Agora sim galera chegou a melhor parte que a hora de a gente meter a mão na massa e codar aí alguns desafios de código para vocês entenderem na prática como esse elemento da nossa metodologia educacional funciona. Para isso, como eu pensei em fazer de uma maneira super leve, acredito que vai ser bem didática também para que vocês entendam o contexto geral e, assim, abstraiam algumas coisas que talvez sejam até fatores que nos bloqueiam no começo, quando a gente está querendo aprender essa parte de lógica e pensamento computacional.

Então, eu separei aqui cinco desafios de código, cada um em uma linguagem de programação diferente. Então, o objetivo aqui é realmente que vocês entendam que a linguagem é um detalhe é só a forma de escrever algo que você já aplicou o pensamento computacional, ou seja, você já abstraiu aquele problema e já tem uma ideia de como resolver, e você vai ter que só saber como escrever naquela linguagem, que é transferir a lógica de programação, transcrever a lógica de programação que você já condensou e já deixou ela preparada na sua mente, beleza?

Então vamos fazer isso, a gente vai fazer juntos todos os cinco, para que vocês entendam as nuances de cada linguagem e também como a gente pode trabalhar esse processo de pensamento computacional, lógica de programação e codificação, beleza? Então bora lá, a gente vai ter o primeiro desafio de código em Java, o segundo em C#, o terceiro em JavaScript, o quarto em Python e o quinto em Kotlin e a gente vai ver que a medida que eles forem sendo resolvidos a gente vai introduzindo conceitos novos para que vocês entendam também algumas coisas interessantes que são características de cada linguagem de programação, mas que o contexto do problema vai sempre se manter o mesmo, o que vai ser legal para a gente conseguir continuar trabalhando a mesma ideia do início ao fim, beleza?

Então, bora lá, vamos para o primeiro aqui, vamos começar aqui com o desafio de código em Java, o enunciado galera, do desafio vai ser o mesmo para todos, então aqui a gente vai trabalhar sempre com o mesmo desafio. Então eu vou ler uma primeira vez e nos outros a gente só vai passando ali no que for novidade, beleza? Então o desafio é o seguinte: faça um programa que calcule e imprima o salário a ser transferido para um funcionário. Para realizar o cálculo receba o valor bruto do salário e o adicional dos benefícios.

O salário a ser transferido é calculado da seguinte maneira... e aqui tem uma fórmula de cálculo do salário. Legal. Para calcular o percentual do imposto, que é um dos elementos que tem aqui na fórmula, segue as alíquotas. Então, aqui a gente tem alíquotas diferentes de acordo com o salário do funcionário.

Então, se ele ganhar até R$ 1100.00, é 5.00%. Se ele ganhar de R$ 1101.01 até R$ 2500.00 é 10.00%, e maior que R$ 2500.00 é 15.00% de alíquota. Então, a gente tem já informações relevantes aqui sobre o nosso desafio. Então, a gente está entendendo o contexto, o domínio do nosso problema, que a gente costuma chamar, e agora, a gente vai entender qual vai ser a entrada.

Por que, galera? A gente sempre vai ter aqui nos desafios uma descrição, uma entrada e uma saída. E, geralmente alguns exemplos uma tabela de exemplos.

Porque, como um algoritmo é feito, qual é a estrutura de um algoritmo? Geralmente um algoritmo tem uma entrada, um processamento que é onde a gente vai trabalhar codificando aqui, e uma saída, né? Então, esses três elementos geralmente compõem um algoritmo: entrada, processamento e saída. E nos desafios de código aqui da DIO, isso não é diferente, na verdade isso fica muito evidente, porque a gente sempre vai ter uma entrada, vai trabalhar no processamento aqui, que vai ser o nosso código, e a gente vai ter uma saída no final, fechou?

Então, aqui, sobre a entrada, a gente está entendendo aqui quais informações a gente vai receber, né? Então, a entrada consiste em vários arquivos de teste que conterá o valor bruto do salário adicional de benefícios. Então, o que seriam esses vários arquivos de teste? São esses testes que estão aqui embaixo, galera, ó: teste 1, teste 2, teste 3, teste 4.

Então, a gente tem testes que são abertos e testes que são fechados. Está vendo esse aqui falando que esse não é um teste aberto? Os dois primeiros não são. Por que isso? Porque é importante terem testes fechados para que vocês reflitam realmente, façam todo o trabalho de pensamento computacional e lógica de programação pensando em diversos cenários, inclusive cenários que vocês não conhecem, porque senão vocês podem ser enviesados pelos dados de entrada e saída.

Então, é sempre importante ter alguns cenários de teste fechados. E aqui, nos abertos vocês conseguem ver o dado que vai entrar e o que o programa espera como saída. A mesma coisa para o teste 4 aqui, que também tem dados de entrada e dados de saída esperado. Então, deu para entender aqui como vão funcionar nossas entradas que vão ser esses dois valores aqui, ó. Beleza?

