Linguagem Kotlin Desenvolvimento Android

Apresentação: Prof. Carlos Alberto

A LINGUAGEM KOTLIN

- Podemos definir Kotlin como uma linguagem de programação pragmática que combina os paradigmas de Orientação a Objetos e Programação Funcional com foco em interoperabilidade, segurança, clareza e suporte a ferramentas - essa é a definição que seus desenvolvedores deram a ela.
- Já a equipe do Android a define como uma linguagem expressiva, concisa e poderosa. A linguagem foi criada pela JetBrains, a mesma empresa criadora do Android Studio, e veio como uma alternativa ao Java para desenvolvimento Android.
- Temos de ter clareza que a intenção do Kotlin não é substituir o Java, mas ser uma linguagem moderna de alta produtividade.

UM BREVE HISTÓ

- Na verdade, o Kotlin não nasceu ontem, o primeiro anúncio da linguagem veio lá em 2011 durante o JVM Language Summit, onde que a JetBrains revelou que estava trabalhando havia quase um ano no projeto Kotlin. A ideia era criar uma nova linguagem estaticamente **tipada** para a JVM. O projeto Kotlin juntava anos de experiência na criação de ferramentas para diversas outras linguagens e tinha como objetivo criar uma linguagem que fosse produtiva e, ao mesmo tempo, de simples aprendizado.

Uma **linguagem** é definida como **estaticamente tipada** quando a pessoa que está programando precisa informar explicitamente o tipo de cada dado utilizado no sistema: variáveis, parâmetros de funções, valores de retorno, etc.

UM BREVE HISTÓRICO

- Em 2016, a versão 1.0 foi lançada, considerada a primeira versão estável da linguagem.
- O Kotlin não é uma linguagem exclusiva do Android, de acordo com seus desenvolvedores, ela é para uso geral.
- Ela é uma linguagem que roda na JVM, ou seja, onde funciona Java, o Kotlin também funciona!
- Em 2017, durante o evento Google I/O, Kotlin foi anunciado oficialmente a linguagem para desenvolvimento Android.
- De acordo com a equipe do Android, a escolha do Kotlin se deve ao fato de ela ser uma ótima linguagem e tornar a criação de aplicativos mais produtiva e divertida.
- Outro ponto chave é que todo ecossistema Android é compatível com Kotlin, então tudo o que já foi desenvolvido, todas as bibliotecas do Android funcionam sem a necessidade de serem reescritas.

INTEROPERABILIDADE

- A interoperabilidade a capacidade de uma linguagem se comunicar com outra.
- O Kotlin tem total interoperabilidade com Java, isso significa que podemos utilizar códigos que foram escritos em Java no nosso código em Kotlin.
- Podemos aproveitar qualquer biblioteca ou classe escrita em Java nos nossos aplicativos feitos em Kotlin.

Exemplo de interoperabilidade

- Vamos a um exemplo prático, vamos supor que você criou uma classe em Java chamada Calculadora e, dentro dela, você criou um método chamado somar , que faz a soma de dois números inteiros. O código seria o seguinte:

```
public class Calculadora{
    public int somar(int a, int b){
        return a+b;
    }
}
```

- classe no seu código Kotlin:

```
val calc = Calculadora()
val resultado = calc.somar(2,2)
println(resultado)
```

UMA LINGUAGEM CONCISA

- Uma linguagem concisa, basicamente estamos falando que precisamos escrever menos código para fazer a mesma coisa que uma outra linguagem faria com mais código.
- A linguagem que utilizaremos como parâmetro será o Java.
- Vamos a mais um exemplo prático, o código Android a seguir define que o click de um botão que tenha um identificador bt_login chamará um método de nome efetuarLogin :

Exemplo 01

- Aplicação Android:

```
Button bt_login = (Button) findViewById(R.id.bt_login);
bt_login.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        efetuarLogin();
    }
});
```

- Aplicação em Kotlin:

```
bt_login.setOnClickListener{
            efetuarLogin()
}
```

```
fun main() {
    val kotlin = "🙂"
    println(kotlin)
}
```

