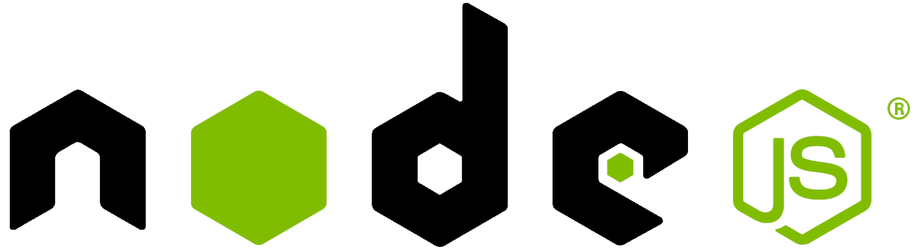
JAVASCRIPT

B1- MIDTERM RECAP

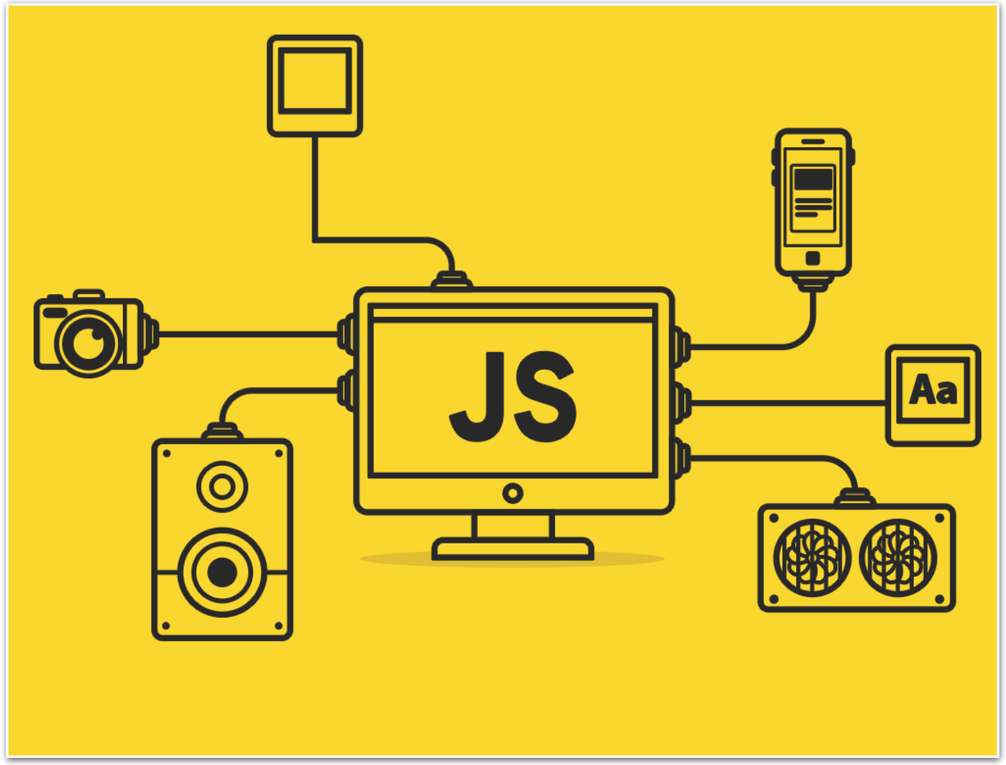
Anirach Mingkhwan

Anirach.m@fitm.kmutnb.ac.th



OUTLINE

- Practice



MIDTERM PRACTICE

Number Split

Published by Matt in JavaScript

arrays math numbers

Given a number, return an array containing the two halves of the number. If the number is odd, make the **rightmost number higher**.

Examples

```
numberSplit(4) → [2, 2]

numberSplit(10) → [5, 5]

numberSplit(11) → [5, 6]

numberSplit(-9) → [-5, -4]
```

Notes

- All numbers will be integers.
- You can expect negative numbers too.

