Kotlin oferece suporte ao paradigma de **programação funcional**. Programação funcional segue muito a ideia de se programar **baseado em argumentos das funções** e **mantendo a imutabilidade** sempre devolvendo algo novo invés de alterar a fonte.

Funções no **Kotlin** são *"first-class"*, isso significa que **elas podem ser armazenadas em variáveis** ou **passadas como argumento de outra função.** Quando **armazenadas em variáveis**, a variável precisa ter a mesma assinatura da função, nos **"()" os argumentos** e depois da **"->" o retorno:**

```
fun main(args: Array<String>) {

    val funcaoEmVariavel : (String) -> Boolean = :: checarSenha

    print(funcaoEmVariavel("Senha"))
}

fun checarSenha(senha: String) : Boolean {
    return senha == "Senha"
}
```

Funções também **podem ser declaradas a partir de Classes**, classes que são "funções" tem o método invoke que precisa ser sobrescrito com a implementação da função:

```
fun main(args: Array<String>) {

    val funcaoEmVariavel : (String) -> Boolean = ChecarSenha()

    print(funcaoEmVariavel("Senha"))
}

class ChecarSenha : (String) -> Boolean {
    override fun invoke(senha: String): Boolean {
        return senha == "Senha"
    }
}
```

A **expressão lambda** é exatamente o **tipo função de uma maneira muito mais enxugada e comum**, onde basta apenas utilizar as ***"{}"*** para escrever a implementação (***lambda expression e função anônimas** são **muito atreladas ao escopo da variável delas**):*

```
fun main(args: Array<String>) {

    val funcaoLambdaEmVariavel: (String) -> Boolean = {  it: String
        it == "Senha"
    }

}
```