10:00

# KOtlin

⇒ It is a Staticly typed programming language, developed by JetBrains.

⇒ Since 2011.

⇒ It is the primary development language for android.

⇒ Extention *.kt.

★ Hello world in Kotlin

```
fun main(){
    Println("Hello Kotlin")
}
```

★ Varibles

⇒ There are two types of varibles in Kotlin

→ Mutable varible

```
var name: String = "Nate"
```

→ Immutable Varible

```
val name: String = "Nate"
```

{Varible name} {Varible Type}

⇒ Named String is not nullable

Val name : String = null {Invalid Statement}

Val name : String? = null {Valid Statement}

↘ Nullable String

⇒ Kotlin can infer type of variable from RHS.

Val name = "Nate"

★ <u>If Statement</u>

```
if (greeting != null){
    Println(greeting)
}else {
    Println("Hi")
}
```

★ <u>When {Similar to Switch}</u>

```
When (greeting) {
    null → Println("Hi")
    else → Println(greeting)
}
```

⇒ Val greetingToPrint = if (greeting != null) greeting else "Hi"

⇒ Val greetingToPrint = When (greeting) {

```
    null → "Hi"
    else → greeting
}
```

## ★ Basic Functions

```
fun getGreeting(): String {
    return "Hello Kotlin"
}
```

**Unit** ⟶ Absence of any useful type {Similar to Void}

```
fun getGreeting(): String = "Hello Kotlin"
```

or | ⟶ {Single Expression function}

```
fun getGreeting() = "Hello Kotlin"
```

```
fun sayHello(itemToGreet: String) {
    val msg = "Hello" + itemToGreet
    Println(msg)
}
```

or

```
val msg = "Hello $itemToGreet"
```

**Top level Variable** ⟶ {Not declared within any Class or function}

**Top level function** ⟶ {Not declared within any class}

# ★ Collections & Iteration

```
Val interestingThings = arrayOf("Kotlin", "Programming"
                                , "Comic Books")
```

- Size
- []
- get(0)

```
for (interestingThing in interestingThings) {
    println(interestingThing)
}
```

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

{By Default}

or

{ lambda Syntax }

```
interestingThings.forEach { interestingThing ->
    println(interestingThing)
}
```

If you have a function and its only parameter is another function, then you can omit the parenthesis and define function there itself.

```
interestingThings.forEachIndexed { index, interestingThing ->
    println("$interestingThing is at index $index")
}
```

```
val alphabet = listOf("A", "B", "C")
```

* (Map)

```
val map = mapOf(1 to "a", 2 to "b", 3 to "c")
```

⇒ listOf, mapOf, arrayOf are immutable (i.e. you cannot add or delete members.

⇒ mutableListOf for mutable list.

↳ Similar for map & array.

List<String>

↳ list of String

* Varargs, named arguments & default parameter values

(# Varargs)

```
function sayHello (greeting: String, vararg itemToGreet: String){
    itemToGreet.forEach { itemToGreet ->
        println("$greeting $itemToGreet")
    }
}
```

{This will be treated as array of String}

```
sayHello ("Hi", "Kotlin", "Programming", "Comic Book")
```

Val interistingThings = arrayOf ("Kotlin", 'Programing", "Comic Book')

SayHello ("Hello", *interestingThings)

{Spread Operator}

#named argument

greetPerson (Greeting = "hi", name = "Nate")

#default parameter value

fun GreetPerson (greeting: String = "hello", name: String = "Kotlin")
= PrintIn ("$Greeting $Name")

# ★ Classes

Class Person (_firstName : String, _lastName: String) {
val firstName: String = _firstName
val lastName : String = _lastName
}

● val person = Person ("Nate", "Ebel")
person . firstName
person . lastName

OR

Class Person (Val firstName : String, val lastName: String)

⇒ Properties will get getters and setters generated automatically for them.

⇒ But we can edit the default behaviour!

```
var nickName: String? = null
    set(value) {
        field = value
        println("the new nickname is $value")
    }
    get() {
        println("the returned value is $field")
        return field
    }
```

⇒ `val nickNameToPrint = nickName ?: "No nickname"`

- `nickName` → { If nickName is not null }
- `?:` → { If nickName is Null }

⇒ In Kotlin visibility of Properties and method are public by default.

- Public → { Available Everywhere }
- internal → { Public within the module }
- Private → { Only available within the file it is implemented }
- Protected → { Only within that class or Sub Classes }

# ★ Interface


```
interface PersonInfoProvider {
    fun printInfo(Person: Person)
}

class BasicInfoProvider: PersonInfoProvider{
    override fun printInfo (person: Person) {
        Println ("PrintInfo")
    }
}
```

⇒ Imterface can have default implemantdion.

⇒ We also need to overite the propatio of imterface:

```
override val proviceInfo: String
    get() = "Basic Info Provider"
```

⇒ super.printInfo(person)

↳ To access this functio from Supa class.

⇒ You can check if a class implements a interface or not

```
if (infoProvide is SessionInfoProvider){
  //Todo
}
```

## * Inheritance

⇒ By default a class in Kotlin is not open for extention, to extend a class you need to add open Keyword in front of the class.

- ↳ Property should also be marked Open.

## * Object Expressions

```
Val provider = object : PersonInfoProvider{
    override val providerInfo : String
        get() = "New Info Provider"

    fun getSessionId() = "id"
}
```

→ {Anonymous Inner Class}

## * Companion Objects

```
Class Entity private constructor(val id: String) {

    companion object {
        fun create() = Entity(id:"id")
    }
}
```

→ {Companion objects have access to private properties & methods}

```
Val entity = Entity.Companion.create()
```

or

```
Entity.create()
```

⇒ Companion object:

```
Companion object Factory {
    - - - -
}

Val entity = Entity.Factory.Create()
```

## * Object Declarations

↳ Convient way to Create thread safe singleton within Kotlin.

```
Objec EntityFactory {
    fun Create() = Entity("id", "name")
}
```

## * Enum Classes

```
enum class EntityType {
    EASY, MEDIUM, HARD
}
```

{ It can also defined function }

```
fun GetText() = name
```

{ refers to the text }

## * Sealed Classes

⇒ Sealed class represent class hierarchie that Provide more control over inheritance.

- → All subclasses of a sealed class are known at compile time.
- → No other subclasses may appear after a module with the sealed class is compiled.

## * Data Class

⇒ You often create classes whose main purpose is to hold data.

⇒ On such classes, some standard functaility and utility function are often mechanicdly derived from the data.

⇒ In Koltin, these are called 'data classes' and are marked with data.

== → Value Comparision

=== → Reference Comparision

## ★ Extension Functions / Properties

```
fun Entity.Medium.printInfo(){
    println("Medium Class: $id")
}

val Entity.Medium.info: String
    get() = "Some info"
```

Extends Entity.Medium class

## ★ Advanced Functions

⇒ "Higher order functions are that which takes other functions as parameter or return functions."

```
fun printFilteredStrings(list: List<String>, predicate
                         : ((String) -> Boolean) ){
    list.forEach {
        if (predicate(it)) {
            println(it)
        }
    }
}
```