Linguagem Kotlin: o que é, para que serve e um Guia para aprender

Formação Linguagem Kotlin

Confira neste artigo:

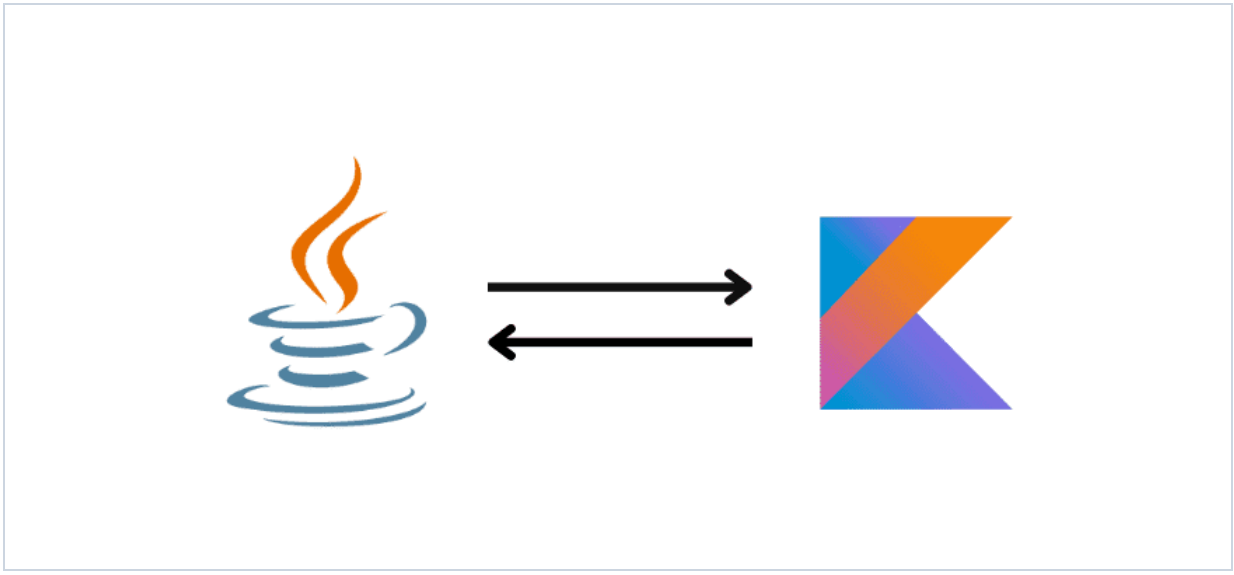
Introdução

No entanto, para muitas pessoas, o Kotlin ainda é uma linguagem relativamente nova e desconhecida, o que pode levantar muitas perguntas.

Neste artigo, vamos nos aprofundar no mundo do Kotlin e descobrir o que o torna uma linguagem de programação tão moderna!

O que é Kotlin?

Kotlin é uma linguagem de programação orientada a objetos compatível com a plataforma Java e que pode ser executada em máquinas virtuais Java (JVM), além de outras plataformas, como o Android. Além disso, Kotlin tem uma sintaxe semelhante ao Java e muitas das suas características e conceitos de programação são parecidas.



É uma linguagem moderna que pode ser **utilizada para desenvolver aplicativos Android, web e desktop**. Conhecida também como uma linguagem de programação prática, ela foi projetada para ser **segura e concisa**, assim permite que pessoas desenvolvedoras escrevam menos código para realizar tarefas simples.

veio para ficar, o **episódio #310 - Case Contabilizei: Kotlin**, do podcast Hipsters Ponto Tech, apresenta um estudo de caso interessante da maior plataforma contábil do Brasil, a Contabilizei, que adotou a linguagem de programação em seu dia a dia. Nele, você aprenderá sobre as vantagens e desvantagens, desafios da implementação e manutenção e se realmente vale a pena investir em Kotlin. Se você é uma pessoa entusiasta de tecnologia ou desenvolvedora curiosa, este episódio é para você!

