

Linguagem Kotlin

Desenvolvimento Android

Apresentação: Prof. Carlos Alberto



FUNÇÕES

- Podemos definir uma função como um conjunto de comandos agrupados em um bloco, que recebe um nome e, através deste nome, pode ser chamado em outras partes do código. Na prática utilizamos funções para separar melhor nossa lógica.
- Para definir funções em **Kotlin** utilizamos a palavra **fun** , seguida do nome da função, seus argumentos e seu retorno. Vamos a um exemplo prático, vou definir uma função com nome somar que recebe 2 argumentos inteiros e retorna a soma:

FUNÇÕES

- `fun somar(n1: Int , n2: Int): Int {`
 - `return n1 + n2 }`
- E para chamar a função:
 - `val resultado = somar(5 , 7)`
- Esse exemplo é de uma função que possui um retorno, por isso eu defini qual é este retorno e utilizei a palavra **return** para devolver o resultado.
- Nem todas as funções possuem um retorno. No exemplo, a função soma recebe dois números e retorna a sua soma, então faz sentido nesse caso retornar um valor. No entanto, há casos em que não é necessário retornar nenhum valor.

FUNÇÕES

- Para esses definimos que o retorno da função é **Unit** e não precisamos utilizar a palavra **return** .
 - ```
fun imprimir(texto: String): Unit{
 println(texto) }
```
- Perceba que essa função tem uma anatomia diferente da anterior: ela não possui a palavra **return** , pois ela não retorna nada, apenas exibe um texto no console. Utilizei a palavra **Unit** para indicar que essa função não possui retorno.

# FUNÇÕES

---

- No entanto, quando uma função é do tipo Unit , eu posso omitir a palavra Unit e o Kotlin vai entender sozinho que essa função não possui retorno. Então essa mesma função poderia ser reescrita da seguinte forma:

```
fun imprimir(texto: String) {
 println(texto)
}
```

# FUNÇÕES-código exemplo

```
fun main(args: Array<String>) {
 val n1 = 5
 val n2 = 7
 val resultado = somar(n1, n2)
 imprimir("A soma de $n1 + $n2 = $resultado")
}
fun somar(n1: Int, n2: Int): Int {
 return n1 + n2
}
fun imprimir(texto: String) {
 println(texto)
}
```

A soma de 5 + 7 = 12



# FUNÇÕES

---

- Um recurso bem interessante da linguagem são as **SingleExpression functions** (Funções de expressão única). Esse recurso simplifica a definição de uma função quando ela possui apenas uma linha, não sendo necessário o uso das chaves ( { } ). As mesmas funções somar e imprimir poderiam ser escritas da seguinte maneira:
  - `fun somar(n1: Int , n2: Int) = n1 + n2`
  - `fun imprimir(texto: String) = println(texto)`
- Perceba que, no caso da função somar , eu não precisei utilizar a palavra **return** e nem definir o tipo de retorno da função, essas informações o próprio compilador descobre sozinho porque ambas são **Single-Expression functions**.

# ORIENTAÇÃO A OBJETOS

---

- O **Kotlin** é uma linguagem que trabalha com o paradigma de Orientação a Objetos (OO) além do Paradigma Funcional. No contexto de aplicativos, é fundamental entender o básico de Orientação a Objetos e como trabalhar orientado a objetos com **Kotlin**.
- Podemos definir como Programação Orientada a Objetos um modelo baseado em objetos. Um objeto é uma abstração de algo que pode ser do mundo real ou não. Assim como no mundo real, pode ter características e pode executar ações. Vamos imaginar um objeto do mundo real, um carro. Ele tem diversas características como modelo, cor, rodas, motor, bancos etc. Um carro também pode executar algumas ações como acelerar, frear, virar à direita, virar à esquerda etc.



# ORIENTAÇÃO A OBJETOS

---

- Em programação, chamamos essas características de propriedades e suas ações de métodos. Então um objeto pode ter propriedades, que são características que o definem, e métodos, que são ações que ele pode executar.
- Para se criar um objeto em Kotlin devemos criar uma classe. Uma classe é onde colocaremos a programação do objeto. Programaremos suas propriedades e seus métodos e, através da classe, podemos criar instâncias deste objeto. Pense na classe como um molde para criação do objeto, de modo que na programação eu posso ter uma classe e, desta classe, criar N objetos.

