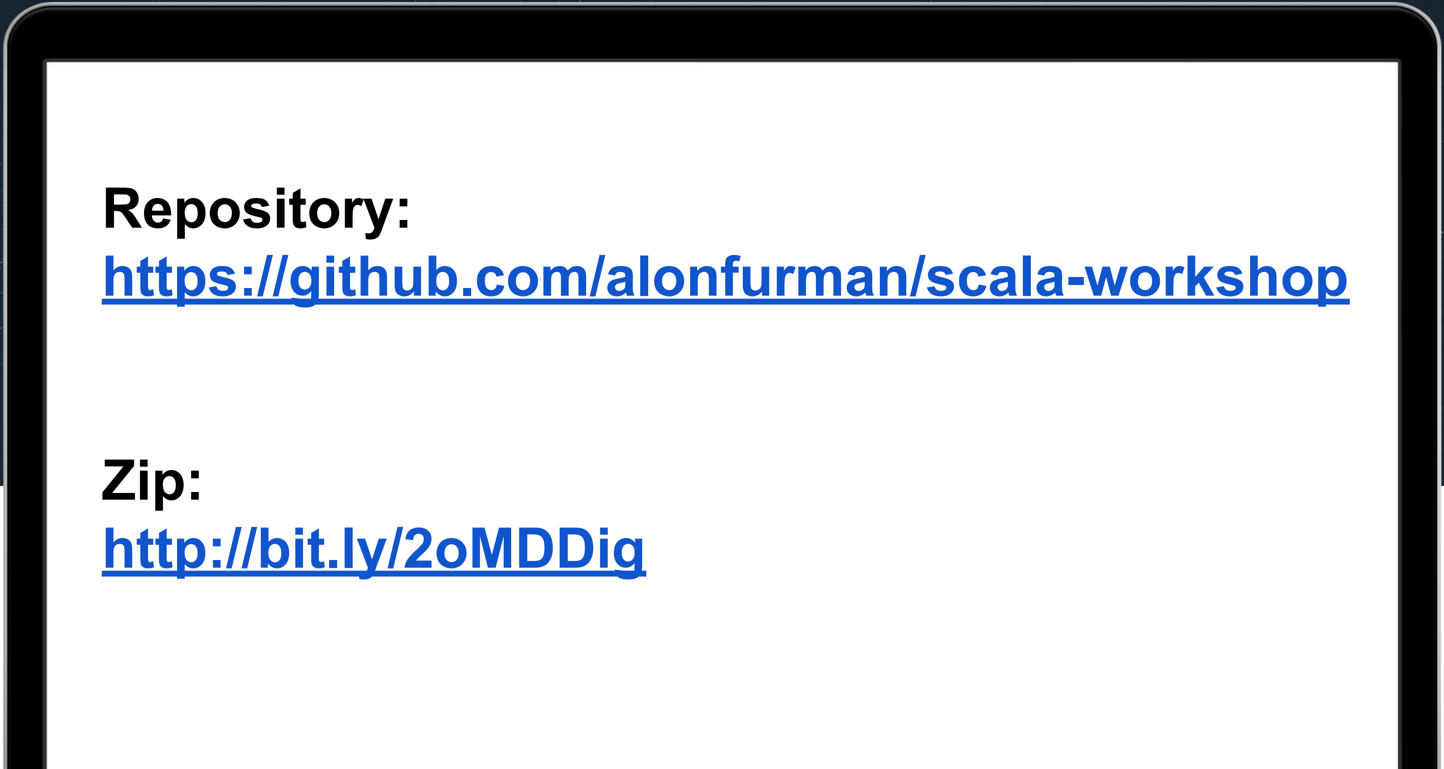
Please clone the repository or download the zip file



