# Inferência automática de tipos;

O Kotlin ele sabe automaticamente qual o tipo da variável, seja String, int, float...

# Como definir uma variável em Kotlin

Var = (mutável) > posso mudar o valor

Val = (imutável) > variável que não posso mudar o valor (constantes)

package Variaveis  
  
fun main() {  
 *println*("Hello Word!!!")  
 var nome = "Alleph";  
 val cpf = 15679025701;  
}

Segue o exemplo, nome podemos mudar, CPF é único.

# Paradigmas

Kotlin trabalha com Orientação a Objetos e Funcional.

# Como iniciar o Kotlin

Assim como o Java vamos iniciar com uma função main

package Variaveis  
  
fun main() {  
 *println*("Hello Word!!!")  
 var nome = "Alleph"  
 val cpf = 15679025701  
}

# Concatenando no Kotlin

fun main() {  
 *println*("Hello Word!!!")  
 var nome = "Alleph"  
 val cpf = 15679025701  
  
 *println*("Seja bem-vindo $nome $cpf")  
}

Saída:

Hello Word!!!

Seja bem-vindo Alleph 15679025701

# Variáveis – Tipos e Instância

fun main() {  
 val nome: String = "Alleph"  
 val cpf: Int = 12345678910  
}

## Tipos das variáveis

Byte, Short, Int, Long, Float, Double, Boolean, Char.

## Constantes

A constante ela deve ser utilizada de duas maneiras, dentro da uma função.

fun main() {  
 val nome: String = "Alleph"  
 val cpf: Long = 12345678910 // Tempo de execução  
}

Fora de uma função

package Variaveis  
  
const val *nomeUsuario*: String = "Alleph Nogueira" // Constante global

// Tempo de compilação  
fun main() {  
 val nome: String = "Alleph"  
 val cpf: Long = 12345678910  
}

Uma variável global, ela está disponível para todas as funções dentro do arquivo.

## Tempo de compilação x Tempo de execução

Tempo de compilação: O computador já tem que saber o valor da variável antes de tudo.

Tempo de execução: O valor pode ser adicionado a uma variável val na hora de compilar.

## Instância com Kotlin

package Variaveis  
  
class Usuario {}  
  
fun main() {  
 val usuario = Usuario()  
 // Lembrar que quando eu faço a instancia com val o usuario não pode ser mudado.  
  
}

# Null Safety e Chamada segura

Imagina que você tem um objeto carro, e aí tudo certo, mas você também pode não conseguir instanciar o objeto e aí ficaria nulo, isso geraria um erro e fecharia a aplicação. Ou seja, você não pode deixar de hipótese nenhuma um item nulo, mas para isso o Kotlin tem uma solução; Null Safety (Nulo seguro) podemos colocar dentro de um objeto um valor nulo, ou seja, não existe o carro ainda, mas em algum momento o carro vai existir.

Exemplo a baixo do Kotlin;

fun main() {  
 var nome: String = "alleph" // Variavel correta  
 var sobrenome: String? // esse ? indica que essa variavel pode receber um valor nullo, ou ate mesmo um texto.  
}

## Usando Java que não tem o Null Safaty

### Null Safaty (Erro em Java)

package Variaveis.NullSafaty;  
  
  
class Carro{  
 String cor = "Prata";  
  
 void acelerar (){}  
}  
  
public class NullSafaty {  
 public static void main(String[] args) {  
  
 // Imagina que não instanciamos o carro  
 Carro carro = null;  
  
 // Imagina que o carro vai ser instanciado em outro local, mas acabou não sendo instanciado  
  
  
 // E ai tentamos usar o carro  
 carro.acelerar();  
  
 // E ai iria gerar o erro NullPointerExecption  
 }  
}

### Null Safaty (Erro tempo de execução)

Imagina que temos um botão, e o erro vai acontecer quando o usuário clicar nele, isso se chama tempo de execução é quando o programa está em funcionamento

package Variaveis.NullSafaty;  
  
  
class Carro{  
 String cor = "Prata";  
  
 void acelerar (){}  
}  
  
public class NullSafaty {  
 public static void main(String[] args) {  
  
 }  
   
 void cliqueBotao(){  
  
 // Imagina que não instanciamos o carro  
 Carro carro = null;  
  
 // Imagina que o carro vai ser instanciado em outro local, mas acabou não sendo instanciado  
  
  
 // E ai tentamos usar o carro  
 carro.acelerar();  
  
 // E ai iria gerar o erro NullPointerExecption  
 }  
}

## Usando o Kotlin que tem o Null Safaty (ERRO)

package Variaveis.NullSafaty  
  
class Carro {  
 var cor = "Prata"  
  
  
 fun acelerar(){}  
}  
  
fun main() {  
  
 // Aqui estou dizendo que essa variavel é do tipo CARRO(Class)  
 var carro: Carro  
  
