AIM: How do we create and implement an algorithm?

HW: finish your algorithm!

Turn-and-talk:

What is the difference between "**algorithm**" and "**code**"?

Or are they the same?

Or is there overlap?

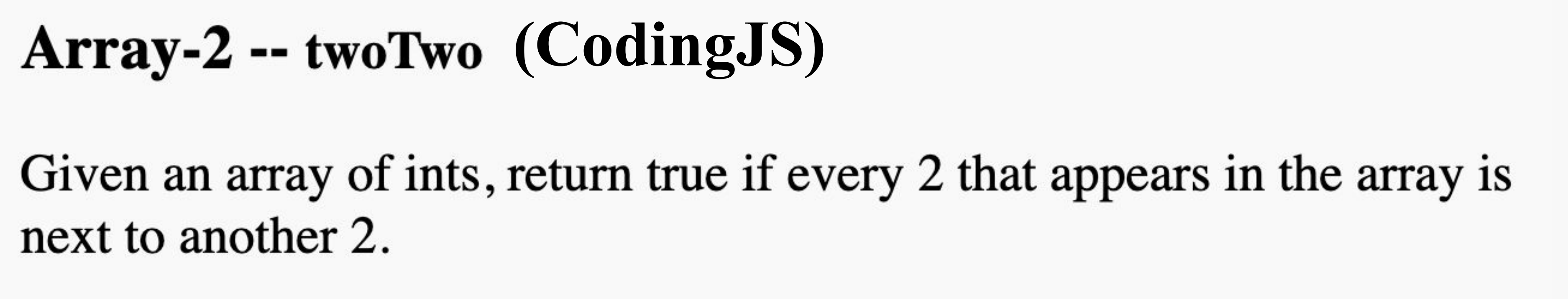
Well?

"Algorithm" and "code" aren't *exactly* the same thing, but they are certainly related!

In this lesson, we will unpack how they overlap.

TwoTwo

Example



Array-2 -- twoTwo (CodingJS)

Given an array of ints, return true if every 2 that appears in the array is next to another 2.

https://the-winter.github.io/codingjs/exercise.html?name=twoTwo&title=Array-2

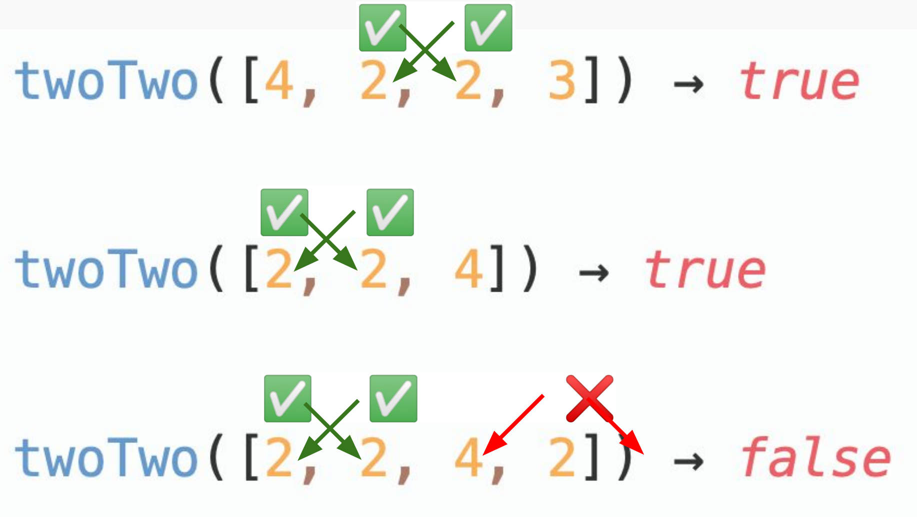
How to approach a problem like this

- Understand the problem
- Write pseudocode for an algorithm to solve the problem
- Turn the pseudocode into actual code
- Revise as necessary
    - Errors --> debug
    - No errors --> refactor for efficiency_

Array-2 -- twoTwo (CodingJS)

Given an array of ints, return true if every 2 that appears in the array is next to another 2.