> *Case Contabilizei: Kotlin – Hipsters Ponto Tech #310*

spotify:5ddMXYV6mP2tKN05zCE2c1:episode

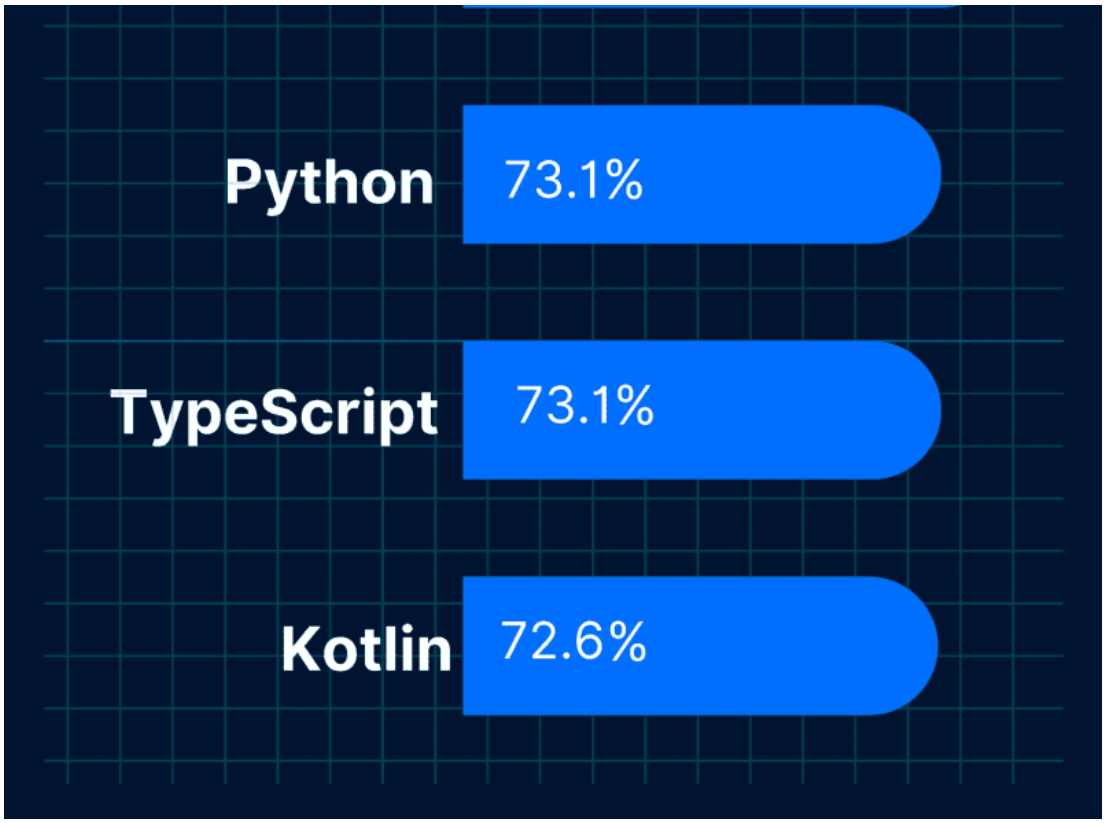
História do Kotlin

A história do Kotlin se **inicia em 2010**, quando a Jetbrains, mesma criadora do IntelliJ IDEA, decidiu criar uma nova linguagem de programação para a JVM (*Java Virtual Machine*) com o objetivo de criar uma linguagem moderna, concisa, segura e além disso, interoperável com o Java.

Em 2011, o Kotlin foi apresentado publicamente pela primeira vez na conferência JVMLS (*Java Virtual Machine Language Summit*). O lançamento oficial do Kotlin 1.0 aconteceu em fevereiro de 2016.

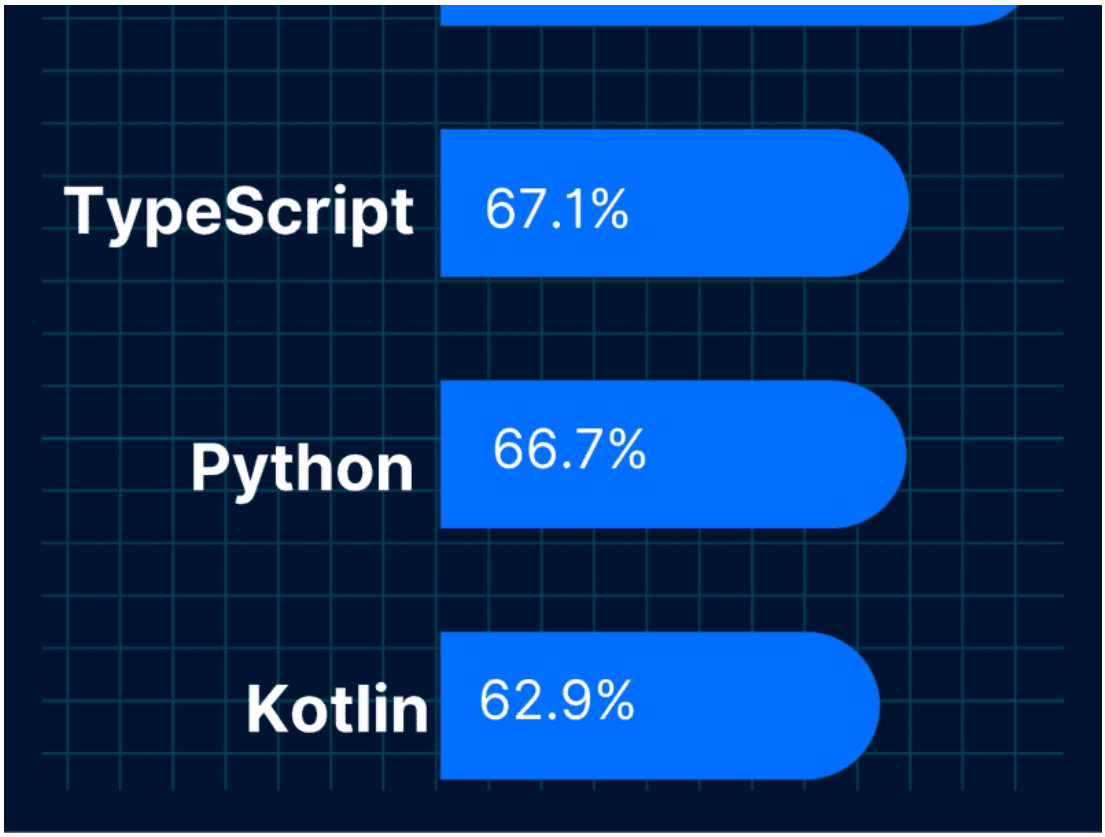
Desde seu lançamento, o Kotlin tem ganhado popularidade rapidamente. Em 2017, a Google anunciou que o Kotlin seria uma linguagem oficial de desenvolvimento para o Android, e, em **2019** e **2020**, o Kotlin foi classificado como a **quarta linguagem de programação mais amada** nas pesquisas 2019 Developer Survey e 2020 Developer Survey realizadas pela Stack Overflow.

