Types, Values, Operators

Part 1

$$a = b + 3$$

Statements

A group of

- words
- numbers
- operators

that performs a specific task is a statement.

Expressions

Statements are made up of one or more expressions.

a = b + 3

3 is a number literal

= and **+** are operators

a and **b** are variables

Types

number

string

boolean

null

undefined

Types: number

```
7, -7

0, -0

0.19, -0.19
```

Types: number

```
7, -7

0, -0

0.19, -0.19

5.1E+7

Infinity(Number.POSITIVE_INFINITY), -
Infinity(Number.NEGATIVE_INFINITY)

Number.MAX_VALUE, Number.MIN_VALUE

NaN
```

Types: number - base Systems

```
Base-10
       0, 3, 100


Base-16 (hexadecimal)
       0x9, 0xA,0x12E


Base-8 (octal)
       00, 07, 0112
```

Types: String

```
"Moon"

'Jupiter'

'Saturn\'s moon'

`Milky
  Way`
```

Types: boolean

```
true

false
```

Type Conversion

number -> string

string -> number

number -> boolean

boolean -> number

string -> boolean

boolean -> string

Type Conversion: Falsy values

- `""`
- `0, -0, NaN`
- `null, undefined`
- `false`

Variables

Containers to store the data.

var

let

const

Types: Null & undefined

null is a language keyword that evaluates to a special value that is usually used to indicate the absence of a value.

Types: Null & undefined

null is a language keyword that evaluates to a special value that is usually used to indicate the absence of a value.

undefined is the value of variables that have not been initialized or of the property of an object which does not exist.

Operators: Unary Operators

`++`(increments)

`--`(decrements)

`-`(sign changes)

`+`(converts)

`typeof`

Operators: Binary Arithmetic Operators

+

−

*

/

%

Operators: Assignment Operators

=

+=

-=

*=

/=

%=

Operators: Binary Comparison Operators

`==`, `===` (equality)

`!=`, `!==` (inequality)

`>`, `>=` (greater than, greater than or equal)

`<`, `<=` (less than, less than or equal)

Operators: Logical Operators

Boolean Algebra

! (logical not)

&& (boolean and)

|| (boolean or)

Ternary operator

?:

Arithmetics: Math

```
Math.pow(n, m)
Math.sqrt(n)
Math.ceil(n)
Math.floor(n)
Math.log(n)
Math.PI
Math.E
```

Conditionals

```
if(expression)
        statement
```

```
if(expression)
        statement

else if(another expression)
        statement

else
        statement
```

Conditionals

```
switch(expression) {
        case value:
                statements
                break;

    default:
                statements

}
```

Thanks!

Types, Values, Operators

Part 2

Repetitions . Loops

Loops help to

- repeat some code
- reduce code
- Find a generic solution

Repetitions . Loops

What if you need to print all numbers between 0 and 1000?

While

```
while (condition) {
        code block to be executed
}
```

Do While

```
do {
        code block to be executed
}
while (condition);
```

For

```
for (statement 1; statement 2; statement 3) {
    code block to be executed
}
```

Infinite Loops

Guess how can we create infinite loops

Infinite Loops

1. `for (;;) {}`
2. `while (true) { //your code }`

Breaking Out of a Loop

Sometimes you need to stop loops or terminates execution of the statements in the current iteration of the current or labeled loop, and continues execution of the loop with the next iteration.

For these cases we have ***break*** and ***continue*** operators.

Break

The **break statement** terminates the current loop, switch and transfers program control to the statement following the terminated statement

```
while (i < 5) {
 i++;

 if (i === 3) {
   break ;
 }

 n += i;
}
```

Continue

The **continue statement** terminates execution of the statements in the current iteration of the current or labeled loop, and continues execution of the loop with the next iteration.

```
while (i < n) {
 i++;

 if (i % n === 0) {
   continue;
 }
console.log(i);
 n += i;
}
```

Types, Values, Operators

Part 3

Nested loops

```
for( ;condition1; ) {
      statement1;
      for( ;condition2; )
{
            statement2;
      }
}
```

```
while(condition1) {
      statement1;
      while(condition2) {
            statement2;
      }
}
```

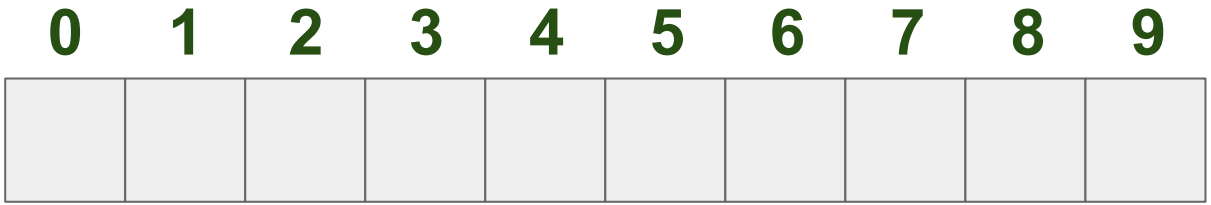
Nested loops

```
1 for(var i = 0; i < 3; i++){
2     console.log(i);
3     for(var j = 0; j < 3; j++) {
4         console.log(j);
5     }
6 }
```

