

**DISCIPLINA: PARADIGMAS DE PROGRAMAÇÃO**

**RELATÓRIO JAVASCRIPT**

**(2022)**

**ALUNOS:**

* **Jefferson Gomes de Almeida - 192080016;**
* **Rafaela Candido Carneiro Fernandes - 192080393;**
* **Ângelo Gabriel Paz da Silva - 192080075;**
* **Eduarda Fernanda Sousa Cunha - 192080296**
* **Kennedy Johnson de Sousa Dantas - 192080156**

**Prof(ª). Janderson Jason Barbosa Aguiar**

  CAMPUS CAMPINA GRANDE, março de 2022

1. **Considerações iniciais**

JavaScript é uma linguagem de alto nível, ou seja, sua sintaxe se aproxima mais da nossa linguagem humana, distanciando-se da linguagem de máquina. Possuindo um nível mais alto de abstração, seu código é mais fácil de ser interpretado, pois não são necessárias instruções para, por exemplo, registradores e saber comandos de processador.

Dada tal flexibilidade de interpretação e escrita, nada mais justo que esta seja uma linguagem multiparadigma. Dito isso, um paradigma é uma forma de classificar uma linguagem de programação tomando como base seu funcionamento e estruturação, alguns exemplos bem conhecidos são os paradigmas de orientação a objetos e imperativo. Sendo multiparadigma, a linguagem JavaScript pode solucionar um problema de formas elegantes e simples, mas isso não é de todo vantajoso visto que normalmente, linguagens multiparadigmas são fracamente tipadas. A tipagem de uma linguagem diz respeito ao fato de suas variáveis possuírem tipos bem definidos ou não, ou seja, uma tipagem fraca dificulta fatores como uma futura manutenção.

Nada mais justo para uma linguagem de alto nível e multiparadigma do que nascer em uma época de revolução digital para resolver os problemas da época. Os browsers na década de 90 ainda eram estáticos, ou seja com layouts específicos que não muda em resposta às ações de um usuário, então Brendan Eich a serviço da empresa Netscape junto com outros desenvolvedores criaram o navegador mais popular da época, o Netscape Navigator. Então projetada para ser uma linguagem com a sintaxe baseada em Java com valores primitivos e objetos, e sofrendo influência de outras linguagens como Scheme, Self, Perl e Python o JavaScript evoluiu para uma das linguagens mais utilizadas no mercado principalmente para desenvolvimento web, estando em 7° lugar no TIOBE, 5° em IEEE Spectrum, tendo em vista que, no desenvolvimento WEB, a linguagem pode ser utilizada no frontend, backend e até mesmo na comunicação de banco de dados.

O JavaScript é uma linguagem interpretada ou seja o código fonte não é traduzido diretamente para uma máquina de destino, em vez disso um programa(Interpretador), lê e interpreta o código.

1. **Conceitos** 
   1. **Nomes**

Um identificador ou nome, é uma cadeia de caracteres utilizada para identificar alguma entidade em um programa, sendo fulcrais no desenvolvimento de códigos em linguagens de programação, elas possuem particularidades dependendo a qual linguagem pertence.

No caso de JavaScript podemos destacar fatores, como por exemplo os identificadores serem “case sensitive” ou seja variáveis são diferenciadas por letras maiúsculas e minúsculas. Outra característica seria que caracteres especiais não são permitidos na nomeação de variáveis visto que alguns deles são utilizados em operações.

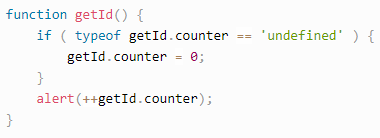
O tamanho máximo de um nome nessa linguagem se dá por um comprimento de - 1 elementos, e para o navegador Firefox tem um máximo de - 2 elementos.

A maioria das palavras especiais são reservadas como por exemplo, for,break, case class e const, mas há algumas palavras chaves que dependem de um certo contexto como por exemplo um modo restrito que engloba palavras como: implements, private, static e public.

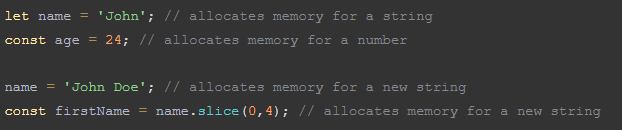
* 1. **Vinculações (Bindings)**

Javascript é uma linguagem de programação interpretada, de tipagem fraca e dinâmica. Um código em Javascript pode estar embutido em documentos HTML e ser interpretado pelo navegador. A validação de dados de entrada em formulários e a criação de documentos HTML dinâmicos correspondem aos principais usos de Javascript na programação web.

Funções em JavaScript são objetos e podem ter propriedades. Portanto, podemos criar variáveis ​​estáticas declarando as propriedades da função. Eles mantêm seus valores como variáveis ​​globais e não podem ser modificados fora das funções, tornando-os muito mais organizados do que as variáveis ​​globais.



O heap é um espaço diferente para armazenar dados onde o JavaScript armazena objetos e funções. Ao contrário da pilha, o mecanismo não aloca uma quantidade fixa de memória para esses objetos. Em vez disso, mais espaço será alocado conforme necessário. A alocação de memória dessa maneira também é chamada de alocação dinâmica de memória .



Os valores primitivos são imutáveis, o que significa que, em vez de alterar o valor original, o JavaScript cria um novo.

Nas Variáveis dinâmicas de pilha vinculações são criadas quando suas sentenças de declaração são efetuadas, mas o tipo é estáticamente vinculado.