Exemplo 02

- Vamos supor que você precise criar uma classe de modelo que vai representar um cliente da sua aplicação. Para simplificar, esse cliente terá somente um e-mail e um nome de usuário. Em Java, sua classe ficaria assim:

```java
public class Cliente {

    private String email;
    private String nomeUsuario;

    public Cliente(String email, String nomeUsuario) {
        this.email = email;
        this.nomeUsuario = nomeUsuario;
    }

    public String getNomeUsuario() {
        return nomeUsuario;
    }

    public void setNomeUsuario(String nomeUsuario) {
        this.nomeUsuario = nomeUsuario;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}
```

- Em Kotlin:

```kotlin
data class Cliente(var email:String, var nomeUsuario: String)
```

LIVRE-SE DAS REFERÊNCIAS NULAS

- Um erro de referência nula ou o famoso **NullPointerException,**
- Um erro de referência nula no nosso programa significa que estamos tentando acessar um objeto que ainda não possui um espaço alocado na memória RAM da máquina.
- exemplo em Java: **String nomeUsuario = null;**
- O código declara uma variável nomeUsuario porém atribui a ela um valor nulo. Isso significa que ainda não existe um espaço na memória do computador para essa variável, e se tentarmos acessála teríamos um erro de referência nula, **NullPointerException**: **String nomeUsuario = null;**
- // Essa linha de código gera um erro de NullPointerException Log.d("TAG", nomeUsuario);

REFERÊNCIAS NULAS

- Programando em Kotlin, você raramente verá esse tipo de erro!
- Possui proteção contra nulos (null safety). Isso diminui drasticamente os erros do tipo NullPointerException e ela resolve isso de uma forma muito prática.
- Em Kotlin, por padrão nenhuma variável ou objeto pode ter um valor nulo;
- Exemplo: var nomeUsuario :String = null
- Esse código simplesmente não compila porque a variável nomeUsuario não pode ter um valor igual a null . O seguinte erro será gerado: **"Null can not be a value of a non-null type String“; "Valores nulos não podem ser definidos ao tipo não nulo String".**

REFERÊNCIAS NULAS

- Para resolver, devemos inicializar a variável com um valor diferente de nulo: var nomeUsuario :String = "“
- Agora sim o código compila, pois inicializamos a variável com uma string vazia, o que definitivamente não é nulo.
- Se o desenvolvedor desejar, ele pode deixar explícito que a variável pode receber um valor nulo. Para isso, basta adicionar o operador **?** após o tipo da variável, assim: var nomeUsuario : String? = null. Dessa forma, a variável pode receber um valor nulo se assim o desenvolvedor desejar. No entanto, para utilizar essa variável compilador nos obriga a fazer uma validação antes:

REFERÊNCIAS NULAS

- var nomeUsuario :String? = null

  if (nomeUsuario !=null){

  println(nomeUsuario.length) }

- Dentro do if , o compilador assegura que aquela variável terá um valor e não ocasionará um erro de referência nula. Podemos ainda utilizar operador ? , chamado de safe call, que simplesmente ignora a chamada se a variável for nula:
  - var nomeUsuario :String? = null

    println(nomeUsuario?.length)

Conhecendo play Kotlin

- Ao acessar o link https://play.kotlinlang.org você verá a tela inicial do play Kotlin:

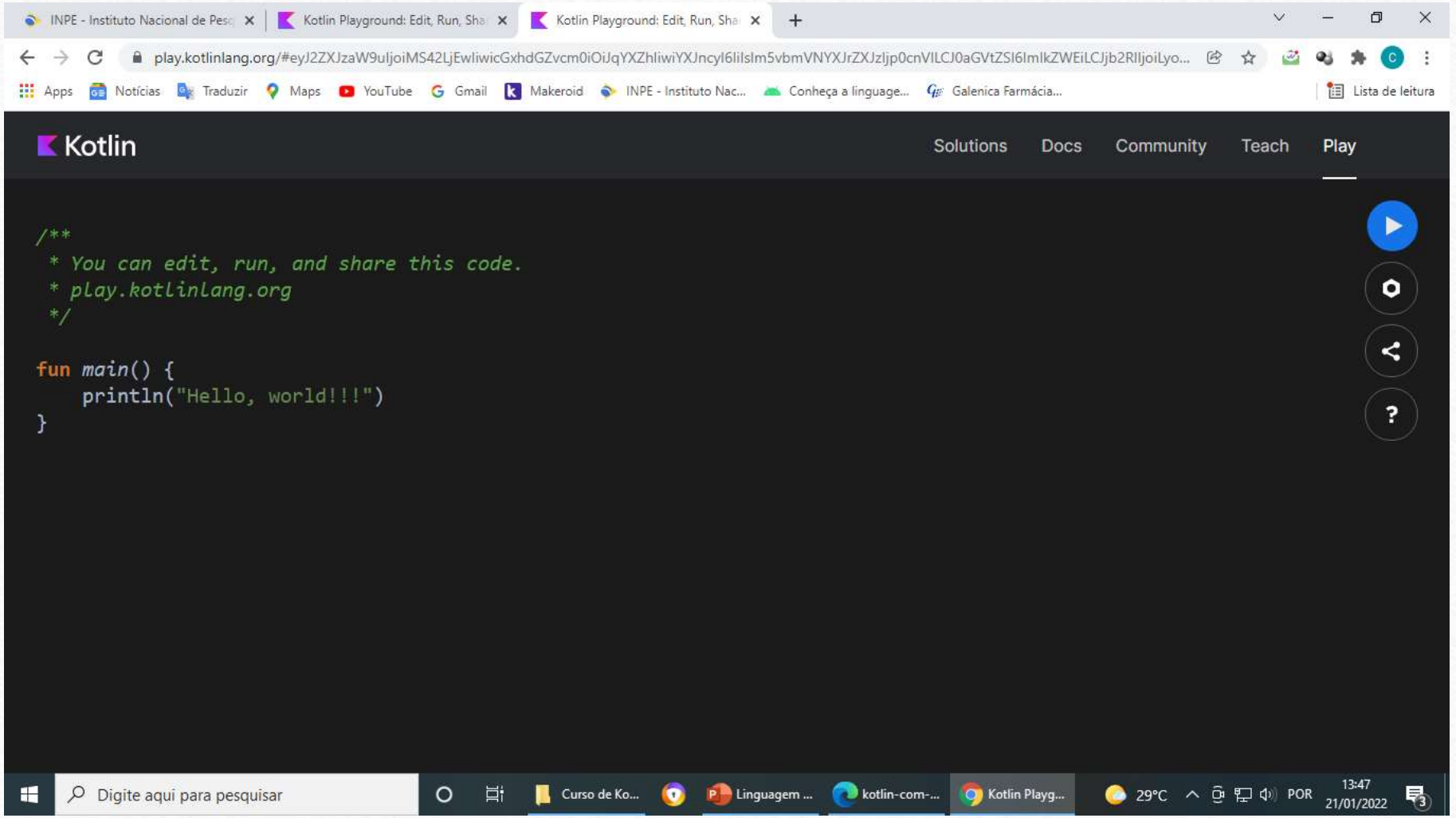