Examples



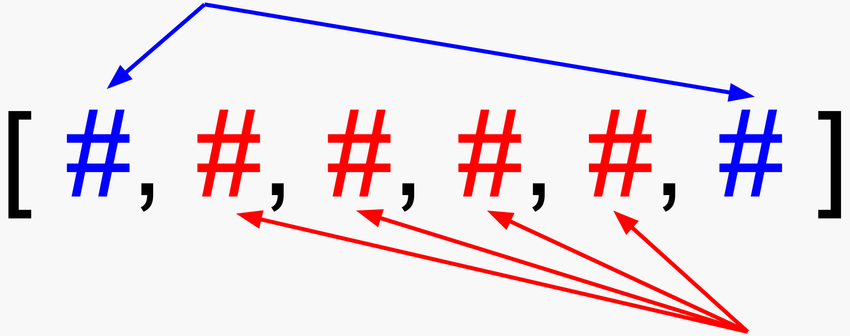
```
twoTwo([4, 2, 2, 3]) → true
twoTwo([2, 2, 4]) → true
twoTwo([2, 2, 4, 2]) → false
```

What about `twoTwo([3, 2, 2, 2, 1])` ?

https://the-winter.github.io/codingjs/exercise.html?name=twoTwo&title=Array-2

Pseudocode the algorithm

- Look for "scenarios" that would return false, otherwise assume all good (return true)
- Only look for 2s, then look at neighbors of 2s
    - Check first and last since those are special cases (only one neighbor)



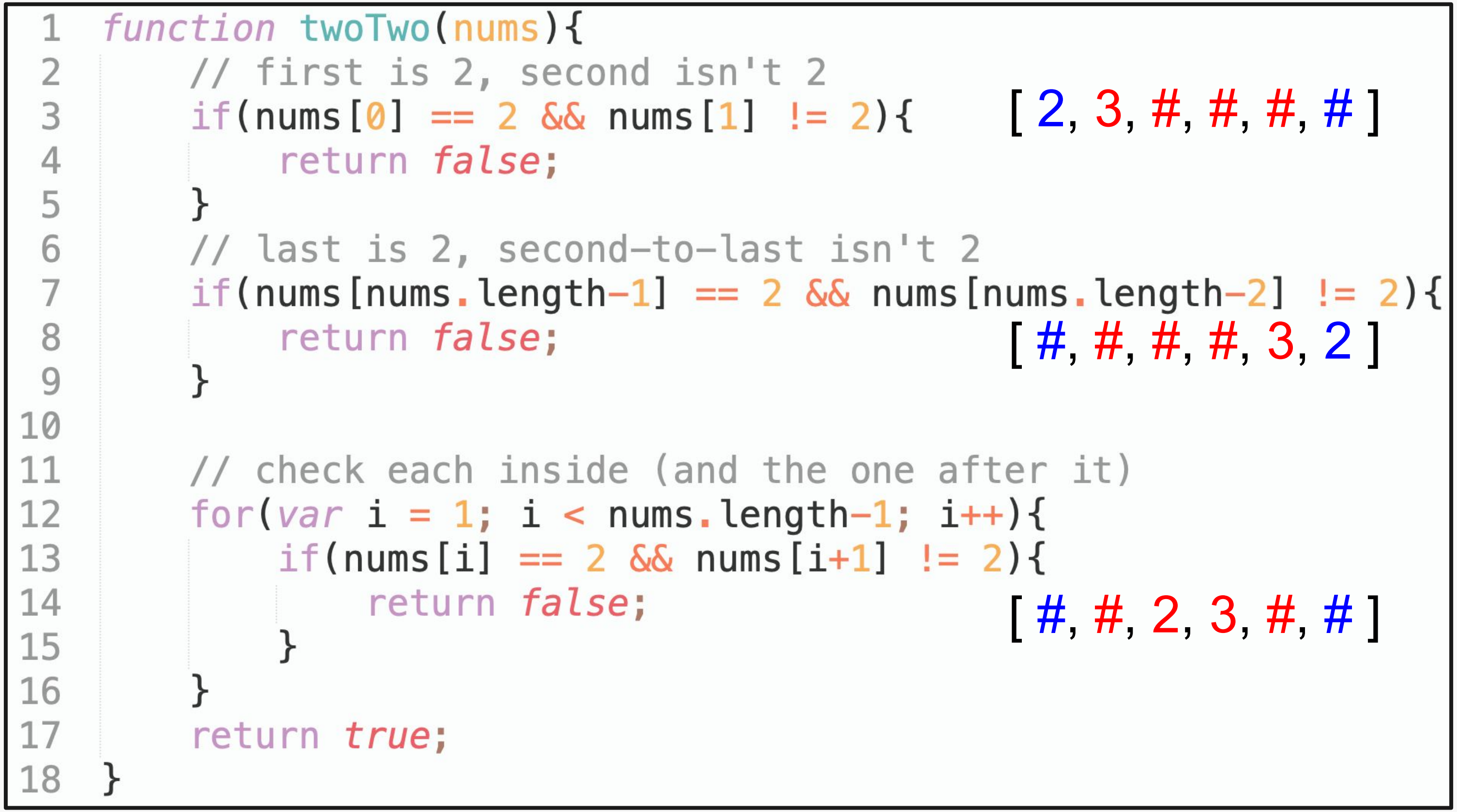
[#, #, #, #, #, #]