# ORIENTAÇÃO A OBJETOS

---

- Para se criar uma classe em Kotlin utilizamos a palavra `class` seguida pelo nome da classe, veja um exemplo:
  - ```
class Carro{  
    }
```
- Toda a programação da classe ficará entre os parênteses (`{ }`) que indicam seu início e fim. Para definir suas propriedades, podemos criar variáveis em seu escopo:
 - ```
class Carro{
 var cor: String = ""
 var modelo:String = ""
}
```

# ORIENTAÇÃO A OBJETOS

- Desta forma, temos a classe carro com 2 propriedades, cor e modelo , ambas do tipo String . Para definir métodos em uma classe, basta criar funções nela, veja:

```
class Carro {
 var cor: String = ""
 var modelo:String = ""
 fun acelerar(){
 println("Acelerando")
 }
 fun frear(){
 println("freando")
 }
}
```



# ORIENTAÇÃO A OBJETOS

```
class Carro {
 var cor: String = ""
 var modelo:String = ""
 fun acelerar(){
 println("Acelerando")
 }
 fun frear(){
 println("freando")
 }
}
fun main(args: Array<String>) {
 val c = Carro()
 c.cor = "Azul"
 c.modelo = "Nissan 350z"
 c.acelerar()
 c.frear();
}
```

```
Acelerando
freando
```

2 métodos na classe, o acelerar e o frear .  
Orientada a Objetos simplesmente é uma  
criação de um carro. A grande vantagem dessa  
linguagem é que, sempre que eu quiser utilizar este objeto,  
eu posso criar um objeto, assim como **val c = Carro()** , então através da variável c eu  
posso chamar os métodos:

# Exemplo 02

```
open class Carro{
 var cor: String = ""
 var modelo:String = ""
 fun acelerar(){
 println("Acelerando")
 }
 fun frear(){
 println("freando")
 }
}
class CarroEspecial : Carro(){
 fun fazerDrift(){//implementação
 println("Classe derivada")
 }
}
fun main(args: Array<String>) {
 val c = CarroEspecial()
 c.acelerar();
 c.frear()
 c.fazerDrift();
}
```

# ORIENTAÇÃO A OBJETOS

---

- **Herança**
- Uma das características mais marcantes da Orientação a Objetos é a capacidade de se fazer uma herança entre classes. Isso quer dizer que conseguimos criar subclasses de alguma outra classe. Por exemplo, imagine que precisaremos criar um modelo de carro específico com propriedades e métodos novos, porém Herança mantendo as propriedades e métodos de um carro comum.



# ORIENTAÇÃO A OBJETOS

---

- Temos duas opções nesse caso, a primeira seria criar essa classe nova e repetir todo o código da classe carro . Essa opção não me parece muito legal, uma vez que uma das vantagens da Orientação a Objetos é a reutilização de código. A melhor opção seria utilizar um recurso de linguagens orientadas a objetos chamado herança.
- O conceito de herança em OO é bem parecido com o conceito de herança no mundo real. Assim como uma pessoa pode herdar características genéticas de seus pais, em programação, uma classe pode herdar características e métodos de outra classe! Para esse caso, seria a solução perfeita. Poderíamos então criar uma nova classe CarroEspecial e fazer uma herança da classe Carro e somente implementar as propriedades e métodos novos.

# ORIENTAÇÃO A OBJETOS

---

- Para se fazer a herança de uma classe, devemos colocar “:” na frente do nome da classe e, em seguida, instanciar a classe pai:
  - ```
class CarroEspecial : Carro() {  
    fun fazerDrift() { //implementação  
    } }
```
- Desta forma, a classe CarroEspecial herda automaticamente todas as propriedades e métodos da classe Carro , e implementa um método novo. Mas para isso funcionar, a classe Carro deve permitir a explicitamente que se faça herança dela.
- Para isso, devemos utilizar a palavra **open** na definição da classe, assim: **open class Carro** . Veja como ficaria o código completo:

ORIENTAÇÃO A OBJETOS

```
open class Carro {  
  
    var cor: String = ""  
    var modelo:String = ""  
  
    fun acelerar(){  
        println("Acelerando")  
    }  
  
    fun frear(){  
        println("freando")  
    }  
  
}  
  
class CarroEspecial : Carro(){  
  
    fun fazerDrift(){  
        //implementação  
    }  
  
}
```

Código executado
com main()

```
open class Carro{  
    var cor: String = ""  
    var modelo:String = ""  
    fun acelerar(){  
        println("Acelerando")  
    }  
    fun frear(){  
        println("freando")  
    }  
}  
class CarroEspecial : Carro(){  
    fun fazerDrift(){//implementação  
        println("Classe derivada")  
    }  
}  
fun main(args: Array<String>) {  
    val c = CarroEspecial()  
    c.acelerar();  
    c.frear()  
    c.fazerDrift();  
}
```

```
Acelerando  
freando  
Classe derivada
```