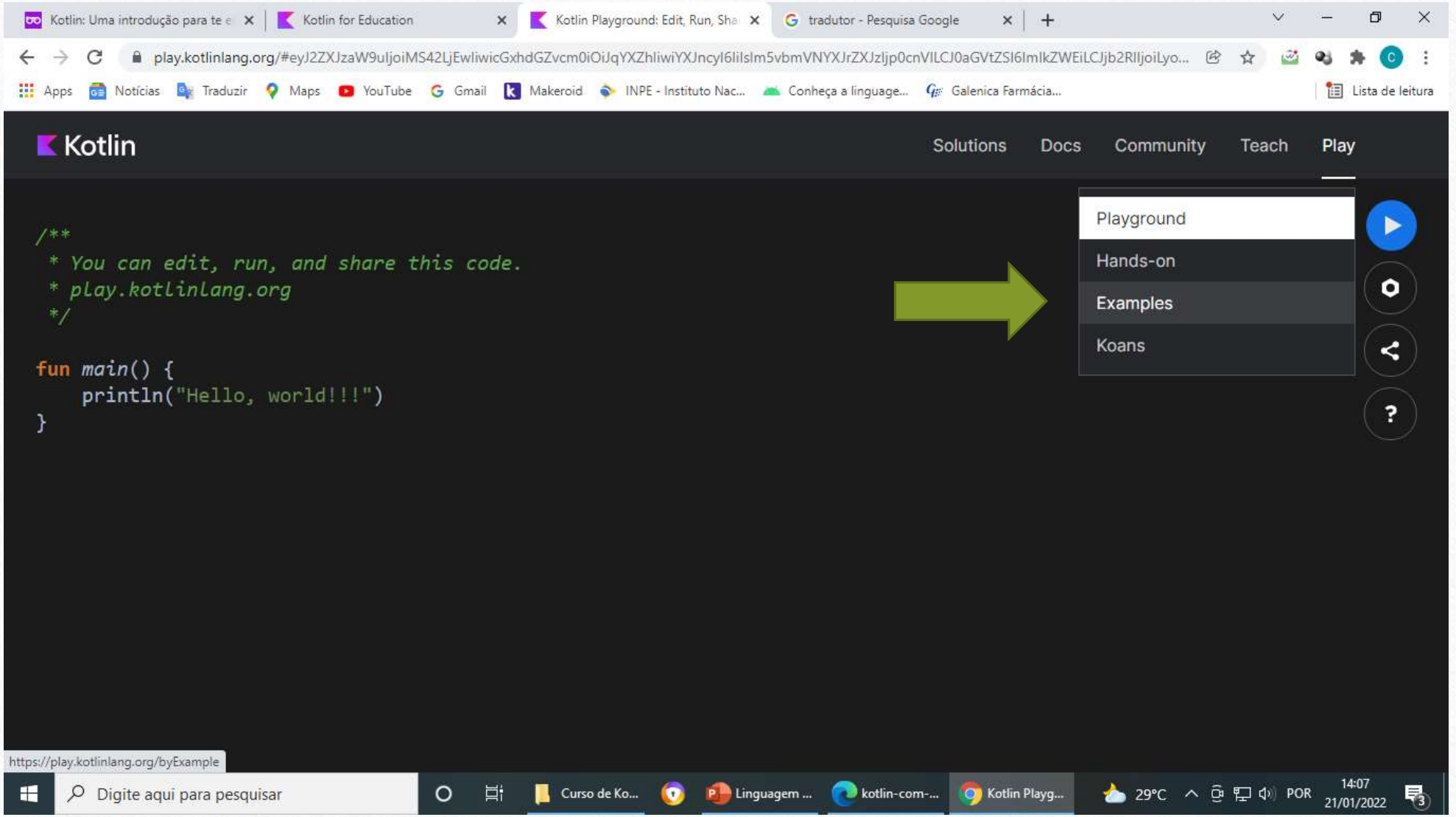


