Kotlin

```
//This is single line comment

/* This is
Multi-Line
Comment */

println("Hello World")            //The code out of main function will not be executed

fun main(){                        //1st type of main function
  println("Hello World")}

fun main(args:Array<String>){          //2nd type of main function
   println("Hello World!!!")}

fun main(){
//Different Datatypes :-
Int = 10, 100, 0 , 85
Long = 9839701872
Float = 99.98f, 10.20F
Double = 9839.701872
Char = 'C', 'F'
String = "Stark","F","9839.701872","99.98f","9839701872","10"
Boolean = true, false

//var variables can be re-assign :-
var name1 = "Stark"  //Same as-  var name1: String = "Stark"
println(name1)
name1 = "Tony Stark"    //variable re-assigned
println(name1)

//val variables can't be re-assigned :-
val name2 = "Mark"  //Same as-  val name2: String = "Mark"
println(name2)
//name2 = "Zuckerberg"     //variable can't be re-assigned

//String Cancatenation :-
val num2 = 10.30f   //Same as-  val num2: Float = 10.30f
println(num2)
println("Number is: $num2")
println("Addition is: ${num2+10}") }
```

```
import java.util.*
fun main(){
   // Arithmetic Operators :-
   val a = 10
   val b = 7
   println(a + b)
   println(a - b)
   println(a * b)
   println(a / b)     // Int value/Int value -> Int
   println(a % b)     // % -> remainder

   // Comparison Operators :-
   val x = 10
   val y = 5
   println(x < y)
   println(x > y)
   println(x <= y)     // <= -> Smaller or Equal
   println(x >= y)     // >= -> Greater or Equal
   println(x == y)     // == -> Equal
   println(x != y)      // != -> Not Equal

   // Logical Operator :-
   val x2 = 10
   val y2 = 20
   val z2 = 30
   println( (x2>y2) && (x2<z2) )     // && -> and
   println( (x2>y2) || (x2<z2) )     // || -> or
   println( !(x2<y2) )     // ! -> not

   // Increment Operator :-
   var n = 10
   println(n++)  //Postfix
   println(n)
   println(++n)  //Prefix

   // Decrement Operator :-
   var n2 = 10
   println(n2--)  //Postfix
   println(n2)
   println(--n2)  //Prefix

   // Augmented Assignments :-
   var num = 27
   num += 3
```

```
println(num)
num *= 10
println(num)

// Getting User's Input by readLine() :-
println("Enter Something :-")
var name1: String? = readLine()  // Same as var name1 = readLine()
var age1: Int? = readLine()!!.toInt()
var price1: Float? = readLine()!!.toFloat()
println("User Input:$name1,$age1,$price1")

// Getting User's Input by Scanner Class :-
var sc = Scanner(System.`in`)    // Scanner Class Instance
println("Enter Something :-")
var name2 = sc.next()   // It takes string
var age2 = sc.nextInt()
var price2 = sc.nextFloat()
println("User Input: $name2,$age2,$price2")}

fun main(){
var str = "Stark"
println(str[2])

// Escape Sequence :-
println("New\nLine")
println("A\tTab")
println("Starn\bk")
println("Don't Erase\rme")

// Raw String :-
val application = """ Dear Sir/Mam
   I just have to say
   Good Bye!
   God Bless You!
""".trimMargin()
println(application)

// String Templates :-
val n = 10
val n2 = 100
println("n is: $n")
println("Addition is $n + $n2")    // this will not be added
println("Addition is ${n + n2}")
```

```
// String Functions :-
val lang = "Kotlin Language Java Language"
println(lang.uppercase())
println(lang.lowercase())
println(lang.length)
println(lang.trim())
println(lang.substring(2,5))}

fun main(){
// if expression :-
var a = 20
var b = 10
var max = 0
if(a > b){     // If condition is true then below code will be executed else it will not execute
   max = a
   println("Max value is ->  $max") }

// Second method of if expression without {}
if(a > b) println("a is Greater!")

