Kotlin Introduction

Agenda

- What is Kotlin?
- Kotlin Features
- main function
- Mutable and Immutable
- Kotlin Strings
- Looping
- Null safety
- Class and Objects
- **Hands on Example - Dinner Decider App in Kotlin**

Introduction

- The Android team announced during Google I/O 2017 that Kotlin is an official language to develop Android Apps.
- It's expressive, safe, and powerful.
- Best of all, it's interoperable with our existing Android languages and runtime. Kotlin is concise while being expressive.
- It contains safety features for nullability and immutability, to make your Android apps healthy and performant by default.

What is Kotlin?

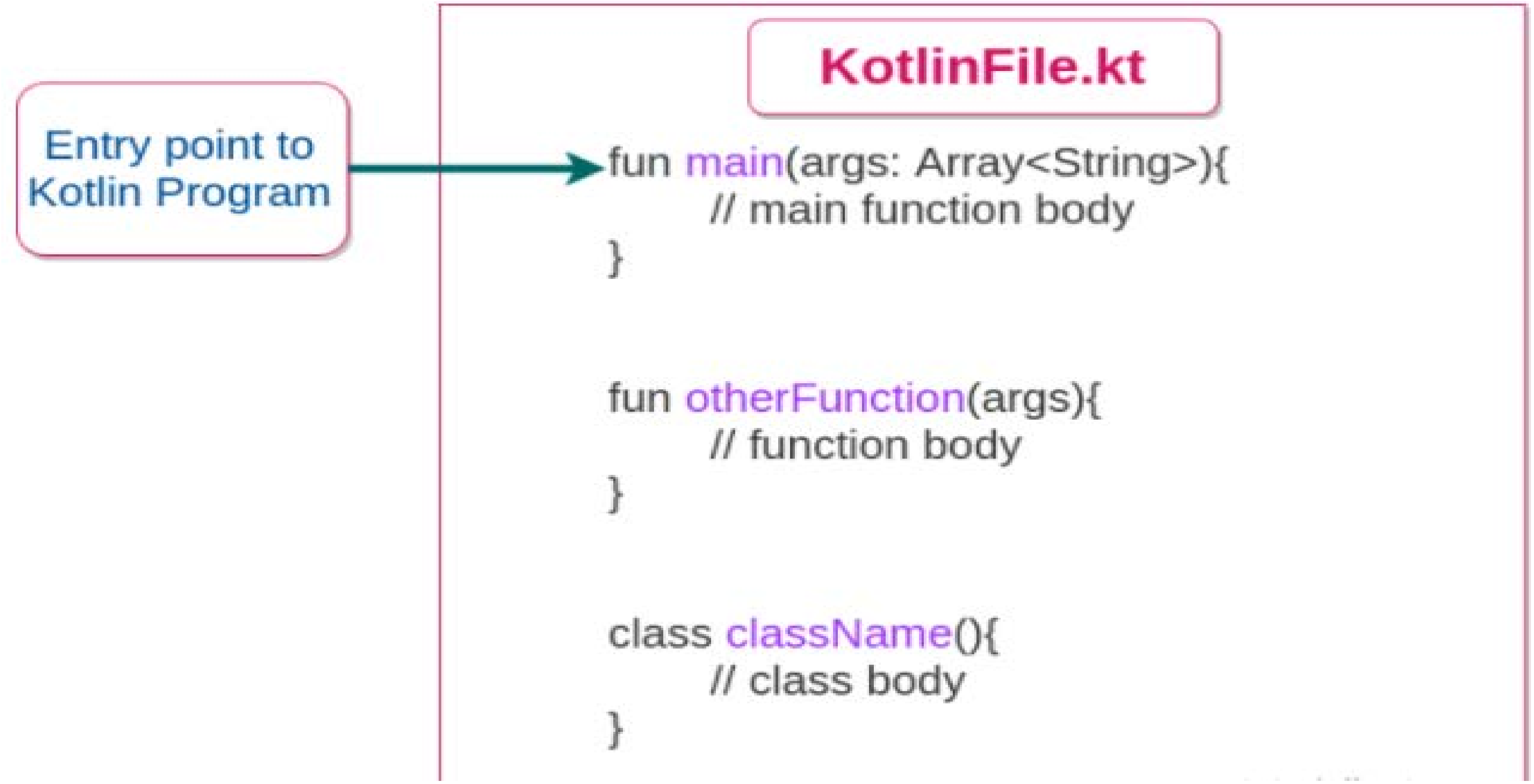
- Kotlin is a JVM based language developed by JetBrains5 , a company known for creating IntelliJ IDEA, a powerful IDE for Java development.
- Android Studio, the official Android IDE, is based on IntelliJ.
- Kotlin was created with Java developers in mind, and with IntelliJ as its main development IDE.
  - Learn more about from https://developer.android.com/kotlin/index.html
  - Try online : https://try.kotlinlang.org/
  - Find Answers about Kotlin from https://developer.android.com/kotlin/faq.html

