PSCS

Problem Solving for Computer Science

IS51021C

Goldsmiths Computing

February 1, 2020

Wk 4

Abstract Data Structures

Abstract models of collections of data: specified in terms of structure and operations

Operations are defined independently of implementation
e.g. do not specify how to compute length of vector

Extremely useful in algorithms

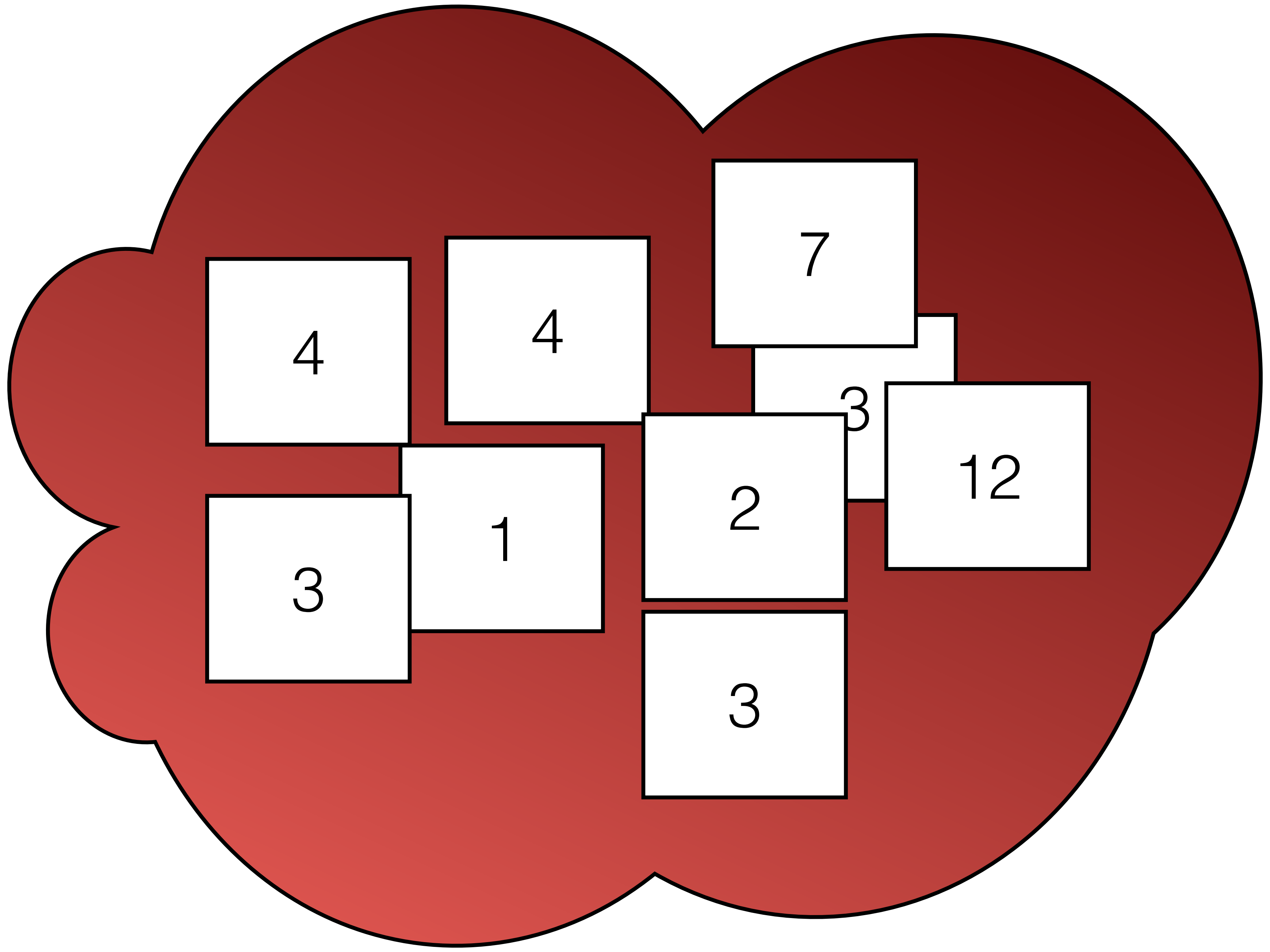
For last Review Seminar: design your own abstract data structure

How is the data structured?

What operations do we have?

My example

The bottomless pit



Allowed operation:

push![o] Adds a new element to the pit with value o

Abstract Data Structures

Abstract models of collections of data: specified in terms of structure and operations

Operations are defined independently of implementation
e.g. do not specify how to compute length of vector

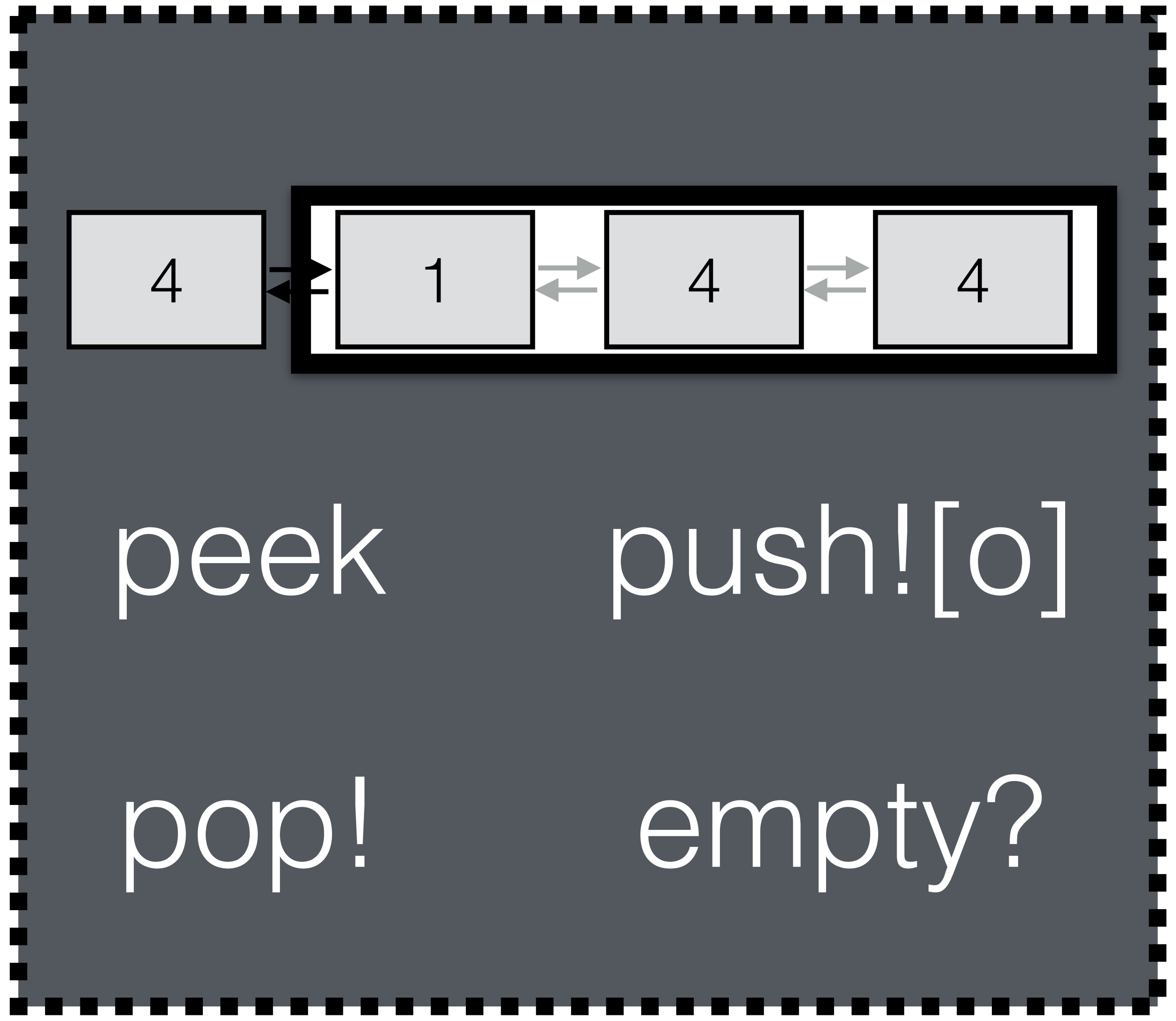
Extremely useful in algorithms

Objects

Used when we come to implement abstract data structures

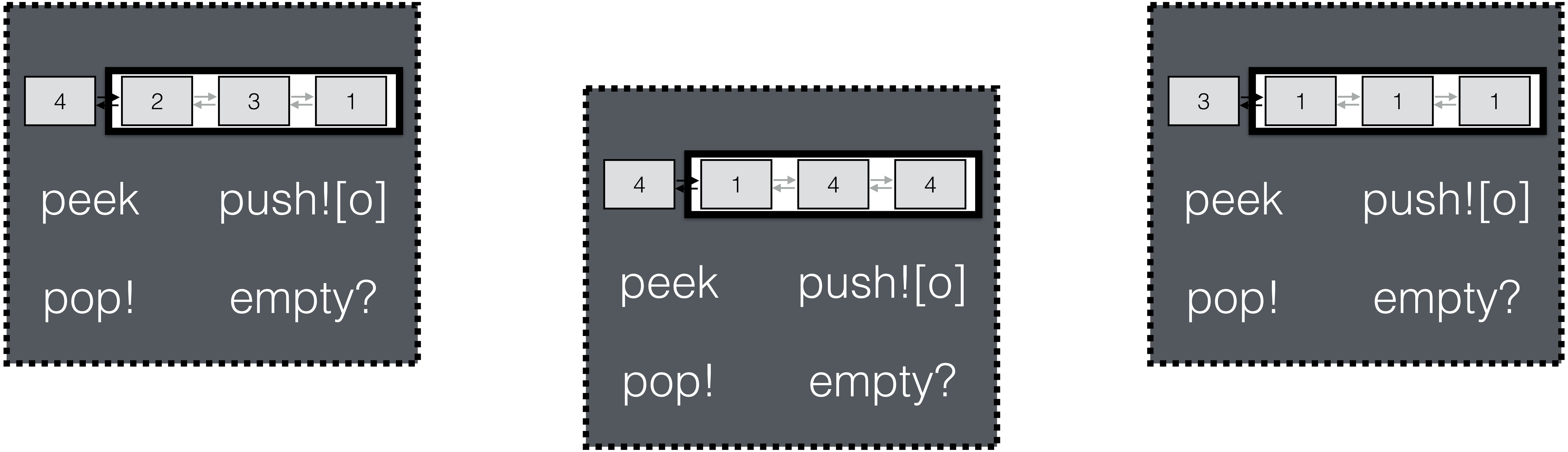
Get comfortable with methods and ***this***

Objects for implementation



- Objects will be used to implement a particular abstract data structure
- Methods will implement our operations

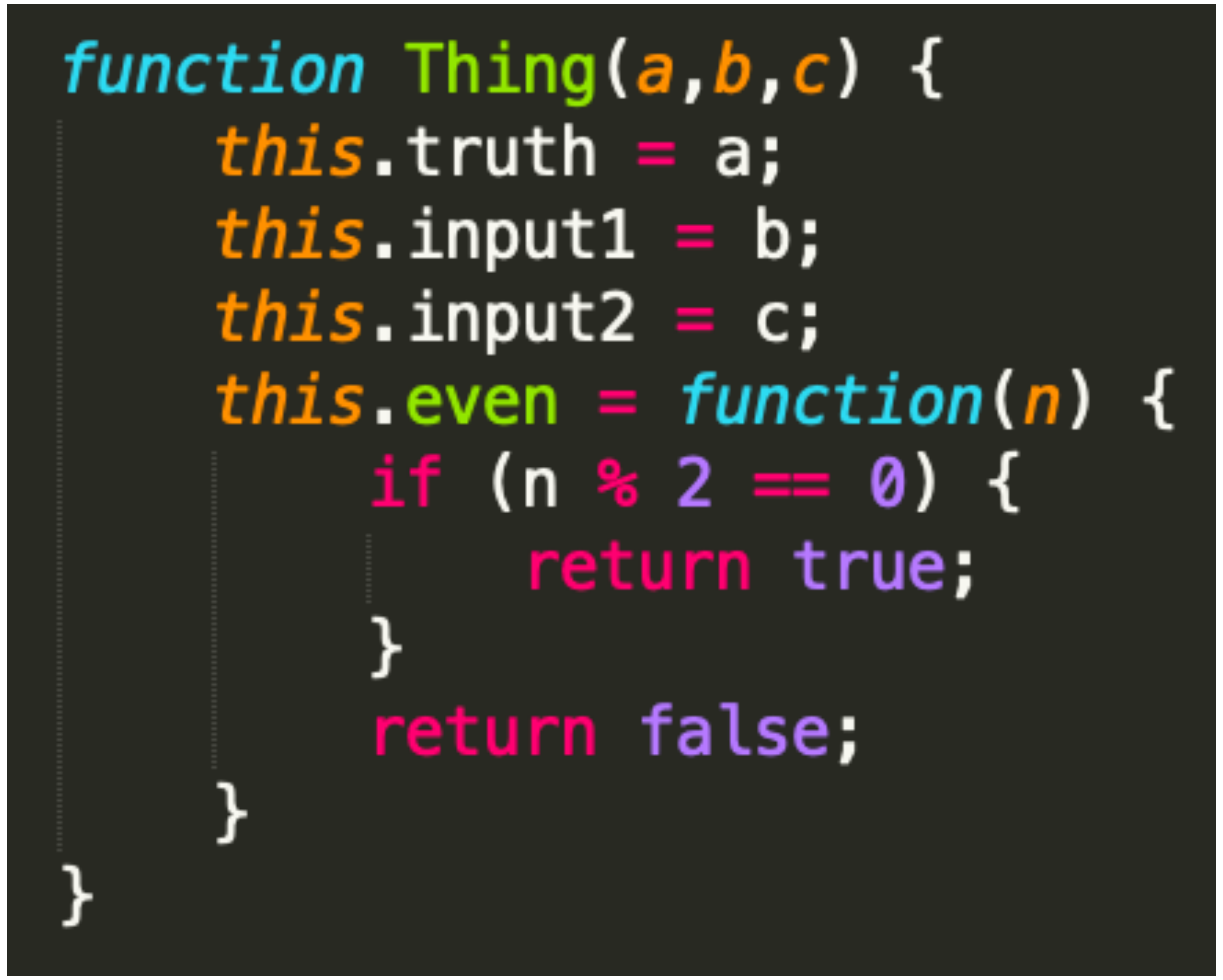
Objects for implementation



- Objects will be used to implement a particular abstract data structure
- Methods will implement our operations
- Want to create multiple implementations with same structure and operations
- We need **constructors** - gets around directly copying objects

Constructors revision

Standard practice: capitalise constructor names

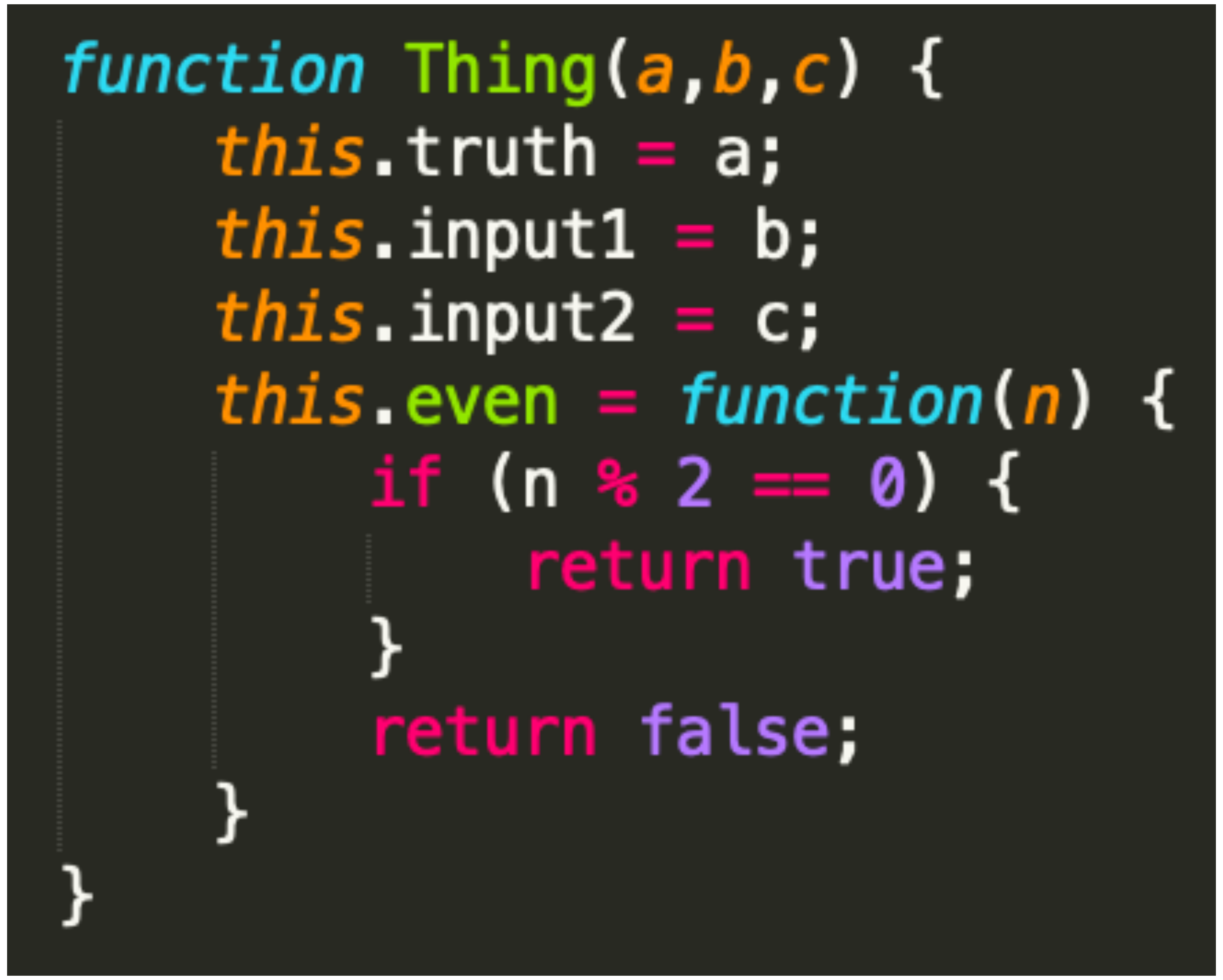


```
function Thing(a,b,c) {
    this.truth = a;
    this.input1 = b;
    this.input2 = c;
    this.even = function(n) {
        if (n % 2 == 0) {
            return true;
        }
        return false;
    }
}
```

this refers to the object created by the function when called

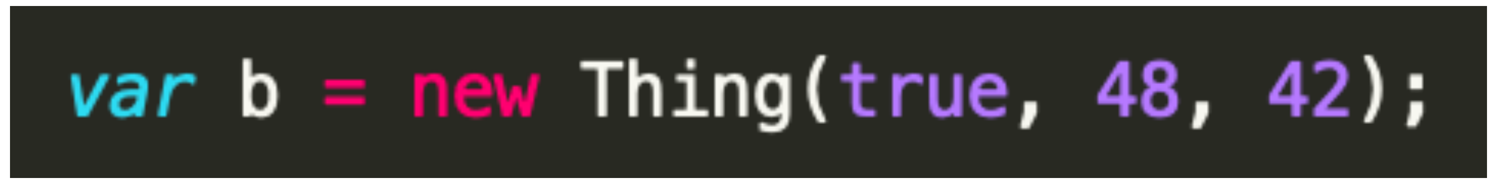
Constructors revision

Standard practice: capitalise constructor names



```
function Thing(a,b,c) {
    this.truth = a;
    this.input1 = b;
    this.input2 = c;
    this.even = function(n) {
        if (n % 2 == 0) {
            return true;
        }
        return false;
    }
}
```

this refers to the object created by the function when called



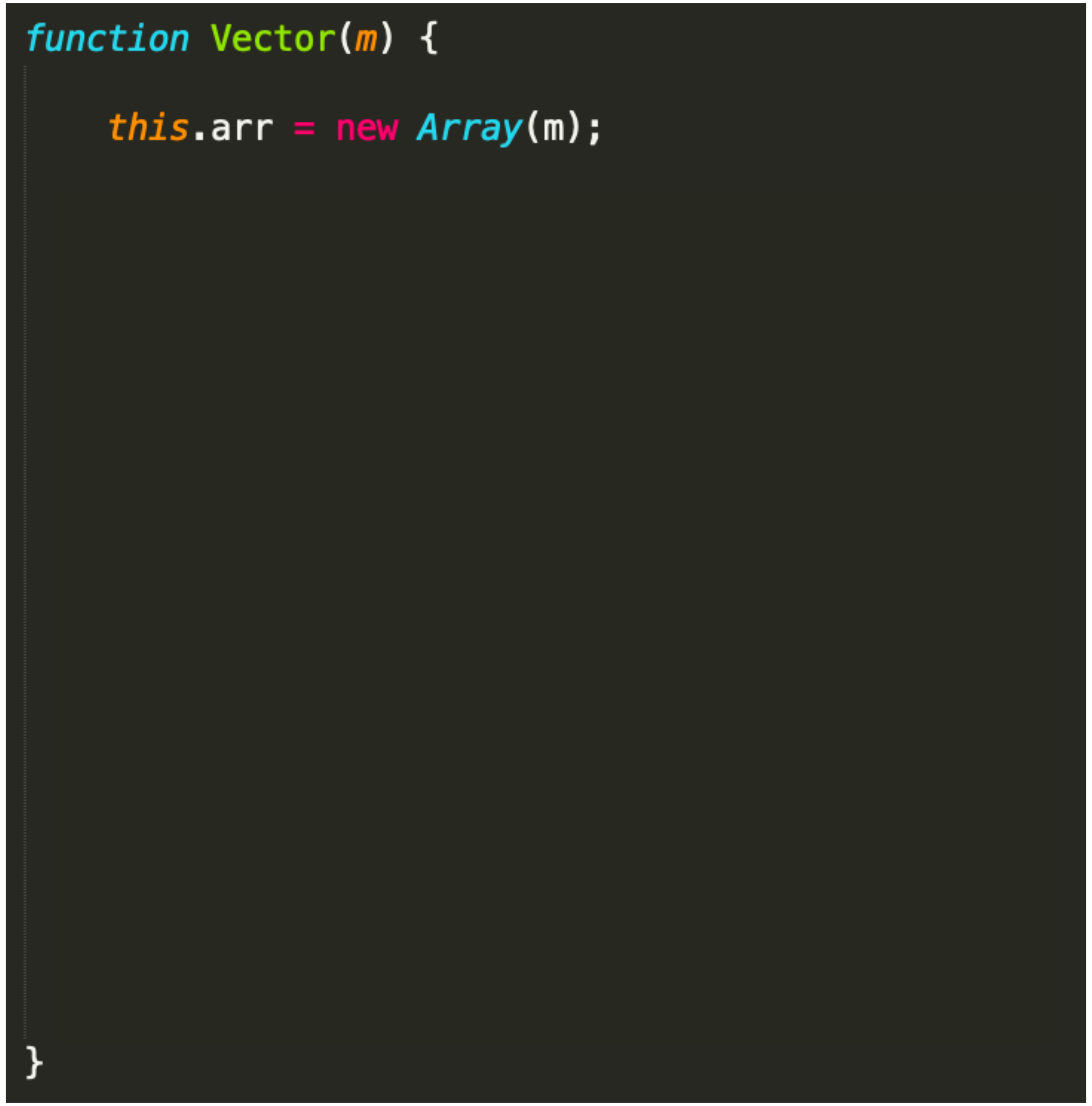
```
var b = new Thing(true, 48, 42);
```

new creates completely new object from Thing constructor and its reference is stored in b

Vector Constructor

Use JavaScript array as the basic object

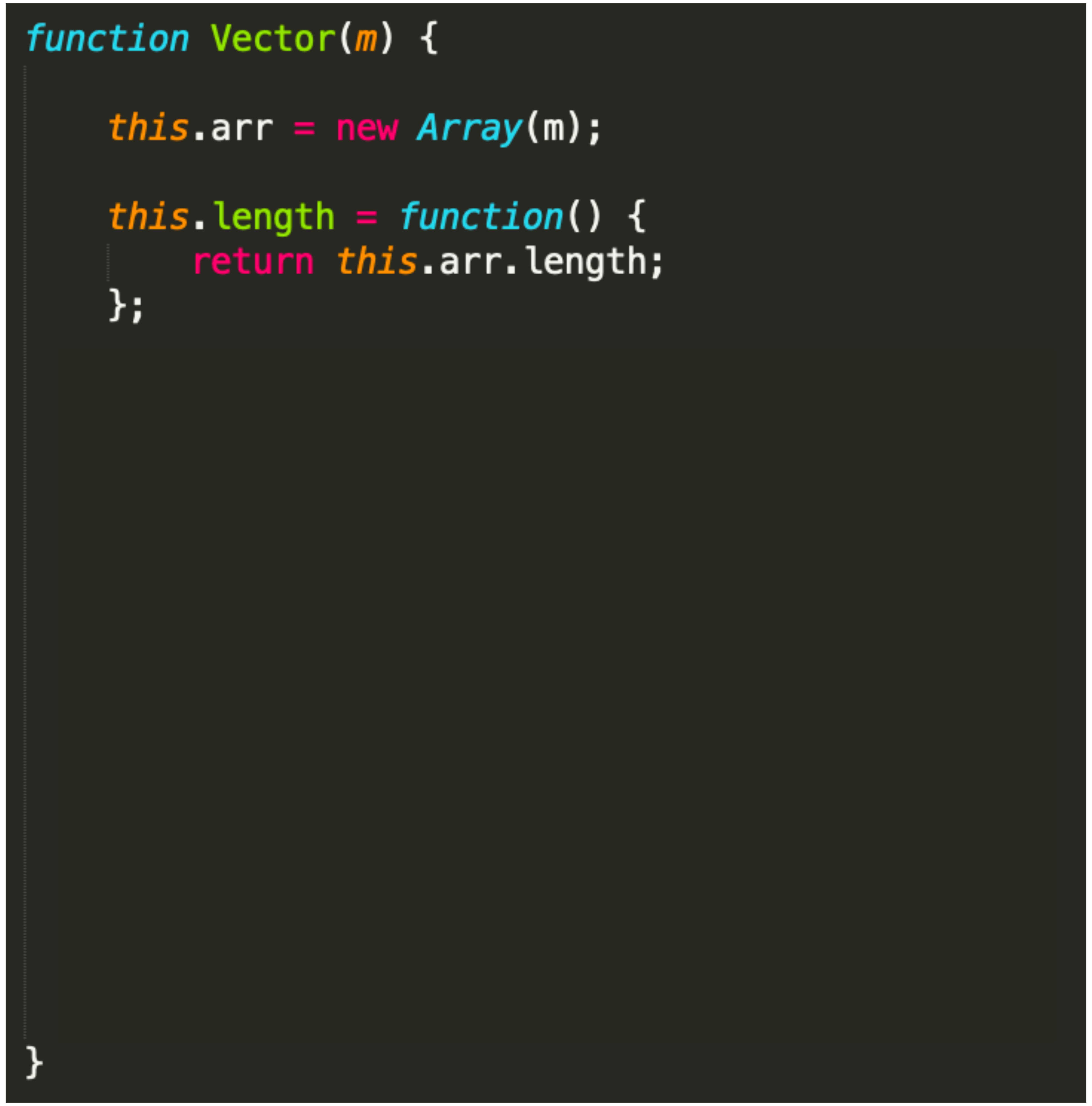
Specify length of vector m



```
function Vector(m) {

    this.arr = new Array(m);

















}
```

Vector Constructor

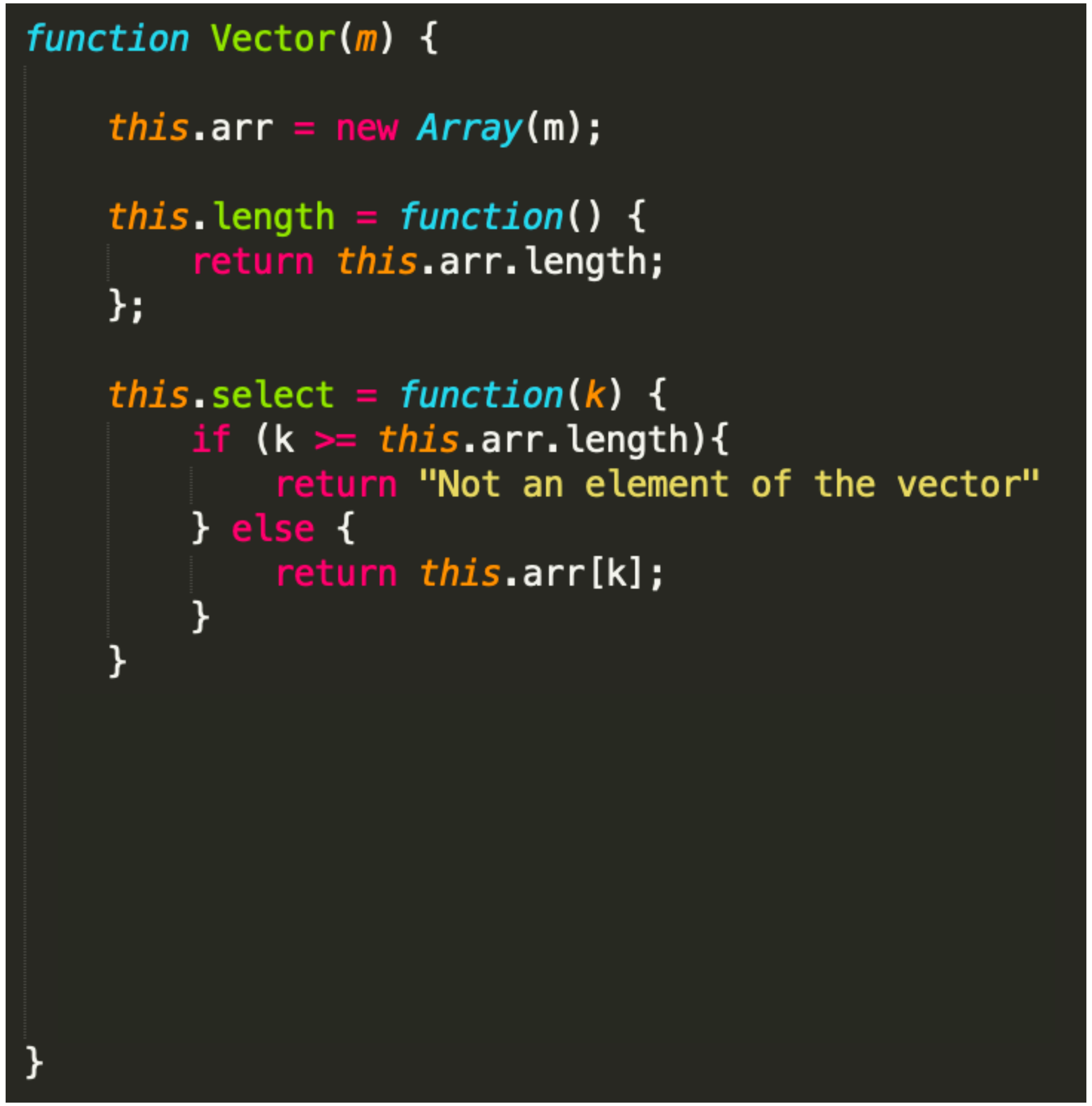
Create **method** that implements length operation



```
function Vector(m) {

    this.arr = new Array(m);

    this.length = function() {
        return this.arr.length;
    };













}
```

Vector Constructor

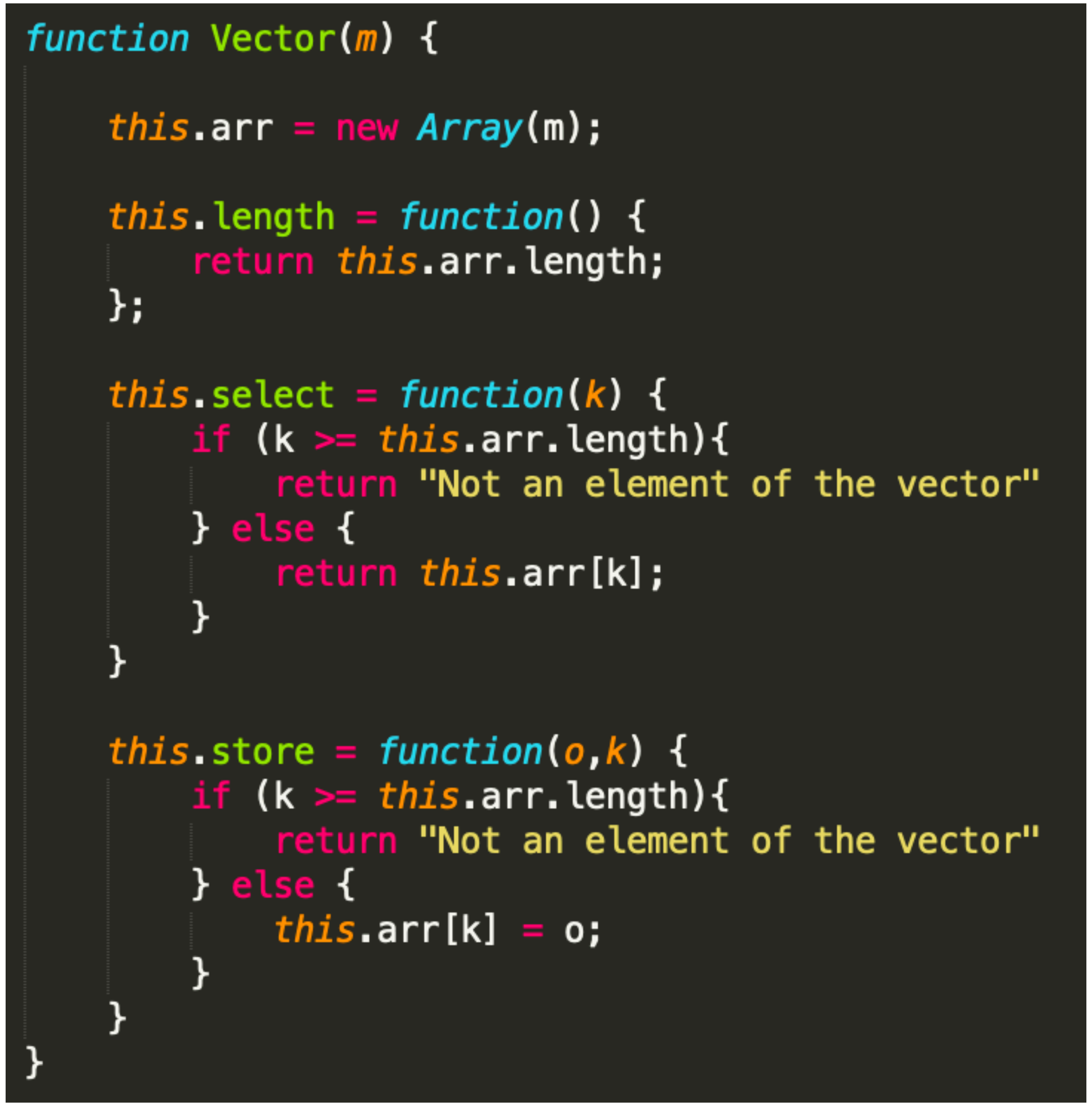
Create **method** that implements select[k] operation



```
function Vector(m) {

    this.arr = new Array(m);

    this.length = function() {
        return this.arr.length;
    };

    this.select = function(k) {
        if (k >= this.arr.length){
            return "Not an element of the vector"
        } else {
            return this.arr[k];
        }
    }









}
```