Highest Digit

Published by BijogFc24 in JavaScript

language_fundamentals numbers

Create a function that takes a number as an argument and returns the highest digit in that number.

Examples

```
highestDigit(379) → 9

highestDigit(2) → 2

highestDigit(377401) → 7
```

Notes

- Don't forget to `return` the result.
- If you get stuck on a challenge, find help in the **Resources** tab.
- If you're really stuck, unlock solutions in the **Solutions** tab.

Move Capital Letters to the Front

Published by Helen Yu in JavaScript

formatting regex strings

Create a function that moves all capital letters to the front of a word.

Examples

```
capToFront("hApPy") → "APhpy"

capToFront("moveMENT") → "MENTmove"

capToFront("shOrtCAKE") → "OCAKEshrt"
```

Notes

Keep the original relative order of the upper and lower case letters the same.

Total Volume of All Boxes

Published by **Helen Yu** in **JavaScript**

arrays geometry math

Given an array of boxes, create a function that returns the total volume of all those boxes combined together. A box is represented by an array with three elements: length, width and height.

For instance, `totalVolume([2, 3, 2], [6, 6, 7], [1, 2, 1])` should return `266` since (2 x 3 x 2) + (6 x 6 x 7) + (1 x 2 x 1) = 12 + 252 + 2 = 266.

Examples

```
totalVolume([4, 2, 4], [3, 3, 3], [1, 1, 2], [2, 1, 1]) → 63

totalVolume([2, 2, 2], [2, 1, 1]) → 10

totalVolume([1, 1, 1]) → 1
```

Notes

- You will be given at least one box.
- Each box will always have three dimensions included.

Get Sum of People's Budget

Published by **felixjoykind** in **JavaScript**

arrays logic objects

Create the function that takes an array with objects and returns the sum of people's budgets.

Examples

```
getBudgets([
  { name: "John", age: 21, budget: 23000 },
  { name: "Steve",  age: 32, budget: 40000 },
  { name: "Martin",  age: 16, budget: 2700 }
]) → 65700

getBudgets([
  { name: "John",  age: 21, budget: 29000 },
  { name: "Steve",  age: 32, budget: 32000 },
  { name: "Martin",  age: 16, budget: 1600 }
]) → 62600
```

Notes

N/A

Remove Duplicates from an Array

Published by Matt in JavaScript

arrays interview language_fundamentals strings

Create a function that takes an array of items, removes all duplicate items and returns a new array in the same sequential order as the old array (minus duplicates).

Examples

```
removeDups([1, 0, 1, 0]) → [1, 0]

removeDups(["The", "big", "cat"]) → ["The", "big", "cat"]

removeDups(["John", "Taylor", "John"]) → ["John", "Taylor"]
```

Notes

- Tests contain arrays with both strings and numbers.
- Tests are case sensitive.
- Each array item is unique.

Return the Sum of the Two Smallest Numbers

Published by Matt in JavaScript

arrays math numbers

Create a function that takes an array of numbers and returns the sum of the two lowest **positive** numbers.

Examples

```
sumTwoSmallestNums([19, 5, 42, 2, 77]) → 7

sumTwoSmallestNums([10, 343445353, 3453445, 3453545353453]) → 3453455

sumTwoSmallestNums([2, 9, 6, -1]) → 8

sumTwoSmallestNums([879, 953, 694, -847, 342, 221, -91, -723, 791, -587])

sumTwoSmallestNums([3683, 2902, 3951, -475, 1617, -2385]) → 4519
```

Notes

- Don't count negative numbers.
- Floats and empty arrays will not be used in any of the test cases.

A Simple Pair

Published by **Mubashir Hassan** in **JavaScript**

algebra | interview | math | numbers

Mubashir needs your help to write a simple algorithm of multiplication.

Given an array of integers `arr` and an integer `n`, find out a pair of numbers `[x, y]` from a given array such that **x * y = n** .

If the pair is not found, return `null`.