Pesquisa 2019 Developer Survey - Stack Overflow



Créditos: "2019 Most Loved, Dreaded, and Wanted Languages", Stack Overflow

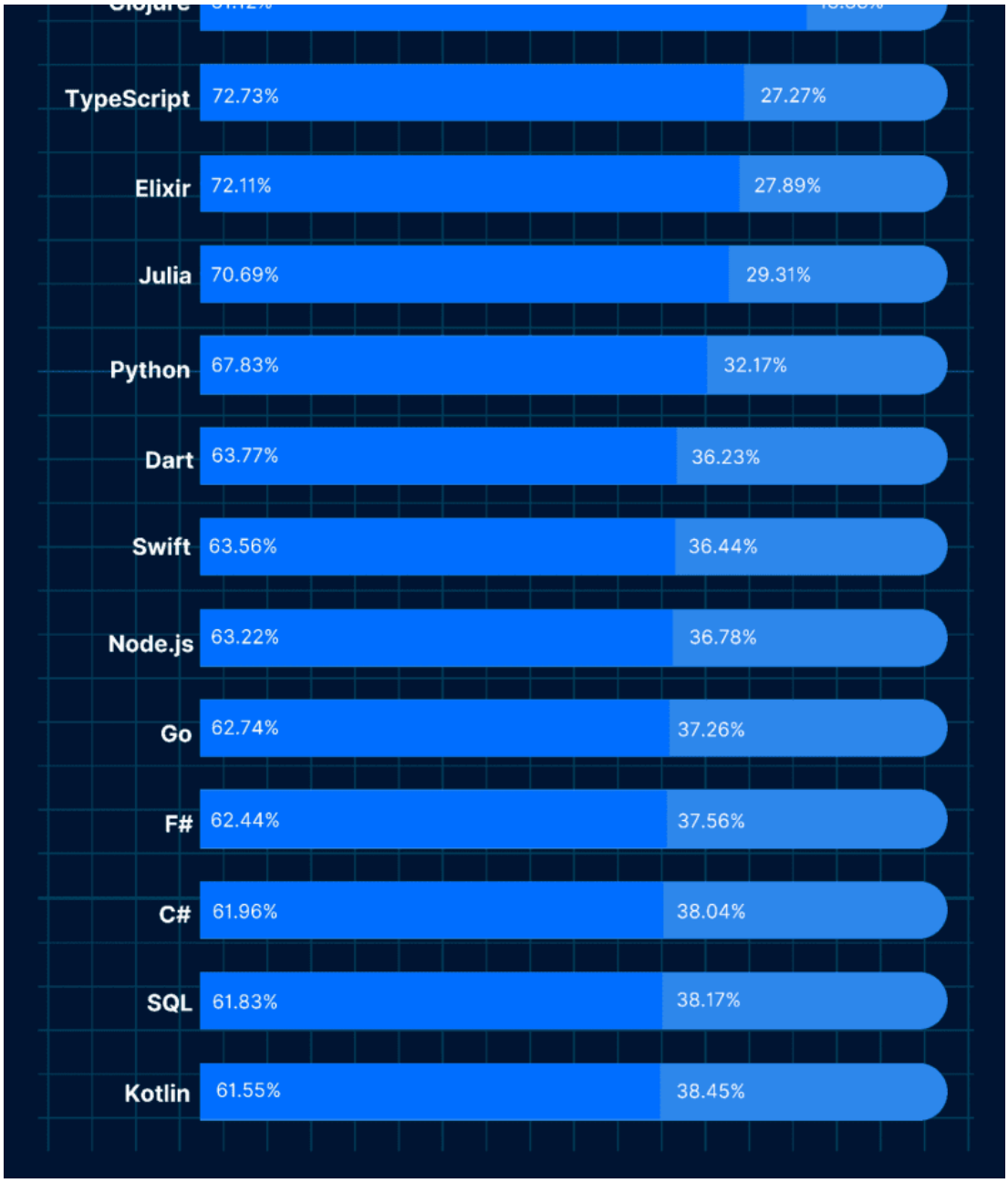
Pesquisa 2020 Developer Survey - Stack Overflow



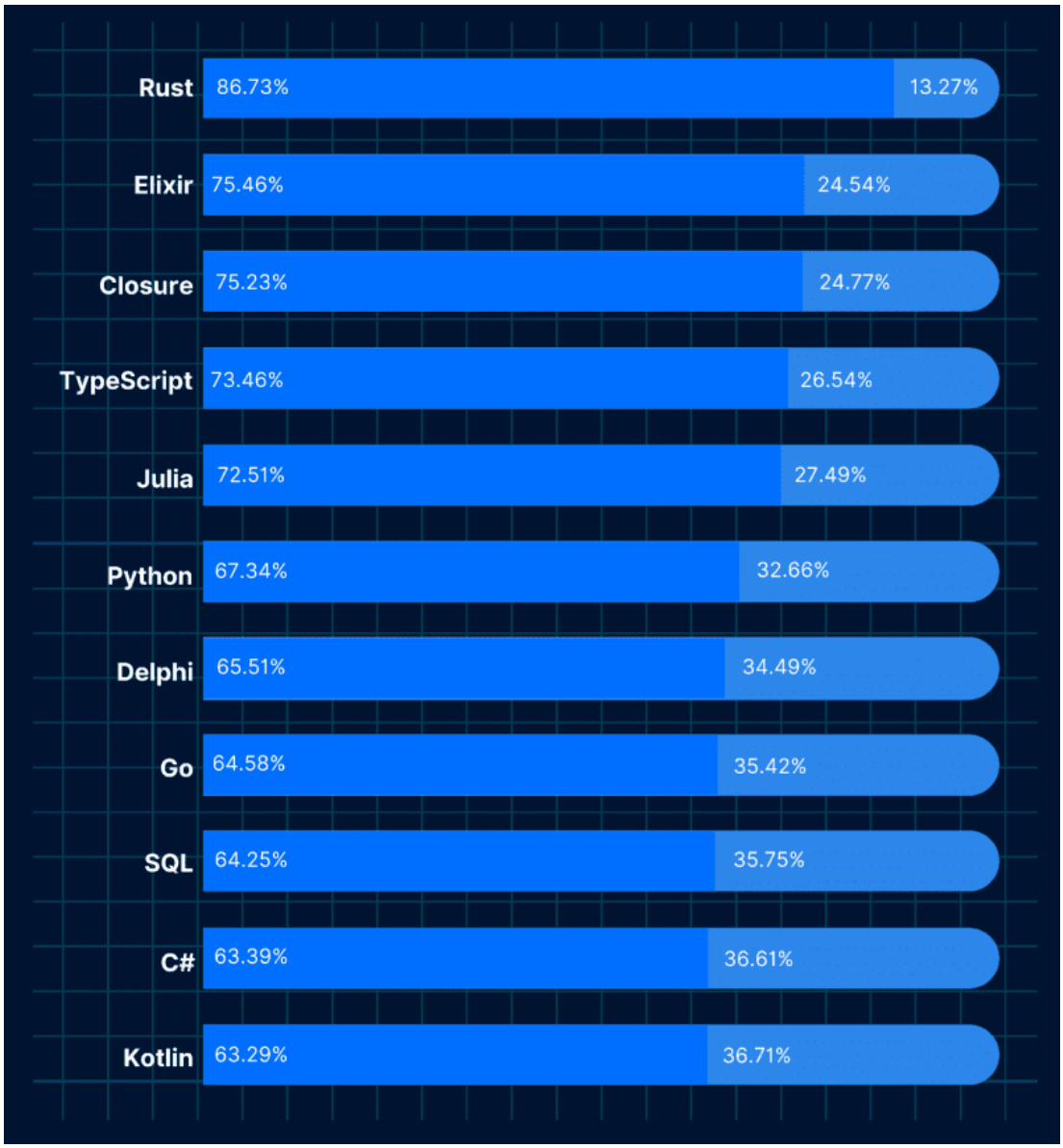
Créditos: "2020 Most Loved, Dreaded, and Wanted Languages", Stack Overflow

No ano seguinte, a linguagem passou para a 14ª posição na pesquisa 2021 Developer Survey, subindo para 11ª posição na pesquisa 2022 Developer Survey entre as linguagens mais amadas.