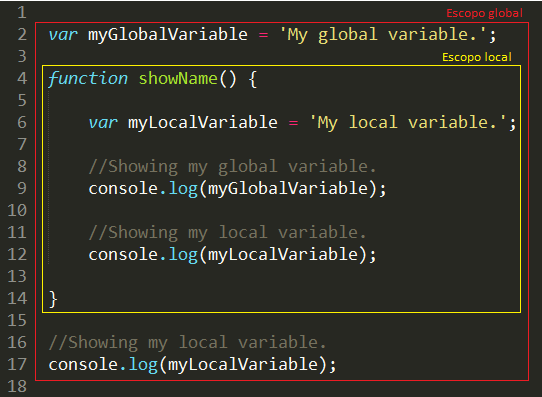
Uma variável dinâmica de pilha implícita é como uma variável dinâmica de pilha explícita, mas é criada sem um operador de alocação explícito. *Sebesta* dá como exemplo a instrução JavaScript,onde a variável que armazena o array de dois elementos é uma variável dinâmica de pilha implícita.



Variáveis dinâmicas explícitas podem ser referenciadas por ponteiros ou variáveis de referência. Javascript não tem ponteiro, mas tem [indireções](https://pt.stackoverflow.com/q/181032/101) e valores acessados por referência.

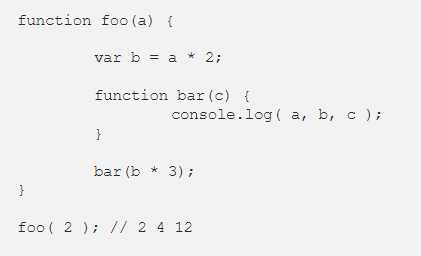
* 1. **Escopo**

No JavaScript existem somente dois tipos de escopos, que são eles, escopo global e local. O código a seguir ilustra basicamente como funcionam esses dois escopos.



Uma variável global é definida quando declaramos uma variável fora de qualquer função, assim ela torna acessível a qualquer parte da nossa aplicação ou site, podendo ser lida e alterada, já uma variável local é declarada dentro de uma função, de tal maneira a qual ela somente estará acessível dentro dessa função.

A linguagem Javascript possui escopo estático, ou seja, é baseado em onde as variáveis ​​e blocos de escopo são criados, por você, no momento da gravação e, portanto, é definido no momento em que o escopo processa seu código. Em uma linguagem com escopo estático, o escopo de um identificador é fixado em alguma região no código-fonte que contém a declaração do identificador. Isso significa que um identificador só é acessível nesta região.

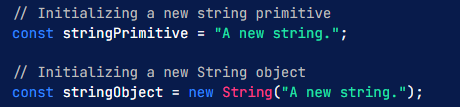


* 1. **Tipos de dados**

Segundo a última versão ECMAScript, a definição dos tipos de dados da linguagem são: [Boolean](https://developer.mozilla.org/pt-BR/docs/Glossary/Boolean), [Null](https://developer.mozilla.org/pt-BR/docs/Glossary/Null), [Undefined](https://developer.mozilla.org/pt-BR/docs/Glossary/undefined), [Number](https://developer.mozilla.org/pt-BR/docs/Glossary/Number), [Symbol](https://developer.mozilla.org/pt-BR/docs/Glossary/BigInt), [String](https://developer.mozilla.org/pt-BR/docs/Glossary/String), [BigInt](https://developer.mozilla.org/pt-BR/docs/Glossary/BigInt) e [Object](https://developer.mozilla.org/pt-BR/docs/Glossary/Object), sendo os 6 primeiros seus tipos primitivos. Já em relação aos tipos compostos, podemos citar objetos, arrays, datas, sets, entre outros.

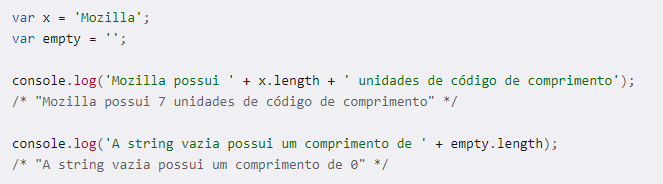
Ao contrário de algumas linguagens de programação que é possível usar o enum para definir tipos de dados enumeráveis, em javascript não é possível usar essa ferramenta, todavia, existem outras técnicas para fazer esse trabalho, como enumerate. Já para subfaixas é possível definir novos tipos que vão vir a partir de classes/objetos que facilitem a legibilidade e confiabilidade do código.

O Javascript trata de forma diferente um primitivo de string, que é um tipo de dados imutável, e o objeto String. O JavaScript é capaz de acessar e usar as propriedades e métodos integrados do Wrapper de objeto String, sem de fato alterar o primitivo de string que você criou em um objeto.

