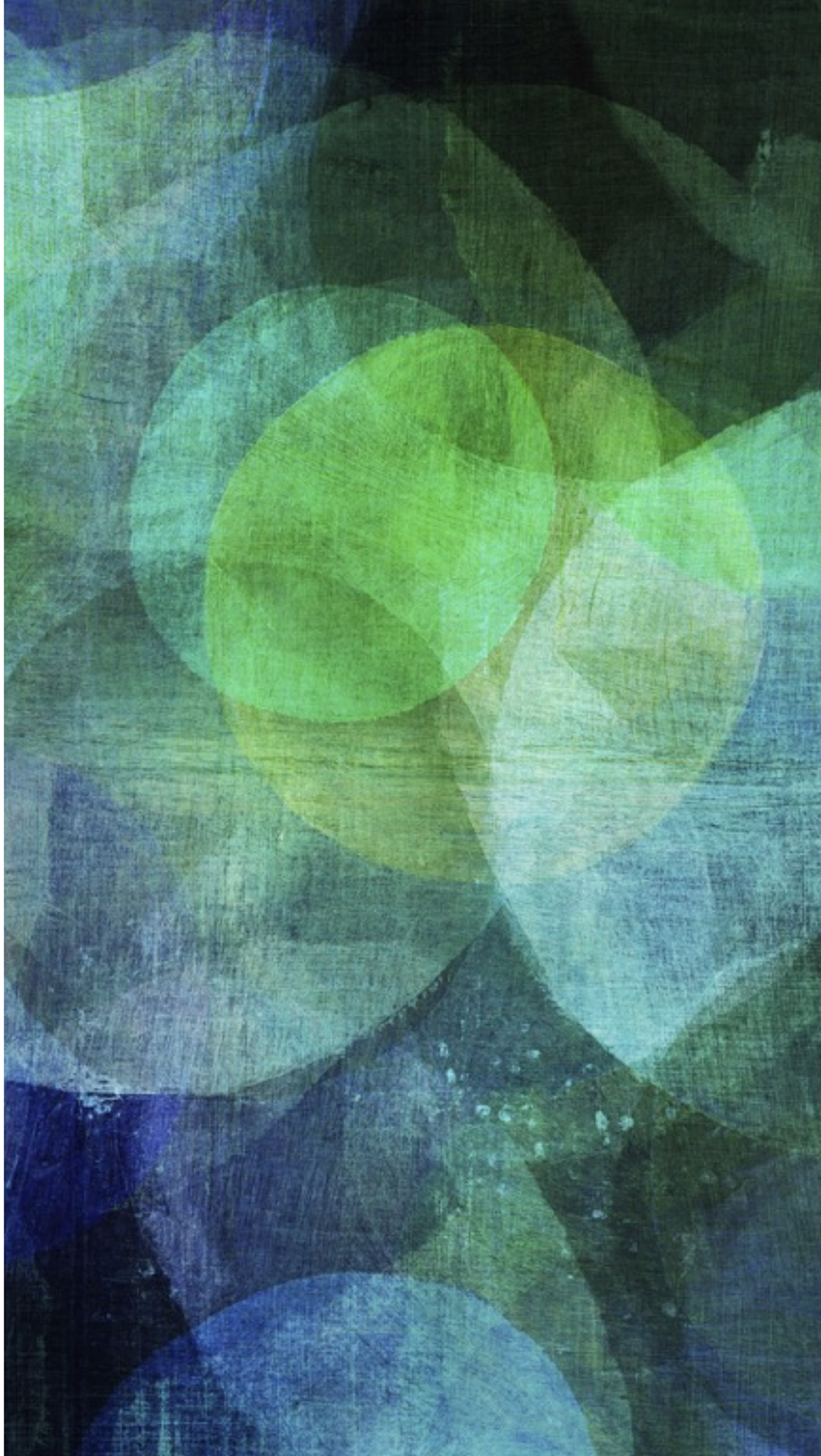
INTRODUCTION TO SCALA

Welcome



LETS MAKE THIS INTERESTING, CHALLENGING, INTERACTIVE

OBJECTIVES

- Introduce you to *enough* Scala with IntelliJ to become proficient writing DSE Analytics applications

HOW?

- Presentation of key concepts
- Hands-on exercises
- Provide recommendations and references
- Listen. Q&A

WELL HELLO

- First, let’s start downloading

- Start downloading or grab a USB drive
- Download URL: TBD URL
- Survey link

INTELLIJ INSTALLED?

- If yes, :)
- If no….

INTELLIJ QUICK STEPS

- Oracle Java Development kit. http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
- IntelliJ IDEA. https://www.jetbrains.com/idea/download/
  - Community 2016.2.5
- Windows note…

WINDOWS NOTE

- Later, if you get an exception as explained in SPARK-2356 [1] that occurs due to a missing WinUtils.exe on Windows. To work around this error, you must download the executable from here [2] to a location like C:\WinUtils\bin. You must then add an environment variable HADOOP_HOME and set the value of the variable to C:\WinUtils.