 // Mas isso daria um erro  
 *println*(carro.cor)  
}

Agora vamos imaginar fazendo da forma correta, onde vamos colocar que o carro pode ser nulo, mas ele vai receber um valor à medida que o código vai crescendo.

package Variaveis.NullSafaty  
  
class CarroK {  
 var cor = "Prata"  
  
  
 fun acelerar(){}  
}  
  
fun main() {  
  
 // Aqui estou dizendo que essa variavel é do tipo CARRO(Class)  
 var carro: CarroK? = null  
  
 // Em algum lugar do codigo o carro vai receber algo e vai deixar de ser nullo  
 carro = CarroK()  
  
 *println*(carro?.cor)

// Pra que serve a chamada? Aqui ele vai verificar se o carro esta nulo, se estiver nulo ele não faz nada, se o carro tiver algo ele chama normalmente a variavel  
}

## Verificando se a variável esta nulo para evitar erro em execução **(Chamada SEGURA > ?)**

package Variaveis.NullSafaty  
  
class CarroK {  
 var cor = "Prata"  
  
  
 fun acelerar(){}  
}  
  
fun main() {  
  
 // Aqui estou dizendo que essa variavel é do tipo CARRO(Class)  
 var carro: CarroK? = null  
  
 *println*(carro?.cor)  
}

Observa que ele verifica o carro é nulo antes com o **?** Se for nulo ele não vai fazer nada, se não for ele segue o processo.

## Uma forma de não exibir o Null se a variável estiver nula

class CarroK {  
 var cor = "Prata"  
  
  
 fun acelerar(){}  
}  
  
fun main() {  
  
 // Aqui estou dizendo que essa variavel é do tipo CARRO(Class)  
 var carro: CarroK? = null  
  
 // Se o carro for nulo e eu quiser exibir uma cor padrao?  
 // ?: Isso é chamado e Elvis Operation  
 // 1 Primeiro vamos testar se o carro é nulo carro?.cor  
 // 2 Se o carro for nulo ele não vai acessar a cor do carro  
 // 3 Então agora ele vai usar a segunda opção que é "Cor padrao"  
 // Funciona bem parecido com o Operador Ternario, ele testa a primeira opção e tem a segunda "Cor padrao"  
 val cor = carro?.cor ?: "Cor padrão"  
  
 *println*(cor)  
   
 *println*(carro?.cor) // Aqui se o carro for nulo vai exibir o nulo.  
}

SAIDA:

Cor padrão

null

# Kotlin – Class – Atributo – Método – Instância

package Variaveis.NullSafaty  
  
class Carro { // Classe  
 var cor = "Prata" // Atributo  
  
  
 fun acelerar(){} // Metodo  
}  
  
fun main() {  
 var nome: String = "alleph" // Variavel correta  
 var sobrenome: String? // esse ? indica que essa variavel pode receber um valor nullo, ou ate mesmo um texto.  
  
  
 var carro = Carro(); // Minha instancia em Kotlin  
}

# Estruturas condicionais (If, Else)

package EstruturasCondicionais  
  
fun main() {  
 var conta = 200  
 var despesas = 0  
 var sol = false  
  
 if (conta > despesas && sol) {  
 *println*("Você pode ir a praia")  
 }else {  
 *println*("O tempo não esta legal para praia hoje.")  
 }  
}

Outro teste usando apenas a variável condição

val notaAluno = 7  
val condicao = notaAluno >= 6 // Nota do aluno é maior ou igual a 6? isso vai retornar um True ou False  
// e logo em baixo vamos confirmar se devemos entrar no if ou não  
if (condicao) {  
 *println*("Aprovado")  
}else{  
 *println*("Reprovado")  
}  
  
*println*(condicao)

Out: Aprovado

True

## If com IN

//////////////////////// IF COM IN ////////////////  
  
// Observa que o IN ele diz assim  
// Se a opcao estiver em(IN) 1 ate 3  
var opcao = 4  
if (opcao in 1..3) {  
 *println*("Cartao de credito")  
}else if (opcao in 4..5){  
 *println*("Pagamento com dinheiro")  
}else {  
 *println*("Opcao invalida.")  
}

Saída: Pagamento com dinheiro

## WHEN Lembrar do Switch

package EstruturasCondicionais  
  
fun main() {  
 val opcao = 6  
  
 when(opcao){ // Opcao que vamos testar.  
 1 -> *println*("Cartao de credito")  
 2 -> *println*("Extrato bancario")  
 in 3..5 -> *println*("Saldo") // Também podemos usar o in em WHEN como você pode ver.  
 else -> *println*("Opcao não encontrada")  
 }  
}

\*\*\*\* OPCAO = 6 \*\*\*\*

Out: Opcao não encontrada