Examples

```
simplePair([1, 2, 3], 3) → [1, 3]

simplePair([1, 2, 3], 6) → [2, 3]

simplePair([1, 2, 3], 9) → null
```

Notes

N/A

Time Conversion

Published by Deep Xavier in JavaScript

algebra math numbers strings

Write a function that takes the number of `seconds` and returns the digital format clock time as a string. Time should be counted from `00:00:00`.

Examples

```
digitalClock(5025) → "01:23:45"
// 5025 seconds is 1 hour, 23 mins, 45 secs.

digitalClock(61201) → "17:00:01"
// No AM/PM. 24h format.

digitalClock(87000) → "00:10:00"
// It's 00:10 next day.
```

Notes

`seconds` is always greater than or equal to 0.

IPv4 Validation

Published by Matt in JavaScript

algorithms regex strings validation

Create a function that takes a string (IPv4 address in standard dot-decimal format) and returns `true` if the IP is valid or `false` if it's not. For information on IPv4 formatting, please refer to the resources in the **Resources** tab.

Examples

```
isValidIP("1.2.3.4") → true

isValidIP("1.2.3") → false

isValidIP("1.2.3.4.5") → false

isValidIP("123.45.67.89") → true

isValidIP("123.456.78.90") → false

isValidIP("123.045.067.089") → false
```

Notes

- IPv6 addresses are not valid.
- Leading zeros are not valid (`"123.045.067.089"` should return `false`).
- You can expect a single string for every test case.
- Numbers may only be between 1 and 255.
- The last digit may not be zero, but any other might.

ADVANCED PRACTICE

Chess Pieces

Published by Matt in JavaScript

algorithms games logic validation

Create a function that takes the name of a chess piece, its position and a target position. The function should return `true` if the piece can move to the target and `false` if it can't.

The possible inputs are "Pawn", "Knight", "Bishop", "Rook", "Queen" and "King".

Examples

```
canMove("Rook", "A8", "H8") → true

canMove("Bishop", "A7", "G1") → true

canMove("Queen", "C4", "D6") → false
```

Notes

- Do not include pawn capture moves and en passant.
- Do not include castling.
- Remember to include pawns' two-square move on the second rank!
- Look for patterns in the movement of the pieces.

Sort Characters by Frequency, Case, Alphabet

Published by **Werdna** in **JavaScript**

algorithms conditions sorting strings

The function is given a string. Sort the characters and return a new string. Sorting conditions:

- Most frequent (case-specific) move in front.
- For the same frequency upper case characters move in front.
- For the same frequency and case sort them alphabetically.

Examples

```
frequencySort("tree") → "eert"

frequencySort("cccaaa") → "aaaccc"

frequencySort("Aabb") → "bbAa"
```

A large flat field measures *50x50* kilometers. At a time `t = 0`, a bomb explodes somewhere on the field. There are 3 scattered sensors with synchronized clocks that record the exact time (in seconds) that the sound of the exploding bomb reaches them.

Given the *x*, *y* coordinates of each of the 3 sensors and the recorded time of each, find the location of the bomb:

```
bomb([[x1, y1, t1], [x2, y2, t2], [x3, y3, t3]]) → [xb, yb]
// Bomb location
```

Examples

```
bomb([[0, 0, 72.886], [0, 50, 72.886], [25, 25, 72.886]]) → [0, 25]

bomb([[0, 50, 145.773], [50, 50, 206.154], [50, 0, 145.773]]) → [0, 0]

bomb([[5, 8, 48.872], [12, 21, 35.107], [24, 20, 22.203]]) → [21, 13]

bomb([[18, 42, 35.558], [39, 16, 106.004], [7, 24, 32.202]]) → [8, 35]
```

Notes

- The origin is at one corner of the square field so all the coordinates are positive integers in km units.
- The bomb's coordinates are integers.
- The speed of sound in air is 0.343 km/sec.

This string, `"sadbpstcrdvaefikkgoenqrt"` has a five letter word embedded within it.

