**Kotlin**

**Day 1**

1-if you don’t want **to count wrong answer**, you can use (try and catch) inside while loop and add the incrementation at the end of (try)

So, if there is any mistake the try will crash before the incrementation happen.

Ex:

while(a<3) {

try {

User should Enter Number

If user enter String, the program will go to catch, and won’t do the next line

a++

}

catch {

}

}

2-you can use **(when)** in this way:

when {

A==5 -> do something

10<b<0 -> out of range

else -> do something

}

Or (the same as switch in Java):

When (x){

1 -> do something

15-> do something

Else -> {something 1

Something 2

}

3-I tried with var **Str=readLine(): 🡺** Str=String (Input Type String)

1. Str.equals(“STRING”)
2. Str == “STRING”

And it all works

4-It is hard to make (for loop) infinite. So instead, if you want **infinite loop** you should use (While Loop)

1. for (I in 1..3)
2. While (true)

5-**var num=readLine()!!.toInt()** 🡺 num=Integer (Change Input to int)

You should use try and catch to catch Errors. 🡺 Ex: (User Enters String)

6-to ask computer for **Random Number** we use:

var x=Random.nextInt(11) 🡺 it will give us number between (0 and 10) 11 not included

or

val x=Random.nextInt(7,15) 🡺 it will give us number between (7 and 14) 15 not included

7**-**difference between **var and val**:

var: can be change. 🡺 var x=15 🡺 x=10 (correct)

val: can’t be change. 🡺 val x=15 🡺 x=10 (Wrong)

8-**String**:

var str=”Hello”

* if(str[0].isUpperCase()) 🡺 true or false 🡺 **Check 1st litter if Upper Case**
* for (i in str) println(i) **🡺 print all character** in the String alone
* Str.equals(“String”) 🡺 true or false 🡺 compare 2 Strings

9-**Any:**

* It’s a variable type
* Is can be any type (String, Int, Array, etc…)
* Most use with arrays and maps so **they can contain any type**
* Ex: var x:Any =5 🡺 x=”hit” 🡺 x=’s’

**Day 2**

1-we could **use readLine with String**:

* var str=readLine()!!.toString()
* var str=readLine()!!
* var str=readLine().toString()

2-we could **check the readLine Entry** by using if condition

* if(!readLine().isEmpty())
* if(!readLine().isBlanck())

3-The **difference between ArrayList and arrayListOf**

When we initialize empty 2D Array we can write:

-ArrayList<ArrayList<String>>()

-arrayListOf<ArrayList<String>>()

but we can't put the arrayListOf inside:

ex: arrayListOf<arrayListOf<String>>()

because arrayListOf is function, and inside these brackets <> only accept Types ex:(String) or classes ex:(ArrayList)

4-we can use these **functions with the list and arrays**

* list.sort() 🡺 to sort numbers
* list.size == list.count() 🡺 to know the length
* list.sum() 🡺 to sum the numbers inside the array

**Ex: val avrg = list.sum().toFloat()/list.size**

5-we **add to the 2D array** using

arary.add(*arrayListOf*(“item1”,”item2”))

6-i can use **print method with 2D array using for loop** inside for loop:

for(i in answers) {  
 for (c in i)  
 *print*("$c ")  
 *println*()  
}

The (i) will take the elements inside the answers. 🡺 i= arrayListOf.

The (C) will take the elements inside the (i). 🡺c= item1 , item2.

So, it like this: Ansewrs 🡺 i 🡺 c

Answers[0] 🡺arrayListOf 🡺 [item1,item2]

Answers[1] 🡺arrayListOf 🡺 [item1,item2]

Etc…

**Or**

for(i in answers)  
 *println*("Country ${i[0]} its Capital ${i[1]}")

Since (i) is arrayListOf, we can Print its Elements.

Don’t Forget To **Use The {} inside the String** to Print The Array

**Or**

println(answer[0][0])

7-**Always use float with numbers** 😊

8-to **enter char (single char – single number)** can’t enter two numbers (55)

val c=*readLine*()!!.*single*()

9-**Functions Can Be Written outsit the main**. Anywhere before or after

**Functions also can be Written inside the main**, however you should write the function on the top of the main Before you call it. Because if you write the function last, and you call it first there will be an error because the compiler couldn’t compile the function yet.