- [1] https://issues.apache.org/jira/browse/SPARK-2356
- [2] http://public-repo-1.hortonworks.com/hdp-win-alpha/winutils.exe

SCALA

- A little background and perspective

WHAT IS SCALA? FROM THE MARKETING DEPT…

- Object-Oriented Meets Functional
- “Have the best of both worlds. Construct elegant class hierarchies for maximum code reuse and extensibility, implement their behavior using higher-order functions. Or anything in-between.”
- https://www.scala-lang.org/

WHY SCALA?

- Type Safety
- Expressive
  - First-class functions
- Concise
- Java interoperability
  - Can reuse java libraries and tools

SCALA HISTORY

- The design of Scala started in 2001 at the École Polytechnique Fédérale de Lausanne (EPFL) by Martin Odersky. It followed on from work on Funnel, a programming language combining ideas from functional programming and Petri nets.

- https://en.wikipedia.org/wiki/Martin_Odersky

HOW SCALA?

- Compiles to Java bytecode
- Works with any standard JVM

SCALA VERSIONS

- 2.10, 2.11, 2.12

- https://www.scala-lang.org/download/all.html

HANDS-ON

- Let’s get started with IntelliJ

- Hello World with worksheets, console and application

NEXT

- Tour of key language elements

EXPRESSIONS

```
1 + 1
1 > 0
0 > 1
```

VALUES

- Named values or expressions with a name

```
val one = 1
val add = 1 + one
```

- You cannot change `val` binding (immutable)

```
add = 2 //doesn't work
```

VARIABLES

- Use `var` instead of `val` if you want to change bindings

```
var one = 1
one = 2 // ok
```

FUNCTIONS

- Create functions with `def`

```
def addOne(m: Int): Int = m + 1
```

```
def three() = addOne(2)
three()
three
```

ANONYMOUS FUNCTIONS

```
(m: Int) => 1 + m
```

- Assign to a `val`

```
val three = (m: Int) => 1 + m
three(2)
```

- As args to other functions… map, flatMap, etc.

FUNCTION FORMATTING

- You can use brackets to organize functions into multiple lines

```
def timesTwo(i: Int): Int = {
  println("hello world")
  i * 2
}
```

FUNCTION FORMATTING

- Same for anonymous functions

```
{ i: Int =>
    println("hello world")
    i * 2
}
```

AHH BREATHE

- For those of us entirely new to Scala, feeling good and making progress

- Let’s take a glimpse at more advanced functional programming aspects of Scala

PARTIALLY APPLIED FUNCTIONS

```
def adder(m: Int, n: Int) = m + n
val add2 = adder(2, _:Int) // underscore wha?
add2(3)
```

CURRIED FUNCTIONS

```scala
def multiply(m: Int)(n: Int): Int = m * n

multiply(2)(3)

val timesTwo = multiply(2)(_)
timesTwo(5)

val timesThree = multiply(_)(3)
timesThree(3)
```

VARIABLE LENGTH ARGUMENTS

```
def capitalizeAll(args: String*) = {
  args.map { arg =>
    arg.capitalize
  }
}
```

```
capitalizeAll("abc", "def")
capitalizeAll("a", "b", "c")
```

TYPES

- Functions can also be generic and work on any type. When that occurs, you'll see a type parameter introduced with the square bracket syntax

```
def display[K](key: K) = {
  println(s"how about that ${key} !?")
}

display(10)
display("ten")
display(10F)
```

BREAK?

- Next up classes, inheritance, traits

CLASSES

CLASSES

- Hey, quick question, what are they?

WHAT ARE CLASSES?

- Static templates which can be instantiated into many objects at runtime

CLASSES

```
class Calculator {
  val brand: String = "HP"
  def add(m: Int, n: Int): Int = m + n
}
```

```
val calc = new Calculator
calc.add(1, 2)
calc.brand
```

CLASSES WITH A CONSTRUCTOR

```
class Calculator(brand: String) {
  val color: String = if (brand == "TI") {
    "blue"
  } else if (brand == "HP") {
    "black"
  } else {
    "white"
  }
  // An instance method.
  def add(m: Int, n: Int): Int = m + n
}

val calc = new Calculator("HP")
calc.color
```

CLASSES CAN SOMETIMES INHERIT FROM OTHER CLASSES

- Huh?

INHERITANCE

```
class ScientificCalculator(brand: String) extends Calculator(brand) {
  def log(m: Double, base: Double) = math.log(m) / math.log(base)
}
```

or for possible better readability

```
class ScientificCalculator(brand: String)
  extends Calculator(brand) {

  def log(m: Double, base: Double) = {
    math.log(m) / math.log(base)
  }
}
```

OVERLOADING METHODS

```
class EvenMoreScientificCalculator(brand: String)
extends ScientificCalculator(brand) {
  def log(m: Int): Double = log(m, math.exp(1))
}
```

CLASSES

- There's more to classes, but we covered the key constructs

- Any questions?

- Next up…. traits

TRAITS

- Collections of fields and behaviors that you can extend or mixin to your classes

```
trait Car {
  val brand: String
}

trait Shiny {
  val shineRefraction: Int
}
```

USING ONE TRAIT

```
class BMW extends Car {
// `extends` like a class (but wait)
  val brand = "BMW"
}
```

TRAITS (USING MORE THAN ONE)

```
class BMW extends Car with Shiny {
  // `extends` and `with`
  val brand = "BMW"
  val shineRefraction = 12
}
```

PART ONE COMPLETE

➤