Então, a gente tem aqui dentro dos testes já exemplos, mas aqui também no enunciado tem exemplos para a gente poder visualizar também. E aí na saída, para cada arquivo de entrada, terá um arquivo de saída, um valor de saída, e como mencionado no desafio, será gerada uma linha com um número que será a diferença entre o valor bruto do salário e o percentual de imposto mediante a faixa salarial somada aos adicionais do benefício, que é basicamente essa formulazinha que a gente já viu aqui em cima.

Então, legal demais. A gente já está entendendo que o que a gente tem que fazer, basicamente é o salário e os benefícios de um funcionário e aplicar ali um imposto basicamente, que seria a alíquota em cima desse valor. E aí, galera, um ponto super importante, depois que a gente leu o desafio, entendeu ali o contexto do problema que a gente está querendo resolver, e a gente está ali fazendo o nosso pensamento computacional para conseguir abstrair aquele problema e pensar em uma lógica de programação para resolver através de um algoritmo, aqui a gente tem sempre nos desafios de código da DIO um template de código, um código já pré-estruturado que muitas vezes tem informações que são muito importantes para que vocês consigam resolver esse problema de maneira efetiva.

Então, por exemplo, aqui no caso, esse é um código Java. Então, aqui tem algumas dicas e comentários aqui em cima. "Para ler e escrever dados em Java, aqui na DIO padronizamos da seguinte forma. New Scanner. Cria um leitor de entradas com métodos úteis com o prefixo next". Então, olha que legal. Aqui, a gente já consegue ver que, na hora de ler os dados de entrada, a gente cria um scanner. E depois, os métodos têm sempre o prefixo next para a gente ler os valores. Então, sempre prestem atenção para a forma como o código está estruturado, os comentários que esse código tem, para que vocês consigam entender realmente quais são as características e nuances da linguagem que a gente está explorando aqui.

Uma outra coisa aqui também é que o System.out.println vai ser o comando para imprimir a nossa saída. Então, a gente precisa saber como recuperar os dados de entrada e também como imprimir os dados na saída. Então aqui a gente já tem alguns comentários que ajudam muito a entender como a gente vai fazer isso com Java.

E aqui pessoal, num primeiro momento pode ser que vocês se assustem, principalmente quem nunca programou na vida, vem e ver esse monte de código e fala "nossa eu não estou entendendo nada", mas fiquem calmos e calmas porque é super simples, é basicamente uma forma de representar como se fosse uma língua mesmo, uma linguagem de programação.

Então, eu escrevo em português de uma forma, em chinês é outra coisa, em japonês, enfim... cada língua tem uma maneira de representar visualmente a forma de se comunicar, e nas linguagens de programação não é diferente. Então, aqui o Java tem algumas características, a gente definiu uma classe chamada desafio, dentro dessa classe tem um método principal que vai ser onde a gente vai executar o processamento do nosso algoritmo. Lembra? Entrada, processamento, saída. Então aqui dentro desse método main vai ser o processamento do nosso algoritmo. E aqui a gente tem comentários que ajudam a entender também: "lê os valores de entrada". Então a gente instancia um leitor aqui, essa variável é o nosso leitor, e aí o nosso leitor, nextFloat, atribui o valor lido aqui para o valor salário e o segundo nextFloat, o valor dos benefícios.

Então a gente já tem duas variáveis na mão, valor do salário e valor dos benefícios. Então, fica fácil entender agora como a gente pode seguir. E aqui galera, basicamente o que a gente tem que fazer é calcular o valor do imposto, porque aparentemente todo o resto já está estruturado aqui pela forma que o código está.

Se eu olhar, a gente tem uma condição. Então, já começamos a entender um pouco de como a linguagem funciona. "If", em inglês significa "se". Então se o valor do salário for maior ou igual a zero, "e" esse aqui é um operador condicional, e valor salário menor ou igual a 1100, que condição que eu estou fazendo aqui?

Exatamente essa daqui, que está no enunciado. Se o valor do salário for de zero a 1100, eu preciso retornar 5% de imposto. Então, ele pega aqui o valor do imposto, que estava zerado antes, e aplica 5% aqui em cima do salário, ou seja, 0.05 que já está ali aplicando a formulazinha para extrair os 5% desse valor de salário. Então aqui está até comentado, atribui a alíquota de 5% mediante o salário. Então olha que legal,

a gente já tem uma primeira condição aqui que faz a gente começar a entender um pouco sobre como Java funciona e como a lógica pode ser estruturada.

E aí pessoal, geralmente nos templates de código que vão ter aqui nos desafios de código vocês vão ter TODOs, que é essa palavrinha reservada aqui, que são coisas que vocês têm que fazer naquele código. Então geralmente vocês não têm que se preocupar com tudo isso que tem aqui na tela, geralmente vocês vão resolver um pedacinho, entende? Então a intenção aqui também é que aos poucos vocês possam ir entendendo a sintaxe da linguagem que é a forma de escrever em Java, por exemplo, em C#, em JavaScript, em Python, em Kotlin enfim. Não importa a linguagem, galera. A lógica e o algoritmo, a lógica de programação é a mesma. Você só vai mudar um pouquinho a forma de escrever e vocês vão ver isso muito claramente aqui nos exemplos seguintes. Então, aqui, o TODO é o que a gente precisa fazer. Pode ter 1, pode ter 2, pode ter 10, pode ter quantos forem necessários para resolver o problema.