    - Check inside (everything with two neighbors)

Turn pseudocode into actual code

[DEMO]

Turn pseudocode into actual code



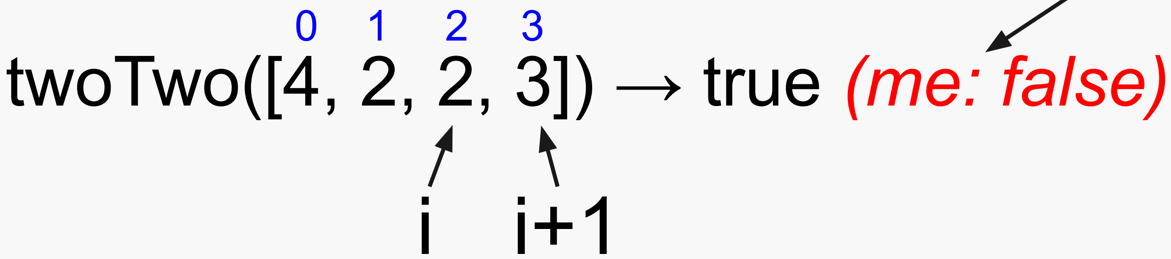
```
function twoTwo(nums){
    // first is 2, second isn't 2
    if(nums[0] == 2 && nums[1] != 2){
        return false;
    }
    // last is 2, second-to-last isn't 2
    if(nums[nums.length-1] == 2 && nums[nums.length-2] != 2){
        return false;
    }

    // check each inside (and the one after it)
    for(var i = 1; i < nums.length-1; i++){
        if(nums[i] == 2 && nums[i+1] != 2){
            return false;
        }
    }
    return true;
}
```

[2, 3, #, #, #, #]

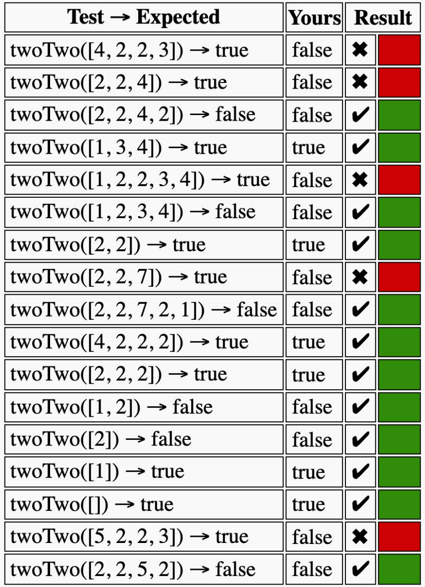
[#, #, #, #, 3, 2]

[#, #, 2, 3, #, #]

