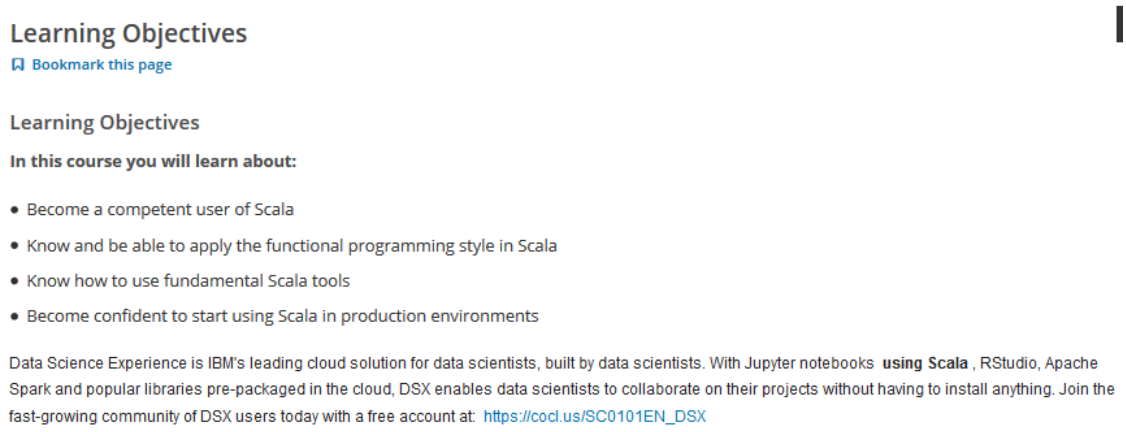
Scala 101



Syllabus

Syllabus

Lesson 1 - Introduction

- Introduction to Scala
- Creating a Scala Doc
- Creating a Scala Project
- The Scala REPL
- Scala Documentation

Lesson 2 - Basic Object Oriented Programming

- Classes
- Immutable and Mutable Fields
- Methods
- Default and Named Arguments
- Objects

Lesson 3 - Case Objects and Classes

- Companion Objects
- Case Classes and Case Objects
- Apply and Unapply

- Synthetic Methods
- Immutability and Thread Safety

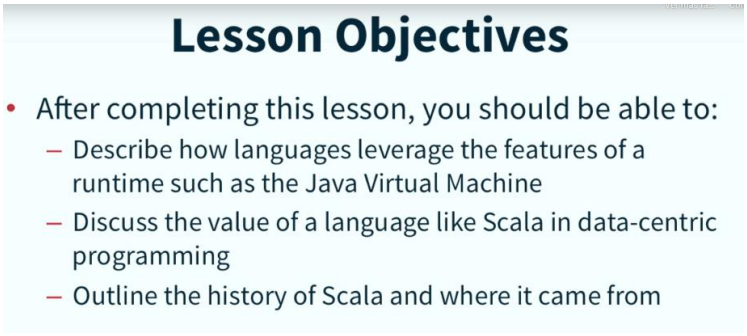
Lesson 4 - Collections

- Collections overview
- Sequences and Sets
- Options
- Tuples and Maps
- Higher Order Functions

Lesson 5 - Idiomatic Scala

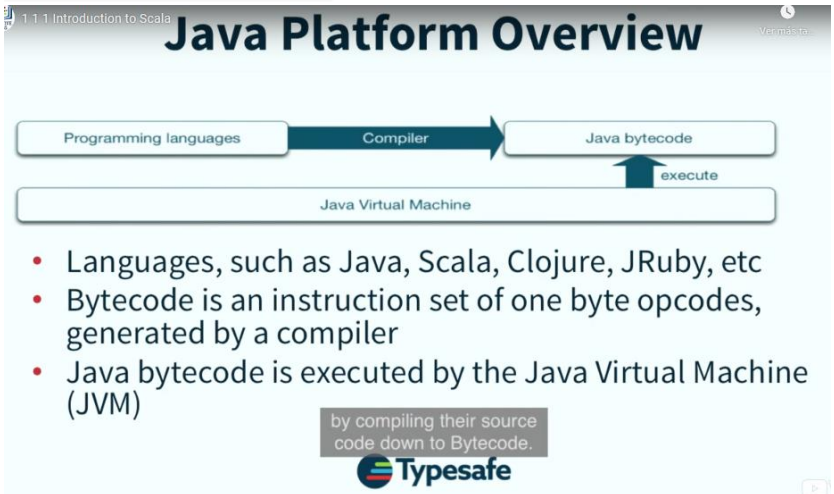
- For expressions
- Pattern Matching
- Handling Options
- Handling Failures
- Handling Futures

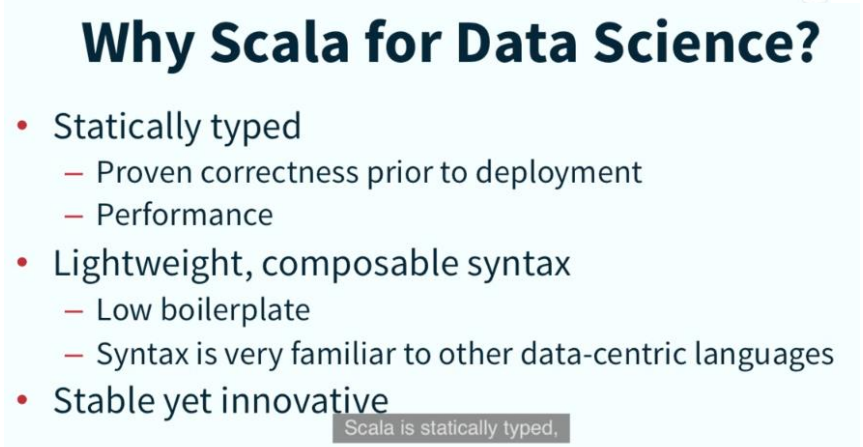
Lesson 1 - Introduction

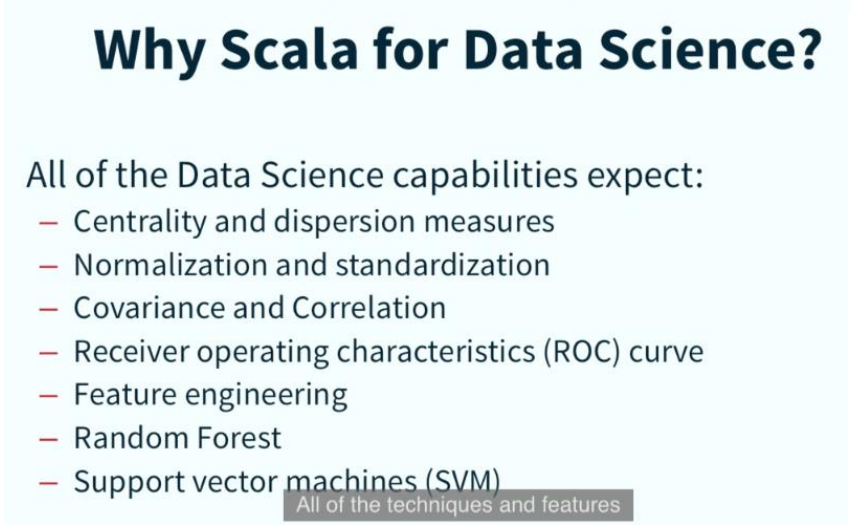


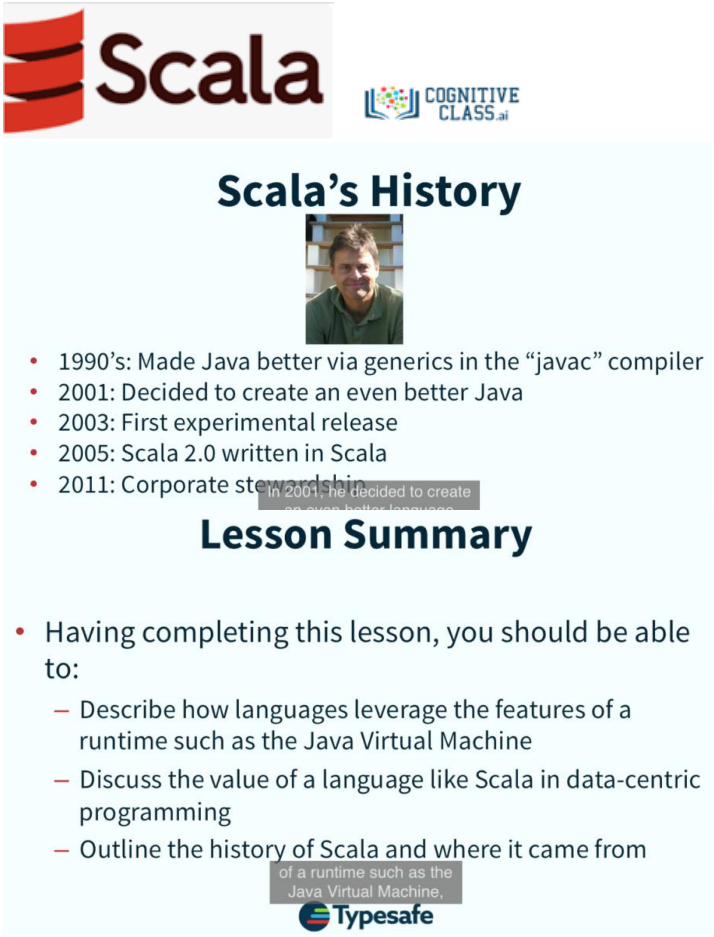
Scala

COGNITIVE CLASS.ai







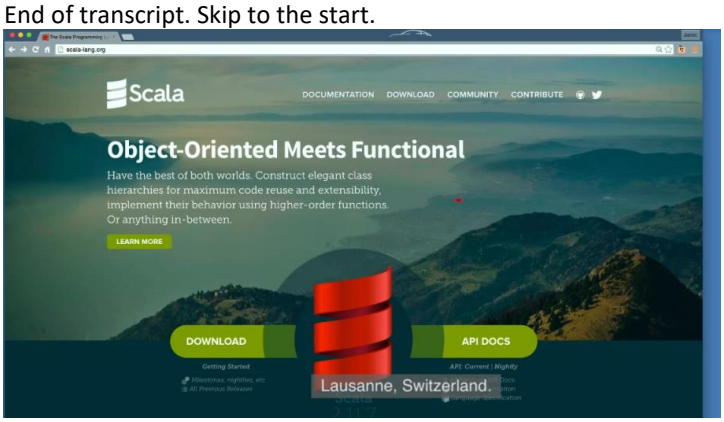


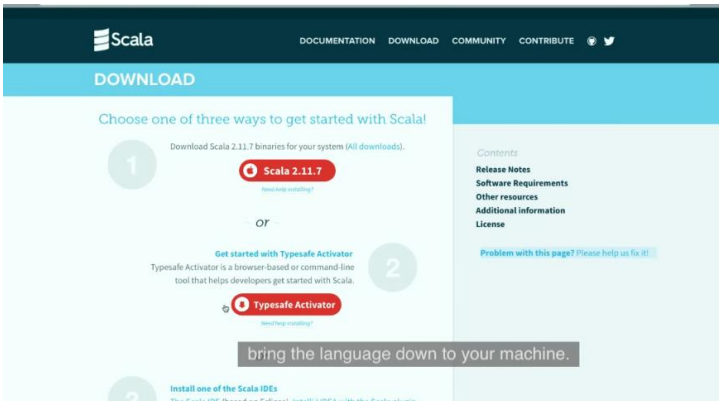
tart of transcript. Skip to the end.
welcome to introduction to Scala for data scientists after completing this lesson you should be able to describe how languages leverage that features of a runtime such as the Java Virtual Machine discuss the value of a language like skyline data-centric programming an outline history of Scotland where it came from
languages can be compiled to run on the Java Virtual Machine also called the JVM by compiling their source code down to bytecode these byte codes are an instruction set of one by top codes are generated by these compilers which is JBM can then execute on various platforms JVM to exist for all kinds of different runtime such as Linux UNIX Mac OS X Windows and more
java byte code is executed by the Java Virtual Machine so long as the JVM exists for the platform you want to run on so why scholar for data science scholars statically typed which means it's got proving correctness prior to deployment if you have a large job which is going to run for multiple hours you don't want to find out halfway through that language the road in was improving that this was a correct implementation you also get tremendous speed and performance from running on the JVM scholar also has modularity which means that you can build a structure of how your application should look so that you don't have one global namespace for all classes that are involved
languages that don't have this become difficult to maintain over time as they get larger scale is lightweight and has composable cent tax this means you have low boilerplate and you don't have to write quite as much code to do very simple things the syntax is very familiar to people who are used to
writing data-centric applications with other languages as well
scholars also stable yet innovative has been around for a long time but is also been used in the enterprise for many years in vertical such as the financial sector retail manufacturing gaming and more innovation continues to take place with Scala where people are finding new ways to prove correctness prior to deployment
so what does scholar bring to the table for people who want to do data science

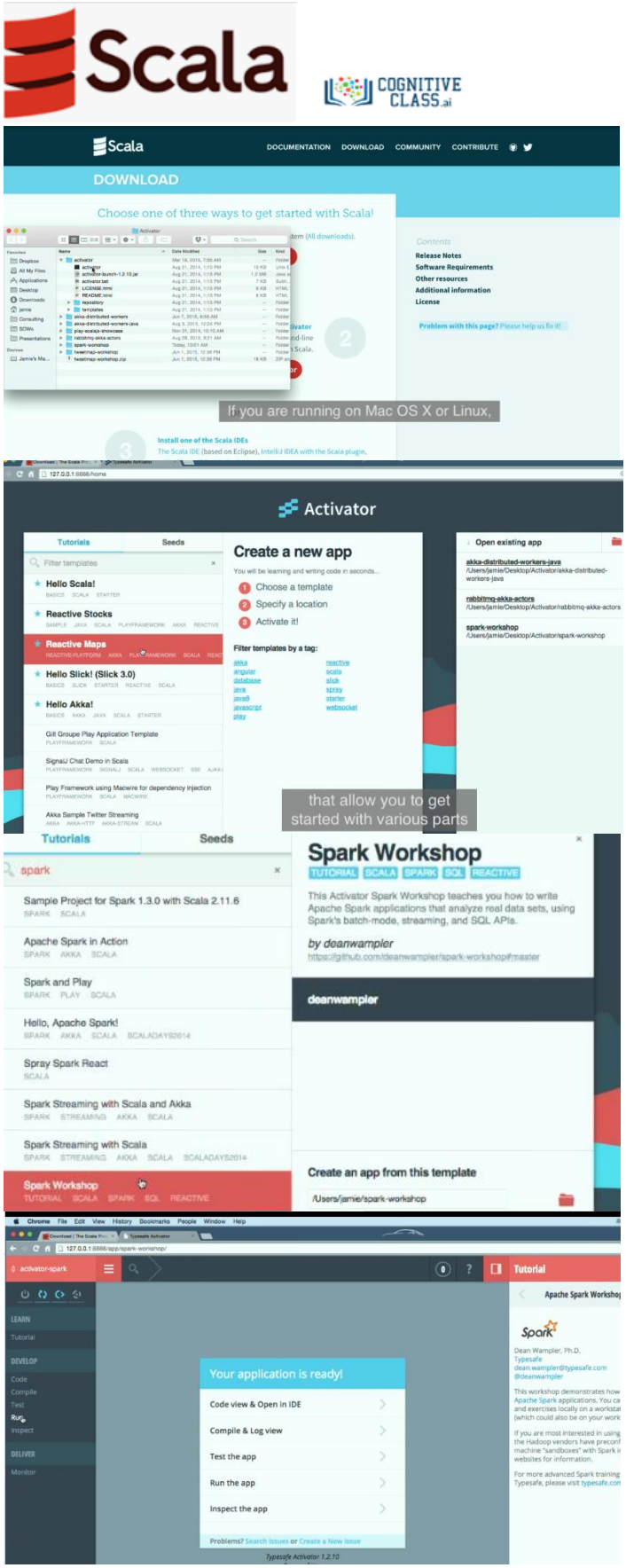
all of the techniques and features that you used to from other data-centric languages are also available in Scala you have centrality and dispersion measures normalization and standardization covariance and correlation your receiver operating characteristics also called ROC curve feature engineering Random Forests support vector machines and more so weird Scala come from marching orders he is the creator of Scala and he was working on the Java compiler prior to his work on Scala he put generics in this job to make sure that you knew when you got data out of a collection exactly what the types of data were in 2001 he decided to create an even better language for the JVM and in 2003 the first experimental release of scholar was brought out to the public by 2005 Scala was compiling itself in its own language which means we're proving that the language was correct at the time it compiled itself and in 2011 corporate stewardship was brought to bear such that you had somebody making sure that the language was enterprise ready at all times

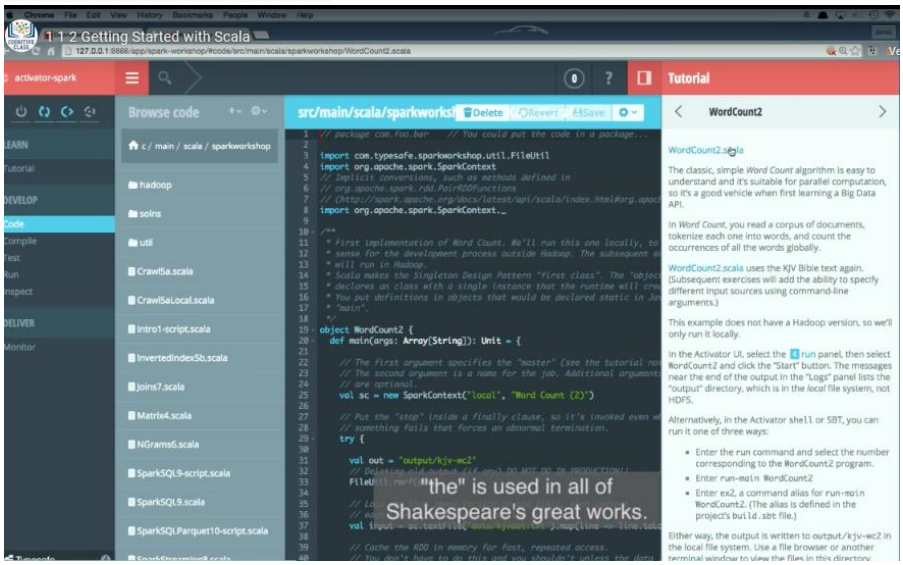
having completed this lesson you should be able to describe how languages leverage the features of a runtime such as the Java Virtual Machine discuss the value of a language like skyline data-centric programming an outline history of Scotland where it came from

End of transcript. Skip to the start.



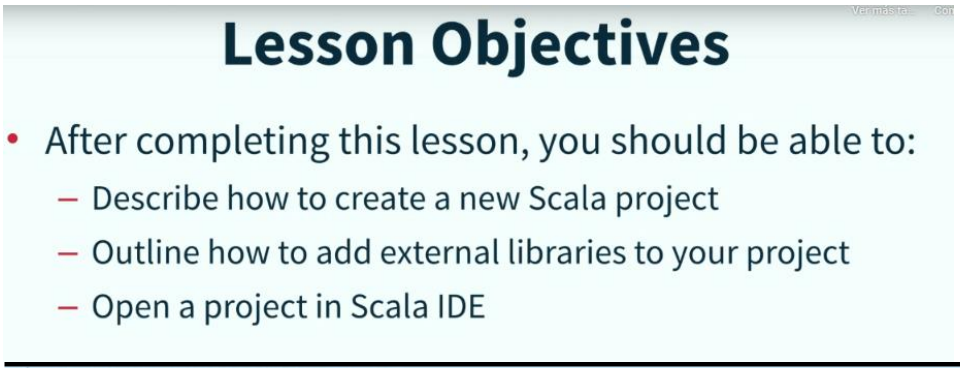


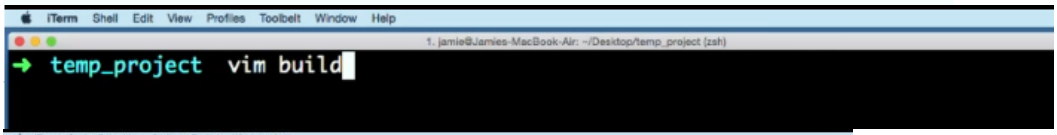

Scala
COGNITIVE CLASS.ai
DOCUMENTATION DOWNLOAD COMMUNITY CONTRIBUTE
DOWNLOAD
Choose one of three ways to get started with Scala!
Release Notes
Software Requirements
Other resources
Additional information
License
If you are running on Mac OS X or Linux,
Activator
Tutorials
Seeds
Create a new app
Choose a template
Specify a location
Activate it!
Open existing app
that allow you to get started with various parts
Spark Workshop
This Activator Spark Workshop teaches you how to write Apache Spark applications that analyze real data sets, using Spark's batch-mode, streaming, and SQL APIs.
by deanwampler
Create an app from this template
Your application is ready!
Code view & Open in IDE
Compile & Log view
Test the app
Run the app
Inspect the app

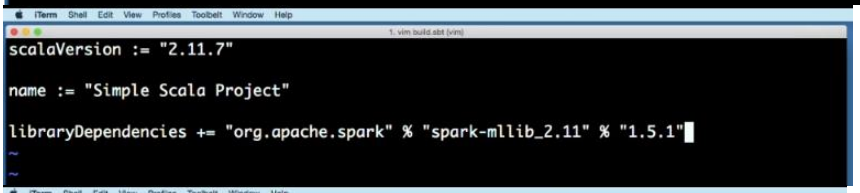


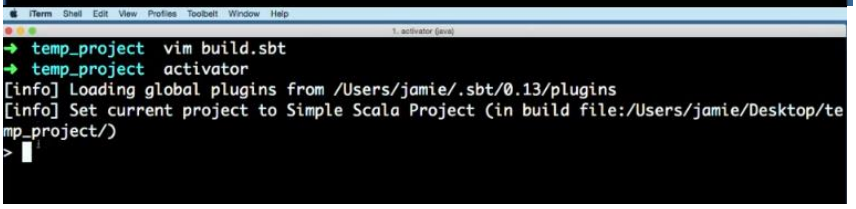
anscript. Skip to the end.
welcome to getting started with Scala after completing this lesson you should be able to describe how to get started with Scala on your machine and outline how to start a new project in Scala Scala language has a website called Scotland dash lang DAW tour with the beautiful picture of Lausanne Switzerland to get started with the language click on the download link on the front page which will give you the option of how you want to bring the language down to your machine you can download the language itself which will give you the compiler and tooling around how to get started with the project or you can download types of activator a tool which is very good for helping you start with templates that show you how to do specific things with Scala what I've downloaded types of activator I have to decompress the zip file has been brought down in this case I put it on my desktop if I open the folder that was from the compressed zip file you'll see you have a folder inside of their called activator with scripts inside of allowing you to run activator on various platforms if you were running on Mac OS X or Linux use the activator shell script if you're running on Windows based platforms use activator batch file I start the activator script and it starts at Shell session which is going to figure out whether it has the most recent version of activator and then start a web UI there are many templates inside of activated allow you to get started with various parts of the skull ecosystem if you want to just start learning scholar you can select hello Scala template other templates include various parts of the ecosystem such as the concurrency toolkit the play web framework and spark for big data handling we also have seeds were you have a project starting point without any functionality already built into it if I look at the tutorials and I searched on spark a bunch of templates show up I'm going to select the spark workshop template I want to choose where on my desk this template will live in this case I will put in my desktop in the activator folder choose and then I press the Create button at this point all of the dependencies this project requires will be brought down from various repositories across the internet
there could be quite a few in this may take some time however it's a one-time cost you do it the first time and after that in immediately loads when you open the template you'll see a new user interface which allows you to explore a tutorial about this project it also has a view to allow you to see the code compile the code executes the tests associated with this project and run the application if that's part of this template the spark workshop template will explain to you what patchy spark is how it's used in Hadoop context and has