Classe de dados (Data class)

- Durante o desenvolvimento de software, é muito comum utilizar classes de dados. São classes que geralmente não possuem nenhum método, somente propriedades, e nos ajudam a transitar e organizar os dados em uma aplicação. Imagine a modelagem de um usuário de um aplicativo. Vamos supor que este usuário terá um nome, um e-mail e também uma senha. Podemos modelar uma classe para isso e, como será uma classe somente para guardar essas informações, ela pode ser uma data class . A grande vantagem das classes de dados em Kotlin é a sua simplificação de implementação. Podemos criar essa classe somente com uma linha de código! Veja:
- **`data class Usuario(var nome:String, var email:String, var senha:String)`**

CONFIGURANDO O AMBIENTE DE DESENVOLVIMENTO

- Para começar a criar nossos projetos, precisamos configurar nosso ambiente de desenvolvimento. Para isso, vamos instalar o Android Studio, a IDE oficial para desenvolvimento Android. Junto ao Android Studio, está o Android SDK, o emulador e tudo mais que for necessário para o desenvolvimento.

```
package classes
class Calculadora {
    private var resultado: Int = 0
    fun somar(vararg valores: Int): Calculadora {
        valores.forEach { resultado += it }
        return this
    }
    fun multiplicar(valor: Int): Calculadora {
        resultado *= valor
        return this
    }
    fun limpar(): Calculadora {
        resultado = 0
        return this
    }
    fun print(): Calculadora {
        println(resultado)
        return this
    }
    fun obterResultado(): Int {
        return resultado
    }
}

fun main(args: Array<String>) {
    val calculadora = Calculadora()
    calculadora.somar(1, 2, 3).multiplicar(3).print()
    calculadora.somar(7, 10).print().limpar()
    println(calculadora.obterResultado())
}
```


O QUE É O ANDROID SDK?

- Podemos dizer que o Android SDK é nosso kit de desenvolvimento. A sigla SDK significa Software Development Kit. O SDK do Android vem com todas as ferramentas, APIs e bibliotecas necessárias para se trabalhar com a plataforma Android. Junto ao SDK, também estão os emuladores que você utilizará para testar seus aplicativos. Não se preocupe em instalá-lo, pois ele vem junto com o Android Studio.

ANDROID STUDIO

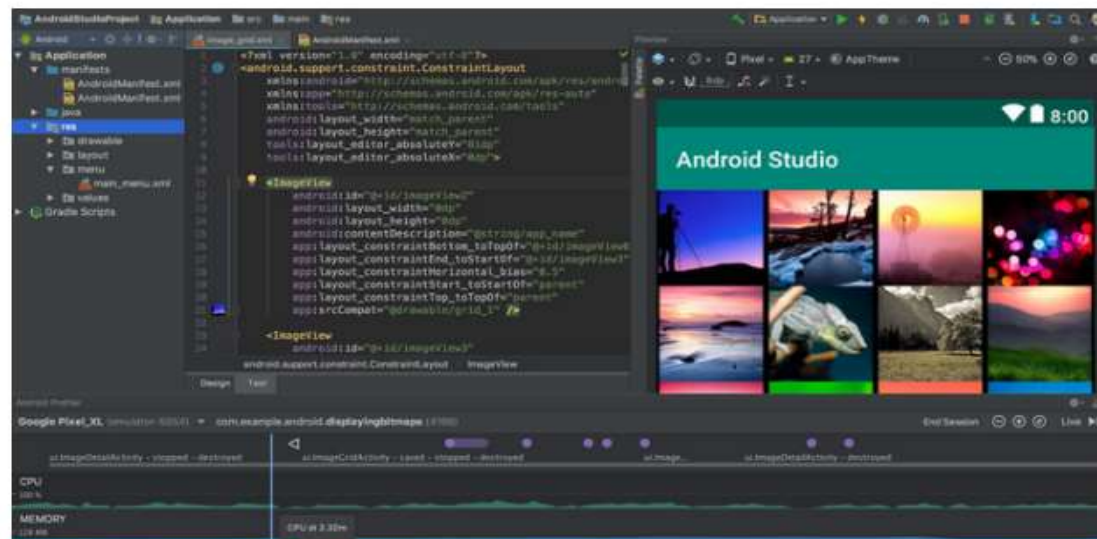
- O Android Studio será nossa IDE de desenvolvimento. Vamos usá-lo porque basicamente ele é a IDE oficial do Android, é muito bom e possui diversos recursos que agilizam o desenvolvimento. Seu editor de código é muito bom, e você não precisa de nenhuma outra ferramenta adicional, isto é, consegue fazer tudo somente com o Android Studio!
- Para baixar o Android Studio, você deve entrar em <https://developer.android.com/studio/index.html>.