Figura 2.1: play.kotlinlang.org

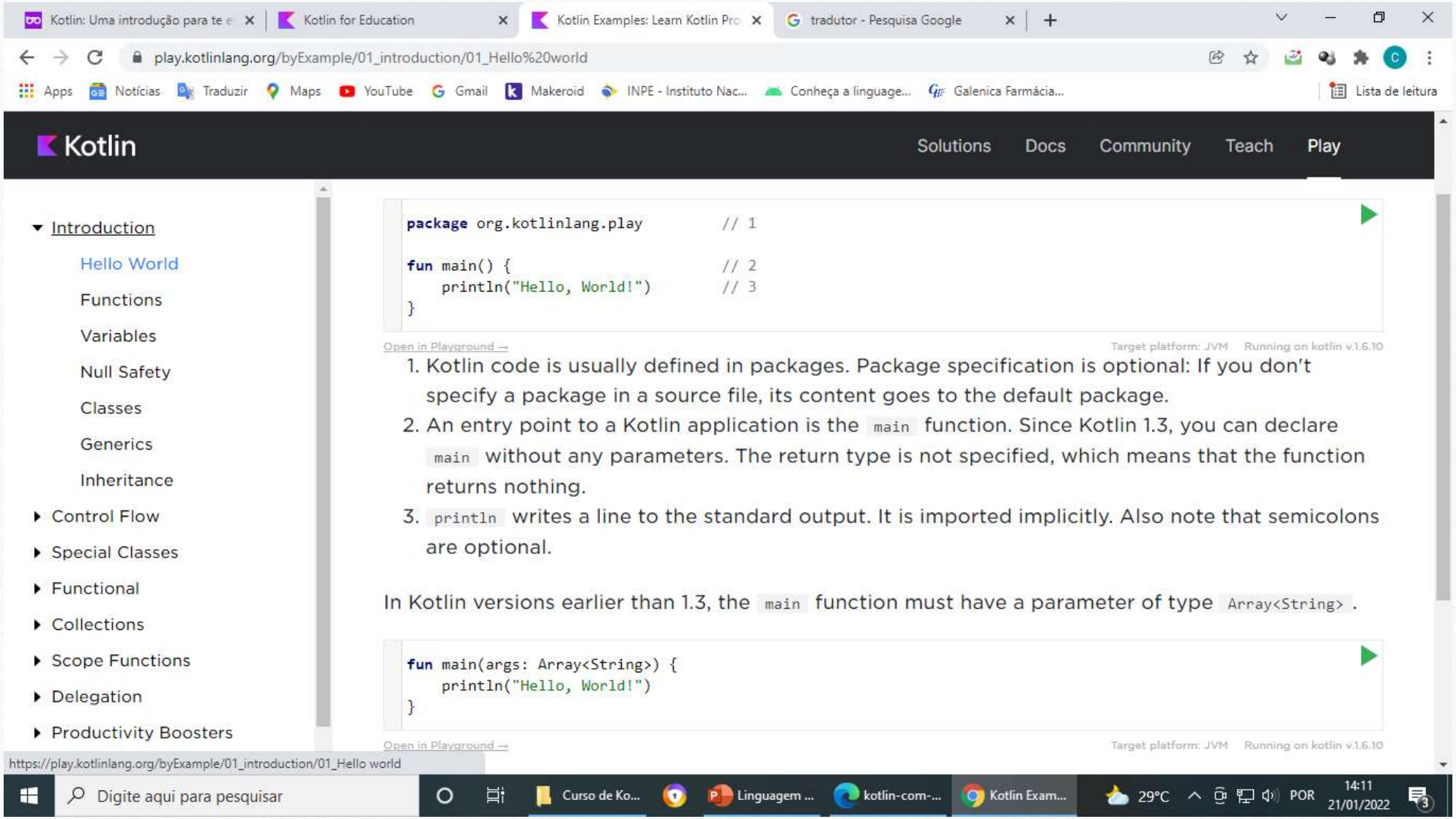
Conhecendo play Kotlin

- Vamos abordar algumas características funcionais para nossa utilização
- 1.Editor de códigos em si. Aqui você escreverá seus códigos em Kotlin.
- 2. Botões para salvar seus códigos. Você só poderá salvar algum código se estiver logado na plataforma.
- 3. Caixa de seleção para escolher se seu código será executado em JVM, JavaScript, JavaScript(Canvas) ou JUnit.
- 4. Botão para executar o código escrito no editor.


Kotlin: Uma introdução para te
Kotlin for Education
Kotlin Playground: Edit, Run, Sha
tradutor - Pesquisa Google
play.kotlinlang.org/#eyJ2ZXJzaW9uIjoiMS42LjEwIiwicGxhdGZvcm0iOiJqYXZhIiwiYXJncyI6IiIsIm5vbmVNYXJrZXJzIjp0cnVlLCJ0aGVtZSI6ImlkZWEiLCJjb2RlIjoiLyo...
Apps
Notícias
Traduzir
Maps
YouTube
Gmail
Makeroid
INPE - Instituto Nac...
Conheça a linguage...
Galenica Farmácia...
Lista de leitura
Kotlin
Solutions
Docs
Community
Teach
Play
/**
* You can edit, run, and share this code.
* play.kotlinlang.org
*/
fun main() {
println("Hello, world!!!")
}
Playground
Hands-on
Examples
Koans
https://play.kotlinlang.org/byExample
Digite aqui para pesquisar
Curso de Ko...
Linguagem ...
kotlin-com-...
Kotlin Playg...
29°C
POR
14:07
21/01/2022

Conhecendo play Kotlin

- Playkotlin, temos uma estrutura de pastas com diversos exemplos de código. Você pode clicar em qualquer arquivo, que seu código será mostrado no editor. lado esquerdo, temos uma estrutura de pastas com diversos exemplos de código. Você pode clicar em qualquer arquivo, que seu código será mostrado no editor.