Pesquisa 2021 Developer Survey - Stack Overflow



Créditos: "2021 Most Loved, Dreaded, and Wanted Languages", Stack Overflow



Créditos: "2022 Most Loved, Dreaded, and Wanted Languages", Stack Overflow

Para que serve o Kotlin?

É possível usar o Kotlin para desenvolver apps em uma variedade de plataformas, incluindo:

No back-end, o Kotlin também está cada vez mais sendo adotado para o **desenvolvimento web** ou **sistemas de servidor**, com diversos frameworks e bibliotecas disponíveis para ajudar o desenvolvedor.

No desenvolvimento para Android, Kotlin se tornou uma linguagem popular por sua **sintaxe concisa e interoperabilidade com o Java**. É uma maneira que pode ser mais fácil e eficiente para desenvolver aplicativos Android, dependendo da necessidade do projeto.

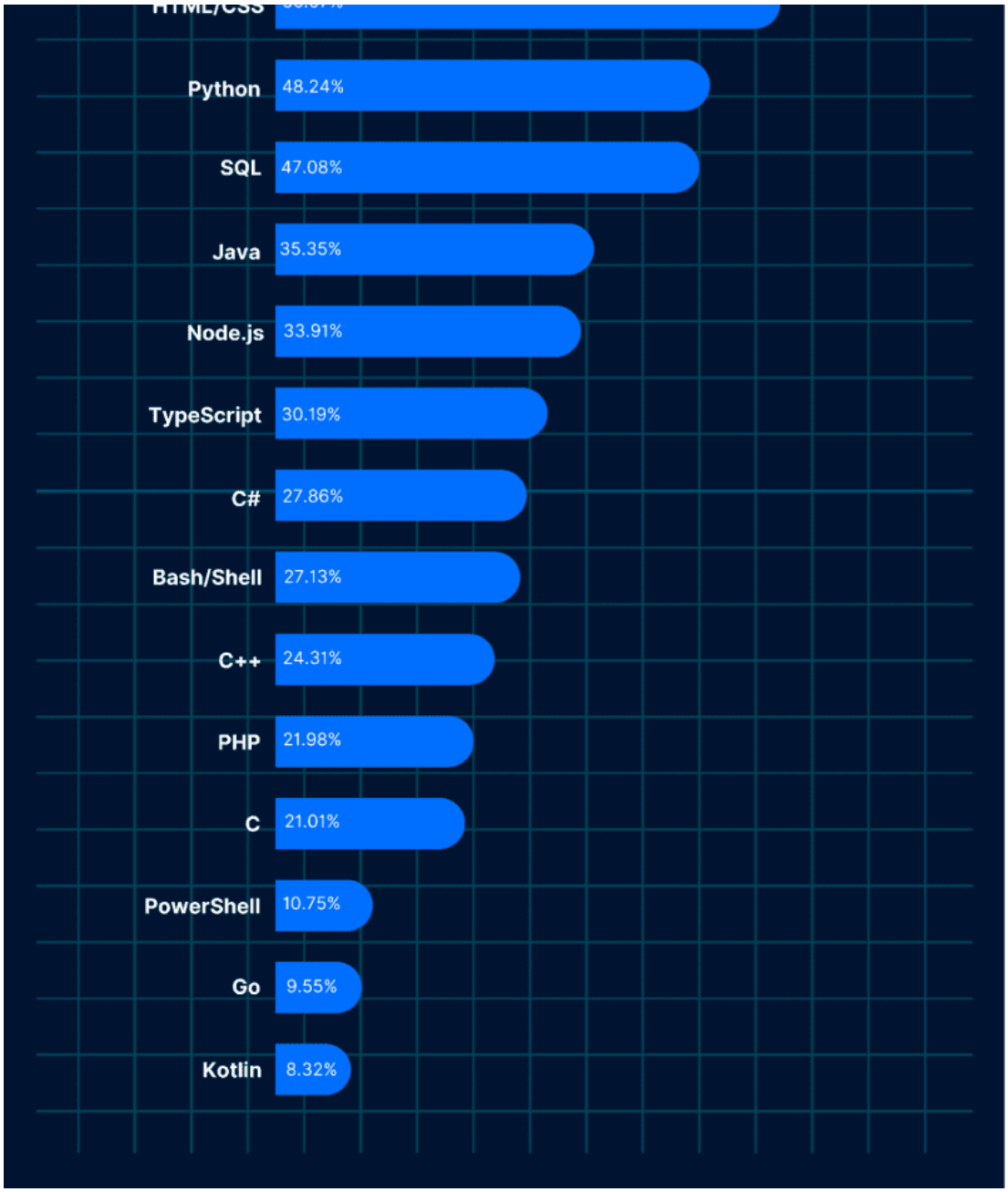
Embora seja possível utilizar tanto no desenvolvimento de aplicativos Android quanto no desenvolvimento de sistemas de back-end, existem algumas diferenças importantes em como a linguagem é usada em cada um desses contextos. Quando utilizamos o Kotlin no desenvolvimento Android, o código é executado em uma máquina virtual Android; já em sistemas back-end, o código é executado em um servidor de aplicativos ou máquina virtual Java.

Por que aprender?

Mas, afinal, por que aprender essa linguagem? Kotlin é uma linguagem moderna, bem expressiva, segura e concisa. Ela possui interoperabilidade com Java, o que pode ser muito positivo para quem já tem intimidade com o Java, pois é possível aproveitar o ecossistema dessa linguagem querida ou temida!

Além disso, **Kotlin é multiplataforma, isto é, suporta diversos paradigmas de programação**, como orientação a objetos, funcional, imperativa, entre outros. A curiosidade também pode ser uma ótima motivação para aprender, afinal, é sempre bom experimentar linguagens novas, para ampliar os horizontes.

