Lesson 04.03 - PROG

Array Methods

REVIEW:

push(), pop(), sort(), reverse(), flat()

NEW:

unshift(), shift(), concat(),

indexOf(), lastIndexOf(),

splice(), slice(), includes(), join()

NEW for 2023:

toShifted(), toSorted(), toReversed()

---

array.push(item)

push() adds the item argument(s) to the end of the array.

1. Declare an array with a few items, and use push() to add items to the end.
   The method changes the existing array--it does NOT return a new array.

```
const fruits = ['blueberry', 'cherry', 'banana'];
fruits.push('kiwi');
fruits.push('orange', 'grape', 'lime');
console.log(fruits);
// ['blueberry', 'cherry', 'banana', 'kiwi', 'orange', 'grape', 'lime']
```

array.pop()

pop() removes the last item from the end of the array, and returns it

2. Remove the last item using pop().
   The method returns the popped item, so save that to a variable:

```
let popped = fruits.pop();
console.log('popped:', popped); // lime
console.log(fruits);
// ['blueberry', 'cherry', 'banana', 'kiwi', 'orange', 'grape']
```

array.sort()

sort() is called on an array of strings and puts them in alphabetical order. The method changes the existing array--it does NOT return a new array.

3. Arrange the items in alphabetical order with sort().
   The method changes the existing array--it does not make a new one:

```
fruits.sort();
console.log('sorted:', fruits);
// ['banana', 'blueberry', 'cherry','grape'  'kiwi', 'orange']
```

array.reverse()

reverse() is called on an array and reverses the order.

4. Reverse the order of the array items. The method changes the existing array--it does NOT return a new array. sorted from A-Z, the result will be items from Z-A:

```
fruits.reverse();
console.log('reversed:', fruits);
// ['orange', 'kiwi', 'grape', 'cherry', 'blueberry', 'banana']
```

array.unshift()

array.shift()

- **unshift()** adds an item to the beginning of an array.
- **shift()** removes the first item and returns it. How to remember which is which:
  - "unshift" is the longer word; it makes the array longer
  - "shift" is the shorter word; it makes the array shorter

5. Use **unshift()** to add an item to the beginning of the fruits array:

```
fruits.unshift('strawberry');
console.log(fruits);
// ['strawberry', 'orange', 'kiwi', 'grape', 'cherry', 'blueberry',
'banana']
```

6. Use **shift()** to remove and return the first item, saving that to a variable:

```
let shifted = fruits.shift(); // strawberry
console.log('shifted:', shifted);
```

```
console.log(fruits);
// ['orange', 'kiwi', 'grape', 'cherry', 'blueberry', 'banana']
```

array1.concat(array2, array3)

- **concat()** concatentates (combines) two or more arrays into one.
- Call concat() on an array, and pass other array(s) in as argument(s).

7. Declare three arrays of fruits:

```
const tropicalFruits = ['mango', 'kiwi', 'banana', 'pineappple',
'papaya'];
console.log('\ntropicalFruits:\n', tropicalFruits);

const citrusFruits = ['grapefruit', 'lemon', 'lime', 'tangerine',
'orange'];
console.log('\ncitrusFruits:\n', citrusFruits);

const blossomFruits = ['apple', 'peach', 'cherry', 'plum', 'pear'];
console.log('\n\nblossomFruits:\n', blossomFruits);
```

8. Use the concat() method to concatenate the three arrays into one:

```
const fruitCocktail = tropicalFruits.concat(citrusFruits, blossomFruits);
console.log('\nfruitCocktail:\n', fruitCocktail);
// ['mango', 'kiwi', 'banana', 'pineappple', 'papaya', 'grapefruit',
'lemon',
// 'lime', 'tangerine', 'orange', 'apple', 'peach', 'cherry', 'plum',
'pear']
```

9. Arrange the fruitCocktail array in alphabetical order:

```
fruitCocktail.sort();
console.log('\nfruitCocktail sorted:', fruitCocktail);
// ['apple', 'banana', 'cherry', 'grapefruit', 'kiwi', 'lemon', 'lime',
'mango',
// 'orange', 'papaya', 'peach',  'pear', 'pineappple', 'plum',
'tangerine']
```

array.splice()

- **splice()** removes items at a specified index or range of indices.
- spliced items may optionally be replaced with new items
- splice() requires two arguments: start_index, and number_of_items to remove.
- splice() returns the removed item(s) as a new array.