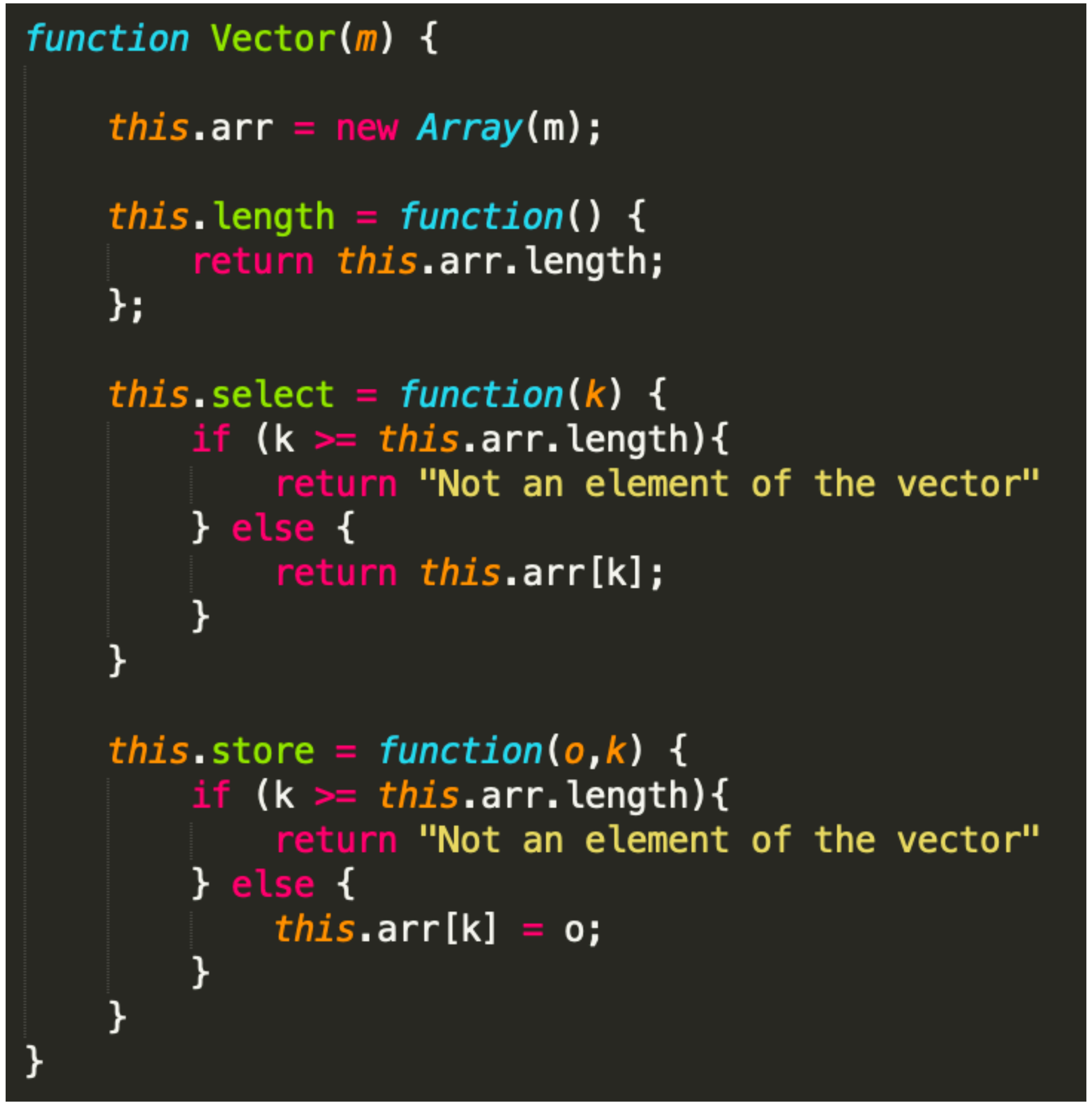
Vector Constructor

Create **method** that implements store![o,k] operation

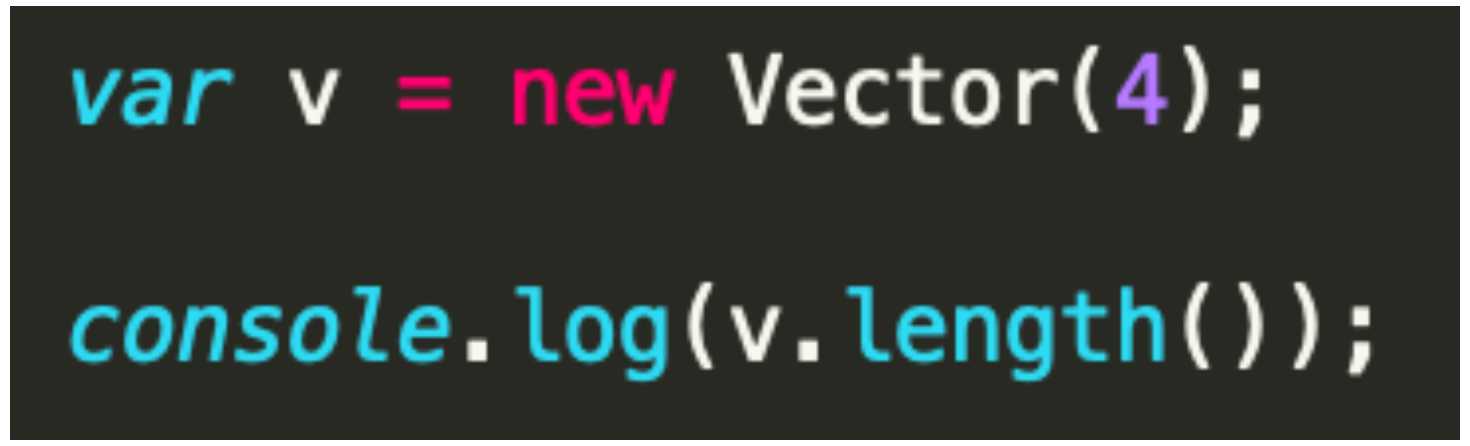


```
function Vector(m) {

    this.arr = new Array(m);

    this.length = function() {
        return this.arr.length;
    };

    this.select = function(k) {
        if (k >= this.arr.length){
            return "Not an element of the vector"
        } else {
            return this.arr[k];
        }
    }

    this.store = function(o,k) {
        if (k >= this.arr.length){
            return "Not an element of the vector"
        } else {
            this.arr[k] = o;
        }
    }
}
```

Vector Constructor



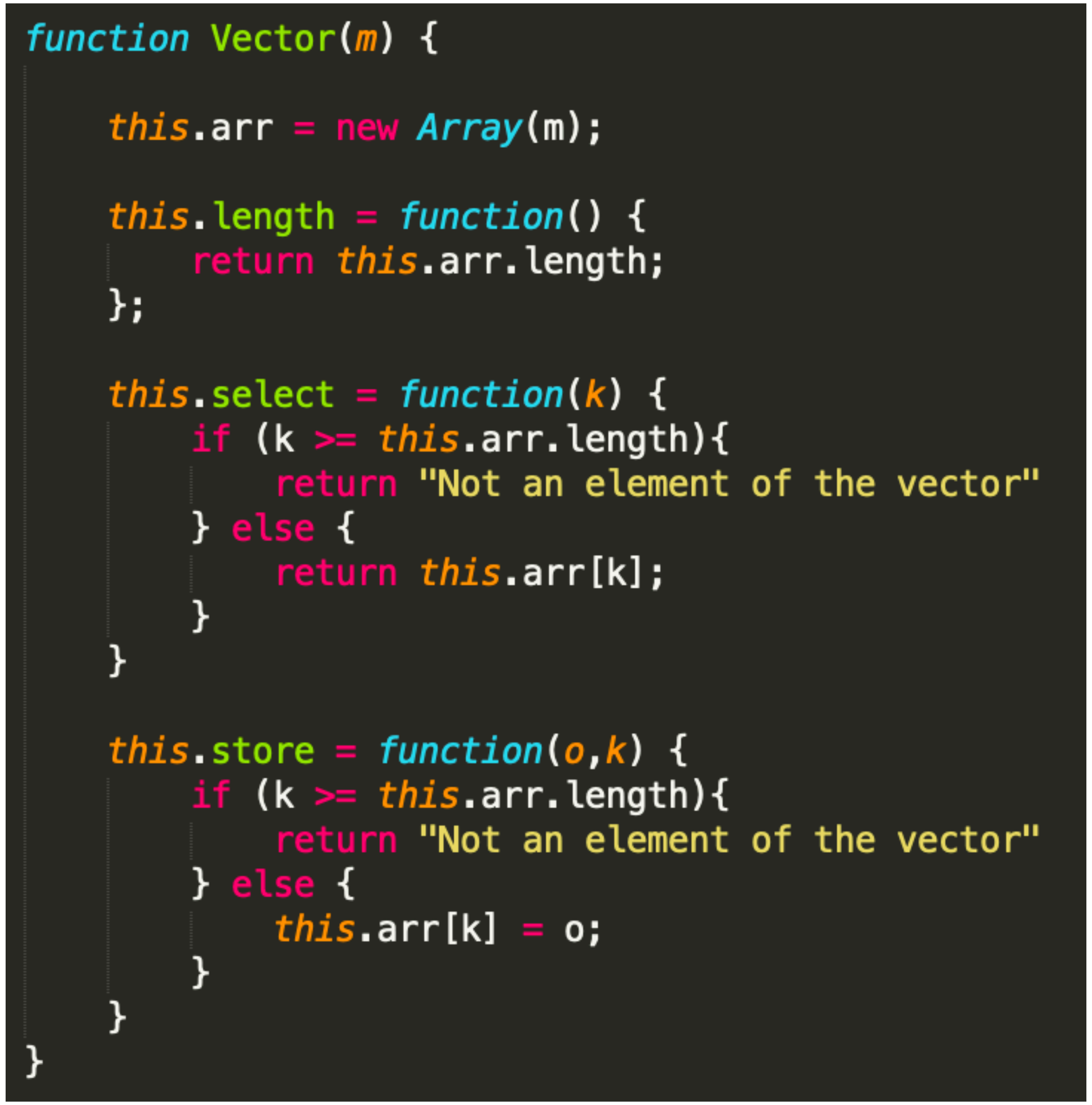
```
function Vector(m) {

    this.arr = new Array(m);

    this.length = function() {
        return this.arr.length;
    };

    this.select = function(k) {
        if (k >= this.arr.length){
            return "Not an element of the vector"
        } else {
            return this.arr[k];
        }
    }

    this.store = function(o,k) {
        if (k >= this.arr.length){
            return "Not an element of the vector"
        } else {
            this.arr[k] = o;
        }
    }
}
```



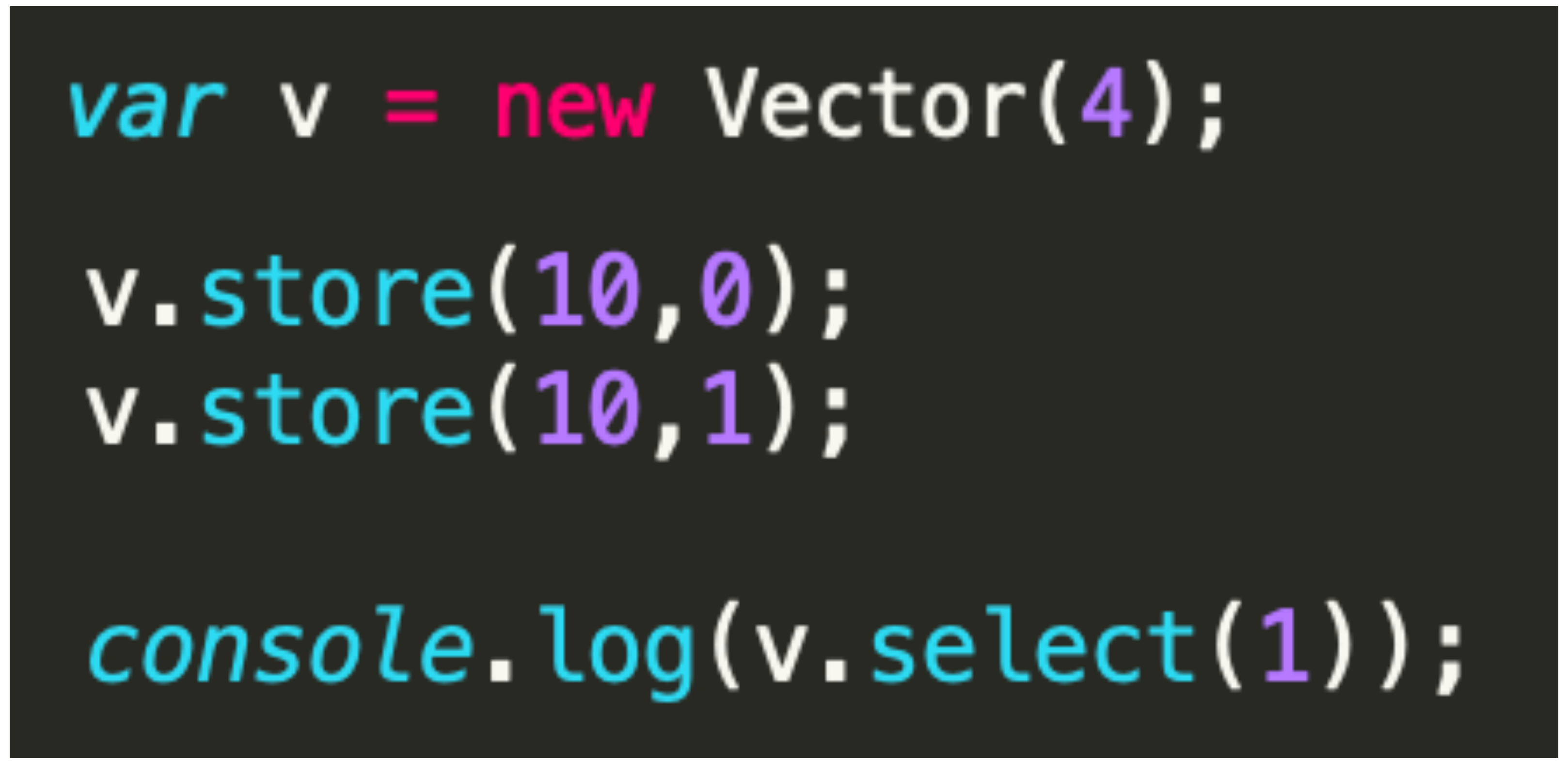
```
var v = new Vector(4);

console.log(v.length());
```

Methods need brackets

Vector Constructor



```
function Vector(m) {

    this.arr = new Array(m);

    this.length = function() {
        return this.arr.length;
    };

    this.select = function(k) {
        if (k >= this.arr.length){
            return "Not an element of the vector"
        } else {
            return this.arr[k];
        }
    }

    this.store = function(o,k) {
        if (k >= this.arr.length){
            return "Not an element of the vector"
        } else {
            this.arr[k] = o;
        }
    }
}
```

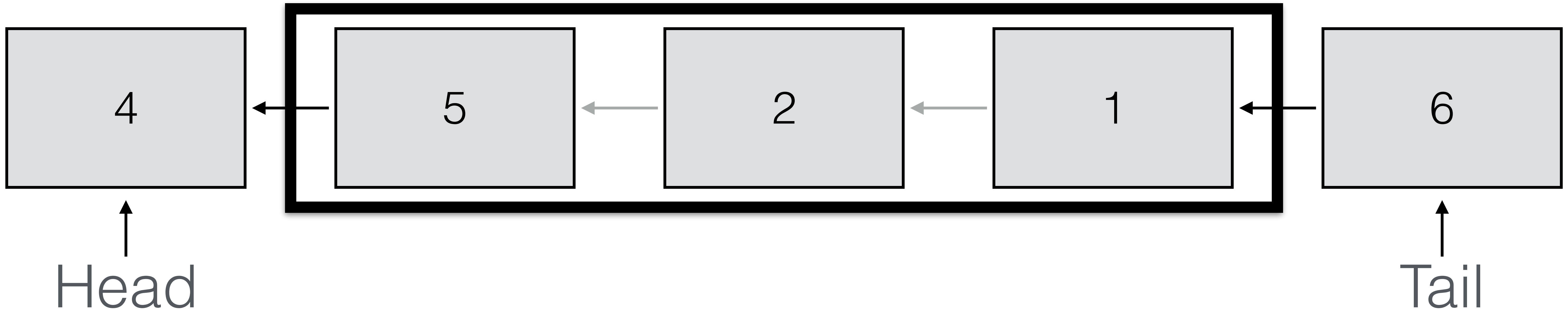


```
var v = new Vector(4);

v.store(10,0);
v.store(10,1);

console.log(v.select(1));
```

What is printed to the console?

Queue Implementation

A queue:



Queue is "flipped"

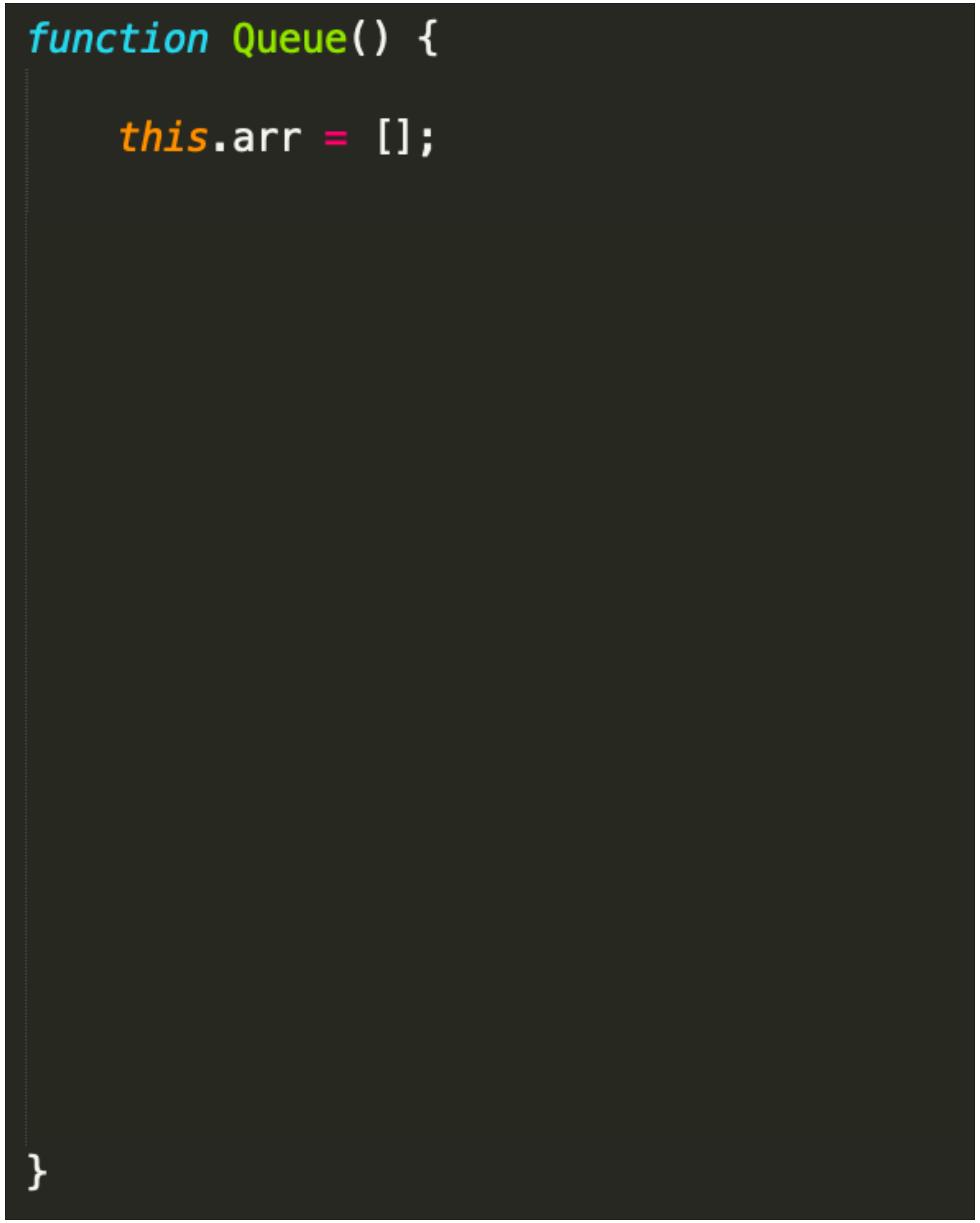
Javascript Array:

4	5	2	1	6
0	1	2	3	4

Head will be first element of array
Tail is rightmost element

Queue Constructor

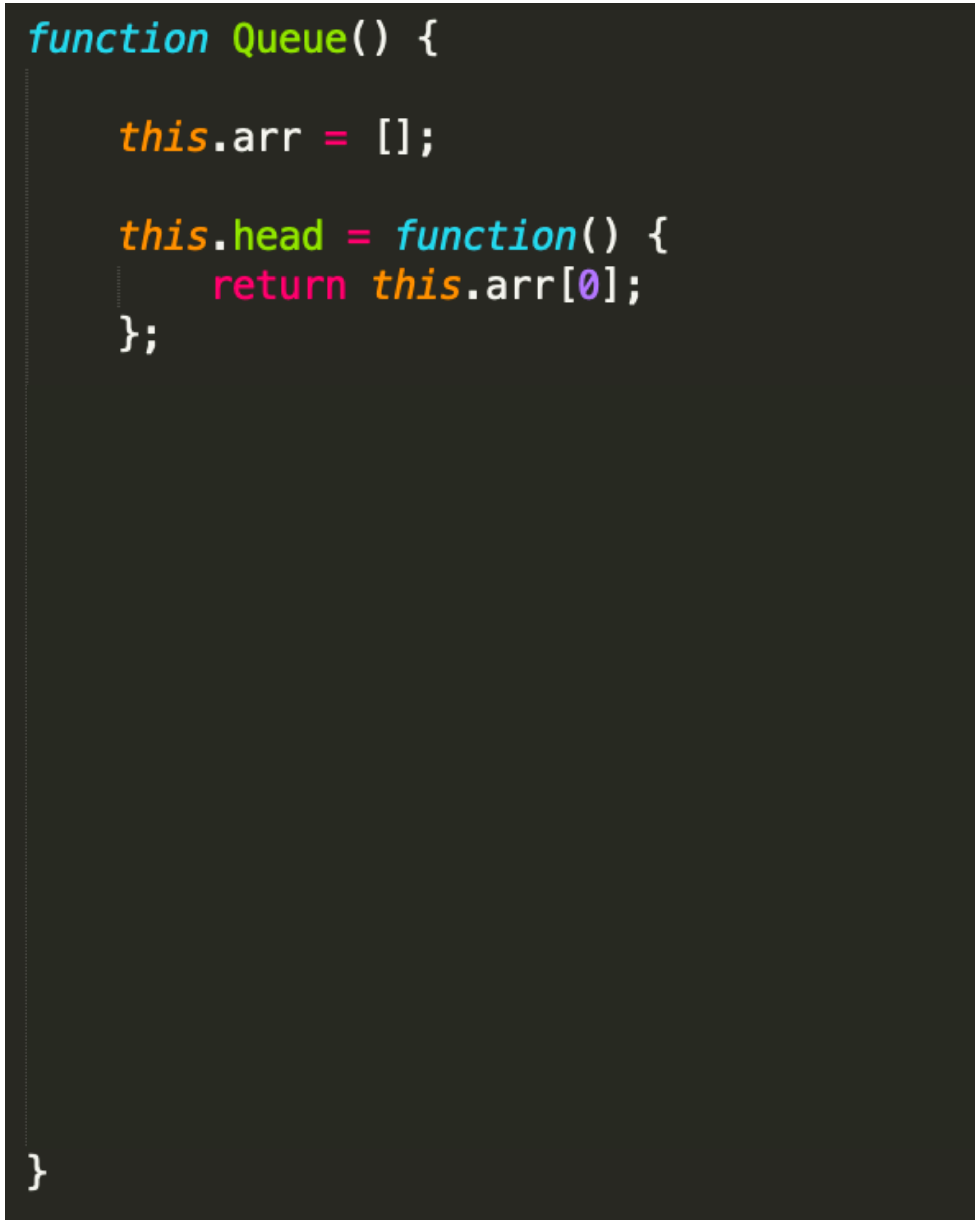
Create new empty array



```
function Queue() {

    this.arr = [];










}
```

Queue Constructor

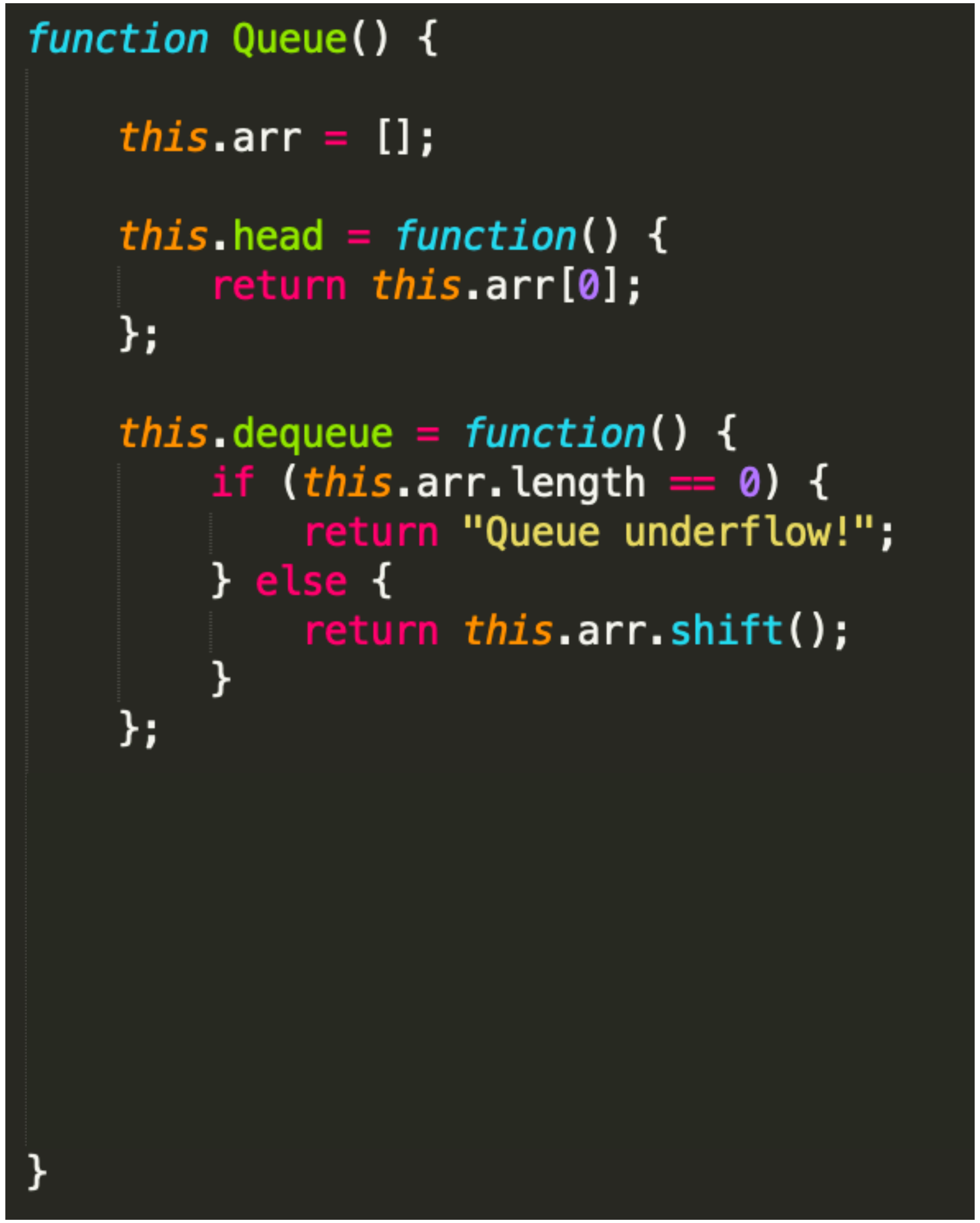
Head will be stored in **first element** of array



```
function Queue() {

    this.arr = [];

    this.head = function() {
        return this.arr[0];
    };

}
```

Queue Constructor

Shift used to remove current head

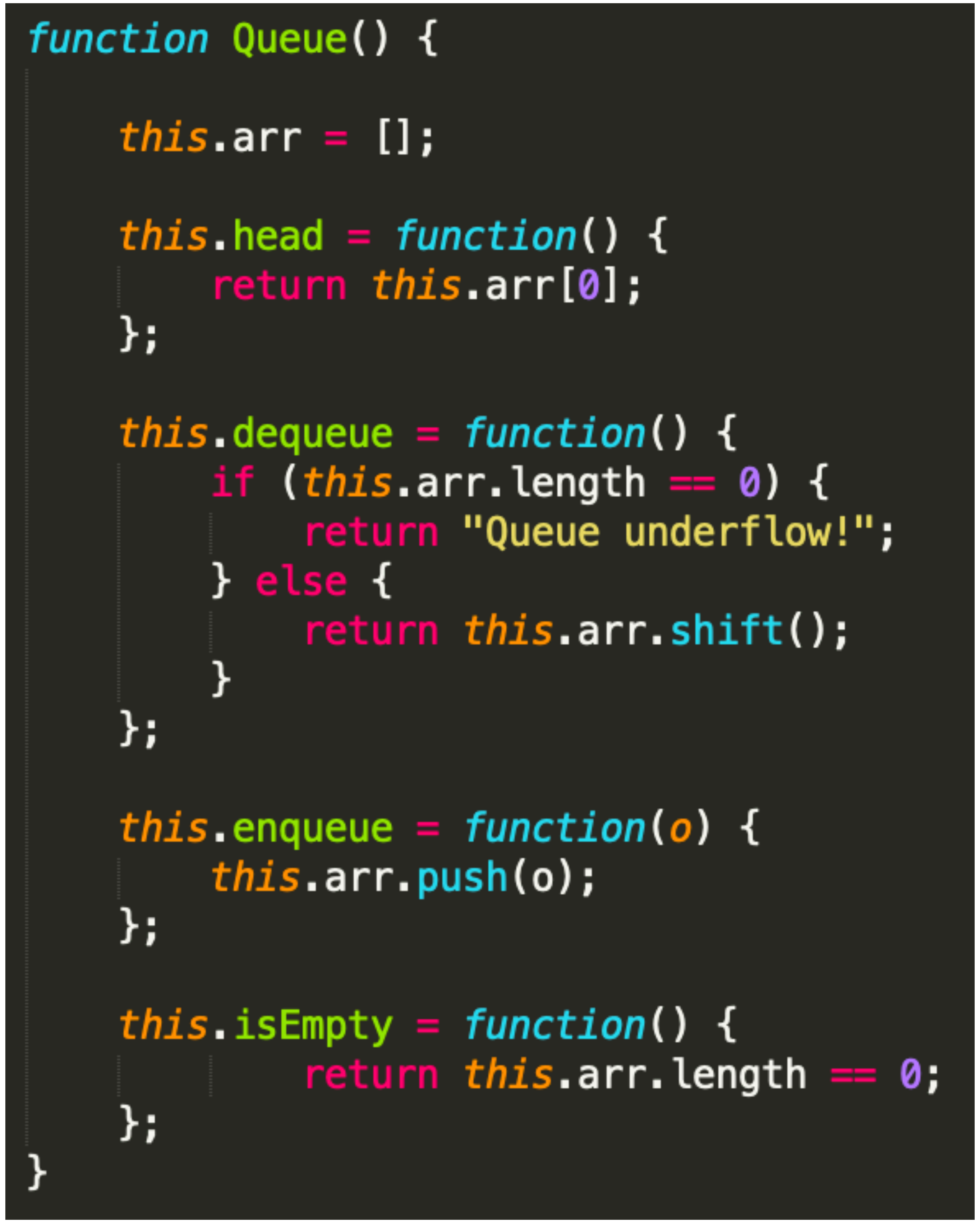


```
function Queue() {

    this.arr = [];

    this.head = function() {
        return this.arr[0];
    };

    this.dequeue = function() {
        if (this.arr.length == 0) {
            return "Queue underflow!";
        } else {
            return this.arr.shift();
        }
    };









}
```