Android Studio provides the fastest tools for building apps on every type of Android device.

[Download Android Studio](#)

2021.1.1 for Windows 64-bit (872 MB)

[Download options](#)[Release notes](#)

ANDROID STUDIO

- Repare que existe um botão grande escrito **DOWNLOAD ANDROID STUDIO** ; é aí que você deve clicar para baixar o programa. O site automaticamente vai identificar seu sistema operacional e encaminhar para o download correto. Você pode desenvolver para Android utilizando um computador com Windows, Mac ou até mesmo Linux!

ANDROID STUDIO

- Para Windows, os requisitos do sistema são: Microsoft® Windows® 7/8/10 (32 ou 64 bits); Mínimo de 3 GB de RAM, 8 GB de RAM recomendados, mais 1 GB para o Android Emulator;
- Mínimo de 2 GB de espaço livre em disco,
- 4 GB recomendados (500 MB para o IDE + 1,5 GB para o Android SDK e as imagens do sistema do emulador);
- Resolução de tela mínima de 1.280 x 800;
- Para o emulador acelerado: sistema operacional de 64 bits, processador Intel® compatível com Intel® VT-x, Intel® EM64T (Intel® 64) e a funcionalidade Execute Disable (XD) Bit.

ANDROID STUDIO

- Para Linux, precisaremos de: Área de trabalho GNOME ou KDE;
- Testado no Ubuntu® 12.04,
- Precise Pangolin;
- Distribuição de 64 bits, capazes de executar aplicativos de 32 bits;
- Biblioteca C do GNU (glibc) 2.19 ou posterior; Mínimo de 3 GB de RAM, 8 GB de RAM recomendados, mais 1 GB para o Android Emulator;
- Mínimo de 2 GB de espaço livre em disco;
- 4 GB recomendados (500 MB para o IDE + 1,5 GB para o Android SDK e as imagens do sistema do emulador);
- Resolução de tela mínima de 1.280 x 800;
- Para o emulador acelerado: processador Intel® compatível com Intel® VT-x, Intel® EM64T (Intel® 64) e a funcionalidade Execute Disable (XD) Bit, ou processador AMD compatível com AMD Virtualization™ (AMD-V™).

ANDROID STUDIO

- Agora com o Android Studio baixado, vamos partir para sua instalação. Não há muito segredo, é basicamente next-next finish . Durante a instalação, o Android Studio vai configurar o SDK e provavelmente baixar algumas atualizações. Isso pode demorar mais alguns minutos. Ao término, você verá a seguinte tela:



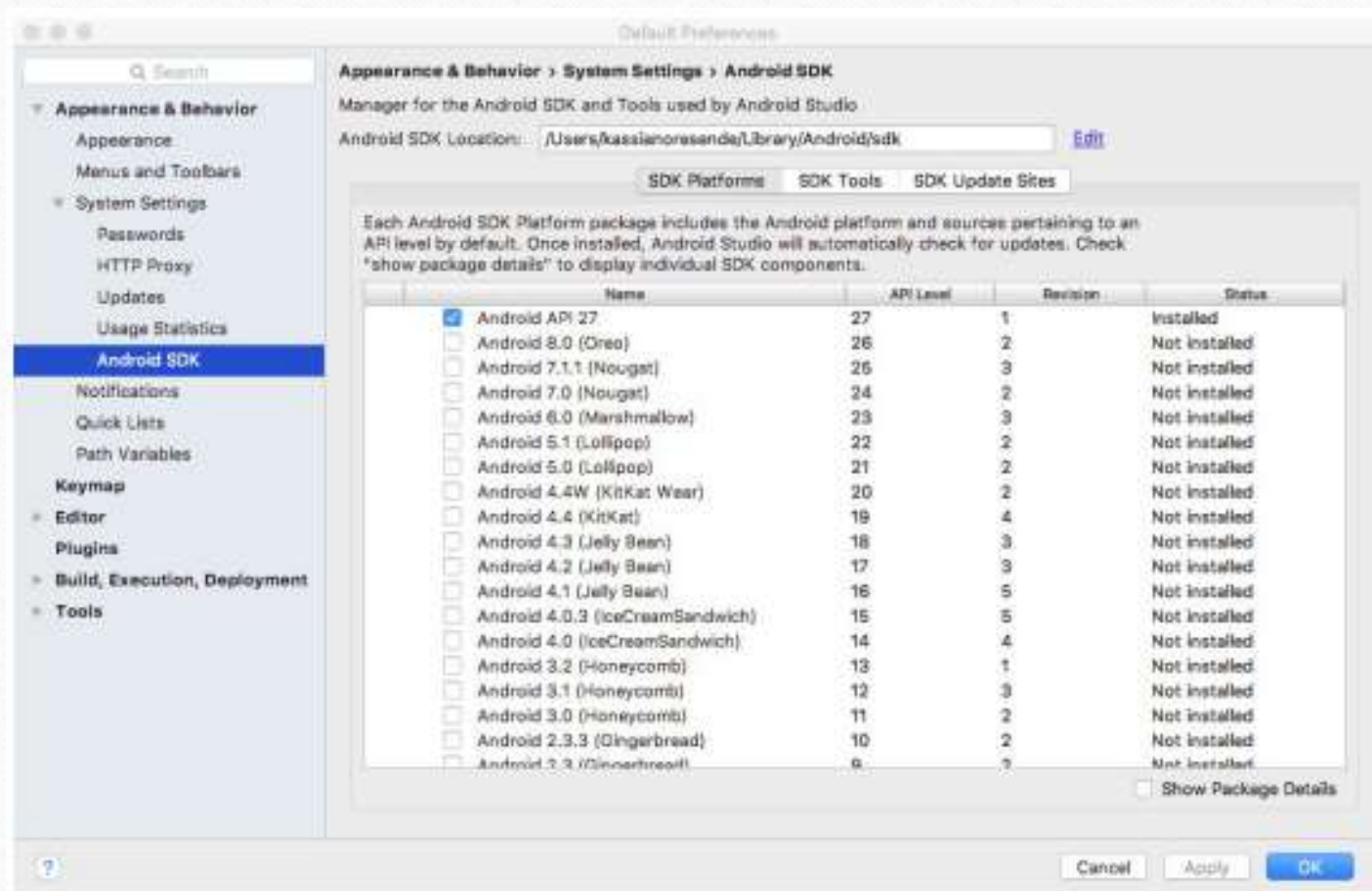
ANDROID STUDIO

- Esta é a tela inicial do Android Studio. A partir dela, você poderá criar um projeto novo, abrir um existente, fazer um check out direto de algum controlador de versão, debugar um APK, importar um projeto Gradle ou importar um exemplo de código. Existe um pequeno botão configure com as seguintes opções:

Opção	Descrição
SDK Manager	Gerenciador do SDK, no qual conseguimos atualizar e instalar novas plataformas e ferramentas.
Preferences	Configurações gerais da IDE, como tamanho de fonte, tema da IDE, controle de versões, formas de compilação etc.
Plugins	Gerenciador de plugins, no qual podemos instalar novos ou atualizar existentes.
Import Settings	Aqui, você pode importar configurações de outra instalação do Android Studio.
Export Settings	É possível exportar as configurações do Android Studio.
Setting Repository	Esta opção habilita a entrada de uma URL para a configuração de um repositório externo.
Check for update	Verifica se há atualizações.
Project Defaults	Esta opção abre links para configurações padrão da IDE.

CONFIGURANDO O SDK

- Um pré-requisito para que tudo funcione corretamente é a configuração correta do SDK. O Android Studio já instala a última versão disponível durante a instalação do programa, então, provavelmente já está tudo configurado corretamente para começarmos a programar.
- No entanto, é importante checar se a instalação está correta. Também é importante conhecer onde se atualiza o SDK. Abra a opção SDK Manager pelo link Configure , visto anteriormente. Isto abrirá a seguinte janela:

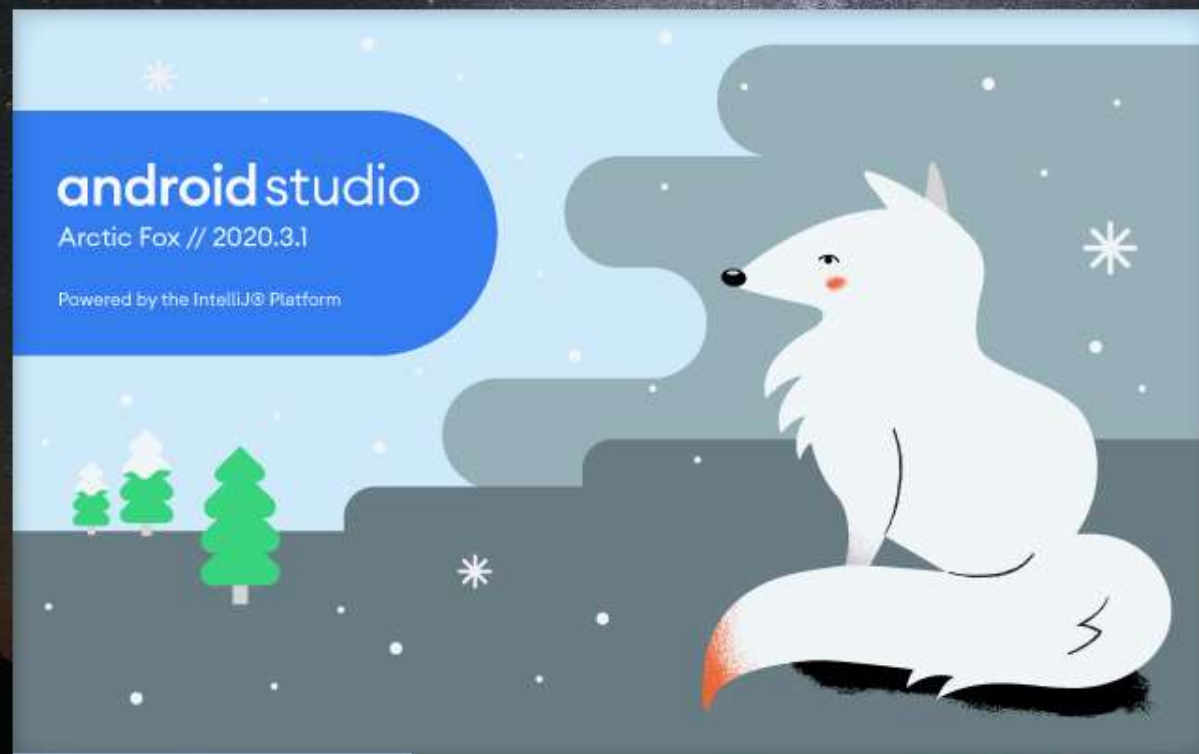


CONFIGURANDO O SDK

- Esta é dividida por três abas: SDK Platforms , SDK Tools e SDK Update Sites . Em SDK Platform , podemos escolher as plataformas Android com qual trabalharemos. Por padrão, já vem instalada a plataforma mais recente, neste caso, a API 27 – correspondente ao Android Oreo 8.1.0. Para cada atualização do sistema operacional, é lançada uma nova versão da API também, sendo que cada versão tem uma enumeração. Assim, o Android 8.1.0 é a API 27; a 26, o Android 8.0; a 25, o Android Nougat 7.1.1 e assim por diante. Você pode conferir todas as versões de API diretamente em:
- <https://developer.android.com/guide/topics/manifest/uses-sdkelement.html?hl=pt-br>

Carregando Android Studio

- Observe o carregando do android, dependendo da configuração do seu hardware pode demora um pouquinho, veja imagem a seguir:



☰ Digite aqui para pesquisar



☀ 29°C Ensolarado 12:26 27/01/2022

daemon started successfully

☰ Digite aqui para pesquisar



☀ 29°C Ensolarado 12:21 27/01/2022

PRIMEIRO PROJETO: HELLO WORLD

- Agora que temos nosso ambiente todo configurado, é hora de criar nosso primeiro projeto! Vamos fazer um clássico Hello World. Nele, não utilizaremos praticamente nenhum código em Kotlin. A intenção é que você conheça um pouco da IDE e também configure um emulador para testar os futuros aplicativos. Então, vamos lá. Na tela inicial, clique em Start a new Android Studio Project . A primeira tela na qual você precisa preencher algumas informações sobre o projeto será:

Arquivo Início Extensões

Recortar Copiar Colar

Área de Transferência

Copiar Colar

Área de Transferência

lixeira

Microsoft Edge

Google Chrome

Apache Beans I...