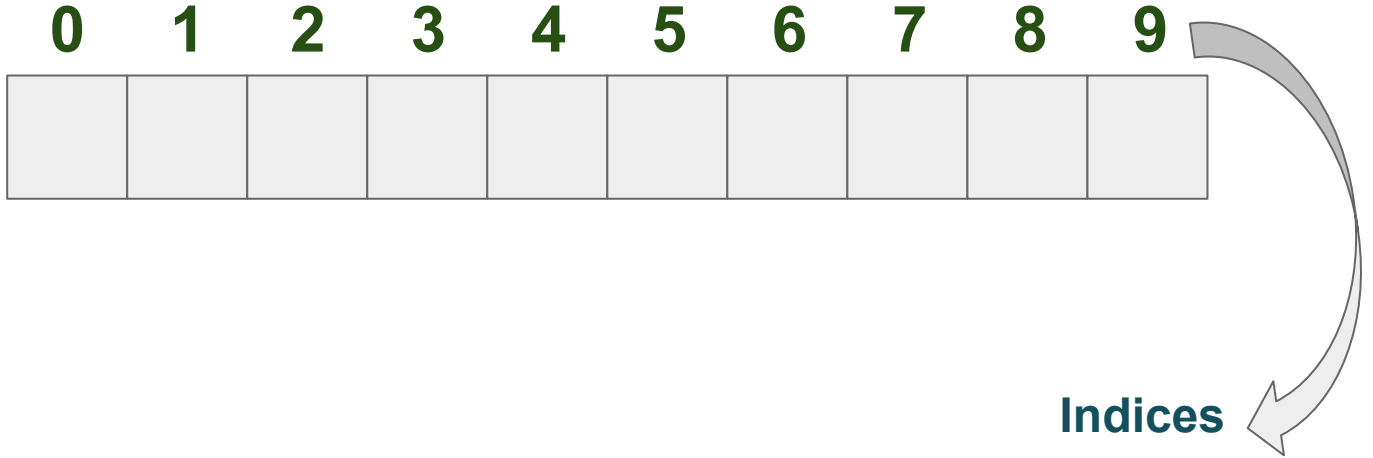
EXAMPLE

Arrays

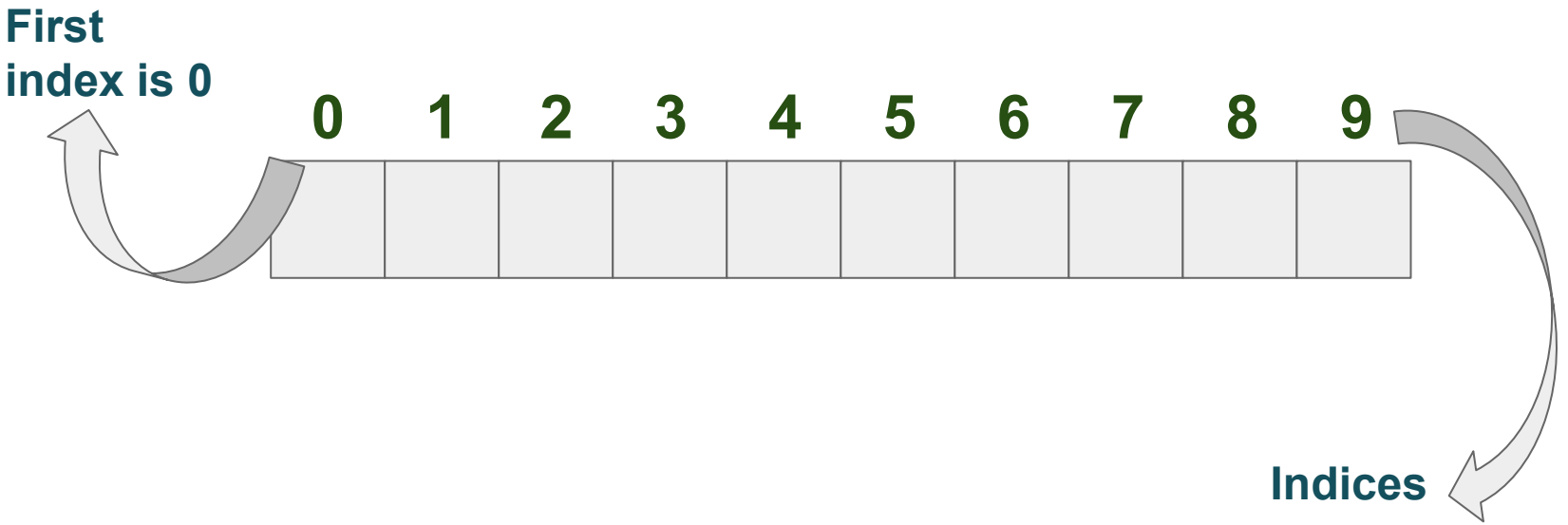
Arrays



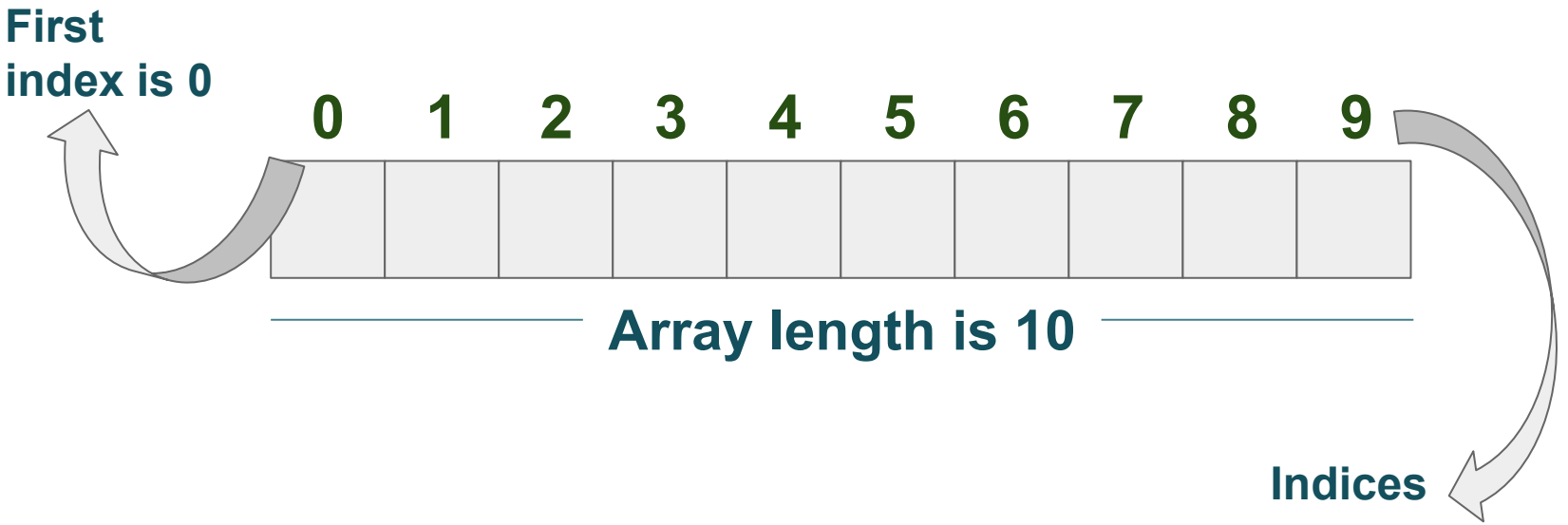
Arrays



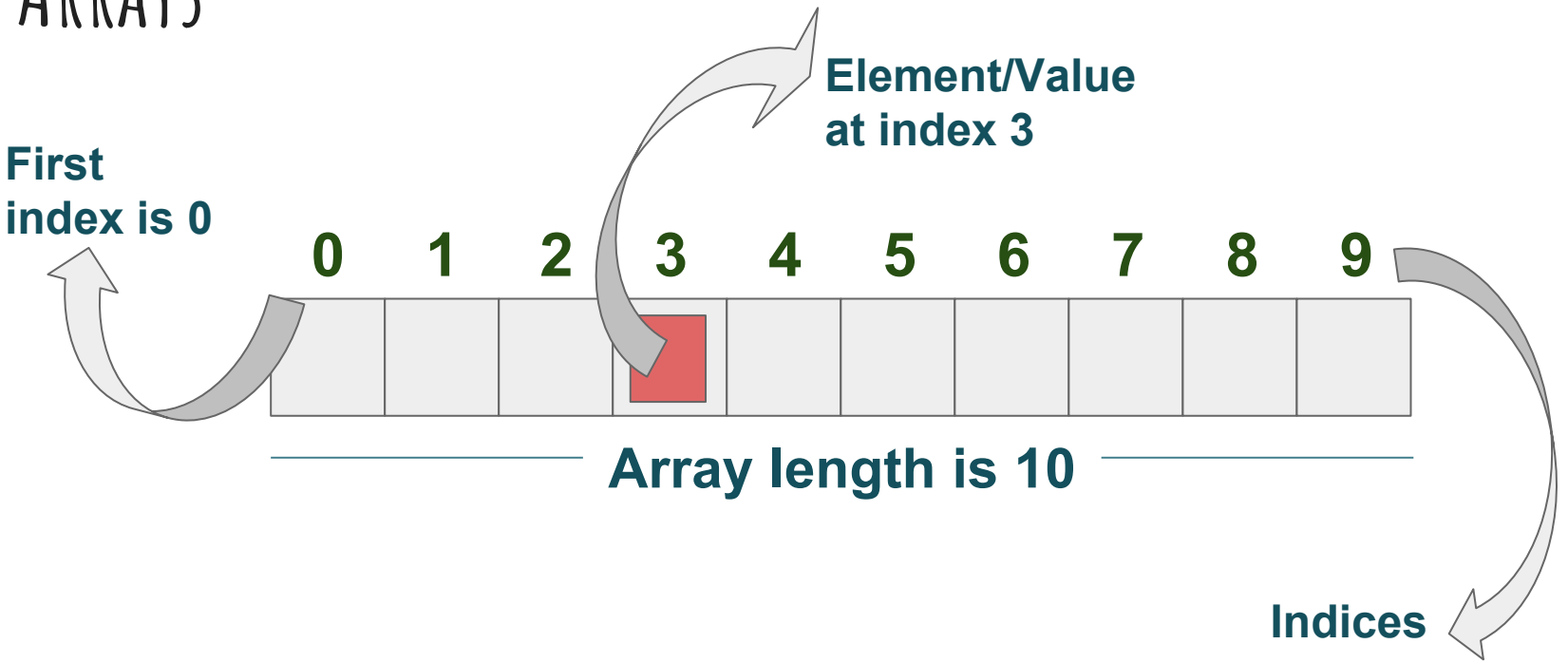
Arrays



Arrays



Arrays


Element/Value
at index 3
First
index is 0
0 1 2 3 4 5 6 7 8 9
Array length is 10
Indices

Arrays: Declare, Get and Change

```
var arr = [10, 20, 30, 40, 50, 60];
var l = arr.length;
var element0 = arr[0];
var element4 = arr[2];
arr[1] = 25;
```

EXAMPLE

Arrays: Pop and Push

```
var arr = [‘ok’, ‘wow’, ‘hi’, ‘yeah’];
var last = arr.pop();
var newLast = ‘bye’;
arr.push(newLast);
```