Queue Constructor

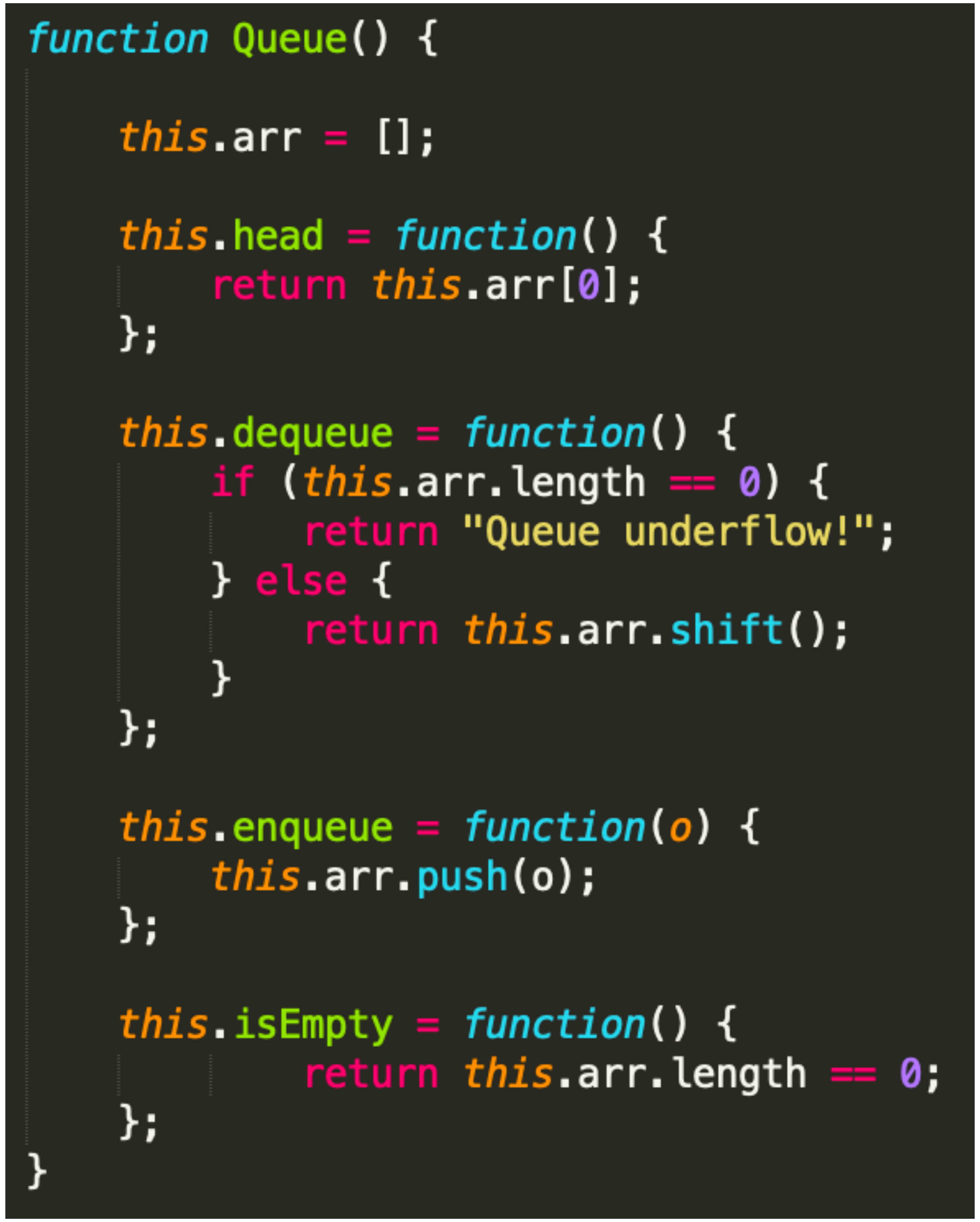
The tail of queue is in **rightmost element**

Use push to add element to right



```
function Queue() {

    this.arr = [];

    this.head = function() {
        return this.arr[0];
    };

    this.dequeue = function() {
        if (this.arr.length == 0) {
            return "Queue underflow!";
        } else {
            return this.arr.shift();
        }
    };

    this.enqueue = function(o) {
        this.arr.push(o);
    };

    this.isEmpty = function() {
        return this.arr.length == 0;
    };
}
```

Queue Constructor

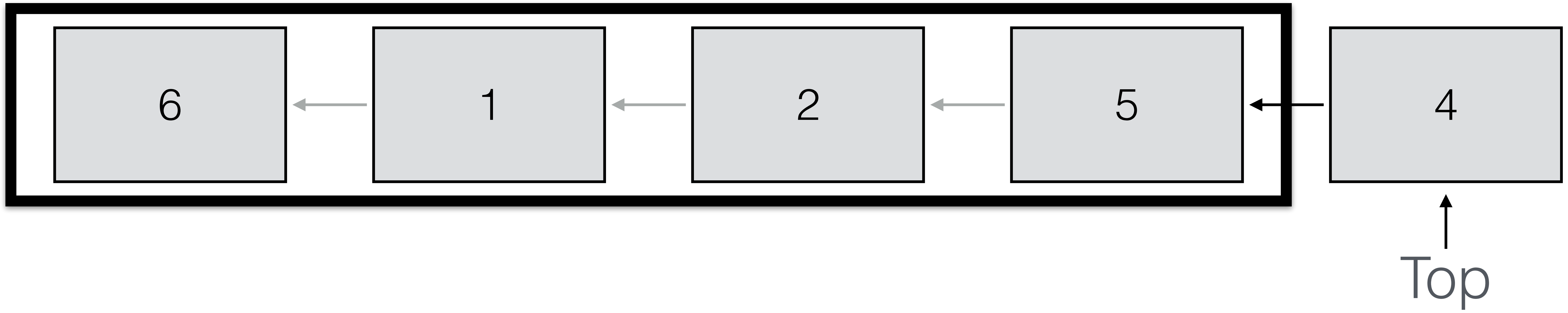


```
function Queue() {

    this.arr = [];

    this.head = function() {
        return this.arr[0];
    };

    this.dequeue = function() {
        if (this.arr.length == 0) {
            return "Queue underflow!";
        } else {
            return this.arr.shift();
        }
    };

    this.enqueue = function(o) {
        this.arr.push(o);
    };

    this.isEmpty = function() {
            return this.arr.length == 0;
    };
}
```

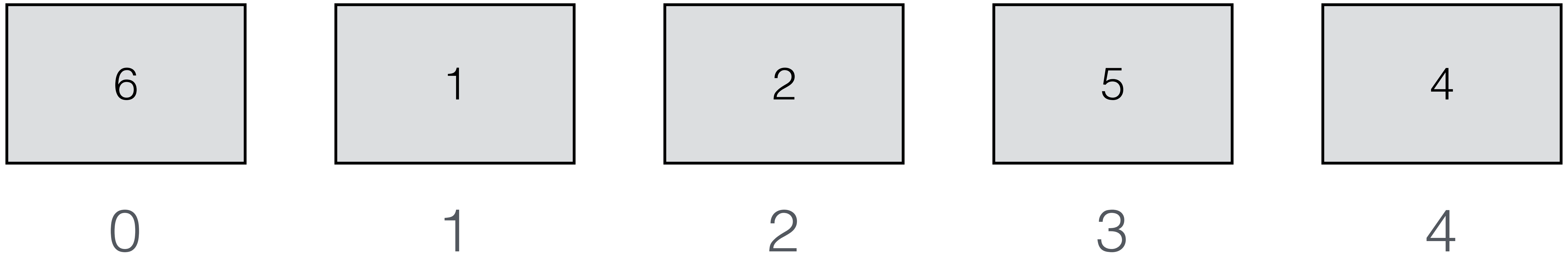
More in Worksheet 3

Stack Implementation

A stack:



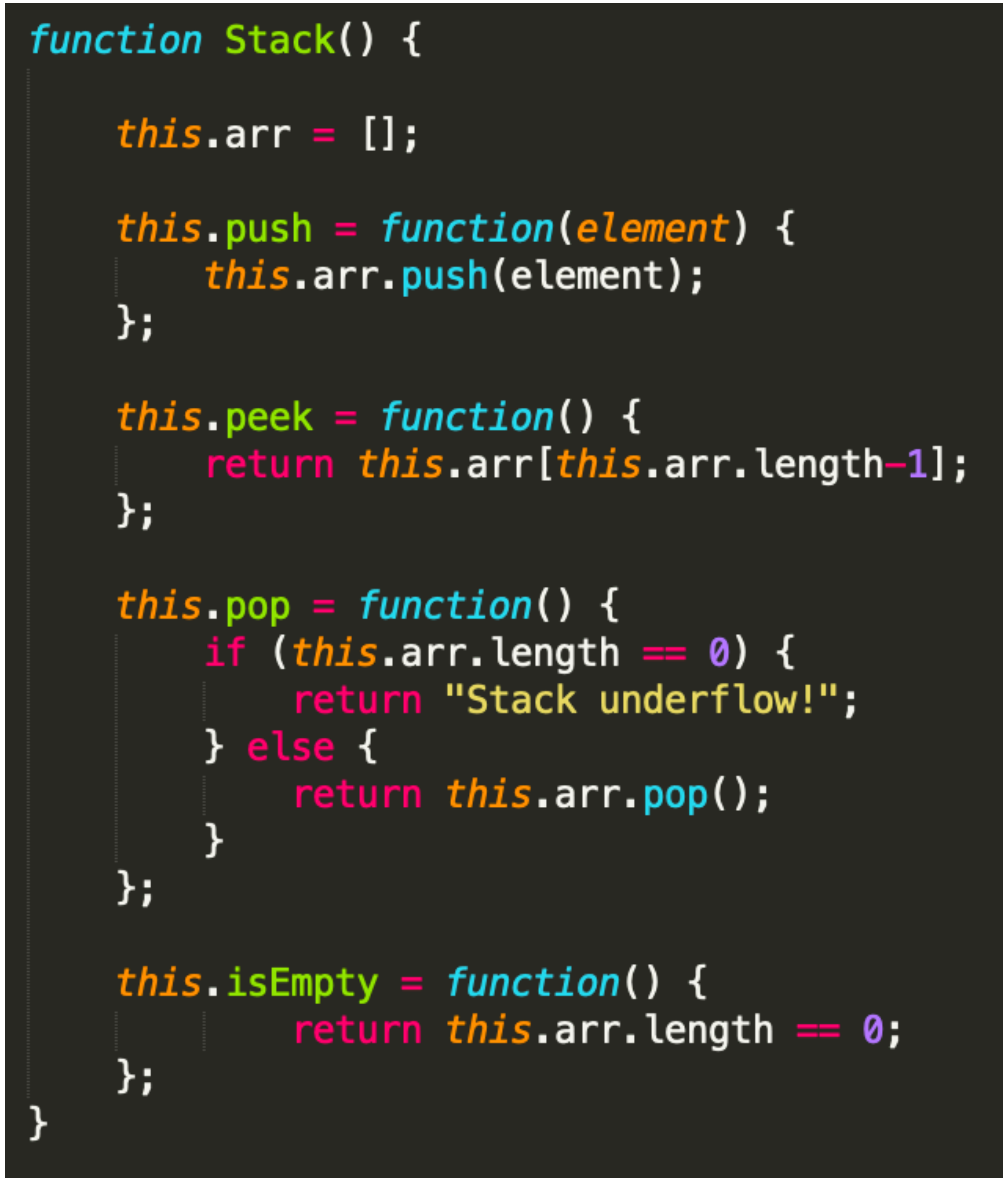
Javascript Array:



Top will be rightmost element

See Review Seminar 2

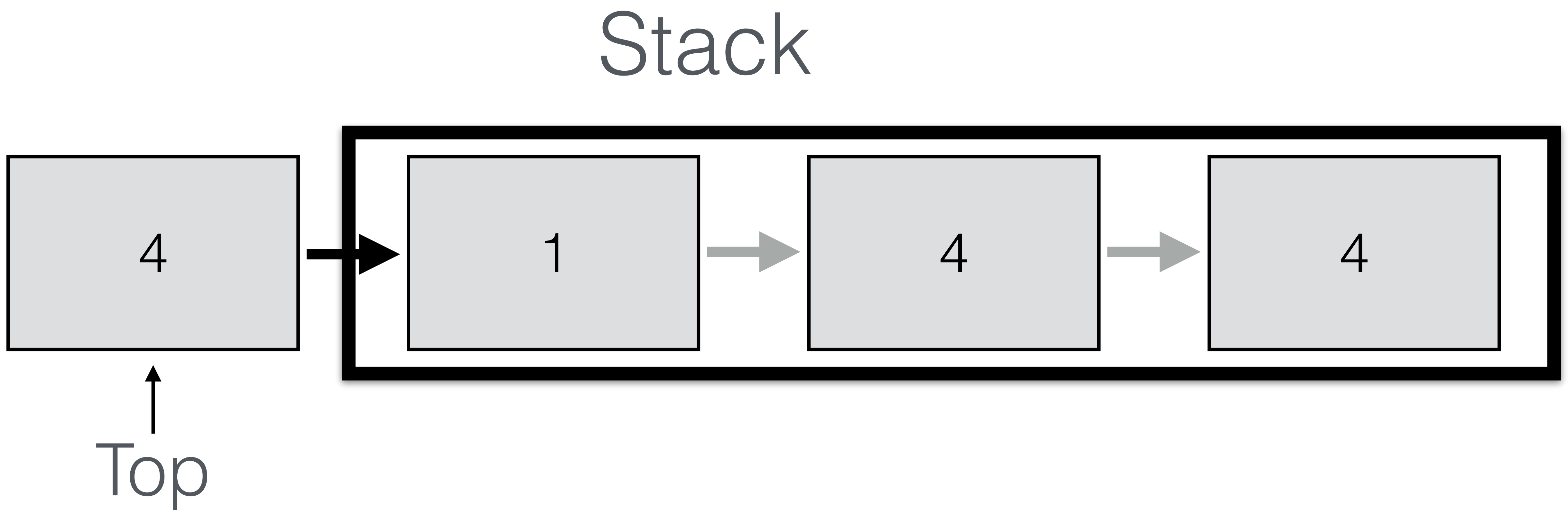
Stack Constructor



```javascript
function Stack() {

    this.arr = [];

    this.push = function(element) {
        this.arr.push(element);
    };

    this.peek = function() {
        return this.arr[this.arr.length-1];
    };

    this.pop = function() {
        if (this.arr.length == 0) {
            return "Stack underflow!";
        } else {
            return this.arr.pop();
        }
    };

    this.isEmpty = function() {
            return this.arr.length == 0;
    };
}
```

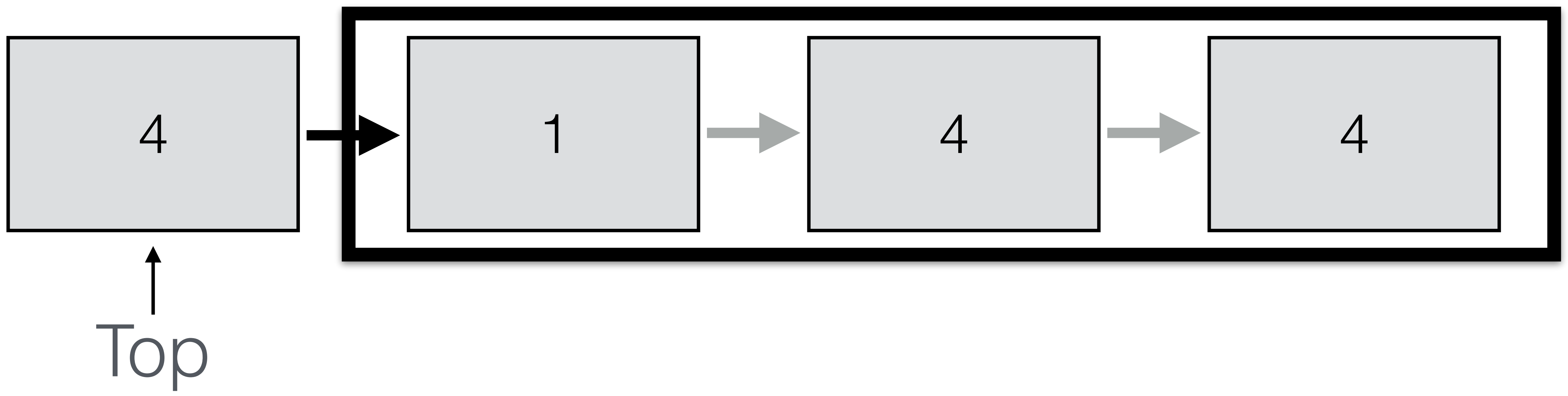
Stacks can be implemented in other ways…

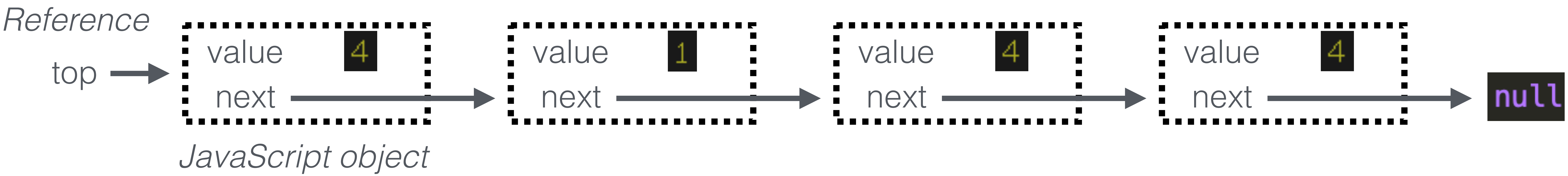
I want to emphasise that stacks are not the same as JavaScript Arrays



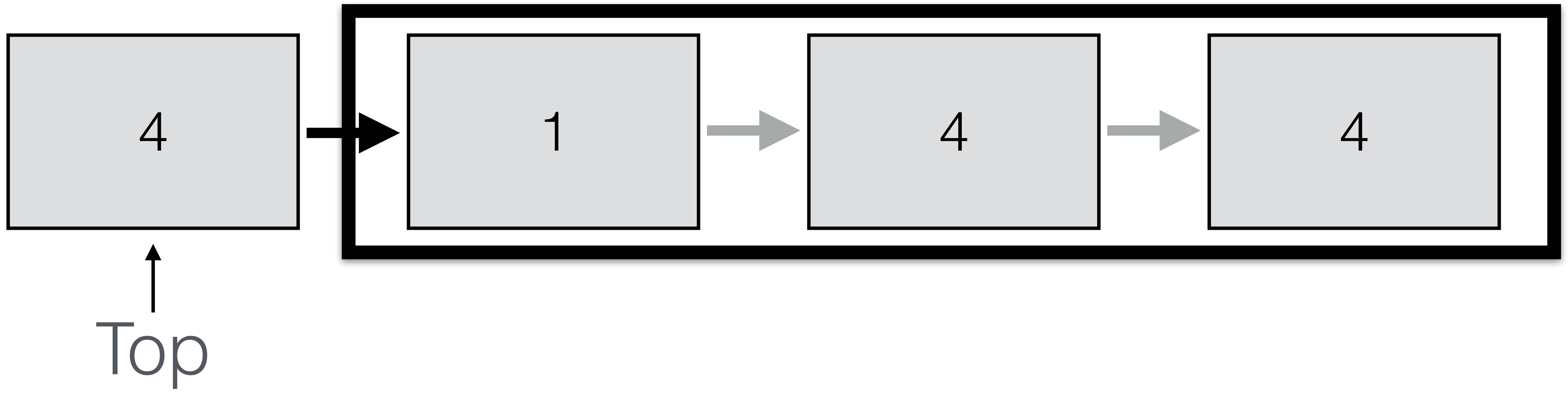
We can create a stack directly from smaller objects

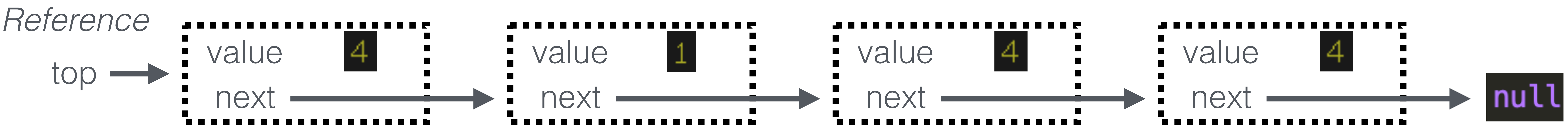
Stack


4
1
4
4
Top

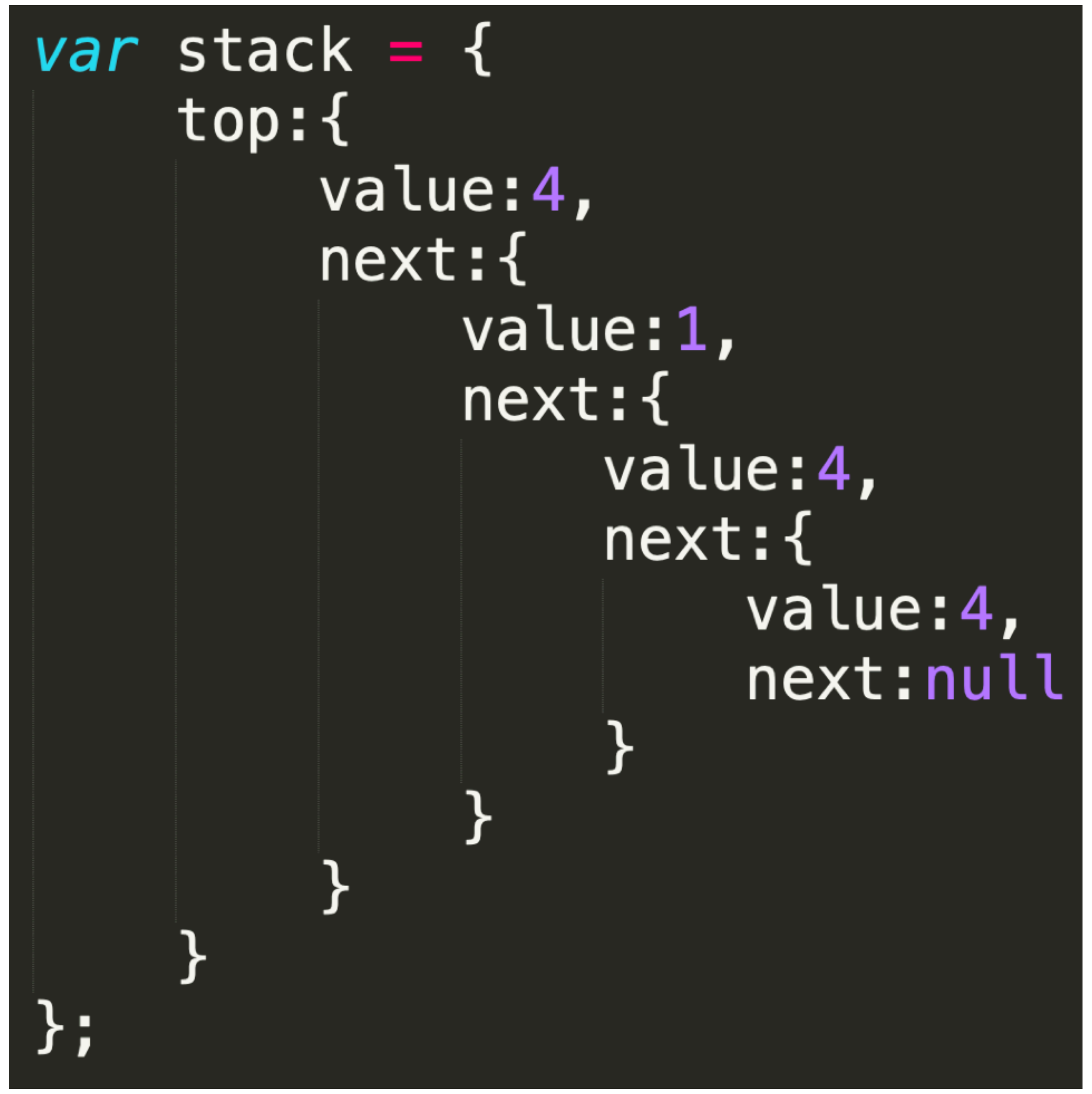

Reference
top
value
4
next
value
1
next
value
4
next
value
4
next
null
JavaScript object

Stack

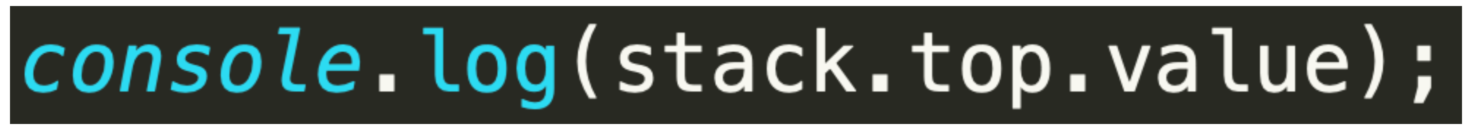




JavaScript object



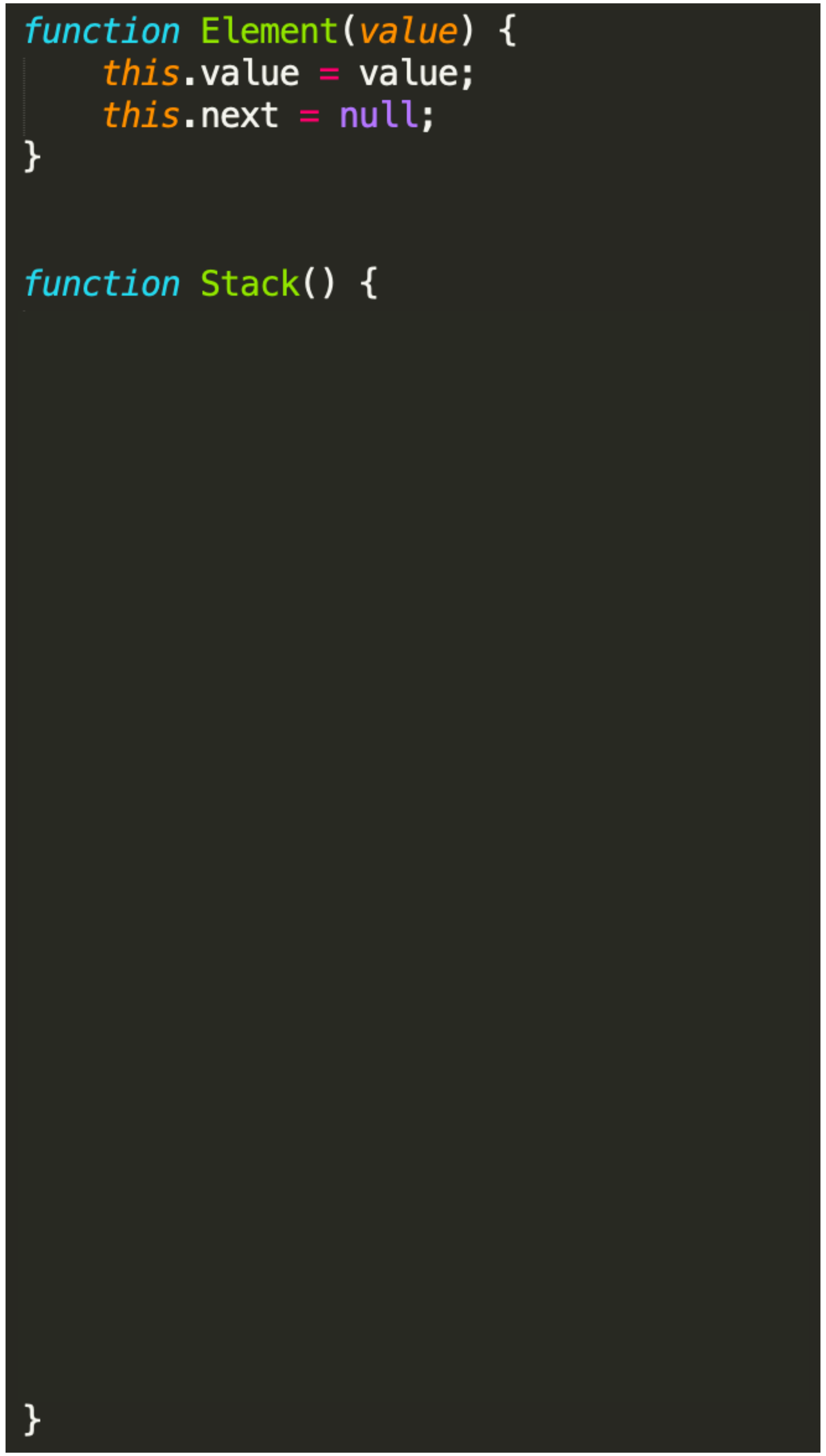
```
var stack = {
    top:{
        value:4,
        next:{
            value:1,
            next:{
                value:4,
                next:{
                    value:4,
                    next:null
                }
            }
        }
    }
};
```



```
console.log(stack.top.value);
```

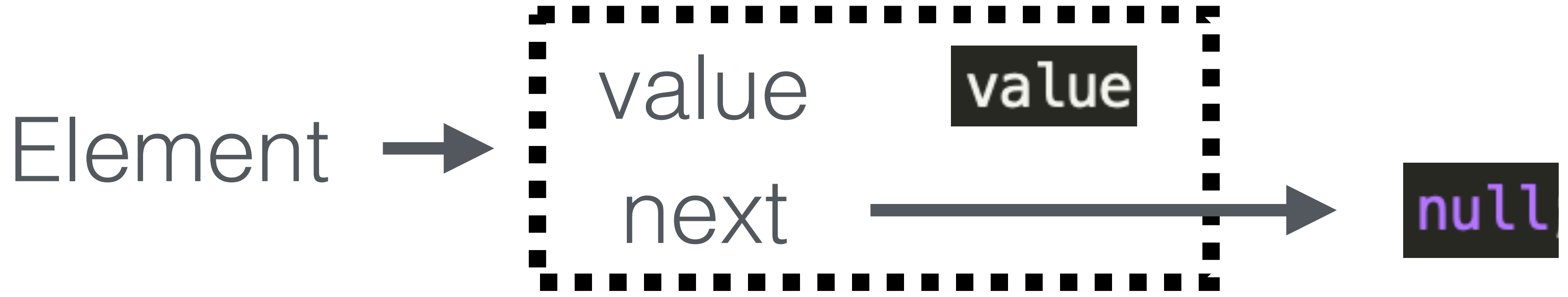
4

Value stored in top element

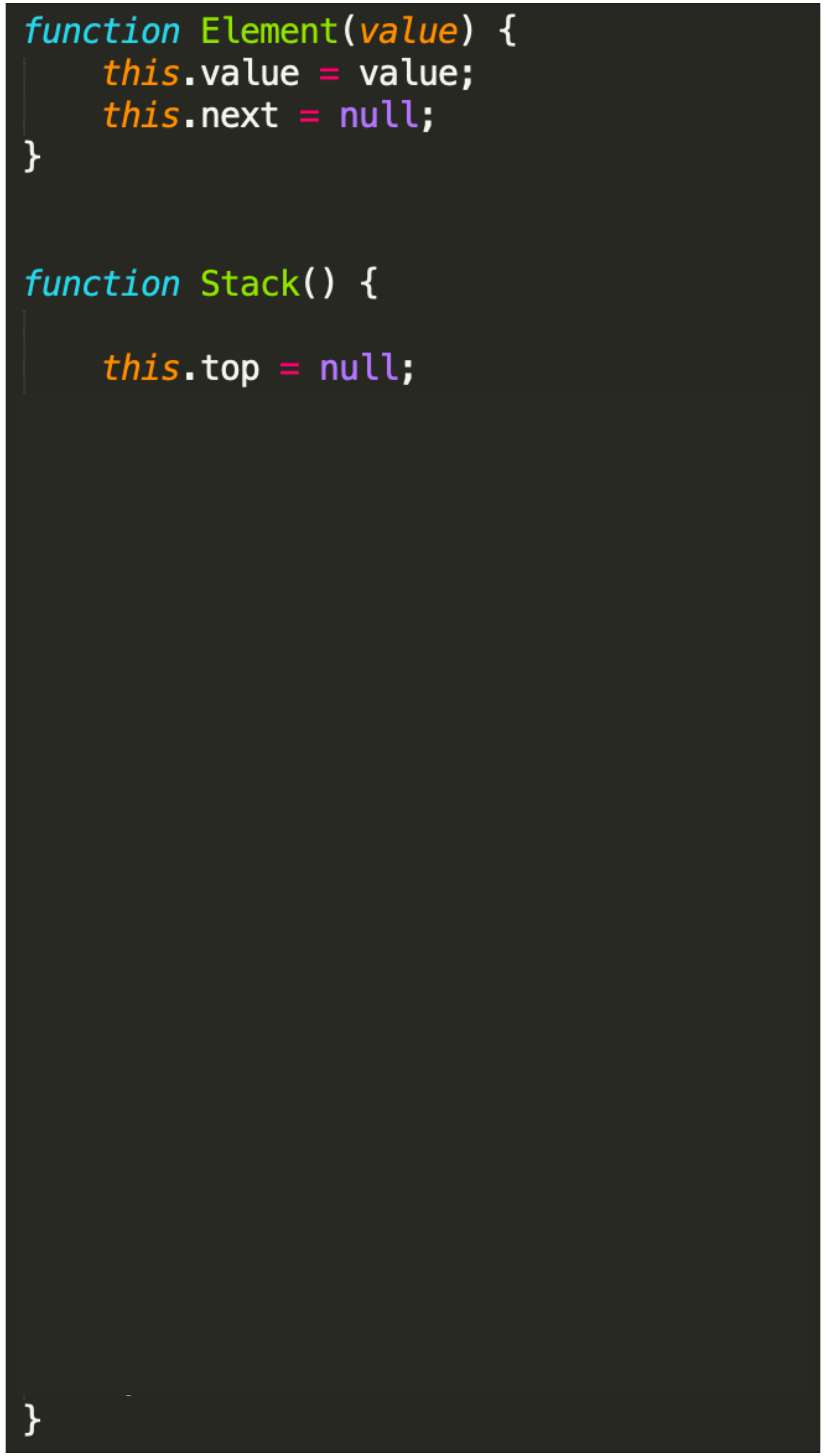


```
function Element(value) {
    this.value = value;
    this.next = null;
}


function Stack() {





















}
```