Repository:
https://github.com/alonfurman/scala-workshop

Zip:
http://bit.ly/2oMDDig

WiX Engineering

Scala Workshop

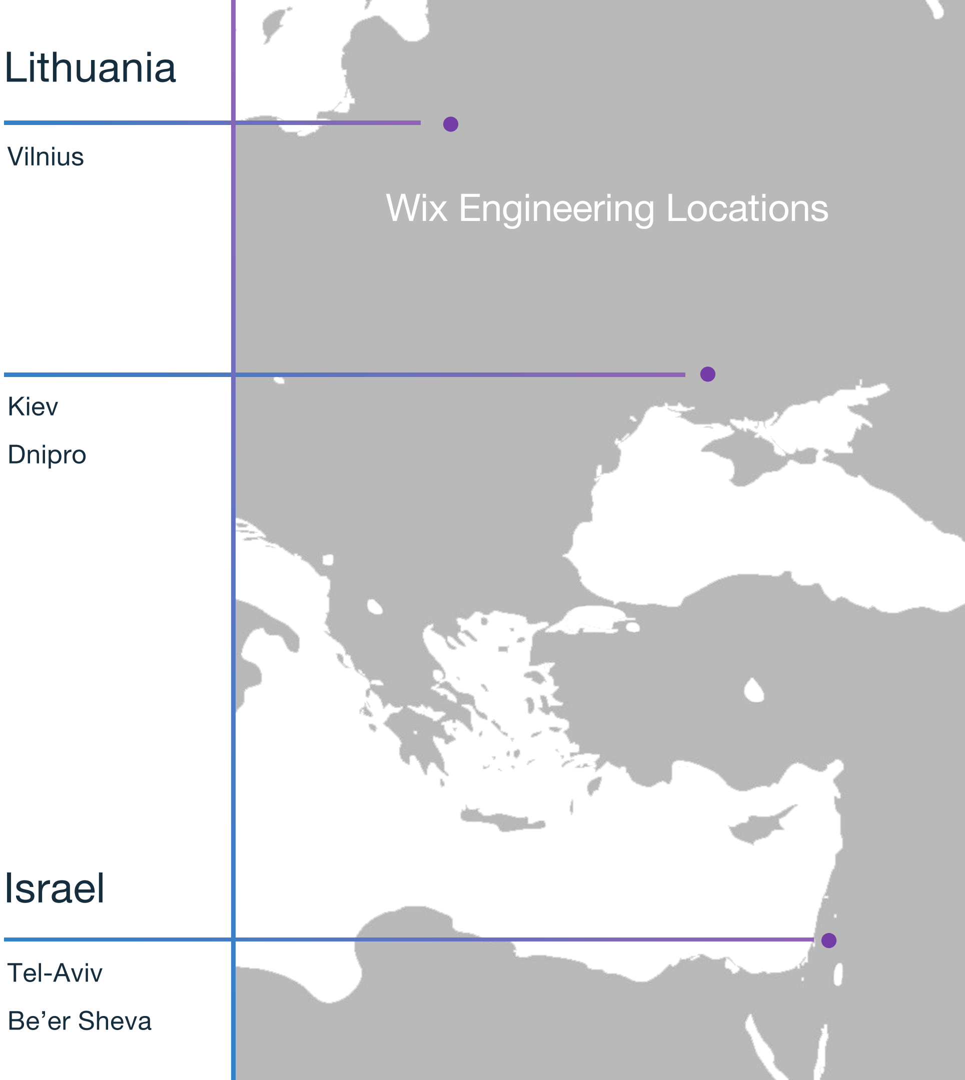
Alon Furman
Software Engineer at Wix

alonf@wix.com

Hi.

I’m Alon Furman

- Backend developer at Wix
- Working on the Wix Bookings application
- 8 years experience
- 2.5 years with Scala
- TDD and clean code enthusiast



Wix Engineering Locations

Lithuania

Vilnius

Kiev

Dnipro

Israel

Tel-Aviv

Be’er Sheva

AGENDA

Basic Syntax

Object-Oriented

Functional

Collections

Pattern Matching

and more

Scala

1. Scalable language
2. Statically typed

★ Runs on JVM

★ Allows mixed Scala/Java projects

★ Can use existing Java libraries, e.g., Spring and JUnit.