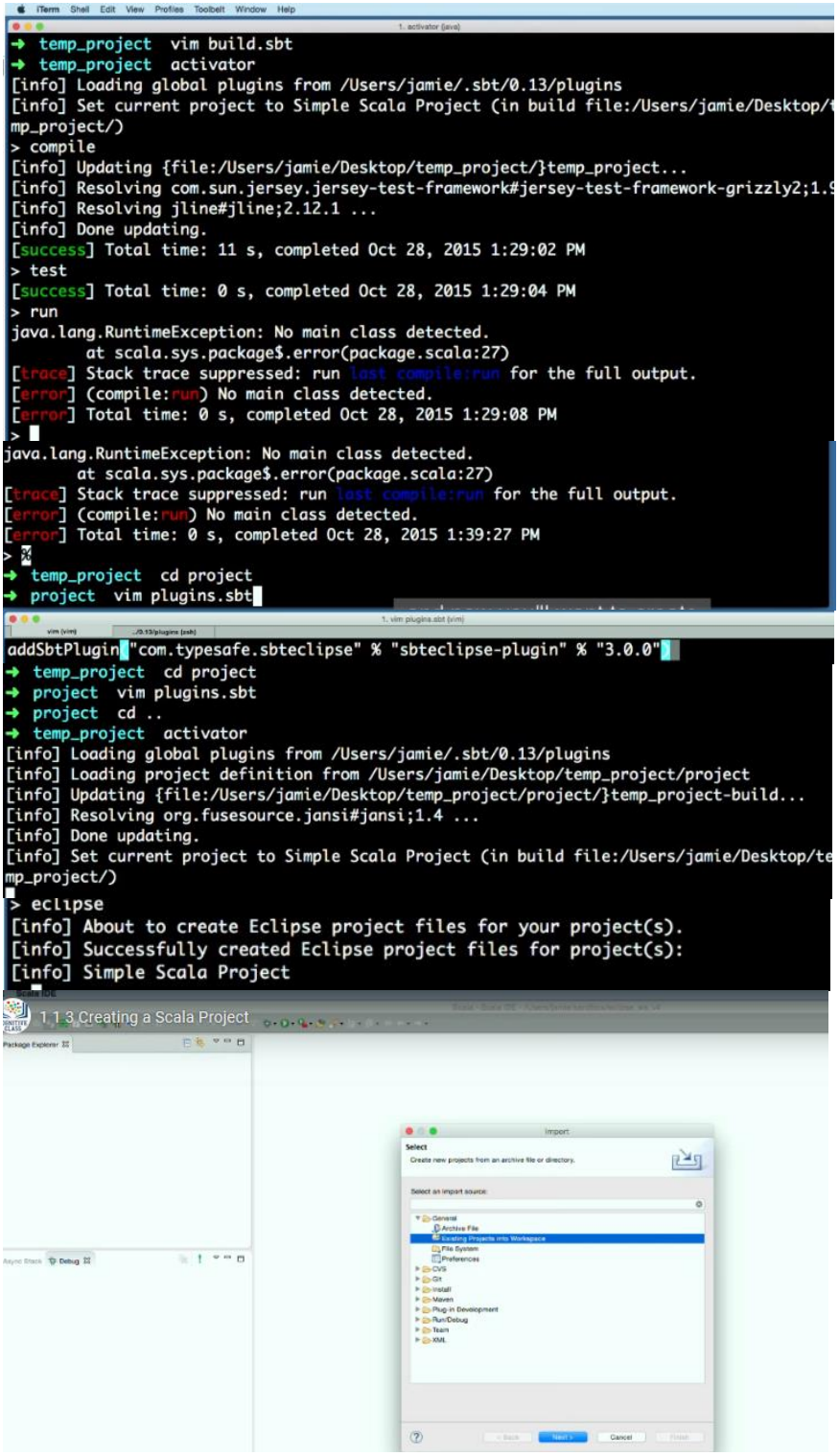
several examples that are considered canonical in the Big Data space such as word count if we navigate through the tutorial you will see that we eventually reach a point where we are describing the various implementations of these applications so here is an example code word count to word count being the kind of application where you're counting the number of times the word is used in all of Shakespeare's great works this is a very simple application it's not that large the comments themselves make up for more of the source code then the logic itself and you can also execute the tests so that one by one the various parts of the project can be executed to be sure that they're implemented correctly we look at the code again we can move on to the other projects such as word count three the matrix for the crawl which is the fifth application in spark the inverted index and and grams which is natural language processing when you're finished using activated to get started you can bring the source code down to your machine locally
clicking on this button right here which will allow you to open the project and the IntelliJ IDEA IDE or the Eclipse IDE or I down to your file system so that you can use it with Sublime Text Emacs VMware any other editor you prefer after having completed this lesson we should understand and be able to describe how to get started with Scala as well as be able to use activator to get started quickly with templates and seed projects
End of transcript. Skip to the start.

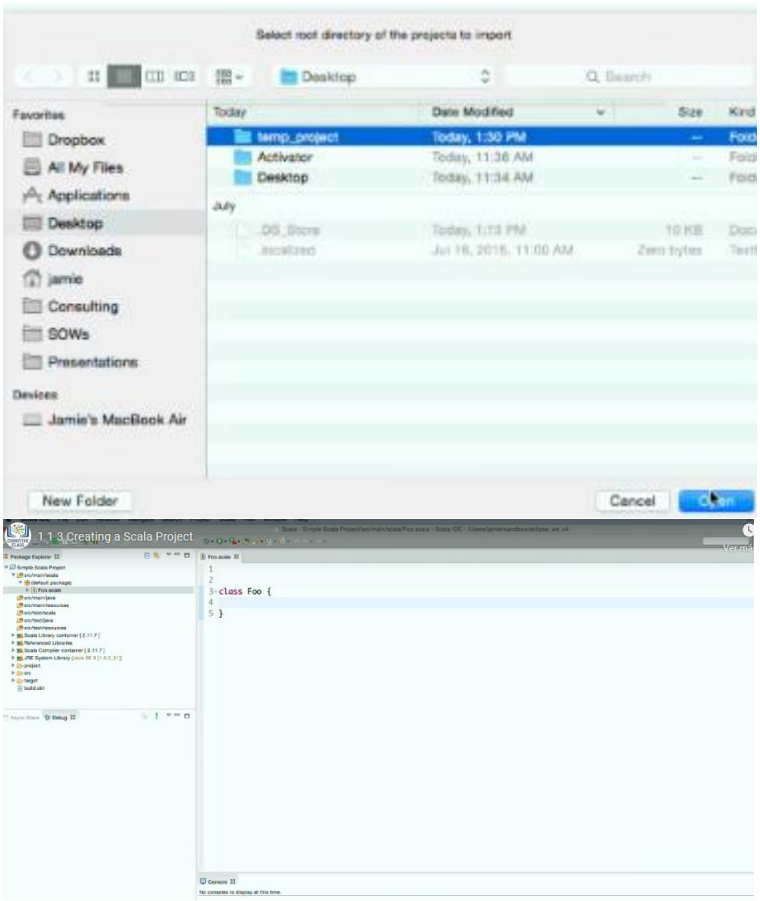






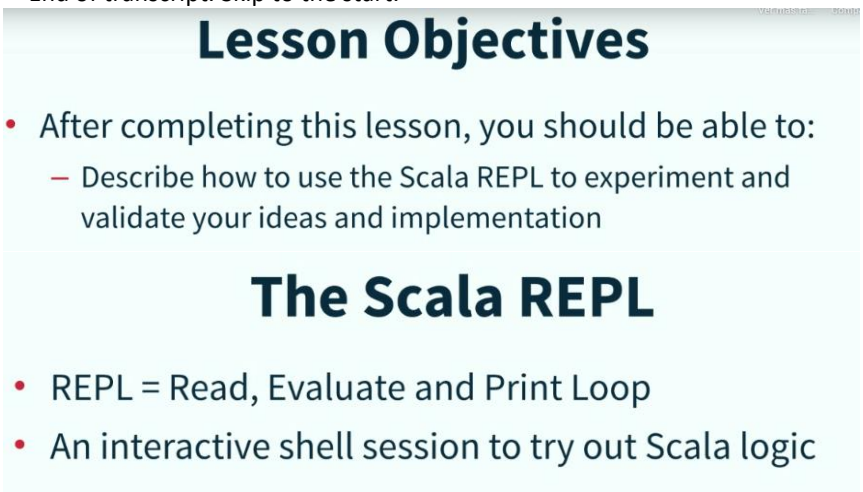



iTerm Shell Edit View Profiles Toolbelt Window Help
1. activator (java)
➜ temp_project vim build.sbt
➜ temp_project activator
[info] Loading global plugins from /Users/jamie/.sbt/0.13/plugins
[info] Set current project to Simple Scala Project (in build file:/Users/jamie/Desktop/t
mp_project/)
> compile
[info] Updating {file:/Users/jamie/Desktop/temp_project/}temp_project...
[info] Resolving com.sun.jersey.jersey-test-framework#jersey-test-framework-grizzly2;1.9
[info] Resolving jline#jline;2.12.1 ...
[info] Done updating.
[success] Total time: 11 s, completed Oct 28, 2015 1:29:02 PM
> test
[success] Total time: 0 s, completed Oct 28, 2015 1:29:04 PM
> run
java.lang.RuntimeException: No main class detected.
at scala.sys.package$.error(package.scala:27)
[trace] Stack trace suppressed: run last compile:run for the full output.
[error] (compile:run) No main class detected.
[error] Total time: 0 s, completed Oct 28, 2015 1:29:08 PM
>
java.lang.RuntimeException: No main class detected.
at scala.sys.package$.error(package.scala:27)
[trace] Stack trace suppressed: run last compile:run for the full output.
[error] (compile:run) No main class detected.
[error] Total time: 0 s, completed Oct 28, 2015 1:39:27 PM
>
➜ temp_project cd project
➜ project vim plugins.sbt
1. vim plugins.sbt (vim)
addSbtPlugin("com.typesafe.sbteclipse" % "sbteclipse-plugin" % "3.0.0")
➜ temp_project cd project
➜ project vim plugins.sbt
➜ project cd ..
➜ temp_project activator
[info] Loading global plugins from /Users/jamie/.sbt/0.13/plugins
[info] Loading project definition from /Users/jamie/Desktop/temp_project/project
[info] Updating {file:/Users/jamie/Desktop/temp_project/project/}temp_project-build...
[info] Resolving org.fusesource.jansi#jansi;1.4 ...
[info] Done updating.
[info] Set current project to Simple Scala Project (in build file:/Users/jamie/Desktop/te
mp_project/)
> eclipse
[info] About to create Eclipse project files for your project(s).
[info] Successfully created Eclipse project files for project(s):
[info] Simple Scala Project
1.1.3 Creating a Scala Project
Import
Select
Create new projects from an archive file or directory.
Select an import source:
Next >
Cancel



Start of transcript. Skip to the end.
welcome to creating a scallop project after completing this lesson we should be able to describe how to create a new scallop project outline how to add external libraries such spark to your project and then open the project inside of Scala IDE to create a new scallop project created directory somewhere on your disk at that point you're going to want to edit a new file called a build-out SPT inside of that fire you only need three things first of all the version of scholar you want to use
the name of the project and finally any dependencies that I may want to add to my project such as libraries that may come from other people who created them that you want to leverage inside of your project to do that we add the library dependencies we give the groupId of who created the project that we want to use the name of the project we want to pull in the Persian scholar with which it is being used and then finally the version of the project itself that we are going to be leveraging once we have these three things we have the basis of an entire project in Scala and now I type activator to start my activator session inside of this project and instead of bringing up the WebUI as we saw earlier because it knows it's inside of an SBT project it will start a build session inside of that project when you receive your prompt which is this carrot right here you can now start typing in commands Command Sgt or commands such as compile to compile all of the source inside of the project you can also run the test command which will run any tests that you have inside of your project or run if you have any classes that start up a JVM and an application in this case we don't have any source yet so there's nothing to run in order to open her project in Scala IDE first we have to make sure that we have the Eclipse plugin added to our activator project to do that we have to leave activator and I find the best way to leave activator is to use control D once you're back at the root of the project you want to move into the

project folder which was created for you when you created this project and ran activator so CD into project folder and now you want to create a plugins . SBT file inside of this file you're going to want to add a single line you want to say add SPT plugin where you add the group com type saved at SBT clips and then the project SPT like Eclipse plugin and the version 3200 as it currently stands once you've done this save the file
return to the temp project folder and then type activator again this will reload the entire context of all of the information about this project including what you just add it now if you type eclipse the command line inside of the SBT prompt it's going to create the Eclipse project files for you so that you can open this project inside Eclipse if you use IntelliJ IntelliJ has the ability to read your build-down SBT file and you don't need to do this step 2 downloads khallid go to Scala dash I D dot org
where you'll find the latest links to download the latest version of the idea itself once you've done that and you have the ideal local on your box you can start the I D which will load eclipse with the Scala IDE plugin as well this may take a few minutes but once it's up and running you don't have to add your project to the Package Explorer inside of Eclipse now even though this is a brand new project we've already created it inside of our shells session and we need to import an existing project into Eclipse to do that we need to right click inside the Package Explorer or go to the file
and select Import once we done that this dialogue should show up asking us what kind of project we want to import we want to say in the general group that we want to bring in an existing project into our current workspace we say next and then we have to select the location of our temp project that we were using in my case it's going to be on my desktop and select the term project I see open and I should see my projects view and side of eclipse the simple Scala project that I created when I say finish eclipse is going to bring it into the Eclipse context and now whenever expand it I'll see that there's a whole bunch of stuff here that was created for me
such as the location for our I want to put my Scala source files any Java source files any resource files such as configuration and we might be using then we also have locations for our test source files as well if I want to create a class I merely right click on the source main Scala and I say new and select scholar class and I can call that anything I want for example through and then the class will be created underneath that grouping
End of transcript. Skip to the start.




```
➜ temp_project  activator
[info] Loading global plugins from /Users/jamie/.sbt/0.13/plugins
[info] Set current project to Simple Scala Project (in build file:/Users/jamie/Desktop/temp_projec
t/)
> console
[info] Starting scala interpreter...
[info]
Welcome to Scala version 2.11.7 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_51).
Type in expressions to have them evaluated.
Type :help for more information.

scala> 1 to 10000
res0: scala.collection.immutable.Range.Inclusive = Range(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37,
38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62
, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86,
87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 1
09, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128,
 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 14
8, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167,
168, 169, 170...
scala>
```

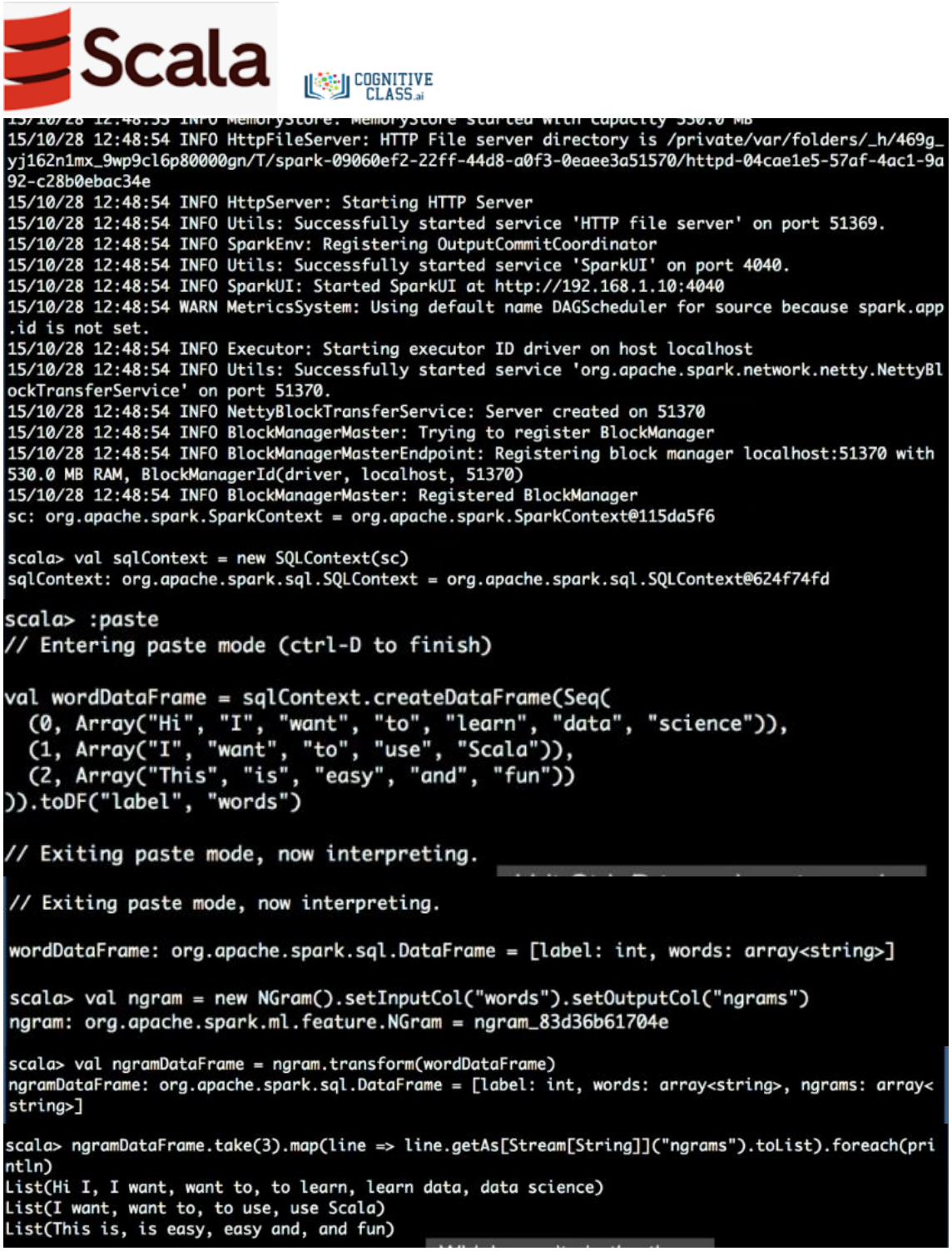
```
scala> res0.map(number => number + 1)
res1: scala.collection.immutable.IndexedSeq[Int] = Vector(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38
, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62,
63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87
, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109
, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 1
29, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148,
 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 16
8, 169, 170, ...
scala>
```

I get a new result in this case, result one.

```
scala> 1 to 20
res3: scala.collection.immutable.Range.Inclusive = Range(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
, 14, 15, 16, 17, 18, 19, 20)

scala> res3.groupBy(number => number % 3)
res4: scala.collection.immutable.Map[Int,scala.collection.immutable.IndexedSeq[Int]] = Map(2 -> Ve
ctor(2, 5, 8, 11, 14, 17, 20), 1 -> Vector(1, 4, 7, 10, 13, 16, 19), 0 -> Vector(3, 6, 9, 12, 15,
18))
```

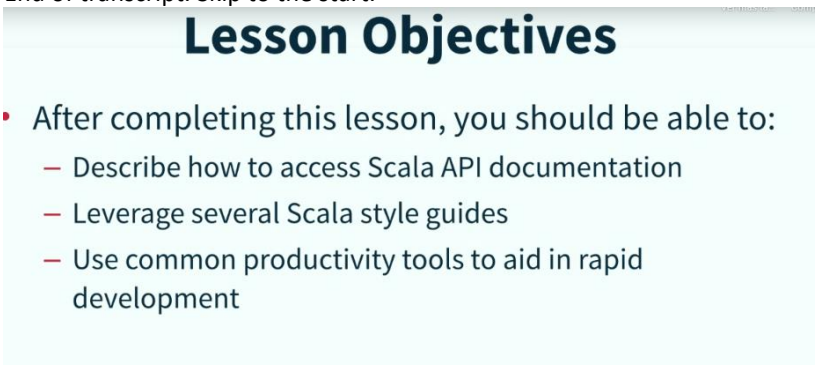
```
scala> :paste
// Entering paste mode (ctrl-D to finish)

import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.sql.SQLContext
import org.apache.spark.ml.feature.NGram

// Exiting paste mode, now interpreting.

import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.sql.SQLContext
import org.apache.spark.ml.feature.NGram
```

