JavaScript – Alura

Parte 1 – Primeiros Passos Com a Linguagem

1. **Aula 1 – Introdução a Linguagem:**
   1. Podemos executar o programa no Power shell digitando “node + nome programa”.
   2. Console.log: Printa o que você escrever em *“ aqui ”*.
   3. SHIFT+ALT+↓ copia e cola uma linha no vscode.
   4. Salvamos variáveis utilizando *“const nome da variável: ;”*.
2. **Aula 2 – Tipos de Variáveis:**
   1. Colocar o comando console é diferente de colocar o comando Console. Quando tem letra maiúscula ele dá erro, pois o JS é Case Sensitive.
      1. Nesse caso, podemos colocar variáveis com cases diferentes, mas com o mesmo nome que o JS irá entender como coisas diferentes, porém, isso é uma má prática, portanto não faça.
   2. Colocar o *const* e não colocar não tem diferença visual na execução do código a princípio. O JS vai continuar interpretando como variável.
   3. Seguir a declaração das variáveis é uma boa prática.
   4. *“let”* é um outro jeito de declarar variáveis.
   5. Seja consistente: se colocar “;” em uma coisa, coloca em tudo, se não, não coloque em nada. Se as variáveis começarem com letras minúsculas, faça para todos, caso contrário, nenhuma.
   6. Colocar um sinal de “+” quando se tem uma string e um valor, ou duas strings faz com que o JS apenas concatene as duas coisas, mas, se colocar dois números, será feita uma soma.
   7. Quando se faz a mesma coisa com outros sinais de operação ele irá fazer a conta matemática corretamente.
   8. Se a string não conter um número, ele irá tentar fazer a conta e retornará um NaN (not a number) ao invés de dar erro.
   9. Para converter uma string para número use o comando *parseInt/parseFloat(“string”);*, sendo elas: números inteiros e flutuantes/reais, respectivamente.
   10. Quando trabalhar com número flutuante, colocar “.” ao invés de “,”, caso contrário ele irá colocar um espaço vazio no lugar dela ao invés de considerar o número como um número real.
   11. Quando printar um texto como nome e sobrenome, podemos simplesmente colocar uma vírgula para que ele separe os itens e fique correto, ao invés de concatenar o nome com um espaço e depois o sobrenome, que dá mais trabalho. *console.log(nome, sobrenome);/console.log(nome + “ “ + sobrenome);*.
   12. Conseguimos colocar interpolação de strings nas versões mais novas do JS utilizando crase para escrever o texto dentro e colocando **${}** ao invés de fazer a concatenação.
   13. Podemos reescrever uma variável com a informação que ela já continha somada a informação de outra variável.
   14. A declaração de variável “*const*” determina que aquela variável é uma constante, ou seja, se é uma constante não pode mudar. Já a declaração “*let*”, é uma variável mesmo e pode sim ser alterada.
   15. Uma boa prática é manter uma variável constante e não ficar alterando o tempo inteiro.
   16. Sempre crie uma variável nova, é bem melhor.
   17. Sempre dê o nome certo para as coisas e seja o mais específico possível para referenciar bem suas variáveis e não ter que ficar procurando para lembrar do que se trata.
3. **Aula 3 – Trabalhando com Muitos Elementos:**
   1. Conseguimos declarar e atribuir valores à variáveis em momentos e linhas diferentes, não preciso fazer isso no mesmo momento sempre.
   2. Podemos comentar várias linhas de uma vez utilizando o atalho CTRL+K+C e tirar o comentário de várias linhas selecionados usando o atalho CTRL+K+U.
   3. Para criar listas utilizamos o comando *const lista = new Array(`elemento1`, `elemento2`, );*, dessa forma criando ela dentro da variável constante *lista*, que poderia ser qualquer outro nome.
   4. Podemos adicionar outros elementos nas listas dinamicamente utilizando o comando *variável.****push(`****elemento novo****`);***.
