Table of Contents

[Basics 1](#_Toc18696821)

[Unified Types 4](#_Toc18696822)

[Traits 6](#_Toc18696823)

[Tuples [1,22 elements] 6](#_Toc18696824)

[**Basics**](https://docs.scala-lang.org/tour/basics.html)

**משתנים:**

val x : Int = 5  
 // x = 3; // Not Compile because its Value (Values cannot be re-assigned.)  
 var y : String = "Hey"  
 y = "World" // Compile because its Variable

**פונקציות Functions:**

(list of parameters) => expression involving the parameters.  
for example:

val addOne = (x: Int) => x + 1  
val addOne : (Int => Int) = (x: Int) => x + 1  
 *println*(addOne(1)) // 2

כאשר (**הצהרה**, **קלט**, **פלט**) והצהרה לא חובה

**שיטותMethods :** מסומן ע״י def

def function(x: Int, y:Int) : Int = { *println*(s"(**$**x,**$**y)"); x + y}  
*println*(function(1,2))

כאשר הינו ערך החזרה

שיטות יכולות לקחת גם כמה ערכים (multiple parameter lists) בעוד שפונקציות לא.

def MultiMethod(a:Int)(b: Int) = { Math.*pow*(a,2).toInt \* b }  
*println*(MultiMethod(2)(3)) // 2^2 \* 3 = 12

ערך דיפולטיבי:

def Sum(x: Int, y: Int = 99) : Int = x + y  
*println*(Sum(1,1)); *println*(Sum(1)) // [2,100]

בנוסף ניתן לקבל אוסף של קלטים באופן הבא: ( מ-ללא קלט עד .. )

def getSum(args: Int\*) : Int = {  
 var sum : Int = 0;  
 for(num <- args)  
 sum += num  
 sum  
}  
*println*(getSum(1,2,3)) // 6

**ההבדל בין Val Function vs. Def**

def x1 = *println*( 1 + 1 ) // Will not print  
val x2 = *println*( 1 + 1 ) // Will print

נשקול את המקרה הבא:

val even: (Int => Boolean) = { *println*("val even()"); \_ % 2 == 0 }  
 // Prints val even()  
 def even2(x:Int) : Boolean = { *println*("def even()"); x % 2 == 0}  
// NOT Print def even()  
  
 *println*(even(2))  
 // true [do not print val even()]  
 *println*(even2(2))  
 // def even()  
 // true

דוגמא נוספת:

val x : Int = { Random.nextInt() }  
*println*(x); // 702823910  
*println*(x); // 702823910  
  
def y = { Random.nextInt() }  
*println*(y); // -777004026  
*println*(y); // -1793062244  
  
lazy val z : Int = { Random.nextInt() }  
*println*(z); // -71605430  
*println*(z); // -71605430

**לסיכום:**

|  |  |
| --- | --- |
| val | def |
| val <name> : <type> = <value> | def(<parm>): <return\_type> = <value> |

// Lazy Val:  
// especially useful to avoid heavy computations  
lazy val \_lazy : Unit = *println*("Im Lazy.") // Will NOT print  
val not\_lazy : Unit = *println*("Im Not Lazy =).") // Will Print

**לולאות:**

for(i <-0 until 10) // [0,10)  
for(i <- 0 to 10 ) // [0,10]  
   
val collection = *Array*(1,2,3)  
for(ele <- collection) *println*(ele)

collection.foreach( ele => *print*(s"Best element + 1 **$**{ele+1} \n"))

for( i<-0 until 2; j<-0 until 2)  
 *println*(s"i = **$**i , j = **$**j")  
  
for(i<-0 until 2)  
 for(j<-0 until 2)  
 *println*(s"i = **$**i, j = **$**j")  
/\* Output: SAME IN BOTH WAYS  
 i = 0 , j = 0  
 i = 0 , j = 1  
 i = 1 , j = 0  
 i = 1 , j = 1 \*/

**:Yield**

var collection = for( i<-0 until 10) yield { i \* 2 }  
*println*(collection) // Vector(0, 2, 4, 6, 8, 10, 12, 14, 16, 18)

val EvenNumbersCollection = for( i<-0 until 10 **if** i%2 == 0 ) yield { i }  
*println*(EvenNumbersCollection) // Vector(0, 2, 4, 6, 8)