- splice() optionally takes additional arguments of replacement item(s).

10. Remove the item at index 7, which is 'mango':

```
let splicedIndex7 = fruitCocktail.splice(7,1); // remove item at index 7
console.log('splicedIndex7:', splicedIndex7); // ['mango']
```

Splicing one item returns array of 1 item -- BUT what if you just want the item itself, without array wrapper

10B. Comment out the 'mango' splice, and do it again, but this time grabbing item append [0] to the end of the expression:

```
let splicedIndex7 = fruitCocktail.splice(7,1)[0]; // remove item at index
7
// console.log('splicedIndex7:', splicedIndex7); // mango
```

10C: use pop() or shift() instead of [0] you can also pop() or shift() the one item from the array:

```
let splicedIndex7 = fruitCocktail.splice(7,1).pop();
console.log('splicedIndex7:', splicedIndex7); // mango
```

11. Starting at index 2, splice out four consecutive items, saving the result to a new array:

```
let spliced4 = fruitCocktail.splice(2,4);
console.log('spliced4:', spliced4); // ['cherry', 'grapefruit', 'kiwi',
'lemon']
console.log('\sfruitCocktail post-splicing:', fruitCocktail);
// ['apple', 'banana', 'lime', 'orange', 'papaya', 'peach',  'pear',
// 'pineappple', 'plum', 'tangerine']
```

To swap item(s) with splice, pass in new item(s) as additional argument(s)

12. Swap 'lime' at index 2 for 'lemon':

```
fruitCocktail.splice(2,1, 'lemon');
console.log('\nfruitCocktail post-lemon-lime-swap:', fruitCocktail);
// ['apple', 'banana', 'lemon', 'orange', 'papaya', 'peach',  'pear',
// 'pineappple', 'plum', 'tangerine']
```

swapping more than one item with splice()
Just add more arguments

13. Replace 'apple' and 'banana' with 'apricot' and 'blueberry':

```
fruitCocktail.splice(0,2, 'apricot', 'blueberrry');
console.log('\nfruitCocktail post-more-swaps:', fruitCocktail);
// ['apricot', 'blueberry', 'lemon', 'orange', 'papaya', 'peach',
// 'pear', 'pineappple', 'plum', 'tangerine']
```

adding item(s) at index, without removing any items

To add one or more items without removing any, specify the start index, and put 0 as the second argument to splice out 0 items.

14. Insert 'grape' and 'grapefruit' in its correct A-Z position, // without removing any items:

```
fruitCocktail.splice(2,0, 'grape', 'grapefruit');
console.log('\nfruitCocktail post-grape:', fruitCocktail);
// ['apricot', 'blueberry', 'grape', 'grapefruit', 'lemon', 'orange',
// 'papaya', 'peach',  'pear', 'pineappple', 'plum', 'tangerine']
```

splicing items at random

If we choose items, again and again, at random, from an array, we will sooner or later get repeats. To avoid repeats, splice out each item as it is chosen.

15. Pick 3 random fruits, one at a time, from fruitCocktail. Keep refreshing the console until you get repeats:

```
let r = ~~Math.floor(Math.random() * fruitCocktail.length);
console.log(`1st random fruit at index ${r}:`, fruitCocktail[r]);

r = ~~(Math.random() * fruitCocktail.length);
console.log(`2nd random fruit at index ${r}:`, fruitCocktail[r]);

r = ~~(Math.random() * fruitCocktail.length);
console.log(`3rd random fruit at index ${r}:`, fruitCocktail[r]);
```