Kotlin Features

- **It's more expressive:** You can write more with much less code.
- **It's safer :** Kotlin is null safe, which means that we deal with possible null situations in compile time, to prevent execution time exceptions.
- **Functional and object-oriented:** Kotlin is basically an object oriented language, also gains the benefits of functional programming.
- **Statically typed:** This means the type of every expression in a program is known at compile time, and the compiler can validate that the methods and fields you're trying to access exist on the objects you're using.
- **Free and open source:** The Kotlin language, including the compiler, libraries, and all related tooling, is entirely open source and free to use for any purpose.
- **It's highly interoperable:** You can continue using most libraries and code written in Java, because the interoperability between both languages is excellent. It's even possible to create mixed projects, with both Kotlin and Java files coexisting. Easily convert the existing Java code into Kotlin by selecting Code >Convert Java File to Kotlin File

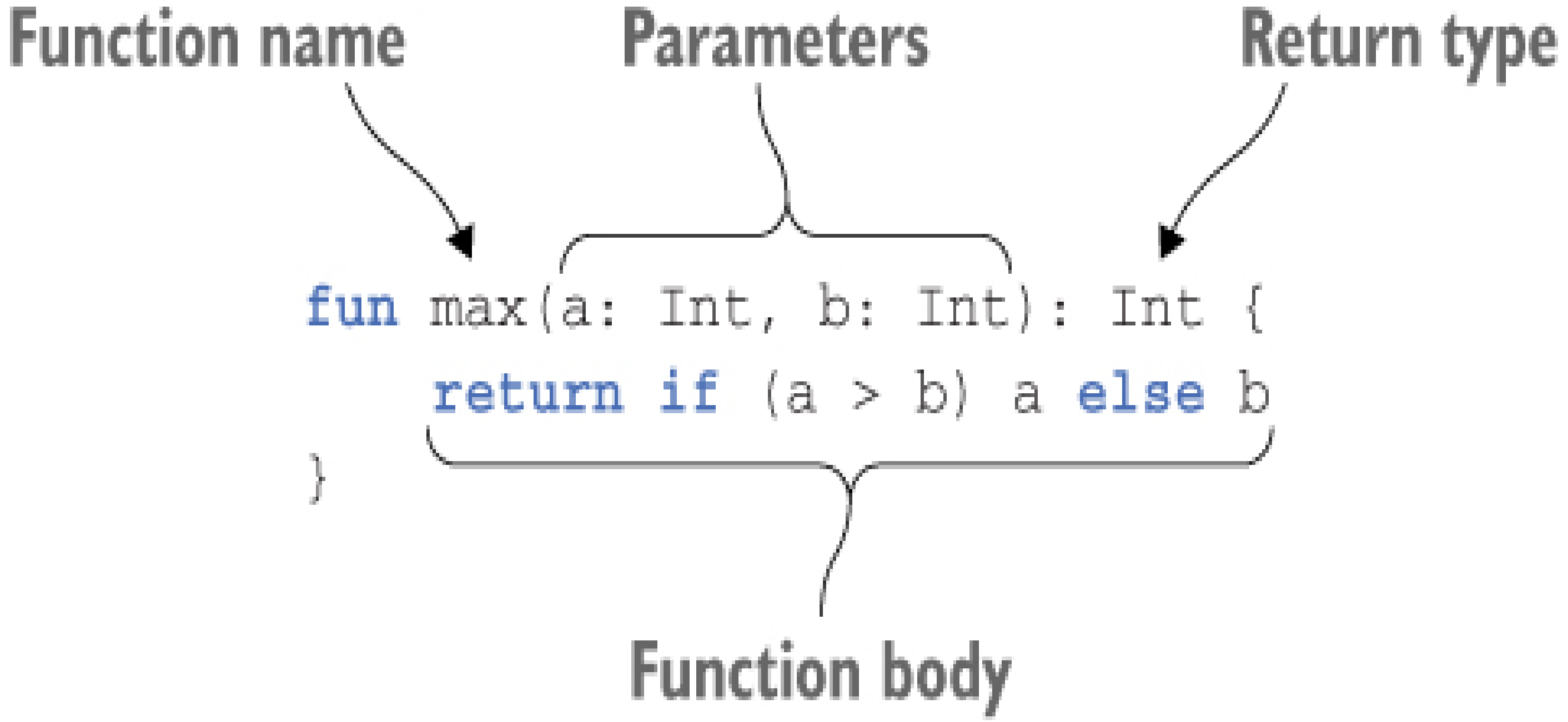
main function in Kotlin

This is the main function, which is mandatory in every Kotlin application. The Kotlin compiler starts executing the code from the main function.



```
fun main(args: Array<String>){
        // main function body
}


fun otherFunction(args){
        // function body
}


class className(){
        // class body
}
```

Declaring Function



```
fun max(a: Int, b: Int): Int {
    return if (a > b) a else b
}
```

Mutable and Immutable

- In Java, you start a variable declaration with a type. This wouldn’t work for Kotlin, because it lets you omit the types from many variable declarations.
- Mutable/Variable : type is not required.

  var answer = 42 is similar to → **var** answer: Int = 42
- Immutable/ Constants :

  val answer = 42 is similar to → **val** answer: Int = 42

Kotlin Strings

- Strings in kotlin represent an array of characters. Strings are immutable. It means operations on string produces new strings.

Eg: val str = " Kotlin Strings"

 println(str)

- we can create a multi line string using """

```
Eg: val x: String = """Kotlin
supports
Multiline
Strings"""
```

- The above code produce the intent spacing from the second line. To avoid spacing you can trim the space by giving