This constructor makes Elements of stack

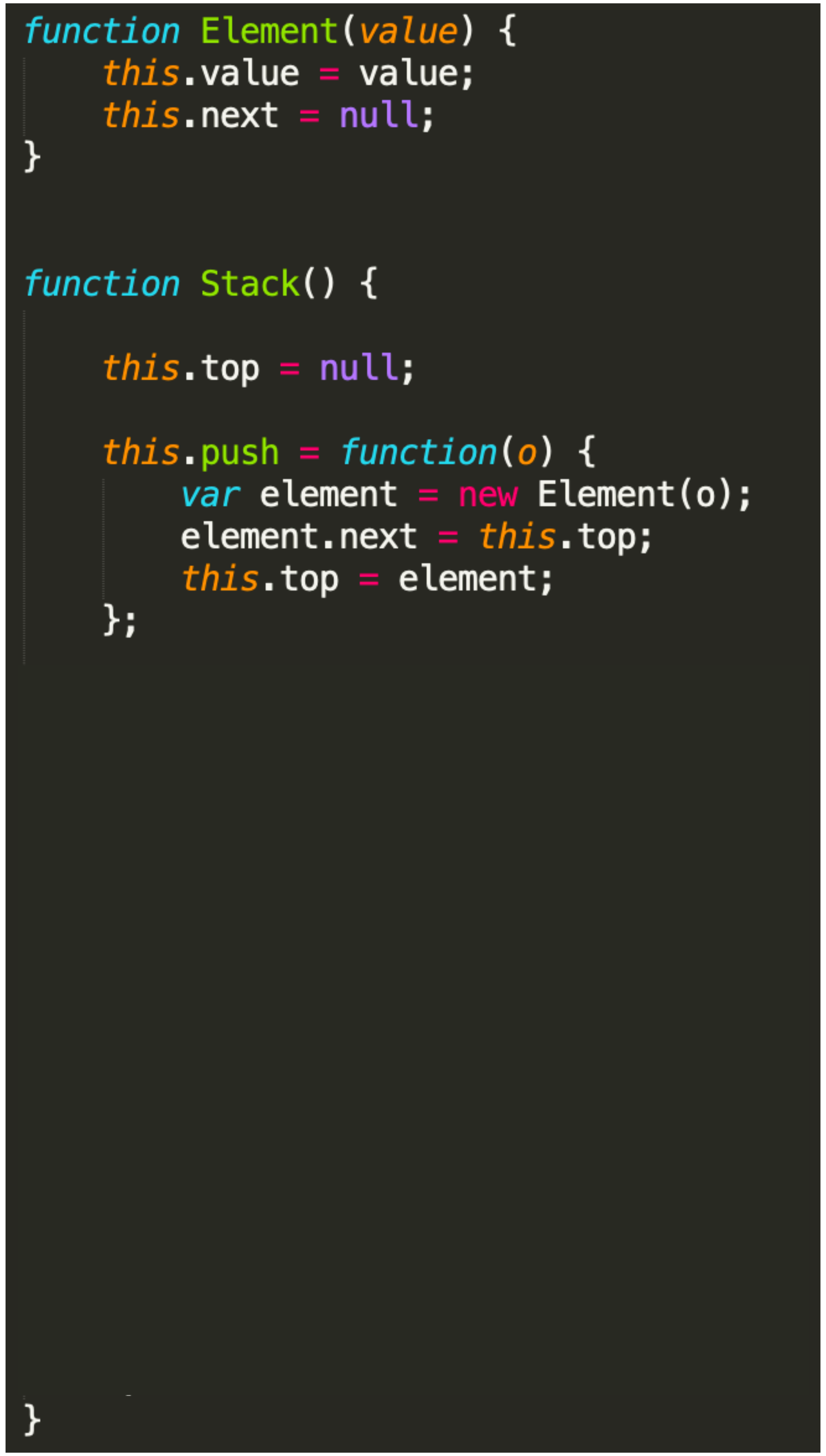


next will tell us *where* the next element is



```
function Element(value) {
    this.value = value;
    this.next = null;
}


function Stack() {

    this.top = null;

}
```

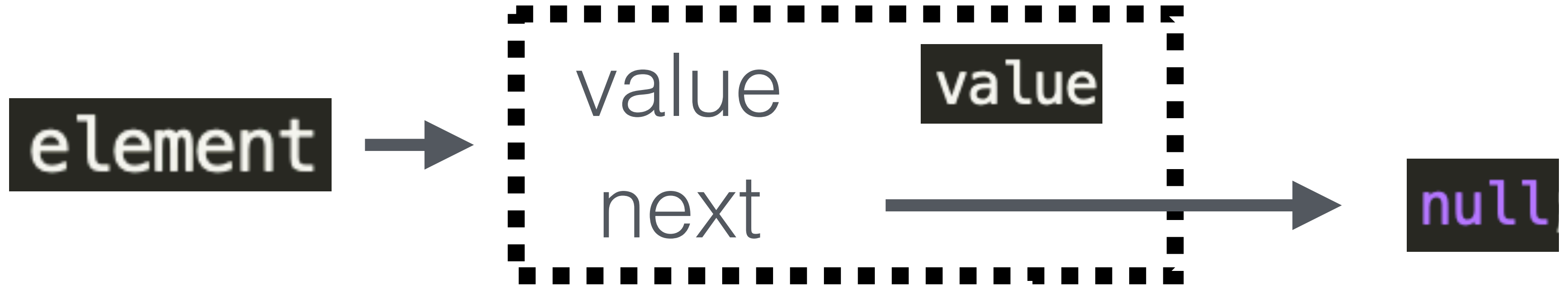
When we create new **empty stack**, top points to nothing



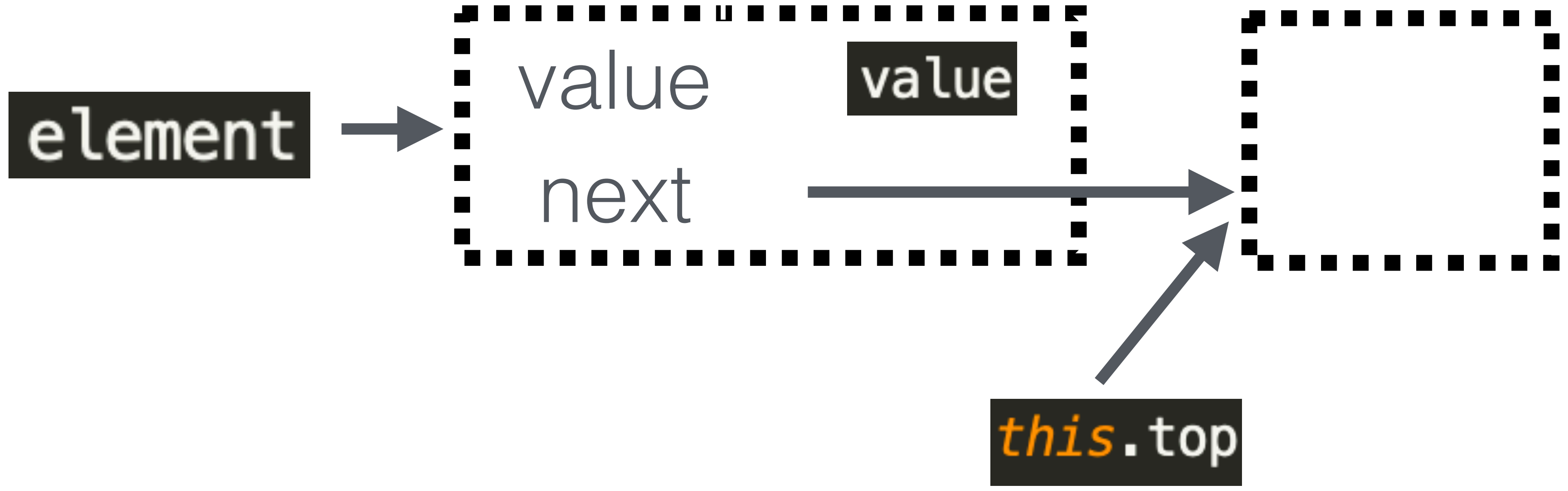
```
function Element(value) {
    this.value = value;
    this.next = null;
}


function Stack() {

    this.top = null;

    this.push = function(o) {
        var element = new Element(o);
        element.next = this.top;
        this.top = element;
    };












}
```

To push a new element:

1. Create element

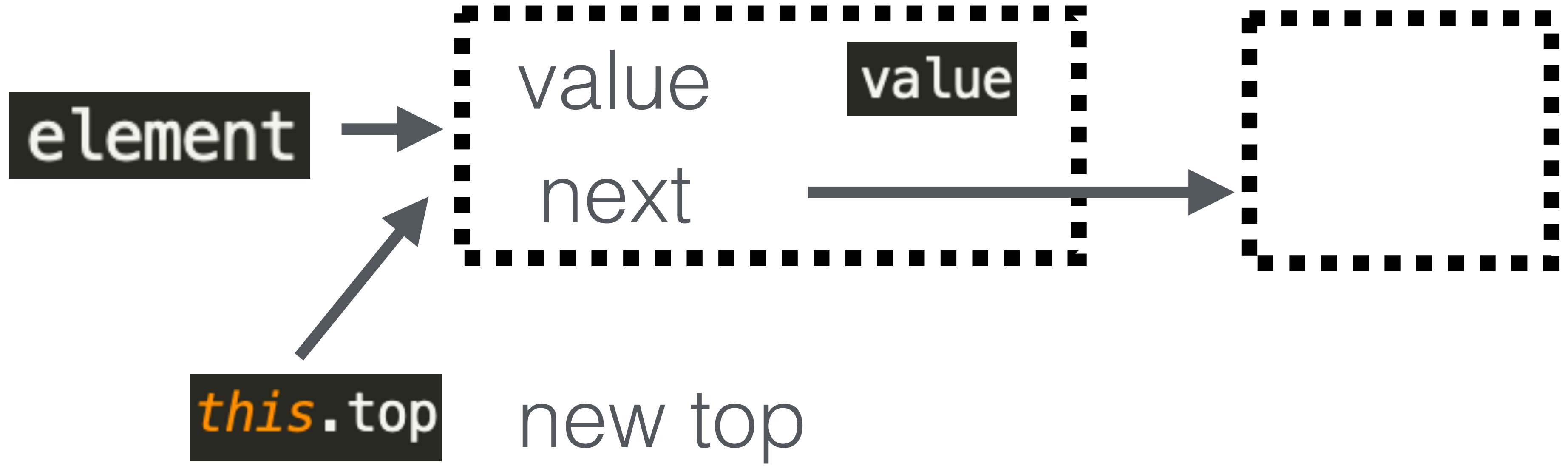


2. Have the *current* top assigned next of element

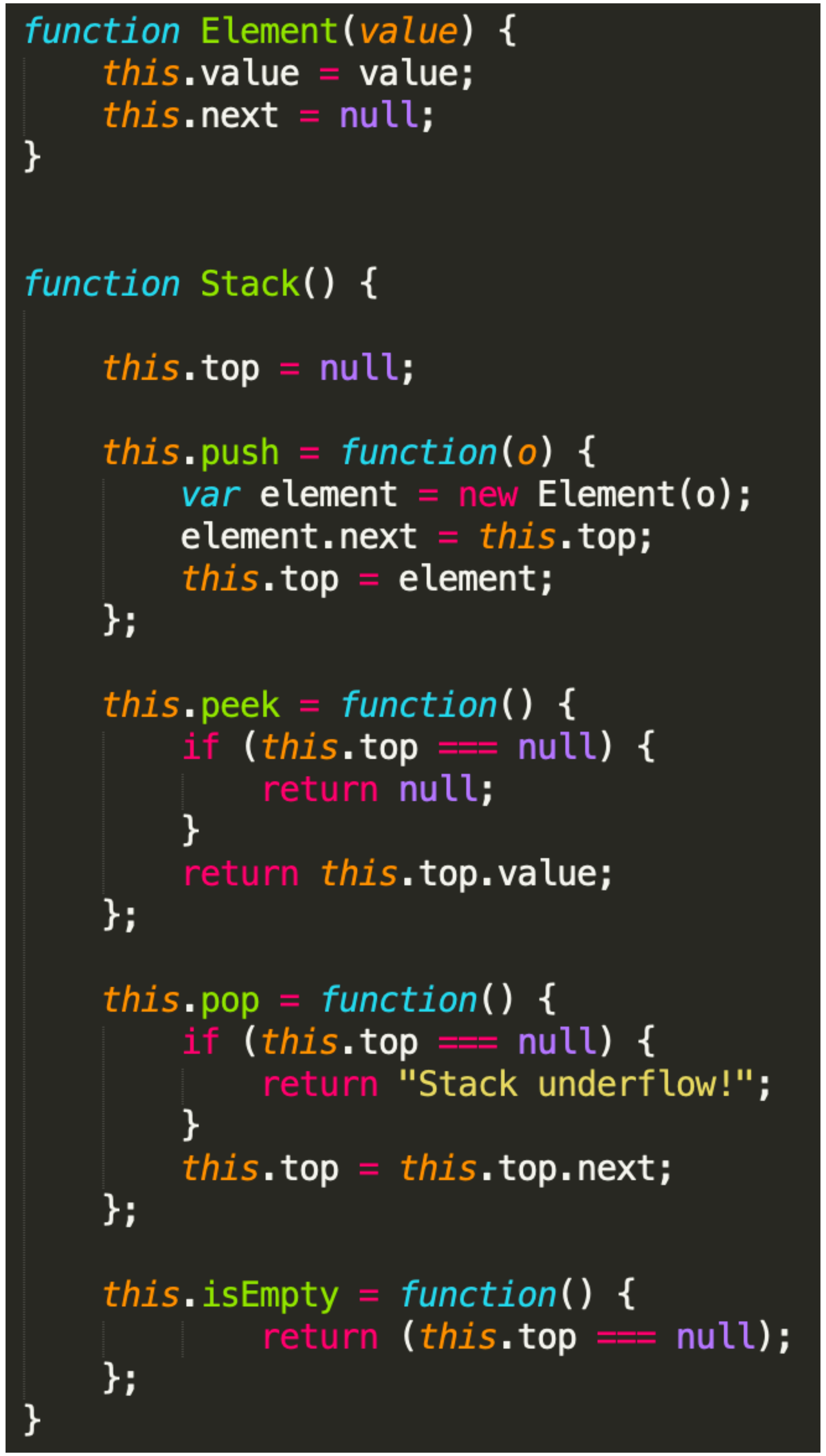


top of current stack

3. Update the top to be new element



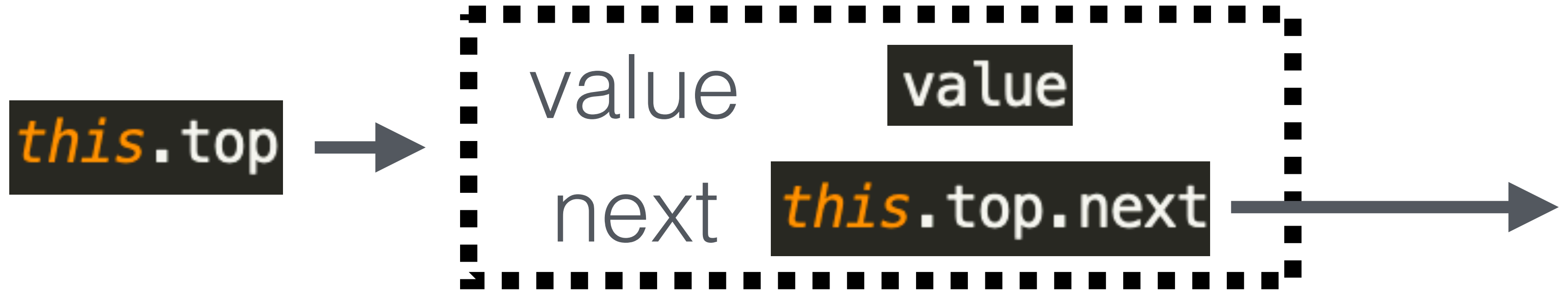
new top



```
function Element(value) {
    this.value = value;
    this.next = null;
}


function Stack() {

    this.top = null;

    this.push = function(o) {
        var element = new Element(o);
        element.next = this.top;
        this.top = element;
    };

    this.peek = function() {
        if (this.top === null) {
            return null;
        }
        return this.top.value;
    };

    this.pop = function() {
        if (this.top === null) {
            return "Stack underflow!";
        }
        this.top = this.top.next;
    };

    this.isEmpty = function() {
        return (this.top === null);
    };
}
```

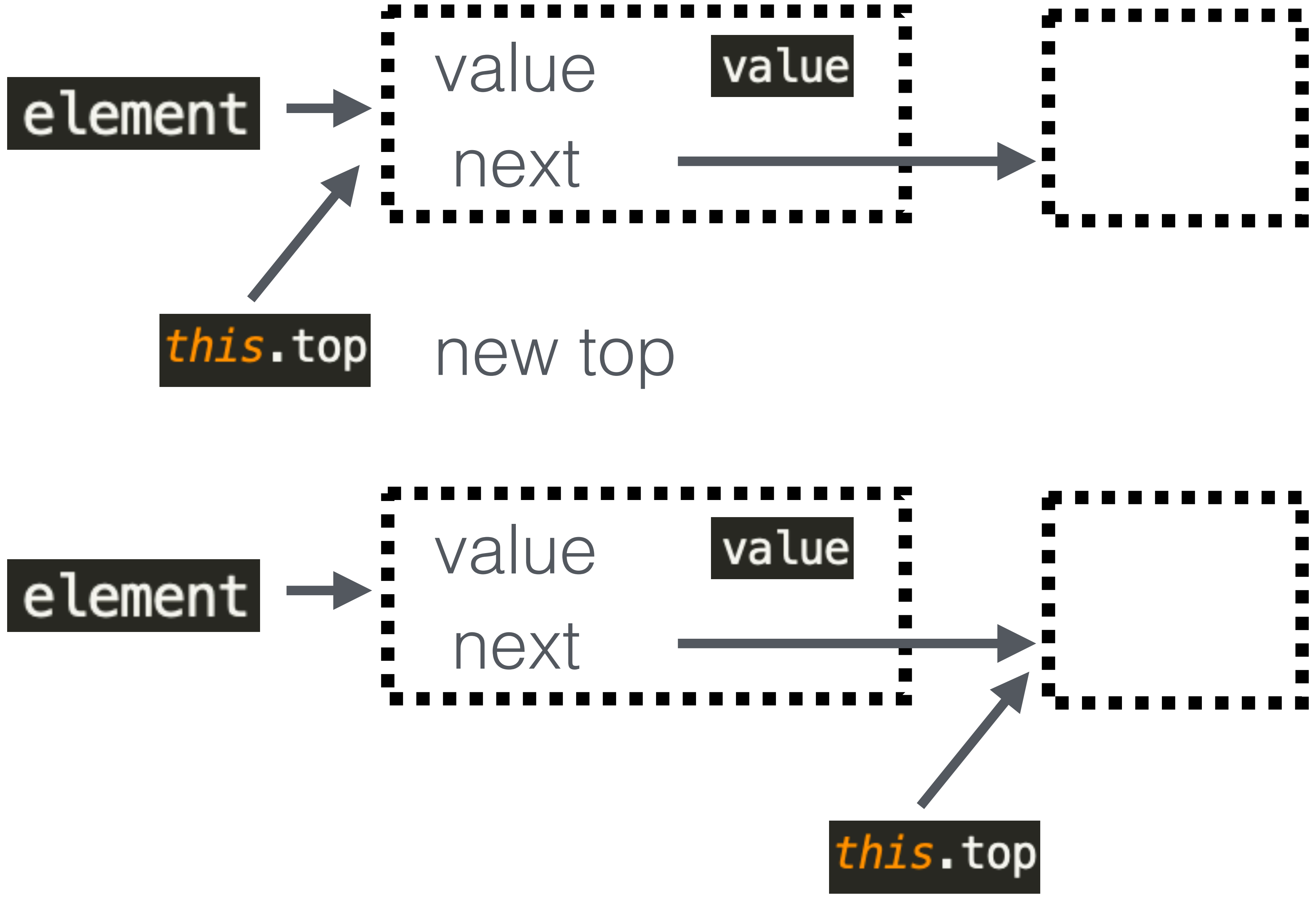
To peek:

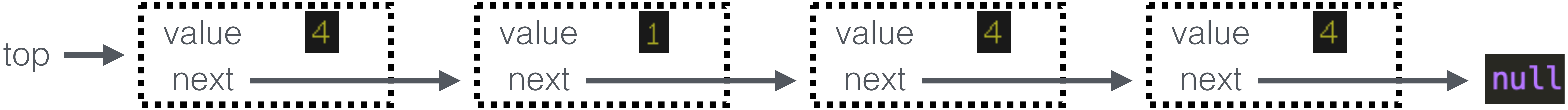
Look at the value of top

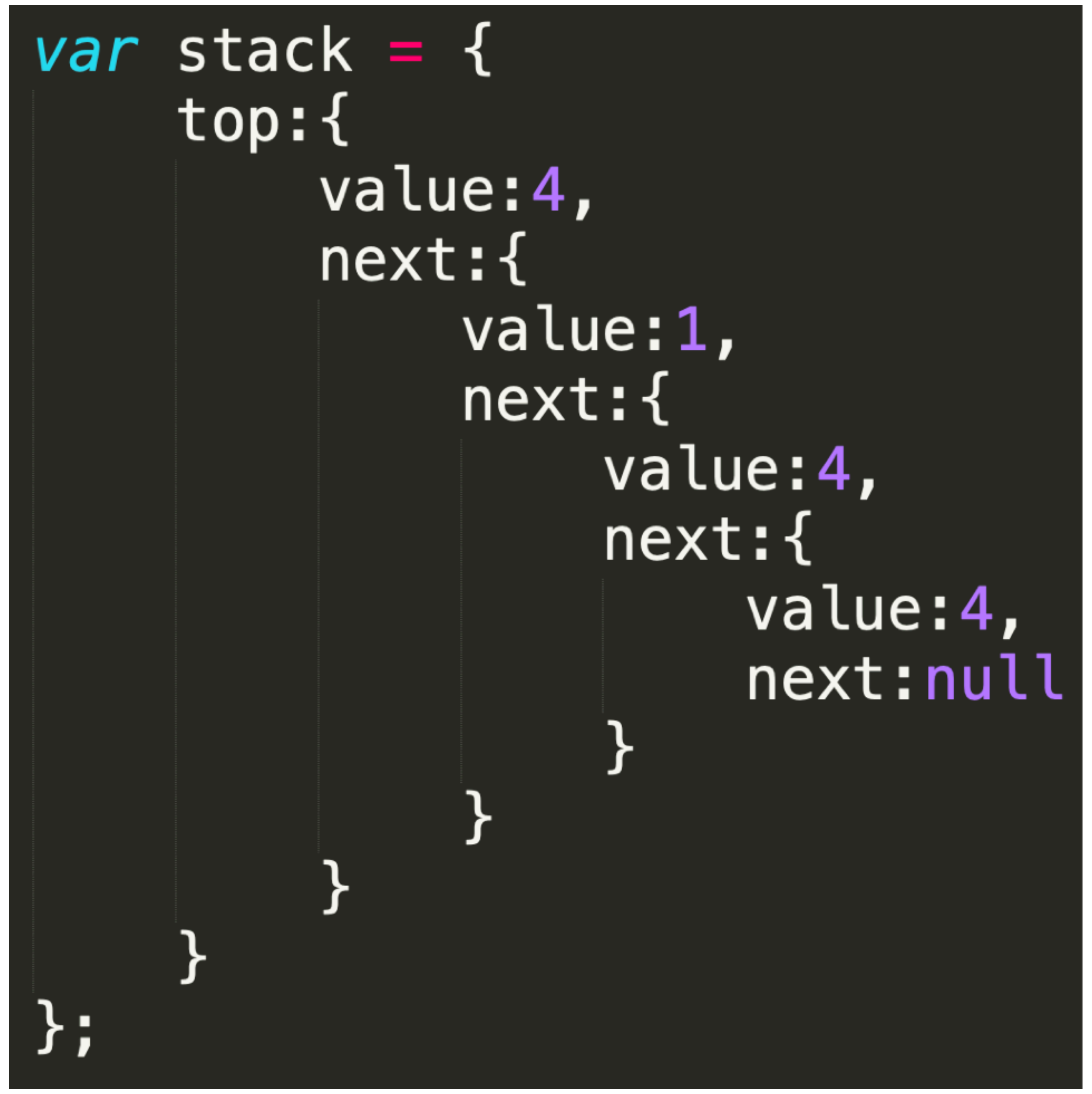


To pop:

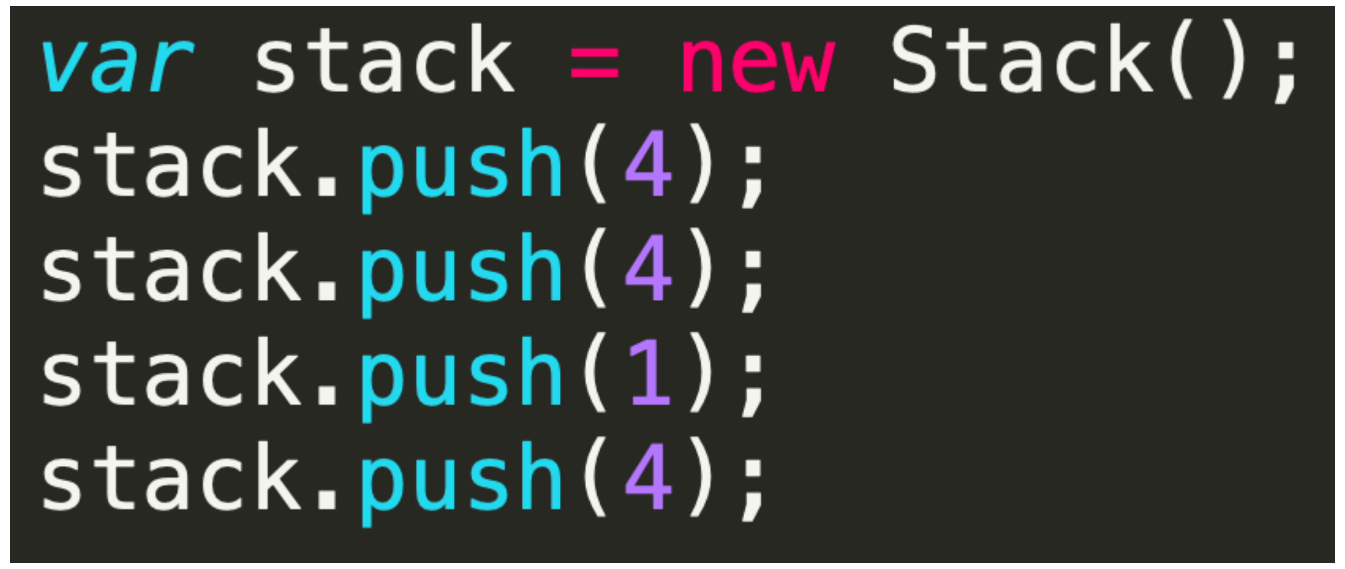
“Reverse” last step of push







```
var stack = {
    top:{
        value:4,
        next:{
            value:1,
            next:{
                value:4,
                next:{
                    value:4,
                    next:null
                }
            }
        }
    }
};
```



```
var stack = new Stack();
stack.push(4);
stack.push(4);
stack.push(1);
stack.push(4);
```

Constructors and methods can hide gory details

Stack demos

Application of a stack: palindromes

http://igor.doc.gold.ac.uk/~afior002/palindrome_stack/index.html

Another nice application of a stack: checking brackets

http://igor.doc.gold.ac.uk/~afior002/balanced_stack/index.html

Problem 2:

Given 48 toys and 42 sweets. Most number of guests invited such that all toys and sweets are distributed equally?

6

Now the guests have siblings:

Guest 1: 0 siblings
Guest 2: 2 siblings
Guest 3: 1 sibling
Guest 4: 0 siblings
Guest 5: 1 sibling
Guest 6: 1 sibling

Maximum number of guests where **6 people in total** (both guests and siblings) get toys and sweets?

Try writing some JavaScript code!

Problem 2:

Given 48 toys and 42 sweets. Most number of guests invited such that all toys and sweets are distributed equally?

6

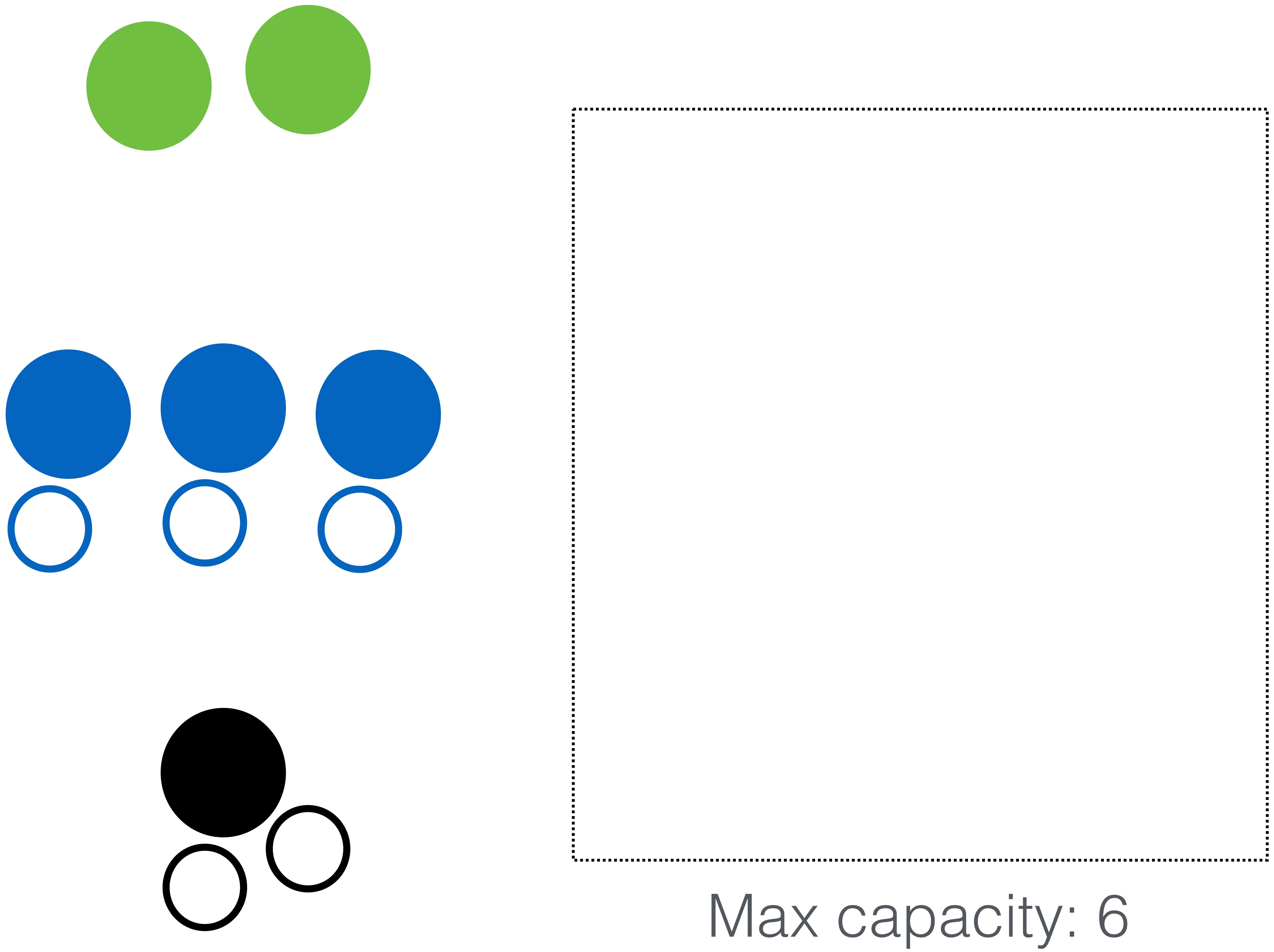
Now the guests have siblings:

Guest 1: 0 siblings
Guest 2: 2 siblings
Guest 3: 1 sibling
Guest 4: 0 siblings
Guest 5: 1 sibling
Guest 6: 1 sibling

Maximum number of guests where **6 people in total** (both guests and siblings) get toys and sweets?