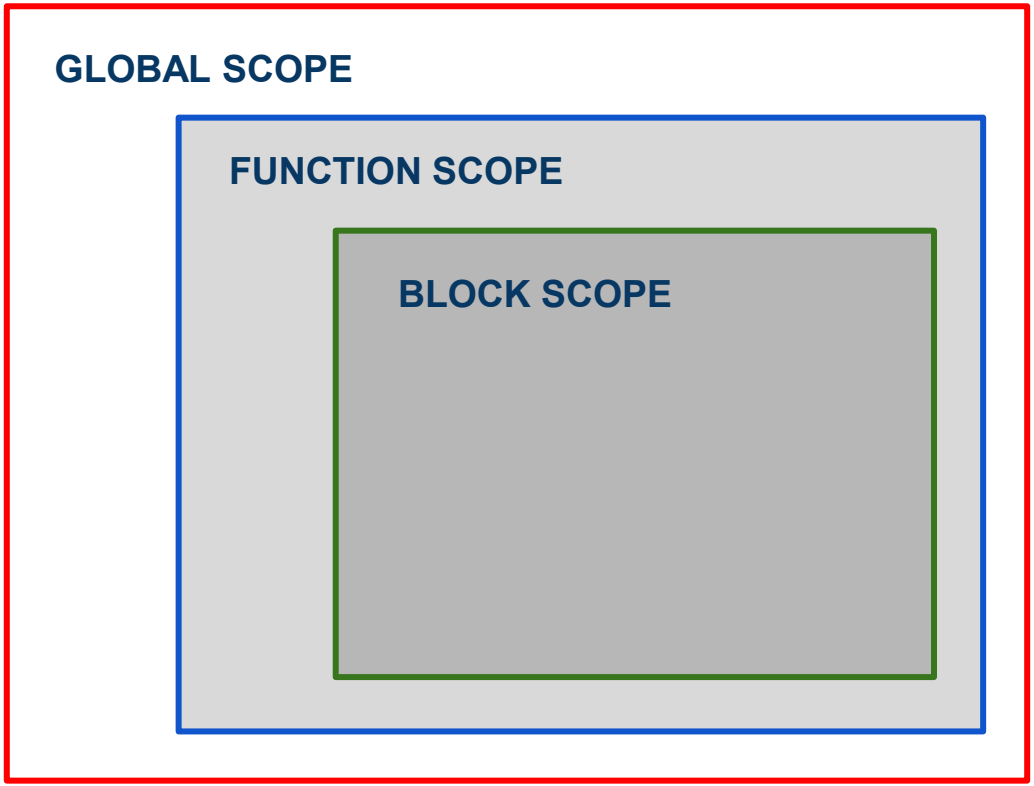
EXAMPLE

Strings

Strings can behave them like arrays.
But there are differences!

EXAMPLE

SCOPE



EXAMPLE

Let, const

Declare variables that are limited in scope to the block.

EXAMPLE

Var

The scope of a variable declared with var is its current execution context, [which is either the enclosing function or,] for variables declared outside any function, global.

EXAMPLE

Hoisting

Variable declarations are processed before any code is executed.

Declaring a variable anywhere in the code is equivalent to declaring it at the top.

This also means that a variable can appear to be used before it's declared.

This behavior is called "hoisting".

Hoisting

let will hoist the variable to the top of the block. However, referencing the variable in the block before the variable declaration results in a ReferenceError.

Thank you

Functions

Part 1

What is a Function?

A function is a block of organized, reusable code that is used to perform a single, related action.

Functions we know

`console.log`

`prompt`

`Math.floor`

`Array.push`

Why Functions?

- to reduce repetition
- to associate names with subprograms
- to isolate these subprograms from each other
- to structure larger programs

Defining Functions

```
function functionName(functionArguments) {
  // function body
}
```

Declaration

```
var myFunction = function(...) {
        ...
}
```

```
function myFunction(...) {
        ...
}
```

EXAMPLE

Invocation

```
function functionName(functionArguments) {
  // function body
}

functionName(argumentValues);
```

Default values

If a parameter is not provided, then its value becomes undefined.

But you can declare default values for such cases.

EXAMPLE

Default values

Keep in mind!
Default values work only for undefined values. If value of the passed argument is null, it will remain null in the whole funcion.

EXAMPLE

Arguments And Parameters

- no declared types
- no type checking
- fewer arguments are ok. they set to undefined
- even sometimes desirable to have optionals, give them reasonable defaults

EXAMPLE

Return

```
return expression;
```

- may appear only within the body of a function
- it is a Syntax Error to appear anywhere else

Return

```
return;
return true;
return false;
return x;
return x + y / 3;
```

Function and Return

When the return statement is executed, the function that contains it returns the value of expression to its caller.

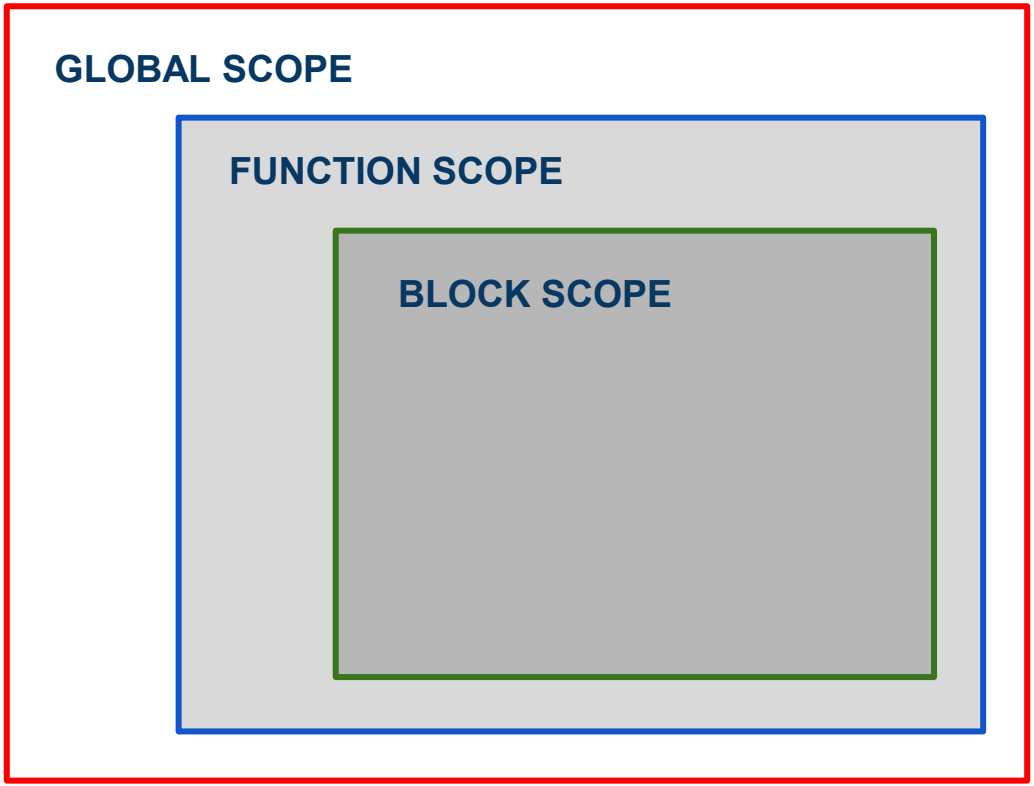
With no return statement, a function invocation simply executes each of the statements in the function body in turn until it reaches the end of the function and returns undefined.