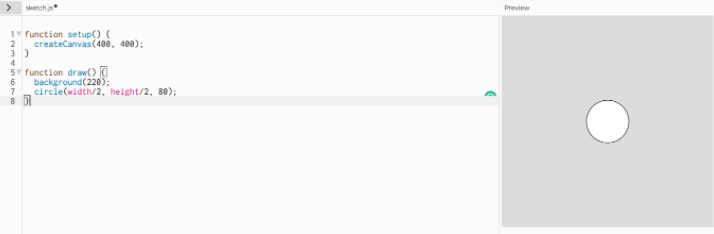


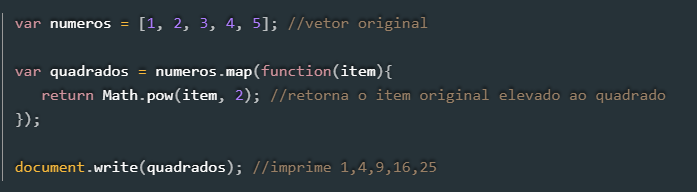
A ECMASCript 2016 estabeleceu um comprimento máximo de 2^53 - 1 elementos (dinâmico). Anteriormente, nenhum comprimento máximo havia sido especificado (estático). No Firefox, as strings têm um comprimento (length) máximo de 2\*\*30 - 2 (~ 1 GB). Para uma string vazia, length é 0.

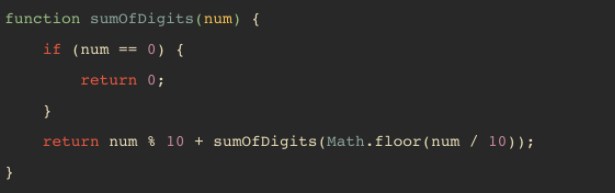
A propriedade estática String.length retorna o valor 1.



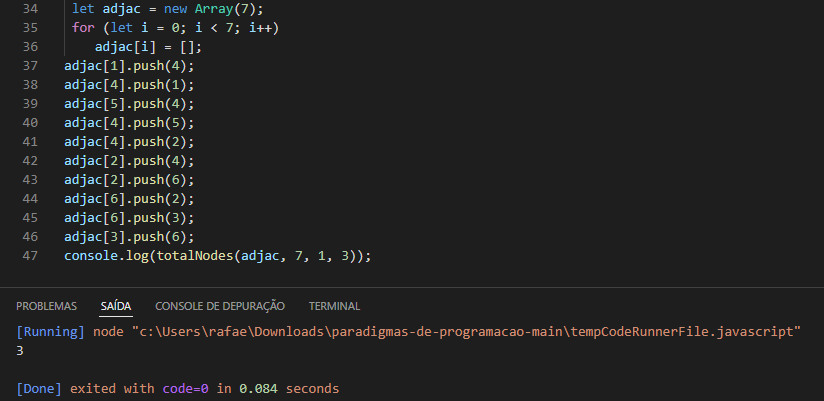
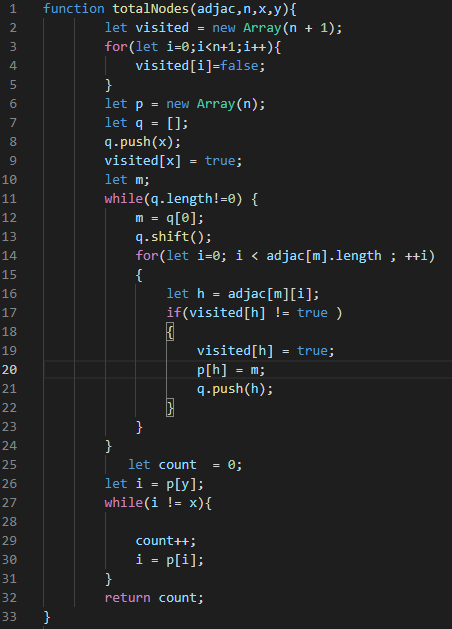
A linguagem é bastante abrangente e possui suporte para plano cartesiano, mapeamento e tipos recursivos.





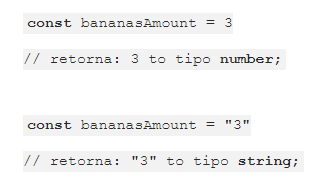


União disjunta em JavaScript:

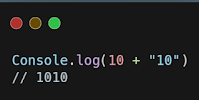


* 1. **Checagem de tipos**

O JavaScript, diferentemente do Java, possui sua tipagem dinâmica, ou seja, o próprio JS consegue saber que tipo é o valor atribuído à variável e fazer o casting automático.



O JavaScript assim como Ruby, possuem uma tipagem dinâmica, ou seja, não precisamos definir o tipo da variável antes de sua declaração, mas, Ruby é fortemente tipada e Javascript fracamente tipada. Por exemplo: na operação entre um número e uma string, em Javascript, o número inteiro será convertido em uma string e o resultado impresso será a string "1010".

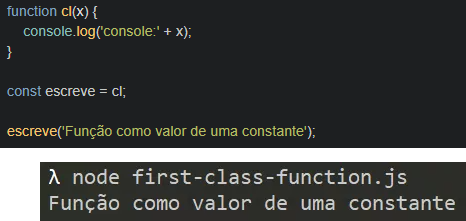


* 1. **Equivalência de tipos**

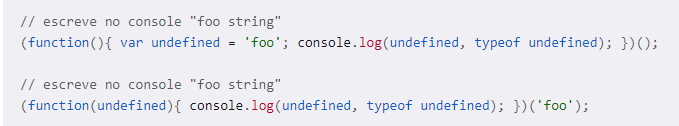
O Javascript possui equivalência nominal, Os seus tipos numéricos podem ser declarados com *var*, *let*, *const* e sem nenhum tipo antecedente. Geralmente, declaramos as variáveis com *const,* caso o valor da variável possa mudar, usamos *let*, caso queira atualizar e declarar novamente a variável dentro do seu escopo, usamos *var*.