01

Basic Syntax

Example #1

Hello World

Namespace

Defining a Class / Object

- Everything is an object
- Classes like in Java
- Public by default

Defining a Method

Invoking a Method

```
package com.wix.academy.examples

object HelloWorld {
  def main( args: Array[ String ] ) {
    println( "Hello world" )
  }
}
```

Type Inference

```scala
val sum = 1 + 2

val list = List("Hello", "World", "!")

val map = Map(1 -> "abc", 2 -> "d")
```

Type Inference

```scala
val sum: Int = 1 + 2

val list: List[String] = List("abc", "def")

val map: Map[Int, String] = Map(1 -> "Hello", 3 -> "World")
```

Example #2

Fibonacci

Type Inference

Variable Declaration

For loops / comprehensions

“Everything is an expression”

```
def fib( index: Int ) = {
  var prev1 = 0
  var prev2 = 1
  for ( i <- 0 until index ) {
    val current = prev1 + prev2
    prev1 = prev2
    prev2 = current
  }
  prev2
}
```

Let’s Talk

Mutability

Scala is immutable by default

But the decision is yours

```
def fib( index: Int ) = {
  var prev1 = 0
  var prev2 = 1
  for ( i <- 0 until index ) {
    val current = prev1 + prev2
    prev1 = prev2
    prev2 = current
  }
  prev2
}
```

Your Turn ✌

Exercise #1

Recursive Fibonacci

Write a method that calculates the *n*-th element in the Fibonacci sequence.

★ Your solution must be recursive

Hold up, grasshopper!

Before You Start

Recursive methods need an *explicit* type annotation.

```
def factorial( n: Int ): Int =
  if ( n == 0 )
    1
  else
    n * factorial( n - 1 )
```

Recursive Fibonacci

Possible Solution

```
def rfib( n: Int ): Int =
  if ( n <= 1 )
    1
  else
    rfib( n - 1 ) + rfib( n - 2 )
```

02

Object-Oriented AND Functional

- Scala is OO

  Everything is an object
- Classes like in Java

```scala
1.toString // 1

(2).+(7) // 9


class Car(val number: Int)
val car = new Car(1)
```

Objects

Like Singleton in Java.

```
object Foo {
  def greet(name: String) = println("Hello " + name)
  def listSum(lst: List[Int]) = lst.sum
}

Foo.greet("David") // Hello David
Foo.listSum(List(1, 2)) // 3
```

A Person Class in Java

```java
public class Person {
  private String firstName;
  private String lastName;
  private int age;

  public Person(String firstName, String lastName, int age) {
      this.firstName = firstName;
      this.lastName = lastName;
      this.age = age;
  }

  public String getFirstName() {
      return firstName;
  }
  public void setFirstName(String firstName) {
      this.firstName = firstName;
  }
        .
        .
        .
  public int getAge() {
      return age;
  }
  public void setAge(int age) {
      this.age = age;
  }
```

```java
@Override
public boolean equals(Object o) {
  if (this == o) return true;
  if (!(o instanceof Person)) return false;

  Person person = (Person) o;

  if (age != person.age) return false;
  if (!firstName.equals(person.firstName)) return false;
     return lastName.equals(person.lastName);
  }
@Override
public int hashCode() {
  int result = firstName.hashCode();
  result = 31 * result + lastName.hashCode();
  result = 31 * result + age;
  return result;
}
@Override
public String toString() {
  return "Person{firstName='" + firstName + '\'' + ", lastName='" +
lastName + '\'' + ", age=" + age + '}';
}
```

Scala Person Class

Provides:

- Constructor
- Getters
- hashCode
- equals
- toString

```scala
case class Person
(firstName: String, lastName: String, age: Int)


val person = Person("Dan", "Cohen", 20)
println(person) // Person(Dan,Cohen,20)
```

Traits

Like Interfaces in Java but allow implementation and more.

Scala allows multiple traits / classes / objects per file.

```
trait Ord {
  def < (that: Any): Boolean
  def <=(that: Any): Boolean =  (this < that) || (this == that)
  def > (that: Any): Boolean = !(this <= that)
  def >=(that: Any): Boolean = !(this < that)
}

class Product(val price: Int) extends Ord {
  override def <(that: Any): Boolean = {
    val other = that.asInstanceOf[Product]
    this.price < other.price
  }
}

val prod1 = new Product(10)

val prod2 = new Product(2)


prod1 >= prod2 // true
```

In Java

```java
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class ClassWithLogs {
  private static Logger logger =
 LoggerFactory.getLogger(ClassWithLogs.class);

  ....
  logger.info("something" +
person.toString());
  logger.debug("debug" +
calculateSomething());
  ....
}
```