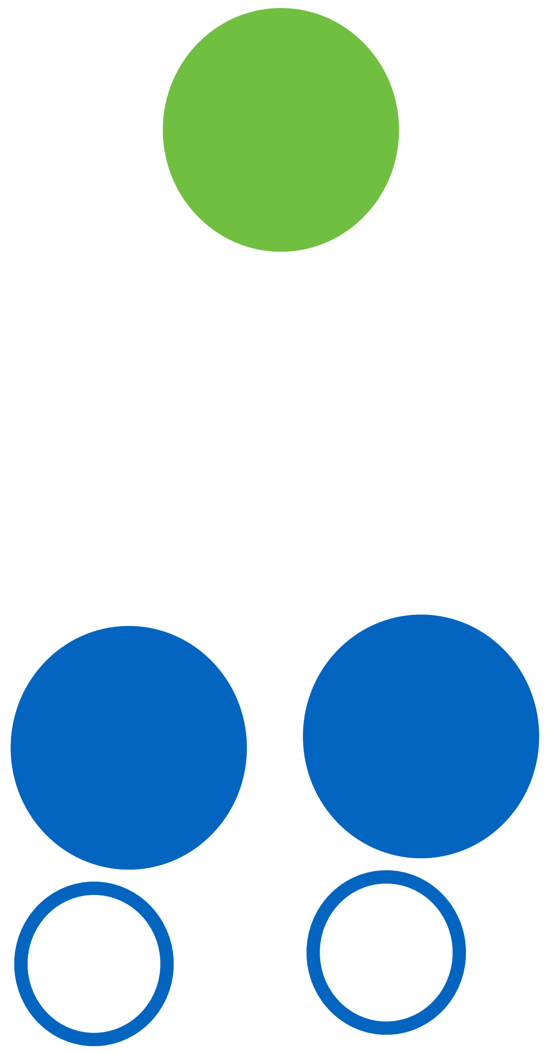
4

Guest 1: 0 siblings
Guest 2: 2 siblings
Guest 3: 1 sibling
Guest 4: 0 siblings
Guest 5: 1 sibling
Guest 6: 1 sibling

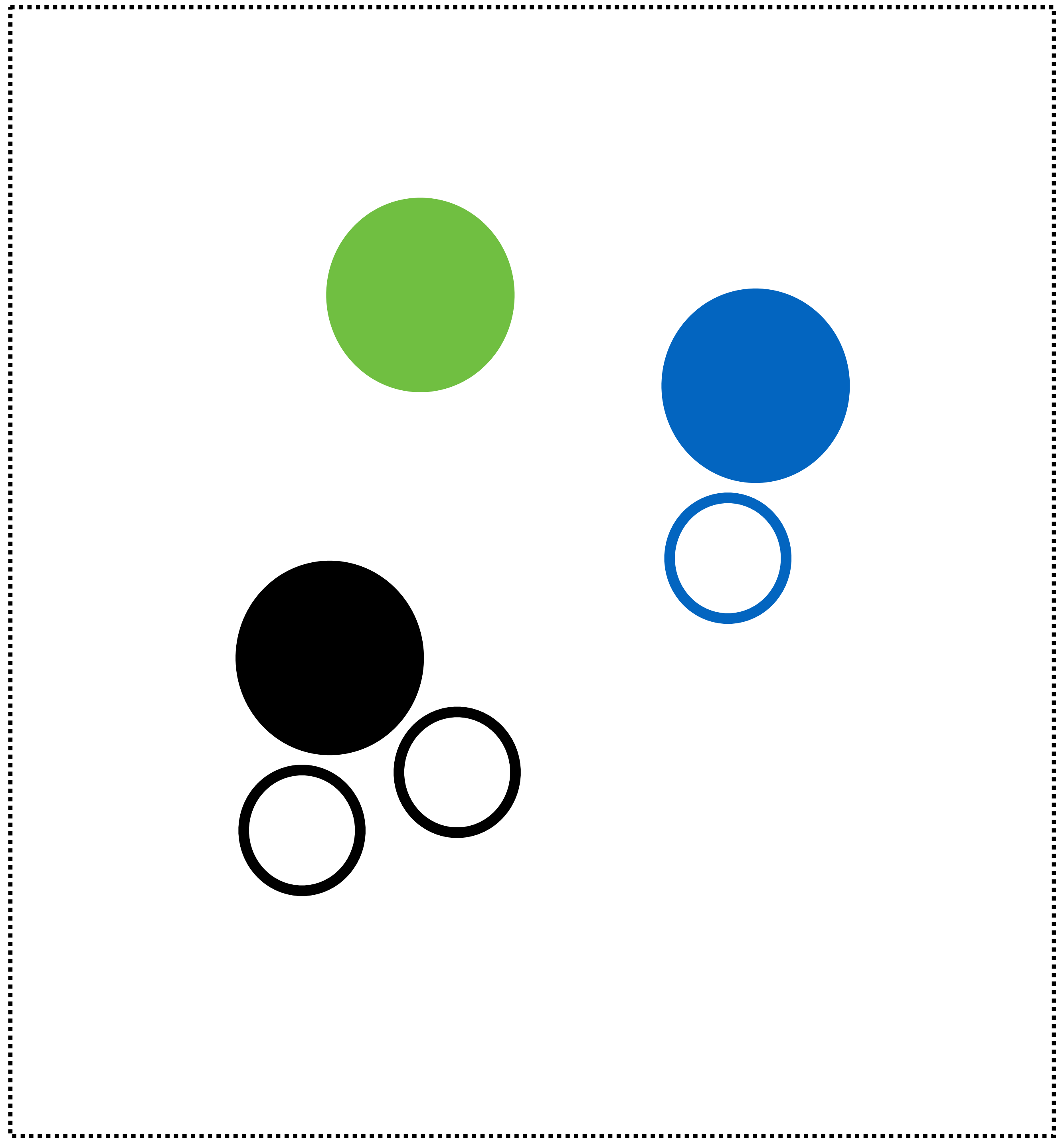


Max capacity: 6

Guest 1: 0 siblings
Guest 2: 2 siblings
Guest 3: 1 sibling
Guest 4: 0 siblings
Guest 5: 1 sibling
Guest 6: 1 sibling

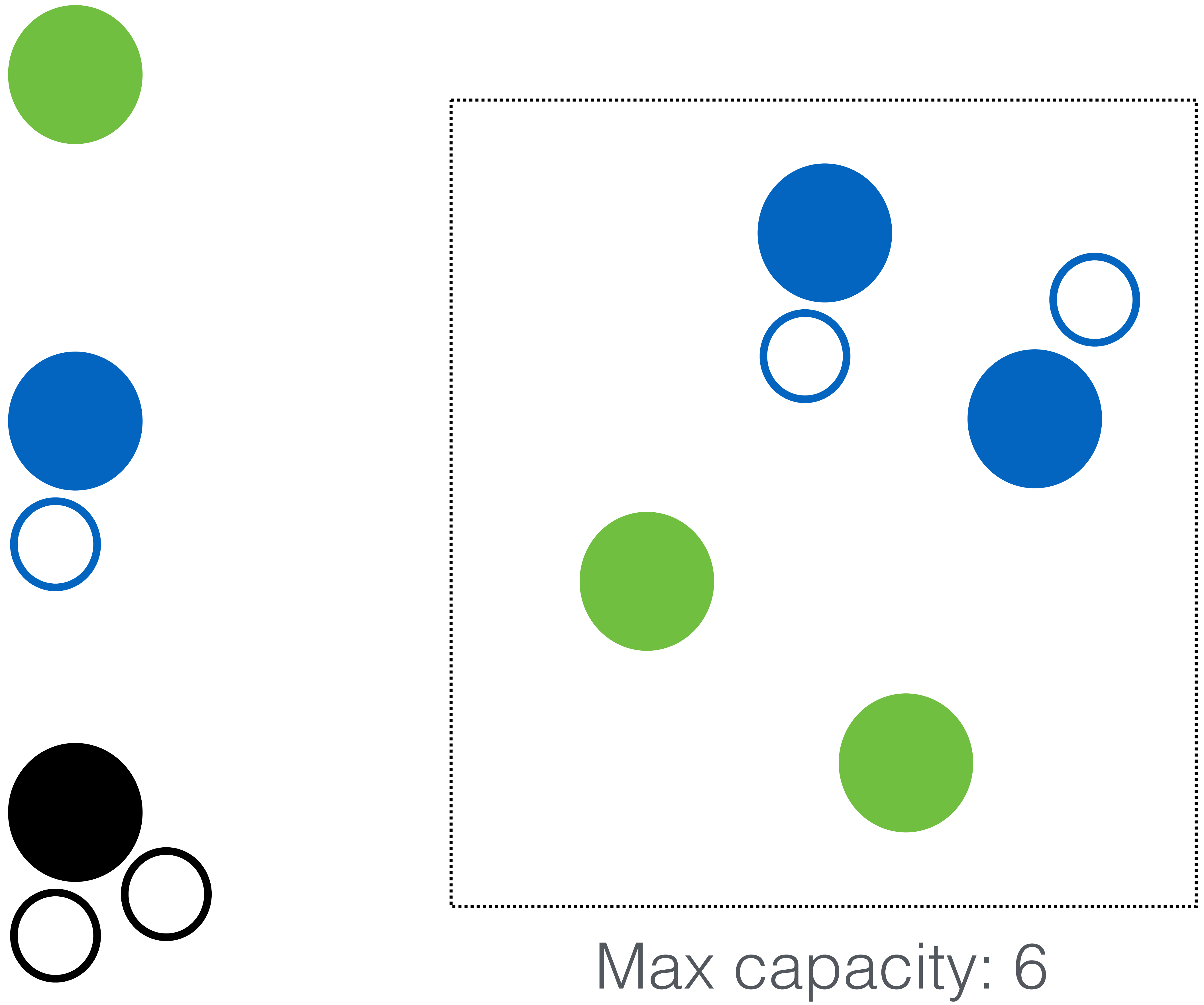


Only 3 guests!



Max capacity: 6

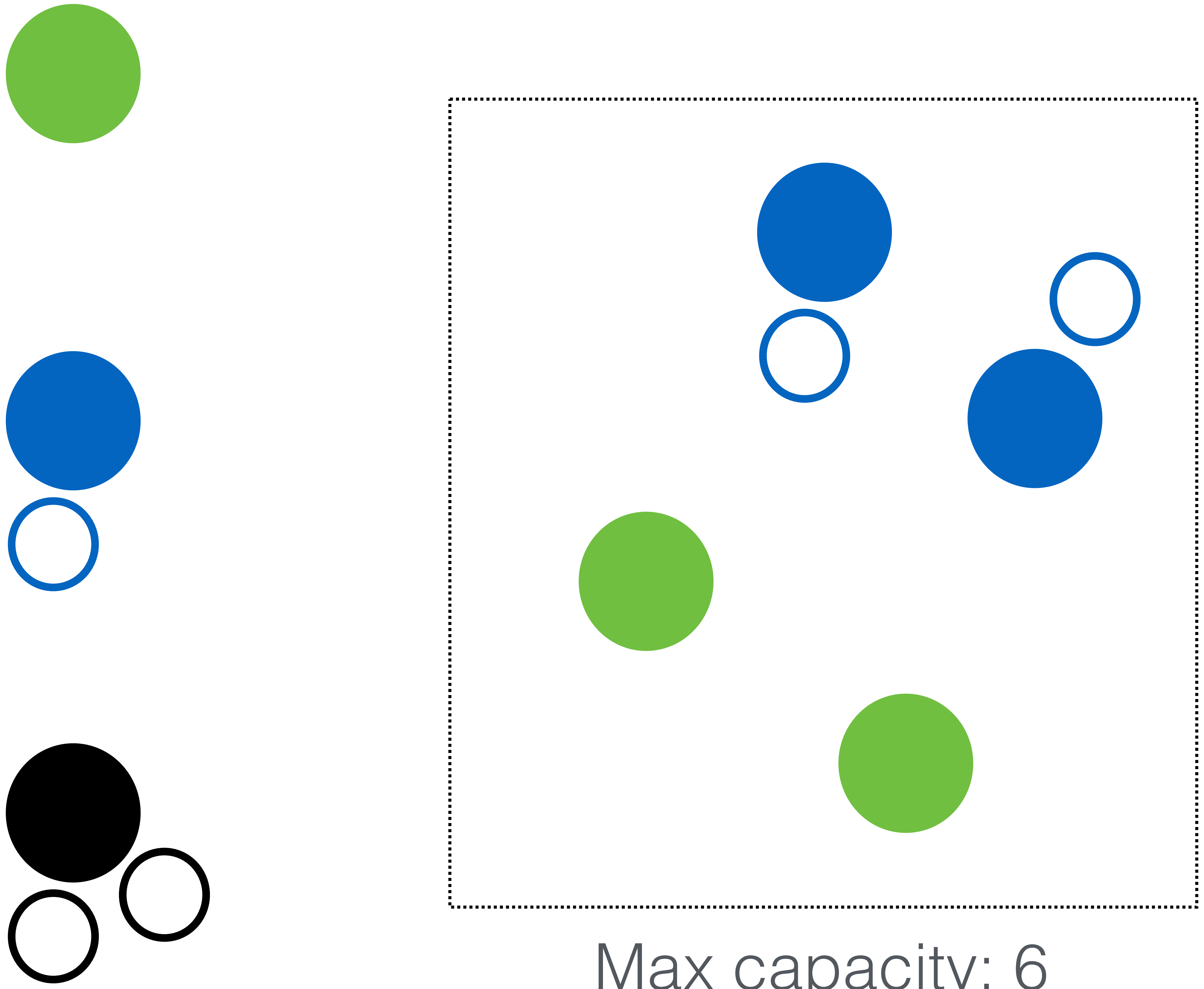
Guest 1: 0 siblings
Guest 2: 2 siblings
Guest 3: 1 sibling
Guest 4: 0 siblings
Guest 5: 1 sibling
Guest 6: 1 sibling



Max capacity: 6

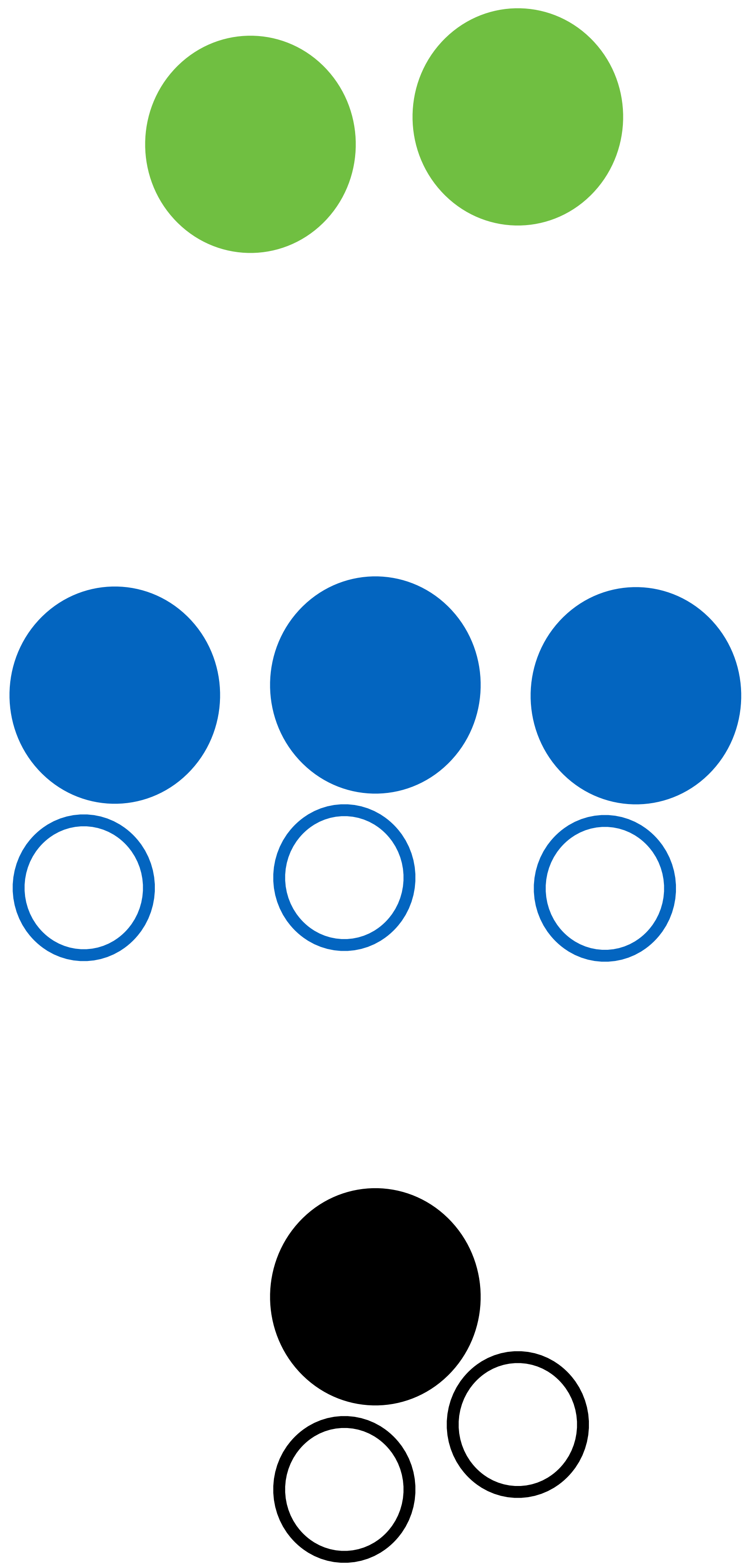
4 guests!

Guest 1: 0 siblings
Guest 2: 2 siblings
Guest 3: 1 sibling
Guest 4: 0 siblings
Guest 5: 1 sibling
Guest 6: 1 sibling



How do we know this is the best?

Guest 1: 0 siblings
Guest 2: 2 siblings
Guest 3: 1 sibling
Guest 4: 0 siblings
Guest 5: 1 sibling
Guest 6: 1 sibling



N_0

Number of guests with 0 siblings

N_1

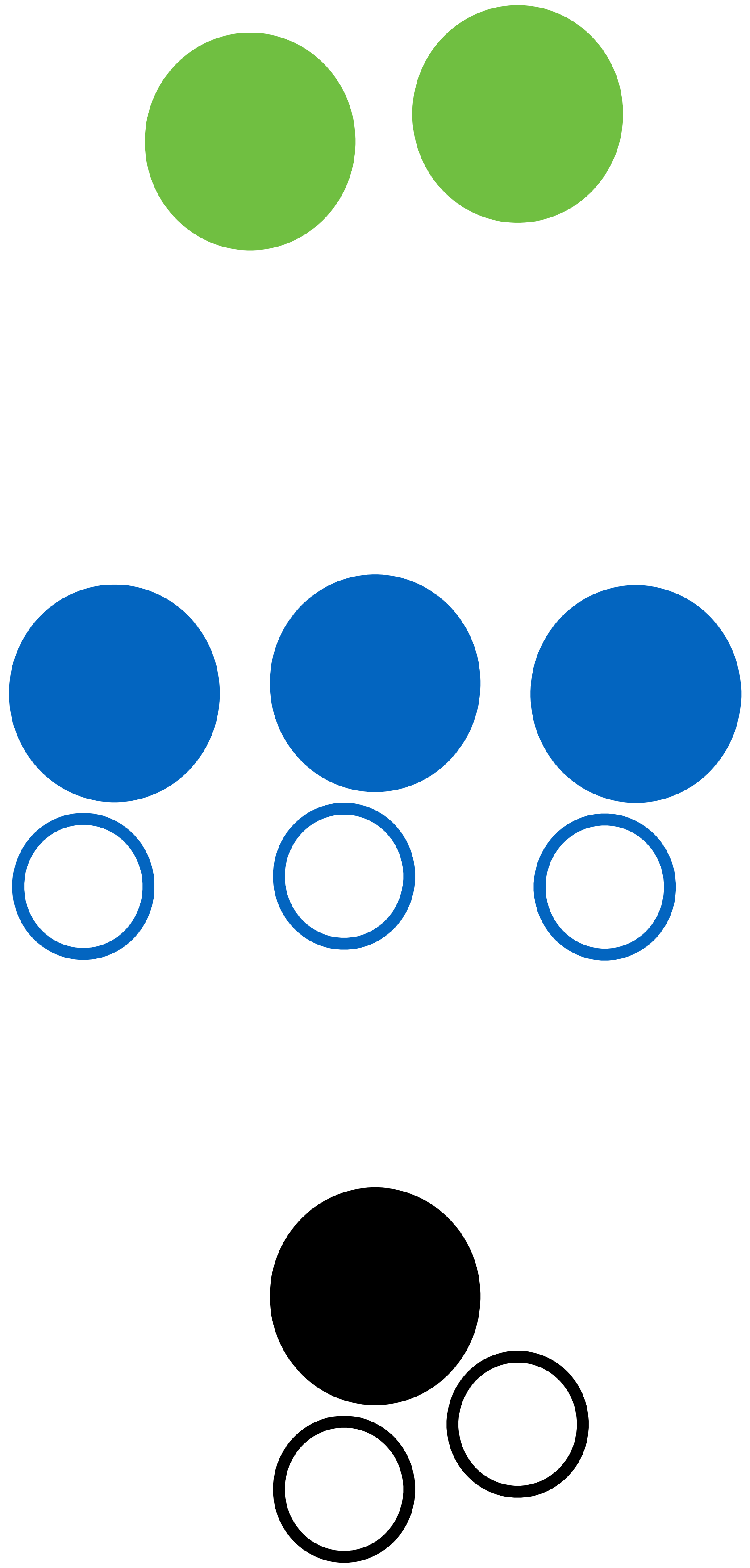
Number of guests with 1 sibling

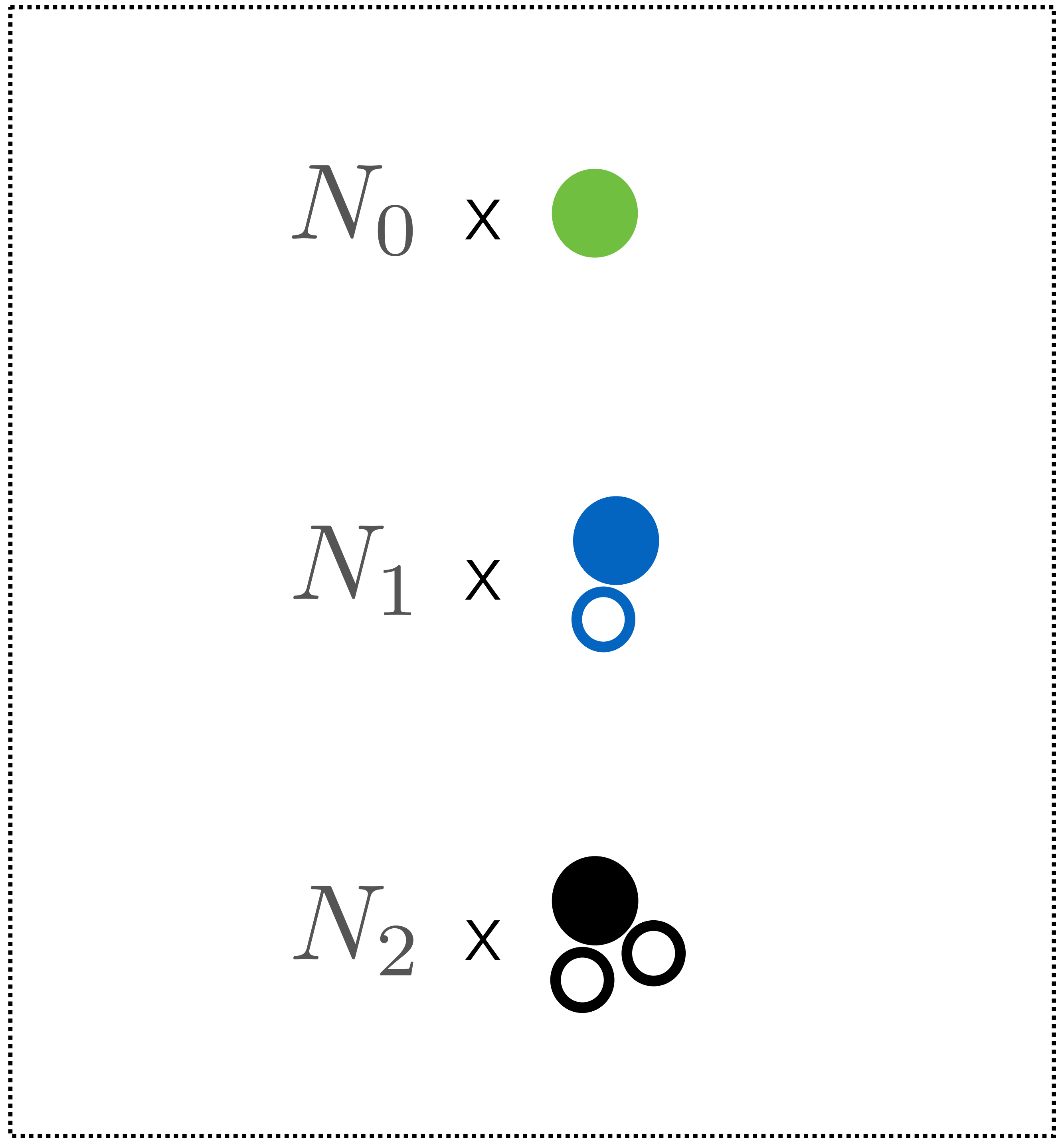
N_2

Number of guests with 2 siblings

Total number of guests $N_0 + N_1 + N_2$

Guest 1: 0 siblings
Guest 2: 2 siblings
Guest 3: 1 sibling
Guest 4: 0 siblings
Guest 5: 1 sibling
Guest 6: 1 sibling





N_0 x

N_1 x

N_2 x

Max capacity: 6

Total number of guests $N_0 + N_1 + N_2$

Guest 1: 0 siblings
Guest 2: 2 siblings
Guest 3: 1 sibling
Guest 4: 0 siblings
Guest 5: 1 sibling
Guest 6: 1 sibling

Total number of guests

variables $N_0 \quad N_1 \quad N_2$

maximise $N_0 + N_1 + N_2$

where

$$N_0 + 2N_1 + 3N_2 = 6$$

$$N_0 \leq 2$$

$$N_1 \leq 3$$

$$N_2 \leq 1$$

Total number of people getting toys and sweets

Guest 1: 0 siblings
Guest 2: 2 siblings
Guest 3: 1 sibling
Guest 4: 0 siblings
Guest 5: 1 sibling
Guest 6: 1 sibling

variables $N_0 \quad N_1 \quad N_2$

maximise $N_0 + N_1 + N_2$

where $N_0 + 2N_1 + 3N_2 = 6$

$N_0 \leq 2$

$N_1 \leq 3$

$N_2 \leq 1$

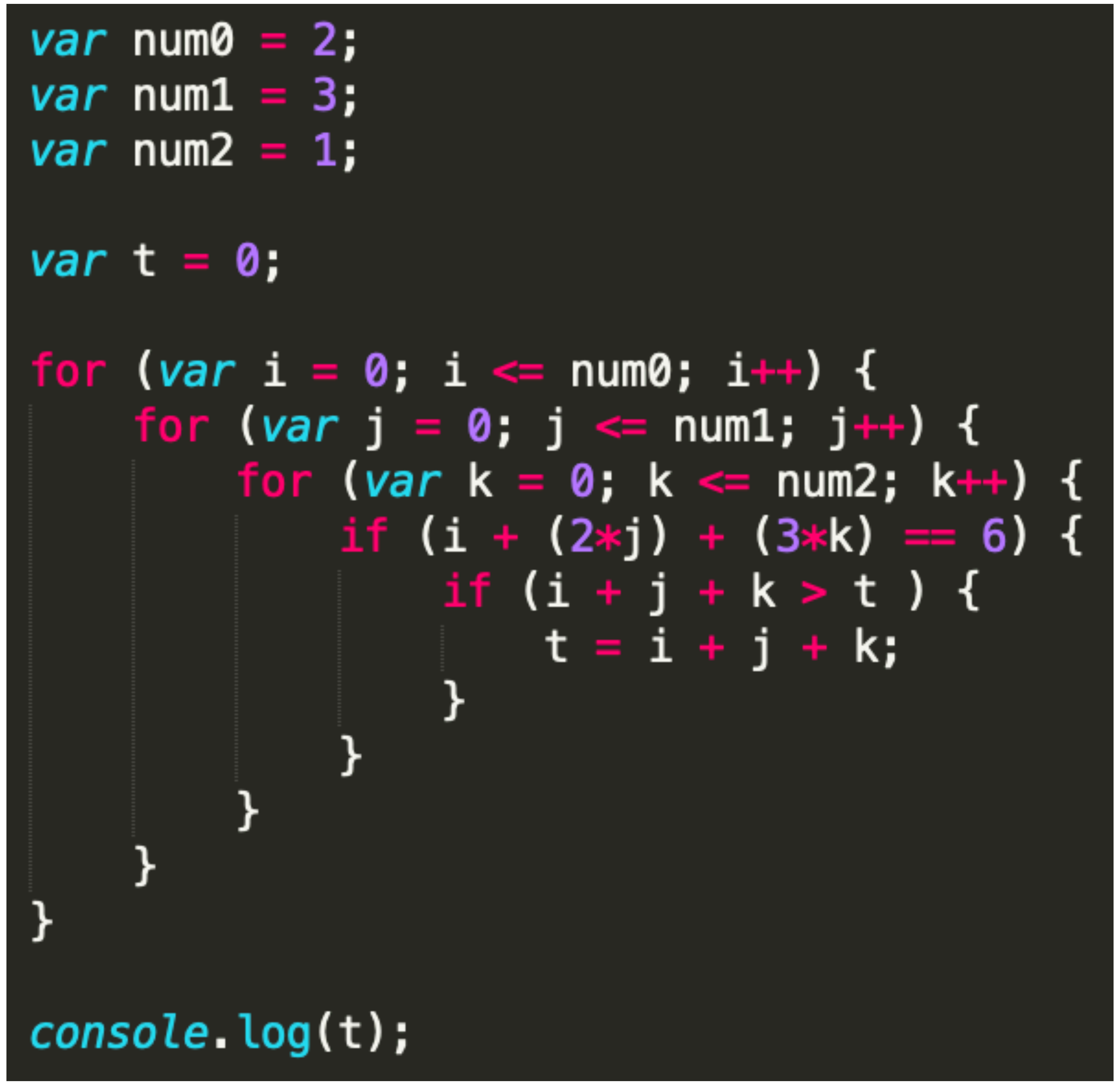
Try all possible assignments of variables to find solution

variables $N_0 \quad N_1 \quad N_2$

maximise $N_0 + N_1 + N_2$

where $N_0 + 2N_1 + 3N_2 = 6$

$N_0 \leq 2 \quad N_1 \leq 3 \quad N_2 \leq 1$



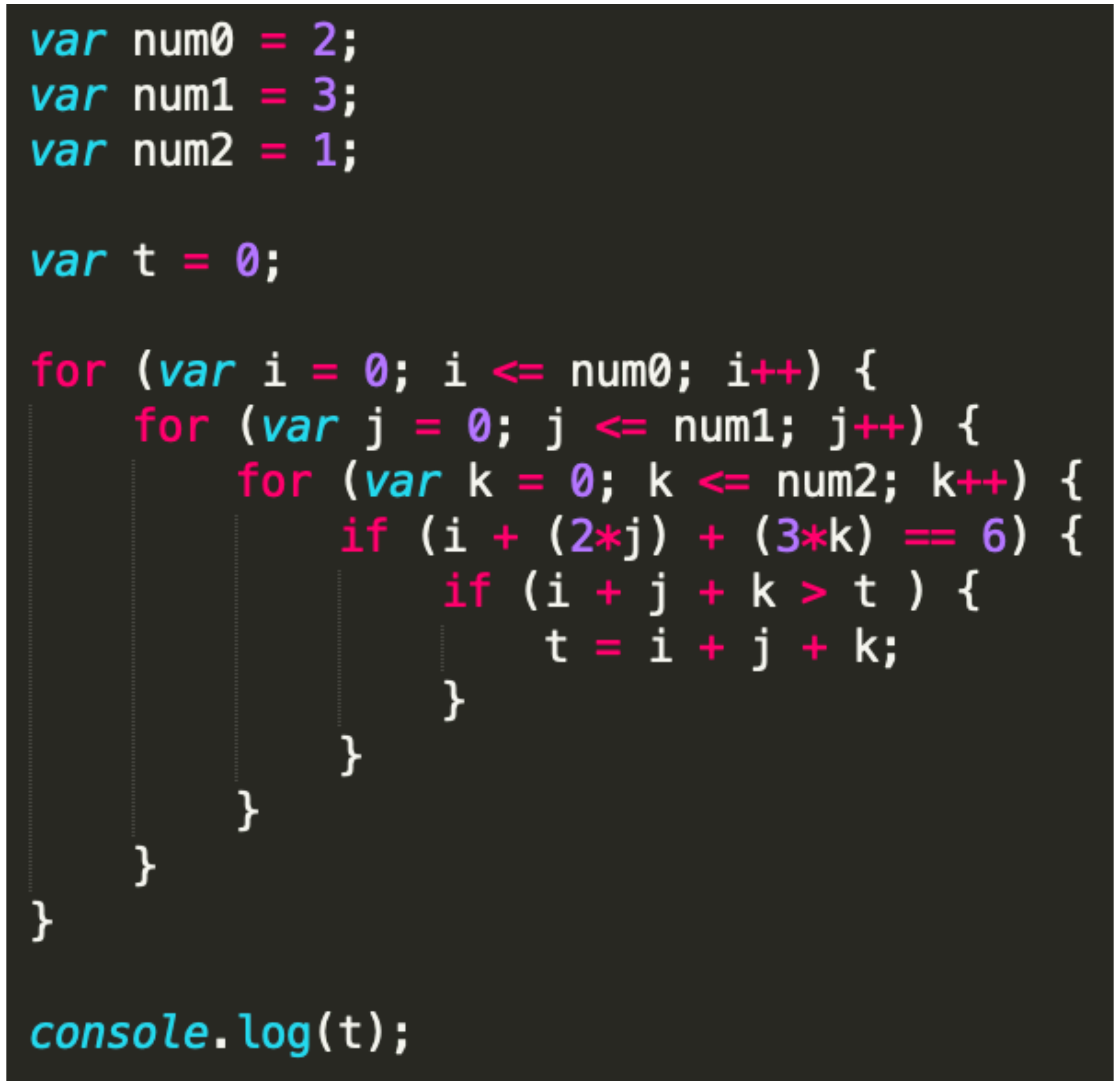
```javascript
var num0 = 2;
var num1 = 3;
var num2 = 1;

var t = 0;

for (var i = 0; i <= num0; i++) {
    for (var j = 0; j <= num1; j++) {
        for (var k = 0; k <= num2; k++) {
            if (i + (2*j) + (3*k) == 6) {
                if (i + j + k > t ) {
                    t = i + j + k;
                }
            }
        }
    }
}

console.log(t);
```

variables $N_0 \quad N_1 \quad N_2$

maximise $N_0 + N_1 + N_2$

where $N_0 + 2N_1 + 3N_2 = 6$

$N_0 \leq 2 \;\; N_1 \leq 3 \;\; N_2 \leq 1$



```
var num0 = 2;
var num1 = 3;
var num2 = 1;

var t = 0;

for (var i = 0; i <= num0; i++) {
    for (var j = 0; j <= num1; j++) {
        for (var k = 0; k <= num2; k++) {
            if (i + (2*j) + (3*k) == 6) {
                if (i + j + k > t ) {
                    t = i + j + k;
                }
            }
        }
    }
}

console.log(t);
```

i	j	k	t
0	0	0	0
0	3	0	3
1	1	1	3
2	2	0	4

Improvements upon this method?

