Private Prep Day Two

Introduction

Goals for tonight

- How to “unstick” a problem if you’re stuck
- Review/extend knowledge of JS:
  - Strings
  - Numbers
  - Objects

What To Do When You’re Stuck

Problem Solving Review

- 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

Explore concrete examples

- Can you work through the logic with a simple input?
  - What is general to other inputs?
  - What is specific to the input you chose?
- Can you solve a simpler version of problem?

Can’t Find a Strategy?

Can’t think of a working strategy? Find solutions that *won’t* work

This can get your brain going & may point to strategies that do work

Reading on Problem Solving

- Understand the Problem <https://www.rithmschool.com/blog/problem-solving-strategies-01>
- Explore Concrete Examples <https://www.rithmschool.com/blog/problem-solving-strategies-02>
- Break It Down <https://www.rithmschool.com/blog/problem-solving-strategies-03>
- Solve a Simpler Problem <https://www.rithmschool.com/blog/problem-solving-strategies-04>

Strings

Strings in JS can **never be changed** *(mutated)*

You can only make new strings from existing strings

Won't work

```
let name = "Jenny";

name.pop();     // no such method & impossible to make!
```

Redefining variable to new string will

```
let name = "Jenny";

name = name.slice(0, name.length - 1);  // ok!
```

Common String Operations

`str[idx]`
Return character at (0-based) index *idx*

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

`str.length`
Return # characters in string

Finding Things

`str.indexOf(chars)`
Return index (0-based) of *chars* or `-1` if not found

`str.includes(chars)`
Returns true/false for whether *chars* is present

Slicing

`str.slice(start, end)`
Returns *new string* of chars from *start* up to (not including) *end*

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

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

Case (Capitalization)

String comparison in JS is *case-sensitive* (`"a" !== "A"`)

Often you'll convert case of strings:

`str.toLowerCase()`
: Returns lower-cased version of string

`str.toUpperCase()`
: Returns upper-cased version of string

Miscellaneous

`str.startsWith(str2)`
: Does *str* start with *str2*?

`str.endsWith(str2)`
: Does *str* end with *str2*?

`str.split(str2)`
: Return array of *str* split at every *str2*

  Call with an empty string to split at every character

`str.repeat(num)`
: Return new string with *str* repeated *num* times

> NOTE **Splitting Strings By Character**
>
> The suggestion above to split a string into an array by character, `str.split('')` works — but doesn't work if you have uncommon characters in your string, like emoji-symbols or complex foreign-language characters.
>
> A better solution is to use `Array.from(str)` — this will give you an array with every character split properly.

Challenge: Vowels Only

Given a string, return string of each vowel in string, in order.

```
vowelsOnly("porcupine");  // "ouie"
vowelsOnly("moop");       // "oo"
```

Solution

```
function vowelsOnly(str) {
  let vowels = "aeiou";
  let vowelsOnlyStr = "";

  for (let char of str) {
    if (vowels.includes(char.toLowerCase())) {
      vowelsOnlyStr += char;
    }
  }

  return vowelsOnlyStr;
}
```

Challenge: Is Palindrome?

Return true/false if a string is a ***palindrome*** (a string that's the same forwards and backwards, like "noon")

If you want to make it harder, have it ignore capitalization and spaces, so that it returns true for `"No on"`.

> NOTE **Fun Fact!**
>
> The favorite palindrome among Rithm staff is the imaginary word "tacocat".

Solution

```
function isPalindrome(word) {
  // loop over half of word length
  for (let i = 0; i < Math.floor(word.length / 2); i++) {
    // check if that position from front == from end
    if (word[i] !== word[word.length - (i + 1)]) {
      return false;
    }
  }

  // if we got here, it must be a palindrome
  return true;
}
```

```
}
  return true;

}
}
  return false;
if (letters.pop() !== letters.shift()) {
while (letters.length > 1) {

let letters = word.split('');
function isPalindrome(word) {
```

Numbers

JavaScript has one type for both integer and ***floating-point*** numbers

```
let votingAge = 18;

let testScore = 88.5;
```

Common Operations

`+` `-` `*` `/`
: As expected: add, subtract, multiply, divide

`%`
: "Modulo" (remainder after division): `7 % 2 === 1`

Infinity

JS has `Infinity` and `-Infinity`

These are often useful: all real numbers are bigger than *-Infinity* and less than *Infinity*.

Converting Strings To Numbers

`Number(strOfNum)`
: Converts to a number *(int or float)*

`parseInt(strOfInt)`
: Converts to an integer number.