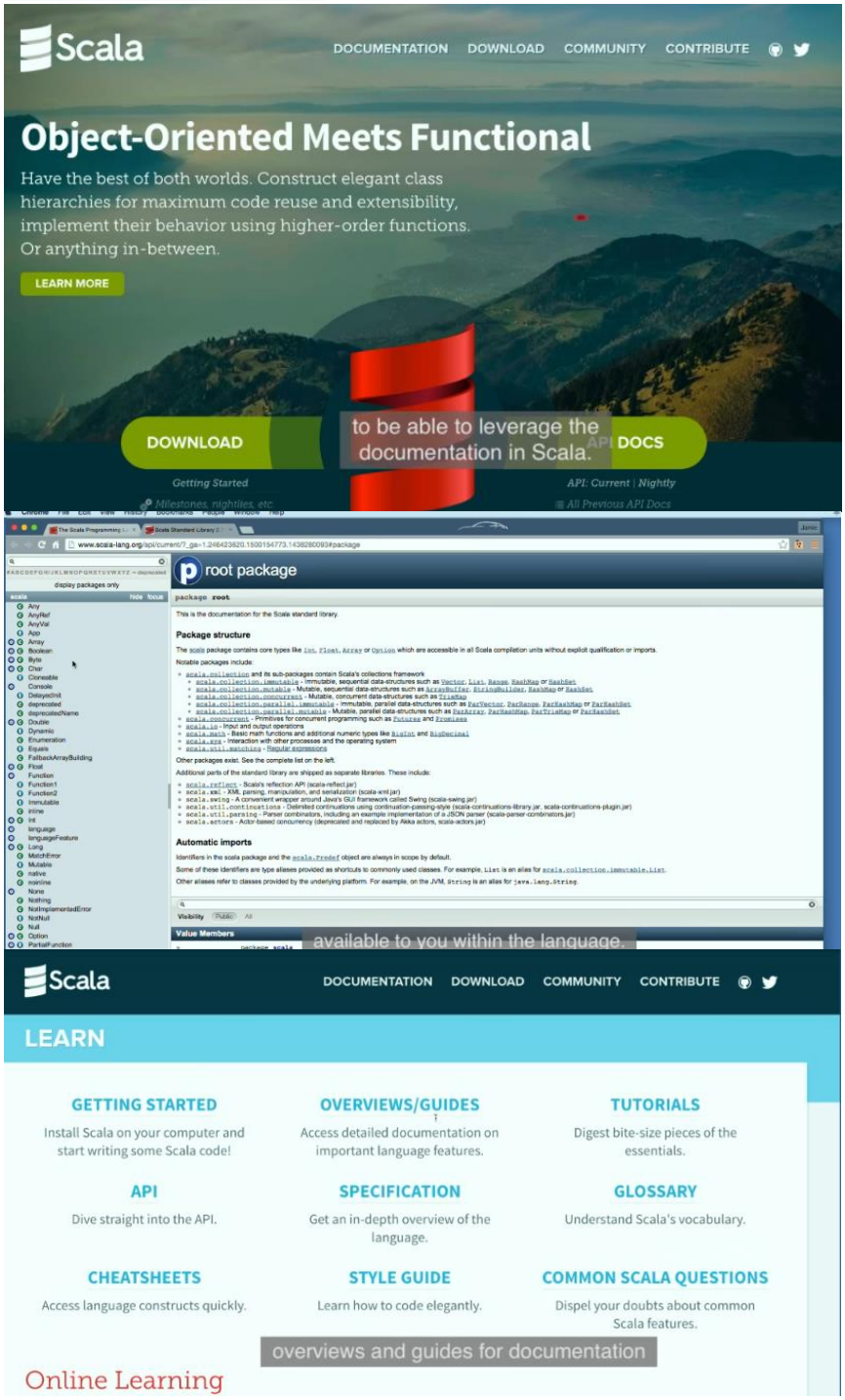
```
scala> val conf = new SparkConf().setAppName("temp-project").setMaster("local[2]")
conf: org.apache.spark.SparkConf = org.apache.spark.SparkConf@35665291
```

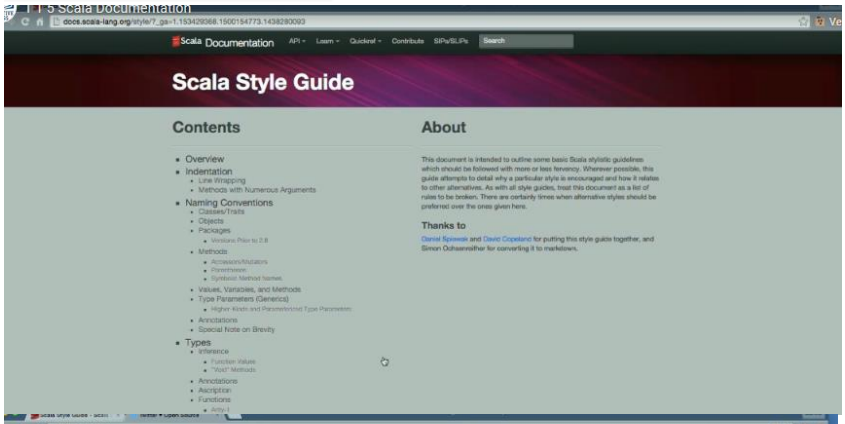


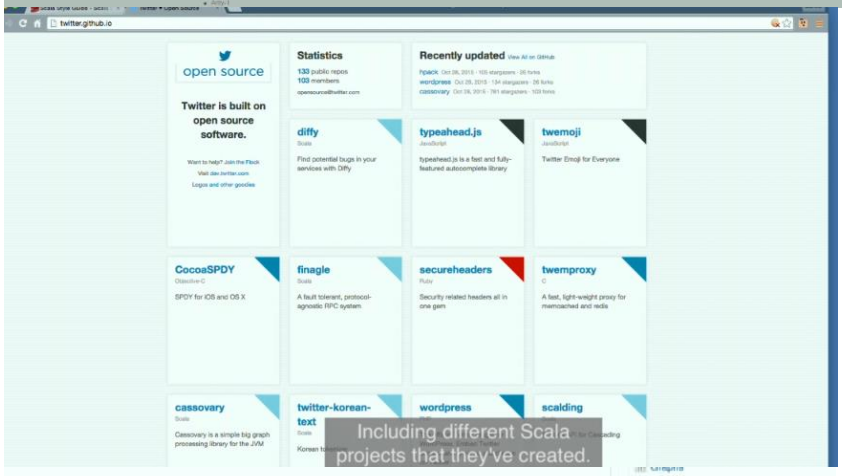
welcome to the Scala rebel in this lesson you will learn how to describe
how to use the Scala report to experiment invalidate your ideas and
implementation before you put it into production
scholar Apple is also called the read evaluate and print loop is a tool that
many languages have in Scala does as well as an interactive shell session to
try out logic before you deploy it into production to enter a rebel session
first of all you have to have an activator project that you want to use
just as we saw in the previous lesson so now that I have my temp project start
activator inside of the root folder of that project and then I take the console
command once I'm at the activator prompt you should now see a Scala prom showing
that you have a console session open in the rubble and I can very easily create
data such as one to ten thousand creating a sequence of numbers from one
to ten thousand inside of a collection you'll notice that the value is
automatically assigned to name called result 0 and if I perform any kind of

transformation such as mapping over the data and saying for every number add one to it I get a new result in this case result one with the values change based upon the transformation that I gave me if I look back at my result 0 you'll see it still exists and hasn't been altered in any way shape or form I can also perform very basic data science operations inside of the rebel for example if I create a range of values from one to 20 I can say for that result three I want to group them by their remainder whenever I say for every number get the modulus of dividing by three
in this case I end up with a collection where the index is the remainder and the value associated with that key is all of values that had that remainder I can also perform spark operations inside of the rubble when we showed off her activator templates earlier we saw an example of how to do in grams and grams or just a sequence of tokens typically words of which there are a number of them we can use the anagram class and sparked transform input features into input grammar anagrams and to do so the first thing I'm going to do is use paste mode inside of the Repsol to paste in a group of import statements and I hit ctrl D which has all of them interpreted as a single command next I'm going to create a spark contacts and don't worry about what sparked context or any of these concepts are yet as will be explaining more about them later but this is just an example of some simple functionality that you can build inside of your Apple distemper project and I have to give it a master of the spark configuration now have a spark in Fig
an hour can create a spark context
using their config next I can create a sequel context from that spark context next less space them in new data frame representation values that we want to convert into an anagram in this case three different sentences index by an index number
control deed and peace mode and evaluate this word data frame now I can create and then gramm by creating an instance of a new and Graham class where I have to set some arguments to the new instance
and then I create an anagram data frame by transforming the word data frame I just created and then I can perform a transformation on the anagram data frame which allow me to say take the first three lines and map over them where it's safe for each line get it as a stream of strength applying type safety as we go on a call and grams and I'm it turned into a list I'm now going to print them out his inaction by saying for each Franklin which results in the three streams being printed out as sentences in a list
End of transcript. Skip to the start.

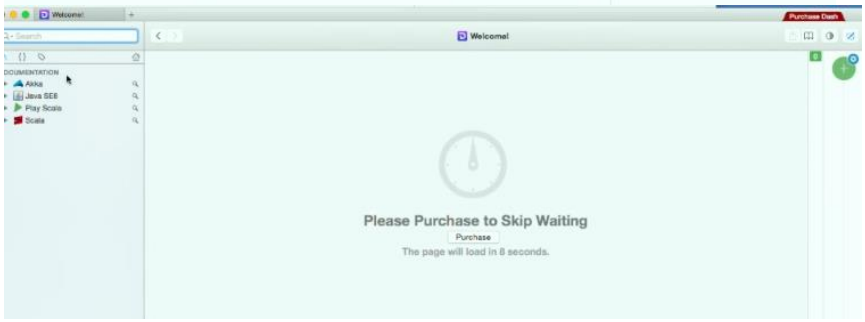


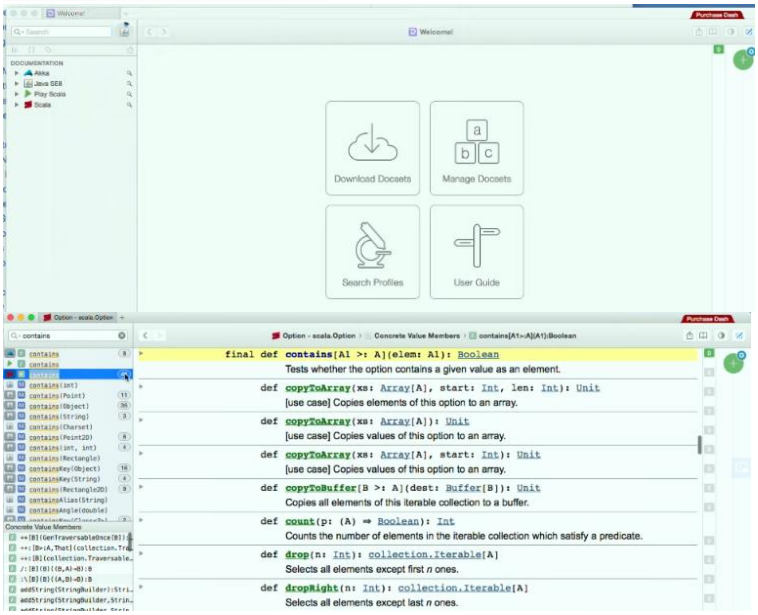

Scala
DOCUMENTATION DOWNLOAD COMMUNITY CONTRIBUTE
Object-Oriented Meets Functional
Have the best of both worlds. Construct elegant class hierarchies for maximum code reuse and extensibility, implement their behavior using higher-order functions. Or anything in-between.
LEARN MORE
DOWNLOAD
to be able to leverage the documentation in Scala.
API DOCS
Getting Started
API: Current | Nightly
Milestones, nightlies, etc.
All Previous API Docs
root package
available to you within the language.
Scala
DOCUMENTATION DOWNLOAD COMMUNITY CONTRIBUTE
LEARN
GETTING STARTED
Install Scala on your computer and start writing some Scala code!
OVERVIEWS/GUIDES
Access detailed documentation on important language features.
TUTORIALS
Digest bite-size pieces of the essentials.
API
Dive straight into the API.
SPECIFICATION
Get an in-depth overview of the language.
GLOSSARY
Understand Scala's vocabulary.
CHEATSHEETS
Access language constructs quickly.
STYLE GUIDE
Learn how to code elegantly.
COMMON SCALA QUESTIONS
Dispel your doubts about common Scala features.
overviews and guides for documentation
Online Learning


Scala Style Guide
Contents
About
Overview
Indentation
Naming Conventions
Types
Thanks to


open source
Twitter is built on open source software.
Statistics
Recently updated
diffy
typeahead.js
twemoji
CocoaSPDY
finagle
secureheaders
twemproxy
cassovary
twitter-korean-text
wordpress
scalding
Including different Scala projects that they've created.


Databricks Scala Guide
With over 800 contributors, Spark is to the best of our knowledge the largest open-source project in Big Data and the most active project written in Scala. This guide draws from our experience coaching and working with engineers contributing to Spark as well as our Databricks engineering team.
Code is written once by its author, but read and modified multiple times by lots of other engineers. As most bugs actually come from future modification of the code, we need to optimize our codebase for long-term, global readability and maintainability. The best way to achieve this is to write simple code.
Scala is an incredibly powerful
following guidelines work well
team, your mileage might vary.
github.com/databricks/scala-styles-guide.
Table of Contents
HTTPS clone URL
https://github.com
You can clone with HTTPS, SSH, or Subversion.
Clone in Desktop
Download ZIP


Welcome!
Please Purchase to Skip Waiting
Purchase
The page will load in 8 seconds.



nscript. Skip to the end.
welcome to the Scala rebel in this lesson you will learn how to describe
how to use the Scala report to experiment invalidate your ideas and
implementation before you put it into production
scholar Apple is also called the read evaluate and print loop is a tool that
many languages have in Scala does as well as an interactive shell session to
try out logic before you deploy it into production to enter a rebel session
first of all you have to have an activator project that you want to use
just as we saw in the previous lesson so now that I have my temp project start
activator inside of the root folder of that project and then I take the console
command once I'm at the activator prompt you should now see a Scala prom showing
that you have a console session open in the rubble and I can very easily create
data such as one to ten thousand creating a sequence of numbers from one
to ten thousand inside of a collection you'll notice that the value is
automatically assigned to name called result 0 and if I perform any kind of
transformation such as mapping over the data and saying for every number add one
to it I get a new result in this case result one with the values change based
upon the transformation that I gave me if I look back at my result 0 you'll see
it still exists and hasn't been altered in any way shape or form I can also
perform very basic data science operations inside of the rebel for
example if I create a range of values from one to 20 I can say for that result
three I want to group them by their remainder whenever I say for every
number get the modulus of dividing by three
in this case I end up with a collection where the index is the remainder and the
value associated with that key is all of values that had that remainder I can
also perform spark operations inside of the rubble when we showed off her
activator templates earlier we saw an example of how to do in grams and grams
or just a sequence of tokens typically words of which there are a number of
them we can use the anagram class and sparked transform input features into
input grammar anagrams and to do so the first thing I'm going to do is use paste
mode inside of the Repsol to paste in a group of import statements and I hit
ctrl D which has all of them interpreted as a single command next I'm going to
create a spark contacts and don't worry about what sparked context or any of
these concepts are yet as will be explaining more about them later but
this is just an example of some simple functionality that you can build inside
of your Apple distemper project and I have to give it a master of the spark
configuration now have a spark in Fig

an hour can create a spark context
using their config next I can create a sequel context from that spark context
next less space them in new data frame representation values that we want to
convert into an anagram in this case three different sentences index by an
index number
control deed and peace mode and evaluate this word data frame now I can create
and then gramm by creating an instance of a new and Graham class where I have
to set some arguments to the new instance
and then I create an anagram data frame by transforming the word data frame I
just created and then I can perform a transformation on the anagram data frame
which allow me to say take the first three lines and map over them where it's
safe for each line get it as a stream of strength applying type safety as we go
on a call and grams and I'm it turned into a list I'm now going to print them
out his inaction by saying for each Franklin which results in the three
streams being printed out as sentences in a list
End of transcript. Skip to the start.

>LAB 1

Module 1: Introduction

All labs are written in notebooks instead of ripples. The operations are similar, and you can always come back and see what you did previously. There are some subtle differences but nothing major. Let's go over a few examples in the first module

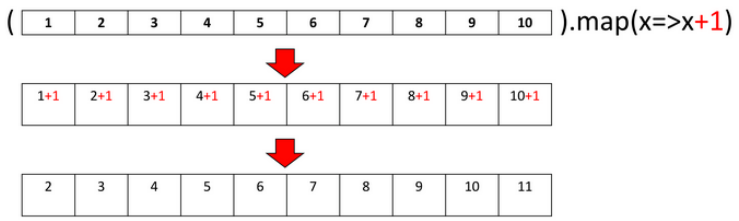
Using Notebooks instead of the ripple

Like in the video, you can create data, for example, you can generate a sequence of numbers:

```
1 to 10
```

```
Range(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

We can increase the value of the element of each number by using the map function as illustrated in the following figure:



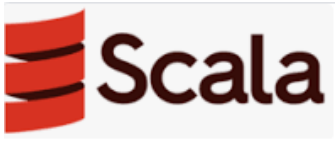
Example of the map in function

It just takes one line of code:

```
(1 to 10).map(x => x + 1)
```

```
Vector(2, 3, 4, 5, 6, 7, 8, 9, 10, 11)
```

The variable `res0` does not exist in the notebook. However, we can explicitly bind a name to a value by using the familiar `=` syntax. The declaration `val a = 1 to 10` says that we want to bind the result of `1 to 10` to the name `a`. It is important to note that when we use `val` we're creating an immutable value. This is the reason why we call it a binding rather than assignment. Once we bind a name to a value, that name cannot be re-bound to the a different value in the same scope, even if the types are the same. You can try this out below. Try to bind a different value to `a`. To do this complete change the line that just contains `a` to be `a = 10 to 15` and observe the compiler error.



```
[3]: val a = 1 to 10
     //sometimes If you don't leave the variable at the end you will not display the results
     a.map(x=>x)

     a = Range(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
[3]: Vector(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

Functions do not mutate the objects they are call on. For instance you can call the `map` function on `a`. This will return a new value.

```
[4]: a.map(x => x + 1)

[4]: Vector(2, 3, 4, 5, 6, 7, 8, 9, 10, 11)
```

the variable `a` has not changed. We have just returned a new value.

```
[5]: a

[5]: Range(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

In order to be able to re-assign values to a name you can use the `var` syntax.

```
[6]: var b = a.map(x => x + 1)
     b

     b = Vector(2, 3, 4, 5, 6, 7, 8, 9, 10, 11)
[6]: Vector(2, 3, 4, 5, 6, 7, 8, 9, 10, 11)
```

Since we declared `b` to be a mutable `var` rather than an immutable `val` we can assign values to it more than once. For instance,

```
[*]: b=1 to 10
     b
```

Note that assigning values multiple times to a single `var`, they have to be the same type. Try assigning a value of type `String` to `b` and observe what happens. To do this, change the line containing `b` to read `b = "hello there"` and observe the error.

```
:  b="hello there"

:  Name: Compile Error
   Message: <console>:27: error: type mismatch;
    found   : String("hello there")
    required: scala.collection.immutable.IndexedSeq[Int]
          b="hello there"
            ^

   StackTrace:
```

In general, prefer using `val` to `var`. It is much easier to understand and think about code when we do not have to consider that it may be mutated by some operation. Working without mutation makes writing multi-threaded programs significantly easier.

Lesson 2 - Basic Object Oriented Programming

Learning objectives

2.1 Classes

After completing this lesson, you should be able to:

- Create and instantiate classes in Scala
- Describe how arguments are passed to Scala class instances
- Outline the lifespan of class parameters in a Scala class instance

2.2. Immutable and Mutable Fields

After completing this lesson, you should be able to:

- Describe the difference between mutable and immutable fields
- Create fields in Scala classes
- Describe the difference between class parameters and fields
- Outline how to promote class parameters to fields

2.3. Methods

After completing this lesson, you should be able to:

- Implement methods in Scala

- Describe evaluation order of methods versus fields in Scala
- Outline how infix notation works in Scala

2.4. Default and Named Arguments

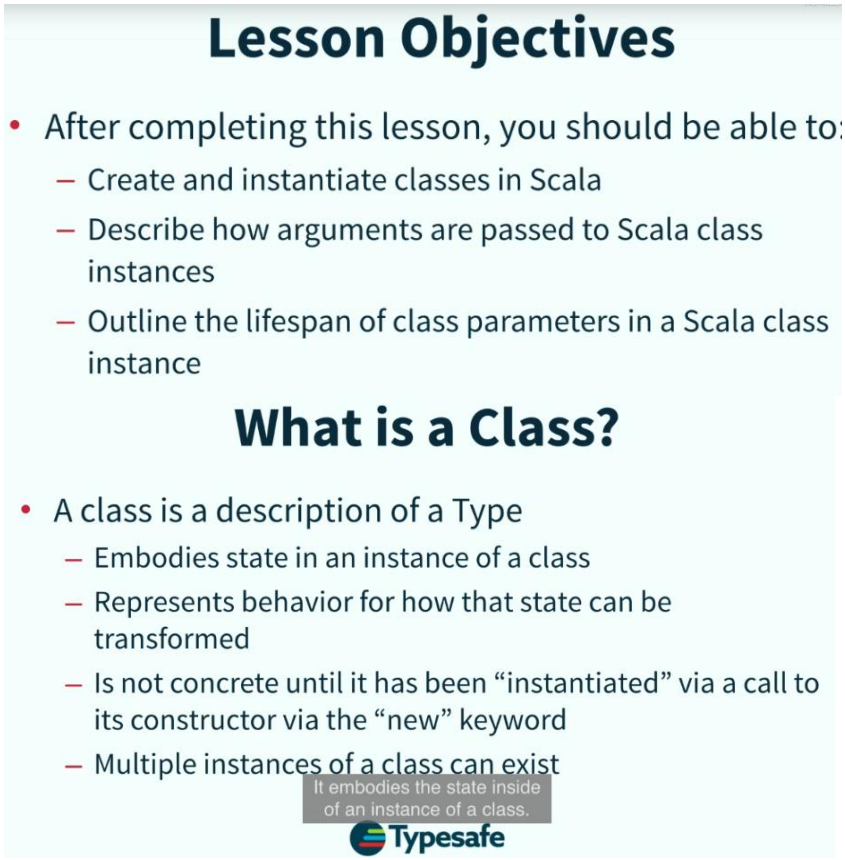
After completing this lesson, you should be able to:

- Utilize default argument values in Scala class constructors and methods
- Leverage named arguments to only pass certain values

2.5. Objects

After completing this lesson, you should be able to:

- Create Singleton objects in Scala
- Describe the difference between a class and an object in Scala
- Outline usages for objects in Scala applications
- Start a Scala application



Scala COGNITIVE CLASS.ai

A Simple Scala Class

