# Learn Kotlin by Doing

<https://play.kotlinlang.org/byExample/01_introduction/02_Functions>

# Kotlin Official Docs

<https://kotlinlang.org/docs/getting-started.html>

## Data Types in Kotlin

Byte – 8 bits

Short – 16 bits

Int – 32 bits

Long – 32bit

Float – 64 bit

Double – 64 bits

Char – 4bit

Boolean – 1 bit

And Custom data types like classes

## Var vs Val

Var is mutable / can be modified while val is like constant, it value cannot be changed.

# If Expression

Unlike Java and so many other languages, if can be used in expressions in Kotlin.

The “else” branch is mandatory when using “if” in expression.

# Class and Inheritance

* By default, all classes in Kotlin are final i.e non-inheritable. Use “open” modifier

# Data Types in Kotlin

* All data types in kotlin are by default “Object”. There is no primitive data type in Kotlin.
* There are no default values for any data type. All types must be initialized. Compile time error otherwise.
* **var** is **mutable,** which means its value can be changed later
* **val** is **immutable**, which means its value cannot be changed later. It defines constant values.

# String Templates/Interpolation

* adding values of variables of adding expressing to strings using $ sign.

# If Conditon

* if condition is used as expression, it can return some value. If **if** is used as expression then **else** clause is necessary

var maxValue : Int = if( a > b )

a else b

# When

* “when” is used in place of “switch” in Kotlin
* Ranges can be used in when statement
* When can be used as expression

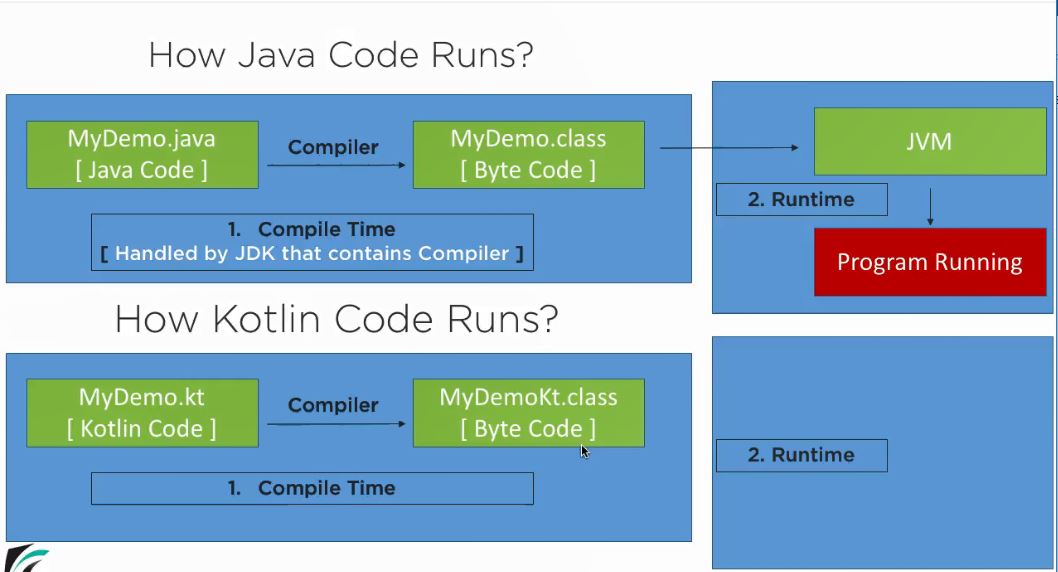
**var** result : String = **when**(a){  
  
 5,6,7 -> {  
 **"entered value is 5, 6 or 7"** }  
  
 **else** ->  
 **"something else is entered"**}  
 *println*(result)

# Functions

* Functions as expression, expression always return a value
* *// this is funciton as expression, expression always return a value***fun** max(value1: Int, value2: Int): Int = **if** (value1 > 2) {  
   *println*(**"value1 is greater"**)  
   value1 *//last statement returns the value*} **else** {  
   *println*(**"value2 is greater"**)  
   value2 *//last statement returns the value*}

# Interoperability

* You can call any Java function from Kotlin and vice versa
* You can have Java and Kotlin files within the same project
* Compiler converts java code into byte code and the jvm runs this
* Same thing happens for kotlin code



* All function in a Kotlin file are by default static, because it is not a class, so it can not have an instance, so it has to call method statically

# Default Functions

* A function accepting default value for any parameters is called default function
* Kotlin support default function, Java does not.
* To call a default function from a java class, we need to annotate the function with annotation @JvmOverloads
* We can call default function without providing value for default parameter
* If we give value for default parameter, it will override the default value
* Default arguments are the arguments having default values

# Named Parameters

* Its is a pure Kotlin feature, not present in Java, so there is not interoperability in case of named parameters
* **fun** area(widht:Int,breadth:Int, height:Int=3):Int{  
   **return** widht\*breadth\*height  
  }

and now calling function with named paramters

*area*(breadth = 2,height = 3,widht = 5)

as you see, when using named parameters, sequence of parameters in not necessary

# Primary Constructors

* You can define properties in primary constructor by defining var or val keyword
* Init block servers as body of constructor and are called with the creation of objects in the sequence in which they are declared