16. **CHALLENGE:** choose 3 random fruits without getting any repeats HINT: splice()

```
r = ~~Math.floor(Math.random() * fruitCocktail.length);
console.log(`1st random fruit at index ${r}:`, fruitCocktail[r]);
// splice out the random fruit so that it cannot be reused:
fruitCocktail.splice(r,1);
console.log(`fruitCocktail`, fruitCocktail);

r = ~~(Math.random() * fruitCocktail.length);
console.log(`2nd random fruit at index ${r}:`, fruitCocktail[r]);
// splice out the random fruit so that it cannot be reused:
fruitCocktail.splice(r,1);
```

```
console.log(`spliced fruitCocktail:\n`, fruitCocktail);

r = ~~(Math.random() * fruitCocktail.length);
console.log(`3rd random fruit at index ${r}:`, fruitCocktail[r]);
// splice out the random fruit so that it cannot be reused:
fruitCocktail.splice(r,1);
console.log(`twice spliced fruitCocktail:\n`, fruitCocktail);
```

Now, no matter how many times you refresh the console, you get 3 unique fruits -- no repeats.

array.sort(callback)

Another way to get unique random items from array without repeats it to pass a callback to the sort() method

- shuffle (randomize) the array and then just pop pop removes items, which are already randomized

What is the sort() callback we already know?

HINT: it's for sorting array of nums in asc / desc order

17. Sort the numsVector array of numbers in ascending order:

```
const lotteryNums = [14,33,25,23,39];
[54,3,25,123,9,45,77,13].sort(function(a,b) {
    return a - b; // sorted ascending order
});
console.log('sorted lotteryNums:', lotteryNums);
```

Math.random() - 0.5

- half the time, the return value will be positive;
- the other half the time, it will be negative.
- the result will be a randomized array:

18. Call sort on the array, passing in a callback that returns

```
fruitCocktail.sort(function() {
    return Math.random() - 0.5;
});

console.log('scrambled fruitCocktail:', fruitCocktail);
console.log('shuffled and popped:', fruitCocktail.pop());
console.log('shuffled and popped:', fruitCocktail.pop());
console.log('shuffled and shifted:', fruitCocktail.shift());
console.log('shuffled and shifted:', fruitCocktail.shift());
```

19. **CHALLENGE:** write a function called **shuffle()** that takes an array as its argument and returns a randomized ("shuffled") version of same array

```
function shuffleArr(arr) {
  return arr.sort(function() {
    return Math.random() - 0.5;
  });
}

console.log('shuffle fruitCocktail:\n', shuffleArr(fruitCocktail));
```

20. **BONUS CHALLENGE:** write a similar function called **doubleAndShuffleArr()** that takes an array as its argument and returns a randomized ("shuffled") array of twice the length of the original with each item represented twice (2 apples, 2 cats, etc.)

```
function doubleAndShuffleArr(arr) {
  const doubles = arr.concat(arr);
  return doubles.sort(function() {
    return Math.random() - 0.5;
  });
}

console.log('double &shuffle fruitCocktail:\n',
doubleAndShuffleArr(fruitCocktail));
```

array.slice(start_index, end_index_exclusive)

- slice() is called on an array, and takes two arguments: start_index and end_index_exclusive (NOT included)
- items are copied (not cut), so original array is unchanged.

21. Make a 2-fruit tropical smoothie from the 3rd and 4th fruits of the tropicalFruits array:

```
console.log('tropicalFruits:', tropicalFruits);
// ['mango', 'kiwi', 'banana', 'pineappple', 'papaya']
const tropicalSmoothie = tropicalFruits.slice(2,4);
console.log('tropicalSmoothie:', tropicalSmoothie);
// 'banana', 'pineappple']
console.log('tropicalFruits:', tropicalFruits);
```

If the second argument is omitted, it slices from the start index all the way to the end:

22. Take a slice of tropicalFruits from the 3rd item to the to last:

```
const slicedToEnd = tropicalFruits.slice(2);
console.log('slicedToEnd:', slicedToEnd);
// ['banana', 'pineappple', 'papaya']
```

23. Get the pineapple as a one item array:

```
const oneItemArr = tropicalFruits.slice(3,4);
console.log('oneItemArr:', oneItemArr); // ['pineapple']
```

24. Do that again, but tack on [0] to return just the item at index 0, this being the only item in the array:

```
const oneItem = tropicalFruits.slice(3,4)[0];
console.log('oneItem:', oneItem); // pineapple
```

negative indexing

slice() allows negative indexing, with the last item at -1

25. Slice the last item:

```
let lastItem = tropicalFruits.slice(-1)[0];
console.log('lastItem neg 1 slice:', lastItem); // papaya
```