EXAMPLE

Arrow functions

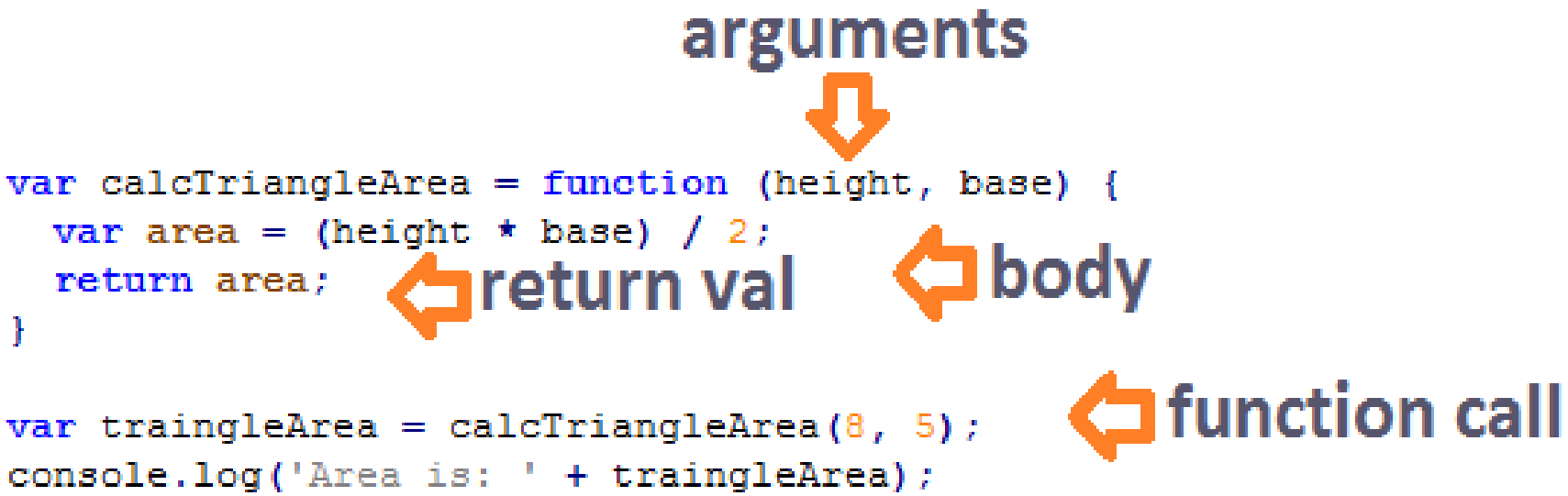
```
let func = (arg1, arg2, ...argN) => expression
```

SCOPE



EXAMPLE

Declaration, Arguments, Return, Call



Thank You!