* 1. **Completude de tipos**

Um dos recursos da programação funcional presentes no JavaScript são as Funções de Primeira Classe, isso é possível por que em JavaScript funções também são tratadas como Objetos de Primeira Classe, isso significa que podemos tratar funções como valores, e armazená-las em variáveis e constantes, passar funções como parâmetros em outras funções e até ter funções como retorno de outras funções.



Um exemplo de valor de segunda classe na linguagem seria uma variável que não teve um valor atribuído é do tipo undefined. Um método ou sentença também retorna undefined se a variável que está sendo avaliada não tem um valor atribuído. Uma função retorna undefined se um valor não for retornado.



Em JavaScript, alguns identificadores são palavras especiais que são reservadas e não podem ser usados ​​como variáveis ​​ou nomes de função. Alguns exemplos de palavras reservadas da linguagem, lembrando que não pode usar essas palavras reservadas como variáveis, rótulos ou nomes de função:



Esta linguagem viola o Princípio de Completude de tipo.

* 1. **Expressões**

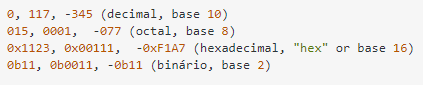
Literais em Javascript são valores fixados, Os literais em Javascript são: Array literal, Literais lógicos, Literais de ponto flutuante, Inteiros literais, Objeto literal,String literal.Um Array literal é uma lista de zero ou mais expressões, onde cada uma delas representam um elemento do array, inseridas entre colchetes ([]):



O tipo lógico tem dois valores literais: true e false.



Inteiros literais: podem ser expressos em decimal (base 10), hexadecimal (base 16), octal (base 8) e binário (base 2). Decimal inteiro literal consiste em uma sequência de dígitos sem um 0 a esquerda. 0 a esquerda em um inteiro literal indica que ele está em formato octal. Octal pode incluir somente os dígitos 0-7. 0x (ou 0X) indica um hexadecimal. Inteiros hexadecimais podem incluir dígitos (0-9) e as letras a-f e A-F. 0b (ou 0B) indica um binário. Inteiros binários podem incluir apenas os dígitos 0 e 1.



Literais de ponto flutuante: Possuem a seguinte sintaxe:[(+|-)][digitos][.digitos][(E|e)[(+|-)] digitos].



Objeto literal: é uma lista de zero ou mais pares de nomes de propriedades e valores associados de um objeto, colocado entre chaves ({}).



Agregados ou construtores são expressões que constroem um valor composto a partir de seus valores de componentes. Um exemplo disso em Javascript são os arrays.

A definição de uma função não a executa. Definir a função é simplesmente nomear a função e especificar o que fazer quando a função é chamada. Chamar a função executa realmente as ações especificadas com os parâmetros indicados. Por exemplo, se você definir a função square, você pode chamá-la do seguinte modo:



O operador condicional é o único operador JavaScript que utiliza três operandos. O operador pode ter um de dois valores baseados em uma condição. A sintaxe é: condição ? valor1 : valor2

Se a condição for verdadeira, o operador terá o valor de valor1. Caso contrário, terá o valor de valor2. Você pode utilizar o operador condicional em qualquer lugar onde utilizaria um operador padrão. Por exemplo, var status = (idade >= 18) ? "adulto" : "menor de idade";

Esta declaração atribui o valor "adulto" à variável status caso a idade seja dezoito ou mais. Caso contrário, atribui o valor "menor de idade".

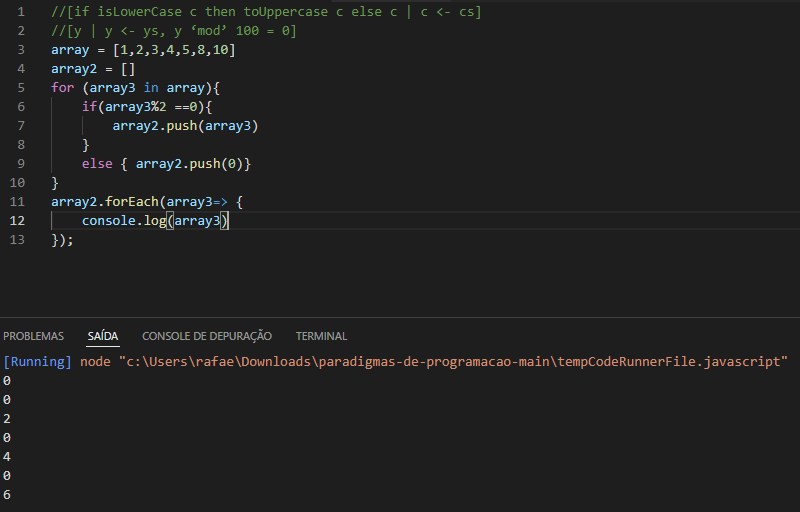
Você pode criar uma constante apenas de leitura por meio da palavra-chave const. A sintaxe de um identificador de uma constante é semelhante ao identificador de uma variável: deve começar com uma letra, sublinhado ou cifrão e pode conter caracteres alfabético, numérico ou sublinhado.



Uma constante não pode alterar seu valor por meio de uma atribuição ou ser declarada novamente enquanto o script está em execução. Deve ser inicializada com um valor.

As regras de escopo para as constantes são as mesmas para as variáveis let de escopo de bloco. Se a palavra-chave const for omitida, presume-se que o identificador representa uma variável.