26. Slice the last 3 items

```
console.log('last 3 items:', tropicalFruits.slice(-3));
```

27. Slice all but the last item:

```
console.log('all but last item:', tropicalFruits.slice(0,-1));
```

28. Slice all but the first and last items:

```
console.log('all but the first and last items:',
tropicalFruits.slice(1,-1));
```

array.includes(item)

- **includes()** is called on an array or string
- returns true if the array or string contains the item argument
- returns false if the array or string does NOT contains the item

29. Call the includes() method twice, so that we get one true and one false:

```
console.log("tropicalFruits includes 'kiwi':",
tropicalFruits.includes('kiwi'));
console.log("tropicalFruits includes 'plum':",
tropicalFruits.includes('plum'));
```

30. Try includes on a string:

```
let vowels = 'aeiou';
console.log("vowels includes 'e':", vowels.includes('e')); // true
console.log("vowels includes 'f':", vowels.includes('f')); // false
```

CHALLENGE:

Write a function that takes in a word and uses string.includes()
to check if inputted word begins with a vowel

- if word begins AND ends with a vowel, such as 'orange',
    - return 'orange begins and ends with a vowel'
- if word begins with a vowel, such as 'apple':
    - return 'apple begins with a vowel'
- if word does not begin wtih a vowel, such as 'banana':
    - return 'banana does not begin with a vowel'
- use str.toLowerCase() to check upper and lower case:
    - 'Apple begins with a vowel'
    - 'Banana does not begin with a vowel'
    - 'Orange begins and ends with a vowel'

```
function checkForVowel(word) {
    let VOWELS = "aeiou";
    if (VOWELS.includes(word[0]) && VOWELS.includes(word.slice(-1))) {
        return word + " starts and ends with a vowel";
    } else if (VOWELS.includes(word[0])) {
        return word + " starts with a vowel";
    } else {
        return word + " does not start with a vowel";
```

```
    }
}

console.log(checkForVowel('orange'));
console.log(checkForVowel('atlas'));
console.log(checkForVowel('bird'));
```

indexOf() and lastIndexOf()

- **indexOf()** is called on an array or string and returns the index of the first instance of the argument. If it is not found, it returns -1.
- **lastIndexOf()** is called on an array and returns the index of the last instance of the argument. If it is not found, it returns -1.

31. Look up the index of an item in the tropicalFruits array:

```
console.log("tropicalFruits index of 'banana':",
tropicalFruits.indexOf('banana')); // 2
```

32. Look up the index of an item that is NOT in the tropicalFruits array:

```
console.log("tropicalFruits index of 'apple':",
tropicalFruits.indexOf('apple')); // -1
```

Given this array of pets, which includes three cats:

```
let pets = ['bunny', 'tarantula', 'gerbil', 'cat', 'dog',
            'goldfish', 'cat', 'iguana', 'cat', 'hamster'];
```

33. Get the index of the first cat:

```
console.log("index of 1st 'cat':", pets.indexOf('cat')); // 3
```

34. Get the index of the last cat:

```
console.log("index of last 'cat':", pets.lastIndexOf('cat')); // 8
```

using indexOf() to get the second item

But what about the other cat -- that second cat. How can we find its index? We need to use indexOf, but we also need to start looking after the first cat. To do ths, provide a second argument: start_index The starting point is one after the first cat, which we get dynamically as **pets.indexOf('cat')+1**

35. Get the second cat, using indexOf with a second argument: start_index:

```
console.log("index of 2nd cat:", pets.indexOf('cat',
pets.indexOf('cat')+1)); // 6
```

join()

- join() method is called on an array and returns a string
- join() returns a string made from joined together array items
- join() takes an argument called the 'delimiter'
- join() with no delimiter, returns a string of comma-separated items: ['bunny','cat','dog'].join() --> 'bunny,cat,dog'
- join('') with an empty string delimiter returns a string with no spaces: ['bunny','cat','dog'].join('') --> 'bunnycatdog'
- join(' ') with space delimiter returns a string of individual words: ['bunny','cat','dog'].join(' ') --> 'bunny cat dog' */