```
scala> class Hello
defined class Hello

scala> new Hello()
res0: Hello = Hello@33bd6867

scala> res0.toString
res1: String = Hello@33bd6867
```

The Primary Constructor

- Each class gets a primary constructor automatically
  - Defines the “signature” of how to create an instance
  - The body of the class is the implementation of the constructor

The Primary Constructor

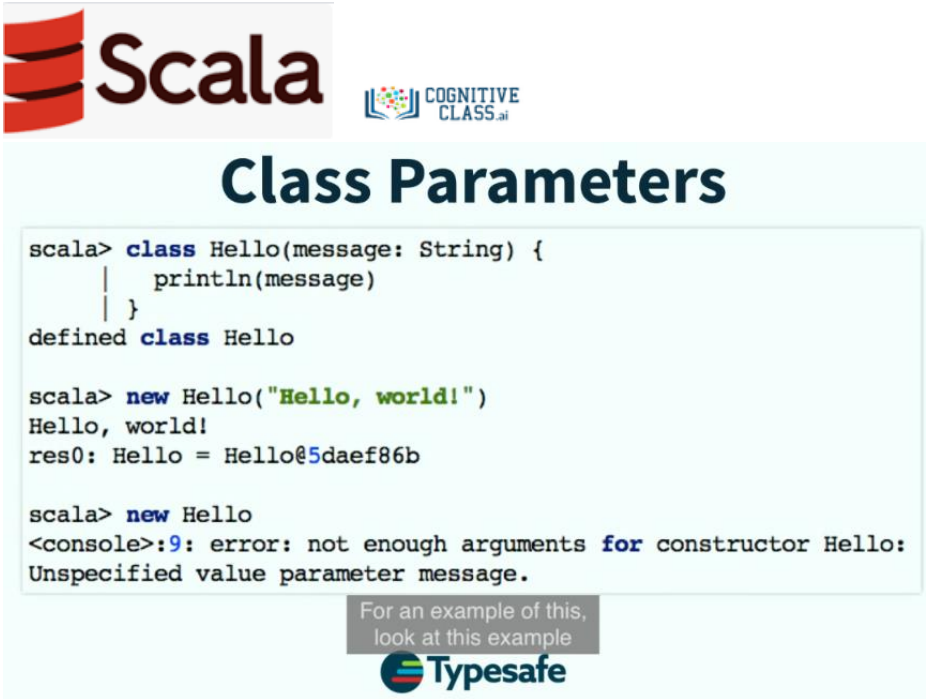
```
scala> class Hello {
     |   println("Hello")
     | }
defined class Hello

scala> new Hello()
Hello
res0: Hello = Hello@7f68f33c
```

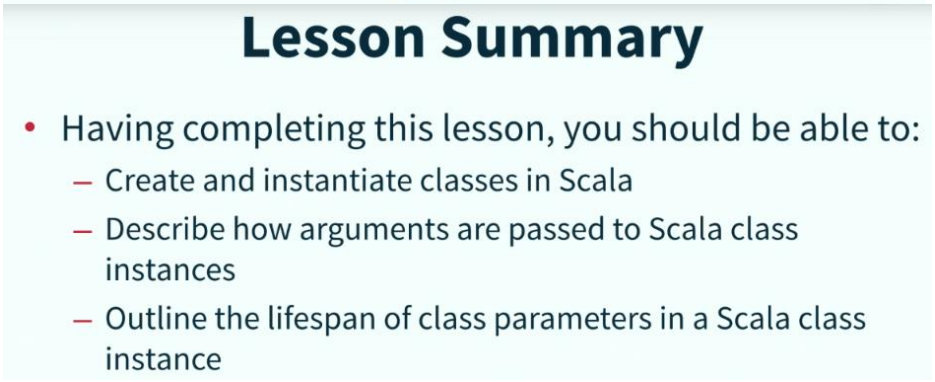
Class Parameters

- You can pass values into an instance of a class using one or more parameters to the constructor
  - You must specify the type of the parameter
  - The values are internally visible for the life of the class instance
  - They cannot be accessed from outside of the class instance

a specific person and you pass values in as paramaters

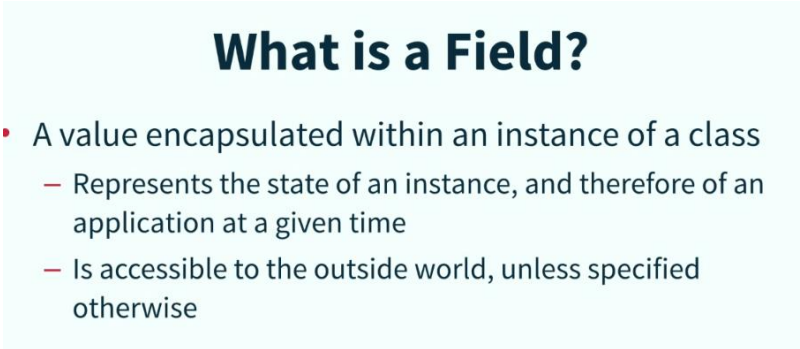
Typesafe





welcome to classes before we can get started doing any serious data science work we first have to understand basic syntax in Scala after completing this lesson you should be able to create instantiate classes in Scala describe how arguments are passed to scholar class instances and outline the lifespan of class parameters and a scholar class instance so what is a Class A classes a description of a type it embodies the state inside of an instance of a class so it is nothing more than a way of representing
type of value inside of your system for example think of a customer inside of a customer you might have states such as their name or their address or their phone number
classes represent behavior for how that state can be transformed for example how you might change someone's address or their name classes are not concrete unless they have been instantiated by a call to their constructor with the new keyword so it class merely describes how you would want to represent concepts such as a customer and then to have an instance of a customer that represents a specific person you have to instantiate the Customer class with the new keyword and multiple instances of a class can exist for example you would have multiple customers near system one would hope
here's a simple Scala class within the rebel we take the keyword class and give it a name
hello this is a very basic class representation which doesn't contain any state or any methods which were described later however we can create an instance of a Hello class by saying new Halo with parentheses at the end of it and an instance is created given the name Res 0 inside of the rebel we can then call a method on the instance of a low by saying Res 0
to strength which will return a string representation of the instance of the

class in this case its location inside of the Java heap the memory space inside of the JVM each class gets a primary constructor automatically from the Scala compiler this defines the signature of how we create instances of a class the body of the class inside of the Scala source code is the implementation of the constructor so again if we look at our class hello that we were defining inside of a rebel we have an open curly brace and close curly brace and with end those curly braces is the body of the class if we put a statement inside of their such as a print money to pronounce the word hello that will create an instance of a class that when constructed using the new keyword will print out hello you can pass values into an instance of a class using one or more parameters to the constructor but you must specify the types of each of these parameters these values are internally visible for the life of the class instances you are creating so if you create a customer that represents a specific person and you pass values in as parameters to that class instance those will be visible inside of the class the entire time that customer object is living however those parameters cannot be accessed from outside of the class instance an example of this look at this example from the rubble we have a class hello and now we have a parameter called message of type string inside of the body of the class in the constructor we have a printed statement to print out the message that is passed into our fellow instance this defines our class alone and then we create the new Halo instance impasse in axed our strength it has been printed out when the class is constructed if we try to create hello without passing in that value we get a compile error before we try to run it saying that we do not have
arguments for the constructor of hello and those class parameters are not accessible if I have that class hello with the message of type string as a parameter and I instantiate an instance of hollow and then I tried to access the message value inside of the Hello instance Res 0
I'm told by the compiler days not visible and again this happens before you put this into production
having completed this lesson you should be able to create and instantiate classes and Scala describe how arguments are passed scholar class instances and outline the lifespan of class parameters and a scholar class instance
End of transcript. Skip to the start.



COGNITIVE CLASS.ai

Immutable Fields

```
scala> class Hello {
     |   val message: String = "Hello"
     | }
defined class Hello

scala> (new Hello).message
res0: String = Hello
```

Mutable Fields

```
scala> class Hello {
     |   var message: String = "Hello"
     | }
defined class Hello

scala> val hello = new Hello
hello: Hello = Hello@3617a35c

scala> hello.message = "Hello, world!"
hello.message: String = Hello, world!
```

we then can access the message

Typesafe

Immutable or Mutable?

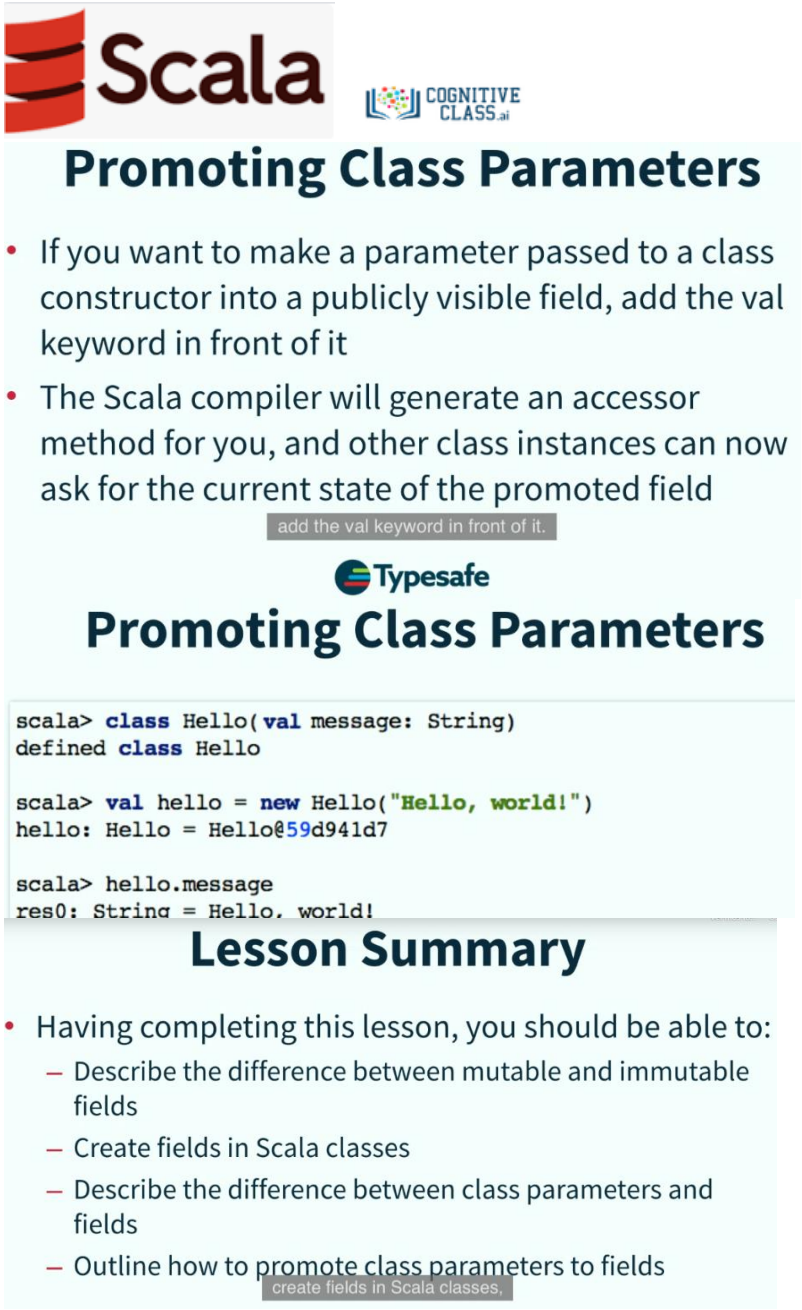
- Immutable fields cannot be changed and are therefore “threadsafe” in a multithreaded environment, such as the JVM
- Mutable fields can be useful, but require diligence to ensure that multiple threads cannot update the field at the same time

Specify Types

- Scala has “type inference”
- It is a good habit to be specific about types anyway

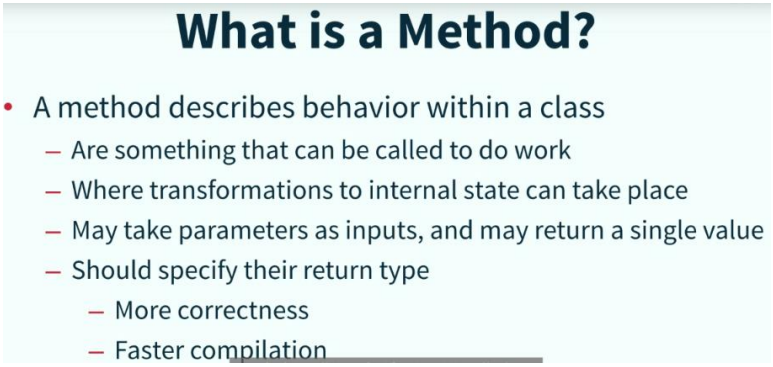
```
scala> class Hello {
     |   val message = "Hello"
     | }
defined class Hello

scala> (new Hello).message
res0: String = Hello
```

It is a good habit to be



welcome to immutable and mutable fields after completing this lesson you should be able to describe the difference between mutable and immutable fields create field since college classes describe the difference between class parameters and fields and outline how to promote class parameters to become fields themselves so what is the field of field assessed value encapsulated within an instance of a class that represents the state inside of that instance and therefore the state of an application at a given time think of a customer as we spoke of earlier in the classes section that customer may have fields inside of it that represent its customer name its customer address and its customer phone number that is state inside of your application which may change at various times fields are also accessible to the outside world unless specified otherwise by this we mean another class is able to ask an instance of a customer for that customers name so what is the difference between fields and class parameters parameters are passed into the constructor of a class and are only visible within that class fields exist inside of the body of the class and are accessible to outside instances for immutable fields we can create them using the BAL keyword immutable is exactly what it sounds like it cannot be changed so inside of a class hello we define a field message using the BAL keyword type string and we set it equal to the value hello this means when we instantiate a new hello class instance

we can access the message field however no one is able to change it at any point in time we also have mutable fields using the bar keyword and this means that the field that you create can be changed
point to a different strengths in this case we have a class alone we create a field called message which is a bar in there for mutable of type string and it points to a string called hello when we create a new instance of hello we then can access the message or we can set it equal to a new value and if we look at the bottom of the slide BC where hello doubt message is being assigned to a new value of hello world instead of just how low so how do we choose when to be immutable or mutable immutable fields cannot be changed and are therefore thread safe inside of a multithreaded environment such as the GBM this is a very desirable property because you don't have to worry about anybody else changing about you on you unexpectedly mutable fields can be useful but they require significant diligence on the part of the programmer to ensure that multiple threads cannot update the field at the same time leading to non-deterministic behavior soyuz immutable fields by default is easier to reason about immutable fields and it will require as much work on your end to make sure that you aren't updating fields at the same time by definition Scala makes our class parameters immutable by default is also important to specify the types Scala has type in prince but this isn't something you want to over use is a good habit to be specific about types regardless if we look at our class hello on the slide we see that we have a field called message but we never say it is equal to a type of strength instead by assigning it to a string the compiler infers that message is a string instance however this is not how we want to write your code it is less clear and does not enforce correctness at compile time we can also promote class parameters become fields themselves if you want to make a parameters as being passed to a class constructor a publicly visible field at the Val keyword in front of it
the Scala compiler will generate an access
for you and other class instances can now ask for the current state of the promoted field here's an example I have my class hello and I still have my parameter to the class called message of type strength but by putting the BAL keyword in front of it when I create an instance of a hello I can now ask to see the message value and is passed out to me having completed this lesson we should be able to describe the difference between beautiful and immutable fields create fields in Scala classes describe the difference between class parameters in fields and outline how to promote class parameters to fields
End of transcript. Skip to the start.



A Simple Scala Method

```
scala> def hello = "Hello"
hello: String

scala> def echo(message: String): String = message
echo: (message: String)String
```

Why Methods Instead of Fields?

- Methods can look like fields
- Methods are evaluated at the time they are called
- Methods are re-evaluated every time they are called
- Fields are only evaluated at the time the class is constructed, and if immutable, only one time

Infix Notation

- Methods are called on an instance of a class
- Scala permits methods to be called with no "." or parentheses, if the method takes only one argument
- This is flexible syntax that supports powerful DSLs
- For readability, you should not use this feature

Infix Method Calling

```
scala> "Martin Odersky".split(" ")
res0: Array[String] = Array(Martin, Odersky)

scala> "Martin Odersky" split " "
res1: Array[String] = Array(Martin, Odersky)
```

Lesson Summary

Having completing this lesson, you should be able to:

- Implement methods in Scala
- Describe evaluation order of methods versus fields in Scala
- Outline how infix notation works in Scala

welcome two methods after completing this lesson you should be able to implement methods with them Scala classes described evaluation order methods vs field in Scala an outline how index notation works inside of Scala to what is a method a method described behavior with them a class instance there something that can be called on to do work on the state that exists inside of a class this is where transformations the state can take place methods may or may not take parameters as inputs and may return a single value in response if any and they should specify their return type if one exists this gives us more correctness a compile-time it also allows for faster compilation is the compiler doesn't have to figure out what subsequent usages of

a value might do and whether or not
operations on them are valid if you specify types the compiler can make
those judgments quicker so here's examples of a simple Scala method first
of all we have the Deaf method hello using the Deaf keyword to specify that
we are creating a method and the Hello method simply returns a Hello strength
in this case we are using type inference because we are not specifying the Hello
returns a string is merely inferred from the fact that we have a string that is
returned from a low method however another example is this method echo
again using the Deaf keyword echo takes a parameter which is a strength called
message and it returns a string explicitly and we set that equal to the
message that was passed in in this case we have a fully formed method that takes
about you and returns another value even though in this case it's merely echoing
the message that was passed
past and so why use methods instead of fields if a method can look like a field
for example and our method hello on the previous slide methods are evaluated
every time they're called at the time they are called so they are re-evaluated
the next time you call them again
fields on the other hand are only evaluated at the time the class is
constructed and immutable only the first time this gives you a balance between
evaluation order methods are evaluated every time you call them
fields are evaluated only when the class is constructed and one-time scholar also
supports something called in pics notation methods can be called on an
instance of a class and Scala permits those calls to be made with no dot or
parentheses but only if the method takes only one argument this is flexible
syntax that supports very powerful D S l's domain-specific languages that allow
you to abstract the way you want to describe a problem in source code
however for readability you should not use this feature let me show you what I
mean
in the first example we have a string Martin order ski and we use dot notation
to call the method split on that strength and we say to split that string
using a space this results in an array of strings of two values the string
Martin and the string oh dear ski because we split the single string by a
space we could have called this in the exact same way but without the dot and
the parenthesis because the split method only takes one argument we are treating
the split method almost as though it is an operator like a plus sign or a minus
sign or multiplication this syntax is very readable whenever you want to write
domain-specific languages for example whenever you write tests using certain
kind
libraries but even though it results in the same thing it could be more
difficult for new developers to Scala to understand syntax when they see no dots
and no per ends we recommend that you stick with ordinary syntex using dots
and per ends so there is no cognitive overhead for new people to your team
having completed this lesson you should be able to implement methods and scholar
using the Deaf keyword described the evaluation order methods forces fields
in Scala and outline how infix notation works
End of transcript. Skip to the start.

Default Argumets

- Allows the developer to specify a value to use for a constructor or method when none is passed by the caller, and omit values that are frequently the same
- Reduces boilerplate in application source code because you don't have to "overload" methods with different signatures

Default Arguments

```
def name(first: String =  "", last: String = ""): String =
  first + " " + last

scala> name("Martin")
res0: String = Martin
```

Best Practice

- If you have a mixture of default arguments and those that do not have a default value, put the arguments without defaults first

```
def name(first: String,
         last: String,
         middle: String = ""): String =
  first + " " +  middle + " " + last
```

Named Argumets

- Leading arguments can be omitted if they have default values
- You can specify only the values you want to pass

Named Arguments

```
scala> name(last = "Odersky")
res0: String = Odersky

scala> name(first = "Martin", last = "Odersky")
res0: String = Martin Odersky
```

welcome to default and named arguments after completing this lesson you should be able to utilize default argument values in Scala class constructors and methods and then leverage named arguments to only pass certain values when you use a method or constructor with default arguments so what are default arguments they allow the developer to specify about you to use for a constructor method when none is passed by the collar and omit values that are frequently the same this reduces boilerplate an application source code base because you don't have to overload methods with different signatures overloading is a language feature and some languages which allows you to have multiple definitions of the same method taking different arguments so here's an example of default arguments inside of a method called name we have two parameters the first is called first and the second is called last they're both strings and they both have default values of empty strings when you call this method is going to concatenate the first name a space and then the last name in the case where you call the name method and only passed the first argument is going to assume that the last value has not been passed and it should use the empty string in its place so when it prints out the result of this method with the first a space and the last name concatenated together you see a string value of just the word Martin as a best practice if you have a mixture of default arguments and those that do not have default values but the arguments without the defaults first for example let's look at that may method again we have the first middle and last name that we went to pass into this method however not everybody has a middle name and in this case we use a default value of an empty string
for middle we put the first and the last rings first and then middle last so that when we call this week and just passed first and last name and not worry about the middle if one doesn't exist
named arguments allow you to emit leading arguments if they have default values you can specify only the values you want to pass as an example let's look at named arguments in news for our named method that we used before where we had first and last name with default values of an empty string if we call name and we only want to pass the last name but it's a last is equal to the value we want to pass in there were referencing the name of the parameter in the call to the method so that we can only set that value we could also be explicit about both of them and say name first is equal to Martin last is equal to order ski and get both values into our method with default values were neither default was used
having completed this lesson you should be able to utilize default argument values in Scala class constructors and methods and leverage named arguments to only pass certain values
End of transcript. Skip to the start.

What is an Object?

The Singleton Pattern

- Defines a single instance of a class that cannot be recreated within a single JVM instance
- Can be directly accessed via its name

A Simple Scala Object

```
object Hello {
  def message = "Hello!"
}

scala> Hello.message
res0: java.lang.String = "Hello!"
```

Why is this Useful?

- Many languages permit the definition of “static” fields and methods
- These are globally available within the runtime, such as a JVM
- They are not related to specific instances of a class

When are Objects Used?

- Class Factories
- Utility methods
- Constant definitions

A Simple Object

```
object Hello {
  val oneHourInMinutes: Int = 60

  def createTimeFromMinutes(minutes: Int) =
    new Time(
      minutes / oneHourInMinutes,
      minutes % oneHourInMinutes)
}
```

Starting a Scala Application

```
object Hello {
  def main(args: Array[String]): Unit =
    println("Hello!")
}
```

Starting a Scala Application

```
$ scala -cp target/scala-2.11/classes/ Hello
Hello
```

```
> run
[info] Running Hello
Hello
[success] Total time: 0 s, completed Jul 20, 2012 6:00:20 PM
```

Lesson Summary

Having completing this lesson, you should be able to:

- Create Singleton objects in Scala
- Describe the difference between a class and an object in Scala
- Outline usages for objects in Scala applications
- Start a Scala application

welcome to objects after completing this lesson you should be able to create singleton objects in Scala describe the difference between a class and an object in Scala outline usages for objects in school applications and be able to start a scholar application so what is an object in the nineteen nineties a very famous book came out called design patterns and included a pattern called the singleton pattern this defines a single instance of a class that cannot be recreated with them a single run time such as a JDM instance they can be directly accessed by in their name so here's an example of a very simple Scala object instead of using the class keyword we use the object keyword and call the object hello inside of this object is a method called message which returns a string hello you'll notice that we are able to use the message method inside of the Hello object without instantiate during an instance of hello there is no keywords such as new and use we use it directly this is because this object is a singleton instance and is already instantiated for us when we tried to call the message method objects are Linsley instantiated so that they are instantiated the first time you tried to use them why is this useful many languages permit the definition of static fields and methods which are not tied to a particular instance of a class there globally available within the runtime such as the JVM so when are scholar objects used a very frequent use cases to create factories for instances of classes so this is an abstraction or usage is not tied directly to an instance of a class but instead results in the creation of instances of classes they're also good for utility method
which do not leverage any state from outside of themselves
anything they use is what is passed as a parameter and then they returned some

value based on a calculation or transformation objects are also used to
house constant definitions and have a look at some examples here is our object
hello again and inside we have a constant defined as an immutable value
called one hour in minutes
it is a type int and is sixty as a value we also have a method called create time
from minutes which takes a minute's value of type int and it damn instead
she ate a new time instants for us using the minutes passed in divided by the
constant we already defined and then the minutes modded by the same value this is
a factory method for time instances to start a scholar application we put a
main method inside of an object the object can be called anything you want
the Deaf main has to follow a specific convention where it has some arguments
of a type array of string and returns a type of unit which is a rough equivalent
value to avoid in the JVM it then can do whatever you want inside of the
bootstrapping mean methods such as create other classes and services to
start the skull application from outside of itself we say the Scala keyword to
start the JVM process and to the classpath we add whatever classes we
want to use such as the Hello class we just to find that hello object and then
if we wanted to run inside an SBT or activator context we use the run command
either one of these would work
having completed this lesson you should be able to create singleton objects in
Scala describe the difference between a class and an object in Scala outline
usages for objects in school applications and start a scholar
application
End of transcript. Skip to the start.

>> LAB

Module 2 Basic Object Oriented Programming

Table of contents

2.1 Classes

By the end of this section you should be able to:

- Create a class in Scala
- Create a class with parameters
- Create an instance of a class (aka object)

Create a new class hello:

We can create a class "Hello" as follows