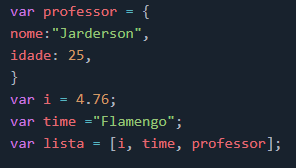
As expressões iterativas executam uma computação sobre uma série de valores resultando em um valor. Exemplo de compreensões de listas:



* 1. **Sistemas de tipos**

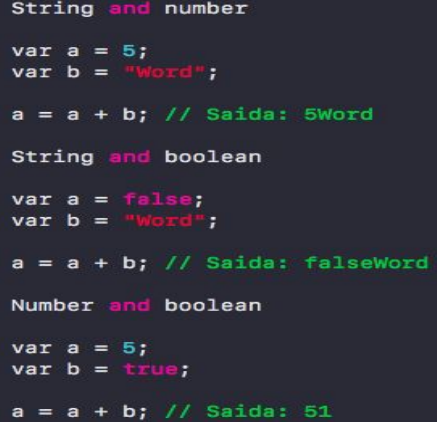
Não há sobrecarga em JavaScript, pois se duas funções forem definidas com o mesmo nome, a última função definida substituirá a função anterior. Por esse motivo, quando é preciso fazer uma sobrecarga, é preciso fazer manualmente.

O polimorfismo paramétrico se expressa através de declarações de tipos. O identificador ‘var’ declara desde primitivos até objetos e uma única variável pode abrigar diferentes tipos.



O Polimorfismo por Inclusão possui relacionamento de hierarquias entre subtipos e supertipos.

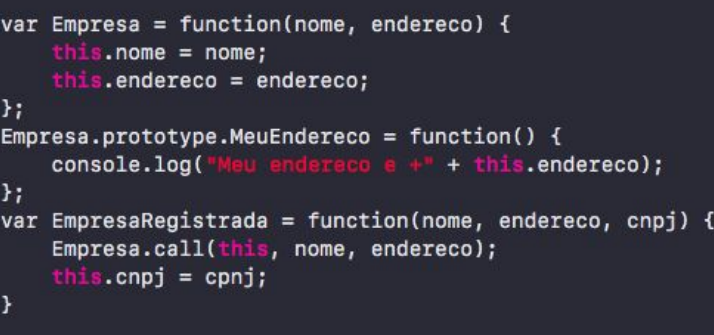
Em Javascript realiza a conversão implícita. Como números são tipo Number (implicitamente float) , conversões ocorrem entre outros tipos:



As funções principais para coerção implícita:

* Boolean():
* Number();
* String();
* Object();

O Polimorfismo clássico da Orientação a Objetos em Javascript está relacionado a hierarquias entre subtipos e supertipos e ao conceito de heranças.



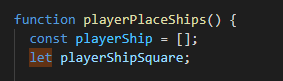
* 1. **Armazenamento**

**Variáveis:**

Variáveis ​​são contêineres para armazenar dados (armazenando valores de dados). Em javascript, existem quatro tipos de variáveis: var, let, const ou nothing.



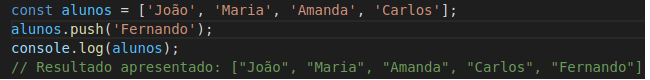
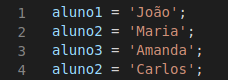
As variáveis temporárias são aquelas que só existem em determinada parte do código. Como podemos ver na figura do código do jogo abaixo, a variável let playerShipSquare, só existe nessa função.



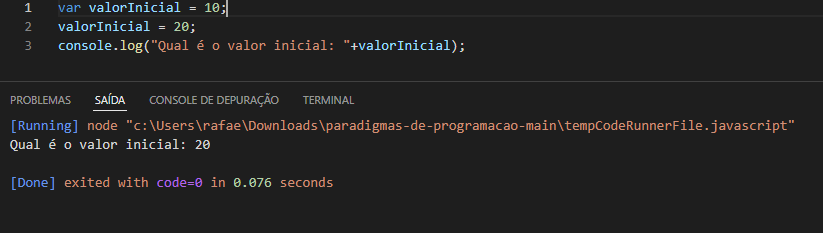
As variáveis persistentes são aquelas que existem independente de qualquer outra variável, por exemplo:

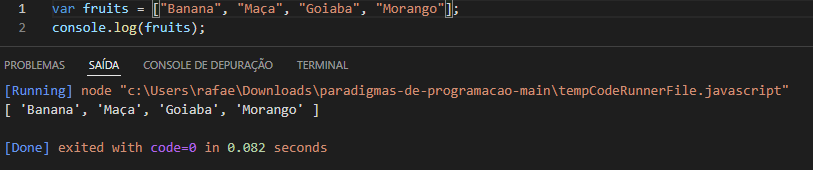


É possível definir variáveis que guardam mais de um valor de um mesmo tipo, essas variáveis são conhecidas como variáveis compostas, variáveis subscritas, variáveis indexáveis ou arranjos (array). O entendimento das variáveis compostas desta linguagem pode nos dar uma possibilidade maior de resolução de problemas dentro da programação. Temos diversos outros métodos que nos auxiliam a resolver problemas mais complexos.



Uma variável tem uma atualização quando inicialmente definimos ela, e depois mudamos o valor atribuido a mesma, por exemplo:



Um Array é uma variável especial, que pode conter mais de um valor por vez. Em javascript um array é usado para armazenar vários valores em uma única variável. 

O array dinâmico pode escolher valores dinâmicos ou nomes de variáveis ​​e nomes de objetos e escolher editar o nome da variável no futuro sem acessar a matriz.

Em Javascript, os valores armazenáveis são chamados de storables, todos os valores armazenados são do tipo primitivo.