Aqui a gente tem um só, que é criar as demais condições para as alíquotas de 10% e 15%. Então, aqui a gente tem uma coisinha para fazer dentro desse código que já está estruturado, porque depois aqui ele já calcula e imprime a saída. Ele já pega o valor do salário, subtrai os impostos e adiciona os benefícios, que é exatamente a fórmula que a gente viu ali no enunciado.

E aqui ele usa o println que a gente viu no comentário lá em cima para fazer a impressão com duas casas decimais aqui, que é a forma de formatar no Java. Então, "não conheço alguma coisa aqui, fiquei com muita dúvida". Galera vocês podem ir pesquisar isso no Google, por exemplo, e, como que, o que significa um string format no Java com 1% .2f?

Vocês vão ver que vai aparecer em vários fóruns de desenvolvimento para vocês irem entendendo. Porque, galera, no começo é muito isso, você vai tendo dúvidas e tal. E hoje em dia, com as inteligências artificiais que a gente tem, por exemplo, o ChatGPT, o Bing Chat, o Bard, vocês podem usar a inteligência artificial para ajudar vocês a entender um código que às vezes está obscuro na cabeça de vocês.

Então, vocês podem pegar aqui essa linha, por exemplo, e falar assim, olha... Poderia me explicar no detalhe o que é esse, para que serve esse código? Então, galera, hoje a gente tem ferramentas muito poderosas que vão ajudar a maximizar o nosso tempo e também a nossa evolução técnica. Então, isso é muito legal.

Então, bora fazer aqui esse desafio para que a gente entenda como as coisas funcionam. Percebam o seguinte, pessoal. Aqui o código ele é funcional já, ele vai rodar mas ele vai dar erro. Então, vamos entender como funcionam os desafios de código aqui na DIO. Se eu clicar nesse botãozinho aqui, executar testes vocês vão ver que ele vai executar os testes e vai falhar.

Por quê? Porque a saída que a gente calculou não está igual à saída esperada. A saída esperada era 2050 e deu 2250. Então, assim, deu errado. Aqui embaixo, a mesma coisa. A saída esperada era uma e a nossa foi outra. Por quê? Porque a gente

não implementou todas as condições. Enquanto a gente tiver clicando nesse executar testes aqui, que é esse botãozinho azul, a gente não perde corações. Então, a gente não perde coraçõezinhos aqui, que é a estratégia aqui de gamificação da plataforma da DIO, que serve para que vocês consigam controlar, galera, aí, quando vocês vão começar uma jornada de estudos para que vocês tenham cadência para que vocês parem, reflitam sobre os erros que vocês tiveram.

Então, vocês só devem usar esse botãozinho aqui verde, entregar o desafio quando vocês tiverem certeza, ou quase certeza, de que o código de vocês está bem legal já. Então, se eu clicar aqui, sem fazer nada, tentar entregar meu desafio de projeto, perceba que eu perdi um coraçãozinho. Então, putz... Perdi aqui um coraçãozinho, então esse coraçãozinho ele vai ficar ali ao invés de 5, 4 e à medida que o tempo passa o coraçãozinho vai recuperando, beleza?

Então isso é super importante, vocês entenderem como funciona, né? Então sempre executa os testes aqui. Enquanto vocês estiverem codando ali, fazendo os testes do algoritmo de vocês, do código com a resolução que vocês implementaram, sempre vai testando aqui por executar testes. E aí quando tiver uma versão que você fala, pô, tá bacana, clica aqui em entregar o desafio, beleza?

Então vamos fazer isso junto. Vamos agora começar a codificar, então vou criar as demais condições. A gente tem um if aqui, então a gente poderia criar exatamente a mesma estrutura pessoal. Isso a gente conseguiria fazer replicando essa mesma estrutura e fazendo as condições específicas para 10% e para 15% de alíquota.

Então, se eu copiar exatamente esse if aqui e colar ele aqui, e colar outro aqui, por exemplo, vocês vão ver que a gente vai conseguir resolver o problema. Por exemplo, aqui, vou até tirar o comentário aqui para ficar um pouco mais conciso o nosso código, mas percebam que no enunciado a gente tem aqui de 1100.01 até 2500.

Então, eu poderia colocar aqui de 1100.01 até 2500. Então veja, aqui a minha condição já é a condição para alíquota de 10% que está no enunciado. Então eu venho aqui e coloco que a alíquota que eu quero aplicar, caso essa condição aqui seja verdade, é de 10%, legal? Então aqui a gente já começa a entender um pouco de como a lógica de programação funciona, porque eu preciso estruturar as minhas condições.