In Scala

```scala
trait Logging {
  private val logger =
LoggerFactory.getLogger(this.getClass)

  protected def logInfo(message: => String) =
    if (logger.isInfoEnabled) logger.info(message)

  protected def logDebug(message: => String) = ....
}

class ClassWithLogs extends Logging {
  ...
  logInfo("something" + person.toString)
  logDebug("debug" + calculateSomething())
  ...
}
```

Functions

1st class citizens

```
(x: Int) => x * x


def square = (x: Int) => x * x

def square: (Int) => Int = (x: Int) => x * x

square(2) // 4



def sum(func: Int => Int, a: Int, b: Int): Int = {

  func(a) + func(b)

}


sum(square, 2, 4) // 20
```

What you can do with Functional

Currying

```
def add(a: Int)(b: Int): Int = a + b

def add4 = add(4)

def add4: (Int) => Int = add(4)

add4(5) // 9
```

Your Turn ✌

Exercise #2

IsSorted

Write a function to check if a list is sorted according to a given comparison function

IsSorted

Possible Solution

```scala
def isSorted(lst: List[Int], compare: (Int, Int) => Boolean) =
{
  var sorted = true
  for(i <- 0 until lst.length - 1) {
    if (!compare(lst(i), lst(i+1))) sorted = false
  }
  sorted
}
```

03

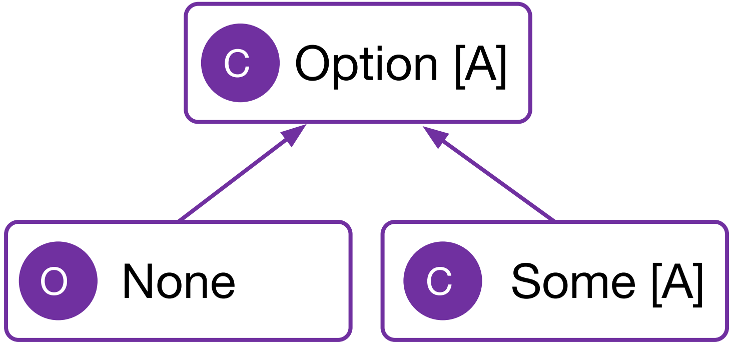
Collections are Crazy in Scala

Defining Collections

```
val sequence = Seq(1, 2, 3)
val lst = List(1, 2, 3)
val set = Set(1, 2, 3)
val map = Map("a" -> 1, "b" -> 2)

val hostPort  = ("localhost", 80)
hostPort._1 // localhost
hostPort._2 // 80
```

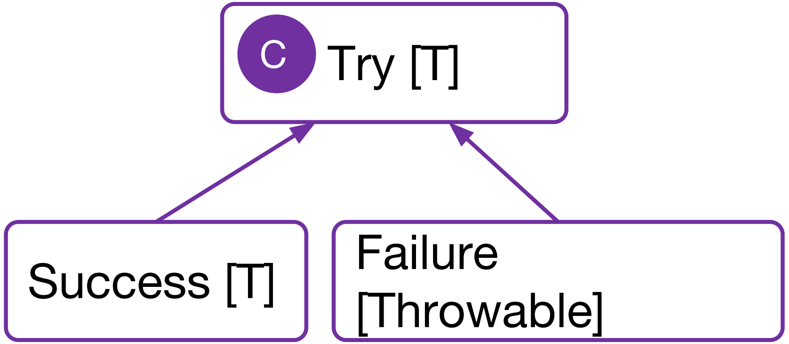
Options



```
val opt1: Option[Int] = Option(5) //Some(5)
val opt2: Option[Int] = Option(null) //None

opt1.isDefined // true
opt1.get  //5
opt2.isDefined //false
opt2.get // Exception
opt2.getOrElse(10) //10
```

Try



```
val try1 = Try(6/2) //Success(3)
val try2 = Try(6/0)
//Failure(java.lang.ArithmeticException:/by zero)


try1.get // 3
try2.getOrElse(0) // 0
```

Functional Combinators

With Collections

```
List(1, 2, 3).head // 1
List(1, 2, 3).tail // List(2, 3)


val lst = List(1, 2, 3)
val lst2 = 1 :: 2 :: 3 :: Nil
lst == lst2 // true


val h::t = lst // h = 1, t = List(2,3)
```

