Android App Development with Kotlin

Range & When

1

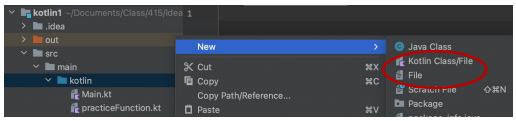
1

Functions

- Defining functions and methods, you are required to **specify the type of the parameters**.
- Unlike Java, Kotlin doesn't require a statement or expression to belong to a method and a method to belong to a class.
- However, when the code is compiled, or executed as script, **Kotlin will create Wrapper classes and methods** as necessary to satisfy the JVM expectations.

2

2



Let's create a new kotlin file to practice functions
(kotlin → New → File) OR
(kotlin → New → Kotlin Class/File → File)

You can name it whatever you want (e.g., `practiceFunction.kt`)

Don't forget to call it from function `main()`

```
fun main() {
    practiceFunction()
}
```

```
fun practiceFunction()
{

}
```

3

3

KISS (Keep It Simple, Silly)

- `return` keyword is **NOT** allowed for **single-expression functions**

```
fun greet() = "I am Groot"
println(greet())
```

- Leave out the return type if it's obvious, specify it otherwise

```
fun greet(): String = "I am Groot"
val message: Int = greet()   // ERROR
```

- Kotlin uses a special type called **Unit** (**void** type of Java)

```
fun sayHello() = println("Well, hello")
val message: Unit = sayHello()
// what does this print?
println("The result of sayHello is $message")
```

4

4

- Kotlin **forces** you to choose between `val` (`Final` in `Java`) and `var` when defining local variables.
- But when defining the function, they are **implicitly val**
  - Any effort to change the parameters' values within functions or methods will result in compilation error

5

5

Don't mix single expression function **=** with block body **{}**

```
fun f1() = 2
fun f2() = { 2 }// fun f2():()->INT={2}


println(f1())    //2
println(f2())    //Not working properly
println(f2()())  //2
```

6

6

Don't mix single expression function = with block body {}

```
fun f3(factor:Int) = {n: Int -> n * factor }


println(f3(2))    //Not working properly
println(f3(2)(3))//6
```

It's highly unlikely that someone who writes such code will have many friends – do not spend the rest of your life alone

7

7

Overloading functions

8

8

Overloading functions

```
println(greet())
println(greet("Eve"))

fun greet() = "I am Groot"
fun greet(name:String) = "Hello $name "
```

Let's test it!

9

9

Named Argument

```
fun createPerson(name:String, age:Int=1, height:Int, weight:Int){
    println ("$name $age $height $weight")
}
```

```
createPerson ("Jake", 12, 152, 43)     // Java style

// you can assign a value to a parameter's name
// in the function call
createPerson (name="Jake", age=12, height=152, weight=43)

// works, but not for readability purpose
// age has a default argument
createPerson (name="Jake", height=152, weight=43)
createPerson ("Jake", height=152, weight=43)
createPerson (height=152, weight=43, name="Jake")

createPerson (height=152, weight=43, "jake") // ERROR
```

10

10

vararg

```
fun greetMany (msg: String, vararg names: String){
    println ("$msg ${names.joinToString(", ")}")
}

greetMany("Hello", "Tom", "Jerry", "Spike")
greetMany("Hello")
greetMany(msg="Hello", "Tom", "Jerry", "Spike")
greetMany("Tom", "Jerry", "Spike", "Hello") // bad
greetMany("Tom", "Jerry", "Spike", msg="Hello") // error
```

- You can use `vararg` when a function takes more than one parameter. But **only one parameter may be annotated as `vararg`**
- The first argument binds to the first parameter and the remaining arguments are passed to the `vararg` parameter

11

11

```
// Return type is required
fun max1(numbers:IntArray): Int {
    var large = Int.MIN_VALUE
    for (number in numbers)
        large = if (number > large) number else large
    return large
}

fun max2(vararg numbers:Int): Int {
    var large = Int.MIN_VALUE
    for (number in numbers)
        large = if (number > large) number else large
    return large
}
```