```
class Hello
```

```
defined class Hello
```

Create a new class hello:

We can create a class "Hello" as follows

```
class Hello
```

```
defined class Hello
```

We can create an instance of the class of type "Hello":

```
new Hello()
```

```
Hello@5acb5533
```

This output is the location of the instance of the class inside the JVM , Hello() is Sometimes called the class constructor.

Lets call a method that converts the instance to a string. We will go more in to methods later on:

```
(new Hello() ).toString()
```

```
Hello@36058b72
```

The body of a class

The body of the class is contained in two curly brackets as shown below:

class Hello{ Body of a Class }

Anything in the body will run when an instance of the class is created, For example, we can create the class hello, in the body we can put a statement in the class to print out "hello" :

```
class Hello {print("hello")}
```

```
defined class Hello
```

When we create a new instance of the class hello, the new object runs the statement and prints out "hello"

```
new Hello()
```

```
hello
Hello@19f2f5ab
```

Class Parameters

You can pass values into a class using parameters, parameters are placed inside the parentheses following the class name as shown here:

class Hello(Parameters){ Body of a Class }

You must specify the type of the parameters, for example we have the class "Hello", the parameter is a message of type string. We can use the parameter in the body of the class but the parameter is not visible outside the class

```
class Hello( message:String){

    println(message)

}
```

```
defined class Hello
```

We can use the parameter in the body of the class, for example we print out the value of "message". When we create an instance of the class we will not have access to the parameter outside the instance:

```
new Hello("what up")
```

```
what up
Hello@3610c582
```

We do not have access to the parameters:

```
(new Hello("what up")).message
```

```
Name: Compile Error
Message: <console>:27: error: value message is not a member of Hello
       (new Hello("what up")).message
                              ^
```

StackTrace:

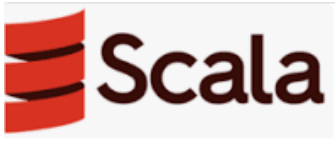
If a class has parameter values and we create an instance of that class without any parameter values we get an error:

```
new Hello()
```

```
Name: Compile Error
Message: <console>:27: error: not enough arguments for constructor Hello: (message: String)Hello.
Unspecified value parameter message.
       new Hello()
           ^
```

StackTrace:

class parameters are not accessible



2.2 Immutable and Mutable Fields

By the end of this section you should be able to:

- Describe the difference between mutable and immutable fields
- create fields in Scala classes
- describe the difference between class parameters and fields
- outline how to promote class parameters to fields

A Field is a value inside an instance of a class, it represents the state of the instance. Unlike parameters fields are inside the body of the class and accessible outside the instance of a class unless specified.

Immutable fields

Immutable fields cannot be changed, we use the "val" keyword to indicate the field is immutable. We can create a class hello, with the field "message" in the body:

```
class Hello1{val message="hello"}
```

```
defined class Hello1
```

We can create an instance of the object Hello1 and access the field "message " by using the dot notation :

```
(new Hello1).message
```

```
hello
```

Mutable fields can be changed, we use the "var" keyword to indicate the field is mutable. We can create a class hello, with the field "message" in the body:

```
class Hello2{var message="hello"}
```

```
defined class Hello2
```

We can create an instance of the object Hello2 and access the field "message " by using the dot notation :

We can change the field value:

```
hello2.message="good bye"
```

```
hello2.message: String = good bye
```

Now the field value has changed for the instance of the class:

```
hello2.message
```

```
good bye
```

If we tried something similar for the class Hello1 we would get an error because the field is a value and we cannot change a "val". Lets create an instance of class hello1,and assign it to the variable H1:

```
val H1=new Hello1()
H1.message
```

```
H1 = Hello1@5b014bae
hello
```

If we try and change the field we get an error

```
hello1.message="good bye"
```

```
Name: Compile Error
Message: <console>:27: error: not found: value hello1
val $ires8 = hello1.message
             ^
<console>:25: error: not found: value hello1
       hello1.message="good bye"
       ^

StackTrace:
```

Promoting class Parameters to become Fields

you can convert a parameter a field by adding the key word "val" to the parameter in the class constructor (in this case, we are adding the "val" inside the round parenthesis, not the curly parenthesis)

```
class Hello(val message: String)
```

```
defined class Hello
```

Let's create a new instance of the class with the parameter:

```
val hello=new Hello("Hello, world!")
```

```
hello = Hello@20aa3114
Hello@20aa3114
```

Because we have "val" key word we have access to the parameter :

```
hello.message
```

```
Hello, world!
```

Question 2.1:

The class person uses the persons birthday as a parameter, in the class body the variable year is the present year of 2017 and the age is the persons age , create an instants of class person and call it bob. Set the parameter to 1990. and show the field of age.

```
class person( BirthYear:Int){

    var year=2017
    var age=year-BirthYear
}
```

```
defined class person
```

```
val bob=new person(1990)
bob.age
```

```
bob = person@57aef3a6
27
```

Question 2.2:

Can you change the field year and what happens to age?

```
bob.year=1984
bob.age
```

```
bob.year: Int = 1984
27
```

2.3 METHODS

By the end of this section you should be able to:

- Implement methods
- Described evaluation order of methods vs fields
- Outline how index notation works

Methods do work on the instance of the class, they may or may not take parameters, and they may return values. We create a method using the "def" key word.

For example, we can create a method that returns the string "Hello":

```
def hello="Hello"
hello
```

```
hello: String
Hello
```

Next, we define the method echo, the method inputs a string and returns the output. In this case, we define the input and output type explicitly.

```
def echo(message: String):String=message
```

```
echo: (message: String)String
```

We can call the method by providing it an argument:

```
echo("Hey")
```

```
Hey
```

The method changes if we provide it a new argument:

```
echo("Hello")
```

```
Hello
```

You can change the value of a method after the object has been created.We can create the class "hey" with the method "SaySomthing":

```
class hey{def SaySomthing(Somthing:String):String=Somthing}
```

```
defined class hey
```

We can create a new object and call a method with an argument of 'Hello':

```
var Hi=new hey()
Hi.SaySomthing("Hello")
```

```
Hi = hey@29e449b6
Hello
```

If we call the method with different parameters we get a different result.

```
Hi.SaySomthing("Se Ya")
```

```
Se Ya
```

You can change the value of a method after the object has been created

---

Infix Notaion

If the method only takes one argument we can use Infix notation we can call a method with no dot or parentheses. For example we can split a string using the split method.

```
"Infix Notaion".split(" ")
```

```
Array(Infix, Notaion)
```

Equivalently using Infix notation

```
"Infix Notaion" split " "
```

```
Array(Infix, Notaion)
```

2.4 Default and Named Arguments

By the end of this section you should be able to:

- Utilize default argument values in Scala class constructors
- Leverage named arguments to only pass certain values

We can create classes with default parameters, for example, we can create a method name, which has two input parameters. The first and last, the method concatenates the value of the strings. We can set default values for the parameters to be empty strings .

```
def Name(first:String="",last:String=""):String=first+" "+last
```

```
Name: (first: String, last: String)String
```

We can call the method as follows:

```
Name("Rob","Roy")
```

```
Rob Roy
```

If we don't include all the parameters we don't get an error

```
Name("Johon")
```

```
"Johon "
```

The parameters are called in the same order as they are called in the class constructor. You can be explicitly and reference the name of the parameter in the method call.

```
Name(last="Barker",first="Bob")
```

```
Bob Barker
```

2.5 OBJECTS

Singlton object

By the end of this section you should be able to:##

- Understand singleton objects in Scala
- Describe the difference between a class and an object

In Scala we can create an object directly using the "object" key word. For example, we can create the object Hello as follows:

```
object Hello{def message ="Hello!"}
```

```
defined object Hello
warning: previously defined class Hello is not a companion to object Hello.
Companions must be defined together; you may wish to use :paste mode for this.
```

Unlike a class we do not need to create the object, we can call it directly:

```
Hello.message
```

```
Hello!
```

We can create a class Time,that converts minutes to hours :

```
class Time(FullHours :Int, PartialHours:Int){println(FullHours+"."+PartialHours+" Hours")}
```

```
defined class Time
```

We can then create an object Hello that creates instances of the class "Time" using the method "GetHours"

We can then create an object Hello that creates instances of the class "Time" using the method "GetHours"

```
object Hello{
    val OneHourInMinutes:Int=60

    def GetHours(minutes:Int)=new Time(minutes/OneHourInMinutes, minutes%OneHourInMinutes )

}
```

```
defined object Hello
warning: previously defined class Hello is not a companion to object Hello.
Companions must be defined together; you may wish to use :paste mode for this.
```

You can apply the method "GetHours" from the object to perform the conversion

```
Hello.GetHours(64)
```

```
1.4 Hours
Time@2de5d5b6
```

```
Hello.GetHours(50)
```

```
0.50 Hours
Time@11ed4776
```

Question 2.6:

Create the object "Hello1", the object is identical to "Hello" but call the rename the method "GetHours" to the method "apply"

```
object Hello1{
    val OneHourInMinutes:Int=60

    def apply(minutes:Int)=new Time(minutes/OneHourInMinutes, minutes%OneHourInMinutes )

}
```

```
defined object Hello
warning: previously defined class Hello is not a companion to object Hello.
Companions must be defined together; you may wish to use :paste mode for this.
```

Question 2.7:

Convert 64 minutes to hours using object "Hello1" and the method apply.

```
Hello1.apply(64)
```

```
1.4 Hours
Time@31b58384
```

Question 2.8:

What happens if you call the object Hello1(64)

```
Hello1(64)
```

```
1.4 Hours
Time@1f1e71f6
```

It turns out that apply is a special method, such that hello1.apply(64) equals hello1(64)

Lesson 3 - Case Objects and Classes

Learning objectives

3.1. Companion Objects
After completing this lesson, you should be able to:
● Create Singleton objects in Scala
● Describe the difference between a class and an object in Scala
● Outline usages for objects in Scala applications
● Start a Scala application

3.2. Case Classes and Case Objects
After completing this lesson, you should be able to:
● Describe when to use case classes and case objects instead of regular classes and objects
● Outline the differences between case classes and objects compared to regular classes and objects

3.3. Apply and Unapply
■ After completing this lesson, you should be able to:
● Illustrate the difference between a type and a term
● Describe how the apply method works in both objects and classes
● Outline how unapply works

3.4. Synthetic Methods
After completing this lesson, you should be able to:
● After completing this lesson, you should be able to:
● Describe how the Scala compiler generates functionality for you
● Explain what the synthetic equals(), hashCode(), toString() and copy() methods do

● Outline how you would use immutable case classes in a program where state is changing

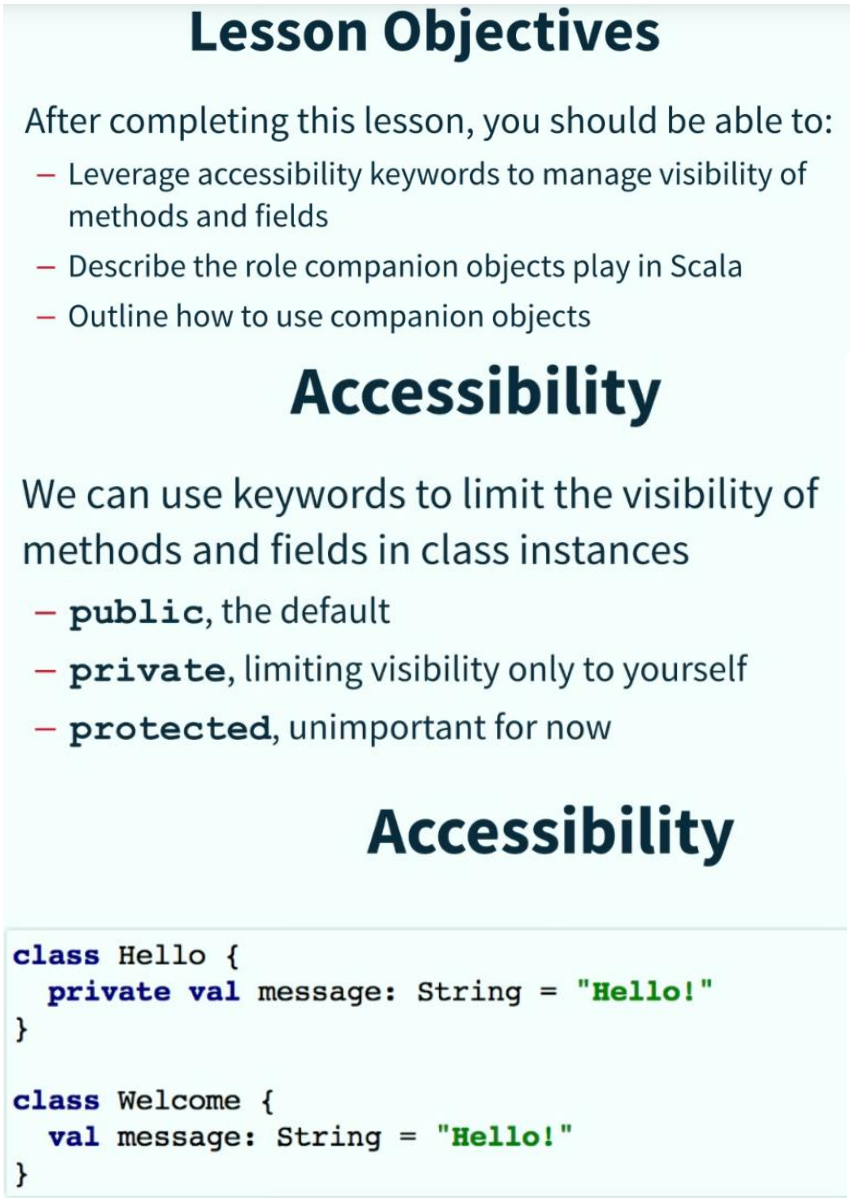
3.5. Immutability and Thread Safety

After completing this lesson, you should be able to:

● Understand basic thread safety in the JVM

● Describe the importance of immutability in multithreaded applications

● Outline how to use snapshots to preserve thread safety with case classes



Companion Objects

If a Singleton object and a class share the same name and are located in the same source file, they are called companions

A companion class can access private fields and methods inside of its companion object

Companion Objects

This is a great way to separate static members (fields, constants and methods) that are unrelated to a specific instance from those members that are related to a specific instance of that class

Companion Objects

```
object Hello {
  private val defaultMessage = "Hello!"
}

class Hello(message: String = Hello.defaultMessage) {
  println(message)
}
```

Lesson Summary

Having completing this lesson, you should be able to:

- Leverage accessibility keywords to manage visibility of methods and fields
- Describe the role companion objects play in Scala
- Outline how to use companion objects

f transcript. Skip to the end.
welcome to accessibility and companion objects after completing this lesson you should be able to leverage accessibility keywords in Scala to manage the visibility of methods and fields within classes and objects we want to describe the role companion objects planes Scala and outline how to use companion objects we can use keywords in Scala to limit the visibility of methods and fields what we call members inside of class instances there's the public key word which is the default you don't have to specify public school assumes the public is what you want from the get-go however if you want to make something hidden inside of a class you use the private keyword thus limiting visibility only to the current instance of the class there's also a protected keyword but

that is unimportant for now here is accessibility in action in the first class hello we've been immutable and private field called message with the value of hello if we were to instantiate an instance of hello and try to access the message field inside of it
the compiler will tell us before we deployed to production that this is an operation is not permitted because we use the private keyword we limited visibility of the message field only to the class hello itself we also have the class welcome which also has an immutable feel message inside of it which is a value of a string hello however this is not marked private so for create an instance of a welcome class and then try to access the message field inside of the welcome that is allowed in Scala we have a concept companion objects if a singleton object and a class share the same name and are located in the same source file they are called companions companion class can access private fields and methods inside of its companion object which is a special relationship because ordinarily when you mark something private you do not have visibility between two of these kinds of entities
companion objects are a great way to separate static members things such as fields constants in methods that are unrelated to a specific instance of a class from those members that are related to a specific instance of a class let's have a look at what I mean here is a companion object hello with a single constant private field it's called default message with the value of hello we also have a class and their defined side of the same source file to have the same name
class hello and objects below these are companions and share that special companion relationship inside of the constructor for a low we have a parameter called message of type string which is a default value of hello . default message we can leverage that parameter inside of the constructor of their low class in a print mine message it is only because the class alone in the object lol companions that the class how low can see a private the old inside of the object alone having completed this lesson you should be able to leverage accessibility keywords to manage visibility of methods and fields describe the role companion objects plane Scala and outline how to use companion objects
End of transcript. Skip to the start.

Scala COGNITIVE CLASS.ai

What is a Service Class?

Some classes represent work to be performed in an application

They know what to do, but they do not hold the data themselves

When an application calls the service, they pass the data to the service, and the service transforms it in some way

Examples:

- Persistence
- Loggers
- Calculation engines

In comparison, think about what a service class might be.

Case Classes

A representation of a data type that removes a lot of boilerplate code

- Generates JVM-specific convenience methods
- Makes every class parameter a field
- Is immutable by default
- Performs value-based equivalence by default

into an instance of a case class, a field.

Case Classes

```
scala> case class Time(hours: Int = 0, minutes: Int = 0)
defined class Time

scala> val time = Time(9, 0)
res0: Time = Time(9, 0)

scala> time == Time(9)
res1: Boolean = true

scala> time == Time(9, 30)
res2: Boolean = false

scala> time.hours
res3: Int = 9
```

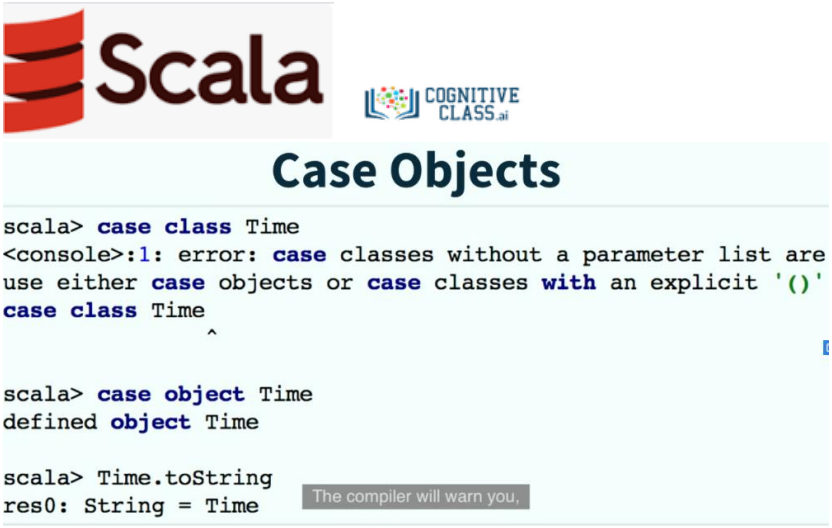
It has two parameters to it: hours and minutes.

Case Objects

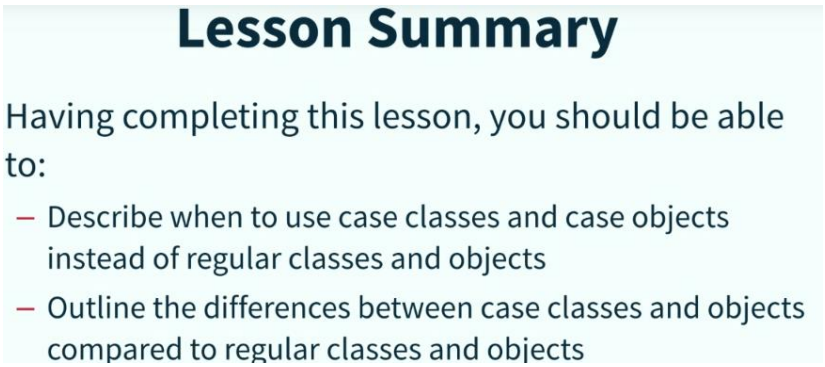
If a case class is an instance-based representation of a data type, a case object is a representation of a data type of which there can only be a single instance

If you try to create a case class with no parameters, it is stateless and should be a case object

If a case class is an instance-based representation



Scala da soporte a la noción de clases case (en inglés case classes, desde ahora clases Case). Las clases Case son clases regulares las cuales exportan sus parámetros constructores y a su vez proveen una descomposición recursiva de sí mismas a través de reconocimiento de patrones.



welcome to case classes in case objects after completing this lesson you should be able to describe when do you use case classes and caves objects instead of regular classes and objects should also be able to outline the differences between case classes and objects compared to regular classes and objects
so what is a day to class some classes represent specific data types in a domain that represents our application imagine an online store that store could have domain types such as customers accounts orders and inventory and more in comparison think about what a service class might be some classes represent work to be performed inside of an application but not the data themselves they know what to do but they have to be provided with the data on which they will do that work when an application called the service they passed the data to the service in the service of transforms and in some meaningful way examples of this would be persisting data to a datastore or logging out information about how well the application is running calculation engines are also built like this where you provide the data to be calculated and calculation returns and answer about a specific kind of calculation case classes are a representation of it data type that removes a lot of extra boilerplate code when you compile a case class the compiler generates JBM specific convenience methods for you and makes every class parameter that you pass into an instance of a case class a field these fields are immutable by default so you don't have to worry about thread safety and it also performs value-based equivalents by default
instead of comparing one instance of a case class to another to see if they are the same instance you're actually checking the values inside of them to make sure that this is value equivalent to another instance of a case class here is an example of a case class where we use the case keyword and the class keyword to create the instance of a case class called time
it has two parameters to it hours and minutes both the type int with default

values of 0 in order to create an instance of a time on the next line we say val time is equal to time 90 note that when we created this instance we didn't say the new keyword will explain why in a subsequent session you also notice that when it's prints out result 0 the time shows up with the representation of the values inside of time instead of the memory space where it was located in the JVM heap we can then do equivalents checks of this time versus another time in the next line we say for the time value that we created let's compare it to a new time instance where we created time without giving the minutes value and we see that the result of this operation is true we compare that original time to another time instance where the minutes value is 30 we get a value-based equivalents of false in the final line we access the field hours inside of time and it prints out the value 9 this is interesting because we never explicitly used the BAL keyword promote our hours or minutes parameters to be filled but every parameter is passed into a case class is automatically made a field if it keeps classes and instance based representation of a datatype the case
object is a representation of a datatype where there can only be a single instance if you try to create a case class with no parameters it is by definition stateless and should be a case object because there is no difference between one instance or another if there is no state inside of the class
here's an example of trying to create a case class time with no class parameters the compiler warnings saying that this is not possible instead if you created as a case object you now have an object of time of which there were only be one instance inside of the JVM
and you can reference it immediately without instantiated it by saying time .
to strength having completed this lesson you should be able to describe when to use case classes in case objects instead of regular classes and objects and outline the differences between case classes and objects compared to regular classes and objects
End of transcript. Skip to the start.

Las Clases Case

El primer uso importante es poder utilizar la clase en una estructura **match**. Para poder entenderlo al completo, veremos la forma en la que el compilador de **Scala** realiza el **pattern matching**, que en esencia se basa en la identificación de patrones para facilitar el trabajo de programación y la simplificación del código.

Empecemos por lo básico, cuando declaramos una clase con la palabra clave **case**, de la siguiente forma:

```
case class Persona(nombre: String, edad: Int)
```

El compilador genera el siguiente código:

```
class Persona(val nombre: String, val edad: Int)
  extends Product with Serializable
{
  def copy(nombre: String = this.nombre, edad: Int = this.edad):
Persona =
```

```
    new Persona(nombre, edad)

  def productArity: Int = 2

  def productElement(i: Int): Any = i match {
    case 0 => nombre
    case 1 => edad
    case _ => throw new IndexOutOfBoundsException(i.toString)
  }

  def productIterator: Iterator[Any] =
    scala.runtime.ScalaRunTime.typedProductIterator(this)

  def productPrefix: String = "Persona"

  def canEqual(obj: Any): Boolean = obj.isInstanceOf[Persona]

  override def hashCode(): Int =
scala.runtime.ScalaRunTime._hashCode(this)

  override def equals(obj: Any): Boolean = this.eq(obj) || obj match {
    case that: Persona => this.nombre == that.nombre && this.edad ==
that.edad
    case _ => false
  }

  override def toString: String =
    scala.runtime.ScalaRunTime._toString(this)
}
```

Además, genera un objeto acompañante con los siguientes métodos:

```
object Persona extends AbstractFunction2[String, Int, Persona] with
Serializable {
  def apply(nombre: String, edad: Int): Persona = new Persona(nombre,
edad)

  def unapply(p: Persona): Option[(String, Int)] =
    if(p == null) None else Some((p.nombre, p.edad))
}
```

Ciertamente, es mucho el código generado para una palabra tan corta como **case**, ¿verdad?.

Esta es una de las grandes ventajas de **Scala**. Es un lenguaje que está pensado para hacernos la vida muy fácil. Pero, ¿qué hace una clase **case** que no haga una clase normal? ¿Qué ganamos con todo este código generado?

Estas son las ventajas:

- Los objetos que provienen de clases **case**, se pueden comparar a nivel de valor. Si tenemos **p1** y **p2**, siendo de la clase Persona, **p1 == p2**, comparará si el nombre y la edad son idénticos, y devolverá **true** en el caso de que ambos

atributos (nombre y edad) devuelvan **true** a la hora de compararlos. Es como hacer **p1.nombre == p2.nombre && p1.edad == p2.edad**. Útil, ¿no es así?.

- Podemos copiar de una manera sencilla un objeto. Para ello, es tan fácil como invocar el método copy sobre un objeto de tipo **Persona**. Por ejemplo, haciendo **val p2 = p1.copy()** haremos una copia de **p1** en **p2**. También podemos copiar el objeto pero modificando un atributo. Haciendo **val p2 = p1.copy(nombre = "Pepe")**, conseguiremos una persona con los mismos atributos que **p1** pero de nombre **Pepe.**
- Se pueden crear objetos sin utilizar la palabra **new**. No es que sea el mejor truco del mundo, pero ahí está. Ahora para crear una Persona sólo tienes que hacer val persona = Persona("Pepe", 20).
- **VENTAJA NÚMERO 1:** A partir de ahora, podremos utilizar cualquier variable de tipo **Persona** dentro de una estructura **match**.

```
persona match {
    case Persona("Pepe", _) => println("Hemos encontrado a Pepe")
    case Persona(_, 40) => println("Alguien tiene 40 años")
    case _ => println("Ni es Pepe ni tiene 40")
}
```

Si observas con atención, utilizar **match** sobre un objeto definido por nosotros es muy ventajoso. En el primer caso, cualquier objeto que tenga como nombre "Pepe" hará match y ejecutará el código de la derecha de la flecha. En el segundo caso, buscamos un objeto cuyo nombre no nos importe pero cuya edad sea concretamente **40**.

NOTA para los Java Lovers

Si vienes de **Java**, te parecerá curioso no ver ningún **break**. Esto es así en **Scala**, no es necesario que cada línea se finalice con un **break**. Sólo se ejecutará la línea o bloque de la derecha de la flecha y una vez hecho, el compilador irá a la siguiente línea de después del **match**.

Esta breve introducción a la funcionalidad de las clases **case**, ya a simple vista, aporta a **Scala** una versatilidad que es difícil de encontrar en otros lenguajes, como por ejemplo **Java**. En el siguiente apartado, veremos formas de escribir nuestros bloques **match**.

Types versus Terms

A type is a description of a concept in an application

- A class is a type

A term is a concrete representation of a type

- Any class instance (including an object) is a term
- A method is a term, as it is also concrete and "callable"

Calling a Term

- Like some other languages, Scala allows you to "call" a term without specifying the method you want to call on it

```
scala> case class Time(hours: Int = 0, minutes: Int = 0)
defined class Time

scala> val time = Time(9, 0)
res0: Time = Time(9, 0)
```

without using the new keyword.

An Example of apply

```
object Reverse {
  def apply(s: String): String =
    s.reverse
}

scala> Reverse("Hello")
res0: String = olleH
```

Another Example of apply

```
scala> Array(1, 2, 3)
res0: Array[Int] = Array(1, 2, 3)

scala> res0(0)
res1: Int = 1
```

Unapply Deconstructs a Case Class

```
scala> case class Time(hours: Int = 0, minutes: Int = 0)
defined class Time

scala> val time = Time(9, 0)
time: Time = Time(9,0)

scala> Time.unapply(time)
res2: Option[(Int, Int)] = Some((9,0))
```

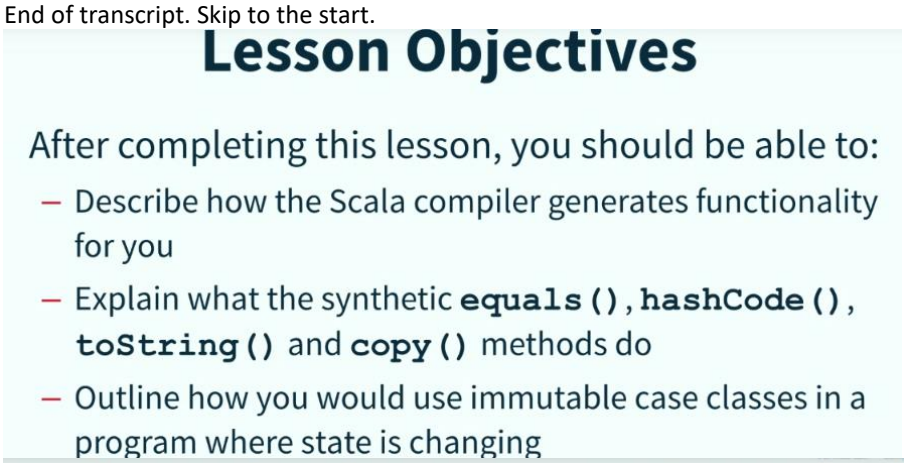
Lesson Summary

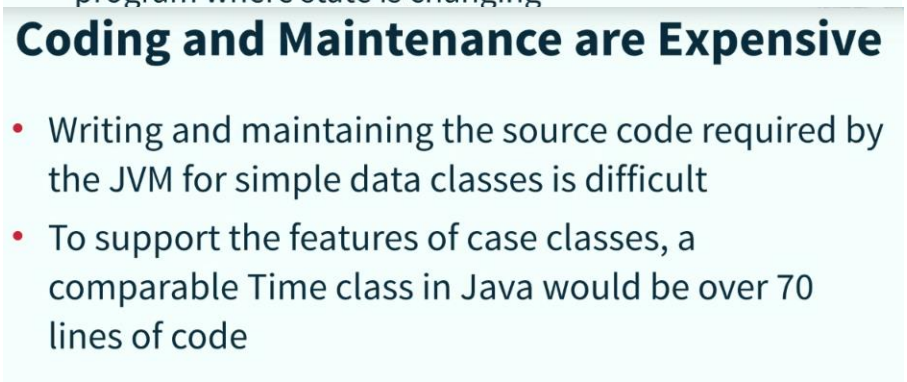
Having completing this lesson, you should be able to:

- Illustrate the difference between a type and a term
- Describe how the apply method works in both objects and classes
- Outline how unapply works

Skip to the end.
after completing this lesson you should be able to illustrate the difference between a type and eight-term describe how they apply method works in both objects and classes and outline how and apply works a type is a description of a concept in an application for example a class is a tight fit term is a concrete representation of a type any class instance including an object is itself eight-term a method is also a term because it is concrete and callable like some other languages Colin allows you to call it term without specifying the method you want to call on it so for example we noticed in the earlier session that we could create a case class time instants without using the new keyword in this example we have the definition of what a case class time is and then we create a field time without using the new keyword create the instance of time that it binds to the question is how did that work when you created your case class the compiler generated a companion object for the class for you and it also generated and apply method inside their companion object which is a factory for that time
instance when you call time and give it the parameters required to instantiate an instance of a time you were actually calling the companion object I'm and delegating to the apply method that was created inside of it so this is not calling the constructor of the time class directly when you see in this first line here that we are calling time with the nine the term that we're using is actually the companion object time and then delegating to the apply method with whatever parameters were provided that apply method then calls the constructor for time and creates a new instance of a time class which is held in res 0 you can do the exact same thing by explicitly calling the apply method as you see in the next line
apply is a very flexible concept in this case have created an object reverse and inside of it isn't apply method the takes a string reverses it returns that reversed representation of the strength if I call reverse hello I never explicitly state that I want to call the apply method however the term I'm using is the reverse object and the apply method is delegated to because I never explicitly called a method on that term is more clear example might be found

inside of collections in this case I'm going to create an instance of an array with three values inside of it one two and three this creates an array of integers but I never explicitly said new or said any method Honore that should tell you that array has a companion object inside of their companion object over a there's an apply method that took the values 123 and created an instance of an array for me but then in Scala to leverage the values inside of that array I don't use the typical square brackets to subscript the array value in this case I have the term results 0 and I passed to it
a number which is going to pull for me the value that exist at that subscript index so when I say Res 0
and give me the first value of us zero-based array it's going to give me the first value which in this case is 11 applied deconstruct a case class if I have a case class time as we've seen before and I create an instance of that time I can then call from the companion object I'm the on apply method passing in the instance of time I want to deconstruct and from that I get a representation of the values that are inside of the time in this case the hours and minutes
having completed this lesson
you should be able to tell the street the difference between type in the term describes how the apply method works in both objects and classes and outline how I apply works
End of transcript. Skip to the start.





equals()

This method is required by the JVM, but the default implementation only compares whether an instance of the class is the exact same instance

Scala provides value-based equivalence, allowing you to compare whether two different instances of a class have the same state

equals()

```
scala> case class Time(hours: Int = 0, minutes: Int = 0)
defined class Time

scala> val time = Time(9, 0)
time: Time = Time(9,0)

scala> time == Time(9)
res0: Boolean = true

scala> time == Time(9, 30)
res1: Boolean = false
```

hashCode()

- This method is required for any class that you might want to put into a hashed collection, such as a HashMap or HashSet

```
scala> case class Time(hours: Int = 0, minutes: Int = 0)
defined class Time

scala> Time(9, 45).hashCode()
res0: Int = 471971180
```

toString()

This method is required by the JVM, but the default implementation prints out a virtual representation of the instance location in memory

The synthetic `toString()` provided by Scala's case class shows you the values inside of the class

You can override this to make it even better

toString()

```
scala> class Time(hours: Int = 0, minutes: Int = 0)
defined class Time

scala> new Time()
res0: Time = Time@4d591d15

scala> case class Time(hours: Int = 0, minutes: Int = 0)
defined class Time

scala> Time()
res1: Time = Time(0,0)
```

As an example, consider

copy()

This method is not required by the JVM

The synthetic **copy()** provided by Scala's case class allows you to remain immutable and use "snapshots" of case classes when state needs to change

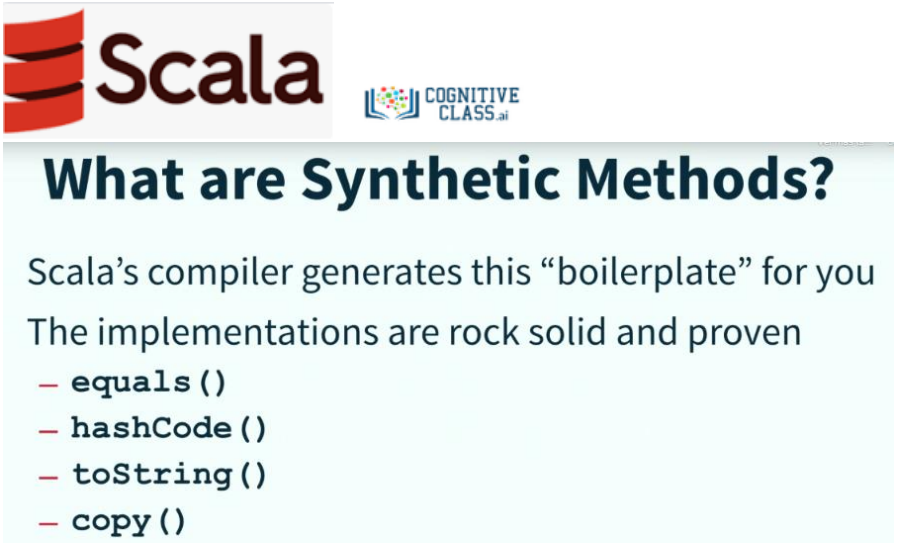
copy()

```
scala> case class Time(hours: Int = 0, minutes: Int = 0)
defined class Time

scala> var time = Time(9, 45)
time: Time = Time(9,45)

scala> time = time.copy(minutes = 50)
time: Time = Time(9,50)
```

Lesson Summary

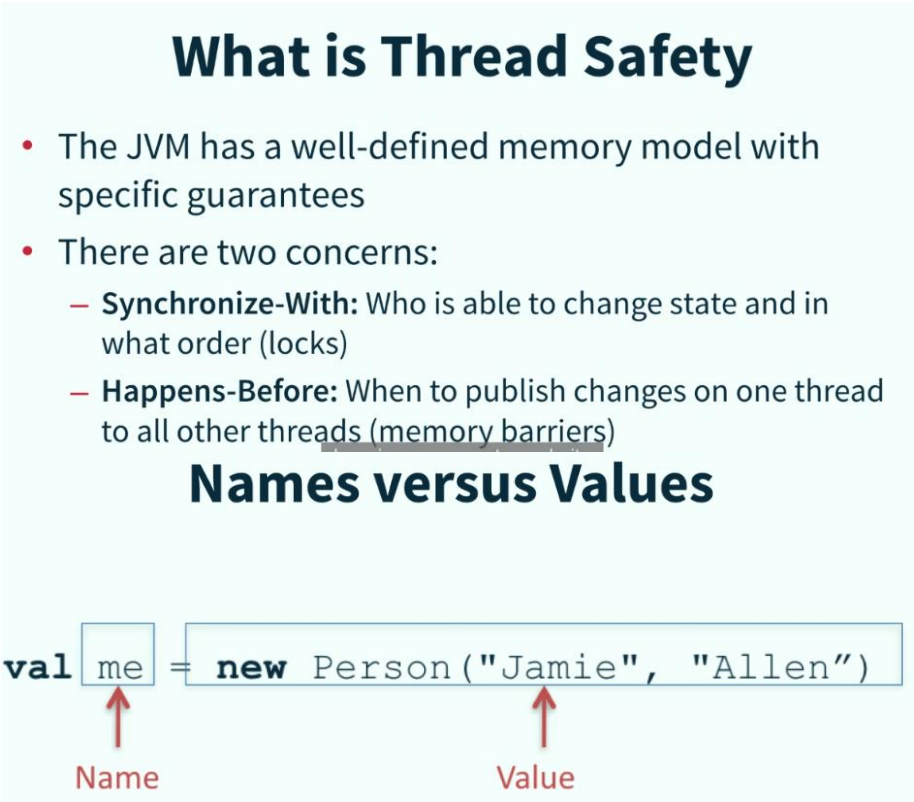
Having completing this lesson, you should be able to:

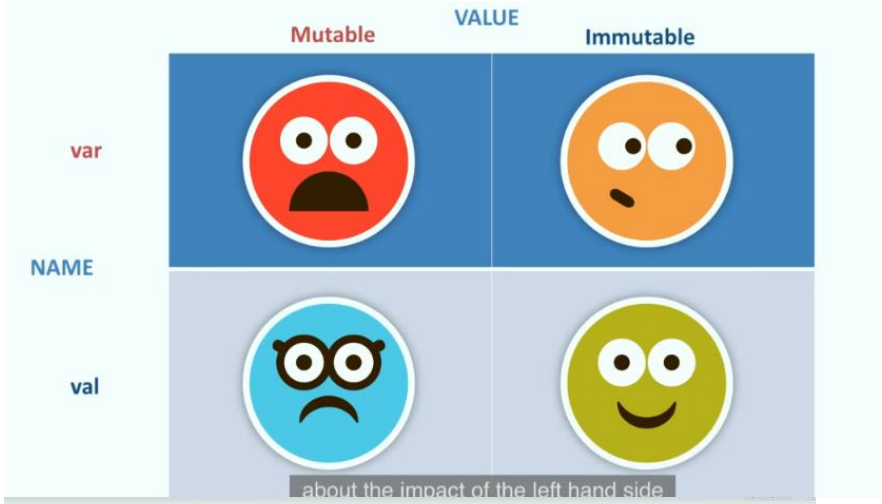
- Describe how the Scala compiler generates functionality for you
- Explain what the synthetic **equals()**, **hashCode()**, **toString()** and **copy()** methods do
- Outline how you would use immutable case classes in a program where state is changing



pt. Skip to the end.
welcome to synthetic methods after completing this lesson you should be
able to describe how the Scala compiler generates functionality for you explain
what the synthetic equals hashCode to strengthen copy methods do an outline
how you would use immutable case classes in a program where state is changing
coding a maintenance are expensive
you are a data scientist and you do not want to be wasting your time writing
code that is irrelevant to what you're trying to accomplish in finding meaning
in your data writing and maintaining the source code required by the JVM for
simple data classes is not easy and to support the features of case classes a
comfortable time class in Java to the one we've been creating in one line of
Scala would be over 70 lines of code so what are synthetic methods scholars
compiler generates the spoiler plate for you at compile time using
implementations at a rock solid improvement in the field there are four
methods provided to you equals hashCode to strengthen copy we also saw that the
Scala compiler created our companion object width apply method so there's a
lot of work that is doing for you so that you don't have to write and
maintain this code yourself equals method is required for any class inside
of the GBM but the default implementation only compares whether an
instance of the class is the exact same instances another it has nothing to do
with whether or not they contain the same values Scala provides value-based
equivalents allowing you to compare whether two different instances of a
class at the same state this is one of the reasons why scholar is a
data-centric language as an example imagine if I create a case class time
with hours and minutes and then I create an instance of that time I can then
compare that instance of time to other instances of sign in the third line
you'll notice that compared to another instance where I didn't give them in its
value and I get a value of true whenever I compare it to another time where the
minutes or thirty I get a false hashCode has met the required for any classes you
might want to put into a trash collection examples of this might be a
HashMap or ash set if we create a case class time as we've been doing all along
and then we create an instance of that class and we call hashCode on it you'll
see that there's a value that has been generated and displayed based upon the
values provided for hours and minutes this is to avoid collisions so that when
you place values into these hash based collections they can be retrieved
quickly the toString method is also required by the GBM for any classes
defined but the default implementation print out the virtual representation of
the instance location in memory which isn't very useful to anyone
the synthetic toString provided by scholars case class shows you the values
inside the case class and you can override this Steven make it prettier
you want as an example consider an ordinary class time to find exactly as
we've been doing case class but without the case keyword when I create a new
instance of that time with the default values for hours and minutes you'll see

the representation of time that comes back is the virtual memory space on the other hand if I use the case class time with the exact same implementation then whenever I create an instance I see the values inside of the hours and minutes instead finally we have the copy method this is not required by the JVM the synthetic copy provided by scholars case class allows you to remain immutable and you snapshots of kids classes when state needs to change it will create a new instance of whatever class you are defining with all the same values except for those that you change when you say to make the copy here's an example of our case class time again I'm going to create an instance of a time using the values 94 hours and 45 minutes you'll notice that the time that I'm finding it too is a bar it is mutable and therefore this time can point to a different instance of time if I wanted to
to however the time instants them creating of 945 is itself immutable and cannot be changed if I then use the copy method to get a new instance of a time I now am assigning a different instance of time to my time variable with different values based upon what I passed him when I called copy
having completed this lesson you should be able to describe how the Scala compiler generates functionality for you explain what the synthetic equals hashCode too stringent copy methods do an outline how you would use immutable case classes in a program where state is changing
End of transcript. Skip to the start.





The Right Side of the Equals Sign

- Represents the value of the current state
- This should always be immutable, meaning that the class instance contains only fields that are defined as **`val`**
- If not, you must protect the state and who can change it using Mutually Exclusive Locks

meaning that the class instance contains

Using a `var` for Snapshots

Allows us to keep the value on the right side of the equals immutable, but still change our current state by replacing what the **`var`** points to with another instance

The case class **`copy()`** method will help you do this

`@volatile`

The **`@volatile`** annotation must be used when you follow the snapshot strategy, to ensure that all threads see your updates

The case class **`copy()`** method will help you do this

anscript. Skip to the end.
welcome to immutability in thread safety after completing this lesson you should be able to understand basic thread safety in the GBM described the importance of immutability in multithreaded applications and online how to use snapshots to preserve thread safety with case classes the JPM has a well-defined memory model with specific guarantees imagine you are at a website and you are trying to order a book or something along those lines when you go to order the book there's a number of books that are available in multiple people might be trying to buy the book at the same time this means that somewhere in the application behind that website something has to manage the count of books available for this specific book that means multiple threads could be accessing discount at the same time the JVM has two concerns there's the idea of synchronize with which is who is able to change state and in what order imagine you trying to buy a book and another person trying to buy a book when there's only one left there's also the concept of happens before if I multiple threads running inside of a GBM and one thread changes about you how do other threads see the change that was made by the first thread when you're writing scholar there are two concepts you have to think of the left hand side of the equals and the right hand side of the equals on the left hand side we have a name that we are binding to the instance of the class in this case the name I give is me and is a pointer to the data inside of the value on the right hand side of the Eagles on the right hand side we have the instance of the class in this case the classes of person class and the instance has values inside of it for a name first name and last name is very important to think about the impact of the left hand side and the right hand side if you look at this table on the left side of the table we have the left hand side of the equal sign the name that were binding the pointer that were using to the instance of the class and whether that pointer can be chained changed if it can it's a bar if it can't it's a valve on the top of the stable we have the value the right hand side of the equals and we have beautiful state or immutable state if we have a bar pointing to mutable stayed inside of the class that we're creating the immutability both in the pointer and inside of the reference what's inside of that reference and that is the worst place that we can be if we move below and we have an immutable pointer to mutable data we still aren't in a really good place as far as the Red Sea he goes in both of these conditions we would have to worry about locking who has access what to do at a specific point in time if we look at the top of this table on the right hand side we have the condition where we have a bar as far as the name that were binding and we have immutable state in this case we have snapshots because if this state is changed on the right hand side of the equals and we are pointing to a new instance we have a bar which has to be updated to all threads and finally in the bottom right quadrant we have pure immutability where we have an immutable pointer and immutable data that is the simplest place for us to be from a thread safety perspective so the left

hand side of the equal sign represents appointed to the current value we want this to be final as much as possible using a bowel however we can reassign the instance were pointing to by using a bar the right hand side of the equal sign represents the data that we're currently using the state of the application should always be immutable meaning the class instance contains only fields that themselves are defined as Valse if not you must protect this state and who can change it by using mutually exclusive locks if we use bar for snapshots pointing to immutable data at all times then we can use the case class copy method to create new instances out our class that holds data so that we don't have to worry about
locks however when we use far we have to make sure that other thread see the change that we're making and in this case we have to use the volatile annotation as an example of this here's a case class customer with the person name and a last name with default values of empty strength if we had a bar pointer to an instance of their customer with the name Martin and order ski we would have to market at volatile and then if we use the customer not copy to change the last name of the instance of our customer and still have it point by Eddie customer pointer will then the ball at all and rotation will make sure that other thread see this change
having completed this lesson should be able to understand basic thread safety in the GBM described the importance of immutability in multithreaded applications and outline how to use snapshots to preserve thread safety with kids classes
End of transcript. Skip to the start.

Module 3 Case Objects and Classes

Table of content

3.1 Companion Objects

By the end of this section you should be able to:

- leverage accessibility keywords
- describe the role companion objects play in, Scala
- outline how to use companion objects

3.1 Companion Objects

By the end of this section you should be able to:

- leverage accessibility keywords
- describe the role companion objects play in. Scala
- outline how to use companion objects

Public and Private fields

We can use keywords to limit the visibility of class fields. Consider the class `Hello` and the class `Welcome`. The classes are identical, but the field `message` in the class `Hello` is set to private:

```
class  Hello {
    private val message: String = "Hello!"
}
```

```
defined class Hello
```

```
class Welcome{
    val message: String = "Hello!"
}
```

```
defined class Welcome
```

If you construct an instance of the the class `Hello`:

```
val hello = new Hello
```

```
hello = Hello@4c3ff574
Hello@4c3ff574
```

The field `message` is private. It is not possible to access it from the outside of class.