```
Eg: val x: String = """Kotlin
supports
Multiline
Strings""".trimMargin()
```

- **String templates**: String templates is a powerful feature in kotlin when a string can contain expression and it will get evaluated.

```
val x = "David"
val y = "My name is $x"
println(y)
Println("My name is $x with the length
${x.length}")

Output : My name is David
         My name is David with the length 5
```

- **String operations**

Eg:

```
var s: String = "Hello"
println("Length of string $s is ${s.length}")
 println("Init cap of string is ${s.capitalize}")
println("Lower case is ${s.toLowerCase}")
println("Upper case is ${s.toUpperCase}")
```

Refer for more info :
https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-string/index.html

Kotlin loops for, forEach, repeat, while, do-while

- **for :** for loop iterates through anything that provides an iterator.

Syntax : for (item **in** collection) { body}

Example :

```
var daysOfWeek =
listOf("Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday")
   for(day in daysOfWeek){
      println(day)
   }
```

- **for with index :**

```
for((index,day) in
daysOfWeek.withIndex()){
      println("$index :$day")
   }
```

- forEach can be used to repeat a set of statement for each element in an iterable.

```
daysOfWeek.forEach{
      println(it)
   }
```

- while and do..while work as usual
- **repeat :** repeat statement is used when a set of statements has to be executed N-number of times.

```
Eg: repeat(4) {
   println("Hello World!")
}
```

Kotlin - Ranges, when expression

Ranges

- You can create a range in kotlin via **..** operator.
- Example

for(i in 1..5) { print(i) }

Result: 1 2 3 4 5

- downTo : Using downTo function we can go reverse in a range.
- for(i in 5 downTo 1) { print(i) }

Result: 5 4 3 2 1

- step : Using step function we can increase the step

for (i in 1..10 step 2) { print(i) }

Result: 1 3 5 7 9

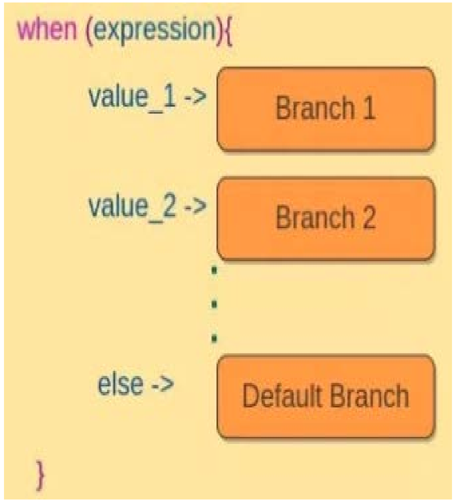
if we want to exclude the last value in a range use until

for (i in 1 until 5) { print(i) }

Result: 1 2 3 4

When expression

- Kotlin when expression is kind of switch case in Java, but concise in syntax, extended in functionality and more fun. Using "Any" object type with **when** expression makes is really broad in the usage.