36. Use join() to make a string from citrusFruits. For this first try, don't provide a delimiter:

```
let fruitStr1 = citrusFruits.join();
console.log("\nfruitStr1 from join():\n", fruitStr1);
// grapefruit,lemon,lime,tangerine,orange
```

37. Repeat, but with an empty string delimiter:

```
let fruitStr2 = citrusFruits.join('');
console.log("\nfruitStr2 from join(''):\n", fruitStr2);
// grapefruitlemonlimetangerineorange
```

38. Again, but with an space delimiter argument:

```
let fruitStr3 = citrusFruits.join(' ');
console.log("\nfruitStr3 from join(' '):\n", fruitStr3);
// grapefruit lemon lime tangerine orange
```

39. And again, but with '-' delimiter argument:

```
let fruitStr5 = citrusFruits.join('-');
console.log("\nfruitStr5 from join('-'):\n", fruitStr5);
// grapefruit-lemon-lime-tangerine-orange
```

40. Once more, but with ' * ' delimiter argument:

```
let fruitStr4 = citrusFruits.join(' * ');
console.log("\nfruitStr4 from join(' * '):\n", fruitStr4);
// grapefruit * lemon * lime * tangerine * orange
```

41. **CHALLENGE:** Given this array of strings, make strings:

```
const baseballArr = ['Mets', 'Win', 'World', 'Series'];
```

- make this headline: **Mets Win World Series**
- make this file name: **mets-win-world-series.jpg**

str.toLowerCase() and str.toUpperCase()

str.toLowerCase() is called on a string and returns a lowercase string: 'Mets'.toLowerCase() returns 'mets' the original string is unchanged

str.toUpperCase() is called on a string and returns string in ALL caps; the original string is NOT changed */

42. Declare a string and make it all lowercase:

```
let str = "World Series";
console.log('str.toLowerCase():', str.toLowerCase()); // world series
console.log('str:', str); // World Series
// to make the change "stick" save var back to itself (or to some other
var)
str = str.toLowerCase();
console.log('str:', str);
```

43. From baseballArr, make the headline: 'Mets Win World Series'

```
let headline = baseballArr.join(' ');
console.log("headline:", headline); // 'Mets Win World Series'
```

44. From baseballArr, the file name mets-win-world-series.jpg:

```
let filename = baseballArr.join("-").toLowerCase() + ".jpg";
console.log('filename:', filename); // mets-win-world-series.jpg
```

array.flat()

The flat() method is called on a matrix (2D) array and returns a new, flat, 1D array

45. Declare a 3x3 (2D) matrix array:

```
const ticTacToe = [
    ['X', 'O', null],
    [null, 'X', 'O'],
    ['O', null, 'X']
];
console.log('ticTacToe: =', ticTacToe); // X wins!
```

46. Flatten the array:

```
const flatTacToe = ticTacToe.flat();
console.log('flatTacToe: =', flatTacToe);
// ['X', 'O', null, null, 'X', 'O', 'O', null, 'X']
// X still wins, but it's hard to tell
```

47. Given these 12 numbers:
    189,28,33,43,11,22,32,42,161,322,392,402
    Make a matrix array of 3 arrays of 4 items each:

```
const numsMatrix = [ [189,28,33,43],
                     [11,22,32,42],
                    [161,322,392,402] ];
console.log('numsMatrix:', numsMatrix);
```

48. Using double square brackets, get the 32 from numsMatrix:

```
console.log(numsMatrix[1][2]); // 32
```

49. Flatten numsMatrix into a 1D vector (line):

```
const numsFlat = numsMatrix.flat();
console.log('flatnumsFlattenedNums:', numsFlat);
```

50. Get the 32 from numsFlat (only one [] required):

```
console.log(numsFlat[6]); // 32
```

Shallow Copy vs. Deep Copy

Primitives (strings, numbers) can be copied by assignment (=).
if you change the copy, the original does NOT also change This kind of "independent" copy is known as a **Deep Copy**

51. Declare a string and then copy it by direct assignment:

```
let originalStr = 'apple';
let deepCopy = originalStr;
```

52. Make a change to the copy:

```
deepCopy = 'pineapple';
```

53. Log both; the original is unchanged, because we made a deep copy:

```
console.log('originalStr:', originalStr, 'deepCopy:', deepCopy);
// originalStr: apple, deepCopy: pineapple
```

Objects, on the other hand, copied by assignment result in a **Shallow Copy**, that is a copy that is still "connected to" the original object. Change the copy and the original changes too.