Kotlin: Uma introdução para te e
Kotlin for Education
Kotlin Examples: Learn Kotlin Pro
tradutor - Pesquisa Google
play.kotlinlang.org/byExample/01_introduction/01_Hello%20world
Apps
Notícias
Traduzir
Maps
YouTube
Gmail
Makeroid
INPE - Instituto Nac...
Conheça a linguage...
Galenica Farmácia...
Lista de leitura
Kotlin
Solutions
Docs
Community
Teach
Play
Introduction
Hello World
Functions
Variables
Null Safety
Classes
Generics
Inheritance
Control Flow
Special Classes
Functional
Collections
Scope Functions
Delegation
Productivity Boosters
package org.kotlinlang.play // 1
fun main() { // 2
println("Hello, World!") // 3
}
Open in Playground →
Target platform: JVM Running on kotlin v.1.6.10
1. Kotlin code is usually defined in packages. Package specification is optional: If you don't specify a package in a source file, its content goes to the default package.
2. An entry point to a Kotlin application is the main function. Since Kotlin 1.3, you can declare main without any parameters. The return type is not specified, which means that the function returns nothing.
3. println writes a line to the standard output. It is imported implicitly. Also note that semicolons are optional.
In Kotlin versions earlier than 1.3, the main function must have a parameter of type Array<String> .
fun main(args: Array<String>) {
println("Hello, World!")
}
Open in Playground →
Target platform: JVM Running on kotlin v.1.6.10
https://play.kotlinlang.org/byExample/01_introduction/01_Hello world
Digite aqui para pesquisar
Curso de Ko...
Linguagem ...
kotlin-com-...
Kotlin Exam...
29°C
POR
14:11
21/01/2022

Conhecendo play Kotlin

- Vamos entender um pouco melhor esse código.

```
fun main(args: Array<String>) {
    println("Hello, World!")
}
```

```
Hello, World!
```

- Essa é a função main do Kotlin, ela recebe um Array de String chamado args , que são os argumentos que podem ser passados para o programa. A função main é o ponto de partida de todo programa feito em Kotlin. É claro que podemos criar outros arquivos, classes, funções e métodos, mas o ponto de partida de um programa em Kotlin sempre será a função main , por isso o comando para exibir a mensagem de "Hello, world!" : println("Hello, world!") está dentro da função main .

Conhecendo play Kotlin

- Vamos adicionar mais uma linha no nosso código, observe que a segunda linha estou finalizando com ponto e vírgula, kotlin aceita os dois formatos.
- Observe os dos códigos, o que podemos concluir?

```
fun main(args: Array<String>) {
    println("Hello, World!")
    println("Meu primeiro programa em Kotlin!");
}
```

```
Hello, World!
Meu primeiro programa em Kotlin!
```

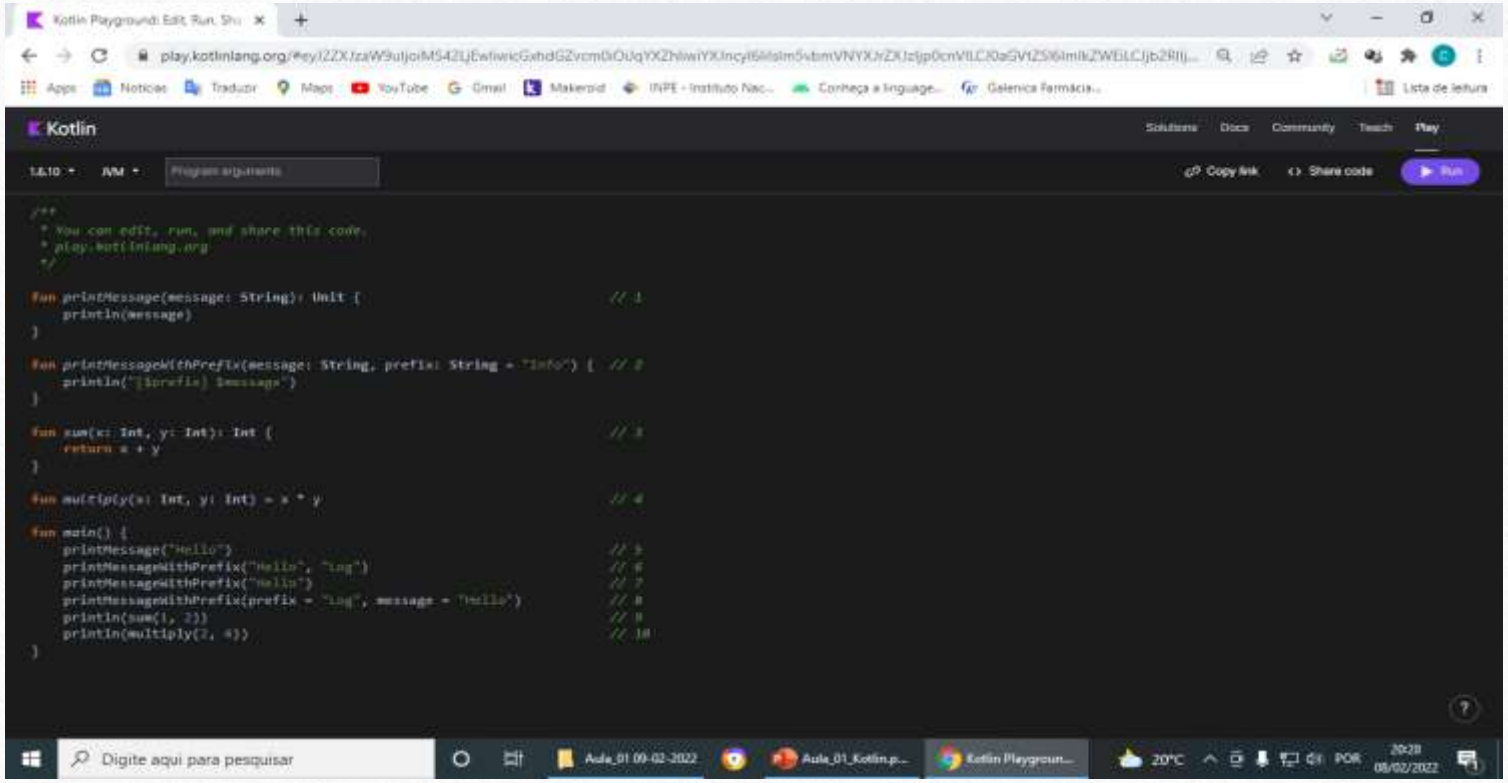
```
fun main(args: Array<String>) {
    print("Hello, World!")
    println("Meu primeiro programa em Kotlin!")
}
```

```
Hello, World!Meu primeiro programa em Kotlin!
```