Em 2021, a linguagem ficou entre as 15 mais usadas no mundo, de acordo com uma pesquisa realizada pela Stack Overflow naquele ano. O Kotlin aparece na 15ª posição no ranking, com 8,32% da preferência das pessoas desenvolvedoras.



Créditos: "2021 Developer Survey", Stack Overflow

Como vimos, Kotlin vem ganhando popularidade nos últimos anos e há **empresas que já adotaram Kotlin** para desenvolver seus aplicativos e projetos. Algumas delas são:

- Netflix.

Com essa crescente demanda por pessoas desenvolvedoras com conhecimento em Kotlin, há oportunidades de carreira para quem deseja se especializar. Os **perfis de pessoas especializadas são diversos**, e alguns dos mais comuns são:

- Devs back-end;
- Devs Java;
- Devs mobile;
- Devs de jogos;
- Engenheiros de software.

Como aprender Kotlin?

O aprendizado de uma nova linguagem pode ser desafiador, especialmente se estamos muito acostumados com a sintaxe de uma linguagem bem diferente da que queremos conhecer.

No entanto, isso varia completamente de pessoa para pessoa. Por ter características parecidas com o Java, o Kotlin pode se tornar mais acessível em um primeiro momento para quem já conhece a linguagem.

Confira o vídeo “Como aprender melhor?” com dicas importantes para aprimorar seus estudos e atingir suas metas, apresentado por Paulo Silveira e Diogo Pires no #HipstersPontoTube. Aprenda a utilizar hábitos, prática, memória e portfólio para expandir seu conhecimento na comunidade de tecnologia.

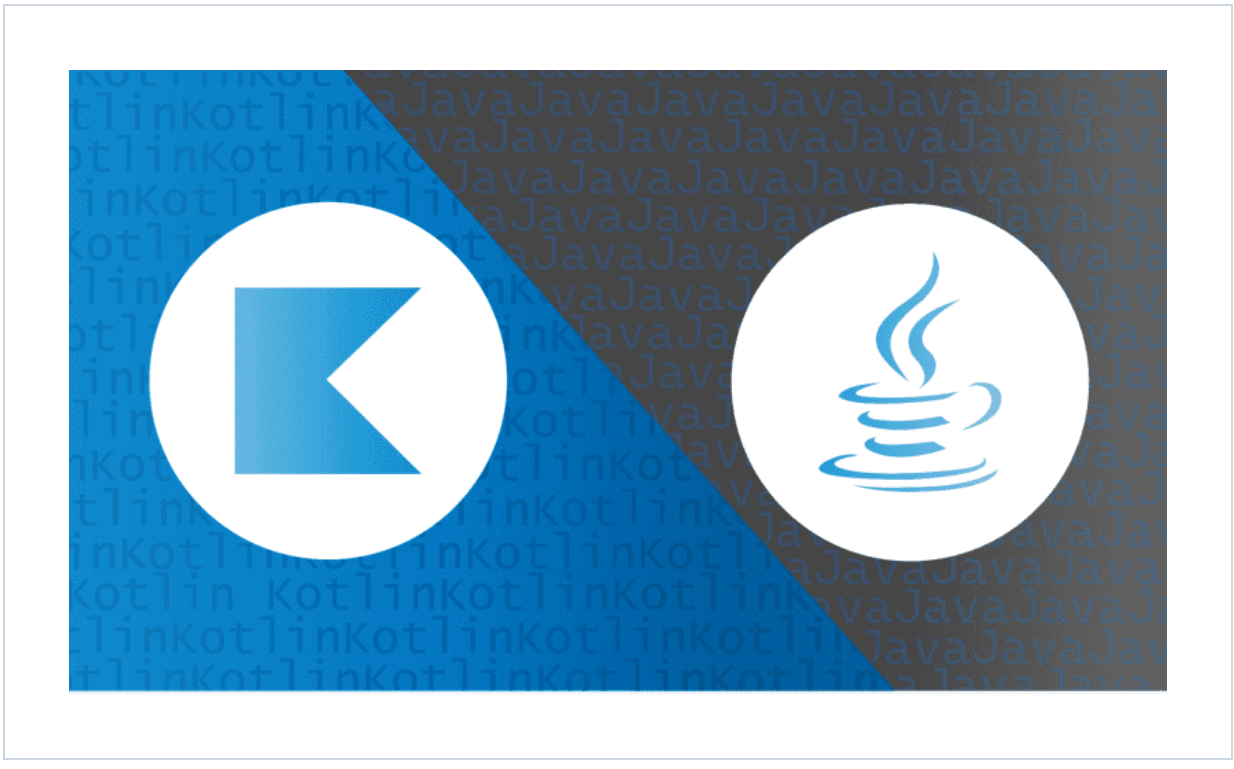
> *Como aprender melhor? Com Diogo Pires | #HipstersPontoTube*
>
> https://www.youtube.com/watch?v=uOagLB3uyeU

Confira o vídeo “Começando no Android com Kotlin” com o instrutor Alex Felipe **desenvolvendo uma aplicação Android do zero** no Hipsters.Talks #22. Se você está começando com Android, esse vídeo é especialmente para você! Aprenda Kotlin e comece a desenvolver suas próprias aplicações.

> *Começando no Android com Kotlin - Hipsters.Talks #22*

https://www.youtube.com/watch?v=k2jWY6jlZao

Kotlin vs Java: qual usar?



Java é uma das linguagens de programação mais populares do mundo e é usada em uma ampla variedade de projetos, dessa forma, possui uma comunidade

erros de referência nula durante a execução do código.

A **escolha de uma das linguagens depende de diversos fatores, necessidades do projeto e preferências da equipe de desenvolvimento**. Tanto Java quanto Kotlin oferecem vantagens e desvantagens, e por isso é necessário avaliar cuidadosamente cada fator.

Flutter vs Kotlin: qual a diferença?

Primeiro, precisamos entender que Kotlin e Flutter são duas tecnologias diferentes para o desenvolvimento de aplicativos.

Kotlin é uma linguagem de programação, assim como Java, Dart, Swift etc. Como vimos, é uma linguagem compatível com Java, então, as pessoas desenvolvedoras podem usar bibliotecas existentes em Java para seus projetos. Além disso, Kotlin pode ser usado para desenvolver aplicativos mobile.