... use case: Making a Deep Copy of an Array or Object

that is, changes to new arr affect original, as well

54. Copy an array by direct assignment:

```
const citrusArr = citrusFruits;
console.log('...citrusFruits:\n', ...citrusFruits);
```

55. Make a change to the copy:

```
citrusArr.push('pomelo');
console.log('citrusArr:\n', citrusArr);
// ['grapefruit', 'lemon', 'lime', 'tangerine', 'orange', 'pomelo']
```

56. Check the original. It also picked up the pomelo.
    This demonstrates that citrusArr is a Shallow Copy:

```
console.log('citrusFruits:\n', citrusFruits);
```

... Destructuring Operator: Dot-Dot-Dot

...array destructures the array, that is "strips" the []
while keeping the actual array items intact a string made from another string by assignment is a Deep Copy

57. Log citrusFruits; then log the destructured version, which is all the items but not as an array:

```
console.log('citrusFruits:', citrusFruits);
// ['grapefruit', 'lemon', 'lime', 'tangerine', 'orange']
console.log('...citrusFruits:\n', ...citrusFruits);
// grapefruit lemon lime tangerine orange
```

58. Using ... make a Deep Copy of an array:

```
const freshCitrus = [...citrusFruits];
```

- ...arr strips the array of its [], so it's not an array anymore -- just loose items
- [...arr] "rebundles" items into a fresh array, which is assigned to the new const
- the result is a Deep Copy;

59. Modify new array and then check if original array also changed:

```
freshCitrus.push('yuzu', 'tangelos');
console.log('freshCitrus:\n', freshCitrus);
// ['grapefruit', 'lemon', 'lime', 'tangerine', 'orange', 'pomelo',
'yuzu', 'tangelos']
console.log('citrusFruits:\n', citrusFruits);
// ['grapefruit', 'lemon', 'lime', 'tangerine', 'orange', 'pomelo']
```

The original did NOT pick up 'yuzu', 'tangelos' because freshCitrus is a Deep Copy

... instead of concat()

Let's say you have a fruit card game which requires a deck with 4 of each fruit

60. Use ... to make your deck of four sets of the same array:

```
const fruitCardDeck = [...freshCitrus, ...freshCitrus, ...freshCitrus,
...freshCitrus];
console.log('fruitCardDeck:\n', fruitCardDeck.length, fruitCardDeck);
```

61. Shuffle (randomize) the fruit card deck:

```
fruitCardDeck.sort(function() {
    return Math.random() - 0.5; // half neg, half pos
});
```

... destructuring objects

same Deep Copy vs. Shallow Copy issue

Given this object:

```
const pet = {
    type: 'rabbit',
    petName: 'Bunny',
    age: 3,
    foods: ['blueberry', 'carrot', 'kale', 'parsley', 'peach'],
}
```

62. Make a shallow copy of pet, and modify the copy:

```
const shallowCopyPet = pet; // Shallow Copy
shallowCopyPet.foods.unshift('apple');
shallowCopyPet.age = 4;
console.log('shallowCopyPet:', shallowCopyPet);
console.log('pet:', pet);
// original pet obj picked up the changes to Shallow Copy
```

63. Make a deep copy of pet, and modify the copy:

```
const deepCopyPet = { ...pet }; // Deep Dopy
deepCopyPet.foods = [...pet.foods]; // separatedly destructure the obj arr
deepCopyPet.foods.push('strawberry');
deepCopyPet.age = 5;
console.log('deepCopyPet:', deepCopyPet);
console.log('pet:', pet);
// original pet obj picked up the changes to Shallow Copy
```

other use cases for ... destructuring

Some methods for finding values in an array only work if the array is flat.