Conhecendo play Kotlin

- Lembre-se: a função main é o ponto de partida de todo programa em Kotlin, por isso todos exemplos de código dessa apresentação devem estar dentro da função main( ).

Conhecendo play.kotlinlang.org



INSERINDO COMENTÁRIOS NO CÓDIGO

- Comentários são textos inseridos no código, mas que serão ignorados pelo compilador e não vão interferir no funcionamento do programa. São úteis para explicar alguma lógica desenvolvida ou simplesmente o que faz, de fato, algum comando. Para quem está aprendendo, é muito importante comentar o código para depois você mesmo poder entender o que está acontecendo. Podemos comentar um programa em Kotlin de duas maneiras. A primeira é utilizando duas barras e, em seguida, o comentário:
  - //Este é um comentário em Kotlin!
- Esta forma de comentário serve para comentar uma única

INSERINDO COMENTÁRIOS NO CÓDIGO

- A segunda maneira de se comentar um código é usando os delimitadores: /* para marcar o início do comentário e */ para marcar o fim do comentário. Essa maneira é particularmente útil quando o comentário possui mais de uma linha, veja um exemplo:
  - /* Este é um comentário em Kotlin,
  - Com mais de uma linha!
  - */

DEFININDO VARIÁVEIS VAL E VAR

- Em programação, é muito comum o uso de variáveis. Utilizamos variáveis para guardar algum valor que usaremos posteriormente no código. Por exemplo, vamos supor que eu gostaria de criar um programa que efetue a soma de dois números inteiros. Logo de cara eu identifico que preciso desses dois números inteiros, então preciso de pelo menos duas variáveis para guardar cada um. Darei um nome a elas, vamos supor x e y , sendo que x contém um valor e y contém outro valor, então, meu programa fará a operação de x + y . Naturalmente, eu gostaria de saber qual o resultado dessa operação, logo eu precisaria de mais uma variável para guardar o resultado da operação. Poderia ser uma variável r , então a operação ficaria assim:
- r = x + y .