10-we could **use list or arrays** this way: 🡺 **to chose random color** from the list.

* val colors = listOf("Red", "Blue", "Green", "Yellow")
* var color = colors[Random.nextInt(colors.size)]

this code will save random color from the colors list

11-**Fuctions**:

**Ex1:**

fun sum (num1:Float , num2:Float) : Float 🡺 num1 and num2=**Parameters**. **|** :Float 🡺 **return** Type

{

return num1+num2

}

**Ex2:**

fun sum (num1:Float , num2:Float=0) : Float 🡺 if user didn’t send num2 value, **the Default Value** =0

13- difference between **array, list and arrayList:**

* list 🡺 can’t change elements – can’t Add – can’t Delete
* array 🡺 **can** Change Elements – can’t Add – can’t Delete
* arrayList 🡺 **can** Change Elements – **can** Add – **can** Delete

they all can **hold** **any type**. 🡺 ex: var x=listOf(5,”hi”,true)

if we want to change elements Better to identify **type = Any** 🡺 ex: var x=arrayListOf<Any>(“hi”,1)

to **print them**:

* print(list) 🡺 will print all elements
* for (i in list) print(i) 🡺 print each element of list
* for (i in array) print(i) 🡺 print each element of array
* for (i in arrayList) print(i) 🡺 print each element of arrayList

to **change Elements**: 🡺 array[0]=newElement

to **add Element**: 🡺 arrayList.add(NewElement)

to **Delete Elements by value**: 🡺 arrayList.remove(ElementValue) 🡺 ElementValue= “Name”

to **Delete Elements by Index**: 🡺 arrayList.remove(arrayList[Index])🡺arrayList[Index]=ElementValue

14-**var x:Int** 🡺initialize variable x with type Int with no value

**Day 3**

1-To **Compare 2 Strings with ignore case**: 🡺 true = ignore case

“String”.*equals*(“String”,true)

2-we can **identify the variables inside the constructer in classes identifier**:

abstract class a(name:String){} 🡺 here we don,t need to use var or val because we aren’t gana use

it, also you can’t call the variable by the object.

class b(val name:String):a(name){} 🡺 here we used val because we are going to use name to send it

to class a, also can be called by object.

3-**2D arrayList**: 🡺 to make it **mix types elements** change **String to Any**

We can **specify the size of it** like this: 🡺 ex: 2D array ( **3\*3** )

**Ex1**: var array =arrayListOf<ArrayList<String>>()

array.add(arrayListOf(“item1”,”item2”,”item3”)

array.add(arrayListOf(“item1”,”item2”,”item3”)

array.add(arrayListOf(“item1”,”item2”,”item3”)

so we add **3 lines and in each line we have 3 items** 🡺 **3\*3**

we can do it in **initializing** the arrayList **without add function**

**Ex2:** var array=arrayListOf( arrayListOf(“item1”,”item2”,”item3”),

arrayListOf(“item1”,”item2”,”item3”),

arrayListOf(“item1”,”item2”,”item3”)

)

Array 2\*2

arrayListOf

arrayListOf

Item2

Item1

Item2

Item1

4-**for Loop:**

* for(x in “Helloe”) 🡺 x = each characters in “Helloe”
* for(x in array) 🡺 x= each element in array
* for(x in 1..10) 🡺 x= numbers from 1 to 10
* for(x in 1 until 11) 🡺 x = numbers from 1 to 10 (11 not included)
* for(x in 0..array.size-1)==for(x in 0 until array.size) 🡺 x = index of array = 0 to array.size-1

(last number (array.size) not included)

5-**Dictionary = Map**

* **Initialize**: val dictionary=mapOf(‘key’ to ‘value’)

🡺 key = any number or string or both.