Math.min() & Math.max()

64. Pass some numbers to Math.min() and check the result:

```
let minNum = Math.min(12,5,7,9,4,23,6,10);
console.log('minNum:', minNum); // 4
```

65. Pass some numbers to Math.max() and check the result:

```
let maxNum = Math.max(12,5,7,9,4,23,6,10);
console.log('maxNum:', maxNum); // 23
```

Passing array of nums to Math.min() or Math.max() returns NaN, since the methods cannot read nums wrapped in an array

66. Pass the destructured array to Math.min() and Math.max():

```
const nums = [12,5,7,9,4,23,6,10];
minNum = Math.min(...nums);
console.log('... minNum:', minNum);

maxNum = Math.max(...nums);
console.log('... maxNum:', maxNum);
```

67. Math.min and Math.max cannot handle nested arrays. They only work on flat arrays.
    Given a 3x3 array, flatten it and get the max value
    using Math.max() and the spread operator ... :

```
let nums2D = [[5,6,7], [15,16,17], [2,9,8] ];
let flatNums = nums2D.flat();
console.log('max nums2D:', Math.max(nums2D)); // NaN
console.log('max flatNums:', Math.max(flatNums)); // 17
```

68. Given this nested array of animals, check if there ia a panda in it:

```
const animals2D = [ ['giraffe', 'zebra', 'elephant'], ['panda', 'koala',
'kangaroo'] ];
console.log('animals2D.includes("panda"):', animals2D.includes('panda'));
// false
```

69. Flatten the animals array and check for the panda again:

```
const flatAnimals = animals2D.flat();
console.log('flatAnimals.includes("panda"):',
flatAnimals.includes('panda')); // true
```

indexOf()

returns index of first instance of argument; if not found, returns -1

70. Use **indexOf()** to find the index of the panda in the nested array:

```
console.log('nestedAnimals.indexOf("panda")',
nestedAnimals.indexOf('panda')); // -1
// The nestedAnimals returns false for includes() and -1 for indexOf(),
```

71. Use **indexOf()** to find the index of the panda in the flat array.

```
console.log('flatAnimals.indexOf("panda")', flatAnimals.indexOf('panda'));
// 3
// This time it works
```

NEW Array Methods for 2023

- There are a few new versions of array methods that modify the existing array
- These new versions return a new array, while leaving the original unchanged.
- These array methods all modify the original array:
    - **splice()**
    - **sort()**
    - **reverse()**
- These new array methods do not modify the original array; instead, they return a new array:
    - **toSpliced()**
    - **toSorted()**
    - **toReversed()**

72. Make a fresh fruit cocktail array. You can use concat() or ...

```
const freshFruits = [...tropicalFruits, ...citrusFruits,
...blossomFruits];
console.log('freshFruits:\n', freshFruits);
```

```
// ['mango', 'kiwi', 'banana', 'pineappple', 'papaya', 'grapefruit',
'lemon', 'lime',
// 'tangerine', 'orange', 'pomelo', 'apple', 'peach', 'cherry', 'plum',
'pear']
```

73. Use **toSpliced()** to add a new item at index to freshFruits;
    the result will be a new array; the original array will not change:

```
const coconutCocktail = freshFruits.toSpliced(3, 0, 'coconut');
console.log('toSpliced() coconutCocktail:\n', coconutCocktail);
console.log('unchanged freshFruits (no "coconut"):\n', freshFruits);
```

74. Starting at index 5, replace 3 fruits in a row with 2 berries;
    the result will be a new array; the original array will not change:

```
const berryGoodCocktail = freshFruits.toSpliced(5, 3, 'raspberry',
'strawberry');
console.log('toSpliced() berryGoodCocktail:\n', berryGoodCocktail);
console.log('unchanged freshFruits (no berries):\n', freshFruits);
```

75. Sort freshFruits without modifying it.
    Instead, save the sorted result to a new array:

```
const sortedFruits = freshFruits.toSorted();
console.log('toSorted() sortedFruits:\n', sortedFruits);
console.log('unsorted, unchanged freshFruits:\n', freshFruits);
```

76. Reverse the alphabetized array without modifying it.
    Instead, save the reverse-alphabetized result to a new array:

```
const reversedFruits = sortedFruits.toReversed();
console.log('toReversed() reversedFruits:\n', reversedFruits);
console.log('unchanged A-Z sortedFruits:\n', sortedFruits);
```

END: Lesson 04.03

NEXT: Lab 04.03

THEN: Lesson 04.04