- **Example**

```
when (x) {
 1 -> print("x == 1")
2 -> print("x == 2")
 else -> { // Note the block
           print("x is neither 1 nor 2")
            }
 }
```

Kotlin - Null Safety

- Null Safety in Kotlin is to eliminate the risk of occurrence of NullPointerException in real time.
- These are the scenarios, you will get NullPointerException in Kotlin
  1. If you make a explicit call to **throw a NullPointerException**
  2. use**!!** operator. (see in the example section)
  3. You may be aware that you can run external Java code in your application. (From outside Kotlin)
  4. Data inconsistency with regard to initialization
- Way to handle Null Safety in Kotlin
  1. Differentiate between nullable references and non-nullablereferences.
  2. User explicitly checks for a null in conditions
  3. Using a Safe Call Operator (?.)
  4. Elvis Operator (?:)

Way to handle Null Safety in Kotlin

- Way – 1 - Differentiate between nullable references and non-nullablereferences.
  - Kotlin’s type system can differentiate between nullable references and non-nullable references. **?** operator is used during variable declaration for the differentiation.

```
//Non- nullable – You cannot assign null to
//the non-null variable
var a: String = “Hello"
a = null // compilation error
```

```
// If you want to assign null value use ? operator
var a: String? = “Hello"
a = null // OK
```

- Way – 2 – Nullable Check
  - The Kotlin system will tell you an error if you want to call a method from a nullable variable. Always check before accessing whether it is null or not.

```
var a: String? = “Hello"
val l = if (a != null) a.length else -1
```

Way to handle Null Safety in Kotlin

- Way – 3 - Safe Calls (?.)
  - The safe call operator returns the variables property only if the variable is not null, else it returns null. So the variable holding the return value should be declared nullable.

```
fun main(args: Array){
    var b: String? = "Hi !"     // variable is declared as nullable
    var len: Int?
    len = b?.length
    println("b is : $b")
    println("length is : $len")

    b = null
    len = b?.length
    println("b is : $b")
    println("length is : $len")
}
```

Result for the Code

```
b is : Hi !
length is : 4
b is : null
length is : null
```

Way to handle Null Safety in Kotlin

- Way – 4 – Elvis Operator (?:)
  - If reference to a variable is not null, use the value, else use some default value that is not null.

    This might sound same as explicitly checking for a null value.

```
fun main(args: Array){
    var b: String? = " David" // variable is declared as nullable
    val len = b?.length ?: -1
    println("length is : $len")
    val noname = b?: “No one knows me"
    println(“Name is : $noname")
}
```

Result for the Code

```
length is : 5
Name is : No one knows me
```

The !! Operator (not-null assertion operator)

- Despite the safety measures Kotlin provides to handle NPE, if you need NPE so badly to include in the code use !! operator.
- You can use this operator, if you are 100% sure that variable holds a non null value, or else you will get NullPointerException.

```
fun main(args: Array){
    var b: String? = "Hello"     // variable is declared as nullable
    var blen = b!!.length
    println("b is : $b")
    println("b length is : $blen")

    b = null
    println("b is : $b")
    blen = b!!.length // Throws NullPointerException
    println("b length is : $blen")
}
```

```
Result for the code
b is : Hello
b length is : 5
b is : null

Exception in thread "main" kotlin.KotlinNullPointerException
    at ArrayaddremoveKt.main(Arrayaddremove.kt:9)
```

Class and Object [Example – 1]

- **Simple Java class Person**

```
public class Person {
 private String name;
  public Person(String name) {
        this.name = name;
 }
public String getName() {
return name;
}
public void setName() {
this.name = name;
}

}
```

- **Simple Kotlin class Person**

```
//If you have only one constructor

class Person(val name: String)

//If you have many constructor

class Person {
  val name:String
  val age:Int
  constructor(name:String) {
    this.name = name
  }
  constructor(name:String, age:Int) {
    this.name = name
    this.age = age
  }
}
```

Note : No need of semicolon in the end of the line

Constructors [Example – 2]