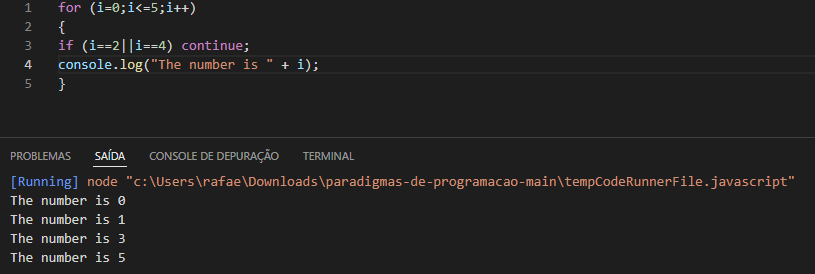
Quando se trata de estrutura de atualização o JavaScript pode ter o tipo total como seletiva, em relação a total pode-se observar a utilização de arrays como exemplos que modificam os dados pelos index. Sobre a seletiva nota-se um exemplo com objetos que podem ter várias chaves e podem ser alterados apenas o valor de uma chave.

Devido a ausência de ponteiros na linguagem, as semânticas de cópias acontecem apenas em valores primitivos.

**Comandos:**

Em javascript os comandos são sensíveis ao tipo de letra (maiúsculas e minúsculas) em sua sintaxe.

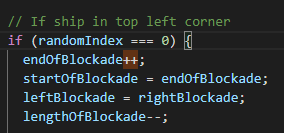
O comando skip significa continue e só pode ser usado em loop:



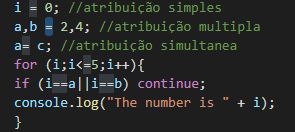
No exemplo acima ele vai pegar todos os números ele vai ignorar o 2 e o 4.

O comando de atribuição utiliza o ‘=’, para atribuir um valor da direita para esquerda, além de ter o operador de incremento ‘++’ e decremento ‘--’ .

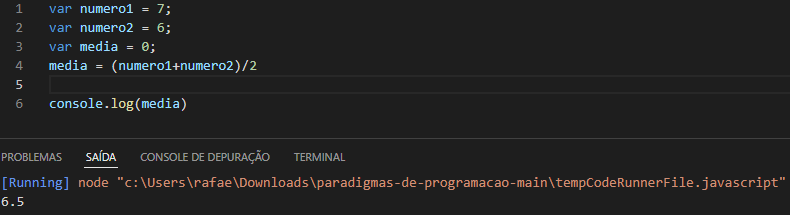
Exemplo do código do jogo:



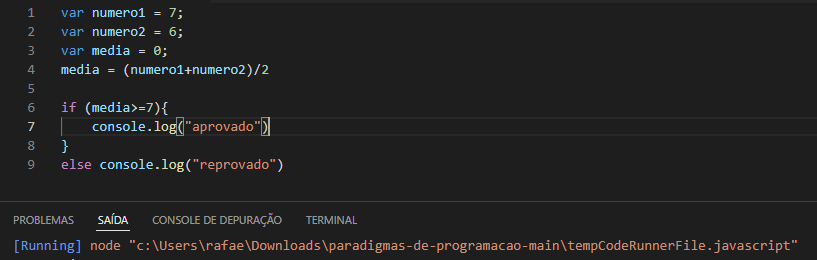
Atribuição múltipla é a possibilidade de retornar múltiplos e tipos diferentes e simultânea é uma atribuição paralela:



Um comando sequencial em javascript é uma sequência que vai ser executada um após o outros, ou seja, as variáveis são executadas sempre da direita para esquerda, de cima para baixo.

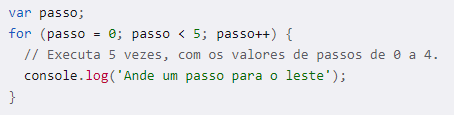


Comandos colaterais é quando se pode declarar mais de uma variável de maneira não determinística, ou seja, criar variáveis de forma sequencial. Em javascript a ordem de execução é irrelevante, visto que a execução os comandos podem ser executados em qualquer ordem.

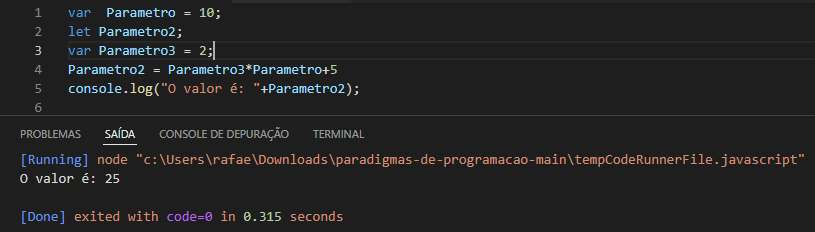
Os comandos condicionais fazem a execução de uma tarefa depender da verdade ou não de uma condição, ou quando uma condição é verdadeira. Em javascript temos o if e else para operador ternário. A estrutura if é usada quando se deseja verificar se determinada expressão é verdadeira ou não, e executar comandos específicos para cada caso, caso essa estrutura seja falsa o bloco de comandos que segue o else será executado.

A linguagem possui uma chamada de procedimento, conhecida como uma chamada de uma função com parâmetro vazio ou com variáveis, o objetivo da função varia de acordo com o desenvolvedor.

