# Private Prep Day One

## Introduction

### Goals for Private Prep

- Develop strategies for building confidence & problem solving skills
- Get more comfortable with JavaScript syntax and commonly used functions
- Prepare you for interviews at immersive coding programs *(like ours!)*

### Goals for tonight

- Learn problem-solving strategies
- Review/extend knowledge of JS:
  - JavaScript arrays & their methods
  - Different ways of looping

## Problem Solving

### Sort Sorted Lists

Given two already-sorted lists, return new list of all elements, sorted

But not using the *.sort()* method

> NOTE **Why can't we use the sort method?**
>
> In many cases, a fine way you could solve this problem would use *.sort()* — it's easy and built in to JavaScript. However, common coding challenges, especially those given to new programmers in interviews, often have some intentional limitations, so people can understand how you can solve problems without having to depend on some built-in features or additional libraries.
>
> In particular, while the understanding of this is outside of this course, sorting very large lists can be very slow, and get even slower as the lists get larger. More direct ways of solving this challenge may take more effort to figure out than *.sort()*, but can run much, much faster, especially with large lists.

### Process Overview

- Make sure you understand problem
- Write down a "test case"

- Try to make a *concrete representation*
- Draw something visual
- Write *pseudocode*
- Test your pseudocode
- *Only now*: write code

## Understand Problem

> *Given two already-sorted lists, return new list of all elements, sorted.*

Restate the problem clearly:

> *I have two array of words, each in alphabetical order.*
>
> *I should return a new array, with all words, in alphabetical order.*

## Make a Test Case

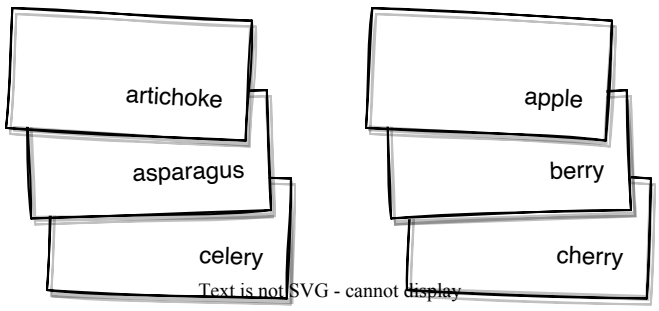
```
LIST A      LIST B                  OUTPUT

apple       artichoke                apple
berry       asparagus       →        artichoke
cherry      celery                   asparagus
                                     berry
                                     celery
                                     cherry
```

## Concrete Representation & Drawing

We're smart people & good at solving problems — thinking in code makes us dumber.

Find a way to conceptualize with real-world analogies




△ *Can imagine as two stacks of cards*

### Pseudo-Code

Explain the strategy, in English

> *Compare top cards, picking the earlier one alphabetically.*
>
> *Once one pile is empty, move remaining cards to end*

Or, perhaps:

> *Compare first items in both arrays, picking lower one.*
>
> *Once one array is empty, push remaining cards to end*

### Test Your Pseudo-Code

Follow your steps carefully to see if it works

*Now* you're ready to write some code!

## Working with JavaScript

The Chrome console is a great place to play with and test things.

But not a great place to write and edit stuff.

Better: write code in an editor and copy/paste to console!

### Variables

There are different ways to declare variables:

```
var name = "Vivien";

let age = 42;

const species = "homo sapien";
```

We won't worry about the differences tonight; you can use either *var* or *let*.

We'll talk about the differences later.

### Checking for Equality

Always use `===`, not `==`

```
"12" == 12;   // returns true (yeck!)

"12" === 12;  // returns false
```

For inequality, use `!==` not `!=`

# Arrays

## Creating New Arrays

```
let arr = [2, 4, 6, 8];
```

`arr.concat(elem, ...)`
: Return *new array* with elem at end; doesn't change *arr*

  Can provide multiple elems, like `arr.concat(10, 12, 14)`

## Common Array Operations

`arr[idx]`
: Return element at (0-based) index *idx*

  Never raises error — returns *undefined* if past array end

`arr.length`
: Return # items in array

## Finding Things

`arr.indexOf(elem)`
: Return index (0-based) of *elem* or `-1` if not found

`arr.includes(elem)`
: Returns true/false for whether *elem* is present

## Changing Arrays

`arr.push(elem, ...)`
: Add *elem* to end of array

  Can provide multiple elems, like `arr.push(10, 12, 14)`

`arr.pop()`
: Remove & return item at end of array

`arr.unshift(elem, ...)`
: Like push, but adds at beginning

`arr.shift()`
: Like pop, but removes from beginning

## Slicing and Splicing

`arr.slice(start, end)`
: Returns *new array* of items from *start* up to (not including) *end*