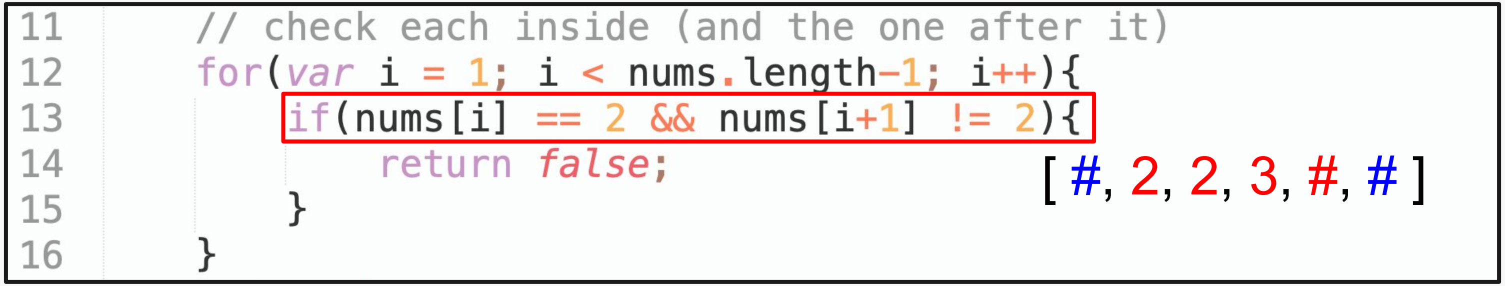
Error!



Accidentally returned false because I wasn't also checking the left neighbor



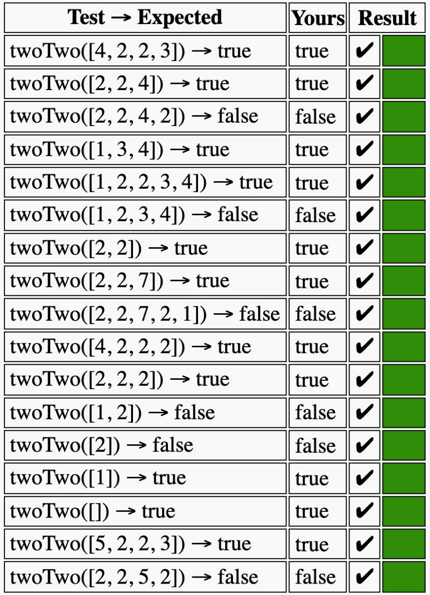
Test → Expected	Yours	Result	
twoTwo([4, 2, 2, 3]) → true	false	✖	🟥
twoTwo([2, 2, 4]) → true	false	✖	🟥
twoTwo([2, 2, 4, 2]) → false	false	✔	🟩
twoTwo([1, 3, 4]) → true	true	✔	🟩
twoTwo([1, 2, 2, 3, 4]) → true	false	✖	🟥
twoTwo([1, 2, 3, 4]) → false	false	✔	🟩
twoTwo([2, 2]) → true	true	✔	🟩
twoTwo([2, 2, 7]) → true	false	✖	🟥
twoTwo([2, 2, 7, 2, 1]) → false	false	✔	🟩
twoTwo([4, 2, 2, 2]) → true	true	✔	🟩
twoTwo([2, 2, 2]) → true	true	✔	🟩
twoTwo([1, 2]) → false	false	✔	🟩
twoTwo([2]) → false	false	✔	🟩
twoTwo([1]) → true	true	✔	🟩
twoTwo([]) → true	true	✔	🟩
twoTwo([5, 2, 2, 3]) → true	false	✖	🟥
twoTwo([2, 2, 5, 2]) → false	false	✔	🟩



```
    // check each inside (and the one after it)
    for(var i = 1; i < nums.length-1; i++){
        if(nums[i] == 2 && nums[i+1] != 2){
            return false;
        }
    }
```

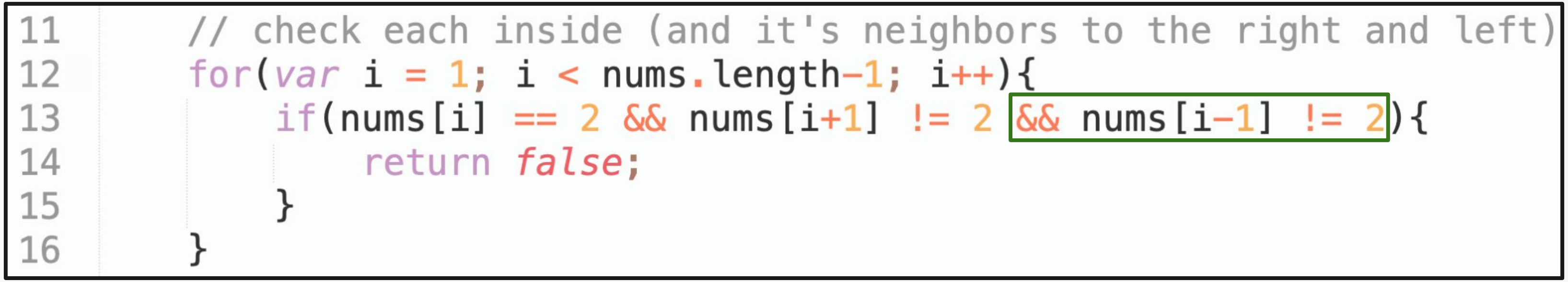
[#, 2, 2, 3, #, #]