Guest 1: 0 siblings
Guest 2: 2 siblings
Guest 3: 1 sibling
Guest 4: 0 siblings
Guest 5: 1 sibling
Guest 6: 1 sibling

variables $N_0 \quad N_1 \quad N_2$

maximise $N_0 + N_1 + N_2$

where

$$N_0 + 2N_1 + 3N_2 = 6$$

$$N_0 \leq 2$$

$$N_1 \leq 3$$

$$N_2 \leq 1$$

Try all possible assignments of variables to find solution

Trying all possibilities

This is one of the first strategies for solving a problem

Recall: solution method 1 of deciding if a square root is an integer

It may not always be the most efficient method

Sometimes it is the only method

Trying all possibilities

This is one of the first strategies for solving a problem

Recall: solution method 1 of deciding if a square root is an integer

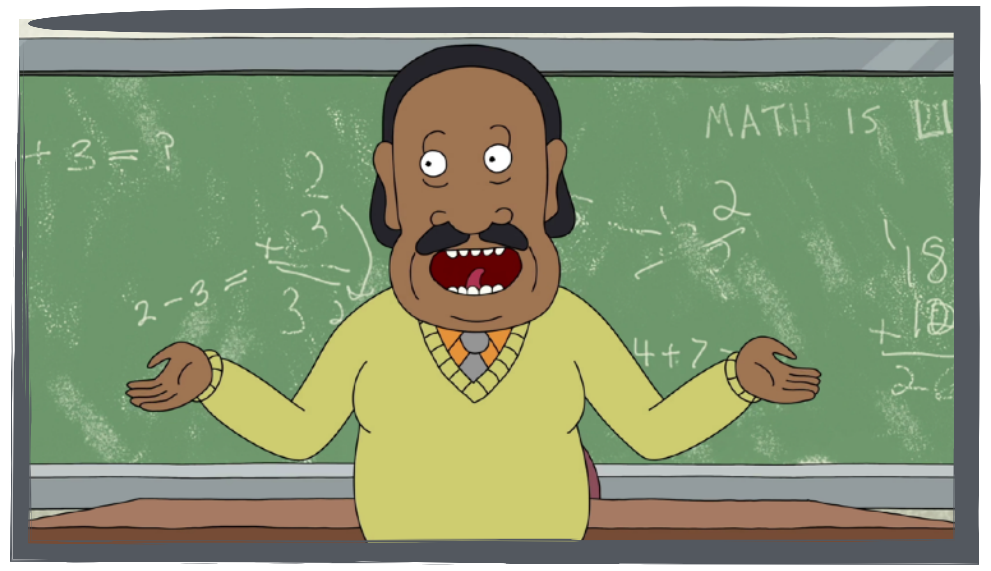
It may not always be the most efficient method

Sometimes it is the only method

Searching Problems

The module

THEORY



EXPERIMENT

Flowcharts

Searching

Algorithms

Recursion

Sorting

Functions

Node.js

Programming
in JavaScript

Queues

Stacks

Abstract Data Structures

**Vectors &
Dynamic Arrays**

Arrays & Objects

Command line

Computer model

Analysis

Time complexity

Admin

- Third quiz available today from 4pm
  - **Deadline for first quiz: TODAY at 4pm**
  - Deadline for second quiz: 8th February 4pm
  - Deadline for third quiz: 15th February 4pm

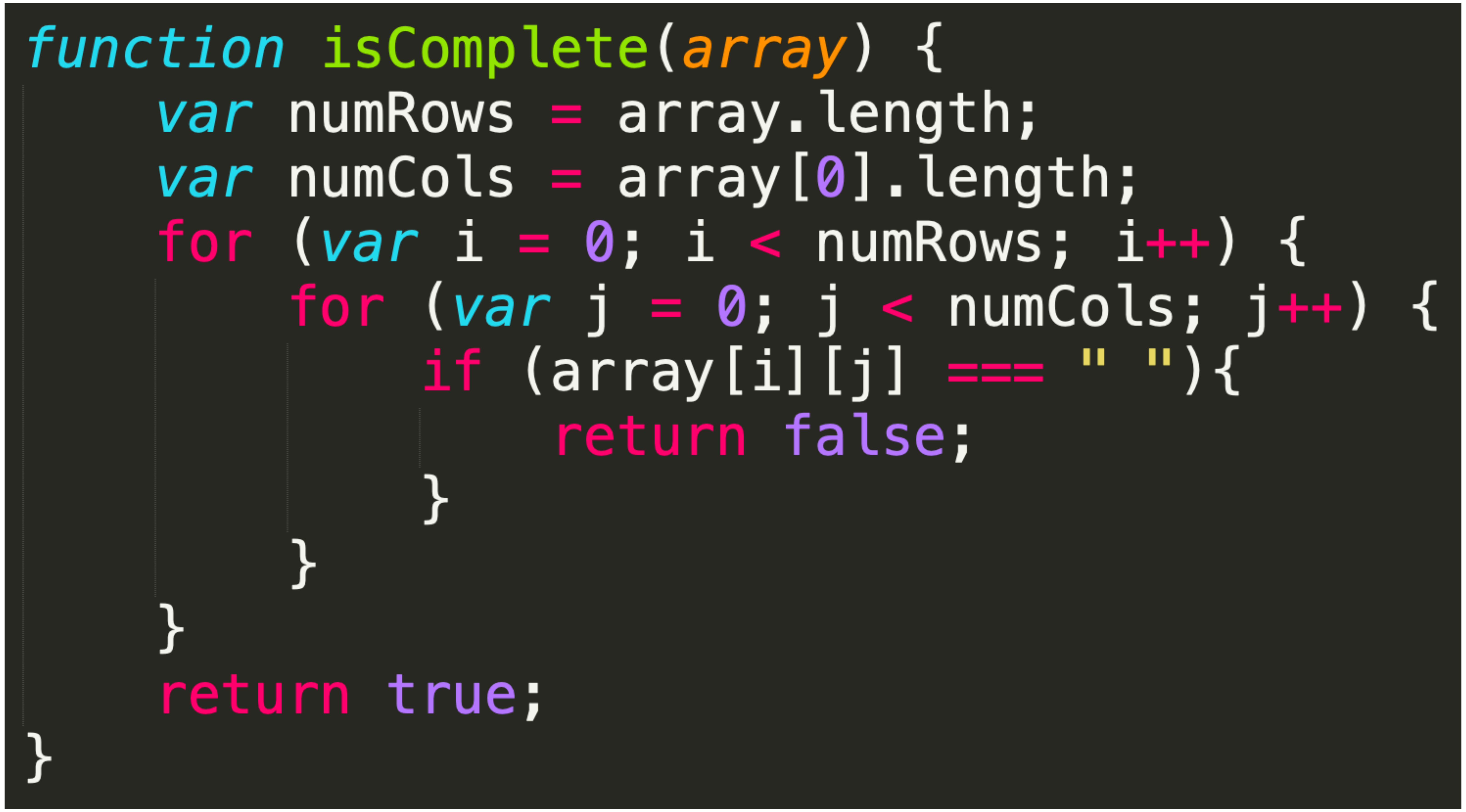
- Worksheet 3 available today from 11am - extremely useful for assignment…

- Sudoku assignment (**assessed**) released **Monday 8th February at 11am**
  - Worksheet for assignment will be in Week 5
  - Same programming environment as worksheets - go through previous worksheets if you haven't done so already
  - Submit js file and written work as separate submissions in Assessments section on learn.gold
  - Deadline is **Monday 1st March at 4pm**

Make your code individual - there will be plagiarism checks

Lab 2

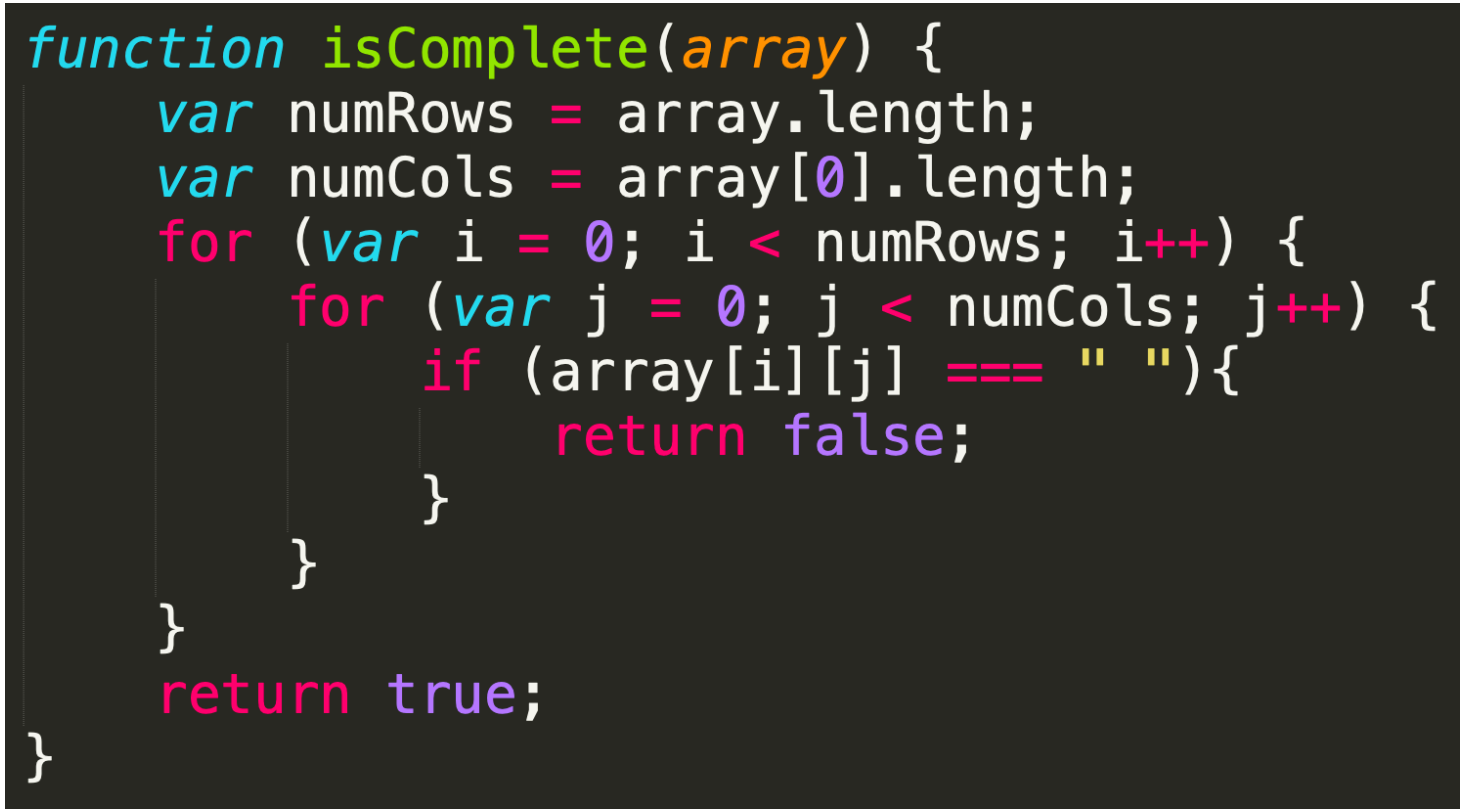
Deciding if an array stores the value “ “



```
function isComplete(array) {
    var numRows = array.length;
    var numCols = array[0].length;
    for (var i = 0; i < numRows; i++) {
        for (var j = 0; j < numCols; j++) {
            if (array[i][j] === " "){
                return false;
            }
        }
    }
    return true;
}
```

Lab 2

Deciding if an array stores the value “ “



```
function isComplete(array) {
    var numRows = array.length;
    var numCols = array[0].length;
    for (var i = 0; i < numRows; i++) {
        for (var j = 0; j < numCols; j++) {
            if (array[i][j] === " "){
                return false;
            }
        }
    }
    return true;
}
```

This is an example of an implementation of a **search algorithm**

Today

1. Searching vectors and dynamic arrays
2. Searching stacks and queues

Today

1. **Searching vectors and dynamic arrays**

2. Searching stacks and queues

Searching algorithms

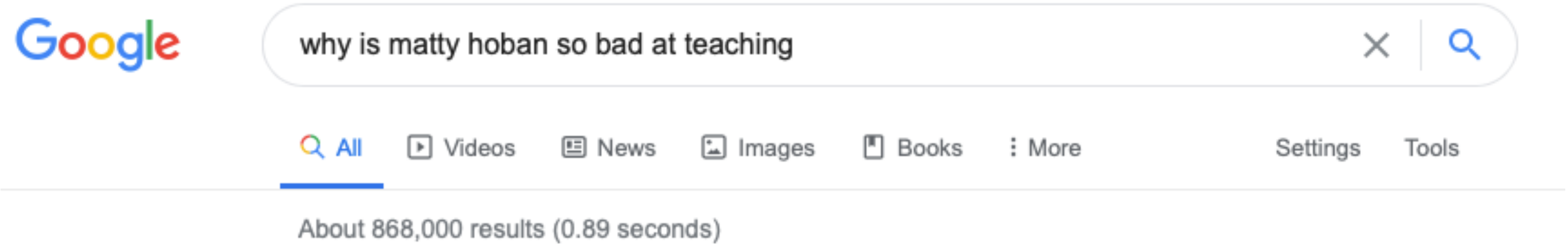
One of the most widely encountered problems

Donald Knuth devotes a whole book of nearly 800 pages to Searching and Sorting (*The Art of Computer Programming Vol. 3*)

Searching algorithms

One of the most widely encountered problems

Donald Knuth devotes a whole book of nearly 800 pages to Searching and Sorting (*The Art of Computer Programming Vol. 3*)



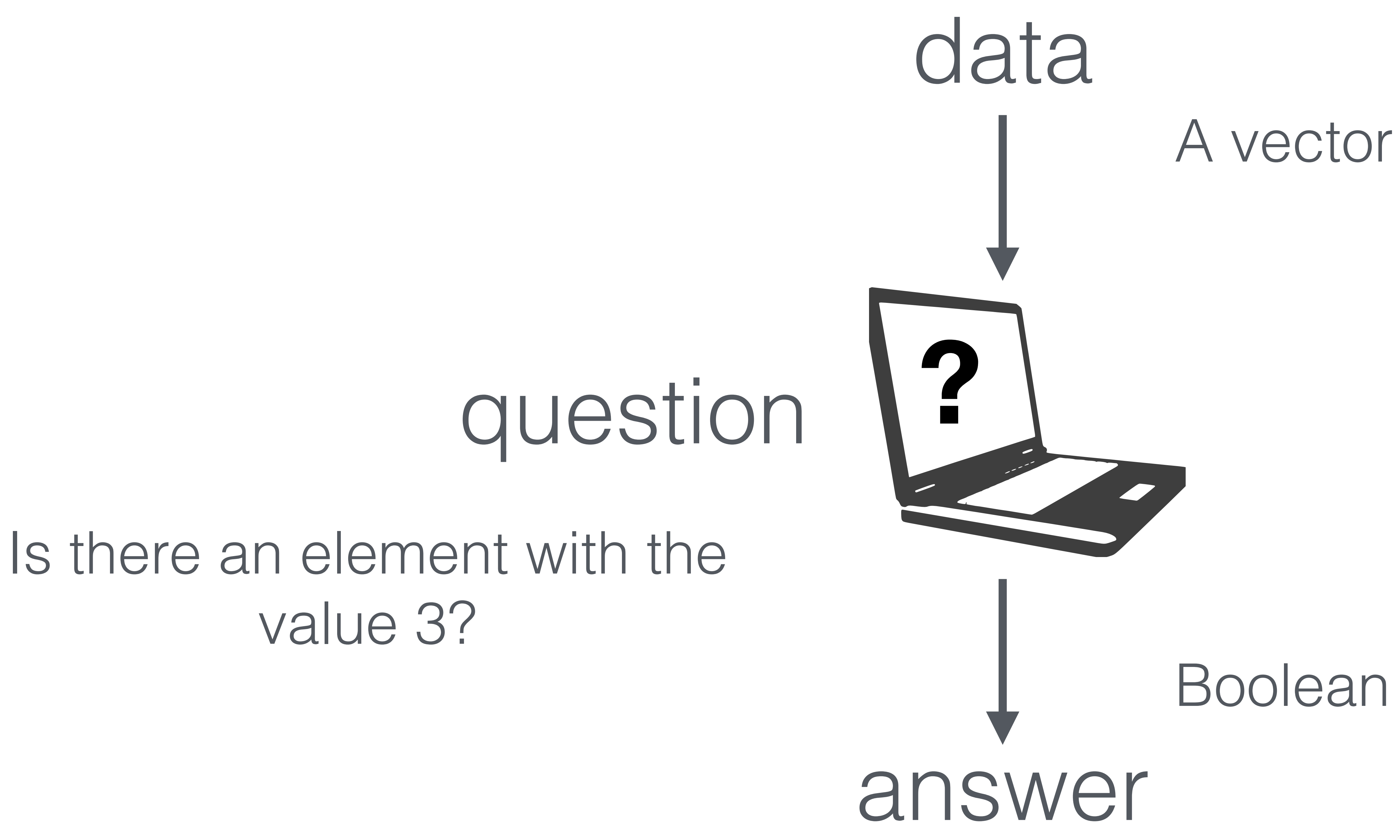
why is matty hoban so bad at teaching

About 868,000 results (0.89 seconds)

Internet Search Engine algorithms utilise the linked nature of the internet - PageRank ranks webpages

Imagine you are given a collection of data without knowing about its contents at all

An abstract problem



data

A vector

question

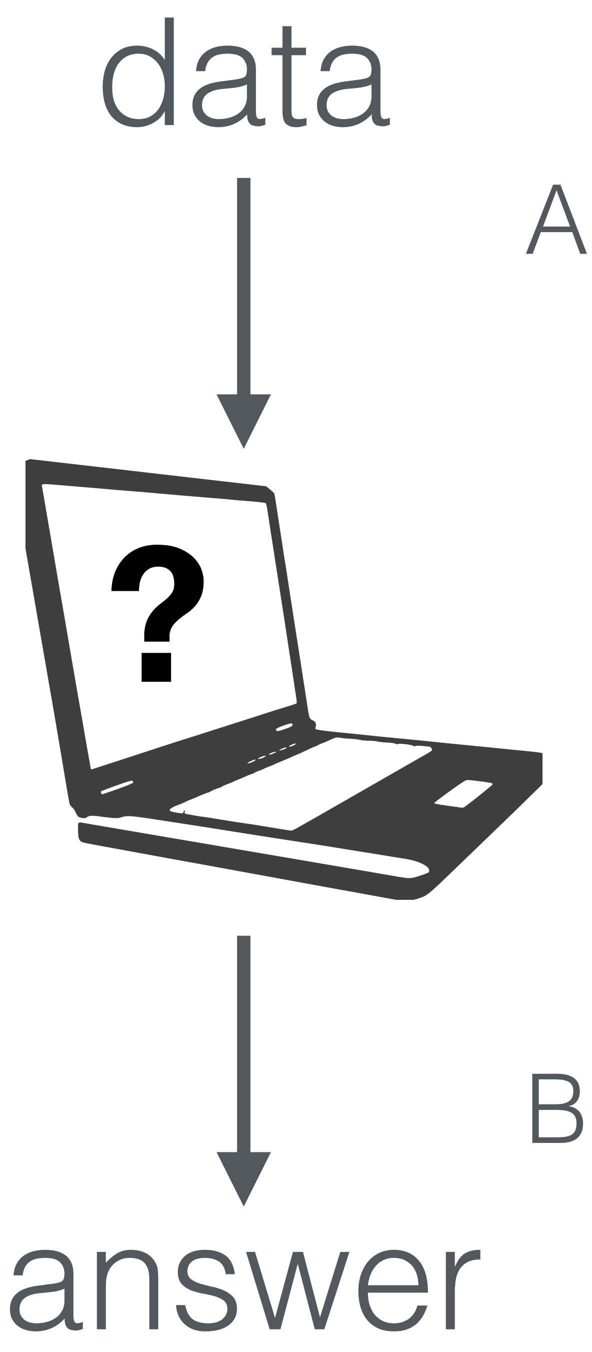
Is there an element with the value 3?

Boolean

answer

How do you solve this using the operations allowed by a vector?

An abstract problem



data

A vector

question

Is there an element with the value 3?

Boolean

answer

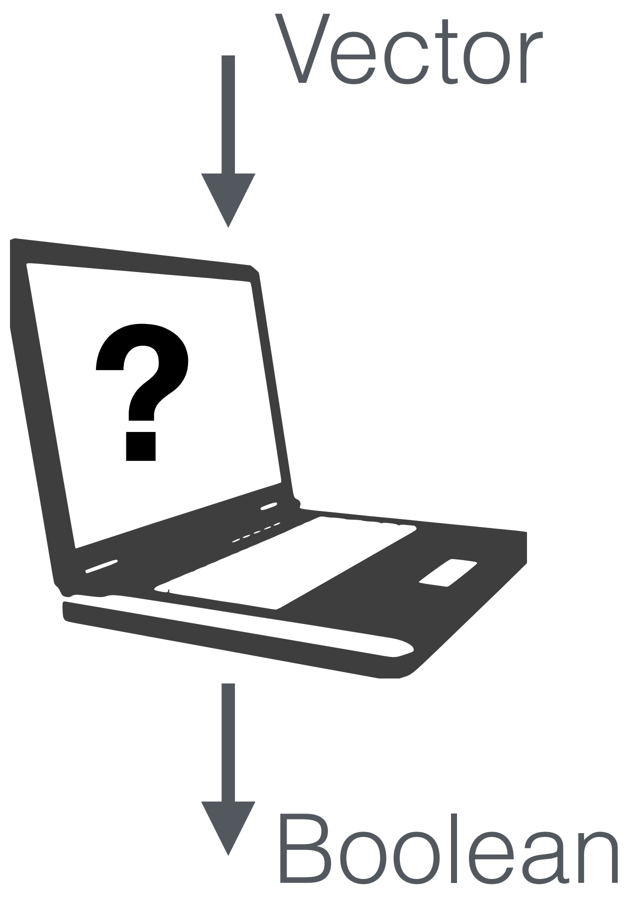
How do you solve this using the operations allowed by a vector?

select[k]

length

store![o,k]

Is there an element with the value x?

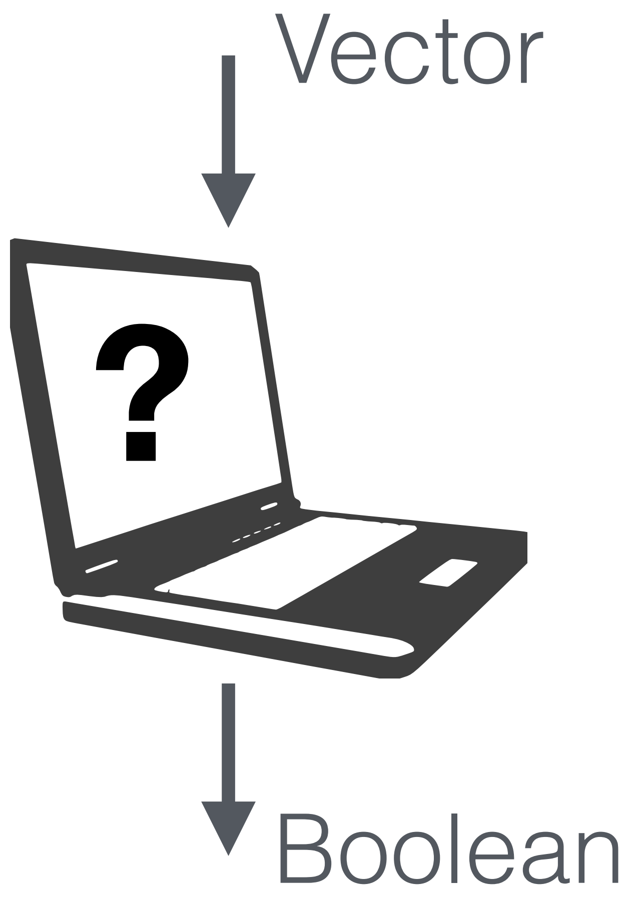


Vector

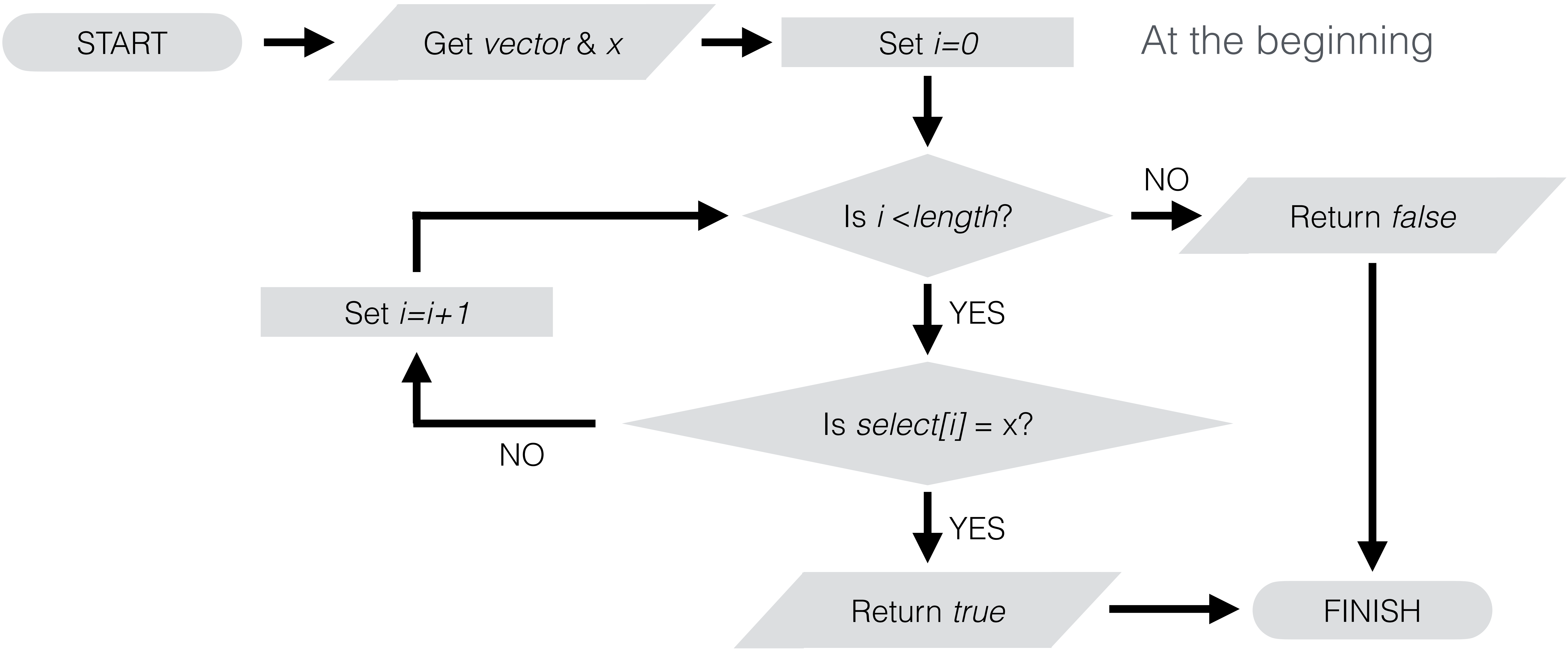
Boolean

We need to systematically "look" at the elements

Is there an element with the value x?



We need to systematically "look" at the elements



START

Get *vector* & *x*

Set *i=0*

At the beginning

Is *i* <*length*?

NO

Return *false*

YES

Set *i=i+1*

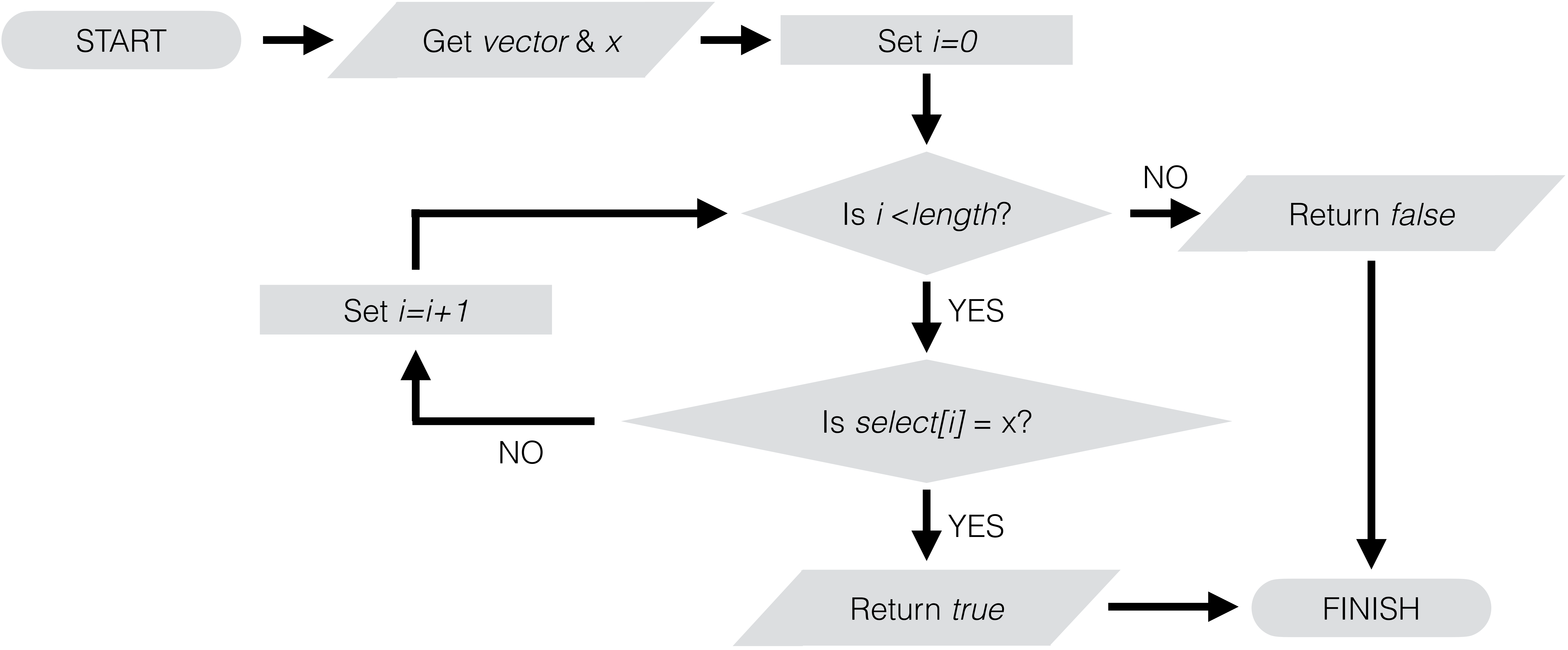
Is *select[i]* = x?