🡺 value = any type (Int, Float, String, arrayListOf)

* Printing **unsexist key** will print **null**
* **.getOrDefault** will print default value if key has no value
* **.keys** call all keys
* **.containsKey(key)==.contains(key)** 🡺 true or false 🡺 check if key exist or not
* **.containsValue(value)** 🡺 true of false **🡺** check if dictionary has this value or not
* We **can’t add, change or delete** to the map
* **Ex:**

|  |  |  |
| --- | --- | --- |
| val dec= *mapOf*( 1 *to arrayListOf*(1,5) , "a" *to* 5 , 6 *to* "jhg" , 5 *to* 1..5 ) | | |
| **Print Method** | **OutPut** | **Notes** |
| for (i in dec)  *println*("$i\t") | 1=[1, 5]  a=5  6=jhg  5=1..5 | Print each key with elements  in different rows |
| println(dec) | {1=[1, 5], a=5, 6=jhg, 5=1..5} | Print all Keys with Elements |
| println(dec["a"]) | 5 | Print key (5) value |
| println(dec.keys) | [1, a, 6, 5] | Print all Keys only |
| for(i in dec.keys)  println(dec[i]) | [1, 5]  5  jhg  1..5 | Print Elements on different row  Using Keys |
| println(dec[2]) | null | No (2) key |
| println(dec.getOrDefault(2,"Hi")) | Hi | Print default since no (2) key |
| println(dec.containsKey(5)) | true | Check key (5) |
| println(dec.contains("jhg")) | false | Check key (“jhg”) |
| println(dec.containsValue("jhg")) | true | Check element (“jhg”) |

6-**difference between Map and MutableMap:**

* Map 🡺 can’t change – can’t Add – can’t Remove
* MutableMap 🡺 can Change – an Add – can Remove

Ex: val MM=mutableMapOf(key to value)

MM[old key]=new value 🡺 Change value

MM[new key]=new value 🡺 Add new Key with Value

MM.remove(key) 🡺 remove Element

* If we **initialize 2 same keys** the program will take the value of last key was initialized

Ex: val MM=mutableMapOf(1 to “hi”, 2 to “bye”,1 to “s”) 🡺 1 = “s”

* If we **remove unsexist key** the program won’t do anything and will continue
* To **initialize empty MutableMap** we identify Type (Any, String , Int etc…)

Ex1: val MM=mutableMapOf<Int,String>() 🡺 key type Int – elements type String

Ex2: val MM=mutableMapOf<Any,Any>() 🡺 keys and elements type Any

* We can **use for loop to take keys and elements in variable**

Ex: for ((key,value) in mutablePam) 🡺 key = keys – value = elements

7-we can **write the classes inside the main**.

Ex: **fun main (){**

abstract class animal (name:String){ 🡺name == constructers

abstract fun speak ()

}

class dog (val name : String) : animal { 🡺 name = constructers 🡺 : inherits animal class

name = name 🡺 the name from animal class will take the name from dog class

override fun speak () { 🡺 should write override for inheriting the fun

println(“Wolf”)

}

}

}

8- to make dog **class shorter**:

class dog (val name : String) : animal (name) { 🡺 name will be sent to animal class to save it

}

**Android Studio**

1-If we want to **initialize variable outside onCreate fun** we should write: **lateinit** before the variable.

Ex: lateinit var number: TextView 🡺 number of type TextView

2-to **change color of Text View**:

Ex: textview.setTextColor(Color.*GREEN*)

3-to **initialize layout types** we use: findViewById<Type>(R.id.layoutID)

Ex: val add=findViewById<Button>(R.id.*Plus*)

4-to **call the color of the view text** we use: textview.currentTextColor

Ex:if (enrtey.currentTextColor==Color.GREEN) 🡺 true or false

5-to **change position of text view** we use: textview.setPadding(left,top,right,down) 🡺 all int

Ex: entery.setPading(0,0,0,10) 🡺 will move the text view up

6- <**TextView**  
 android:id="@+id/Numbers" 🡺 **Text View ID**  
 android:layout\_width="wrap\_content" 🡺 **layout size**  
 android:layout\_height="wrap\_content"  
 android:text="0" 🡺 **content of the text view**   
 android:textSize="100dp" 🡺 **change text size**  
 app:layout\_constraintBottom\_toBottomOf="parent" 🡺 **layout position**  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent" />

7- **<Button**  
 android:id="@+id/Mines"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_weight="1" 🡺 the space inside the layout  
 android:text="-" />

8- the **space inside the layout**: if there are many layouts, if all =1 they will take equal spaces. But if one =1 and the other = 2 then the other will take two third of the space.

Ex: android:layout\_weight="4"

9-**Snackbar**: is used to **show message to the user**, we can show it inside the main class uses constraint layout, But we do that we should initialize id for the constraint layout.