  Can leave off *end* to go through end of array

`arr.splice(start, deleteCount, item, ...)`
: Starting at *start*, delete *deleteCount* items (can be 0)

  Then, add *item*(s)

> **TIP** **Can use negative indexes**
>
> For both, can provide negative indexes ( `-1` is last, `-2` next-to-last)

## Checking Equality

Two different arrays **do not compare equal** because they hold same items:

```
let a = [1, 2, 3];
let b = [1, 2, 3];

a === b;    // false!
```

They only compare equal if they *are the same reference*:

```
let a = [1, 2, 3];
let b = a;   // not a copy, but refers to same array!

a === b;   // true!
```

Checking for "empty array":

```
arr === []         // never true!

arr.length === 0   // this will work!
```

# Challenge: Arrays Equivalent?

## Arrays Equal?

Given two arrays that contain only numbers, are they "equivalent"?

(*equivalent*: do they contain the exact same items, in the same order)

This should return *true* or *false*

## Solution

```
function areArraysEquiv(numsA, numsB) {
  if (numsA.length !== numsB.length) {
    return false;
  }

  for (let i = 0; i < numsA.length; i++) {
    if (numsA[i] !== numsB[i]) {
      return false;
    }
  }

  return true;
}
```

## “Fail Fast” vs “Win Fast”

Many problems are either “fail fast” or “win fast”

*areArraysEqual* “fails fast” — as soon as we find any mismatch, we know we’ve lost; there’s no point at continuing to check

Other problems, like “are *any* items at same position equal?” would win fast.

# Looping

There are several ways to loop in JavaScript, and you should be familiar with them.

## Classic Counter Loop

*Simple loop*

```
for (let i = 0; i < 10; i++) {
  console.log("Number " + i);
}
```

*Looping over items in array*

```
let arr = ["a", "b", "c"];

for (let i = 0; i < arr.length; i++) {
  console.log(arr[i]);
}
```

- Flexible, can use in general or with arrays

## For…Of Loops

Consider code like this:

*Looping over items but with more logic*

```
let arr = ["a", "b", 7, "d"];

for (let i = 0; i < arr.length; i++) {
  if (arr[i] === 7) {
    console.log("LUCKY 7");
  } else if (arr[i] === 13) {
    console.log("UNLUCKY 13");
  } else {
    console.log(arr[i]);
  }
}
```

- We keep using *arr[i]* every time
- Do we *really* care about "where are we, index-wise, in array?"

*Looping over items with for ... of*

```
let arr = ["a", "b", 7, "d"];

for (let elem of arr) {
  if (elem === 7) {
    console.log("LUCKY 7");
  } else if (elem === 13) {
    console.log("UNLUCKY 13");
  } else {
    console.log(elem);
  }
}
```

- Less code & easier to read!
- But now we can't get access to "where are we, index-wise, in array"
- Works with elements-in-array and characters-in-string

## While Loops

*Loop until condition is hit*

```
while (Math.random() > 0.1) {
  console.log("hi");
}
```

Useful when you doing more "creative" looping than every-item

> **TIP** **Math.random**
>
> *Math.random()* is a very useful function in JavaScript. It returns a random fractional (*floating-point*) number from 0 to 0.99999...
>
> You can learn more about Math.random() <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/random>.

## Break

Stop the loop right now!

*Get out of loop with break*

```
let arr = ["a", "b", "STOP", "c"];

for (let elem of arr) {
  if (elem === "STOP") {
    break;
  }

  console.log(elem);
}
```

Works with any kind of loop ( `for (;;)` , `for (of)` , `while` )

> **TIP** **continue**
>
> A less common but still useful keyword for looping is *continue*. This drops immediately to the bottom of a loop so it can loop again.
>
> *Example of continue*
>
> ```
> for (let num of nums) {
>   // if number is even, skip it & go to next loop
>   if (num % 2 === 0) continue;
>
>   // must be odd, so do stuff here ...
>   // ...
> }
> ```

# Challenge: Sort Sorted Lists

Given two already-sorted lists, return new list of all elements, sorted.

But not using the *.sort()* method

## Solution

```
function sortSorted(listA, listB) {
  let combined = [];

  while (listA.length > 0 && listB.length > 0) {
    if (listA[0] <= listB[0]) {
      combined.push(listA.shift())
    } else {
      combined.push(listB.shift())
    }
  }

  while (listA.length > 0) {
    combined.push(listA.shift())
  }

  while (listB.length > 0) {
    combined.push(listB.shift())
  }

  return combined;
}
```