Difference?
Write test statements (e.g., `print()`)

To create an array of values, use the `intArrayOf()` function that belongs to the `kotlin` package

12

12

```
println (max1(intArrayOf(1,5,2,12,7,3)))

// ERROR - Type mismatch Int vs. IntArray
println (max1(1,5,2,12,7,3))

// we can pass any number of arguments and
// Kotlin's type checking will ensure that
// the different arguments are of the right type
println (max2(1,5,2,12,7,3))
println (max2(1,5,2))
println (max2()) // can be zero parameter passing
println (max1()) // ERROR

// ERROR - Type mismatch Int vs. IntArray
println (max2(intArrayOf(1,5,2,12,7,3)))
```

13

13

Spread Operator *

- By prefixing it with the spread operator *, you may pass an array.
- You are asking Kotlin to spread the values in that array as discrete values for the `vararg` parameter

```
println (max2( *intArrayOf(1,5,2,12,7,3)))
```

How about Python?

14

14

Destructing

```
// Java Style
val result = getFullNames()
val first = result.first
val second = result.second
val third = result.third
println("$first $second $third")

// Kotlin Style
val (fst, snd, trd) = getFullNames()
println("$fst $snd $trd")
```

Note that it is not `Tuple()` in Python

```
fun getFullNames() = Triple("John","Quincy","Adams")
```

Python, C++, Haskell do this

15

15

16

16

Range and Iteration

- Imagine telling someone to count from one to five by
  - "set i equal to 1, but while keeping i less than 6, increment i and report the value"

17

17

```
// Range Classes
val oneToFive: IntRange = 1..5
val aToE: CharRange = 'a'..'e'

println (oneToFive)
println (aToE)

val seekHelp: ClosedRange<String> = "hell".."help"
println(seekHelp.contains("helm"))
println(seekHelp.contains("helq"))
println(seekHelp)
```

18

18

```
// Forward Iteration
for (i in 1..5)             print("$i ")
println()
for (ch in 'a'..'e')        print(ch)
println()
// Reverse Iteration
for (i in 5.downTo(1))      print("$i ")
println()
for (i in 5 downTo 1)       print("$i ")
println()
```

```
// Skipping Values in Range
for (i in 1 until 5)              print("$i ")
println()
for (i in 1 until 10 step 3)      print("$i ")
println()
for (i in 10 downTo 0 step 3)     print("$i ")
println()
```


19

```
// filter() method takes a predicate
// (a lambda expression) as argument (will cover later)

for (i in (1..9).filter {it % 3 == 0 || it % 5 == 0})
    print("The number is $i, ")
```

20

Iterating over Arrays and Lists

```
// Java code
for (String name: names){
     System.out.println(name);
}
```


21

```
val array = arrayOf (1,2,3)
for (e in array) print("$e ")
println(array.javaClass)

val list = listOf (1,2,3)
for (e in list) print("$e ")
println(list.javaClass)
```

- To create an array of values, use the `arrayOf()` function that belongs to the `kotlin` package

22

```
val names = listOf("Tom", "Jerry", "Spike")

for (index in names.indices)
    println("Position of ${names[index]} is $index")

// get both index and value
for ((index, name) in names.withIndex())
    println("Position of $name is $index")
```

- call `withIndex()` method on an instance of `ArrayList` obtained from the JDK using Kotlin's `listOf()` method.
- Kotlin adds many additional functions (methods) to various existing Java collections.