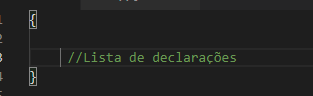
Chamada de procedimento, comumente conhecida como laços, oferecem um jeito fácil e rápido de executar uma ação repetidas vezes. Existem várias formas diferentes de laços, mas eles essencialmente fazem a mesma coisa: repetir uma ação múltiplas vezes (inclusive você poderá repetir 0 vezes). Os vários mecanismos diferentes de laços oferecem diferentes formas de determinar quando este irá começar ou terminar. Há várias situações em que é mais fácil resolver um problema utilizando um determinado tipo de laço do que outros. Os possíveis laços de repetição em JavaScript: for\_statement, do...while\_statement, while\_statement, label\_statement, break\_statement, continue\_statement, for...in\_statement, for...of\_statement.

****

Em javascript as expressões com efeitos colaterais são feitas quando atribuímos uma expressão a uma variável e atribuímos um valor a ela.

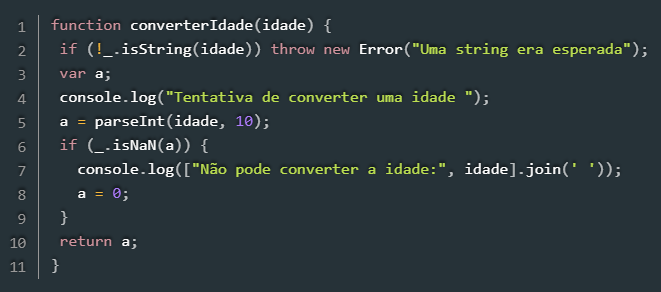


A declaração em bloco ({...}) permite o uso múltiplas declarações onde o JavaScript espera apenas uma declaração. Combinar declarações em blocos é uma prática comum em JavaScript.



* 1. **Abstrações**

A linguagem também possui métodos de abstração, que são funções que ocultam os detalhes de implementação das mesmas. Por exemplo, quando precisamos implementar algum tipo de mecanismo de gerenciamento e exibição de erros de report e/ou avisos sistêmicos.



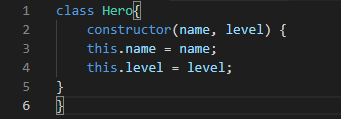
O Javascript também possui abstração por array, que pode representar uma estrutura de dados abstrata, sendo ela a pilha  *last-in-first-out*. Além disso, também há a possibilidade de utilização de pacotes que são comumente chamados de módulos.

Em javascript os objetos também são variáveis e todos os argumentos de uma função são passados por valor (a linguagem copia os valores das variáveis passadas). Mas os objetos podem conter muitos valores.



A passagem de parâmetro por referência em javascript não é feita diretamente, todavia, pode usar abstração para passar uma referência. Exemplo: um array, que a função altere o valor do parâmetro anterior

Uma maneira de definir uma classe é usando uma declaração de classe. Para declarar uma classe, você deve usar a palavra-chave ‘class’ seguida pelo nome da classe.



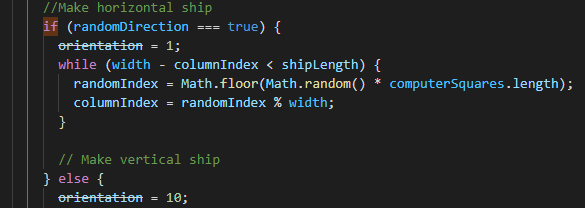
Esta linguagem copia os valores das variáveis passadas como parâmetro para qualquer objeto(unidade genérica), dito isso, os argumentos de uma função serão sempre passados por valor.

* 1. **Sequenciadores**

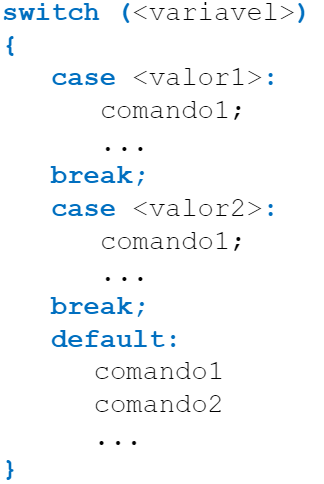
Sequenciador é a construção que modifica o fluxo normal de controle. Ele transfere o controle para algum outro ponto no programa.

**If, If else** e **SWITCH Case** são os comandos de desvio condicional em Javascript. Exemplos (do código do jogo) de desvio condicional:

If e Else:

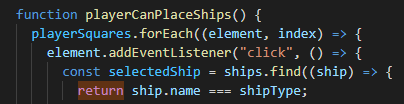
****

Esse trecho está verificando a posição do navio, em horizontal e vertical:

****

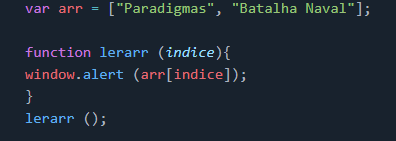
Em Javascript escapes podem ser return, break.

Exemplo de return do código do jogo:

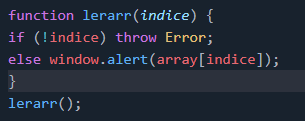


O tratamento de exceções em JavaScript é semelhante ao fornecido pela linguagem Java. Temos os comandos throw, try, catch e finally. Com eles é possível desenvolver uma aplicação em JavaScript capaz de tratar possíveis erros em tempo de execução, melhorando significativamente sua robustez.