   5. A vantagem das listas é não ter que criar uma variável para cada item, o que levaria muito tempo tornando inviável fazer isso com muitos elementos.
   6. E o mesmo vale para printar eles na tela, uma vez que teria que reescrever todas as variáveis dentro do console.log(); mais uma vez.
   7. Do mesmo modo que podemos adicionar itens dinamicamente na lista, também podemos retirá-los utilizando o comando *variável****.splice (****posição do**item****,*** *quantidade de elementos****);***. A posição começa no 0, ou seja, a posição 0 é a do primeiro item.
   8. Podemos printar somente 1 elemento colocando entre “[]” o número da posição do item que queremos ver dá nossa lista.
   9. Não podemos criar variáveis com nomes reservados.
4. **Aula 4 – Condicionas e Fluxo de execução:**
   1. É uma boa prática sempre tirar os comentários de comandos inutilizados.
   2. A condição é determinada pelo comando *if(condição){bloco de código para execução}*.
   3. Caso a condição colocada acima seja falsa, podemos colocar um *else{bloco de código}* depois dele para que isso seja executado no lugar.
   4. Podemos colocar vários ifs ou elses encadeados/aninhados dentro de outros para fazer um sistema complexo de escolhas.
   5. Apertando ALT+SHIFT+F o vscode irá formatar automaticamente o código para você.
   6. Muitas condicionais no código pode acabar gerando erros e dificultar na hora de achar.
   7. Ter código duplicado não é uma boa prática.
   8. Podemos criar condições onde uma OU outra operação tem que ser verdadeira para que ele execute o bloco. Do mesmo modo, podemos criar para se uma E a outra seja verdadeira, utilizando as operações **||** e **&&**, respectivamente para indicar isso.
5. **Aula 5 – Iterando Sobre Coleções:**
   1. Posso jogar operações de um if para uma variável.
   2. Podemos colocar um comando break em uma condição para que a partir do momento que ela seja satisfeita, o programa quebre direto para o final, sem ter que ficar passando várias e várias vezes pelo programa até chegar ao fim do contador.
   3. Podemos usar o While para poder repetir interações quantas vezes quiser a partir de um contador.
   4. Também podemos criar laços de repetição usando o *for*. Para colocar ele, precisamos escrever de forma diferente sendo: *for(declara contador; contador; contador++){ bloco de código como no while }*.
   5. Pessoal usa a variável “i” para contador.
6. **Extra – Funções e Classes:**
   1. Classe é o nome dado para estruturas para criar objetos.
   2. Praticamente tudo em JS é um objeto.

**Parte 2 – Programando a Orientação a Objeto:**

1. **Aula 1 – Repetição de Código:**
   1. Quando temos que fazer algo que se repete muito, não é legal ficar repetindo códigos, portanto, uma saída é fazermos um “molde” para as informações que a gente queira, desse modo, se precisar acrescentar ou tirar uma informação de mil usuários, por exemplo, não será necessário alterar manualmente um por um.
   2. Nós chamamos esse molde de classe, que nada mais é que uma estrutura que será repetida/replicada.
   3. Para criar uma classe segue-se a seguinte estrutura: *class nomeDaClasse {variáveis que ela contém}*. Quando falamos de classes, não falamos variáveis, apesar de ser o mesmo conceito, nós chamamos de campo/atributos/propriedades.
   4. Depois da classe criada, quando quisermos utilizar ela para criar um novo cliente de um banco, por exemplo, colocamos o comando *new nomeDaClasse();*, porém, é necessário criar uma variável para ela, portanto, o código completo fica assim: *const variável = new nomeDaClasse();*.
   5. Para acessar um atributo da variável, que está dentro da classe, nós colocamos o nome da variável (criada acima) um “.” E o nome do atributo: *variável.nomeDoAtributo = conteúto;*.