Debugging, Call Stack and Recursion

Debugging and Debugger

Function Call stack

Call Stack

- a data structure
- which records function calls

RECURSION

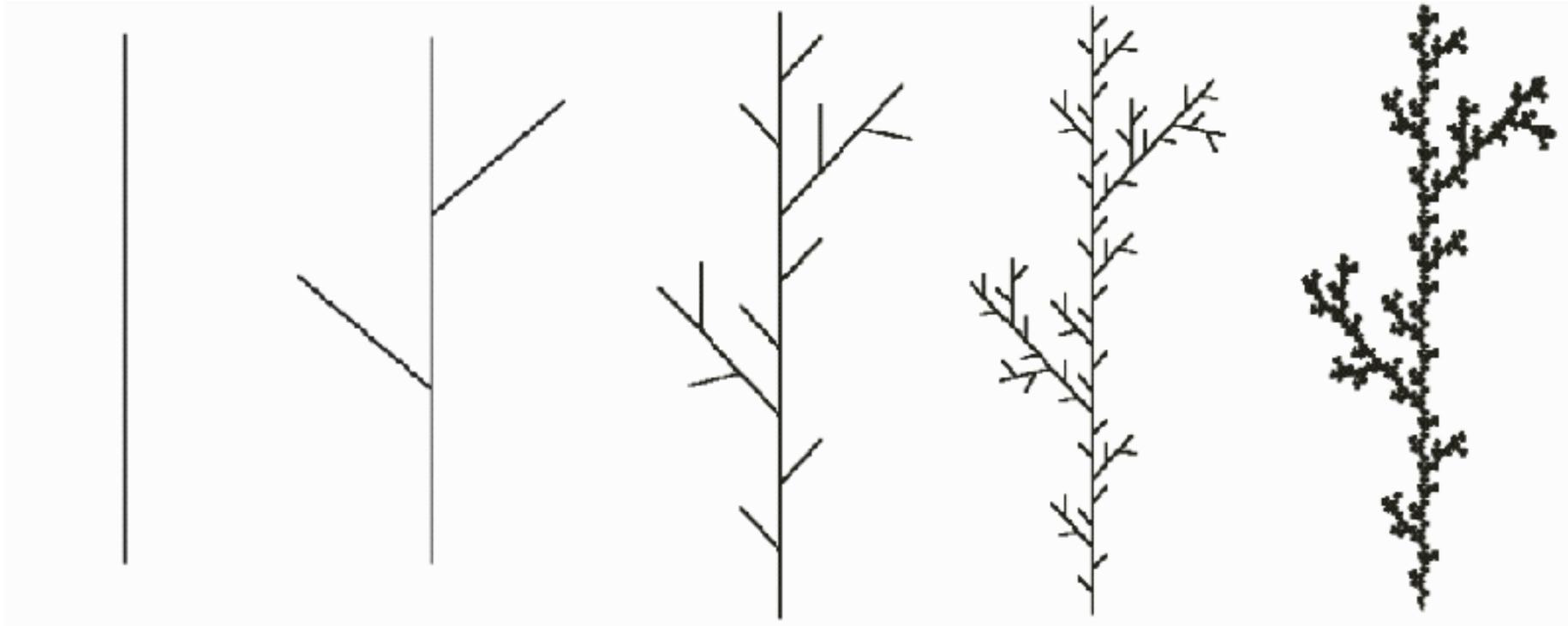
Recursion

In order to understand recursion you must first understand recursion!

© Unknown Philosopher

Recursion

It is a technique for creating figures which are defined by "replacement" rules.



Recursion in programming

Recursive function is a function that calls itself;

By calling itself more than once a function can produce multiple copies of itself.

Recursion in programming

The main thing about Recursion, is that

- It sounds simple, but it is complex in practice.
- It is perfect for complex problems, as it splits the problem into simple subproblems.

Recursion in programming

Base Case - can be solved directly,it is pretty trivial.

Recursive Case(Inductive - mathematical) - taking a piece of the problem and solving the subproblem recursively, which is identical to the original problem.

Recursion in JS

Leave Event - control statement that allows the function to exit the recursive loop.

EXAMPLE

Objects

Data Types

number

string

boolean

undefined

null

object

Type Object

An object is a composite value.

{[key]: value}

Object Literal

Object consists of properties.

key -> property name

value -> property value

object literal notation -> {}

EXAMPLE

Object Literal

Get property value with:

. operator

[] operator

Object Literal

Object is dynamic.

- You can add properties.
- You can delete properties.

EXAMPLE

Operator delete

EXAMPLE

IF OBJECT HAS THE PROPERTY

You can check against undefined!

Is that sufficient?

EXAMPLE

Object.hasOwnProperty
Key in object

EXAMPLE

For...in operator

```
for (key in object) {
    object[key]
}
```

EXAMPLE

Reference vs Value

EXAMPLE

Object destructuring

EXAMPLE

Spread Operator

EXAMPLE

Sorting
Array and String Methods

Complexity: Big O notation

The Big O is basically a way to describe an algorithms “abstract” performance in operation as the operation’s inputs grow in size toward infinity.

Bubble Sort

A. Compare each pair of adjacent elements from the beginning of an array and, if they are in reversed order, swap them.

B. If at least one swap has been done, repeat step A.

Bubble Sort

6 5 3 1 8 7 2 4

Bubble Sort Complexity

Worst-case performance - $O(n^2)$

Best-case performance - $O(n)$

Average performance - $O(n^2)$

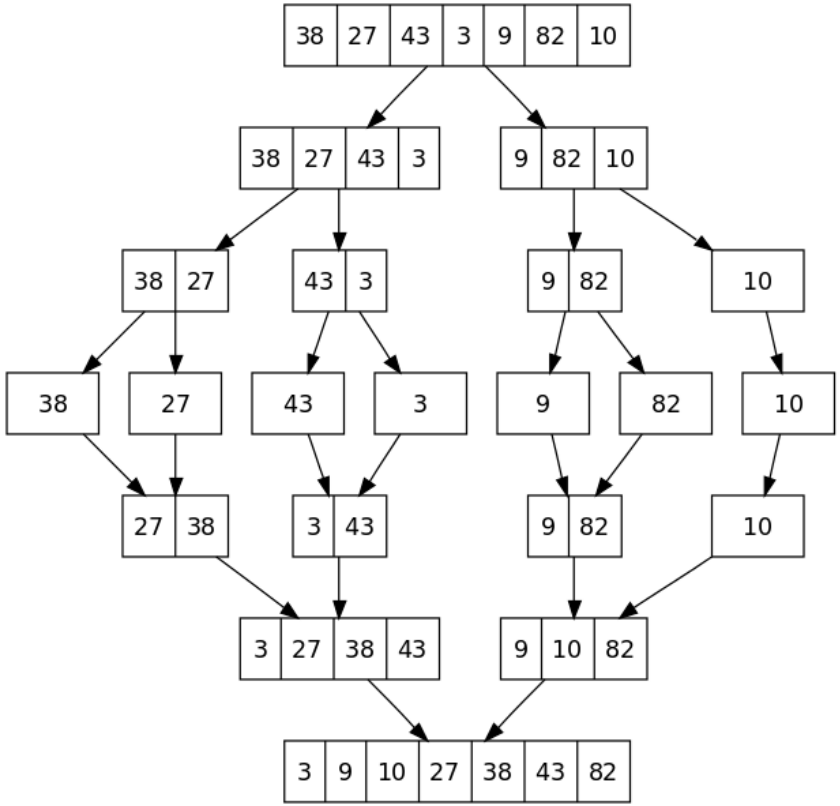
MergeSort

A. If the list is of length 0 or 1, then it is already sorted. Otherwise:
B. Divide the unsorted list into two sublists of about half the size.
C. Sort each sublist recursively by re-applying merge sort.
D. Merge the two sublists back into one sorted list.