Exemplo:



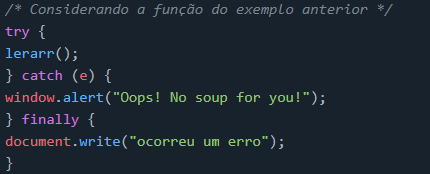
Essa chamada vai dar um erro, pois como não passamos o parâmetro índice ele pode conter lixo ou não definido. Vamos usar o comando Throw para a função gerar uma exceção:



Quando executamos, aparece a mensagem:

"uncaught exception: function Error() { [native code] }

Para resolver isso, usamos o try catch:



Com isso, se houver uma exceção ela será lançada.

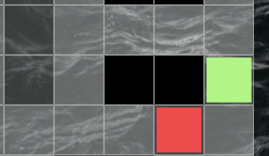
1. **Considerações finais**

Com relação às dificuldades encontradas sobre a produção do documento, foram principalmente dificuldades com relação a pesquisa sobre os assuntos da maioria das perguntas na internet/livros ou encontrar/fazer exemplificações (por mais que achávamos que seria algo fácil de se encontrar devido a linguagem ser bastante conhecida e bastante usada atualmente).

Apesar de já termos contato com javascript em outras cadeiras, aprendemos que JavaScript evoluiu para uma das linguagens mais utilizadas no mercado principalmente para desenvolvimento web, estando em 7° lugar no TIOBE, 5° no IEEE Spectrum, além de ser multiparadigmas ela também suporta estilos de programação funcionais e imperativos (incluindo orientado a objetos e prototype-based).

Para finalizar, essa parte de considerações do relatórios vimos que JavaScript assim como Ruby, possuem uma tipagem dinâmica, ou seja, não precisamos definir o tipo da variável antes de sua declaração, mas, Ruby é fortemente tipada e Javascript fracamente tipada.

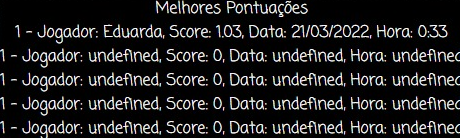
Com relação ao jogo, encontramos dificuldades na construção de lógica de jogadas do computador quando ele acerta um navio. Inicialmente, quando ele acertava um navio, logo depois ele fazia uma jogada aleatória. Pode ser visto isso no exemplo abaixo:



Mas apesar da dificuldade, aprendemos a implementar uma IA para jogadas do computador, de maneira que quando ele acerte um navio ele tente acertar ao lado, ou embaixo/cima.



Além disso, tivemos dificuldade em salvar o ranking de ganhadores do jogo. Isso pode ser visto na imagem abaixo:



Quando se trata de legibilidade, javascript é de fácil leitura e compreensão, com uma sintaxe limpa que ajuda a leitura e o entendimento. Para quem já tem experiência em outra linguagem de programação, a linguagem possui facilidade de escrita. É uma linguagem que disponibiliza recursos de confiabilidade como: Tratamento de exceções (throw, try, catch e finally), além de possuir vários exemplos de palavras reservadas da linguagem, ser multi-paradigma e dinâmica. Levando em consideração todos fatores citados acima, pode-se concluir que JavaScript possui um bom custo.



1. **Referências**

WATT, David A. . Programming Language Design Concepts. John Wiley

Sons, Ltd, 2004.

AGÊNCIA MACAN. Disponível em: <https://www.agenciamacan.com.br/blog/site-dinamico-vs-estatico-qual-a-diferenca-entre-eles>. Acesso em: 21 mar. 2022.

ROSA, Daniel. **Linguagens de programação interpretadas x compiladas**: qual é a diferença? 20 dez. 2021. Disponível em: <http://www.freecodecamp.org/portuguese/news/linguagens-de-programacao-interpretadas-x-compiladas-qual-e-a-diferenca/>. Acesso em: 21 mar. 2022.

JAVASCRIPT Naming Conventions. Disponível em: <http://www.robinwieruch.de/javascript-naming-conventions/>. Acesso em: 21 mar. 2022.

JAVASCRIPT keywords and reserved words - JavaScript Institute. Disponível em: <https://www.javascriptinstitute.org/javascript-tutorial/javascript-keywords-and-reserved-words/>. Acesso em: 21 mar. 2022.

AKAR, Osman. **Javascript — Lexical and Dynamic Scoping?** 12 fev. 2019. Disponível em: <https://medium.com/@osmanakar_65575/javascript-lexical-and-dynamic-scoping-72c17e4476dd>. Acesso em: 21 mar. 2022.

ESTRUTURA de dados do Javascript - JavaScript | MDN. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Data\_structures. Acesso em: 21 mar. 2022.

STRING.LENGTH - JavaScript | MDN. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global\_Objects/String/length. Acesso em: 21 mar. 2022.

JAVASCRIPT this. Disponível em: <https://www.w3schools.com/js/js_this.asp>. Acesso em: 21 mar. 2022.

JAVASCRIPT keywords and reserved words - JavaScript Institute. Disponível em: <https://www.javascriptinstitute.org/javascript-tutorial/javascript-keywords-and-reserved-words/>. Acesso em: 21 mar. 2022.

SAMPAIO, Julio. **Programação funcional com JavaScript - DevMedia**. Disponível em: <https://www.devmedia.com.br/programacao-funcional-com-javascript/34377>. Acesso em: 21 mar. 2022.

JAVASCRIPT Palavras reservados. Disponível em: <https://www.w3bai.com/pt/js/js_reserved.html>. Acesso em: 21 mar. 2022.