Flutter é um framework, assim como Android, React Native etc. Esse framework específico para desenvolvimento de aplicativos mobile que é escrito na linguagem de programação Dart.

Ambas as tecnologias são importantes e muito usadas para o desenvolvimento de aplicativos móveis. Se sua dúvida for sobre qual deve ser usada, uma ferramenta **híbrida ou nativa**, saiba que isso depende sempre do contexto e da necessidade do projeto.

Por exemplo, se o objetivo do projeto é **entregar um aplicativo para diferentes plataformas com apenas uma base de código**, o **Flutter** tende a ser uma ferramenta mais interessante.

Caso a ideia seja garantir que o **uso de recursos nativos** de uma plataforma funcione como esperado ou **acessar as novidades de uma plataforma e ter mais garantia quanto ao desempenho do aplicativo**, optar pelo **nativo** tende a ser a melhor opção.

Há também uma opção do Kotlin quanto à parte de multiplataforma, o famoso KMM ou Kotlin Multiplatform. A ideia é ter um código-base focado em regras de negócio compartilháveis em qualquer projeto de diferentes plataformas e, então, cada plataforma utiliza a implementação que deseja para reutilizar o código de regras de negócio e apresentar a tela com os frameworks que desejar.

Confira o vídeo “Desmistificando Mobile - Por onde eu começo? #01”, que **simplifica com exemplos práticos vários assuntos do mundo dos aplicativos de celular**. No primeiro episódio, os instrutores da escola de mobile da Alura ajudam a responder a pergunta: por onde começar a estudar sobre o mundo dos

https://www.youtube.com/watch?v=2Z6T8VN_Pro

Acesse também o Alura+ sobre "Desenvolvimento Android", onde o Paulo Silveira e o Felipe Torres abordam os **fundamentos essenciais para começar a desenvolver**, destacando as principais ferramentas utilizadas no desenvolvimento mobile.

> *Desenvolvimento Android #HipstersPontoTube*

https://www.youtube.com/watch?v=fWscDFHKgw8

Kotlin é orientada a objetos ou funcional?

Por se tratar de uma linguagem derivada do Java, o Kotlin se assemelha muito ao seu modelo em diversos aspectos, como ser uma linguagem orientada a objetos. Assim como em Java, no Kotlin existe uma **superclasse** `Any` que fornece suas características a todas as outras classes em seu ecossistema, semelhante à classe `Object` do Java. Além disso, a linguagem possui **herança de interfaces** e permite a criação de aplicações com o paradigma orientado a objetos.

No entanto, **o Kotlin foi idealizado para ser uma alternativa mais moderna e concisa do Java** e, por isso, apresenta recursos que são novidades ou aperfeiçoamentos para que a linguagem se destaque em relação à sua inspiração.

Um desses recursos é a abordagem mais voltada à programação funcional, com funções de *Callback*, *Higher-order functions* e outras funcionalidades conhecidas no contexto do paradigma funcional.

Sendo assim, **o Kotlin é uma linguagem multiparadigma que suporta recursos para lidar tanto com a orientação a objetos quanto com o paradigma funcional**

Como é uma das linguagens derivadas do Java, o Kotlin tem como sua principal característica rodar no ecossistema Java: seu código compila para bytecode, que é interpretado pela *Java Virtual Machine*, assim como acontece em sua linguagem-mãe. Essa propriedade possibilita que o Kotlin aproveite código escrito em Java em suas aplicações também, ou seja, existe **interoperabilidade** com as duas linguagens.

Sendo assim, o Kotlin pode utilizar bibliotecas escritas em sua linguagem modelo e reaproveitá-las sem a necessidade de escrever o código do zero, o que contribui, inclusive, para que a linguagem seja tão robusta quanto o Java.

Sintaxe

Um dos maiores destaques do Kotlin é sua **sintaxe enxuta e de simples compreensão**, que facilita a vida das pessoas que programam. Se em Java até para escrever um "Olá, mundo!" são necessárias muitas linhas de código, em Kotlin a sintaxe é bem mais simples, como é possível perceber abaixo:

```
fun main(){
    println("Olá, mundo!")
}
```

Além disso, o Kotlin adotou a **inferência de variáveis** como padrão para declaração de variáveis, ou seja, não é necessário explicitar o tipo da variável que está sendo criada para que o compilador entenda, ele se encarrega de entender de qual tipo se trata! Sendo assim, basta usar a palavra-chave `var` para declarar uma variável, como abaixo:

No entanto, é muito importante que ao declarar variáveis seja atribuído um valor a elas para que a inferência seja feita, caso contrário ocorrerão erros de compilação. Outro ponto importante é que diferente do que ocorre em linguagens como Javascript e Python, **no Kotlin não existe tipagem dinâmica**, ou seja, quando atribuímos um valor inicial a uma variável, esta será para sempre do tipo informado, não é possível alterar o tipo de valor contido nela.

Outra característica interessante sobre a linguagem é que ela extinguiu a necessidade de ponto e vírgula ao final de cada instrução, como pôde ser percebido nos códigos de exemplo acima.

Tipos básicos

Em Kotlin, assim como no Java, temos tipos primitivos para trabalhar com textos, valores numéricos e lógicos:

Categoria	Tipo
Numérico	Int, Double, Float
Lógico	Boolean
Texto	String, Char

Além disso, também existem tipos complexos, como as classes, enums e interfaces, que são criadas por pessoas desenvolvedoras para representar o mundo real e são utilizadas na **Orientação a Objetos**.

Funções

maneira procedural, como são mais conhecidas, e sua sintaxe é bem mais enxuta do que em sua linguagem-mãe. Por exemplo:

```
fun soma(a: Int, b: Int): Int {
    return a + b;
}
```

A palavra-chave `fun` define que se trata de uma função, que recebe o nome *soma* seguido dos parênteses, que são onde ficam os parâmetros recebidos, explicitando o nome das variáveis de escopo e seu tipo. Além disso, como está sendo retornado o valor da soma entre dois números inteiros, é necessário também sinalizar que a função devolve um valor do tipo *Int*. Caso não fosse retornado nenhum valor, essa parte da função não seria necessária, bem diferente do Java, onde seria necessário utilizar a palavra-chave `void` para indicar que a função não tem retorno.