Functional combinators

```scala
List(1, 2, 3).map(i => i * 2) // List(2, 4, 6)
List(1, 2, 3).foreach(i => println(i)) //
List(1, 2, 3).find(_ > 2) // Some(3)
List(1, 2, 3).find(_ > 6) // None
List(1, 2, 3).min // 1
List(1, 2, 3).max // 3
List(1, 2, 3).sum // 6

List(1, 2, 3).filter(i => i % 2 == 0) // List(2)
List(1, 2, 3).partition(_ % 2 == 0) // (List(2), List(1, 3))
List(1, 2, 3, 4, 5).take(2) // List(1, 2)
List(1, 2, 3, 4, 5).takeWhile(_ <= 3) // List(1, 2, 3)
List(1, 2, 3, 4, 5).dropWhile(_ <= 3) // List(4, 5)
```

Functional combinators

```scala
val person1 = Person("Dan", "Cohen", 20)
val person2 = Person("Tali", "David", 24)
val people = List(person1, person2)

people.map(p => p.firstName) // List("Dan", "Tali")
people.filter(_.age > 22) // List(person2)

val youngest = people.minBy(p => p.age)
```

Your Turn ✌

Exercise #3

Phone Book

Given a list of lecturers in a department prepare a map which aggregates the lecturers names by the first letter in sorted order.

e.g.

given: Avi, Alina, David, John, Jessica

result: Map(D -> List(David), J -> List(Jessica, John), A -> List(Alina, Avi))

Task name

Possible Solution

```
def aggregateByName(lst: List[Lecturer]):
 Map[Char, List[String]] = {
  val lecturersNames = lecturers.map(_.name)
  lecturersNames.sorted.groupBy(_.head)
}
```

04

Pattern Matching

Pattern matching

```
val x: Any = ...
x match {
 case 1 => "Integer 1"
 case "1" => "String 1"
 case b: Boolean => "Boolean " + b.toString
 case i: Int if i > 0 => "Positive int"
 case _ => "Unknown"
}
```

You Can Match Almost Anything

...and handle its values

```
Option(something) match {
 case Some(value) => value
 case None => 0
}
val person = Person("John", "Johnson", 20)
person match {
 case Person(_, _, 20) => "Person with age = 20"
 case Person("David", _, _) => "Person named David"
 case Person(_, _, age) if age > 60 => "Senior Person"
 case _ => "Default Person"
}
def length(lst: List) {
  lst match {
    case Nil => 0
    case h::t => 1 + length(t)
  }
}
```

Pattern Matching

Java

```java
enum MaritalStatus {
  SINGLE, MARRIED, DIVORCED, WIDOWED
}

enum Gender {
  MALE, FEMALE
}

class Person {
  ...
  private Gender gender;
  private MaritalStatus maritalStatus;
  ...
}
```

Scala

```scala
sealed trait MaritalStatus
case object Single extends MaritalStatus
case object Married extends MaritalStatus
case object Divorced extends MaritalStatus
case object Widowed extends MaritalStatus

sealed trait Gender
case object Male extends Gender
case object Female extends Gender
```

Pattern Matching

Java

```java
public String getSalutation() {
  if (gender == null) return null;
  switch(gender) {
    case MALE:
      return "Mr.";
    case FEMALE:
      if (maritalStatus == null)
         return "Ms.";
      switch(maritalStatus) {
        case SINGLE:
          return "Miss";
....
```

Scala

```scala
def salutation: Option[String] =
  (gender, maritalStatus) match {
    case(Some(Male), _)              => Some("Mr.")
    case(Some(Female), Some(Single)) => Some("Miss.")
    case(Some(Female), None)         => Some("Ms.")
    case(Some(Female), _)            => Some("Mrs.")
    case(None, _)                    => None
  }
```

Your Turn ✌

Exercise #4

Expression Evaluator

Given an expression (either “const” or “sum”) evaluate the expression.

```
evaluate(Sum(Const(5), Sum(Const(1),
Sum(Const(3), Const(2))))) == 11
```