**רקורסיה:**

def main(args: Array[String]): Unit = {  
 *printf*("Factorial of 5 is %d",*factorial*(5))  
}  
def factorial(num : BigInt) : BigInt = {  
 if(num == 1)  
 1  
 else  
 *factorial*(num - 1) \* num  
}

**מערכים: (Array, ArrayBuffer) - Mutable**

val myNumsFixed = new Array[String](3)  
val myNumsDynm = ArrayBuffer[Int]() // scala.collection.mutable.ArrayBuffer  
myNumsFixed(0) = "Hey"  
myNumsFixed(1) = "You"  
myNumsFixed(2) = "Zvi"  
myNumsFixed(3) = "!" //java.lang.ArrayIndexOutOfBoundsException  
  
myNumsDynm.insert(0,555)  
myNumsDynm.insert(1,666)  
myNumsDynm += 777  
myNumsDynm ++= *Array*(888,999)

for(ele <- myNumsDynm) *println*(ele)

**מערכים דו מימדיים:**

val multDimArray = Array.*ofDim*[Int](10,5)  
for(i<-0 until multDimArray.length; j<-0 until multDimArray(i).length)  
 *println*(multDimArray(i)(j))

val sortedArray = multDimArray(0).sortWith(\_>\_)

**Maps:**

// Immutable  
val employees = *Map*("VP RND" -> "Yoni", "Software Developer" -> "Zvi")  
*println*( if(employees.contains("Software Developer")) "Y" else "N" )

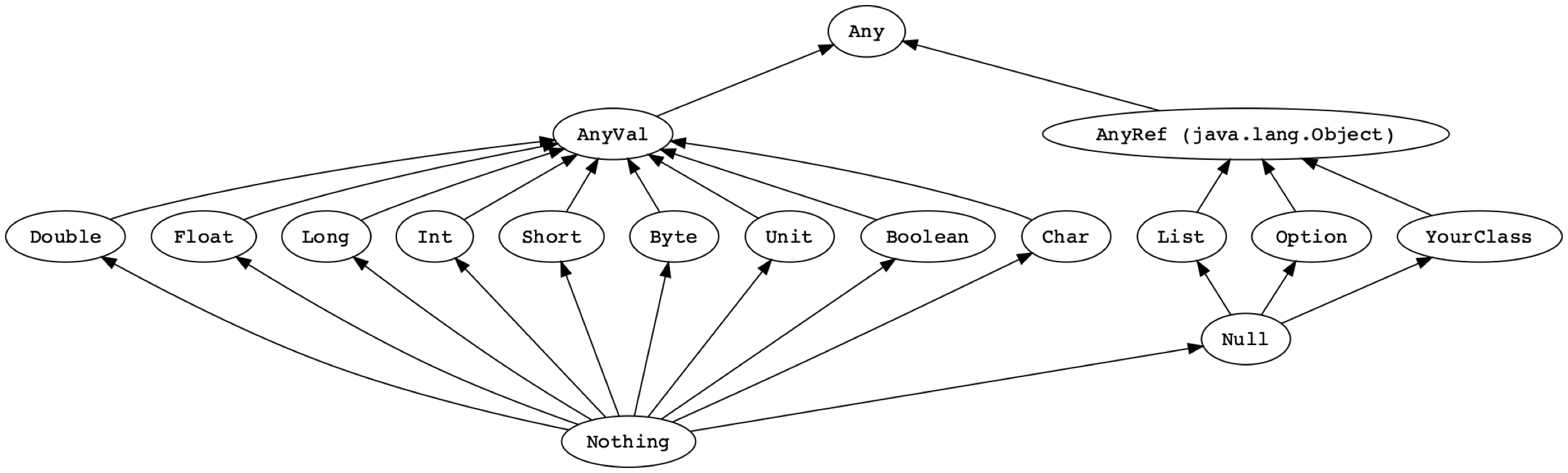
// Mutable  
val employees = scala.collection.mutable.Map("VP RND" -> "Yoni", "Software Developer" -> "Zvi")  
for( (k,v) <- employees ) *println*(s"k = **$**k, v = **$**v")