Here's a clue on how to find it:

1. The `str` can be broken up from left to right into a series of overlapping letter triplets.
2. The letter values for each triplet are summed with a=1, b=2, ..., z=26.
3. The values of the triplets that contain the letters of the secret word as the middle member form an increasing arithmetic series.

Given the `str` and `len` of the secret word, improvise a function that finds the secret word.

Examples

```
secretWord("sadbpstcrdvaefikkgoenqrt", 5) → "brake"
// sa(dbp)st(crd)(vae)f(ikk)g(oen)qrt
// The values of the triplets in parentheses are 22, 25, 28, 31, 34.
// An arithmetic series with difference of 3.

secretWord("aheiayd", 3) → "hey"
// (a[he)i](ayd)
// The triplets with the secret letters can overlap.
// ahe=14, hei=22, ayd=30; a series with a difference of 8.
```

A frog wants to cross a river. Unfortunately, he can't jump across in a single leap. Luckily, there are `n` stones in the river.

The frog can jump from the near bank to stone `1` and from stone `n` to the far bank. He can also jump from stone to stone, forward and backward. However, on each stone, a number `j` is written and he must only jump exactly `j` stones backward or forward.

Return the minimum number of jumps to cross the river (including jumps to the first stone and from the last stone (or any other stone, if possible) to the far bank) or `no chance :-(` if it's not possible to cross the river.

Examples

```
jumpingFrog(5, [1, 1, 1, 1, 1]) → 6

jumpingFrog(5, [1, 3, 1, 1, 1]) → 4

jumpingFrog(5, [1, 1, 0, 1, 1]) → "no chance :-("
```

Notes

- The frog may also reach the far bank from another stone than `n` if a large enough number is written on it.
- `n` is at least 2.

Given a string containing just the characters `(` and `)`, find the length of the longest valid (well-formed) parentheses substring.

Examples

```
longestValidParentheses("(()") → 2
// Longest valid parentheses substring is "()"

longestValidParentheses(")()())") → 4
// Longest valid parentheses substring is "()()"

longestValidParentheses("()))))(()())(") → 6
```

Write a function that checks if a given string contains an **additive sequence** or not. A string contains an additive sequence if its digits can make a sequence of numbers in which every number is the sum of the previous two numbers. A valid additive sequence should contain at least three numbers.

Examples

```
isAdditive("112358") → true
// The digits can form an additive sequence: 1, 1, 2, 3, 5, 8.
// 1 + 1 = 2, 1 + 2 = 3, 2 + 3 = 5, 3 + 5 = 8

isAdditive("129881000") → true
// Each number can contain 1 or more digits: 12, 988, 1000.
// 12 + 988 = 1000

isAdditive("12988110101891") → true
// 129 + 881 = 1010, 881 + 1010 = 1891

isAdditive("123456789") → false

isAdditive("300045007500") → true
```

Notes

- The string will contain only digits `0 → 9`
- Numbers in the additive sequence **cannot** have leading zeros, so sequence `1, 2, 03` or `1, 02, 3` is invalid.

You are provided with an array of animals given below:

```
animals = ["dog", "cat", "bat", "cock", "cow", "pig",
"fox", "ant", "bird", "lion", "wolf", "deer", "bear",
"frog", "hen", "mole", "duck", "goat"]
```

Rule: Return the **maximum number** of animal names. See the below example:

```
txt = "goatcode"

count_animals(txt) → 2
# First animal = "dog"
# Remaining string = "atcoe",
# Second animal = "cat".
# count = 2 (correct)

# If you got a "goat" first time
# remaining string = "code",
# no animal will be found during next time.
# count = 1 (wrong)
```

Examples

```
count_animals("goatcode") → 2
# "dog", "cat"

count_animals("cockdogwdufrbir") → 4
# "cow", "duck", "frog", "bird"

count_animals("dogdogdogdogdog") → 5
```

THANK YOU

- Class
- Module
- Error Handling