```
hello.message
```

```
Name: Compile Error
Message: <console>:28: error: value message in class Hello cannot be accessed in Hello
       hello.message
             ^

StackTrace:
```

When we create a new instance of class `Welcome`, we have access to the field

```
val welcome = new Welcome()
welcome.message
```

```
welcome = Welcome@440874f8
Hello!
```

Question 3.1:

Create a class `BankAccount`. The constuctor parameters should be: `name`, `balance` and `pinNumber`. They should be set as unmutable fields in the class constructor. For numerical values the default values should be zero and the pin number should be private.

Note: in general, using default numerical types such as `Int`, `Double` or `Float` is not a good practice. Use the `BigDecimal` included in Java8 or JavaMoney

Click here for the solution

```
class BankAccount(val name: String, val balance: Float = 0, private val pinNumber:Int = 0)
```

```
class BankAccount(val name:String,val balance:Float=0,private val pinNumber:Int=0)
```

Question 3.2:

Create two objects John and Sara of class "BankAccount" the fields should be set as follows:

`name` : John Smith

`balance` : 12500

`pinNumber` :1224

`name` : Sara Ski

`balance` : 0

`pinNumber` : 2222

```
val john= new BankAccount("john Smith",12500,1224)
val sara= new BankAccount("Sara Ski",0,2222)
```

```
john = BankAccount@fe910a3
sara = BankAccount@239ec503
BankAccount@239ec503
```

```
val john: BankAccount = new BankAccount(name = "John Smith", balance = 12500, pinNumber = 1224)

val sara: BankAccount = new BankAccount(name="Sara Ski", pinNumber=2222)
```

Question 3.3:

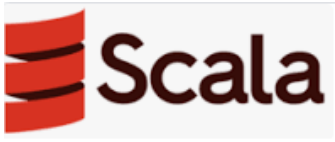
Acess the fields for the object John, what happens when you access the field "PinNumber".

```
john.name
```

```
john Smith
```

```
john.pinNumber
```

```
Name: Compile Error
Message: <console>:28: error: value pinNumber in class BankAccount cannot be accessed in BankAccount
       john.pinNumber
            ^

StackTrace:
```



Companion Objects

If a singleton object and a class share the same name and are located in the same source file, they are called companions. A companion can access both private and public fields and methods inside of its companion.

```
object Hello {
    private val defaultMessage: String = "Hi"
}

class Hello(val message: String = Hello.defaultMessage) {
    import Hello._ // import all fields and methods from the companion class into here

    println(defaultMessage + "!") // since everything was imported above, we can refer to `defaultMessage` directly
    println(message)
}
```

```
defined object Hello
defined class Hello
```

```
new Hello
```

```
Hi!
Hi
Hello@fc5a484
```

3.2 Case Classes and Case Objects

By the end of this section you should be able:

- describe when to use case classes
- be able to describe when to you use case classes and cases objects instead of regular classes

Case classes are a particular class that can be used to represent data. It is simpler to create a case class than a regular class. In a case class, the parameters are immutable fields by default. Furthermore, case classes have a number of other features that make them very useful in reprsenting data.

We can create the case class time with the parameters `hours` and `minutes`. We are assingning default values to the arguments here, but we dont have to.

```
case class Time(hours: Int = 0, minutes: Int = 0)
```

```
defined class Time
```

We can create a new instance of that class and assign it to a value:

```
val time1 = Time(9, 10)
```

```
time1 = Time(9,10)
```

We can create a new instance of that class and assign it to a value:

```
val time1 = Time(9, 10)
```

```
time1 = Time(9,10)
Time(9,10)
```

We can represent the instance or object "time1" with a table, the field names are in the right column, and the field values are located in the left column.

Class: Time	Object :time1
hours	9
minutes	10

We can view the actual fields

```
time1.hours
```

```
9
```

```
time1.minutes
```

```
10
```

One nice features of case classes is that how they are displayed when we print them. The string representation of a case class contains the name of the class and the values of the fields.

```
time1
```

```
Time(9,10)
```

We can create a second instance of `Time` called `time2`. We set the field values to be the same.

```
val time2 = Time(9, 10)
```

```
time2 = Time(9,10)
Time(9,10)
```

Similarly, we can visualize the object using a table

Class: Time	Object :time2
hours	9
minutes	10

We can view the actual fields

```
time2.hours
```

```
9
```

```
time2.minutes
```

We can also do equality checks to verify if two instances of the same class are the equal. When the fields have the same values, the result of the check is `true`.

```
time1 == time2
class Person(name: String)
case class Record(person: Person)
val recordComp = Record(new Person("petro")) == Record(new Person("petro"))

case class Records(records: Vector[Person])
val recordsComp = Records(Vector(new Person("petro"))) == Records(Vector(new Person("petro")))
```

```
defined class Person
defined class Record
recordComp = false
defined class Records
recordsComp = false
false
```

The procedure is summarized in the figure below, where each table is compared. As both fields are the same we get true :

Class: Time	Object :time1
hours	9
minutes	10

==

Class: Time	Object :time2
hours	9
minutes	10

hours	True
minutes	True

True

we can create a new instance without using the `new` keyword:

```
val time3 = Time(9, 20)
```

```
time3 = Time(9,20)
Time(9,20)
```

If we perform an equivalence check we get a false as the fields are different

```
time2 == time3
```

```
false
```

If we perform an equivalence check we get a false as the fields are different

```
time2 == time3
```

```
false
```

We can see that the field minutes in `time3` is set to `20`, hence the equivalence operation produces a `false`, as shown in the following figure:

Class: Time	Object :time1
hours	9
minutes	10

==

Class: Time	Object :time3
hours	9
minutes	20

hours	True
minutes	False

False

You cannot create a case class without parameters

```
case class Data
```

```
Name: Compile Error
Message: <console>:1: error: case classes without a parameter list are not allowed;
use either case objects or case classes with an explicit `()' as a parameter list.
case class Data
                ^

StackTrace:
```

but you can create a case object

```
case object Data
```

```
defined object Data
```

There is no need to instantiate it. A single instance of the object will be created at run-time.

```
Data
```

```
Data
```

Question 3.4:

Create a case class object `Birthday` that represents the birthdate of a person. With the integer fields in the following order: `day`, `month` (from 1 to 12) and `year`.

```
case class Birthdaty(day:Int,month:Int,year:Int)
```

```
defined class Birthdaty
```

Question 3.5:

Create an instance of the case class `Birthday` called `pat`, born on 11th of October (11 month) in `1991`.

```
val pot=Birthday(11,11,1991)
```

```
pot = Birthday(11,11,1991)
Birthday(11,11,1991)
```

Question 3.6:

Print out all the fields of the instance "pat" along with the instance itself.

```
println("day", pot.day)
println("mounth", pot.month)
println("year", pot.year)
println("instance of object:", pot)
```

```
(day,11)
(mounth,11)
(year,1991)
(instance of object:,Birthday(11,11,1991))
```

Question 3.7:

Create an instance of the case class `Bob`, set all the fields to the exact same value as `pat`, but change the value of `year` to `1992`. Finally, perform an equivalence check between `bob` and `pat`.

```
val bob=Birthday(11,11,1992)
pot==bob
```

```
bob = Birthday(11,11,1992)
false
```

3.4 Apply and Unapply

By the end of this section you should be able to:

- Illustrate the difference between a type and a term
- Describe how the apply method works in both objects and classes
- Outline how unapply works

Type

A type is a description of a concept in an application. For example, a class is a type.

Term

- A term is a concrete representation of a type
- Any class instance, including an object, is a term
- Methods are also a term

Apply

When you create a case class you are actually creating a companion object to create the instance of that class

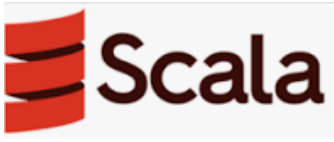
```
]: case class Time(hours: Int = 0, minutes:Int = 0)
```

```
defined class Time
```

we can verify this by typing the name of the class we get the object:

```
]: val t = Time
```

```
t = Time
]: Time
```



We can see an example of the apply method by creating the object `Reverse`. The method apply takes the string `s` and reverses it. Being able to use `apply` in this fashion allows us to create "smart constructors" that include validation and specialized construction logic.

```
object Reverse {
  def apply(s: String): String = s.reverse
}
```

```
defined object Reverse
```

We can call the apply method explicitly

```
Reverse.apply("hello")
```

```
olleh
```

or just using parentheses

```
Reverse("hello")
```

```
olleh
```

Something similar happens when you create an array (we will cover arrays in the next section). We can use the apply method or merely use the parentheses as shown in the following example:

```
val A=Array.apply(1,2,3)
println("A(0):",A(0))
println("A(1):",A(1))
println("A(2):",A(2))
```

```
(A(0):,1)
(A(1):,2)
(A(2):,3)
A = Array(1, 2, 3)
Array(1, 2, 3)
```

```
val B=Array(1,2,3)
println("B(0):",B(0))
println("B(1):",B(1))
println("B(2):",B(2))
```

```
(B(0):,1)
(B(1):,2)
(B(2):,3)
B = Array(1, 2, 3)
Array(1, 2, 3)
```

The following will be used for the next question. The class `OffTime` takes the first name of the employee, the employee's lunch time and break time as fields. Due to the limitations of the notebook environment, case clases and their companion objects need to be defined in the same cell.

```
class OffTime(val name: String, val lunchTime: String , val breakTime: String)

object OffTime {
  def apply(Name:String, LunchTime : String, BreakTime: String): OffTime =
      new OffTime(Name, LunchTime, BreakTime)
}
```

```
defined class OffTime
defined object OffTime
```

The following companion object creates an instance of the class

Question 3.8:

Use the companion object `OffTime` to create an instance of the class called 'employ1' and print the object:

```
name="John"
lunchTime="12:30 pm"
breakTime="3:15 pm"
```

```
val employ1=OffTime("John","12;30 pm","3:15 pm")
```

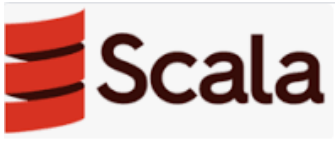
```
employ1 = OffTime@2af8c259
OffTime@2af8c259
```

Question 3.9:

Print out the fields of the instance:

```
println(employ1.name)
println(employ1.lunchTime)
println(employ1.breakTime)
```

```
John
12;30 pm
3:15 pm
```



Question 3.10:

Create a case class "OffTime1" with the identical fields "OffTime" and create an instance called "Employ2" with the exact same fields as 'Employ1'and print out the instance

```
case class OffTime1(name:String,lunchTime:String,breakTime:String)
val employ2=OffTime1("John ", "12:30 pm", "3:15 pm")
print(employ2)
```

```
OffTime1(John ,12:30 pm,3:15 pm)
defined class OffTime1
employ2 = OffTime1(John ,12:30 pm,3:15 pm)
OffTime1(John ,12:30 pm,3:15 pm)
```

Question 3.11:

Print out the fields of the instance "Employ2":

```
println("Name: " + employ2.name)
println("Luch Time: " + employ2.lunchTime)
println("Break Time: " + employ2.breakTime)
```

```
Name: John
Luch Time: 12:30 pm
Break Time: 3:15 pm
```

Unapply

If the `apply` method is used to construct object, `unapply` is used to destruct the instance into the parameters that created it. Consider the following example, we create the instance of `Time` using the two arguments: an integer 8 (the hours) and the integer 30 (the minutes).

```
val time1 = Time(8, 30)
```

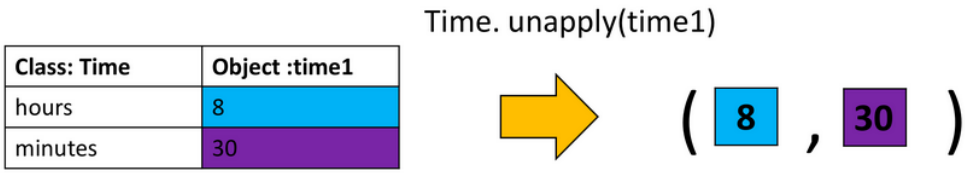
```
time1 = Time(8,30)
Time(8,30)
```

We can use the `unapply` method that is automatically generated for us by the computer. We can call it directly using the companion object.

```
Time.unapply(time1)
```

```
Some((8,30))
```

The procedure is summarized in the following figure, the instance or object time is represented by a table. Each of the different fields is represented with different rows in the table using colors to help distinguish them. After unapply is used, the field values are mapped to a tuple.



Synthetic Methods

By the end of this section you should be able:

- describe how the Scala compiler generates functionality
- explain the methods: equals, hashCode, toString and copy
- outline how you would use immutable case classes what state is changing

Synthetic methods are prebuilt methods that are included when you build a case class, similar to the apply method.

equals()

for `equals()` check out the previous example

The method `hashCode` maps instances with different fields to a number, consider the case class `Time`.

```
case class Time(hours: Int = 0, minutes: Int = 0)
```

```
defined class Time
```

We can create an instance of the case class with the field `hours` set to `8` and the field `minutes` set to `30`.

```
val time = Time(8, 30)
time
```

```
time = Time(8,30)
```

When we create the instance of the case class the compiler generates the method `.hashCode`. We can use the method to generate a value that we can use later in a `HashMap` or `HashSet`.

```
time.hashCode
```

```
448866544
```

The method maps the instance to the number `448866544`

If we create a different instance and apply the method `hashCode` we get a different value.

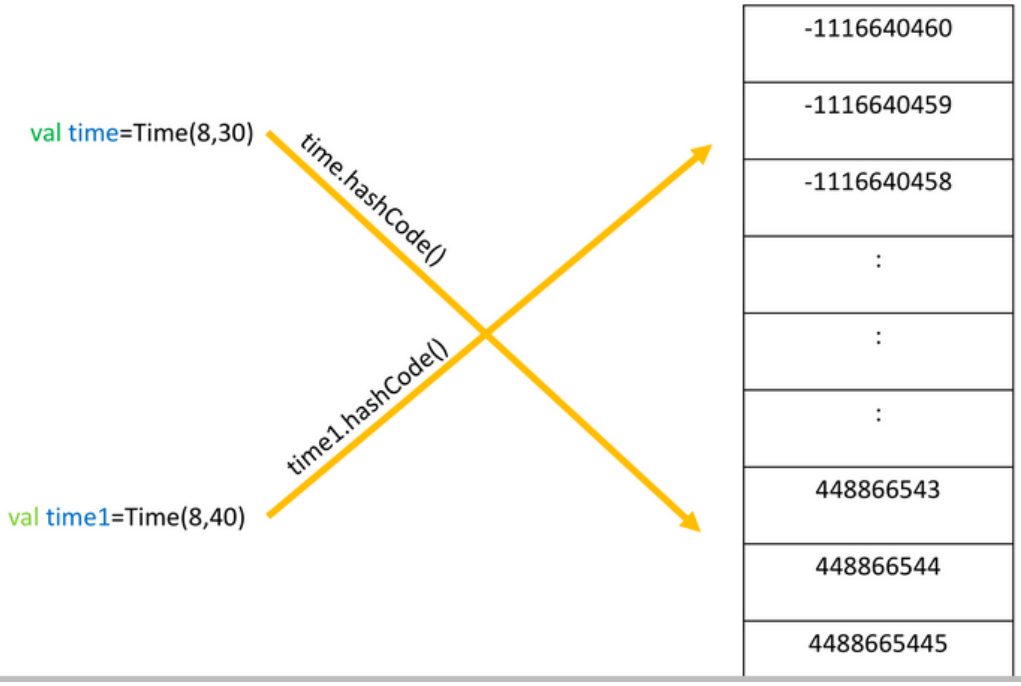
```
Time(8, 40).hashCode == Time(8, 50).hashCode
```

```
false
```

```
val time1 = Time(8, 40)
time1.hashCode
```

```
time1 = Time(8,40)
-1116640459
```

The following image summarizes the result, after applying the method `hashCode` to each instance we get a different value.



If we create two instances with the same fields, then apply the method ".hashCode()" the same value is generated:

```
val time3 = time.copy(8, 40)
time3.hashCode
```

```
time3 = Time(8,40)
-1116640459
```

```
val time4 = Time(8, 40)
time1.hashCode
```

```
time4 = Time(8,40)
-1116640459
```

The result is summarised in the following figure:

toString()

The Scala compiler will automatically generate a `toString` method for every case class we define. Consider the class Dog with the fieldd `name` and `breed`. When we create a new instance of the class (note that this is not a case class) and print out the name we see the virtual memory space.

```
class Dog(name:String, breed: String)
val dog1 = new Dog("Max", "German Shepard")
println(dog1)
```

```
$line123.$read$$iw$$iw$Dog@5e9459a0
defined class Dog
dog1 = Dog@5e9459a0
Dog@5e9459a0
```

When you create a case class the method `toString` is generated. When you print out an instance of the case class the name of the case class and the fields are displayed. Consider the case class `K9` with the field `name` and `breed`. When you print the instance of the object you get the name of the class as well as the values of the fields.

```
case class K9(name:String,Breed:String)
val dog2 = K9("Max","German Shepard")
print(dog2)
```

```
K9(Max,German Shepard)
defined class K9
dog2 = K9(Max,German Shepard)
K9(Max,German Shepard)
```

copy()

The method copy allows you to copy an instance of a case class; it also gives you the ability to change only a few fields while making a copy. Consider the instance of the case class `K9` called `dog1`:

```
val dog1 = K9("Max","German Shepard")
dog1
```

```
dog1 = K9(Max,German Shepard)
K9(Max,German Shepard)
```

We can create a new instance and call it dog2:

```
val dog2 = dog1.copy()
dog2
```

```
dog2 = K9(Max,German Shepard)
K9(Max,German Shepard)
```

we can check if the two instances are equal

```
dog1 == dog2
```

```
true
```

we can copy the instance to dog3 and change the field name to Bob

```
val dog3 = dog1.copy(name = "Bob")
dog3
```

```
dog3 = K9(Bob,German Shepard)
K9(Bob,German Shepard)
```

COGNITIVE CLASS.ai

Immutability in thread safety

By the end of this section you should be able to:

- understand basic thread safety in the JVM
- described the importance of immutability in multithreaded applications
- online how to use snapshots to preserve thread safety with case classes

consider the case class customer

```
case class Customer(fistName:String, lastName:String)
```

```
defined class Customer
```

we can create a new instance of the class and assign it to the "val" person1:

```
val person1 = new Customer("Joie", "JoJo")
person1
```

```
person1 = Customer(Joie,JoJo)
Customer(Joie,JoJo)
```

we can not assign the value to a new instance of customer as val's are immutable

```
person1 = new Customer("Joie","JoJo")
```

```
Name: Compile Error
Message: <console>:29: error: reassignment to val
       person1 = new Customer("Joie","JoJo")
                 ^

StackTrace:
```

we can access the fields

we can access the fields

```
person1.fistName
```

```
Joie
```

but as the are val's we can not change them

```
person1.fistName = "john"
```

```
Name: Compile Error
Message: <console>:27: error: reassignment to val
       person1.fistName = "john"
                 ^

StackTrace:
```

If we assign an instance of a case class to a var we must be more careful. To ensure that any threads know about any changes we use the "@volatile" annotation:

```
@volatile var person2 = Customer("Martin","Odersky")
person2
```

```
person2 = Customer(Martin,Odersky)
Customer(Martin,Odersky)
```

Therefore, if we use the method `copy` and change the field `lastname` other threads will see this change.