Mas a grande mágica do paradigma funcional acontece justamente quando as funções são o centro do desenvolvimento, e elementos como composição de funções, *High-Order functions* e *lambdas* vem à tona. A **maneira com a qual o Kotlin lida com as funções é bem diferente do Java**, e se **assemelha** muito mais ao **Javascript**: funções são tratadas como parâmetros e podem ser atribuídas a variáveis.

Como exemplo, é possível reescrever a função `soma()`, mostrada anteriormente, utilizando lambdas e atribuição à uma variável de função, confira:

```
fun main() {
    var soma: ((Int, Int) -> Int) = {a, b -> a + b}
}
```

meio estranho de código, mas que ilustra o que acontece dentro da função!

Ao se atentar ao trecho `((Int, Int) -> Int)`, deve ser entendido: *"A expressão recebe como parâmetros dois inteiros e o resultado final será um inteiro"*. Neste caso, o indicador do que será retornado é o que vier após a flecha da expressão `(->)`.

Funções é um tema bastante extenso e com diversos recursos devido ao paradigma funcional. Além dos já citados, existem outros como os *Named Parameters*, que são uma maneira de garantir que os argumentos corretos estão sendo passados na função, dentre outros recursos. Para se aprofundar mais em funções e programação funcional em Kotlin, é recomendada a leitura da documentação da linguagem sobre funções, que trata de maneira mais aprofundada sobre o assunto!

Null Safety

Dentro do dia a dia das pessoas que desenvolvem em Java, existe um grande fantasma que as acompanha: a *NullPointerException* (NPE), que ocorre quando existe uma chamada para uma referência inexistente, ou seja, nula. Essa característica é bastante controversa entre pessoas desenvolvedoras, pois é um problema que é tratado em execução, e não se pode prever com clareza se ele vai ocorrer durante o desenvolvimento, pois não há auxílio em compilação.

Com isso em mente, as pessoas responsáveis pelo desenvolvimento do Kotlin trouxeram um recurso para minar a existência desse tipo de problema no código, o ***Null Safety***.

Basicamente, ao programar em Kotlin, **não haverá nenhum *NullPointerException* a não ser que essa exceção seja lançada explicitamente** ou **através da interoperabilidade com a linguagem Java, que pode causar esse tipo de exceção**. Mas como o Kotlin lida com esse tipo de problema, já que não há NPE implícita?

Para entender melhor, imagine o seguinte trecho de código:

```
conta?.titular?.getNome()
```

Primeiro, o compilador tenta acessar o valor da variável *conta* e, caso ele exista, tenta acessar o valor de *titular*, para então chamar o método `getNome()`. No entanto, a grande sacada é que se a variável `conta` ou `titular` tiverem valor nulo, o código retornará `null` e não tentará executar a função `getNome()`, evitando assim uma exceção de referência nula. Dessa forma, é possível evitar a necessidade de tratar esse tipo de problema no código.

Outro recurso similar é utilizado quando o objetivo é armazenar valores em variáveis, também utilizando o operador `?`, conhecido como *Nullable receiver*, ou *recebedor nulo*. Sua sintaxe é a seguinte:

```
val conta: Conta? = null
```

Ou seja, no momento da declaração de uma variável é necessário especificar seu tipo, seguido do operador `?`, que vai indicar ao compilador que o valor contido nessa variável pode ser um objeto do tipo *Conta*, ou um valor `null`. **Quando não há o uso do operador `?`, objetos não podem assumir o valor nulo**.

Além disso, o *Nullable Receiver* realiza alguns tratamentos interessantes para evitar a ocorrência da temida NPE. Um desses tratamentos, bastante útil, ocorre quando precisamos invocar o método `toString()` de um objeto qualquer. Em vez de tentar chamar o método a partir da referência nula, o compilador reconhece que se trata de um valor inexistente e retorna a String `"null"`, o que é bastante útil durante a depuração do código.

Kotlin e Desenvolvimento Web

Devido à sua proposta inicial de ser uma substituta natural do Java, o Kotlin procurou fortalecer-se justamente onde sua linguagem-mãe era mais forte: no desenvolvimento web, de maneira robusta e escalável. Sendo assim, buscou alternativas que pudessem ser mais interessantes para atrair a adesão da linguagem.

Dentre os recursos da linguagem, destaca-se a interoperabilidade com o Java, o que é especialmente relevante para o desenvolvimento back-end. Essa interoperabilidade permite que o Kotlin seja utilizado em conjunto com as diversas ferramentas, bibliotecas e frameworks já existentes no ecossistema Java, o que já traria condições da linguagem ser adotada comercialmente. E foi o que aconteceu! Devido à capacidade do Kotlin de interagir com o ecossistema Java, a adaptação dos frameworks existentes foi facilitada e, com isso, grandes frameworks utilizados no Java passaram a ter implementações para o Kotlin também, como o ***Spring***, ***Quarkus*** e ***Javalin***.

Ainda assim, migrar um sistema inteiro de linguagem pode ser uma tarefa desafiadora, devido aos possíveis conflitos que possam surgir caso a migração seja feita seja feita em partes. Além disso, seria um grande desafio reescrever um sistema do zero em uma nova linguagem.

No entanto, mais uma vez, o Kotlin se beneficia de sua arquitetura baseada na JVM para facilitar o processo de migração de software. Uma vez que código Java e código Kotlin se conversam, é possível fazer mudanças graduais de uma linguagem para a outra em aplicações grandes: é viável escrever novas funcionalidades de um software em Kotlin e integrá-las em um sistema em produção Java, sem grandes dificuldades, e migrar progressivamente o código legado para a nova linguagem.

Sendo assim, **é possível realizar a migração do Java para o Kotlin de maneira sutil, sem que exista a necessidade de lidar com grandes conflitos e inconsistências entre as linguagens**.

Se você quiser conhecer um pouco mais sobre desenvolvimento web em Kotlin, recomendamos a Formação Kotlin e Spring Boot, um dos principais frameworks

relacionar código Kotlin com JavaScript. Diferente do que acontece ao se relacionar com o Java, em que o Kotlin usa a *Java Virtual Machine* para interpretar bytecode e incorporá-lo em aplicações, **ao se relacionar com o JavaScript, o Kotlin adota uma abordagem diferente, conhecida como "transpilação"**.