Então, se o valor de salário for dentro desse intervalo é 5%. Se for dentro desse intervalo é 10%. E se for maior do que 2500, é 15%. Então, aqui, maior que 2500, eu posso colocar uma condição só. Então, o valor de salário maior do que 2500, eu aplico 15%. Então, percebam que cada if, cada condição aqui, cada estrutura condicional representa uma regra de alíquota do nosso enunciado.

Legal, né, galera? Aqui, pessoal, a gente pode simplificar ainda mais. Conforme a gente for evoluindo a nossa capacidade de lógica de programação e for ganhando ferramentas e entendendo mais sobre a sintaxe das linguagens, vocês vão ver que o código de vocês vai melhorando gradualmente isso é algo totalmente natural.

Aqui, por exemplo, eu não necessariamente precisaria fazer sempre um if embaixo do outro. Por que? Porque se essa condição não for satisfeita, ou se ela for satisfeita por exemplo, se for verdade isso daqui eu nem preciso testar as outras, vocês concordam? Porque se o salário tiver entre 0 e 1100, ele não vai ser maior do que 1100.01 e nem maior que 2500. Então, aqui a gente pode fazer uma condição meio que está unida na outra. Então, eu posso colocar aqui um else if que faz com que essa segunda condição aqui agora só seja testada se o primeiro if falhar. Então, se a minha primeira condição falhar, faz sentido eu testar a segunda. Se as duas falharem, vocês concordam que o salário sempre vai ser maior do que 2500, partindo da premissa de que o salário não pode ser negativo?

Então, a gente poderia vir aqui e colocar simplesmente um else, que seria caso esse if não seja verdade e esse else if também não seja verdade, ele cai negativo no else. Legal? Então, olha que bacana. Aqui, ó, a gente já fez a técnica de refatorar um código, né? Então, galera, é isso que vai acontecendo na prática com vocês.

Vocês vão ir captando um pouco da essência da linguagem, vão ir entendendo como ela funciona e quando vocês menos esperarem, vocês vão estar refatorando o código, ou seja, tornando o código melhor, mais legível, mais performático e mais adequado para o seu contexto. Então muito provavelmente, pessoal, esse código que a gente tem aqui, essas novas condições que a gente adicionou aqui, vão atender ali as necessidades e requisitos que foram definidos no nosso enunciado. Então, vou executar os testes vamos ver se isso é realmente verdade. Nos testes abertos passou. Então, tudo certinho nos testes abertos deu verdinho. Então, agora é a hora de eu tentar entregar aqui o meu desafio.

Então, bora tentar entregar. Vamos ver se vai dar tudo certo. Pode ser que falhe. Não falhou, deu tudo certo. Mas pode ser que falhe nos testes fechados. Então, se falhar, a gente precisa analisar e ver onde foi o nosso erro. Mas show de bola, a gente ganhou aqui uma badge, concluiu aqui os quatro testes passaram.

Então, aqui a gente já viu um pouquinho de como é codar em Java, como é refletir um pouco sobre um problema. E agora vamos passar para uma outra linguagem de programação. E aí vocês vão ver, galera, que as linguagens de programação muitas vezes são extremamente similares.

Então, vou vir aqui para o segundo desafio. Ele é exatamente igual em termos de enunciado ao primeiro que a gente já resolveu em Java, mas ele é em C#, que é uma outra linguagem de programação. Mas vejam que os códigos galera, eles são muito parecidos. Extremamente parecidos. Muito, muito, muito parecidos. O que muda é o que a gente chama de sintaxe que são as particularidades de cada linguagem de programação.

Então, olha aqui que legal. Aqui, no C#, percebam que ele é muito parecido com o Java. Tanto que se a gente pegar aquele mesmo código que a gente fez lá para resolver o nosso problema e colocar aqui, muito provavelmente vai funcionar. Então, vamos ver se isso é verdade. Vamos tentar executar e vamos ver o que vai acontecer.

Passou nos testes. Vocês viram? Então, vejam que Java e C# são linguagens muito próximas então o fato de eu já ter feito lá em Java fez com que basicamente o meu código aqui fosse idêntico no C#. E aí galera, o que muda de uma linguagem para outra? É um conhecimento que vocês vão adquirindo com o tempo.

Cada linguagem tem a sua convenção, tem convenções boas práticas, que são formas mais elegantes e padronizadas que a comunidade recomenda que você codifique naquela linguagem. Por exemplo, no caso do Java, o escopo aqui das condições de métodos e tudo mais, a gente coloca na mesma linha, a gente abre na mesma linha, e geralmente a gente encadeia uma outra condição também na mesma linha.