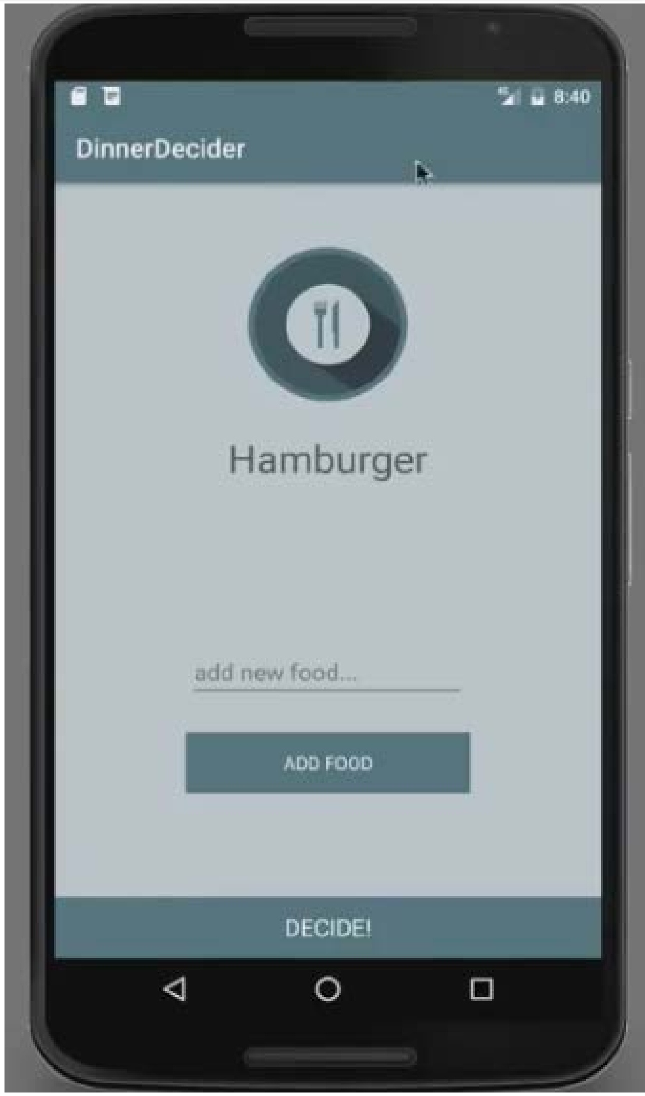
- There could be only one **Primary constructor** for a class in Kotlin.
- The primary constructor comes right after the class name in the header part of the class.
- Constructors that are written inside the Body of Class are called **Secondary constructors**.
- Secondary Constructor should call primary constructor or other secondary constructors using **this** keyword.

```
fun main(args: Array<String>){
    var person_1 = Person(“David",25, "Teaching")
    person_1.printPersonDetails()
}

// Kotlin Primary Constructors
class Person constructor(var name: String, var age: Int){
    var profession: String = "Not Mentioned"

// Kotlin Secondary Constructors

    constructor (name: String, age: Int, profession: String):
this(name,age){
        this.profession = profession
    }

    fun printPersonDetails(){
        println("$name whose profession is $profession, is
$age years old.")
    }
}
```

Output : David whose profession is Teaching, is 25 years old.

Kotlin – Dinner Decider App

- If the user clicks the DECIDE! Button, randomly select the stored values from the ArrayList and then change the label text with the random value.
- If the user is not happy with the default vales, they could add the food by typing in the EditText component and add the value by clicking the ADD FOOD button.



MainActivity.java

```
class MainActivity : AppCompatActivity() {
// Initialize the arraylist with default values and no need to give ; at the end
    val foodList = arrayListOf("Chinese", "Hamburger", "Pizza",
"McDonalds", "Barros Pizza")
/*onCreate function declaration, no need for @ symbol on Override
 * methods starts with the keyword fun, argument and type separated
* by colon, Bundle? says may be Bundle object null */
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
```

onClick() implementation

```
// no need to configure ids from xml using //findviewByid(),
//directly you can access the ids
// decide the dinner
decideBtn.setOnClickListener {
        val random = Random()
        val randomFood = random.nextInt(foodList.count())
        selectedFoodTxt.text = foodList[randomFood]
    }

// adding food to the list
    addFoodBtn.setOnClickListener {
        val newFood = addFoodTxt.text.toString()
        foodList.add(newFood)
        addFoodTxt.text.clear()
    }
```