// if-else expression :-
if (a > b){
   println("$a is greater than $b") }
else{
   println("$a is smaller than $b") }

// 2nd method :-
var max2 = if (a > b){
   a }
else{
   b }
println("Max2: $max2")

// 3rd method :-
var max3 = if (a > b) a else b
println("Max3: $max3")

// else-if :-
if (a > b){
   println("a is greater than b") }
else if(a < b){
   println("a is smaller than b") }
else if(a == b){
```

```
   println("a is equal to b") }

// when Expression :-
var x = 9
when(x){
   1 -> println("x is 1")
   2 -> println("x is 2")
   18 -> println("x is 18")
   3,4,5 -> println("x is 3 or 4 or 5")
   6,7 -> {
      println("x is Either 6")
      println("Or 7") }
   in 1..10 -> println("x is from 1 to 10")    // 1..10 means from 1 to 10
   else -> println("No values matched with x")}  }

fun main(){
// for loop :-
for(num1 in 1..5){
   println(num1)}

for(num1:Int in 1..5){    //Same as above
   println("Number: $num1")}

for(num1 in 5 downTo 1){
   println(num1)}

// while loop :-
var num2 = 9
while(num2 <= 9){
   num2++
   println(num2)}

// do-while loop :-
var num3 = 99
do {
   num3++
   println(num3)
} while (num3 > 100)

// break and continue :-
var n = 0
while (n < 10){
   n++
   if (n == 8){
```

```
            break }
        if (n == 5){
            continue}
        println(n)}  }

// Function :-
fun langName(){
    println("Kotlin")}

// return :-
fun add():Int {
    return 10 + 90}

// Functions with parameters :-
fun info(name:String, age:Int){
    println("Name: $name and Age: $age")}

// Function with default arguments :-
fun detail(name:String, age:Int = 18){
    println("Name: $name, Age: $age")}

// Function with named arguments :-
fun place(name: String, city: String){
    println("Name: $name, City: $city")}

// High Order Function :-
fun quote(line1:String, line2:String): String{
    return "$line1, $line2"}

fun HOF(l1:String, l2:String, callback:(String,String) -> String): String{
    val quote_function = callback(l1,l2)
    println(quote_function)
    return quote(l1, l2)}

fun main(){
    langName()

    val addFunc = add()
    println(addFunc)

    info("Stark",42)

    detail("Mark")
```

```
 place(city = "NYC",name = "Tim")

 println(HOF("Talk is Cheap","Show me the Code!", ::quote))

 // Lambda Expression/Function :-
 val div = {n1: Int, n2:Int -> n1/n2}
 val result = div(10,2)
 println(result)
 println(div(10,2))

// 2nd method of Lambda Expression/Function :-
 val div2:(Int,Float) -> Float = {a,b -> a/b}    // Int/Int=Int, Int/Float=Float
 println(div2(15,3.5f))

// Anonymous Function :-
val af = fun(s1: String): String{
    return "$s1" }    //"$s1" same as s1 but not "s1"
 println(af("Anonymous!")  }

fun main(){
 var name1:String = "Starks"
//name1 = null     //can't re-assign null
 var name2:String? = "Steve"
 name2 = null     //using null safety, can re-assign null

 var name_length = name1.length
 var name2_length = name2?.length   // must use null safety

// Working with variables that can be null(1st Method) :-
 var length = if(name2 != null) name1.length else "Null!"
 println(length)

 // Working with variables that can be null(2nd Method) :-
 val length2 = name2?.length     // ?.-> safe call
 println(length2)

 // Working with variables that can be null(3rd Method) :-
 //val length3 = name2!!.length  //if name2 is null, its throw exception
 //println(length3)

 // Array :-
 val data = arrayOf("Stark",38,'S',20.20f)
 println(data)    //can't get values of arrays by printing the array
```

```
println(data[2])
//println(data[10])    // Error
println(data.size)
data[0] = "Tony"
println(data[0])
data.set(0,"Tony Stark")
println(data[0])
println(data.get(0))

// Getting values along with index using for loop :-
for( (index,value) in data.withIndex()){
   println("Index: $index , Value: $value")}

// Getting array's value using for-each loop :-
data.forEach { values -> println(values) }

// Getting index using .indices :-
for (index in data.indices){
   println("$index -> ${data[index]}")}

// Creating array with Array Constructor :-
val A = Array(5, {n -> n*n})
A.forEach { n -> println(n) }

// List :-
val list = listOf<String>("Java","C")
println(list)
println(list[1])
println(list.size)
//list[1] = "Kotlin"  //list can't be re-assign

// Getting values of list with for loop :-
list.forEach { values -> println(values) }

// mutableList :-