No C#, é uma boa prática, é uma convenção da linguagem que as chaves aqui das condições fiquem na linha seguinte e também as condições. Então, aqui, se a gente for seguir as convenções, a gente teria que fazer algo assim. Beleza? Então, a gente teria um código um pouquinho diferente se a gente for seguir exatamente as convenções, mas é exatamente a mesma coisa, né? Em termos de código, código, é exatamente a mesma coisa. Só muda aqui a forma de formatar, vamos dizer assim, né? A nossa solução. Então, se eu executar de novo aqui, vocês vão ver que vai passar, a gente não mudou nada, né? A gente basicamente só seguiu as convenções, as boas práticas do C# em relação às estruturas condicionais que a gente implementou aqui para resolver o nosso problema.

E vejam que o que muda de uma linguagem para outra é, por exemplo, a forma de ler as entradas. Então, veja que aqui em C#, a gente usa o Console.ReadLine. E aqui está descrito também nos comentários ali que são introdutórios do desafio de código. E para escrever a saída, Console.WriteLine, que está aqui embaixo.

E para formatar com duas casas é dessa forma aqui. Então, a gente começa a perceber as particularidades de cada linguagem. Então, no Java, era de um jeito aqui é de outro. Então, aqui é ToString passando 0.00. No Java, para formatar com duas casas decimais a saída, era %.2f. Então, cada linguagem tem padrões de formatação, padrões de codificação e classes e formas de fazer coisas diferentes, o que representa as características daquela linguagem, beleza?

Então, galera, ó, basicamente a gente finalizou aqui o nosso segundo desafio, né? Vamos tentar entregar ele agora, né vamos ver se vai dar bom. Vou, ó, beleza. Deu tudo certo. E bora partir. Para o próximo agora, que esse vai ser bem diferente desses dois primeiros que a gente viu em Java e em C#. Agora a gente chegou aqui no desafio que é mesmo desafio galera, em termos de enunciado, mas agora a gente vai codar com JavaScript, a gente vai codificar com JavaScript.

E perceba que o código já é bem diferente, né? Então Java e C#, a gente tem muito mais código, vamos dizer assim, né, o JavaScript é mais conciso, né, veja que boa parte do código aqui é comentário, né, tudo isso que está em cinza aqui são comentários, então o código de fato são poucas linhas, então a gente já percebe que o JavaScript é menos verboso, né, a gente costuma dizer isso, que o JavaScript é menos verboso do que o Java e o C#, por exemplo, né, você coda menos para ter o

mesmo resultado vamos dizer assim, beleza? Isso é uma característica da linguagem, não quer dizer que ela é melhor ou pior, é só uma característica dela, beleza? Então aqui a gente tem o gets, que é uma função que a gente implementou aqui dentro da própria DIO para ler uma linha de dado, então aqui para pegar o valor do salário o valor dos benefícios a gente usou o GETS e o PRINT que a gente também implementou aqui na DIO, que não são funções nativas do JavaScript, por isso que eu estou explicando, ele também está disponível aqui, então a gente sempre vai colocar aqui esse cabeçalho vamos dizer assim, com comentários que são úteis para vocês entenderem como entregar os desafios.

E aqui, nesse código JavaScript, o que é legal? Tem alguns comentários que vão nos ajudar a entender algumas coisas. Por exemplo, aqui ele fala que o cálculo do imposto é feito através da função calcular imposto. Está vendo que tem uma função aqui embaixo? Então, aqui a gente já começa até a botar o pezinho em entender que existem paradigmas de programação.

O que é um paradigma? É uma forma de resolver problemas. Aqui, quando a gente fala de paradigma funcional, por exemplo, é um paradigma que usa funções para tentar organizar melhor o código. Então, veja que lá no Java e no C#, apesar de serem linguagens que são orientadas a objetos e podem utilizar diversos paradigmas de programação, aqui a gente usou uma abordagem que a gente foi fazendo ali o código para resolver o problema. Então, a gente não se preocupou em usar paradigma de programação nenhum. A gente se preocupou em resolver o problema. Nesse pequeno código JavaScript, tem aqui o uso de uma função que já demonstra um pouco de um paradigma funcional. Obviamente de uma maneira muito simplista a gente está falando aqui, mas você já começa a ver características de paradigmas de programação que são essencialmente formas diferentes de programar ou de resolver problemas.

Então, aqui, galera, a gente tem uma função responsável por calcular o imposto, que vai ser aqui que a gente vai fazer os nossos ifs e else's e condições. Então, percebam que existe uma característica específica aqui. Nessa função, a gente tem uma variávelzinha aqui chamada alíquota que ela está recebendo a porcentagem de alíquota que tem que ser aplicada de imposto. Então, basicamente as nossas condições vão ser muito similares às que a gente fez aqui, só que a diferença é que a gente vai aplicar, atribuir para a variável que foi definida na função apenas esse valorzinho aqui, 0.10 e 0.15, que seria o equivalente a 10% e 15%. Legal?