DEFININDO VARIÁVEIS VAL E VAR

- Podemos definir uma variável como um espaço reservado na memória do computador em que eu posso guardar valores e resgatá-los através de um nome. Para criar variáveis em Kotlin, utilizamos a palavra-chave var seguida da definição de seu tipo, assim:
  - var idade:Int = 22
- Esse código indica que a variável idade é do tipo Int e tem um valor inicial de 22 . Em Kotlin, é obrigatório que a variável tenha um valor inicial, porém não é obrigatória a indicação de seu tipo. Por exemplo, o mesmo código poderia ser escrito da seguinte maneira:
  - var idade = 22

DEFININDO VARIÁVEIS VAL E VAR

- Imagine uma situação em que você deve guardar a altura de uma pessoa. Como a altura normalmente não é um número exato, poderíamos utilizar uma variável do tipo Double para guardar uma altura, assim é possível guardar valores quebrados, 1.60 , 1.75, 1.80 etc. Mas imagine que, na definição da variável, você defina o valor inicial de uma pessoa com 2 metros de altura, assim: **altura = 2** . Dessa forma, o compilador assume que a variável altura é do tipo Int porque 2 é um número inteiro! Para não cair nesse tipo de erro, você pode sempre definir o tipo da variável ou quando for definir seu valor, passar um valor Double , assim: **idade = 2.0** . Com isso, o Kotlin entende que a variável idade é do tipo Double e infere o tipo correto.

Os tipos básicos que o Kotlin aceita são:

Tipo	Descrição
`Double`	Tipo numérico para valores de ponto flutuante, com `Double` é possível guardar valores com precisão bem grande pois ele ocupa 64 bits
`Float`	Tipo numérico para valores de ponto flutuante, com a diferença de que tem uma precisão menor que o `Double` pois ele ocupa 32 bits
`Long`	Usado para valores inteiros, a diferença dele para o `Int` é que, como ele ocupa 64 bits, você consegue representar números muito maiores
`Int`	Talvez um dos tipos numéricos mais comuns, é para guardar números inteiros com uma precisão de 32 bits
`Short`	Também guarda valores inteiros, porém com capacidade máxima de metade de um `Int` , pois ocupa somente 16 bits
`Byte`	Também guarda números inteiros, porém seu número máximo é 127 pois ele ocupa somente 8 bits
`String`	É um tipo para guardar valores em texto, uma frase, um nome ou qualquer valor que seja um texto
`Char`	É um tipo para guardar um único caractere, diferente da `String` , em que você pode guardar textos
`Boolean`	É usado para valores booleanos, ou seja, valores que são verdadeiro (`True`) ou falso (`False`)

Variáveis imutáveis(constantes)

- Utilizamos a palavra-chave val quando queremos definir uma variável imutável, ou seja, uma variável que não terá seu valor alterado posteriormente. Há diversos casos em que o uso de uma variável imutável é mais adequado do que de uma variável comum. A principal diferença é que a variável, como seu próprio nome diz, não possui um valor fixo, podendo variar de acordo com necessidade; já uma variável imutável tem seu valor fixo, e uma vez definido seu valor este não será mais alterado.

Variáveis imutáveis(constantes)

- Exemplo: se eu declarasse a variável idade como val e posteriormente tentasse mudar seu valor, o compilador reclamaria e o código não iria compilar:

Esse erro ("Val cannot be reassigned“) → ("Val não pode ter seu valor redefinido“), ou seja, se utilizamos um val não podemos mais alterar seu valor depois que ele foi definido.

Variáveis imutáveis(constantes)

- Mas então, quando usar var e quando usar val ? Bem, por via de regra, defina suas variáveis como val e, caso haja necessidade, mude para var . Isso melhorará o resultado final dos seus códigos em questão de segurança e clareza. Segurança, porque você garante que variáveis imutáveis não terão seu valor trocado por qualquer motivo; e clareza, porque ficam claras as intenções de design utilizadas na construção do código. Vejamos um exemplo a seguir:

Exemplo 01

- Vamos fazer um pequeno exercício e criar um pequeno programa que calcula a área de um retângulo, um cálculo bem simples de ser feito. Para isso, abra o Try Kotlin e apague todo o código existente, pois vamos criar um código novo. Para o cálculo de área de um retângulo, basta multiplicar sua base pela altura, então no nosso programa usaremos primeiro uma variável para guardar o valor da base , depois uma variável para guardar o valor da altura e, por fim, uma variável para guardar o resultado do cálculo:

Exemplo 01(continuação)

```
fun main(args:  Array<String>)  {
        val base     =   3
        val altura   =   7
        val resultado    =   base    *   altura
        println("O resultado é: "+resultado)
}
```

```
O resultado é: 21
```