val mutList = mutableListOf("Kotlin","Java",100)
mutList[0] = "Python"
mutList.add(3,1000)
println(mutList[3])
mutList.removeAt(0)
println(mutList)

// List constructor :-
val list2 = List<Int>(5,{n -> n*n})
```

```
// Set :-
val set = setOf("RAM","ROM",10,"RAM")
println(set)

// mutableSet :-
val ms = mutableSetOf("Stark","IronMan",100)
ms.add(200)
println(ms)
ms.remove("IronMan")
println(ms)

// map/dictionary :-
val kv = mapOf(1 to "One", 2 to "Two")
println(kv)
println(kv[1])
println(kv.keys)
println(kv.values)

val KV = mapOf<Int,String>(1 to "Apple", 2  to "Black",  3 to "Cat")
println(KV)

// mutable map :-
val mm = mutableMapOf("A" to "Apple", "B" to "Bat")
mm["C"] = "Cat"
println(mm)
mm.put("A","Aluminium") //same as mm["A"] = "Aluminium"
println(mm)
mm.keys.remove("A")
mm.values.remove("Bat")
println(mm)  }

// Class :-
class Phone {
    var price = "10k"      // property
    fun about(){           // function
        println("Price: $price")} }

// Primary Constructor :-
class Employee(var name:String, age:Int, salary:Int){
  var Eage = age
  var Esalary:Int
  init {
      Esalary = salary} }
```

```
// Secondary constructor :-
class Employee2{
   var name:String
   constructor(name:String, age: Int, salary: Int){
      this.name = name
      println("Name: $name")} }

// Inheritance :-
open class Father{
   var Fmoney = 10000
   fun fatherinfo(){
      println("Father's money: $Fmoney")} }

class Son: Father(){
   var Smoney = Fmoney/2
   fun soninfo(){
      println("Son's money: $Smoney")} }

// Inheritance with Primary Constructor :-
open class Realme(rRam: String){
   var realmeRam = rRam
   fun parentPc(){
      println("Parent Ram: $realmeRam")} }

class Mi(rRam:String, mRam: String):Realme(rRam){
   var miRam = mRam
   fun childPc(){
      println("Parent Ram: $realmeRam")
      println("Child Ram: $miRam")} }

// Inheritance with Secondary Constructor :-
open class Rm{
   constructor(par:Int){
      println("This is Rm Class's Constructor")} }

class M:Rm{
   constructor(par: Int): super(par){
      println("This is M Class's Constructor")} }

/*Every class has a default constructor,
class extend with its constructor,
if class has secondary constructor then secondary constructor extend parent class constructor,
child class:parent class(){....} //if parent class has default constructor
child class:parent class(par){....} //if parent class has primary/secondary constructor
```

```
child class:parent class{
   constructor(par:int){.....}} //if parent class has default constructor
child class:parent class{
   constructor(par:int):super(par){.....}} //if parent class has primary/secondary constructor*/

// Overriding Properties and Functions ;-
open class Car{
   open var car = "Lambhorghini Aventador"
   open fun cInfo(){
      println("Car: $car")} }

class Bike:Car(){
   var bike = "Hero"
   override var car = "BMW"
   override fun cInfo() {
      println("Bike: $bike")} }

// super :-
open class Car2{
   var Pcar = "Lambhorghini Aventador"
   fun PInfo(){
      println("Car: $Pcar")} }

class Bike2:Car2(){
   var Ccar = super.Pcar
   fun CInfo(){
      super.PInfo()} }

// Modifiers :-
open class mod{
   private var pri = "Private" //visible in only inside this class
   protected var pro = "Protected" //visible in subclasses
   internal var int = "Internal" //visible in same module/package
   public var pub = "Public"  //visible in all classes
   private fun m1(){
      println("I am  $pri function")}
   protected fun m2(){
      println("I am $pro function")}
   internal fun m3(){
      println("I am $int  function")}
   public fun m4(){
      println("I am $pub function")} }
```

```
class cl:mod(){
   //var pi = pri   //can't access private
   var pr = pro
   var i = int
   var pu = pub }

//Abstract class can contain initialized & uninitialized variables, abstract and non-abstract
methods.
abstract class c{
   abstract var ss1: String
   open var ss2: String = "Normal variable"
   abstract fun show1()
   open fun show2(){
      println("Normal function")} }

class c2:c() {
   override var ss1: String = "Override variable"
   override var ss2: String = "Normal variable"
   override fun show1(){
      println("Override function")}
   override fun show2(){
      println("Normal function")} }

//Interface can contain uninitialized variables, abstract and non-abstract methods.