NO

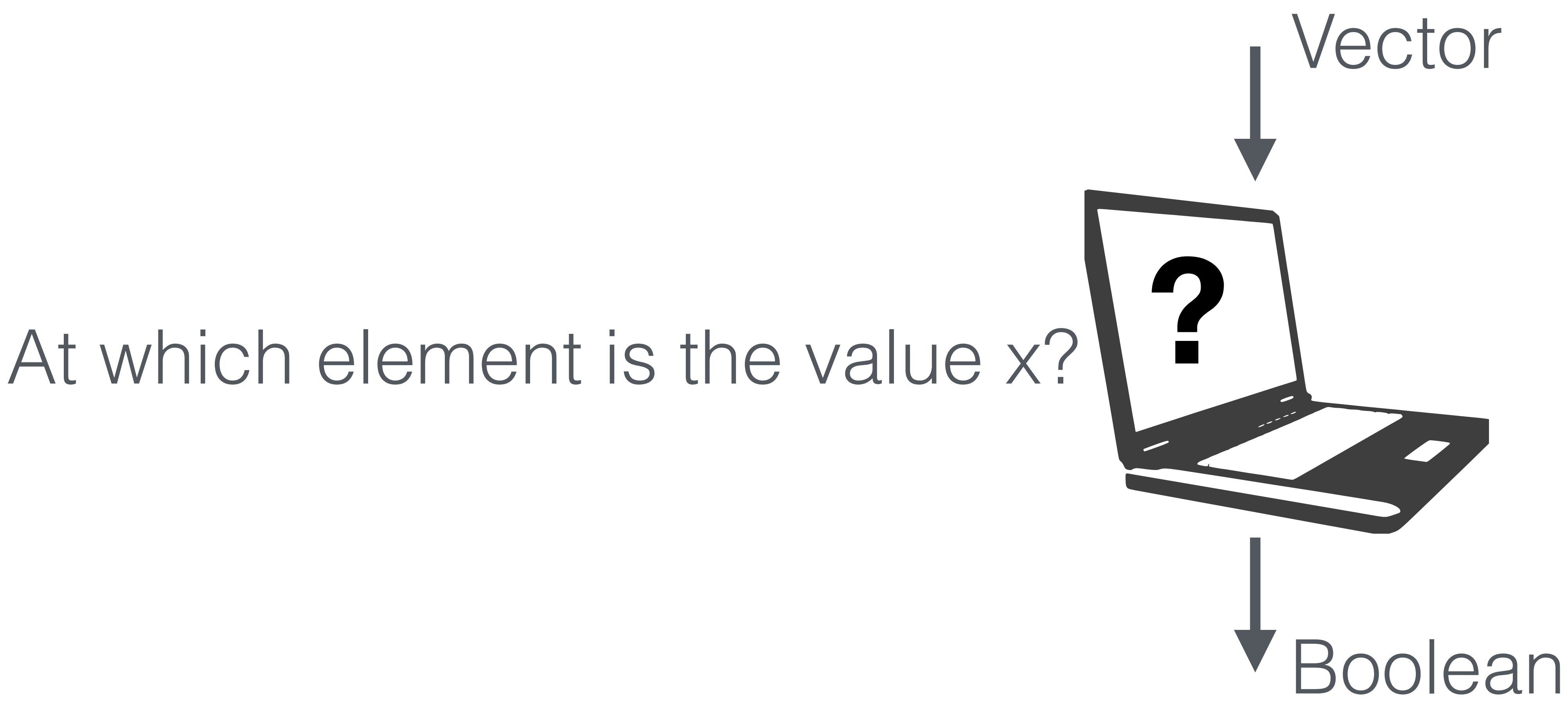
YES

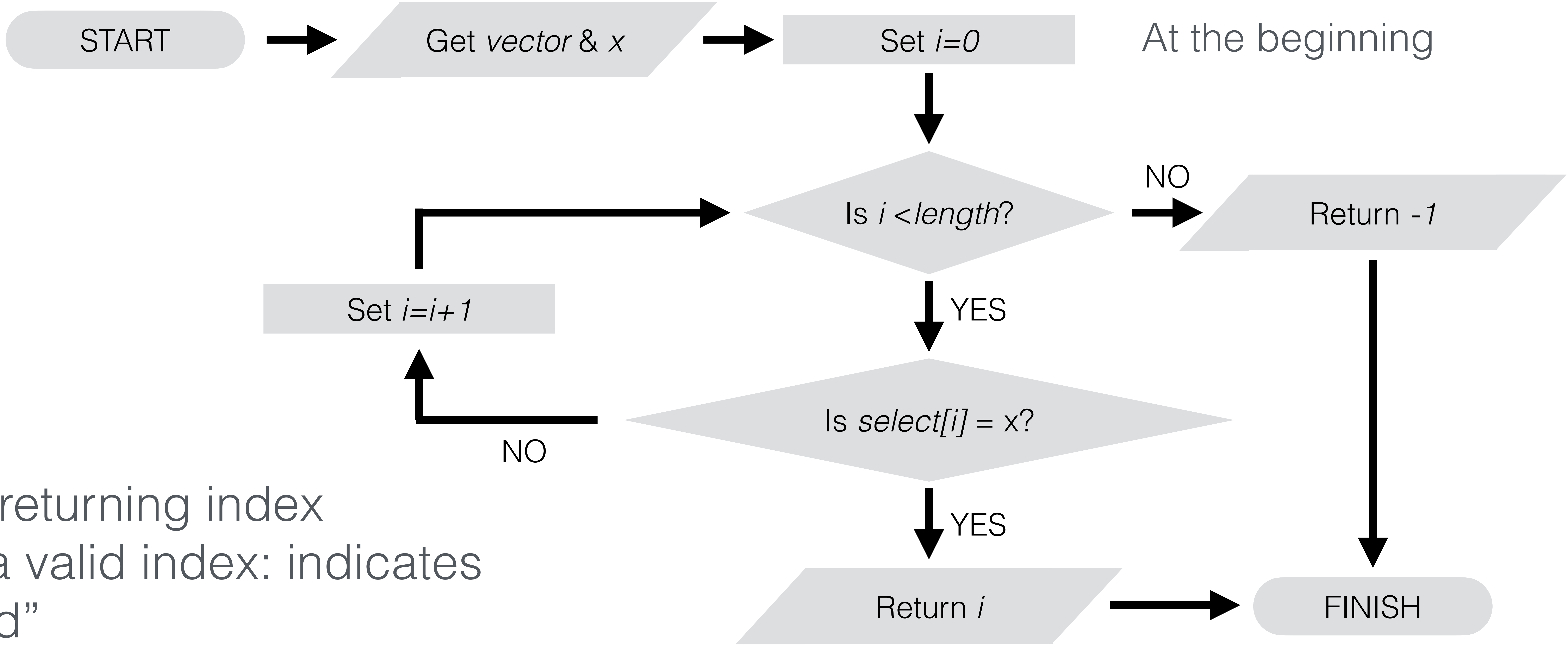
Return *true*

FINISH



- We traverse the vector using a loop
- If we find the value, return and algorithm is finished
- If loop completes then value was not found, return false

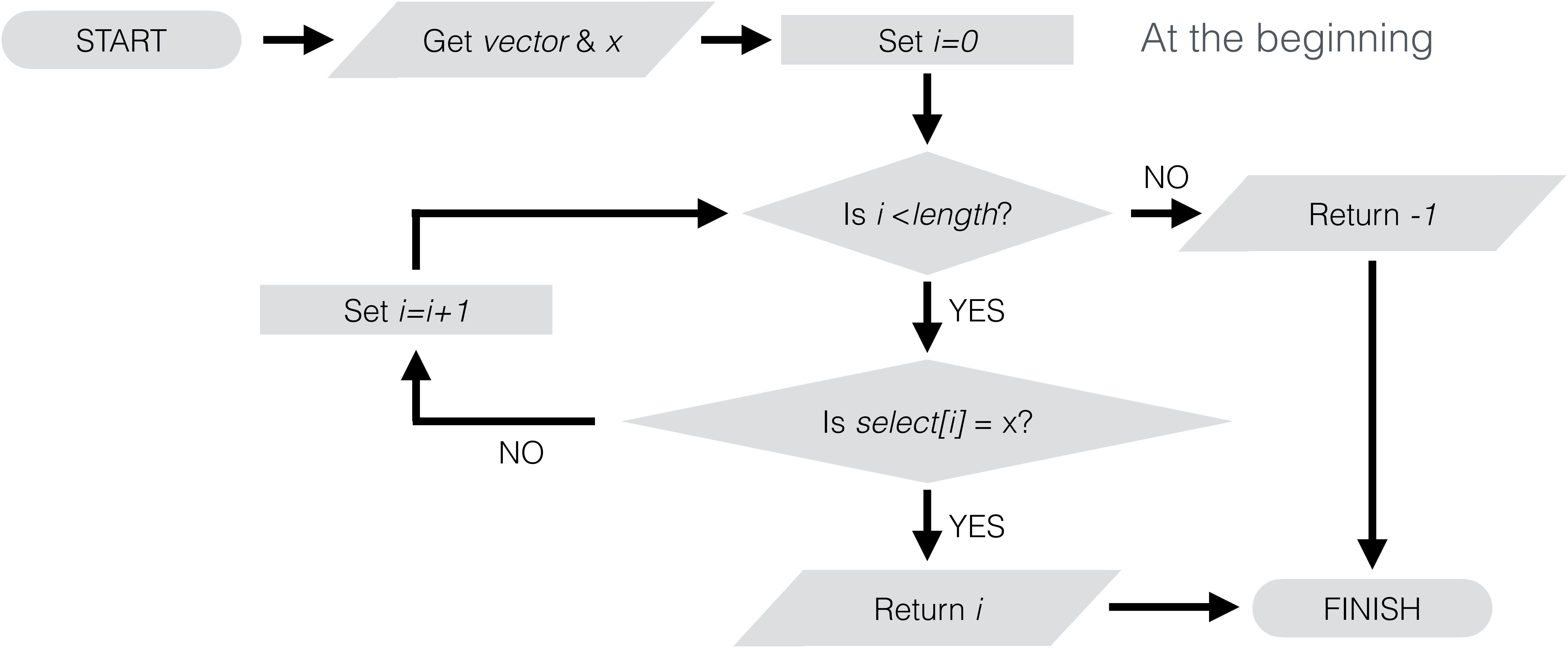




- Variation returning index
- -1 is not a valid index: indicates "not found"

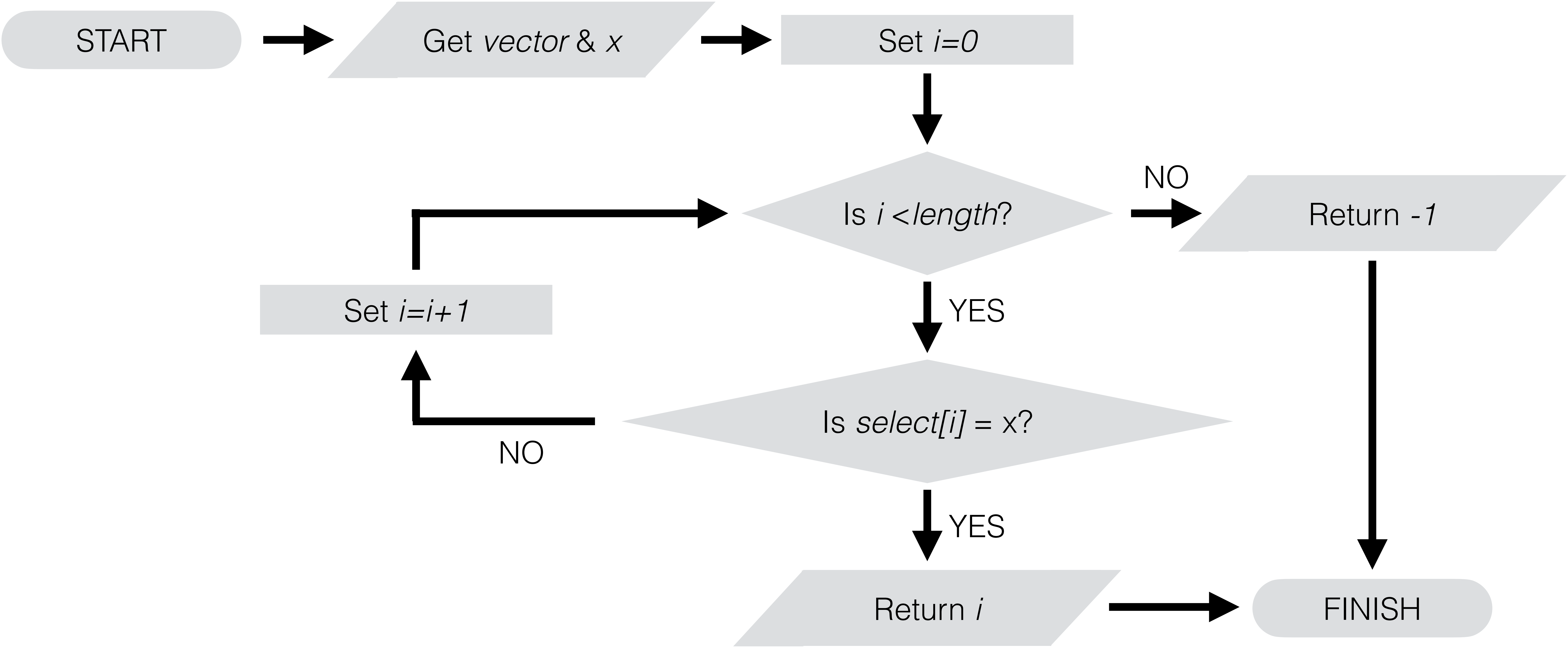
Called the ***Linear Search Algorithm***

(Sequential Search)



Called the ***Linear Search Algorithm***

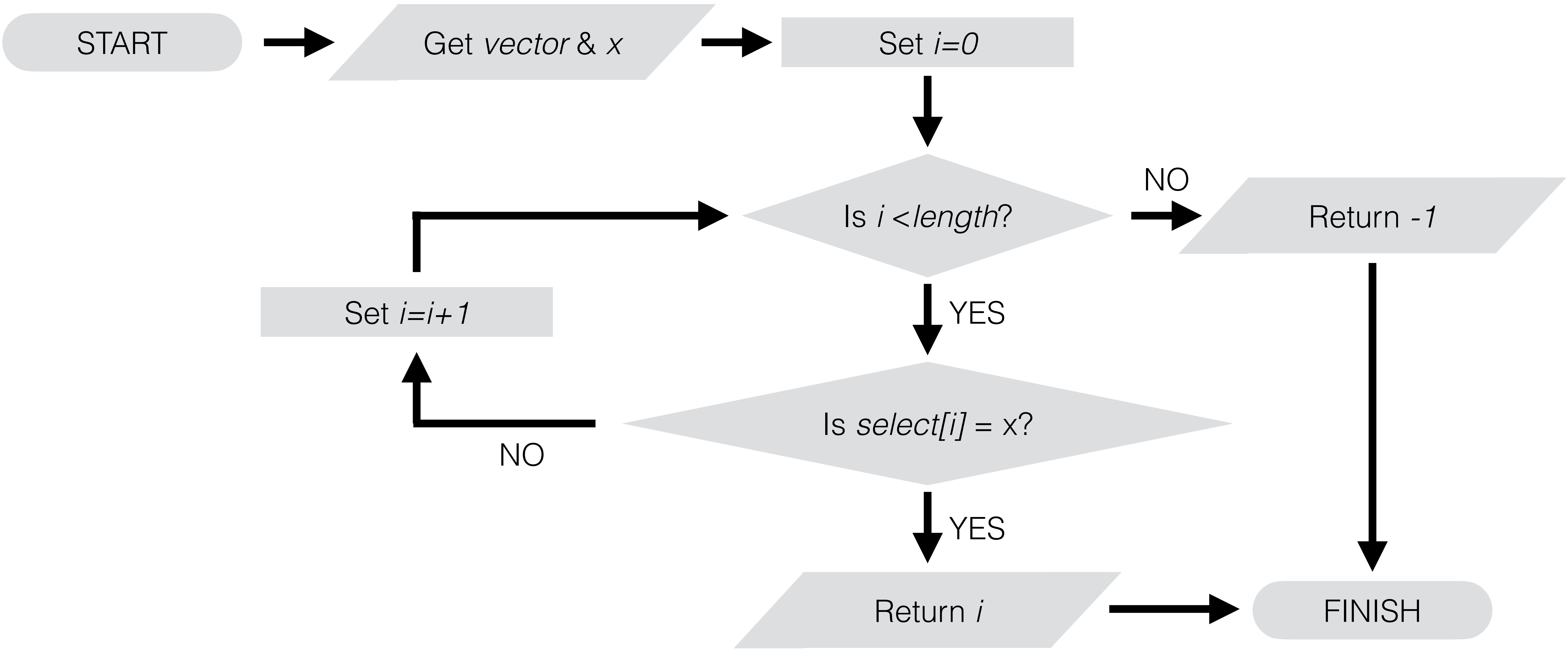
Why start at the beginning?



Called the ***Linear Search Algorithm***

Why start at the beginning?

As good as any place if you know nothing

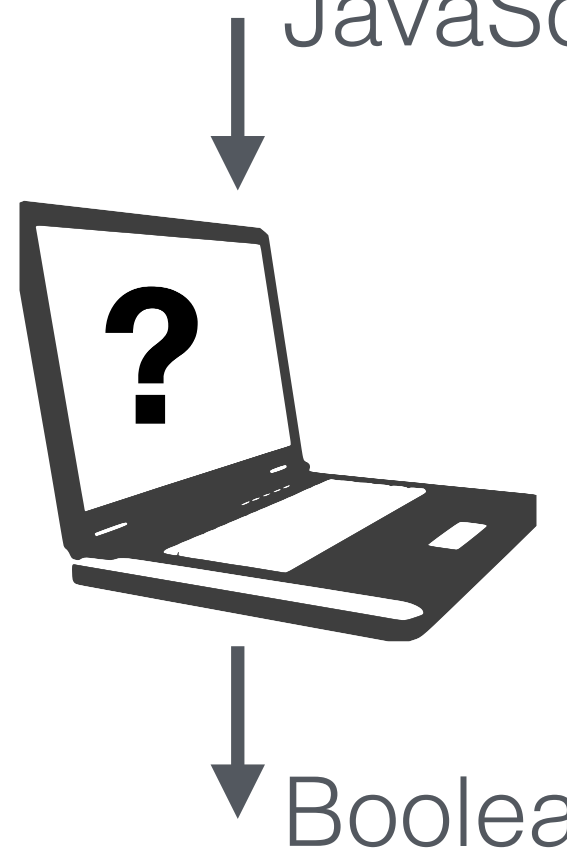


We just need to be able to read any of the elements

Therefore, works for Dynamic Arrays:

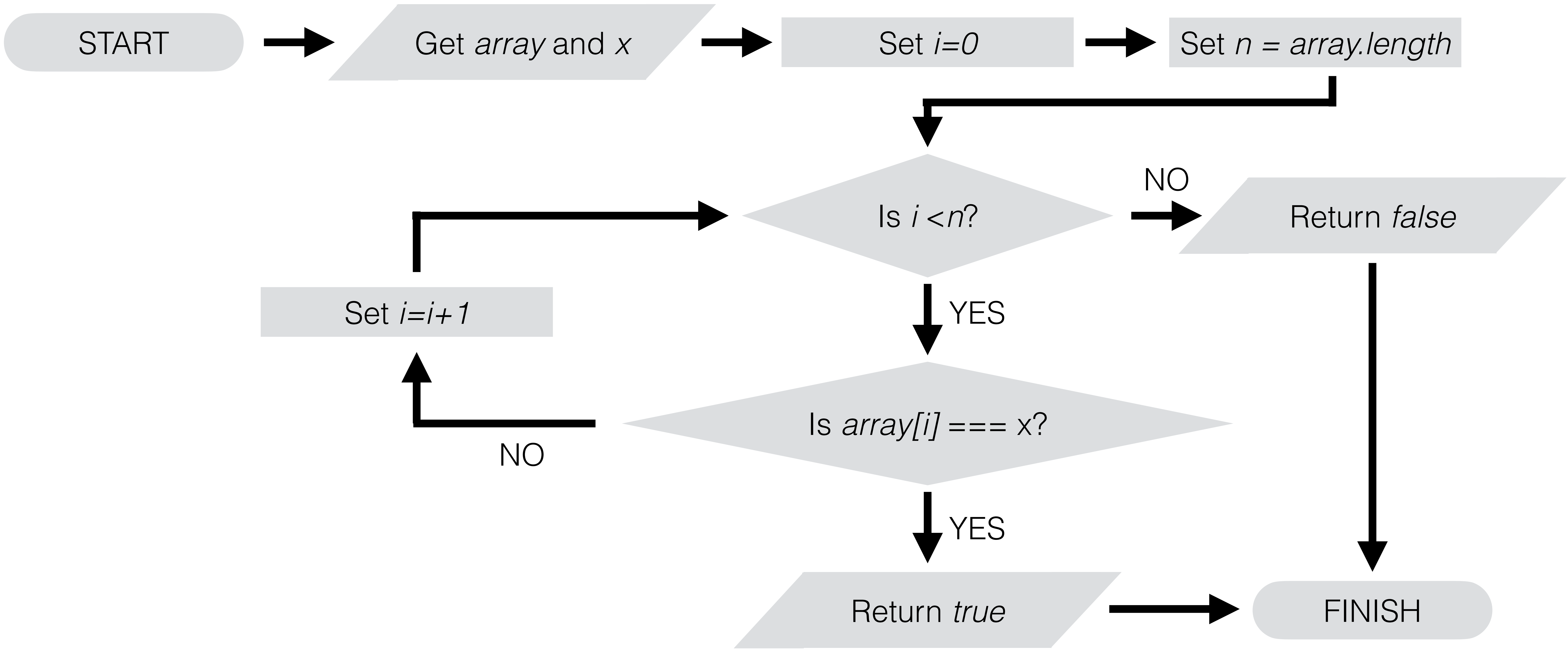
JavaScript Arrays

Is there an element with the value x?



JavaScript array

Boolean



START

Get *array* and *x*

Set *i=0*

Set *n* = *array.length*

Is *i* <*n*?

NO

Return *false*

YES

Set *i=i+1*

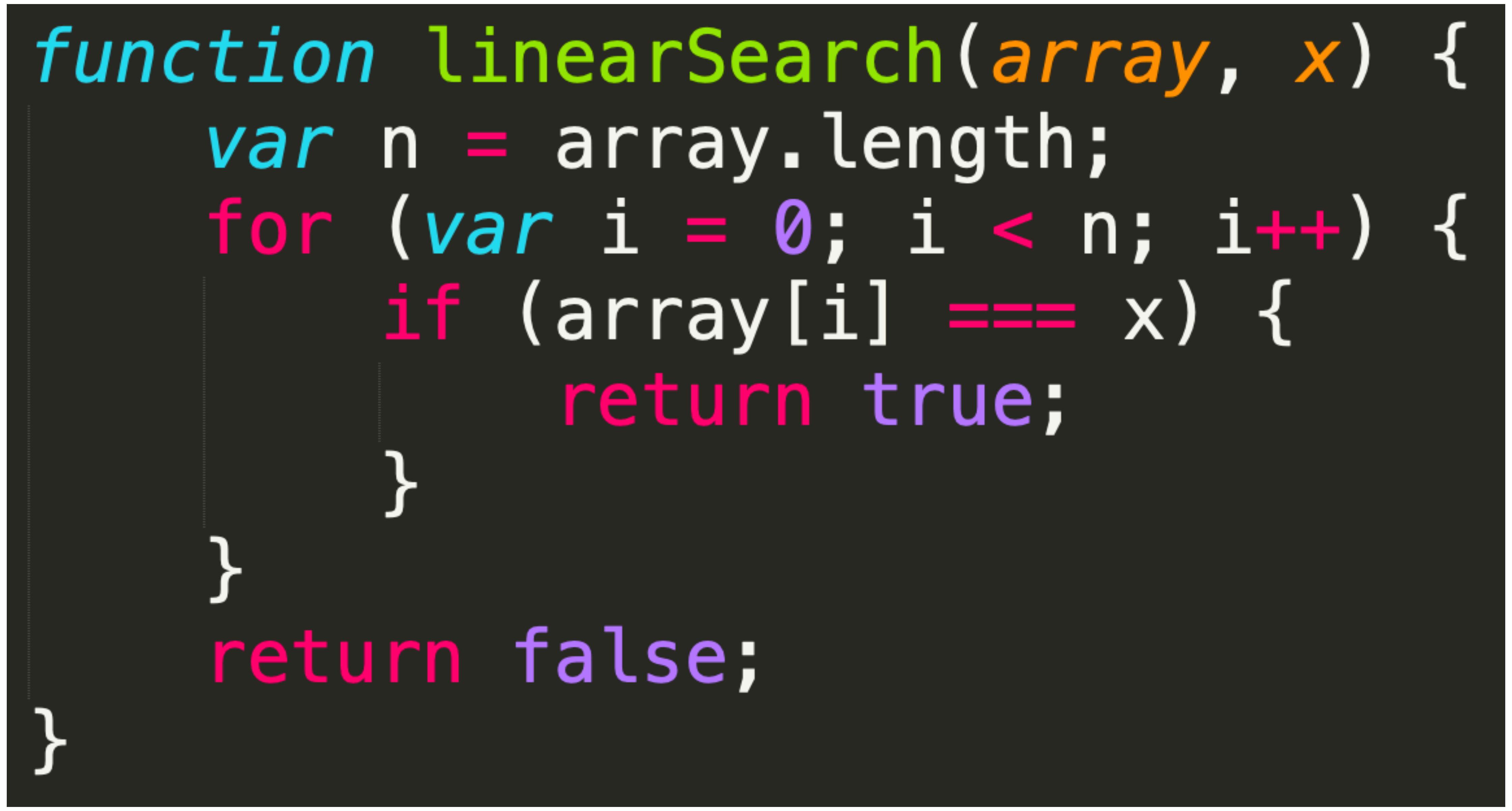
Is *array[i]* === x?

NO

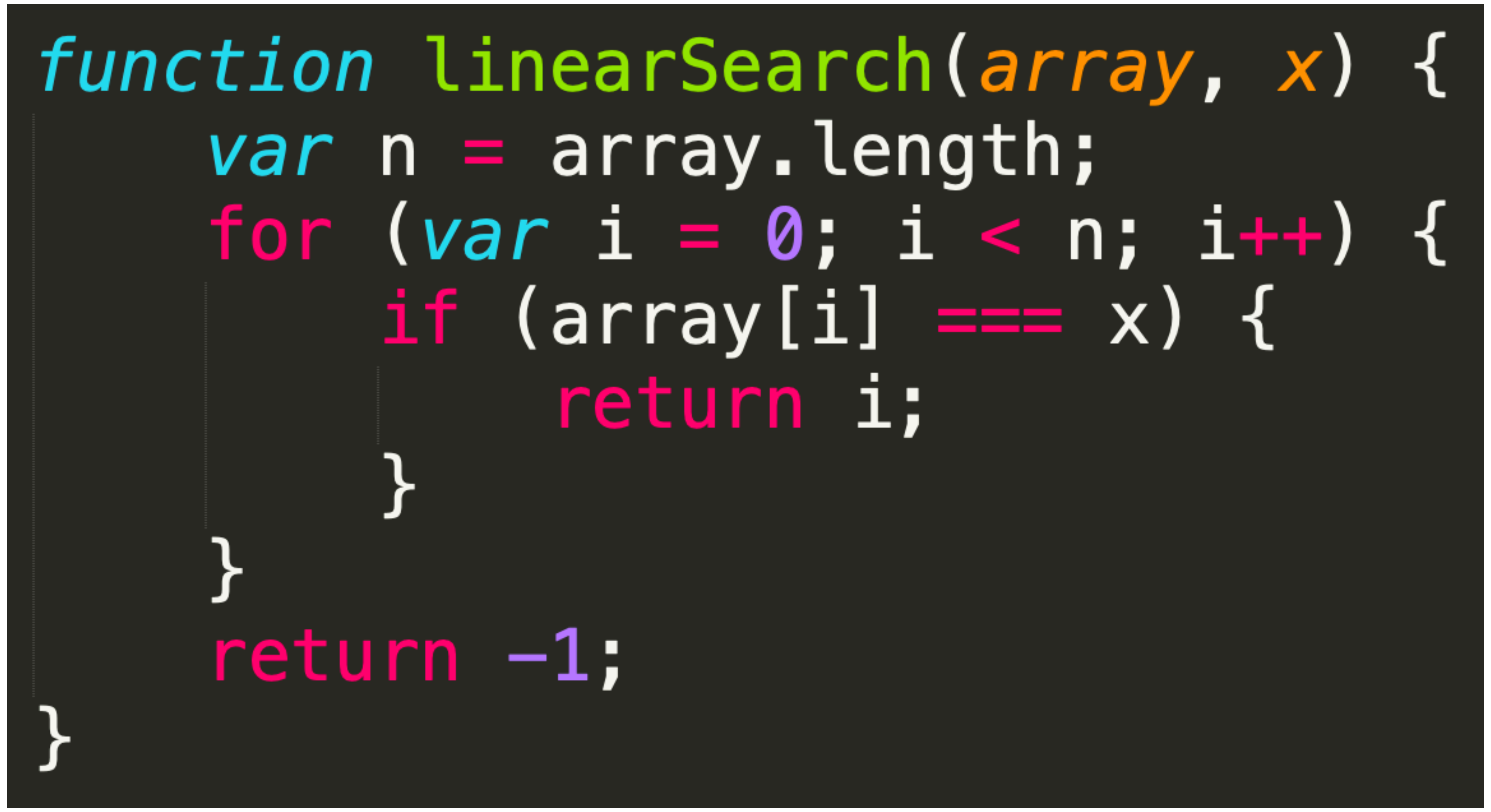
YES

Return *true*

FINISH



```
function linearSearch(array, x) {
    var n = array.length;
    for (var i = 0; i < n; i++) {
        if (array[i] === x) {
            return true;
        }
    }
    return false;
}
```

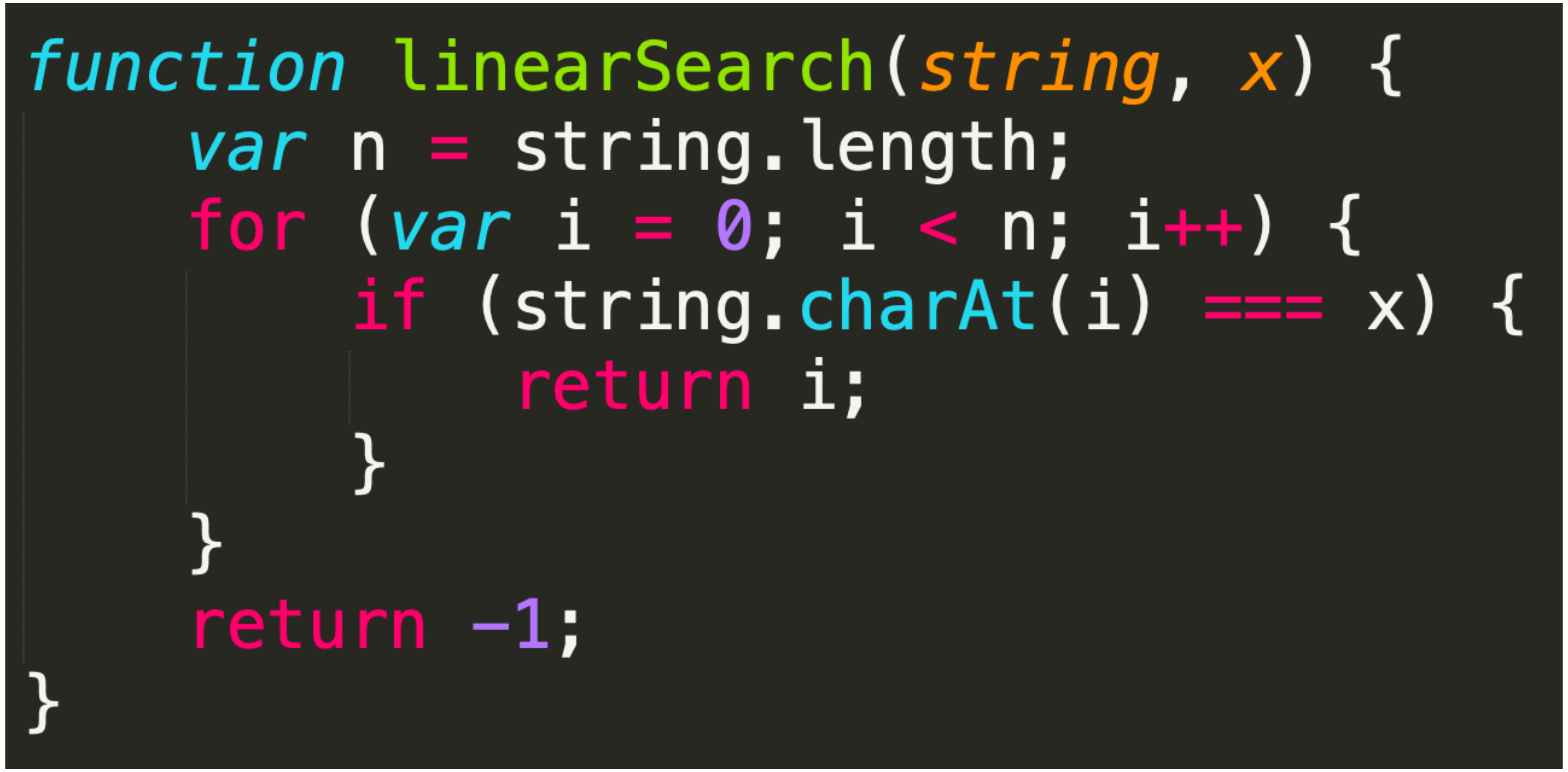


```
function linearSearch(array, x) {
    var n = array.length;
    for (var i = 0; i < n; i++) {
        if (array[i] === x) {
            return i;
        }
    }
    return -1;
}
```

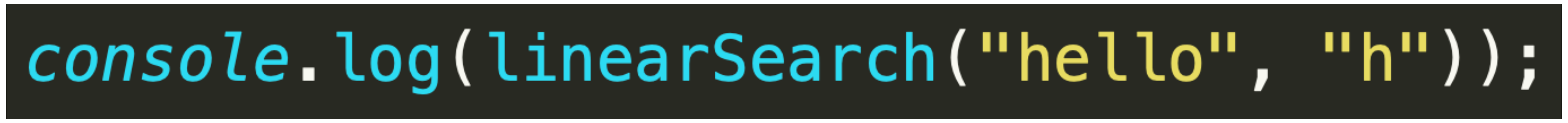
Linear Search Algorithm can be applied to JavaScript Strings

Strings behave similarly to arrays (elements can only store characters)

Must be a single character



```
function linearSearch(string, x) {
    var n = string.length;
    for (var i = 0; i < n; i++) {
        if (string.charAt(i) === x) {
            return i;
        }
    }
    return -1;
}
```



```
console.log(linearSearch("hello", "h"));
```

Review Seminar

How can we look for strings inside other strings?

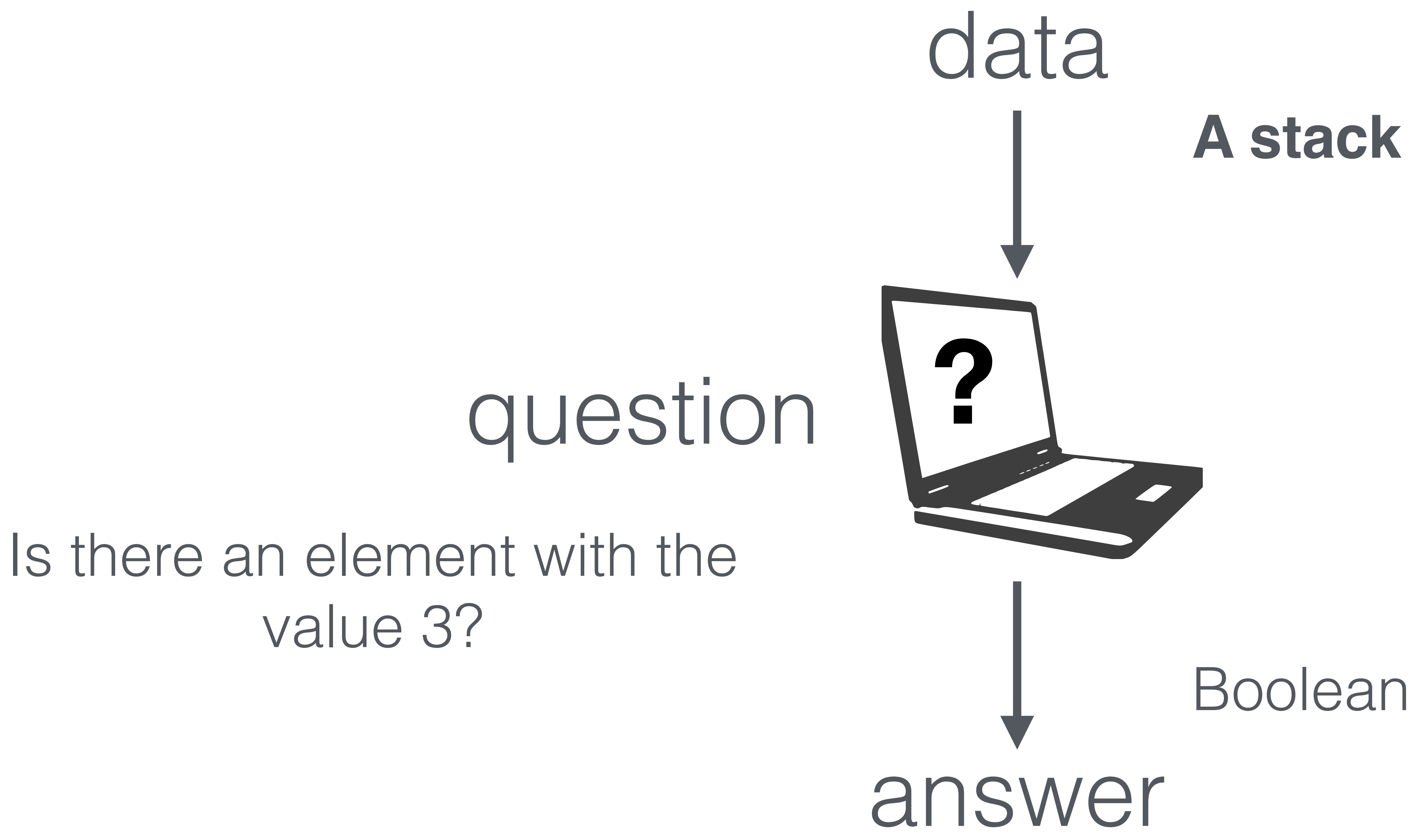
e.g. “Hello” inside “Hello, World!”