Expression Evaluator

Possible Solution

```scala
def evaluate(expr: Expression): Int = expr match {
  case Const(v) => v
  case Sum(l, r) => evaluate(l) + evaluate(r)
}
```

05

Back to Default

Easy Defaults

Named Parameters

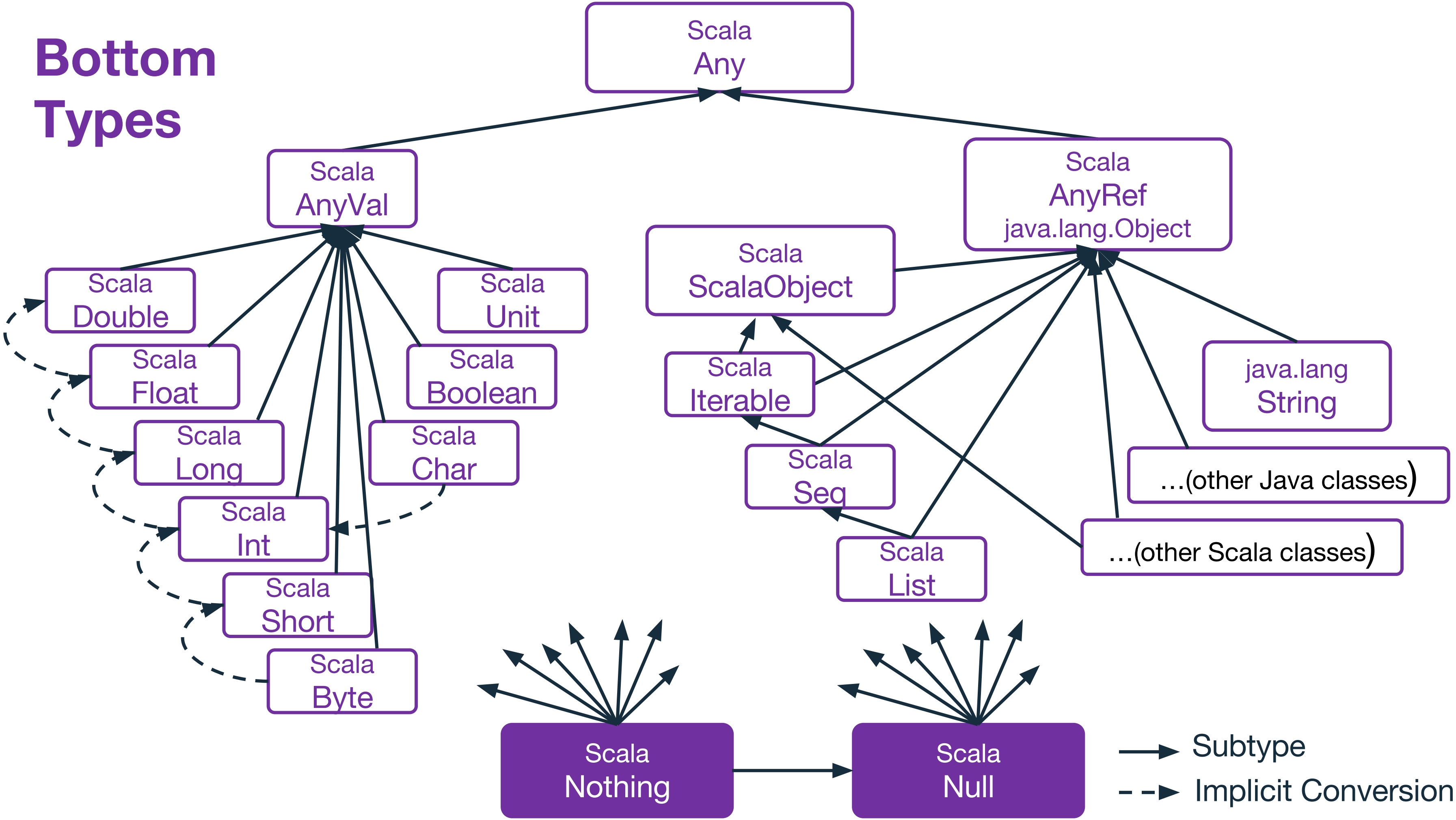
Java

```java
public Pizza deliver(String address, int phone, Map<String, Boolean> toppings) {...}
public Pizza deliver(String address, int phone) {deliver(address, phone, new HashMap<>());}
public Pizza deliver(String address) {deliver(address, 1234, new HashMap<>());}
public Pizza deliver() {deliver("", 1234, new HashMap<>());}
```