**Map,Filter,Foreach**

var \_list : List[Int] = *List*(1,2,3,4,5,6,7)  
val filterFunction : (Int => Boolean) = \_ % 2 == 0  
val mapFunction : (Int => Int) = (x:Int) => { x\*2 }  
\_list.filter(filterFunction).map(mapFunction).foreach(*println*)

// 4,8,12

[**Unified Types**](https://docs.scala-lang.org/tour/unified-types.html)



ניתן ליצור רשימה באופן הבא:

object HelloWorld {  
 class Element  
 def main(args: Array[String]): Unit = {  
 var Any\_list : List[**Any**] = *List*("String", new Element);  
 var AnyVal\_list : List[**AnyVal**] = *List*(1,1.5,true);  
 var AnyRef\_List : List[**AnyRef**] = *List*("String",() => "Function",

new Element, null);  
 }  
}

העברת פונקציות:

A **higher**-**order function** takes other **functions** as a parameter or returns a **function** as a result. This is possible because **functions** are a first-class value in **Scala**. ... It means that **functions** can be passed as arguments to other **functions**, and **functions** can return other **functions**.

// Higher Order Functions  
def times3(x:Int) = x\*3  
def MultNum(f:(Int=>Int), x: Int) = f(x)  
  
*println*(MultNum(times3,10))

**Try-Catch Blocks:**

def divide(a: Int, b:Int) = {  
 try { a/b  
 } catch {  
 case ex : java.lang.ArithmeticException => "Cant Divide by 0"  
 } finally {  
 *println*("Finished.")  
 }  
 }  
*println*(divide(2,0)) // Finished. Cant Divide by 0

# [**Classes**](https://docs.scala-lang.org/tour/classes.html)

**case class** CasePoint(x: Int, y: Int) { }  
**class** Point(x: Int, y: Int) { }

var p1 = new Point(0,0)  
var p2 = new CasePoint(0,0)  
var p3 = *CasePoint*(0,0) // You can instantiate case classes **without** new keyword.  
 // case classes have an apply method by default  
 // which takes care of object construction.  
*println*( p2 == p3 ) // true => they are compared by **value**.  
*println* ( p1 == p2 ) // false => They are compared by reference( p1 == p1 )

1.Case Class **doesn't** need explicit new, while class need to be called with new

2.By Default constructors parameters are **private** in class , while its **public** in case class

3.case class **compare** themselves by **value**

4.By Default constructors parameters are **val** in case class and **var** in class

5. case class  are **immutable** by default

case class Ele1(x : Int)  
class Ele2(x : Int)

var *immutable* = *Ele1*(1)  
*immutable*.x = 3  
var *mutable* = Ele2(1)  
*mutable*.x = 3

מחלקת נקודה:

object HelloWorld {  
 def main(args: Array[String]): Unit = {  
 var p1: Point = new Point(1,2)  
 var p2 : Point = new Point(-1)  
 *println*(p2.getX) // 0  
 var p3 : Point = new Point()  
 *println*(p3) // (3,0,0)  
 p3.x\_(4)  
 *println*(p3.x)  
 }  
 class Point(private var \_x: Int,  
 private var \_y: Int) {  
 private val *\_id* = Point.*getUniqueID* // "Static" Method  
 this.setX(\_x)  
 this.setY(\_y)  
 // Constructors  
 def this(x: Int) = this(x,0)  
 def this() = this(0,0)  
  
 // Getters and Setters  
 def x\_(x: Int) = { this.\_x = if(x>=0) x else 0 }

def x = \_x  
  
  
 def setX(x: Int) = { this.\_x = if(x >= 0) x else 0 }  
 def setY(y: Int) = { this.\_y = if(y >= 0) x else 0 }  
 def getX = this.\_x;  
 def getY: Int = this.\_y  
  
 // Override  
 override def toString = s"(**$**{this.*\_id*},**$**{this.\_x},**$**{this.\_y})"  
 }

object Point {  
 private var *count* : Int = 0  
 def getUniqueID() : Int = {  
 *count* += 1  
 *count* }  
 }  
}