MergeSort

6 5 3 1 8 7 2 4

MergeSort - Sorting Tree



MergeSort Complexity

Worst-case performance - O(n log n)

Best-case performance - O(n log n) typical,

O(n) natural variant

Average performance - O(n log n)

https://www.toptal.com/developers/sorting-algorithms

https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

http://www.stoimen.com/blog/2010/07/09/friday-algorithms-javascript-bubble-sort/

http://codingmiles.com/sorting-algorithms-bubble-sort-using-javascript/

http://www.stoimen.com/blog/2010/07/02/friday-algorithms-javascript-merge-sort/

HTTP
AJAX Requests
JSON

Asynchrony

Asynchronous Programming vs Parallel Programming

Asynchronous Code

In synchronous programs, if you have two lines of code (L1 followed by L2), then L2 cannot begin running until L1 has finished executing.

In asynchronous programs, you can have two lines of code (L1 followed by L2), where L1 schedules some task to be run in the future, but L2 runs before that task completes.

Callbacks

“Each function gets an argument which is another function that is called with a parameter that is the response of the previous action.”

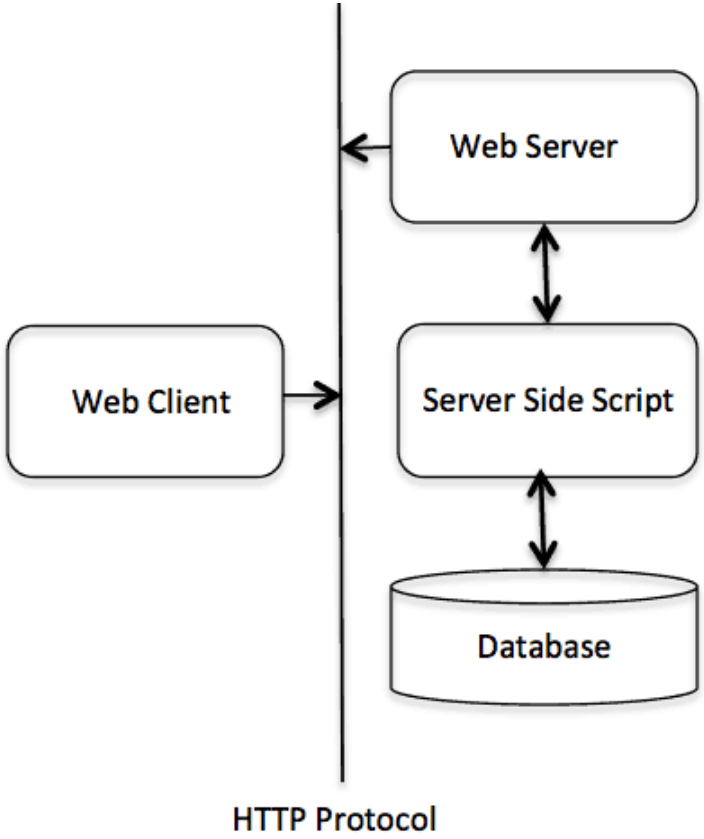
HTTP - Hypertext Transfer Protocol

HTTP is the foundation of data communication for the World Wide Web.

HTTP is the protocol that allows for sending documents back and forth on the web.

HTTP - Hypertext Transfer Protocol

HTTP - Hypertext Transfer Protocol



HTTP - Hypertext Transfer Protocol

HTTP is based on the client-server architecture model and a stateless request/response protocol that operates by exchanging messages across a reliable TCP/IP connection.

HTTP Request

An HTTP client sends an HTTP request to a server in the form of a request message.

The request method indicates the method to be performed on the resource identified by the given Request-URI

HTTP Request - Methods

GET

The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.

POST

A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms.

HTTP Response

After receiving and interpreting a request message, a server responds with an HTTP response message.

HTTP Response - Status

1	**1xx: Informational** It means the request was received and the process is continuing.
2	**2xx: Success** It means the action was successfully received, understood, and accepted.
3	**3xx: Redirection** It means further action must be taken in order to complete the request.
4	**4xx: Client Error** It means the request contains incorrect syntax or cannot be fulfilled.
5	**5xx: Server Error** It means the server failed to fulfill an apparently valid request.

JSON - JavaScript Object Notation

A *language-agnostic* lightweight data-interchange *format*

JSON - JavaScript Object Notation

JSON is built on two structures:

- A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.

- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

XHTTP Request - Get Example

```
var xmlHttp = new XMLHttpRequest();
    xmlHttp.onreadystatechange = function() {
        if (xmlHttp.readyState == 4 && xmlHttp.status == 200)
            console.log(xmlHttp.responseText);
    }
    xmlHttp.open("GET", 'https://ghibliapi.herokuapp.com/films/58611129-2dbc-4a81-a72f-
77ddfc1b1b49', true); // true for asynchronous
    xmlHttp.send(null);
```

Fetch API - GET Example

```
fetch('https://ghibliapi.herokuapp.com/films/58611129-2dbc-4a81-a72f-77ddfc1b1b49', {
    method: 'get'
})
    .then(function (response) {
        return response.json()
            .then(
                function (data) {
                    console.log(data);
                }
            )
    }).catch(function (err) {
    // Error :(
    console.log('bebebe');
});
```

Public APIs

https://github.com/toddmotto/public-apis

Promises

Promise

A Promise is an object representing the eventual completion or failure of an asynchronous operation.