Today

1. Searching vectors and dynamic arrays

2. **Searching stacks and queues**

A problem



data

A stack

question

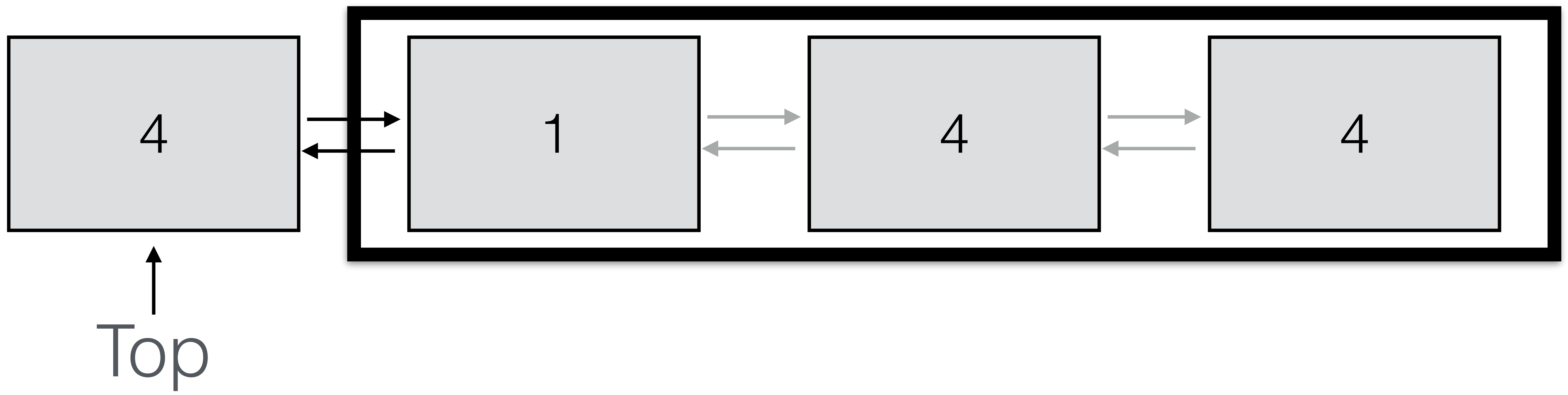
Is there an element with the value 3?

Boolean

answer

How do you solve this using the operations allowed by a stack?

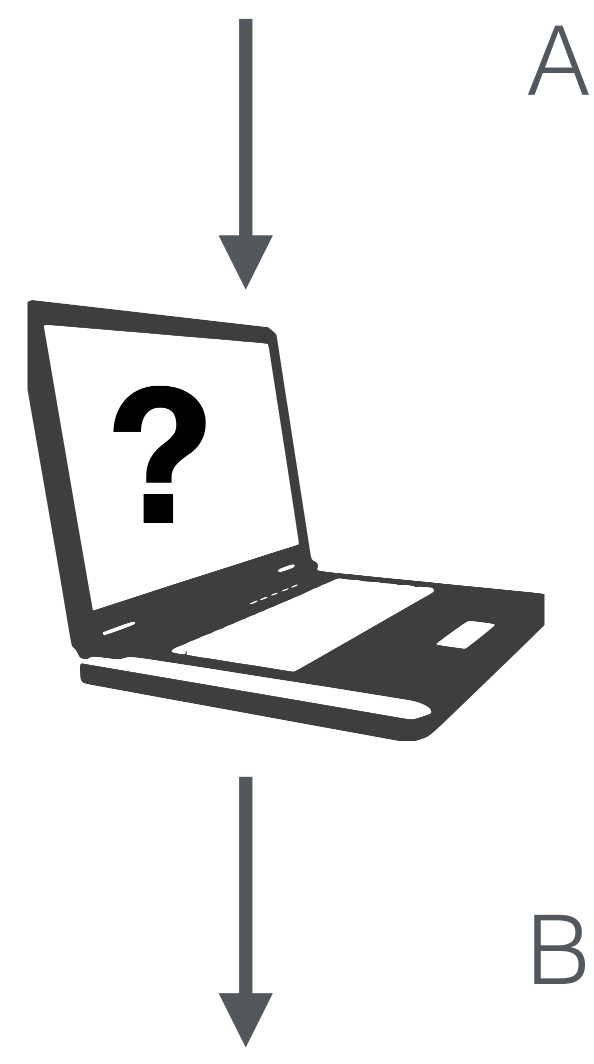
Stacks



Allowed operations:

push![o]	Adds a new element to the top with value o
peek	Reads out the value of the top element
pop!	Removes top element and returns its value
empty?	Checks if stack is empty

A problem



data

A stack

question

Is there an element with the
value 3?

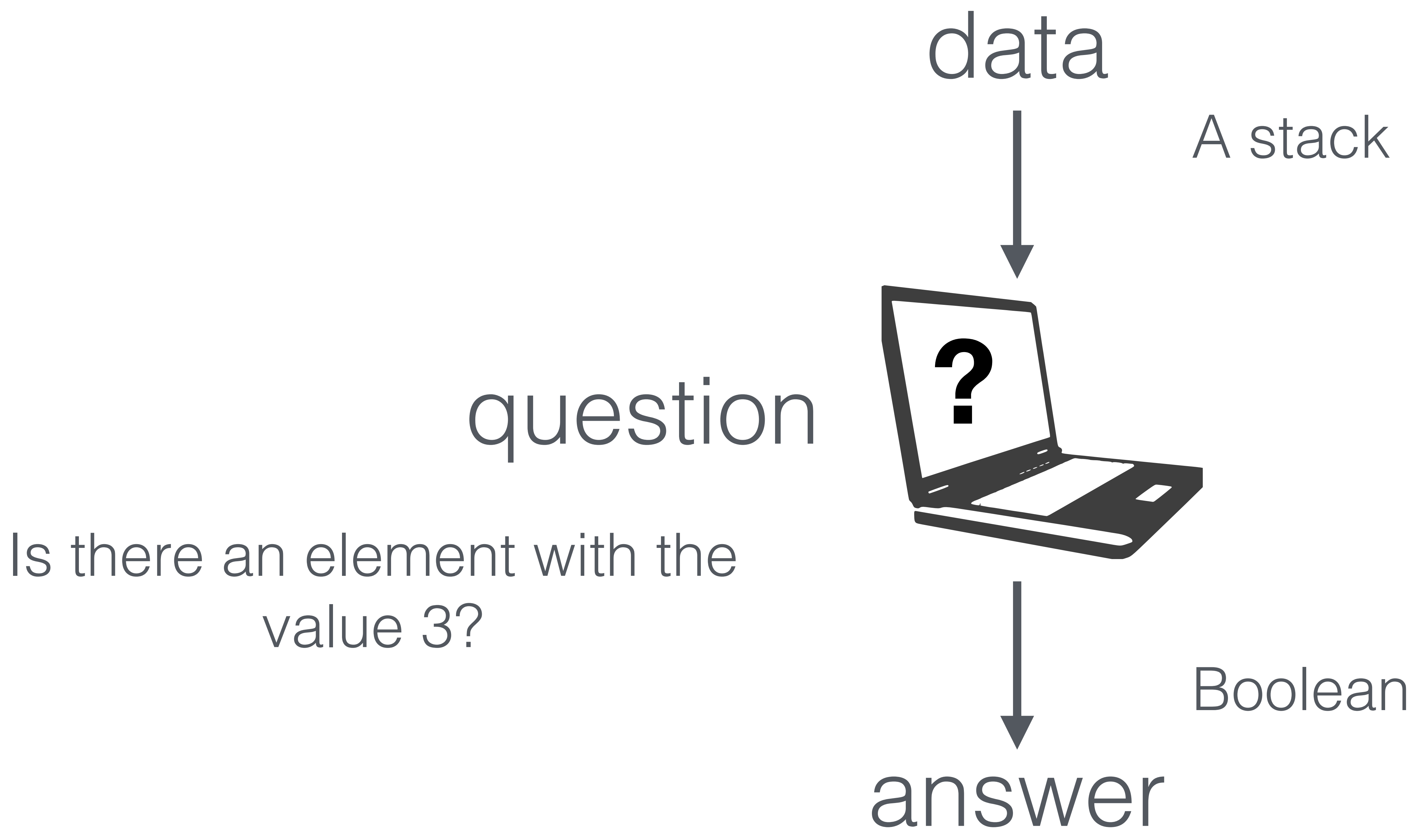
Boolean

answer

Can’t we just use the Linear Search Algorithm?

We can’t select arbitrary elements of a stack!

A problem



data

A stack

question

Is there an element with the value 3?

Boolean

answer

Can’t we just use the Linear Search Algorithm?

Put all data (using pop!) from stack in an array
Search that array using Linear Search

A problem

data

A stack

question

Is there an element with the value 3?

Boolean

answer

Can’t we just use the Linear Search Algorithm?

Is there another way?

Searching stacks

Just peek! then pop! to see if value is stored in element

This could completely destroy the stack

Searching stacks

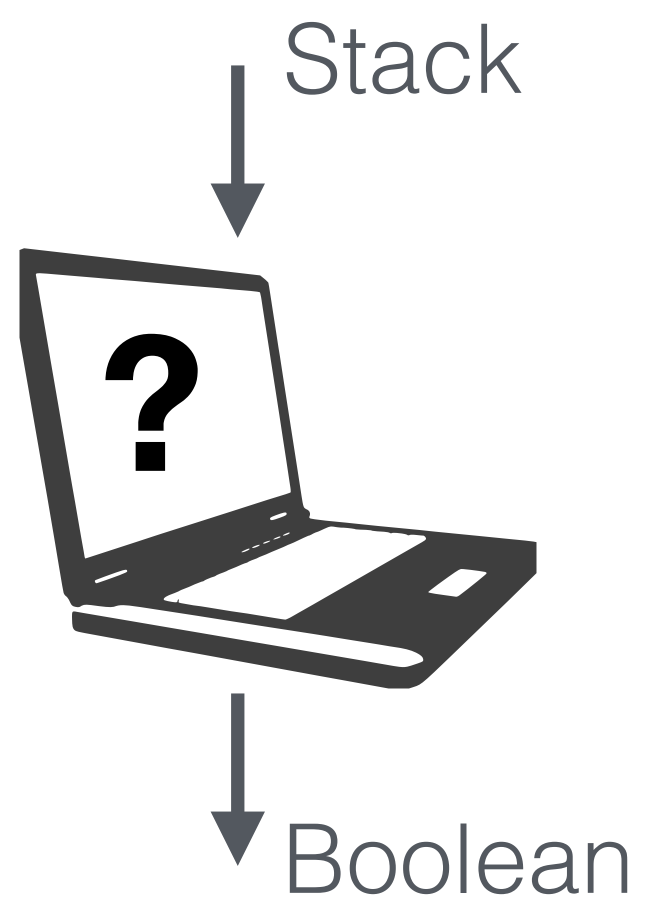
Just peek! then pop! to see if value is stored in element

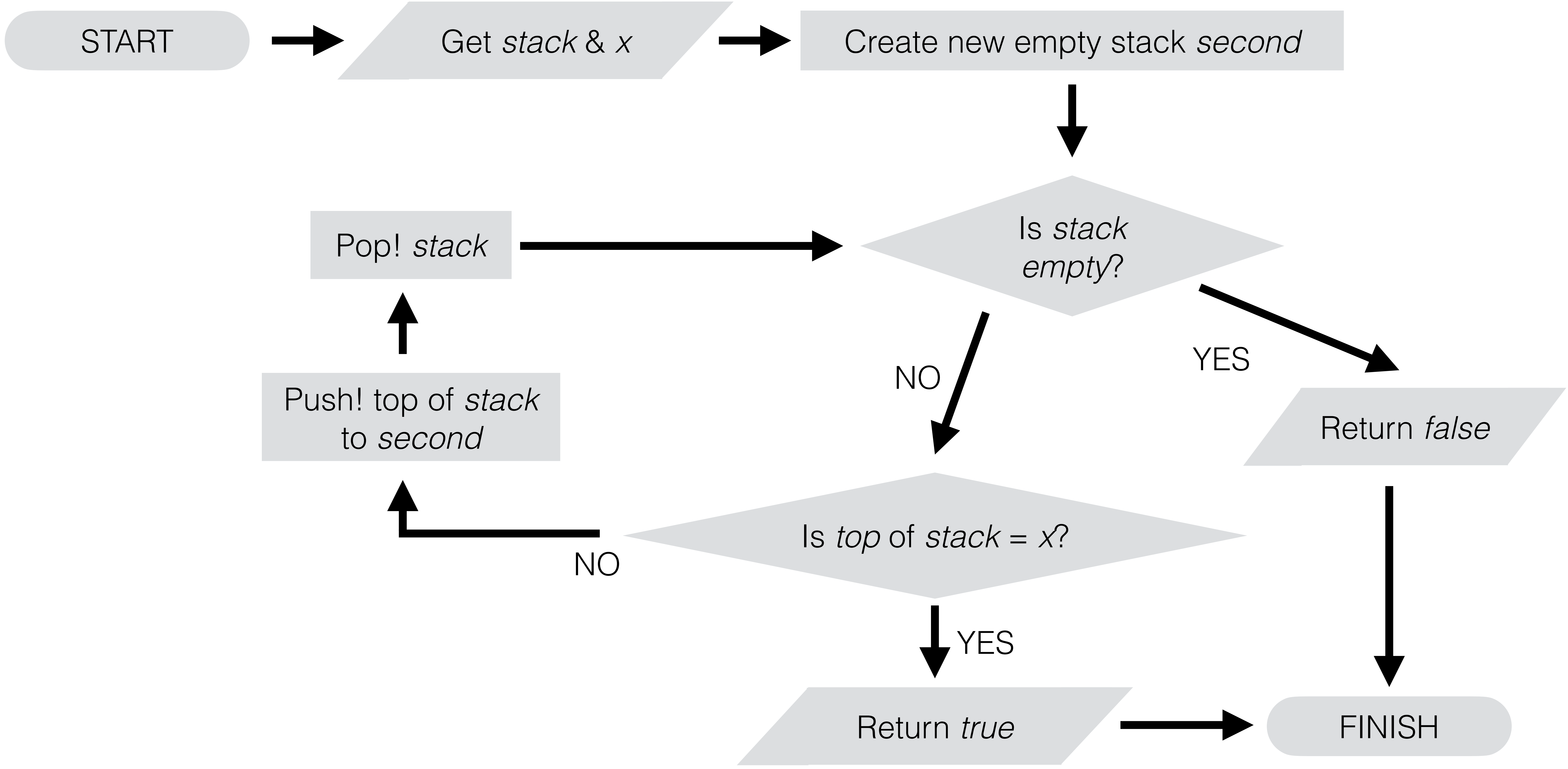
This could completely destroy the stack

Use a second stack!

Everything popped from initial stack is pushed to second stack

Is there an element with the value x?





START

Get *stack* & *x*

Create new empty stack *second*

Is *stack* *empty*?

YES

Return *false*

NO

Is *top* of *stack* = *x*?

YES

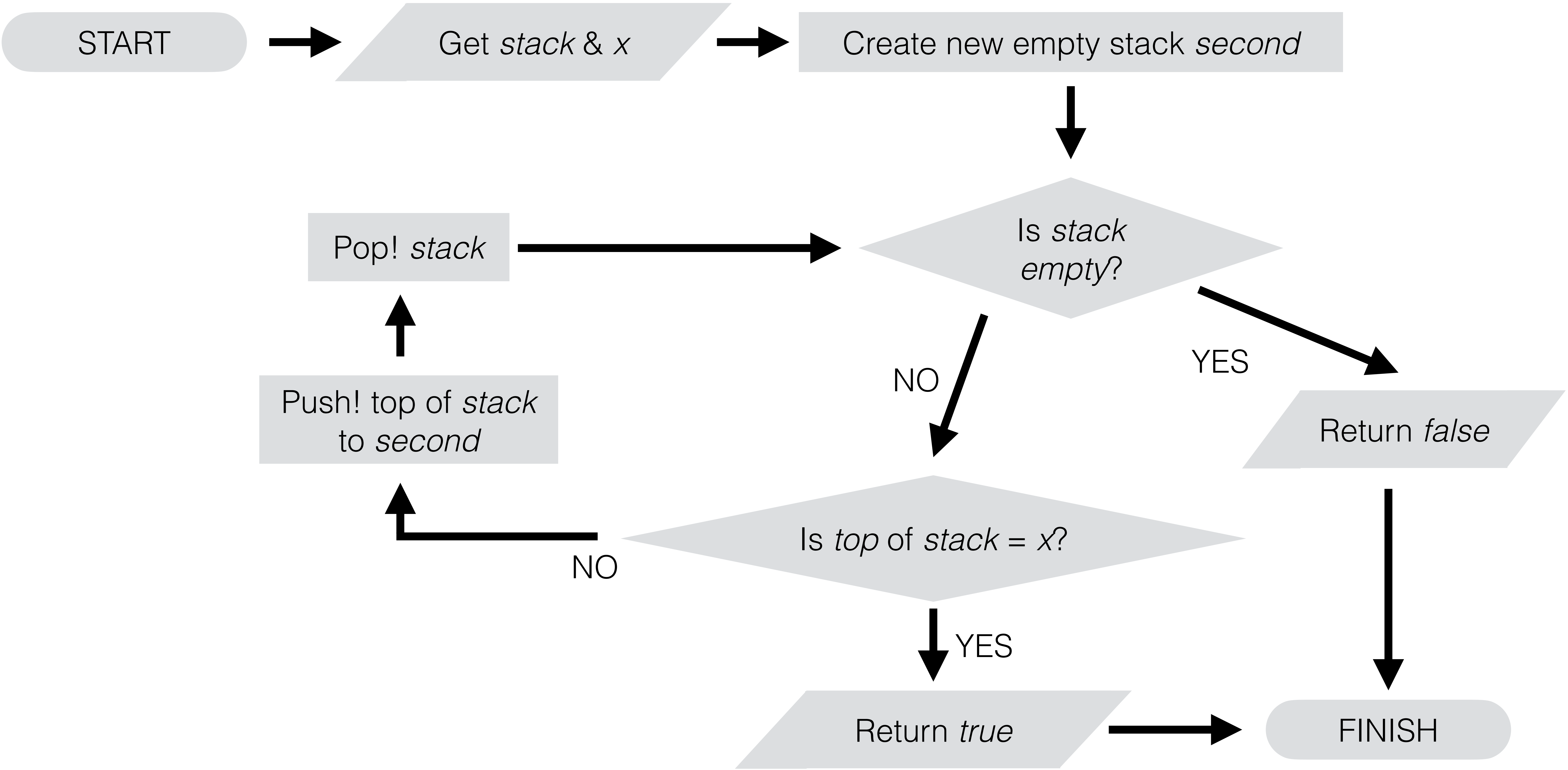
Return *true*

FINISH

NO

Push! top of *stack* to *second*

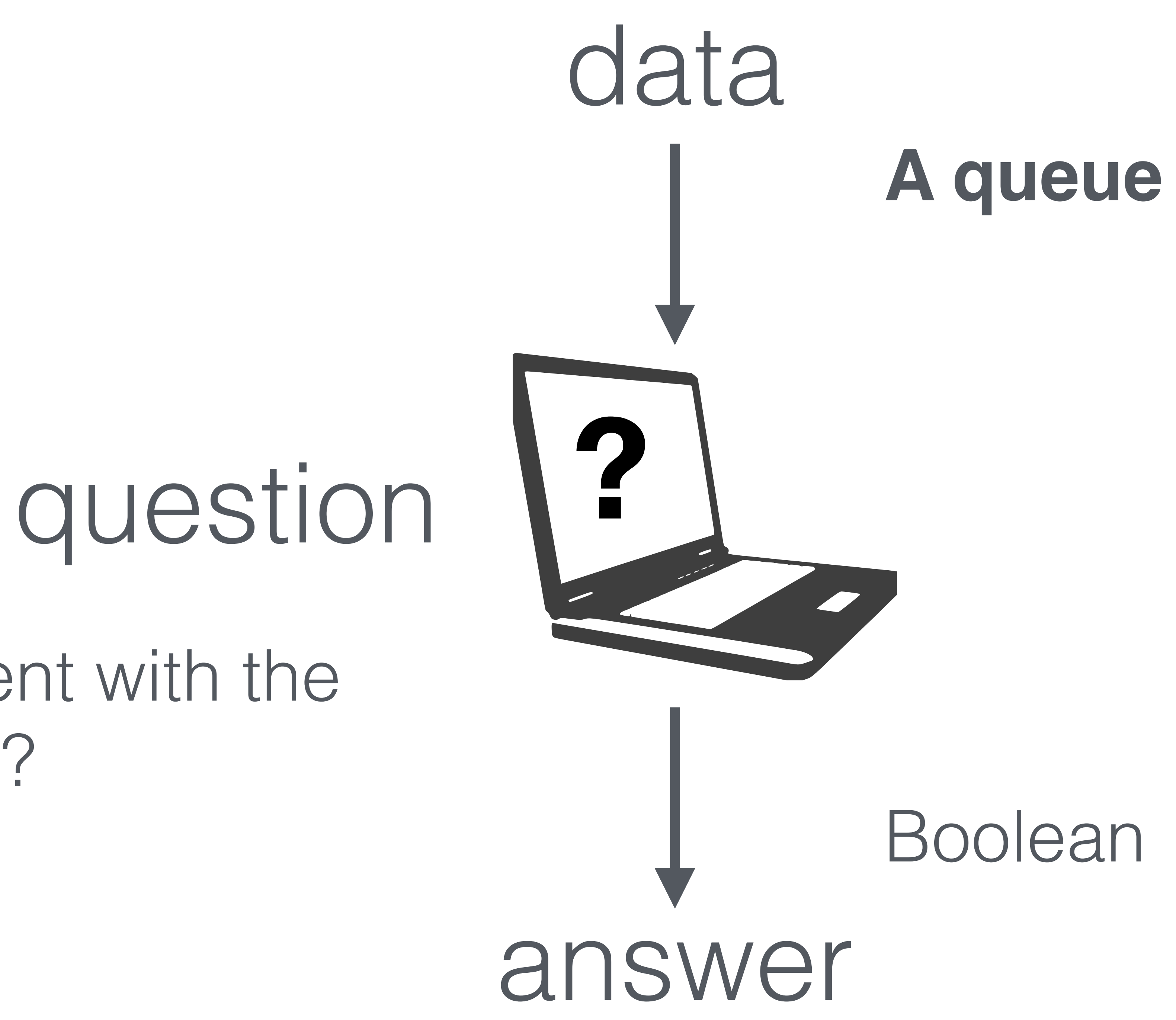
Pop! *stack*



After this, some of the elements of the original *stack* are reversed in *second*

Push values back into original stack

A problem



data

A queue

question

Is there an element with the value 3?

answer

Boolean

How do you solve this using the operations allowed by a queue?

Searching queues

How do you solve this using the operations allowed by a queue?

Put all data (using dequeue!) from queue in an array
Search that array

Is there another way?

Use a second queue?

Searching queues

How do you solve this using the operations allowed by a queue?

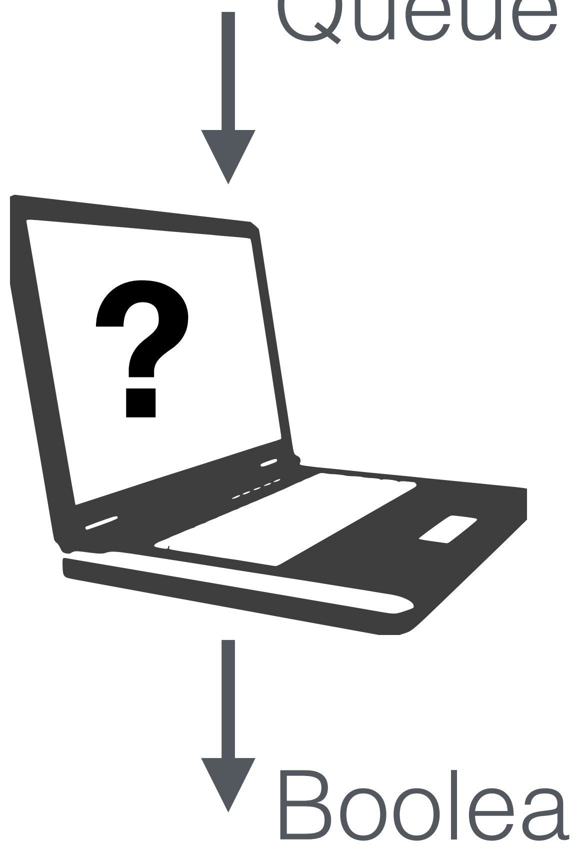
Put all data (using dequeue!) from queue in an array
Search that array

Is there another way?

Yes, just like with a stack

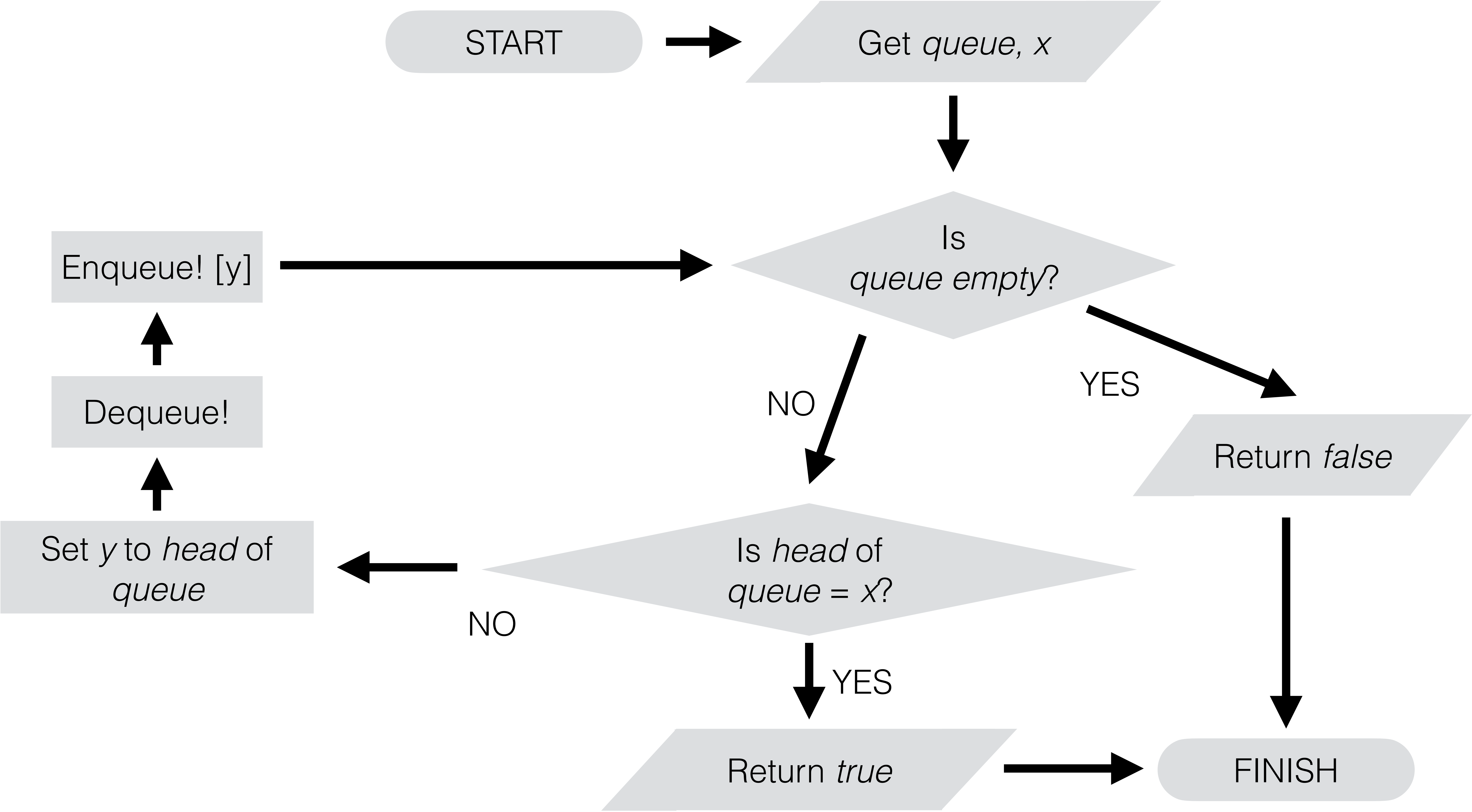
Can we do better?

Queue



Is there an element with the value x?

Boolean



START

Get *queue, x*

Is *queue empty*?

NO

YES

Return *false*

Is *head* of *queue = x*?

NO

YES

Return *true*

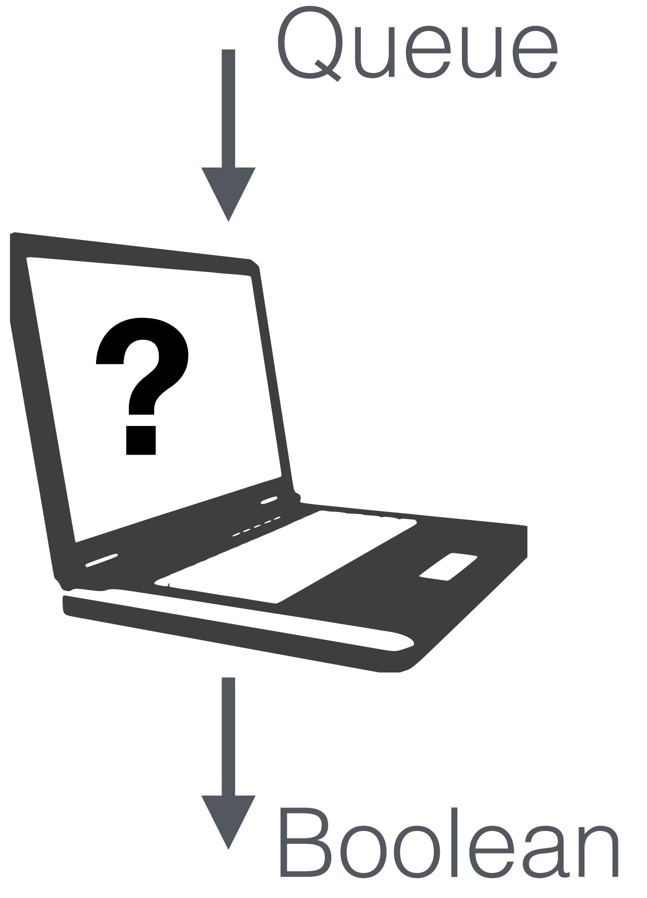
FINISH

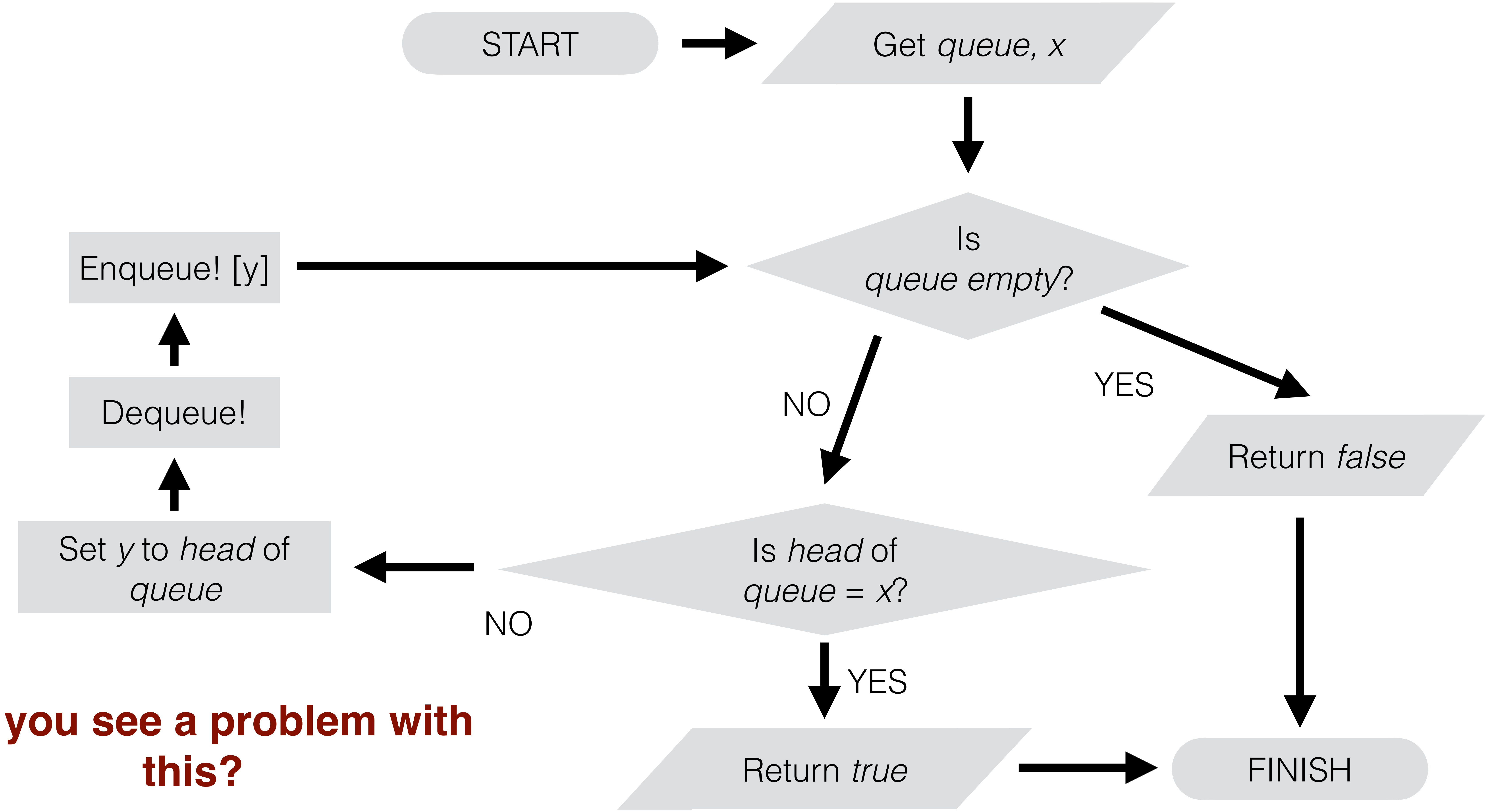
Set *y* to *head* of *queue*

Dequeue!

Enqueue! [y]

Is there an element with the value x?





Can you see a problem with this?

Searching

Depending on our abstract data structure different searching algorithms are required

Vectors and dynamic arrays use **Linear Search algorithm**, stacks and queues cannot

We can *also* search stacks and queues with different algorithms

Being able to go between data structures will improve your problem solving and computing skills

Problem 3:

You have been asked to organise a lottery for the national Chess Boxing and Knitting club, which has 589 members

It is decided that the lottery will be based on having a unique birthday

- You win if no one else has your birthday
- No one wins if they share birthdays

You are given a list of all members' birthdays along with their membership number as a table

Write an algorithm and/or JavaScript implementation to determine who will win the lottery