**ירושה: (Inheritance)**

class OtherCoolPoint(x1 : <type>, x2: <type> ... ) extends Point {}

**מחלקה אבסטרקטית:**

abstract class A(private var \_x: Int) {  
 var \_y : Int  
 def getX() : Int  
 override def toString() = "Implemented Function."  
}  
class B(\_x: Int) extends A(\_x) {  
 // Must to Implement:  
 override var *\_y*: Int = *???* override def getX(): Int = *???* // New Methods:  
 def Move() = *???*}

**Traits**

כמו ממשק בג'אווה, ההבדל הוא שניתן לממש פונקציות.

trait Flyable {  
 def fly : String  
}  
trait BulletProof {  
 def HitByBullet : Boolean  
 def getState : String = "Easy."  
}  
class Superhero(name: String) extends Flyable with BulletProof {  
 // Must to implement  
 override def fly: String = *???* override def HitByBullet: Boolean = *???*}

[**Tuples**](https://docs.scala-lang.org/tour/tuples.html) **[1,22 elements]**

val ingredient = ("Sugar" , 25)  
*println*(ingredient.\_1) // Sugar  
*println*(ingredient.\_2) // 25

ניתן לאתחל בעזרתם משתנים באופן הבא:

val (name, quantity) = ingredient  
*println*(name) // Sugar  
*println*(quantity) // 25

ניתן לעבור על Tuples באופן הבא:

ingredient.productIterator.foreach( x => *println*(x))

או לחלופין:

*println*(ingredient.toString()) // (Sugar,25)

[Class Composition with Mixins](https://docs.scala-lang.org/tour/mixin-class-composition.html)

[Higher-order Functions](https://docs.scala-lang.org/tour/higher-order-functions.html)

[Nested Methods](https://docs.scala-lang.org/tour/nested-functions.html)

[Multiple Parameter Lists (Currying)](https://docs.scala-lang.org/tour/multiple-parameter-lists.html)

[Case Classes](https://docs.scala-lang.org/tour/case-classes.html)

[Pattern Matching](https://docs.scala-lang.org/tour/pattern-matching.html)

[Singleton Objects](https://docs.scala-lang.org/tour/singleton-objects.html)

[Regular Expression Patterns](https://docs.scala-lang.org/tour/regular-expression-patterns.html)

[Extractor Objects](https://docs.scala-lang.org/tour/extractor-objects.html)

[For Comprehensions](https://docs.scala-lang.org/tour/for-comprehensions.html)

[Generic Classes](https://docs.scala-lang.org/tour/generic-classes.html)

[Variances](https://docs.scala-lang.org/tour/variances.html)

[Upper Type Bounds](https://docs.scala-lang.org/tour/upper-type-bounds.html)

[Lower Type Bounds](https://docs.scala-lang.org/tour/lower-type-bounds.html)

[Inner Classes](https://docs.scala-lang.org/tour/inner-classes.html)

[Abstract Type Members](https://docs.scala-lang.org/tour/abstract-type-members.html)

[Compound Types](https://docs.scala-lang.org/tour/compound-types.html)

[Self-type](https://docs.scala-lang.org/tour/self-types.html)

[Implicit Parameters](https://docs.scala-lang.org/tour/implicit-parameters.html)

[Implicit Conversions](https://docs.scala-lang.org/tour/implicit-conversions.html)

[Polymorphic Methods](https://docs.scala-lang.org/tour/polymorphic-methods.html)

[Type Inference](https://docs.scala-lang.org/tour/type-inference.html)

[Operators](https://docs.scala-lang.org/tour/operators.html)

[By-name Parameters](https://docs.scala-lang.org/tour/by-name-parameters.html)

[Annotations](https://docs.scala-lang.org/tour/annotations.html)

[Default Parameter Values](https://docs.scala-lang.org/tour/default-parameter-values.html)

[Named Arguments](https://docs.scala-lang.org/tour/named-arguments.html)

[Packages and Imports](https://docs.scala-lang.org/tour/packages-and-imports.html)

[Package Objects](https://docs.scala-lang.org/tour/package-objects.html)