interface i1{
   var ss1: String
   //var ss2: String = "Normal variable"  //Not allowed in interface
   fun show1()
   fun show2(){
      println("Normal function1")}
   fun show3(){
      println("Normal function2")} }

class i2:i1{
   override var ss1: String = "Override variable"
   override fun show1() {
      println("Override function")}
    override fun show2(){
      println("Override normal function1")}}

//interface & class having same function :-
interface ii1{
   fun callMe() }
```

```
interface ii2{
   fun callMe() }

open class ci{
   open fun callMe(){
      println("Normal function2")} }

//class can extend only one class & can implement multiple interfaces
class cl2:ci(),ii1,ii2{
   override fun callMe() {
      println("Call Me Here")} }

// data class :-
data class Det(var name:String, val age:Int)

fun main(){
   val obj = Phone()    // Object
   val obj2 = Phone()   // you can create more than one object
   println(obj.price)
   obj.price = "20k"
   println(obj.price)
   obj.about()

   // Primary Constructor :-
   val mark = Employee("Mark",28, 10000)
   println(mark.Esalary)

  // Secondary constructor :-
   val tim = Employee2("Tim",38,20000)

   // Inheritance :-
  val son = Son()
  println(son.Fmoney)
  son.fatherinfo()

  // Inheritance with Primary Constructor :-
   val realMe = Realme("10GB Ram")
   realMe.parentPc()
   val mi = Mi("20GB Ram","30GB Ram")
   mi.childPc()

   // Inheritance with Secondary Constructor :-
   var m = M(100)
```

```
// Overriding Properties and Functions ;-
   var bike = Bike()
   bike.cInfo()

   // super :-
   var b2 = Bike2()
   println(b2.Ccar)
   b2.CInfo()

   // Visibility Modifier :-
   var c = cl()
   c.pr  //same as cl().pr
   c.i
   c.pu
   c.int
   c.pub

   // Abstract class, method and property :-
   //val c = c()    // can't make object of abstract class
   val c2 = c2()
   c2.show1()
   c2.show2()

   // Interface :-
   val i2 = i2()
   i2.show3()
   i2.show2()

   // Interface having same function :-
   val c3 = cl2()
   c3.callMe()

  // Data class :-
   val  det1 = Det("DataString1",100)
   val  det2 = Det("DataString2",200)
   println(det1)
   println(det1.name)
   println(det1.age)
   println(det2)
   println(det2.name)
   println(det2.age)

  // object Destructuring :-
  val(name1,age1) = det1
```

```
  val(name2,age2) = det2
  println(name1)
  println(age1)
  println(name2)
  println(age2)  }

 fun main(){
    try{
       val a:Int = 10/0
       println(a)}
    catch(error:Exception){
       println("$error, Error!")
       println(error.message)}
    finally {
       println("I am finally block")} }

package com.coding.tech
fun printSome(){
   println("Something") }

package com.coding.tech2
fun printSome(){
   println("Something2") }

import com.coding.tech.printSome
import com.coding.tech2.printSome as printSome2
fun main(){
   printSome()
   printSome2() }

public class JavaCode {
   public int a;
   public int GetValue(){
      return a; }
   public void SetValue(int n){
      this.a = n;}  }

fun gg(n: Int):String{
   return if (n < 10){
      "Smaller"}
   else{
      "Greater"} }
```

```
val pi: Float by lazy {                 //Not initialized untill use it, so best use of memory.
    3.14f }
val pii: Float? by lazy {               //variable can be var/val & nullable/non-nullable
    null }

fun main(){
// Using Java class's codes :-
val jClass = JavaCode()
jClass.SetValue(100)
println(jClass.GetValue())

// elvis operator ?:
var name:String? = "Stark"
name = null
var sameName = name?: "Tony"
println(sameName)

// .inc() & .dec()
var a = 10
println(a.inc())
println(a.dec())

println(gg(5))

val p = Pair("A","B")
println(p.first)
println(p.second)

val(f,s) = Pair("F","S")
println(f)
println(s)

val t = Triple("A","B","C")
println(t.third)

lateinit var name2:String
name2 = "Steve"
println(name2)

val area1 = pi*4*4  //Initialize pi variable in cache memory
println(area1)

val area2 = pi*8*8  //Load pi variable from cache memory
println(area2)  }
```