Basicamente, o Kotlin adicionou ao seu kit de ferramentas um compilador Kotlin para JavaScript, que gera arquivos `.js` e até mesmo arquivos TypeScript `.ts` a partir de código Kotlin nativo. Essa capacidade permite criar aplicações híbridas entre as linguagens, aproveitando o melhor de ambas as plataformas, em diferentes tipos de aplicações. Por exemplo, é possível criar aplicações front-end, incluindo aplicações React, a partir de código Kotlin, bem como aplicações back-end, utilizando a interação da linguagem com a plataforma Node.js.

Além disso, é possível aproveitar a interação das duas linguagens para escrever aplicações multiplataforma, pois o código escrito em Kotlin pode ser reutilizado no Kotlin/JS e até mesmo em aplicativos Mobile.

Outra possibilidade é a criação de bibliotecas que podem ser consumidas via JavaScript e TypeScript para criar aplicações híbridas e ter a possibilidade de, a partir de um código escrito em Kotlin, poder utilizar suas funcionalidades tanto em aplicações Kotlin puras, bem como em aplicações JavaScript ou TypeScript.

Para se aprofundar ainda mais, você pode conferir a documentação oficial de Kotlin/JS, que trata de maneira mais detalhada sobre o assunto!

Kotlin Native

Já ficou entendido que o Kotlin se beneficia da robustez do ecossistema Java, principalmente da JVM, para compilar e rodar seus programas, aproveitando todas as vantagens que esse modelo oferece. No entanto, também enfrenta algumas desvantagens, como a performance. Embora a JVM seja responsável por gerenciar os recursos do computador, nem sempre o faz de maneira

compilar código para binário nativo, que vai ser lido e interpretado diretamente pelo processador da máquina e não mais por uma camada de máquina virtual.

Com isso em mente, as pessoas responsáveis pelo desenvolvimento do Kotlin criaram o ***Kotlin Native***, que nada mais é que uma variação da linguagem que, **ao invés de compilar em bytecode** que seria interpretado pela JVM, **compila direto em binário nativo do sistema operacional**, o que permite, por exemplo, melhor otimização da utilização do hardware do computador.

No entanto, um dos recursos mais interessantes é a **possibilidade de desenvolvimento para sistemas embarcados**, como **celulares e televisões**, onde uma máquina virtual pode ser muito custosa devido às limitações de hardware desse tipo de dispositivo. Assim, é possível desenvolver para Android e iOS nativamente, além dos principais sistemas operacionais desktop: MacOS, Windows e Linux.

Além disso, o Kotlin Native dá suporte à interoperabilidade com algumas linguagens que são suportadas dentro dessas plataformas alvo, como a linguagem C e o Swift, dentre outras. Isso permite que, a partir de um código Kotlin, sejam geradas bibliotecas para as outras linguagens, bem como que o Kotlin utilize bibliotecas geradas a partir de código gerado nelas.

Kotlin para Data Science

Com o acesso cada vez maior à informação, existe uma grande quantidade de tráfego de dados circulando pela internet, que, por si só, podem não representar nada concreto. Por esse motivo, a área de Ciência de Dados tem recebido bastante destaque, uma vez que, uma vez que um bom **tratamento desses dados** facilita processos como a **tomada de decisão** por parte de gestores.

Como uma linguagem moderna, o Kotlin também entrou para a área da Ciência de Dados, oferecendo ferramentas para a **criação de pipelines**, **modelos de machine learning** e até mesmo **inteligência artificial**. Se aproveitando dos benefícios da JVM e dos seus recursos, como a *Null Safety* e a facilidade de

Para ir mais a fundo, é recomendável a leitura da documentação oficial de Kotlin para Data Science, que lista, inclusive, muitas bibliotecas interessantes para iniciar!

Aprenda mais sobre Kotlin gratuitamente

Acesse gratuitamente as primeiras aulas da Formação Linguagem Kotlin, feita pela Escola de Programação da Alura e continue aprendendo sobre temas como:

1. Kotlin: orientação a objetos
2. Kotlin: herança, polimorfismo e Interface
3. Kotlin: recursos da linguagem com pacotes e composição
4. Kotlin: lidando com exceptions e referências nulas
5. Kotlin: desenvolva com coleções, arrays e listas
6. Kotlin Collections: Set e Map
7. Kotlin: recursos do paradigma funcional

Acesse gratuitamente as primeiras aulas da Formação Desenvolva seu primeiro app Android com Kotlin, feita pela Escola de Mobile da Alura e continue aprendendo sobre temas como:

1. Android com Kotlin: criando um app
2. Android com Kotlin: personalize o seu app
3. Android com Kotlin: persistência de dados com o Room
4. Android com Kotlin: Migrations e relacionamento com o Room
5. Android com Kotlin: comunicação com Web API

Conclusão

Vimos que Kotlin é uma linguagem de programação que tem ganhado destaque pela sua **facilidade de uso, segurança e flexibilidade**. Ao ser desenvolvida para ser uma linguagem compatível com a plataforma Java, Kotlin se tornou uma opção viável para quem busca uma **alternativa moderna e mais produtiva** para o desenvolvimento de software.

Com o suporte da JetBrains, empresa responsável pelo desenvolvimento do Kotlin, e de uma comunidade em crescimento, a linguagem tem se estabelecido como uma escolha cada vez mais popular em diversos setores da indústria de software. Com uma sintaxe limpa e intuitiva, recursos modernos e grande eficiência, Kotlin pode ser uma boa escolha para desenvolvedores e projetos que se identificam com a linguagem.

Quer se aprofundar ainda mais na tecnologia Kotlin? Deixamos aqui algumas referências que serão de grande ajuda:

- Tech Guide Android: Kotlin - Fundamentos, um guia de referência para estudar os fundamentos do Kotlin no universo Android
- Tech Guide Android: Kotlin - Assíncrono, um guia de referência para estudar assincronismo e Kotlin no universo Android
- Tech Guide Android: Kotlin - Comunicação com APIs, um guia de referência para estudar Kotlin e comunicação com APIs no universo Android
- Tech Guide Android: Kotlin - Injeção de Dependências, um guia de referência para estudar injeção de dependências e Kotlin no universo Android
- Artigo
  Como preparar o ambiente e escrever seu primeiro código com Kotlin (Java)

Veja outros artigos sobre
Programação