   6. Isso é bom porque se quiser printar todas as informações de um cliente, basta colocar o nome dele e aparecerá tudo, ao invés de criar um log para cada variável ou ficar digitando variável por variável para cada um deles.
      1. Mas, se quiser printar uma propriedade específica, basta colocar o nome do cliente, um “.” E o nome do atributo na frente dentro do log.
2. **Aula 2 – Adicionando Comportamentos:**
   1. As classes também suportam comportamentos, ou seja, não é apenas um conjunto de dados armazenado.
   2. Os “()” representam que o que você está fazendo é uma operação/comportamento, portanto, sempre que for fazer uma operação, coloque-os.
   3. Quando nos referimos a uma conta corrente específica ou um objeto ao qual a classe é referente, podemos colocar apenas o comando *this.*, desse modo, o código saberá que você está se referindo a qualquer conta que você esteja mexendo.
   4. Após colocar uma operação em uma classe, quando vai fazer a utilização dela, basta colocar o valor que quiser dentro dos parênteses.
   5. Podemos colocar “#” para indicar que é uma propriedade/campo privado, não permitindo que ela seja alterada a não ser que passe pelas verificações colocadas dentro da classe, esse método ainda não está em vigor oficial.
   6. Oficialmente, colocamos um “\_” para indicar que aquele atributo é privado, mas nenhum atributo é realmente privado hoje em dia.
   7. Nós podemos fazer retornos (return) dentro de métodos para, no caso do banco, retornar o valor esperado para o cliente após um saque.
   8. Return sempre no final do método/função.
   9. Ao invés de ir colocando diversos ifs um dentro do outro até chegar no resultado esperado, podemos colocar um return e ir selecionando por métodos negativos o resultado que eu quero. Essa técnica se chama Early return, usada para melhorar a legibilidade do código. Nesses casos onde o comando do if fica somente em uma linha, não tem a necessidade de utilizar os {} do if, basta colocar a condição e o return na frente que já será o suficiente.
3. **Aula 3 – Modularizando o Código:**
   1. É uma boa prática ir separando classes por arquivos, pois o código vai ficando cada vez maior conforme seu desenvolvimento tendo mais chances de erros.
   2. Cada vez que criar um arquivo com a primeira letra maiúscula, representa que dentro dele possui uma classe. O mesmo vale para nomes de classes, primeira letra sempre maiúscula.
   3. Fazendo essa organização, a index acaba sendo somente a execução.
   4. Para fazer o link dos arquivos precisamos usar os *export* e o *import{classe} from “CaminhoDoArquivo.js”*, dessa forma fazendo essa conexão. Colocamos o export nos módulos antes das classes e o importe no começo do arquivo que está consumindo aquela calasse.
   5. Para que o nodejs execute o código com os módulos, precisamos colocar no terminal *npm init*, ir preenchendo de acordo com o que pede e no final ele irá te gerar um arquivo package.json. Dentro desse arquivo coloque um *“type”: “module”,* e ele irá funcionar normalmente.
   6. Conseguimos jogar classes como atributos de outras.
4. **Aula 4 – Acessando Atributos Privados:**
   1. O JS não te retorna o objeto quando vc tenta acessar ele, ele te retorna uma referência àquele objeto, ou seja, o caminho até aquela memória.
   2. Null é um valor definitivo para nulo, sem informação, indefinido oficialmente.
   3. A diferença do null para o indefinido é que o null é proposital, alguém quis que aquela informação fosse nula, já o indefinido quer dizer basicamente que a informação daquela variável foi esquecida de ser colocada, foi por acaso/sem querer.
   4. O JS tem maneiras específicas de dar acesso a atributos privados de maneira segura. Essa palavra de acesso é *set*.
      1. Diferente de métodos, quando se usa essa técnica, podemos atribuir valor com a elas com o ‘=’, ao invés de colocar ‘()’ igual o depositar() ou sacar().