Então, aqui, o que a gente pode fazer? Galera, a gente poderia pegar exatamente aqui a forma de representar as condições do C# e do Java também. Então, eu vou vir até aqui e vou fazer isso. A única coisa que vai mudar aqui é que assim, galera, é até uma boa dica aqui, muitas vezes você vai copiar código que você já fez, que é similar ao problema que você quer resolver, mas nem sempre esse código que você copiou está adequado para o contexto que você está querendo usar.

Por exemplo, está vendo que aqui eu tenho a variável valor salário, só que o valor salário não existe no escopo dessa função. Na função, ela se chama salário. Então, aqui, tudo que tiver valor salário, eu preciso trocar por salário, beleza? E aqui, ó, perceba que eu não tenho valor imposto aqui dentro da função, ó. A função, pessoal,

ela define um escopo próprio. Então, tudo que está aqui é escopo da minha função. Minha função não está enxergando essa galera que está aqui em cima. Ela não está preocupada com isso, não deveria. Então, aqui eu preciso só usar alíquota igual foi feito aqui no primeiro if, atribuindo o valor para ela.

Então, olha. Aqui a alíquota é igual a alguma coisa e embaixo também a alíquota é igual a alguma coisa. Aqui no JavaScript eu não preciso colocar o F de ponto flutuante no valor. Então eu posso simplesmente colocar 0.10 e 0.15. Então percebam que as nossas condições ficaram até mais simples. A gente simplesmente atribuiu a alíquota aqui. Por quê? Porque essa função está fazendo o cálculo aqui embaixo já, da alíquota vezes o salário. Então, ela criou uma variável auxiliar aqui para receber o valor da alíquota e, no final, ela faz o cálculo do imposto com base no salário que ela recebeu. Então, olha que bacana. A gente usou uma função calcular imposto que está sendo chamada aqui.

E que retorna o valor do imposto para essa constante aqui que a gente chamou de valor do imposto, legal? Então, show de bola, né galera? Veja que o JavaScript já tem uma pegada diferente. Aqui, por exemplo, em JavaScript, a gente não se preocupa em colocar tipos nas variáveis, ó. Uma variável é const quando ela tem um valor que não vai ser reatribuído, ele não vai mudar ao longo do tempo, ou ele pode ser um let, por exemplo, quando ele vai mudar. Por exemplo, a alíquota recebeu valor aqui, recebeu o valor aqui, recebeu o valor aqui. Apesar de serem atribuições Individuais e que vão ser exclusivas, aqui a ideia é representar que o let tem essa característica de poder ter atribuições múltiplas tanto que a gente poderia começar zerado aqui e ele poderia receber outras atribuições aqui na sequência sem maiores problemas, porque o let e o var do JavaScript, ambos funcionariam dessa maneira, eles servem para isso.

Enquanto se a gente olhar ali no Java, a gente tinha que colocar aqui o tipo float, que é o tipo de ponto flutuante. O leitor de entradas ó, é do tipo scanner. No caso do C#, load aqui também, legal, então galera o JavaScript já traz essa característica do tipo dela ser dinâmico. Dentro dessa variável aqui eu poderia colocar um float, eu poderia colocar um texto, eu poderia colocar um inteiro, eu poderia colocar o que eu achasse mais adequado e poderia mudar isso no momento que eu quisesse. Então, assim com grandes poderes vem grandes responsabilidades, né. Então assim é uma linguagem muito bacana, muito fluida, muito dinâmica, mas vocês precisam se atentar às características dela.

Então, foi legal aqui também para a gente conhecer um pouquinho do JavaScript. Então, vamos ver se os nossos testes vão funcionar, né? Então, bora ver aí, ó, show, galera! Passou nos testes então, ó funcionou aqui também o nosso código e agora... Vou entregar aqui, tentar entregar o nosso desafio de código. Show de bola, galera, deu certo também.

E bora para o próximo. Beleza. Galera agora a gente chega em uma outra linguagem que é o Python, que é uma linguagem também incrível. Assim como os outros desafios a gente tem também aqui um cabeçalho explicando que o input vai ler a linha de entrada e o print vai imprimir a saída pra gente. E aqui, galera, no Python, eu queria explicar uma coisa importante pra vocês.

Vamos supor que você tá codando aqui e, cara do nada, você, putz fez uma zoeira aqui, velho. Eu falo putz cara, perdi meu código, perdi o template, caramba. E aí, você tá com o seu código salvo aqui, e aí se você carrega de novo continua com aquele código que você tinha salvo. Então, você fala, meu Deus do céu, perdi o template que eu precisava para conseguir codar aqui. O que você faz, galera? É só apertar aqui em Restart Code, beleza? Clicou, voltou o template do jeito que estava no início do desafio, beleza? Você volta o template original, tá bom? Então isso é super importante, tá galera? Beleza? Então, bora conhecer agora aqui o Python.

Então, galera, o Python é bem próximo ali do JavaScript, ele também não tem definição de tipo, vocês vão ver isso. E o Python tem ainda uma característica que é muito particular, que é ele não tem as chaves aqui que o JavaScript tem, por exemplo para definir o escopo de uma função.

