**História e Conceitos**

**JavaScript**: Lançado em setembro de 1995. **Moca**, **livescript** e por fim **JavaScript.**

**Criado por:** Brendan Eich

**ECMAScript:** “S6, S7”. É uma especificação de uma linguagem de programação que é padronizada pelo **ECMA262,** servindo como base para outras linguagens **script,** como **JavaScript**, **JScript** e **EctionScript.**

**TC39:** É o comitê responsável pela evolução JavaScript. Ele vem para manter e atualizar os padrões do **ECMAScript** que é no projeto **ECMA262**, ele identifica, desenvolve e mantem padrões para as bibliotecas que estende nas **fitures** do **ECMAScript** que é o **ECMA402.** Ele desenvolve **suifts** de testes para que as propostas podem ser testadas, esse projeto se chamada **ECMATR104.**

**Fluxo da Proposta:**

**- Stage 0: strawman:** Vai submeter um formulário com a ideia de evoluir o **ECMAScript**, deve ser feita por um membro ou contribuidor registrado no **TC39.**

**Obrigatorio:** Documento revisado no encontro da **TC39** e depois adicionados as pages da proposta como stage 0.

**- Stage 1: proposal:** Surge a proposta formal da funcionalidade.

**Obrigatório:** Vai ter um **Champion** e ele precisa ser membro do **TC39.** Bem descritos os objetivos da proposta e exemplos.

**- Stage 2: draft:** Primeira versão dessa proposta que vai entrar na verificação.

**Obrigatório:** A proposta precisa ter uma descrição formal da sintaxe e semântica da funcionalidade, a descrição precisa ser o mais completa possível. Duas implementações são necessárias dessa proposta, sendo que uma delas pode rodar pelo BABEL.

**- Stage 3: canditate:** Feedback de implementação e de usuário super ID, especificação completa nesta etapa, assinatura do editor.

**Obrigatório:** Deve ter pelo menos duas especificações compatíveis com a implementação.

**- Stage 4: finished:** Pronta para ser incluída na especificação, mas ainda tem alguns pontos para progredir.

Obrigatório: Estar passando na suwit de testes, duas implementações de conformidade com especificação que passa nos testes, experiencia prática significativa na implementação, baseada no feedback anterior e o Editor de especificação do ECMAScript deve assinar o texto dessa esécificação.

**ES2018:** Última especificação que temos

- Operadores rest/spread

- Alteração assíncrona

- Promisse.prototype.finally()

**ES.Next:** Especificação que possui futuras implementações

**Linguagem interpretada:**

O JavaScript é uma linguagem interpretada, ou seja, o código é executado de cima para baixo e o resultado dessa execução do código é imediatamente retornado

**Linguagem de tipagem fraca e dinâmica:**

**Tipagem dinâmica:** não é necessária especificar o tipo da variável na sua declaração. Por exemplo: var número.

**Tipagem estática:** é necessário especificar o tipo da variável na sua declaração. Por exemplo: int número.

**Fraca:** Não a verificação em todas as operações no JavaScript, ou seja, é possível utilizar o operador “+” com uma string e um número sem que ocorra um erro.

**Forte:** retorna um erro caso utilize o operador “+” com uma string e um inteiro.

**TypeScript:**

Super7 do JavaScript. Além de adicionar tipos de desenvolvimento para o JavaScript, ele adiciona funcionalidades que o JavaScript não tem por padrão.

**Flow:**

Semelhante ao TypeScript com relação a checagem de tipos, mas ele não é um Super7 da linguagem. Como se fosse uma versão ais simplificada do TypeScript.

**Funções de primeira classe e ordem maior:**

**Função de primeira classe:** Significa que a função pode ser atribuída a uma variável, estrutura de dados (object ou array) e ela pode ser passada por argumento.

**Closure**:

A capacidade de uma função lembrar do ambiente em que ela foi criada

**Currying: para cada parâmetro será criado uma função**

É a técnica de transformar uma função com diversos parâmetros em apenas uma função que recebe apenas um parâmetro e para cada parâmetro a gente vai retornando uma função.

**Hoisting: significa levantar ou suspender algo**

**Pode ser separado em dois tipos: “variáveis” e “função”**

**Variável:** só eleva a criação da variável e não da função, se a variável for criada depois da chamada, você recebera um undefined, significa que “a variável foi criada, mas não foi encontrada”.

**Função:** ela é elevada ao topo como um todo, pode ser criada/declarada e atribuída depois da chamada da função e mesmo assim ela será executada normalmente.

**Imutabilidade: depois da variável ser criada “P1” não poderá mudar seus valores.**

Deverá criar outra variável “**P2**” para manipular os valores, sem alterar definidamente os valores da “**P1**”.

**Tipos e Variáveis**

**Obs: Antigamente o JavaScript só possuía dois tipos de escopo sendo eles: “Escopo global” e “Escopo de Função”. O “Escopo de bloco” veio junto com o “let” e “const”.**

**Var: Não respeita(entende) o “escopo de bloco”. Só vai aceitar “Escopo de função” e “Escopo global”.**

**Let: Respeita o “Escopo de bloco”**

**Const: Respeita o “Escopo de bloco”**

**Const:** Se for uma **const** eu não consigo alterar a propriedade, só é possível ser alterada se for um **objeto** ou **array**, mas eu não consigo mudar para onde ele está apontando.

**{ }:** Escopo de bloco

**var escopo:** Escopo global

**Function test( ){**

**}:** Escopo de função

**Tipos, variáveis, operadores, condicionais e repetição**

**O javascript possui 6 tipos permitidos de variáveis:**

String, number, boolean, null, undefined, symbol, object, function, array.

**Undefined:** A variável existe mais ainda não foi setada (Não foi atribuído um valor para ela).

**Symbol:** Nos permite criar valores únicos.

**Operadores**

**Aritméticos:**

**Atribuição:  
Comparação:**

**Condicional:  
Lógicos:**

**Spread:**

**Estruturas condicionais e repetição**

**If, else e else if:  
switch:**

**REPETIÇÃO:**

**For:**

**While:  
do...while:**

**For...in:**

**Continue:  
break:**

**--------------------------------------------------------------------------------------------------------------------------------------------------------------**

**Orientação a objetos**

**Herança:**

**-** Baseada em protótipos

**- prototype:** uma variável que armazena as definições do objeto.

**- \_proto\_:** aponta para o prototype que foi criado

**- contructor:** Baseado no constructor e criado o prototype e acessado pelo \_\_proto\_\_