23

24

When

- Kotlin does **not have** `switch`; instead, it has `when`

```
// Block Structure - NOT Kotlin style
fun isAlive1(alive: Boolean, numberOfLiveNeighbor: Int): Boolean{
    if(numberOfLiveNeighbor < 2) return false
    if(numberOfLiveNeighbor > 3) return false
    if(numberOfLiveNeighbor == 3) return true
    return alive && (numberOfLiveNeighbor == 2)
}
// functional style - Kotlin style
fun isAlive2(alive: Boolean, numberOfLiveNeighbor: Int) = when {
    numberOfLiveNeighbor < 2 -> false
    numberOfLiveNeighbor > 3 -> false
    numberOfLiveNeighbor == 3 -> true
    else -> alive && (numberOfLiveNeighbor == 2)
}
```

25

25

When

- Kotlin does **not have** `switch`; instead, it has `when`

```
// Block Structure - NOT Kotlin style
fun isAlive1(alive: Boolean, numberOfLiveNeighbor: Int): Boolean{
    if(numberOfLiveNeighbor < 2) return false
    if(numberOfLiveNeighbor > 3) return false
    if(numberOfLiveNeighbor == 3) return true
    return alive && (numberOfLiveNeighbor == 2)
}
// functional style - Kotlin style
fun isAlive2(alive: Boolean, numberOfLiveNeighbor: Int) = when {
    numberOfLiveNeighbor < 2 -> false
    numberOfLiveNeighbor > 3 -> false
    numberOfLiveNeighbor == 3 -> true
    else -> alive && (numberOfLiveNeighbor == 2)
}
```

When does this function return true?

Further refactor with **single expression function**

26

26

```
// functional style - Kotlin style
fun isAlive2(alive: Boolean, numberOfLiveNeighbor: Int) = when {
    numberOfLiveNeighbor < 2 -> false
    numberOfLiveNeighbor > 3 -> false
    numberOfLiveNeighbor == 3 -> true
    else -> alive && (numberOfLiveNeighbor == 2)
}
```

```
// further refactored
fun isAlive3(alive: Boolean, numberOfLiveNeighbor: Int) =
  (alive && (numberOfLiveNeighbor == 2)) || (numberOfLiveNeighbor == 3)
```

Let's test it!

Don't forget to add print statement for testing

27

27

when as an expression

```
fun whatToDo(dayOfWeek: Any) = when (dayOfWeek){
  "Sat", "Sun" -> "Relax"
  in listOf("Mon","Tues","Wed","Thur") -> "work hard"
  in 2..4 -> "work hard"
  "Fri" -> "Party"
  is String -> "What?"
  else -> "No Clue"
}
```

Kotlin Type Checking at runtime - `is`

Java: `instanceOf()` Python `isinstance()`

28

28

when as an expression

```
fun whatToDo(dayOfWeek: Any) = when (dayOfWeek){
  "Sat", "Sun" -> "Relax"
  in listOf("Mon","Tues","Wed","Thur") -> "work hard"
  in 2..4 -> "work hard"
  "Fri" -> "Party"
  is String -> "What?"
  else -> "No Clue"
}
```

Test cases:

"Sun"
"Wed"
"3"
"Fri"
8
"Another day"

Let's test it! (add print statements)

29

29

when as a statement

```
fun printWhatToDo(dayOfWeek: Any){
  when (dayOfWeek) {
    "Sat", "Sun" -> println("Relax")
    in listOf("Mon","Tue","Wed","Thur") -> println("work hard")
    in 2..4 -> println("work hard")
    "Friday" -> println("Party")
    is String -> println("What?")
    else -> println("No Clue")
  }
}
```

Let's test it!

Test cases:

"Sun"
8

30

30

when and variable Scope

```
// numberOfCores in function scope

fun systemInfo1(): String{

  val numberOfCores = Runtime.getRuntime().availableProcessors()

  return when (numberOfCores){
      1 -> "1 core, packing this one to the museum"
      in 2..32 -> "You have $numberOfCores cores"
      else -> "$numberOfCores cores!, I want your machine"
  }
}
```

31

31

when and variable Scope

```
// better: limiting scope for variables is good design
// numberOfCores in when scope

// directly return the result of when and
// remove the outer block {} and return

fun systemInfo2(): String =
  when (val numberOfCores = Runtime.getRuntime().availableProcessors()){
      1 -> "1 core, packing this one to the museum"
      in 2..32 -> "You have $numberOfCores cores"
      else -> "$numberOfCores cores!, I want your machine"
  }
```

32

32

```
public class NoName {
   public static void main(String[] args) {
       for (int number = 1; number < 101; number++) {
           if (number % 15 == 0) {
               System.out.println("DingDong");
           } else if (number % 3 == 0) {
               System.out.println("Ding");
           } else if (number % 5 == 0) {
               System.out.println("Dong");
           } else {
               System.out.println(number);
           }
       }
   }
}
```

Java: What does this do?