Ex: Snackbar.make(ConstraintLayoutName, "Please Enter Valid Value", Snackbar.*LENGTH\_LONG*).show()

.show used to show the message.

10-EditText: used to take input from user.

Ex: var input = EdetTextNsme.text.toString() 🡺 change the input into String

11-to **check if user enter something** or not we use **isNotBlank()**.

Ex:if (EditTextname.text.isNotBlank)

12-to make the **input field empty** after user input something we use **null**.

Ex: EditTextName.text = null

13-**<EditText**  
 android:layout\_width="0dp" **🡺 change the size to fit the space and stretch**, instead of using (wrap\_content) and (match\_parent)  
 android:background="#000000" 🡺 **change background color** to Black  
 android:hint="Please Enter Here" **🡺 show a hint to user** to know what to enter

android:textStyle="bold" 🡺 **add style to text**. Ex: **bold**  
 android:textColor="#FFFFFF" 🡺 **set color to input text** (white)  
 android:textAlignment="center" 🡺 make the text **centered**   
 app:layout\_constraintBottom\_toTopOf="@+id/rvMain" /> 🡺 make it **connected from bottom to another element top** id (rvMain)

14-**Lisener**: used with **anything needed for user reaction**. Ex: **Button – CheckBox – LayoutOnClick**

Ex: ButtonName.setOnClickListener **{** TextView.text = “Hello”**}** 🡺 will happen when user click button

15-**this**: refer to the same class that used in it.

Ex: this.title = “My App Name” 🡺 **changing the Name** of the app

16-**recreate**: used to **restart the app**.

Ex: this.recreate()

17-**intents**: used to **switch between classes**. We can use it with button to change to another class.

Ex: startActivity(Intent(this, TheCassThatWeWantToGoName::class.*java*))

Note: we need to **initialize all classes** in AndroidManifest.xml file.

Ex: <activity  
 android:name=".ClassName"  
 android:exported="true">  
 <intent-filter>  
 <action android:name="android.intent.action.MAIN" />  
 </intent-filter>  
 </activity>

18-**Toast**: the same as Snackbar. Used to **show message to user**.

Ex: Toast.makeText(this, "New Task Added Successfully", Toast.*LENGTH\_LONG* ).show()

Differences between Snackbar and Toast:

|  |  |
| --- | --- |
| **Toast** | **Snackbar** |
| Will stay up the whole full time until it finishes, even if we ask it again. After finish it will show the next asked message. | When ask again, the old message will disappear and the new message will show |
| Small, faded and gray background color | Big, bold and black background color |
| Uses this (class) to show the message | Uses layout to show message |

19-**Alert Dialog**: used the same as Message Dialog in Java. Used to **show Message Box** to user to alert the user or to **take Input from user** in Message Box.

Ex: val input = EditText(this) 🡺 imaginary EditText to take input  
input.*hint* = "Enter Task Here" 🡺 show hit to user for input  
dialogBuilder = AlertDialog.Builder(this) 🡺 initialize Alert  
dialogBuilder.setMessage("Please Enter Task to Do:") 🡺 set the Message for the alert dialog box  
 .setCancelable(false) 🡺 user can’t skip this box Message  
 .setPositiveButton("ADD") **{ 🡺** set the button namedialog,id **->** val str =input.*text*.toString() 🡺 taking input  
 if(str.*isNotBlank*()) { 🡺 check if input = null  
 toDoList.add(str) 🡺 add the input to arrayList  
 Toast.makeText( 🡺 showing toast Message to user  
 this,  
 "New Task Added Successfully",  
 Toast.*LENGTH\_LONG* ).show()  
 }  
 else  
 Snackbar.make( 🡺 showing Snackbar Message to user  
 outMainL,  
 "Please Enter Task",  
 Snackbar.*LENGTH\_LONG* ).show()  
 **}** .setNegativeButton("Cancel") **{**dialog,id **->** dialog.cancel()**}** 🡺 set another button to cancelval alert = dialogBuilder.create() 🡺 create the Alert Dialog  
alert.setTitle("Add New Task") 🡺 set Title to Alert Dialog  
alert.setView(input) 🡺 let the user see the input field  
alert.show() 🡺 Display the Alert Dialog

20-