Tudo no Python é baseado na identação. O que é a identação? São esses espacinhos aqui, né? Então, por exemplo, tudo que está com dois espaços aqui faz parte dessa função calcular imposto. Então, tudo isso aqui faz parte dela. Então, o Python tem essa característica que é bem legal, dessa linguagem. E aí, o nosso TODO está aqui também, dentro de uma função que é criar as demais condições. Então, veja que o Python consegue ser ainda mais simples que o JavaScript, né? Pelo menos para esse exemplo.

Então, aqui, o valor de salário vai ser o primeiro input, o valor do benefício o segundo input. A gente chama a função de calcular imposto recebendo o valor de retorno em uma variável nova. E a saída é a fórmula que está aqui no nosso enunciado. E aí, a gente imprime. E aqui, para formatar com duas casas é .2f algo muito similar ali ao que a gente viu no Java. Então bora codar aqui. E aqui, galera, a gente vai ver o seguinte... que o Python tem, por não ter a necessidade de colocar as chaves aqui, ele vai ter uma sintaxe um pouquinho diferente das outras. Então, se a gente pegar o nosso código JavaScript aqui e tentar colocar lá, o que vai acontecer?

O nosso código JavaScript tem as chaves aqui. Então, pô... Não pode ter em Python, então dois pontos, beleza? Outra coisa, no Python o else if é o elif, beleza? E aqui a gente poderia ter o else aqui, beleza? Então olha que legal, a gente tem exatamente a mesma lógica a diferença aqui também é uma diferença importante o Python não precisa de ponto e vírgula. Então em todas as outras linguagens que a gente viu até agora, em Java, C# e JavaScript, a gente estava usando ponto e vírgula. No JavaScript nem é obrigatório, mas a gente estava usando, mas aqui no Python não. Então é uma característica da linguagem também. Então, vamos ver aqui se a nossa estrutura condicional aqui está ok.

Então, vamos executar os testes e vamos ver se aquelas condições funcionaram. Então, olha aqui, a gente tem algum problema aqui, galera. Algum problema aconteceu aqui no nosso elif. Então, está falando que é uma sintaxe inválida. Então, algum problema tem aqui nessa condição. E já dá para ver aqui que no Python, a gente tem que usar o "and", ao invés dos dois e comerciais (&&). Então, o operador lógico aqui é diferente. Beleza? Então, vamos ver se corrigindo isso, o erro some. Então, olha que legal, galera. Então, a própria plataforma aqui da DIO vai dar feedbacks para vocês através de stacks de erro, que a gente chama. Então, se eu

voltar aqui, vocês vão ver que o erro que apareceu aqui me ajudou a entender o que estava acontecendo. Ele falou que tinha um erro de sintaxe né? Uma sintaxe inválida nesse elif aqui. Então, pô o que poderia ser? Ah, caramba aqui é and. Legal?

Então... Basicamente isso, galera. A gente consegue ter esse feedback também e reagir ao que os testes nos dizem também em termos de erro. Então, isso é uma característica muito importante para um desenvolvedor saber ler as pilhas de erro, saber entender o que está acontecendo nos erros, porque são os erros que vão até nos ajudar a buscar no Google, perguntar para um chatbot, para uma inteligência artificial, como a gente pode resolver os nossos problemas aqui, legal? Então, coloquei o and, vou executar os testes, vou tentar entregar aqui para ver se vai funcionar certinho, e show de bola. Agora, a gente vai aí para o último desafio né, dessa jornada aí, de a gente ir entendendo aí que a linguagem de programação galera, é só um detalhe.

Vocês perceberam que a lógica de programação até agora é a mesma? Em todas as linguagens a gente fez a mesma lógica a gente fez uma condição para essa regra e uma outra condição para essa regra, fazendo com que essas três regras fossem verificadas no nosso código. Então, essa é a nossa lógica de programação.

E o pensamento computacional é o que a gente imaginou para resolver esse problema. A forma como a gente estruturou isso mentalmente para resolver na prática. Então, bora para o último aqui. O último é um pouco mais capcioso, eu diria. Ele tem algumas particularidades aqui, porque com Kotlin, galera, a gente traz um pouquinho de um conceito que é um paradigma novo também aqui para o nosso cenário que a gente está aprendendo aqui e conhecendo essas linguagens.

Aqui com o Python, a gente quis dar uma pinceladinha no conceito de orientação a objetos. O que é orientação a objetos? É você tentar fazer com que o seu código represente de uma maneira quase que um para um, quase que como um espelho da realidade. Então, aqui, por exemplo, a gente criou um objeto, uma estrutura chamada Receita Federal, e a Receita Federal calcula imposto. Então, olha que legal que fica o nosso código, a gente fala que ele fica mais legível, porque essa aqui é a nossa função principal. Aqui em Kotlin, para ler a linha, para ler a entrada é o Readline, então eu estou lendo o valor do salário. Estou lendo o valor dos benefícios, certo?