Solution



Test → Expected	Yours	Result	
twoTwo([4, 2, 2, 3]) → true	true	✔	🟩
twoTwo([2, 2, 4]) → true	true	✔	🟩
twoTwo([2, 2, 4, 2]) → false	false	✔	🟩
twoTwo([1, 3, 4]) → true	true	✔	🟩
twoTwo([1, 2, 2, 3, 4]) → true	true	✔	🟩
twoTwo([1, 2, 3, 4]) → false	false	✔	🟩
twoTwo([2, 2]) → true	true	✔	🟩
twoTwo([2, 2, 7]) → true	true	✔	🟩
twoTwo([2, 2, 7, 2, 1]) → false	false	✔	🟩
twoTwo([4, 2, 2, 2]) → true	true	✔	🟩
twoTwo([2, 2, 2]) → true	true	✔	🟩
twoTwo([1, 2]) → false	false	✔	🟩
twoTwo([2]) → false	false	✔	🟩
twoTwo([1]) → true	true	✔	🟩
twoTwo([]) → true	true	✔	🟩
twoTwo([5, 2, 2, 3]) → true	true	✔	🟩
twoTwo([2, 2, 5, 2]) → false	false	✔	🟩

Only return false if right AND left aren't 2

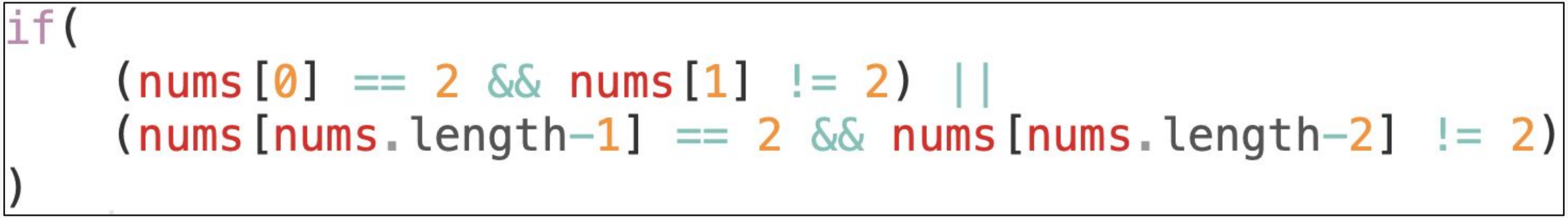


```
// check each inside (and it's neighbors to the right and left)
for(var i = 1; i < nums.length-1; i++){
    if(nums[i] == 2 && nums[i+1] != 2 && nums[i-1] != 2){
        return false;
    }
}
```

Refactoring

- The scenarios where
  - There is a 2 at the beginning
  - There is a 2 at the end

can be combined, although the expression is quite lengthy.



```
if(
    (nums[0] == 2 && nums[1] != 2) ||
    (nums[nums.length-1] == 2 && nums[nums.length-2] != 2)
)
```

Recap

- Understand the problem **(examples & non-examples)**
- Write pseudocode for an algorithm to solve the problem **(check outside/inside, looking for false)**
- Turn the pseudocode into actual code **(JS code)**
- Revise as necessary
    - Errors **(check right <u>and</u> left neighbors)**
    - Efficiency **(combine scenarios with 2 on the end)**

Activity

In pairs, decide on which task to try:

- Tough: swapEnds
- Tougher: caughtSpeeding
- Toughest: tripleUp

Navigator
Driver

- Understand the problem
- Write pseudocode for an algorithm to solve the problem
- Turn the pseudocode into actual code
- Revise as necessary
    - Errors --> debug
    - No errors --> refactor for efficiency

Summary

- Did you have to change your original algorithm?
    - If so, how
- Finished?
    - How close was your code to your algorithm?
    - Did you under/over-plan?
- Not finished?
    - Is there anything in your algorithm that you don't know how to turn into code?