      2. Apesar de se parecer com um método ele se chama acessor.
      3. Um segundo acessor é o *get*.
      4. O acessor sempre é público.
      5. Utilizando o acessor get, nós podemos colocar um return dentro dele que se refira ao atributo privado, mas com o nome público, desse modo, quando a pessoa quiser ver um saldo da conta, por exemplo, ela pode colocar um log do saldo, mas se tentar atribuir um valor à ele, será negado, pois aquela é uma variável pública que serve apenas para exibir o valor real da variável privada \_saldo. Ex na aula 4 do módulo 2 do curso de JS na formação front-end.
   5. Para condicionais também podemos colocar o valor (*instanceof* *Classe)*, desse modo, caso você esteja fazendo um sistema de proteção para clientes, ele só vai atribuir valor se a instancia for do typo cliente, caso contrário ele não permite.
5. **Aula 5 – Construtores e estáticos:**
   1. Usando construtores você pode fazer com que a função new Cliente(); na hr de atribuir um cliente consiga armazenar todos os dados dele, basta colocar dentro dos parênteses new Cliente(“nome”, cpf); uma vez que o construtor tenha sido construído da mesma maneira constructor(nome, cpf){this.nome = nome; this.\_cpf(pq agr é privado) = cpf;}. Dessa forma facilita o código e se alguém tentar alterar o cpf por cliente1.cpf = QualquerOutroValor, ele não irá conseguir, pois o cpf é um getter, e não a variável original, pois ela está privada e não deve ser mexida.
   2. Podemos utilizar assessores dentro de construtores, mas, uma vez que atribuídos, não podem ser mudados a partir de assessores, tem que ser o atributo original, que está privado.
   3. Quando se cria um elemento estático, você está criando algo que assim que for alterado, ele irá se alterar em todas as contas, mesmo que elas sejam coisas separadas e individuais. A palavra chave para criar algo estático é a *static*.
6. **Extra: JS Linter:**
   1. Usamos o linter para ter padrões no projeto.
   2. Para iniciar precisamos colocar npx eslint –init, no terminal e geralmente escolhemos a terceira opção, onde ele vai procurar problemas e erros no código.
   3. Baixar extenção eslint.
   4. Apertar CTRL+. Ele dá quick fix nos erros do projeto.

**Parte 3 – Interfaces e Herança em Orientação a Objetos:**

1. **Aula 1 – Conhecendo o Problema do Cliente:**
   1. É uma boa prática inicializar um atributo do construtor e já declarar ele dentro dele mesmo. Portanto, não tem a necessidade de declarar ele fora do construtor, como estava sendo feito na classe cliente com o nome e cpf até agora.
   2. Não é legal ter duplicação de código.
   3. É interessante sempre colocarmos a variável inicial como privada e colocar um getter para poder ver os valores atribuídos a ela.
   4. Ter código totalmente compartilhado é ruim por falta de especificidade dependendo da alteração específica que queira fazer em uma parte só.
2. **Aula 2 – Herança:**
   1. Uma saída para colocar mais especificidade no código seria colocar uma variável de tipo, para que toda vez que uma conta for criada, você sempre saberá qual é o tipo dela, desse modo, podemos colocar vários if’s com as regras que quisermos.
      1. Porém isso possui a fragilidade de que sempre que criarmos uma nova conta, existe a chance de escrevermos o tipo de forma errada, e então as regras não irão se aplicar a elas, mesmo que devesse.
   2. Para concertar esse problema utilizamos a palavra-chave *extends class*, dessa forma, o código vai pegar tudo o que pertence naquela classe e trazer para essa nova classe que você está criando. Assim você pode adicionar nessa nova classe apenas as alterações que deseja fazer específicas dela.
   3. Chamamos isso de herança.
   4. Muitas vezes precisamos chamar o *super(valorCoConstrutor1, valorDoConstrutor2...)* para que uma classe derivada funcione corretamente.