E aqui, o valor do imposto olha que bacana. O valor do imposto é igual a receita federal, ponto, calcular imposto, passando o valor do salário. Então, quem vai calcular o imposto é a receita, olha que legal. Então, assim, você já começa a fazer um link com o mundo real. Então, isso é muito massa. Então, aqui, a gente começa a enxergar um pouquinho de qual é a ideia do paradigma orientado a objetos.

O paradigma orientado objetos traz essa visão de você abstrair um problema tentando fazer com que a forma de você modelar ele aqui, de você resolver ele na prática, fazer com que a sua lógica de programação represente isso através de classes, objetos, como a gente está vendo aqui, a gente tem aqui um objeto receita federal que tem a função calcular imposto, isso faz com que na hora que a gente ler o código, ele seja mais fácil de entender, vamos dizer assim, mais fácil de relacionar com o mundo real.

Então, isso é bem bacana e é um paradigma bem interessante e muito presente em todas essas linguagens que a gente falou aqui. Então, galera, é legal que vocês tenham essa percepção também, que à medida do tempo, vocês vão ir conhecendo novos paradigmas conhecendo novas técnicas, conhecendo novas formas de resolver problemas. Muitas vezes o mesmo problema, muitas vezes uma alternativa mais interessante, uma alternativa com menos código, uma alternativa que é mais performática.

Então, vocês precisam ter esse interesse também de putz deixa eu ver se tem uma solução melhor para isso. Então, isso é super importante, você realmente ser protagonista da sua jornada. Legal? Então, bora aqui fechar esse último desafio aqui, que é em Kotlin, que é uma outra linguagem de programação, e aqui, galera, por essa estrutura dá para ver que ele tem uma estrutura chamada when, onde a gente consegue colocar as condições dentro.

Então, o Kotlin tem umas características diferentes. Então, o que a gente pode fazer? A gente pode simplesmente copiar essa primeira estrutura que já tem algo pronto para a gente. E aqui, a gente pode realmente fazer as mesmas condições que a gente fez aqui nos outros. Então, por exemplo, pegar essa condição aqui, colocar e falar que para esse caso a gente tem 10%. E no else, por exemplo, eu poderia falar que a gente tem 15%. Então, olha aqui, em Kotlin basicamente seria isso. Então, vamos tentar executar vamos ver se é isso mesmo. Vamos ver se vai dar bom...

Então, galera, é legal enxergar isso, linguagens de programação diferentes resolvendo o mesmo problema. Então deu para captar um pouco das características de cada uma delas. A ideia desse conteúdo, desse vídeo, não é que vocês saiam daqui especialistas em nenhuma dessas linguagens, mas só mostrar para vocês que as linguagens de programação têm muitas similaridades porque elas se propõem resolver os mesmos problemas, e isso é super importante.

Olha aqui, galera, mais um erro aqui, para a gente interpretar. Aqui ele está falando que tem algum problema nessa condição aqui. Legal? Então, aqui a gente já consegue enxergar um potencial problema. Aqui em Kotlin é o e comercial (&) não é o and igual no Python. Então, olha a gente sendo punido por ter copiado o código. Então vamos tentar executar de novo. Bora ver se agora vai dar bom. Acredito que era mais esse ponto mesmo. Vamos ver se der algum outro erro. A gente analisa e corrige também.

Então, isso que é legal, galera. Vocês não precisam sair da plataforma da DIO para codar. Vocês vão ter os desafios de código aqui à disposição de vocês para poder realmente sempre estar exercitando essa parte. Então, funcionou. Agora, a gente conseguiu aqui fazer esse código também em Python, em Kotlin, nesse último.

Então, galera, recapitulando, a gente pegou o mesmo desafio de código a nível de descrição, a nível de enunciado. E a gente viu um pouco de como as linguagens de programação, como diferentes linguagens de programação poderiam aplicar a mesma lógica de programação. Então, olha que massa, a gente viu código em Java, a gente viu código em C#, a gente viu código em JavaScript, a gente viu código em

Python, a gente viu código em Kotlin, cada um com as suas características, cada um com a sua forma de escrever, como se fossem línguas mesmo, distintas, as linguagens de programação trazem maneiras de a gente colocar de forma concreta a nossa lógica de programação.

Então galera, espero que vocês tenham curtido aí, a ideia era realmente desmistificar um pouco essa questão de que você precisa saber só uma linguagem de programação, a lógica não é tão importante, o importante é saber a linguagem. Não galera, a lógica de programação e pensamento computacional talvez seja mais importante do que a linguagem, porque a linguagem é só um meio de você resolver o problema. O pensamento computacional e a lógica é o como. Então, isso é o mais importante. Beleza?

Então, galera, espero que vocês tenham curtido. A gente se vê aí durante alguma experiência educacional que vocês estiverem fazendo na DIO. Conto com o feedback de vocês nesse conteúdo, tamo junto e vamos pra cima galera, sejam protagonistas das suas jornadas e contem conosco aqui da DIO. Um abraço e até mais.