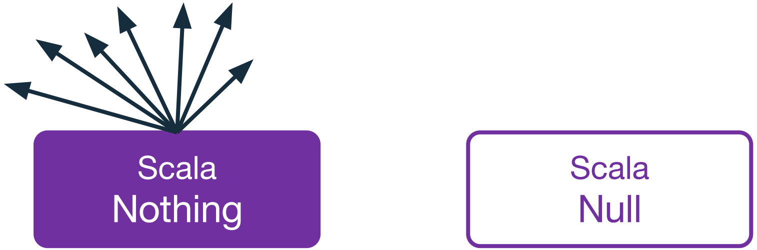
Scala

```scala
def deliver(address: String = "", phone: Int = 1234, toppings: Map[String,Boolean] = Map.empty)
deliver()
deliver(phone = 054123123, address = "Beer Sheva")
deliver(toppings = Map("olives" -> true))
```

Bottom Types



The Nothing



A subtype of every other type,
there exists no instances of this type.

A return type for methods
which never return normally.

`value scala.collection.immutable.Nil` is of type `List[Nothing]`

Lazy Val

Evaluated when it is accessed for the 1st time (as opposed to `val` which is executed when defined).

```
val checkCapacity = {      val checkCapacity = {
  print("capacity")          print("capacity")
  false                      true
}                          }

//heavy operation          //heavy operation
lazy val canRegister = {   lazy val canRegister = {
  print("register")          print("register")
  true                       true
}                          }

x && y // capacity         x && y // capacity register
```

Call by Name

Typically, parameters to functions are *by-value* parameters (value evaluated before they are passed).

By-name parameters aren’t evaluated until they are used within function.

```
def something(): Int = {
  println("calculation")
  222
}

def byValue(value: Int): Unit = {
  println("byValue1: " + value)
  println("byValue2: " + value)
}

def byName(value: => Int): Unit = {
  println("byName1: " + value)
  println("byName2: " + value)
}
```

```
byValue(something())

// calculation
// byValue1: 222
// byValue2: 222


byName(something())

// calculation
// byName1: 222
// calculation
// byName2: 222
```

Easy

Strings

```
val name = "Alon"
val greeting = s"Hello $name, welcome!" //Hello Alon, welcome!

val special = """Sentence with "quotes" and / the easy way""" //
Sentence with "quotes" and / the easy way
```

Partial Functions

Provides an answer only for a subset of possible data, and defines the data it can handle.

```
List(41, "cat") map { case i: Int ⇒ i + 1 }
scala.MatchError: cat (of class java.lang.String)

val partial = new PartialFunction[A, B] {
  def apply(d: A): B = ...
  def isDefinedAt(d: A): Boolean = ...
}
```

Partial Functions

Can be queried to determine if it can handle a particular value.

```
val incAny: PartialFunction[Any, Int] = { case i: Int ⇒ i + 1 }

incAny.isDefinedAt(41) //true
incAny.isDefinedAt("cat") //false

List(41, "cat") collect incAny //List(42)

val fraction = new PartialFunction[Int, Int] {
  def apply(d: Int) = 42 / d
  def isDefinedAt(d: Int) = d != 0
}

List(0,1,2) collect { fraction } // List(42, 21)
```

For Comprehensions

```
for (i <- 1 to 5) yield i //Vector(1, 2, 3, 4, 5)
for (i <- 1 to 5) yield i % 2 //Vector(1, 0, 1, 0, 1)

val names = List("chris", "ed")
val capNames = for (e <- names) yield e.capitalize
//List("CHRIS", "ED")

val people = List(Person("Dani", "", 20), Person("Dina","", 15))
for (person <- people; if person.age < 18) yield person.firstName
// List(Dina)
```

For Comprehensions

```scala
case class Person(firstName:String, lastName:String)

val maybeFirstName : Option[String] = Some("Bruce")
val maybeLastName : Option[String] = Some("Wayne")

for (firstName <- maybeFirstName; surname <- maybeLastName)
  yield Person(firstName, surname) //Some(Person(Bruce,Wayne))

val maybeFirstName : Option[String] = Some("Bruce")
val maybeLastName : Option[String] = None

for (firstName <- maybeFirstName; surname <- maybeLastName)
  yield Person(firstName, surname) //None
```

Your Turn ✌

Exercise #5

Student Ages

Print all students ages in the department

Notice: student age is not a mandatory field!