Skype

Android Studio

Welcome to Android Studio

Android Studio Arctic Fox | 2020.3.1 Patch

Search projects

New Project Open Get from VCS

MeuAPP
~\AndroidStudioProjects\MeuAPP

My Application
~\AndroidStudioProjects\MyApplication

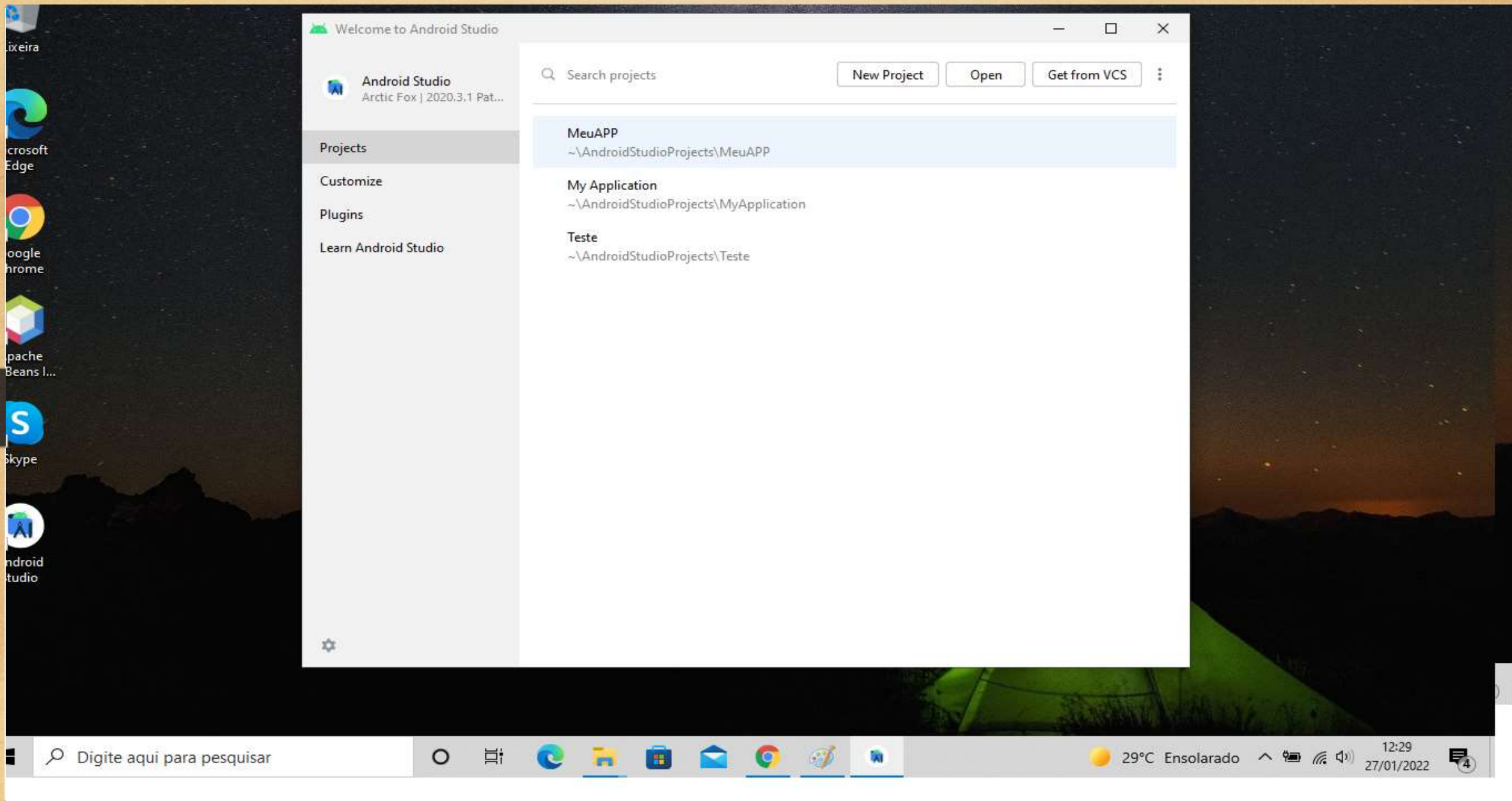
Teste
~\AndroidStudioProjects\Teste

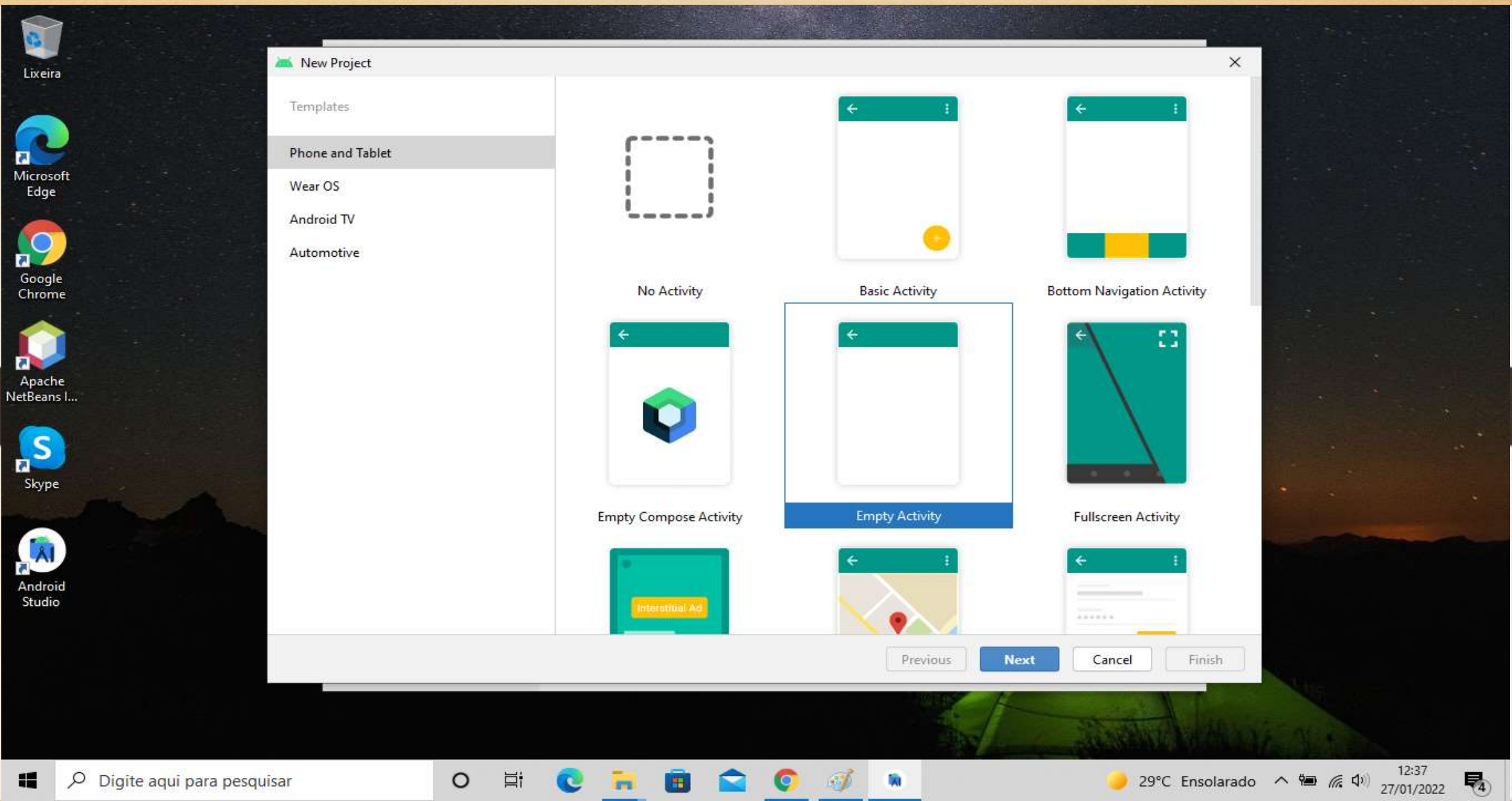
Projects

Customize

Plugins

Learn Android Studio





Digite aqui para pesquisar



29°C Ensolarado



12:37

27/01/2022



Empty Activity

Creates a new empty activity

Name

Package name

Save location





Language



Minimum SDK



 Your app will run on approximately **98,0%** of devices.
[Help me choose](#)

☐ Use legacy android.support libraries 

Using legacy android.support libraries will prevent you from using
the latest Play Services and Jetpack libraries

Previous

Next

Cancel

Finish

PRIMEIRO PROJETO: HELLO WORLD

- Em **Application Name** , digitei Hello world , já que este é o nome do projeto.
- Em **package name** , coloquei com.example.helloworld, para a criação do nome de pacote da aplicação.
- Em Project Location , você deve indicar a pasta para salvar o projeto. No meu caso, configurei uma pasta chamada android dentro de Documentos , deixando todos os projetos lá.
- Por fim, linguagem kotlin que será usada e os principais dispositivos suportados.