# Secondary Constructor

* Secondary constructor cannot have var or val keywords; i.e they cannot declared properties
* Secondary constructor must have to call primary constructor first
* The body of secondary constructor is executed of inti block execution

# Inheritance

* In Kotlin, all class are “public” and “final” by default
* Because class is “final” by default, you cannot inherit it by default, use keyword “open” to make class inheritable
* Any is super class for all class, just like in java Object class is super class for all classes

# Abstract

* All abstract functions/classes are “open” by default, if you use keyword “abstract”, you don’t have to use “open” keyword
* Abstract methods can only reside in abstract classes
* An Abstract class cannot be instantiated
* Sub classes must have to override/implement abstract methods of its super class
* Abstract properties cannot be initialized in their own class, they are meant to be implemented/initialized in sub class.

# Interfaces

* Properties in interface are abstract in nature
* Normal function are **public** and **open** by default, not **Final** like in Classes
* Interface cannot be instantiated
* When two different interfaces have a same normal (not abstract) method with same name say “onClick”, and a class implements both , the class must have to override that method “onClick”, and in its function definition , it can call super method with its type like

**super<interface1>.onClick()** or **super<interface2>.onClick()**

* When two different interfaces have a method with same name “onClick”, but in one interface, it is abstract function and in other interface it is normal function, then the instance of implementing class will by default call the interface with normal function, not the abstract function.

# Object Declaration / Singleton / Alternative to Static

* In kotlin, we can define object without defining its class. That object will be a single object of its type in whole application, and serves as “Singleton”.
* Object can extend a class and access its properties and functions in it.
* All properties and functions of “object declaration” acts behave like static, but they are not actually static. They are accessed with the name of “object declaration” i.e surrounding object
* Object declaration outside of any class is used to serve the purpose of **Singleton**

**object** MyNetworkingLib{  
 **var url**=**""  
  
 fun** makeNetworkCall(){  
 *println*(**"making network call to $url"**)  
 }  
}

**and then calling the object**

**fun** main(orgs: Array<String>){  
  
 MyNetworkingLib.**url**=**"www.mywebsite.com"** MyNetworkingLib.makeNetworkCall()  
}

# Companion Object

* Companion object is just like “object declaration” outside the class, the only different is difference is that companion object resides inside the class and acts like a member of the class
* It serves the purpose of “static” like in java
* Its members (properties and function) are accessed with its surrounding class name
* Companion object can any class
* When the first ever object of the class is created, first of all companion object’s body is executed and then init block is executed, companion object is not called with the creation of subsequent objects
* Because with the creation of first ever object, class is loaded to memory and static members are loaded to memory, since the companion object servers the purpose of static , so it is loaded to memory first of all.
* For java compatibility , annotate each function in companion object with @JvmStatic

# High Level Function/ Higher Order Function

* A function that can accept another function as parameter or can return a function or can do both

# Ranges

* var x = 1..5 //this range contains values 1,2,3,4,5
* downTo is also Range but in reverse order
* var x = 5 downTo 1 //this range contain values in reverse order e.g 5,4,3,2,1
* var xy = 10 downTo 1 setp 2 //this range contains values 10,8,6,4,2
* var var3 = “a”.. “z” //this range contains values “a”, “b”,”c”,…..,”z”
* var ifExist = “a” in var3 //returns Boolean values
* var moveDown = 10.downTo(1) //contains values 10,9,8,…..,1
* var moveUp = 1.rangeTo(10) //contains values 1,2,3,…,10

Ranges are helpful while running loops in kotlin

# Labeled for loop and Break statement

* by default ***break*** breaks in immediate loop.
* If you are using inner loops and break the outer loop the you can add label as following

myLoop@ **for** (i **in** 1..10){  
 **for** (j **in** 1..10){  
 *println*(**"$**i **$**j**"**)  
 **if**(i==2 && j==2){  
 **break**@myLoop  
 }  
 }  
}

# Labeled for loop and Continue Statement

* By default **continue** statement continues the immediate loop
* If you want to continue with an outer loop, you can add label with that loop as following

myLoop@ **for** (i **in** 1..10){  
 **for** (j **in** 1..10){  
 *println*(**"$**i **$**j**"**)  
 **if**(i==2 && j==2){  
 **continue**@myLoop  
 }  
 }  
}

# How to makes singleton in Kotlin

* **Object** keyword is used to create singleton in kotlin
* You can access members fields or method of object with the name of object
* All the members of object are not static, but actually behaves like static
* In Java, to make singleton, we need to make constructor private and static method to access constructor, but kotlin does this behind the scene, when you declare **object**
* Behaves like static means, members are accessed with the name of object like in java, members are accessed with the name of class

# Companion object

* Companion object is just like object but it is declared inside the class
* When you declare companion object inside the class, no need to mention the name of object
* Members of companion object are actually static in nature,
* Member of static object are accessed with the name of class