Create Promise

```
new Promise( /* executor */
    function(resolve, reject) { ...
}
);
```

Executor

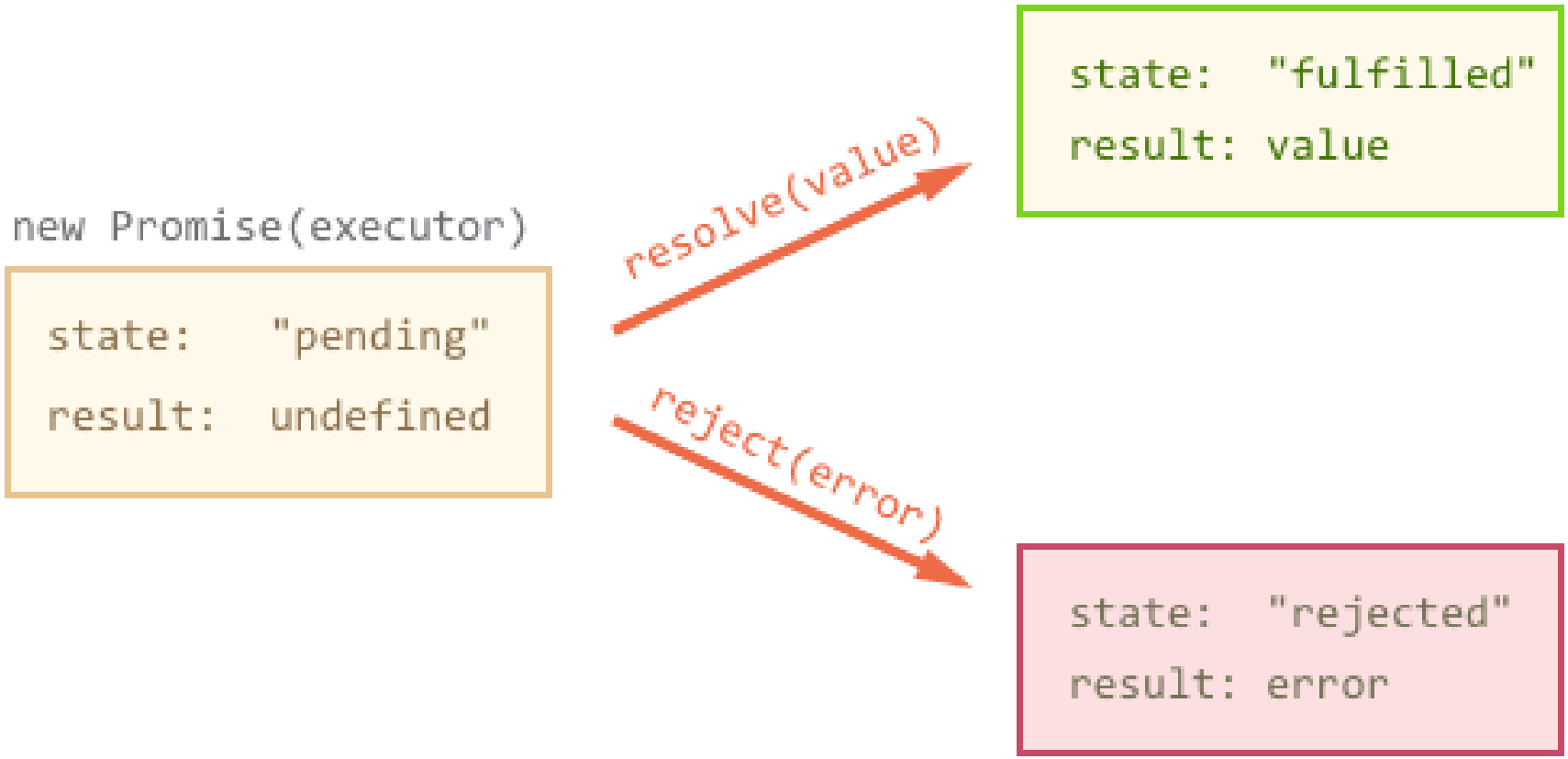
A function that

is passed with the arguments *resolve* and *reject*.
is executed immediately by the Promise implementation,
passing *resolve* and *reject* functions.

Resolve/Reject

The *resolve* and reject *functions*, when called, resolve or reject the promise, respectively.

Promise object



Promise Tips

There can be only one result or an error. The executor should call only one *resolve* or *reject*.

The promise state change is final.

resolve/reject with more than one argument - only the first argument is used, the next ones are ignored.

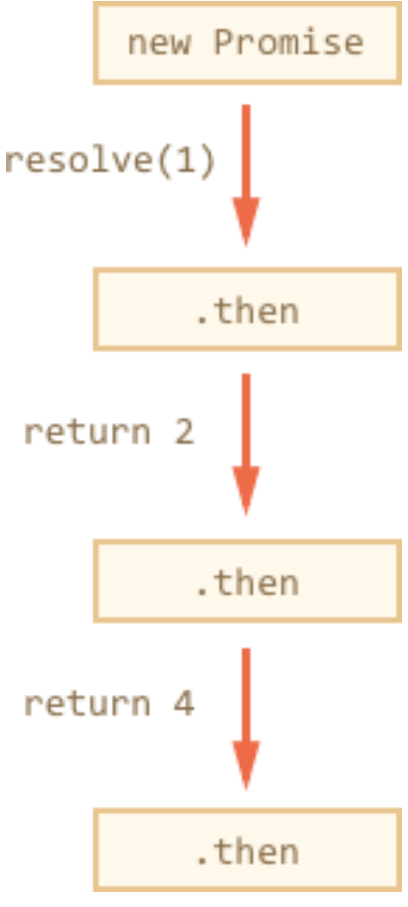
Use *Error* objects in *reject* (or inherit from them).

Promise Tips

Properties state and result of a promise are internal. We can’t directly access them, but we can use methods *.then/catch* for that.

.then and .catch

Promises chaining



Promises chaining