Arrange, Act, Assert
- Clarity and staying isolated is more important than DRY code in tests
- Stay skeptical - break your tests in an expected way to check for false positives