Student Ages

Possible Solution

```scala
def studentAgesList(dept: Department): List[Int] = {
  for (
    course <- dept.courses;
    participant <- course.participants;
    optAge <- participant.age
  ) yield optAge
}
```

Implicit Conversions

```
val i: Int = "123"  //Compilation Error
implicit def strToInt(str: String) = str.toInt
math.max("123", 111) //123


implicit def intToDigits(i: Int): List[Int] = i.toString.toList.map(_.asDigit)
250.map(_ * 2) //List(4, 10, 0)
```

Implicit Conversions

```
object Greeter {
   def greet(name: String)(implicit prompt: PreferredPrompt) = {
      println("Welcome, " + name + ". The system is ready.")
      println(prompt.preference)
   }
}

val bobsPrompt = new PreferredPrompt("relax> ")
Greeter.greet("Bob")(bobsPrompt)
//Welcome, Bob. The system is ready.
//relax>
```

Implicit Conversions

```
object JoesPrefs {
  implicit val prompt = new PreferredPrompt("Yes, master> ")
}

Greeter.greet("Joe")
//error: could not find implicit value for parameter prompt: //PreferredPrompt

import JoesPrefs._
Greeter.greet("Joe")
//Welcome, Joe. The system is ready.
//Yes, master>
```

Streams

```
val stream = 1 #:: 2 #:: 3 #:: Stream.empty
// scala.collection.immutable.Stream[Int] = Stream(1, ?)

val stream = (1 to 100000000).toStream //Stream(1, ?)
stream.head // 1
stream.tail // Stream(2, ?)

stream.take(3) // Stream(1, ?)
stream.filter(_ > 200) // Stream(201, ?)
stream.map(_ * 2) // Stream(2, ?)

stream.sum
stream.max
// Out of memory
```

Streams

```
stream(0)  // returns 1
stream(1)  // returns 2
stream(10)  // returns 11

stream.take(5).foreach(println) // 1 2 3 4 5

def numsFrom(n:Int): Stream[Int] = {
  Stream.cons(n,numsFrom (n+1))
}

numsFrom(5).take(10).foreach(print)
// 5 6 7 8 9 10 11 12 13 14
```

Exercise #6

Game of Life

The universe of the Game of Life is a two-dimensional grid of square cells, each of which is in one of two possible states, alive or dead.
Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent.
At each step in time, the following transitions occur:

1. Any live cell with fewer than two live neighbours dies, as if caused by underpopulation.
2. Any live cell with two or three live neighbours lives on to the next generation.
3. Any live cell with more than three live neighbours dies, as if by overpopulation.
4. Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

Your Turn ✌

Exercise #7

Tic Tac Toe

Implement a tic tac toe game and determines when the status is victory or draw.

We didn’t cover:

- DSLs
- Macros
- Future and promises
- Type members
- Structural types
- Extractor objects
- and much much more...

Not everything is perfect:

- Long compile times
- Limited automatic refactorings options
- Many ways to write the same thing may lead to confusion

Multiple ways to do the same thing

```
people.foreach((person: Person) => println(person))

people.foreach(person => println(person))

people.foreach{person => println(person)}

people.foreach{println(_)}

people.foreach(println)

people foreach println
```

WiX Engineering

Q&A

✉ alonf@wix.com

Thank You

✉ alonf@wix.com

Credits

- http://www.slideshare.net/holograph/5-bullets-to-scala-adoption
- http://www.slideshare.net/maximnovak/joy-of-scala
- http://aperiodic.net/phil/scala/s-99/
- Programming in Scala, Third Edition by Martin Odersky, Lex Spoon, and Bill Venners
- https://github.com/softwaremill/simple-http-server
- http://blog.mgm-tp.com/2012/03/hashset-java-puzzler/
- https://gist.github.com/OlegIlyenko/771842