33

33

```
def fizzBuzz():
    for number in range(1, 101):
        if number % 15 == 0:
            print("FizzBuzz")
        elif number % 3 == 0:
            print("Fizz")
        elif number % 5 == 0:
            print("Buzz")
        else:
            print(number)
```

Python

Implement FizzBuzz in Kotlin using **when**
(e.g., FizzBuzz.kt)

34

34

FizzBuzz.kt

```
fun fizzBuzz() {
  for (number in 1..100) {
     when {




     }
  }
}
```

35

35

FizzBuzz.kt

```
fun fizzBuzz() {
  for (number in 1..100) {
     println(
        when {
           number % 15 == 0 -> "FizzBuzz"
           number % 3 == 0 -> "Fizz"
           number % 5 == 0 -> "Buzz"
           else -> number
        }
     )
  }
}
```

```
fun fizzBuzz() {
  for (number in 1..100) {
     when {
        number % 15 == 0 -> println("FizzBuzz")
        number % 3 == 0 -> println("Fizz")
        number % 5 == 0 -> println("Buzz")
        else -> println(number)
     }
  }
}
```

36

36

```java
public class Main {
   public static void main(String[] args) {
       Scanner scan = new Scanner(System.in);
       Random random = new Random();
       int n = 1 + random.nextInt(10);
       int guess = 0;
       System.out.printf("Guess a number from 1 to 10");
       while (true) {
           System.out.print("Your guess: ");
           guess = scan.nextInt();
           if (guess < 1 || guess > 10) {
               System.out.println("Try again");
           } else if (guess > n) {
               System.out.println("Too high");
           } else if (guess < n) {
               System.out.println("Too low");
           } else {
               System.out.println("Correct!");
               break;
           }
       }
}}
```

Java

What does this do?

37

37

```python
import random
n = random.randint(1, 10)
print("Guess a number from 1 to 10”)
guess = 0
while True:
   try:
       guess = int(input("Your guess: "))
   except ValueError:
       print("Try again")
       continue
   if guess < 1 or guess > 10:
       print("Try again")
   elif guess == n:
       print("Correct!")
       break
   elif guess < n:
     print("Too low")
   else: print("Too high")
```

Python

Implement this in Kotlin using **when**
(e.g., GuessNumber.kt)
Random.netInt(10)

38

38

```kotlin
import kotlin.random.Random

fun guessNumber() {
   val n: Int = 1 + Random.nextInt(10)




   }
}
```

You can convert `String` to `Int` using function
`readLnOrNull()?.toInt()`

Kotlin: use when

39

39

```kotlin
import kotlin.random.Random

fun guessNumber() {
   val n: Int = 1 + Random.nextInt(10)
   println("Guess a num from 1 to 10\n")
   while (true) {
       print(" Your guess: ")
       when (readLnOrNull()?.toInt()) {




       }
   }
}
```

Kotlin: use when

You can use keyword `return` to break out of loop

40

40

```kotlin
import kotlin.random.Random

fun guessNumber() {
   val n: Int = 1 + Random.nextInt(10)
   println("Guess a number from 1 to 10\n")
   while (true) {
       print(" Your guess: ")
       when (readLnOrNull()?.toInt()) {
           n ->  { println("Correct!") ; return }
           in n + 1 .. 10 ->  println("Too high")
           in 1 .. n - 1  ->  println("Too low")
           else           ->  println("Try again")
       }
   }
}
```

GuessNumber.kt

41

41