  Ignores trailing invalid stuff: `parseInt("123 ok") === 123`

`+strOfNum`
: Converts to a number *(int or float)*

Not A Number

JS has a special value, *NaN*:

Commonly result of:

- failing a conversion: `Number("hello")`
- illogical math, like `0 / 0` or `Math.sqrt(-1)`

Tricky part: **nothing is ever equal-to NaN**

Even another NaN!

```
let x = NaN;
let y = NaN;

x === y;   // false!
```

To figure out: is something not-a-number

`isNaN(num)`

Is *num* NaN or can it not be converted to a number?

`isNaN(NaN) === true`

`isNaN("hello") === true`

`isNaN("7") === false`

`isNaN(7) === false`

Challenge: Find Largest Number

Find Largest Number

Write a function that finds the biggest number in an array.

- array contains numbers and strings-of-numbers
- array can contain strings that cannot be converted
  - print a warning about those and then skip them
- do this *without sorting the array*.

```
largestNum([2, 1, "oh no", "4"]);
// return number 4 and logs '"oh no" is not a number'
```

(If you'd like a harder problem, do *secondLargest*)

Solution

```
function largest(nums) {
  let max = -Infinity;

  for (let x of nums) {
    let n = Number(x);

    if (isNaN(n)) {
    } if (n > max) {
      max = n;
    } else {
      console.log("Not a number:", n);
    }
  }

  return max;
}
```

Objects

Objects map a *key* to a *value*

```
let fruits = {
  "apple": "red",
  "berry": "blue",
  "cherry": "red",
};
```

For the key of “apple”, the value is “red”

As long as the *keys* are simple strings, can leave of quote marks:

```
let fruits = {
  apple: "red",
  berry: "blue",
  cherry: "red",
};
```

Keys are always strings — JS will turn them into strings for you:

```
let nums = {
  1: "one",
  2: "two",
};

nums["1"];   // "one"

nums[1];     // finds same --- "one"
```

Values don’t get turned in strings — they stay what they were set to:

```
let city = {
  name: "San Francisco",       // stays a string
  squareMiles: 49,             // stays a number
  awesome: true,               // stays a boolean
  nicknames: ["SF", "Frisco"], // stays an array
};
```

Accessing/Adding to Objects

Two ways!

- Dot notation: `fruits.banana = "yellow"`
- Bracket notation: `fruits["banana"] = "yellow"`

Dot vs Bracket notation

- If you know with 100% certainty what the key is — always use dot
- If you are not 100% sure what the key will be, you **must** use bracket

JavaScript evaluates whatever you put in and converts it to a string!

```
let fruits = {
  apple: "red",
  berry: "blue",
  cherry: "red",
};

let favFruit = "apple";

fruits.apple;       // "red"
fruits[favFruit];   // "red"
```

Comparing Objects

Similar to array, can't use `==` or `===` to compare different objects:

```
let a = {"apple": "red"};
let b = {"apple": "red"};

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

This only works if it's the same *reference*:

```
let a = {"apple": "red"};
let b = a;

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

Checking for Key in Object

What's the possible bug?

```
if (fruits.apple) {
  // ...
}
```

Break if *fruits.apple* is *falsy*

Much better!

```
if ("apple" in fruits) {
  // ...
}
```

Always works!

Getting All Keys/Values

```
let fruits = {
  "apple": "red",
  "berry": "blue",
  "cherry": "red",
};

Object.keys(fruits);   // ["apple", "berry", "cherry"]
Object.values(fruits); // ["red", "blue", "red"];
```

More Looping

Looping Over Objects

Use `for ... of` to loop over **arrays and strings**

Use `for ... in` to loop over **objects** *(by key)*:

```
let fruits = {
  "apple": "red",
  "berry": "blue",
  "cherry": "red",
};

for (let fruit in fruits) {
  console.log("A", fruit, "is", fruits[fruit]);
}
```

Challenge: Letter Counts

For a word, return an object of the counts of letters

```
letterCount("hello"); // { h: 1, e: 1, l: 2, o: 1 }
```

Solution

```
function letterCount(str) {
  let counts = {};

  for (let char of str) {
    if (!(char in counts)) {
      counts[char] = 1;
    } else {
      counts[char] += 1;
    }
  }

  return counts;
}
```

Challenge: Invert Object

Given an object like:

```
capitalsToStates = {
  "Annapolis": "MD",
  "Sacramento": "CA",
  "Trenton": "NJ",
}
```

return an object the swaps the keys and values:

```
statesToCapitals = {
  "MD": "Annapolis",
  "CA": "Sacramento",
  "NJ": "Trenton",
}
```

Write a version that works where a state can have many cities:

```
cities = {
  "Annapolis": "MD",
  "Baltimore": "MD",
  "Trenton": "NJ",
}
```

return an object the swaps the keys and values:

```
statesToCities = {
  "MD": ["Annapolis", "Baltimore"],
  "NJ": ["Trenton"],
}
```