```
person2 = person2.copy(lastName="Doe")
person2
```

```
person2 = Customer(Martin,Doe)
Customer(Martin,Doe)
```

Lesson 4 - Collections

Learning objectives

4.1. Collections overview
After completing this lesson, you should be able to:
● Understand the structure of the Scala Collections hierarchy
● Describe how to apply functions to data in collections
● Outline the basics of structural sharing
● Illustrate the performance characteristics of different data structures

4.2. Sequences and Sets
After completing this lesson, you should be able to:
● Describe the various kinds of sequence collections and their properties
● Outline the desirable properties of the Vector collection type
● Describe the properties of set collections

4.3. Options
After completing this lesson, you should be able to:
● Describe the relevance of Option in the Scala type system
● Outline how to use Option in your types

4.4. Tuples and Maps
After completing this lesson, you should be able to:
● Describe what a tuple is and how they are used
● Outline how to deconstruct tuples
● Describe the properties of a map

4.5. Higher Order Functions
After completing this lesson, you should be able to:
● Describe the application of functions to data
● Outline basic usages of higher order functions in Scala

Scala COGNITIVE CLASS.ai

Scala Collections

Higher Order Functions

Scala is a functional programming language

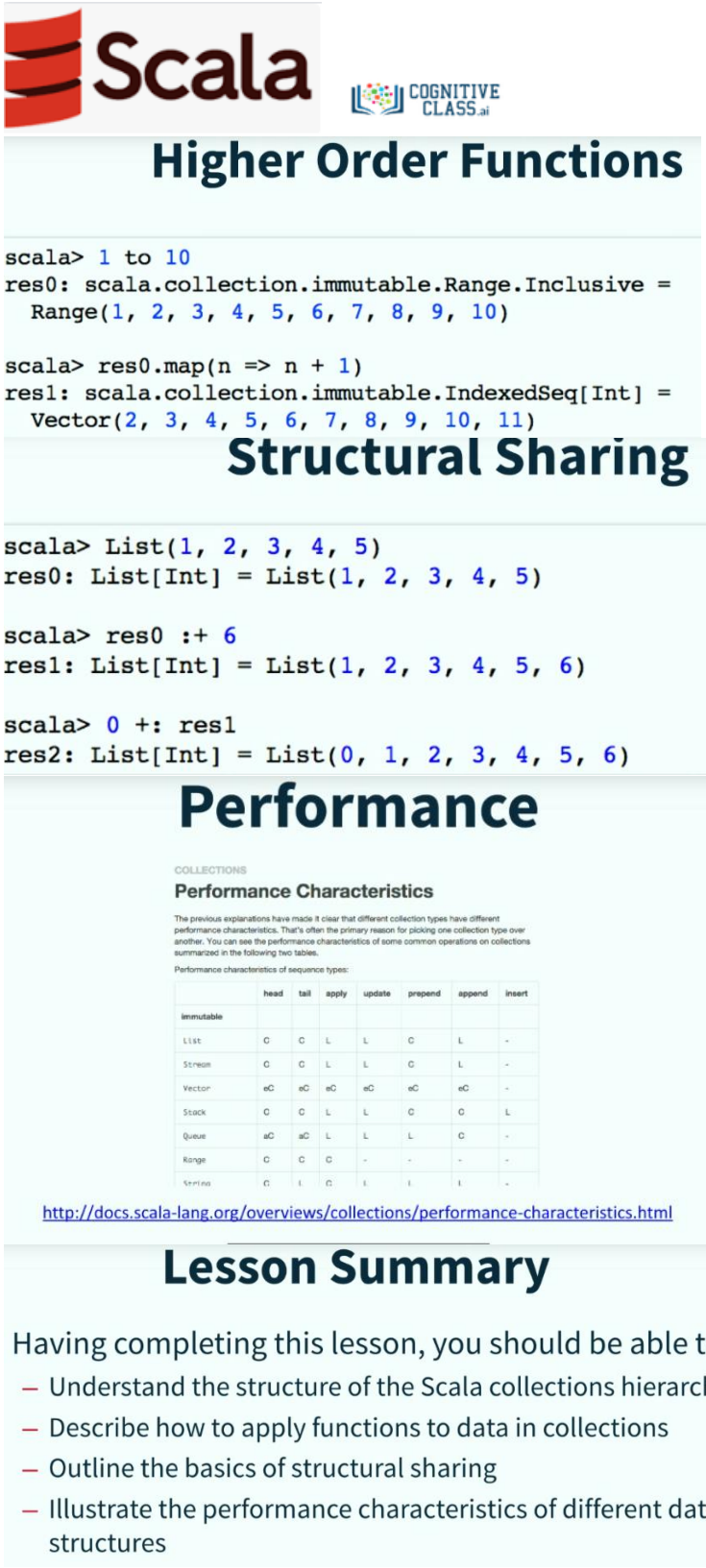
We apply functions to data inside of containers

There are many higher order functions available to you across the collections library

Structural Sharing

When you create a collection, it is an aggregation of references to individual values in the Java heap

If you use immutable collections of immutable values, those references can be shared between collection instances



rt of transcript. Skip to the end.
welcome to the collections overview after completing this lesson we should be able to understand the structure of the Scala collections hierarchy describe how to apply functions to data in collections outlined the basics of structural sharing and illustrate the performance characteristics of data in different data structures the Scala collections hierarchy represents a structure of collections that can contain data in different ways at the top is the idea of something that can be traversed or walked next in the hierarchy is a specialization of that behavior that represents iteration the idea that could be walk to single time that you could say is there a next value

is there a next value is there a next value and then beyond adorable we have specializations of a sequence a set and a map the sequence breaks down a little bit further you have the idea of a linear sequential which is something that has to be walked one by one to find the elements you want such as a linked list or an indexed sequential which gives you the ability to directly access about you inside of the sequence scholars collections have higher order functions because colin is a functional programming language instead of walking a collection and performing a transformation and putting some data into a new collection we instead apply a function to the values inside of a container and a collection is merely one form of such a container there are many higher-order functions available to you across the Collections library which we will discuss in a later session
here's an example of higher-order functions just saw the you have an idea of what I'm talking about if we open the rubble and we create a collection of numbers from 1 to 10 we can then map over this data applying a function where we say for every number which we're just calling
and we're going to add one to it and the result of that is a new collection with the values inside of it from two to 11 when you have data inside of a immutable collection in Scala it is an aggregation of references to individual values in the Java heap so for example on the earlier slide when we were looking at data of one to 10 inside of a list we had references to individual integer instances of the integer class they could be scattered around the heap and we use immutable collections of immutable values those references can be shared between collection instances here is an example of structural sharing we have a list of members one two three four and five each one of those numbers inside the list is not actually the number itself but a reference to a value on the EP that is one it just looks like the value inside of the list but really the list is calling on two references that point to those numbers if we depend 62 the end of this list we then have a new list with one two three four five and six
the only thing that's really been added is a new collection of references pointing to the one pointing to the same 2012 same three with a new six and then if we take that res 1 and we prepend 0 in front of it again
reuse the one the 23 the poor the five and six from res 12 create a new list of references within new reference 20 and don't worry about these prepend a nap and operators that you see inside of this I'll be explaining them in a later session scholars collections have well-defined performance characteristics and if you go to this web page is shown or if you do a web search for Scala collections performance characteristics you'll find a page that has every collection in the Collections library and its performance using barriers operations such as
accessing the head of a list which is the first value the tale of the list which is everything after the first value or updating the list or pre pending the list these operations have different performance characteristics that are based on whether their constant time or they are linear such as I have to walk to the end of the list to perform this work or effectively
constant time or amortize constant I'm having completed this lesson we should be able to understand the structure of Scala collections hierarchy describe how to apply functions to data in collections and outline the basics of structural sharing you should also be able to illustrate the performance characteristics of different data structures
End of transcript. Skip to the start.

What is a Sequence?

An ordered collection of data

Duplicates are permitted

May or may not be indexed

Array, List, Vector

The apply method on an instance is a lookup

Array

A fixed size, ordered sequence of data

Very fast on the JVM

Values are contiguous in memory

Indexed by position

Array

```
scala> Array(1, 2, 3, 4, 5)
res0: Array[Int] = Array(1, 2, 3, 4, 5)

scala> res0(2)
res1: Int = 3

scala> res0(5) = 6
java.lang.ArrayIndexOutOfBoundsException: 5
  ... 33 elided
```

List

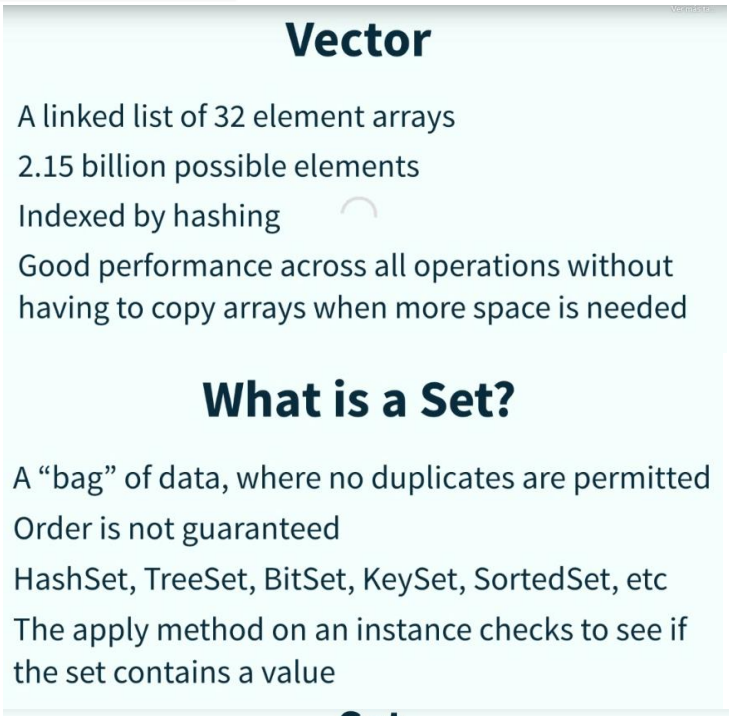
A linked list implementation, with a value and a pointer to the next element

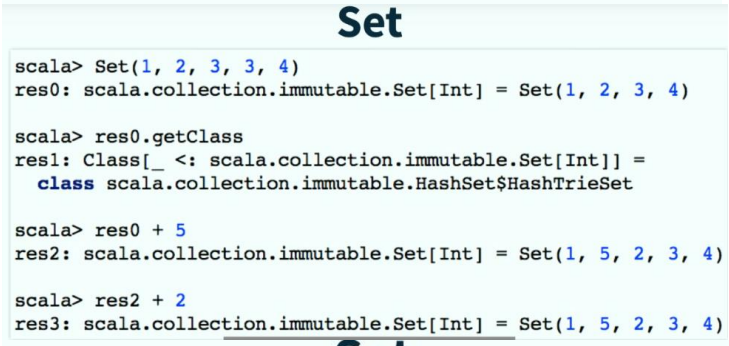
Theoretically unbounded in size

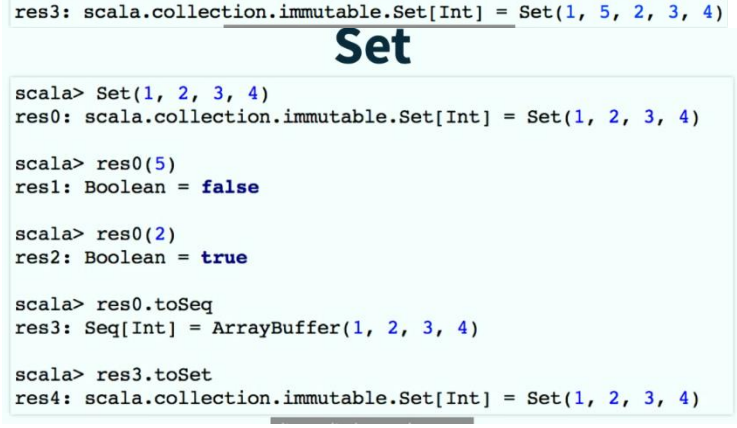
Poor performance as data could be located anywhere in memory, and must be accessed via "pointer chasing"

List

```
scala> List(1, 2, 3, 3, 4)
res0: List[Int] = List(1, 2, 3, 3, 4)

scala> res0.distinct
res1: List[Int] = List(1, 2, 3, 4)

scala> res0
res2: List[Int] = List(1, 2, 3, 3, 4)

scala> res0 :+ 5
res3: List[Int] = List(1, 2, 3, 3, 4, 5)

scala> 0 +: res1
res4: List[Int] = List(0, 1, 2, 3, 4)
```



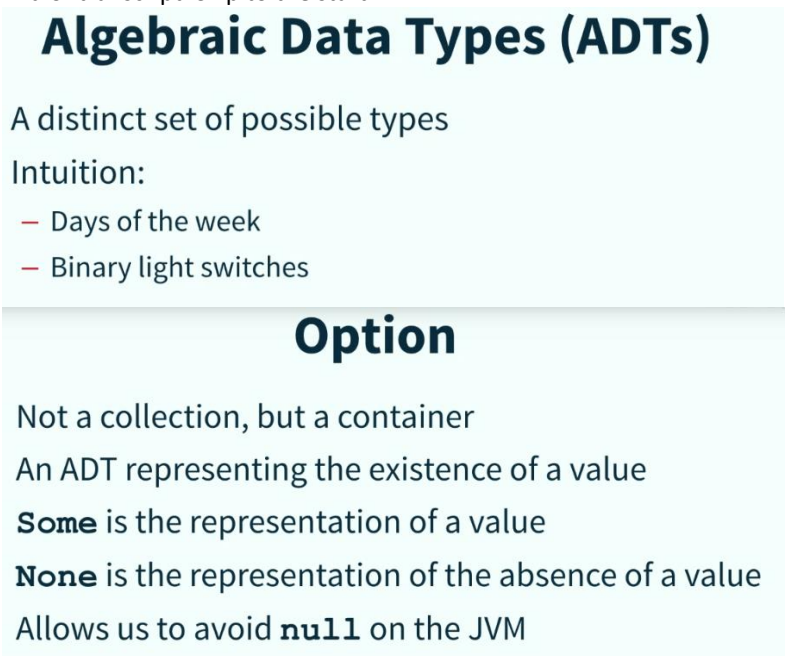




f transcript. Skip to the end.
welcome to sequences and sets after completing this lesson we should be able to describe the various kinds of sequence collections and their properties outline the desirable properties of the vector collection type and describe the properties of set collections so what is the sequence sequence is an ordered collection of data where duplicates are permitted there may or may not be index such that you can access a value directly into it without having to walk the entire collection and some of the common types you find in sequences are raised lists and vectors the apply method when you have an instance of a sequence is a lock up at that index so consider an array of arrays are fixed in size and they are an ordered sequence of data they are very

fast on the JVM and on other platforms because they are always contiguous and memory and data can be prefetch down to the course of execution at the time instructions are going to be executed a razor also indexed by their position as an example of a scholar a we create an array using the companion object apply method and we put the values of one two three four and five inside of it this gives us an array of events because the razor zero-based when you access the third value inside of the array you actually look it up with a too because the first element would be zero the second will be one and that there would be two however if we try to access something beyond the size of the array such as going with the result 0 and indexing it at the fifth element which is actually the sixth position we are not able to access a value or try to set the value because we are outside of the arrays size here is how we would operate on a raise
we use the companion object apply method to create a new instance of an array but saying array with the values of one two three four and five arrays are zero-based which means that the first element is access 10 the second element with one the third element with two etcetera
so when we say result 0 sub 2 we are axing about you three if we tried to update this array and all arrays in scholar actually mutable collections if we try to update the element at the fifth position which is actually six because this is a zero-based index if we try to change that value we get an array index out of bounds exception because the array only had five elements and the last one would have been indexed at four scholars lists are a linked list implementation with a value in a pointer in every cell the pointer points to the next element or next cell with another value if there is one they are theoretically unbounded and size though that isn't entirely true because there is always the limitation of the amount of memory that the GBM is allocated or on the machine there is some performance hit in using lists because data could be located anywhere in memory every time you go from cell to cell it could be going anywhere else in memory and that means you're accessing data using pointer chasing which can lead to stalls at the core of execution however compared to arrays they're very flexible because you don't have to worry about if I exceed the size of a list I only have to worry about how much memory I have in this case again I use the companion object apply method to create an instance of a list and I put the values 123 another instance of three and four into the list because sequences can have duplicate values this is not a problem I can get rid of the extra three by using the distinct method on the instance of the list that was created however if I look at the original result 0 you'll notice it was not change when I did that distinct operation instead a new list was created using the exact same references to the values that we were
you
before if we want to add value to a list we can either up and or prepend and their operators that allow us to do that however because lists are sequences where order matters we have to use an associative operator in this case you'll notice that we have res 0 and we want to
upended value 5 to the end of it we have to have a colon + operator with a colon faces toward the sequence if we want to prepend about you in front of the sequence we have to use the + colon again with the colon facing toward the sequence so we say zero plus colon Res 1 resulting in a value of a list with zero one two three and four factors are unique kinds of sequences in that they represent a combination of both arrays and lists vectors are comprised of 32 elementa raised and as you add more elements than a single array can hold it actually creates a structure
raised that point to one another their 2.15 billion possible elements and every operation you perform on a vector is effectively constant time factors are

indexed by hashing and they give you good performance characteristics across all operations what is the scent asset is a bag of data were no duplicates are permitted no order is guaranteed inside of the collection and common usages of sets include hash sets three sets bit sets key sets and sorted sets there are more as well
the apply method on asset instead of giving you evaluate a specific place inside of a sequence instead tells you whether or not the value exists in the set because there is no order to asset per se
here's an example of a set I create the set using the companion object apply method and I put the values 1233 again
for intuit this results in a set of one two three and four because we already had 13 inside of it we didn't need to have a second one sets are unique set of values you can see that the default implementation of asset that we get whenever we create one news in the companion object apply is a hash set and because there's no Assoc tivity order we want to add a value to a said we don't have to reference its position weathers pre pending or up ending we are merely throwing about you into a bag so we say result zero plus five and it puts 5 into the collection of set however if we then went to add about you that already exists it results in no change here we have an instance of a set one two three and four we can check to see whether a value is inside of that set by using the apply method on the instance it was created result 0 it we see a result 075 its legacy of fives inside of that said and we get false and returned it will check to see if Tues inside of it we get true if we take the result 0 and want to make it an ordered sequence of values we can use the to seek method and then we can take the resulting sequence and converted back to us said by calling to set on that instance having completed this lesson you should be able to describe the various kinds of sequence collections and their properties outline the desirable properties of the vector collection type and describe the properties of set collections
End of transcript. Skip to the start.



Option

```
scala> Option("Jamie")
res1: Option[String] = Some(Jamie)

scala> res1.get
res2: String = Jamie

scala> res1.getOrElse("Jane")
res3: String = Jamie
```

Option

```
scala> Option(null)
res0: Option[Null] = None

scala> res0.get
java.util.NoSuchElementException: None.get
  at scala.None$.get(Option.scala:347)
  at scala.None$.get(Option.scala:345)
  ... 33 elided

scala> res0.getOrElse("Foo")
res2: String = Foo
```

This results in a None.

Option

Option allows us to create APIs where the possible absence of value is encoded in the type system

We can then perform behavior without asking whether or not the value is **`null`** in advance

Option

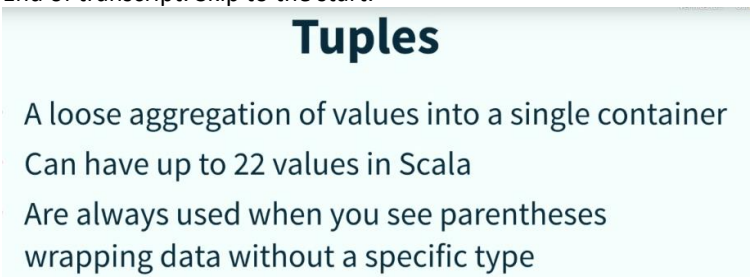
```
scala> case class Customer(
                    first: String = "",
                    middle: Option[String] = None,
                    last: String = "")
defined class Customer

scala> Customer("Martin", last = "Odersky")
res0: Customer = Customer(Martin,None,Odersky)
```

Lesson Summary

Having completing this lesson, you should be able to:

- Describe the relevance of Option in the Scala type system
- Outline how to use Option in your types

to the end.
welcome to option after completing this lesson we should be able to describe the relevance of option in the Scala type system outline how to use option inside of your types first let's describe what an algebraic data type is is a distinct set a possible types that composes all of the possibilities in the entire world represents think an algebraic data type that represents the days of the week it could only be one of seven values Monday through Sunday another algebraic datatype might be a light switch where you could only be on and off option is itself an algebraic data type that represents the existence of a value is not itself a collection but more of a data container however his relevance to things we are going to discuss in further sessions the value some is the representation of the existence of a value whereas none is the representation of the absence of value this allows us to avoid the concept of no inside of the GBM consider the creation of an option of a string I do that using the companion object apply method on the option parent type and I passing a string which results in a sum of a string Jamie in the case where I do I get that value I received the value inside of the wrapping some however I could also use a get or else which is a safer way to performing operations side of an option in case the value does not exist however the default value chain is not applied because there is a value inside of this result consider if I created an option of a null value this results in a non and if I try to do a get on the resulting instance of the option I'm going to get a no such element exception from the JVM this is where we want to use the get or else instead is considered bad form in Scala to use the option dot get by using get or else in this case I'm providing the default value of a strength Pooh and says there is no value inside of the option I get a food whenever I do together else option allows us to create API's where the possible absence of value is encoded in the type system and we can then perform behavior without asking whether or not the value is known in advance
here's an example of a case class we saw this example before where a customer may not have a middle name our first name because it is not an option is implicitly a required value the last name is the same thing however the middle is an option of a string where the default value is none
whenever I create an instance of a customer and I only give her first name and last name the middle name is a nun
having completed this lesson you should be able to describe the relevance of option in the Scala type system and outline how to use option in your types
End of transcript. Skip to the start.



COGNITIVE CLASS.ai

Tuples

```
scala> Tuple2(1, "a")
res0: (Int, String) = (1,a)

scala> Tuple2(1, 2)
res1: (Int, Int) = (1,2)
```

```
scala> (1, "a")
res0: (Int, String) = (1,a)
```

Tuples

Can be accessed using a 1-based accessor for each value

Can be deconstructed into names bound to each value in a tuple

Tuples

```
scala> val tuple = (1, "a", 2, "b")
tuple: (Int, String, Int, String) = (1,a,2,b)

scala> tuple._3
res0: Int = 2

scala> val (first, second, third, fourth) = tuple
first: Int = 1
second: String = a
third: Int = 2
fourth: String = b
```

I can then deconstruct that tuple

Tuple2

```
scala> (1, "a")
res0: (Int, String) = (1,a)

scala> (2 -> "b")
res1: (Int, String) = (2,b)

scala> (3 -> "c" -> 4)
res2: ((Int, String), Int) = ((3,c),4)
```

Unapply Deconstructs a Case Class

```
scala> case class Time(hours: Int = 0, minutes: Int = 0)
defined class Time

scala> val time = Time(9, 0)
time: Time = Time(9,0)

scala> Time.unapply(time)
res2: Option[(Int, Int)] = Some((9,0))
```

Maps

A grouping of data by key to value, which are tuple "entries"

Allows you to index values by a specific key for fast access

Common implementations: HashMap, TreeMap

Maps

```
scala> 1 to 5
res0: scala.collection.immutable.Range.Inclusive =
  Range(1, 2, 3, 4, 5)

scala> 'a' to 'g'
res1: scala.collection.immutable.NumericRange.Inclusive[Char]
  = NumericRange(a, b, c, d, e, f, g)
```

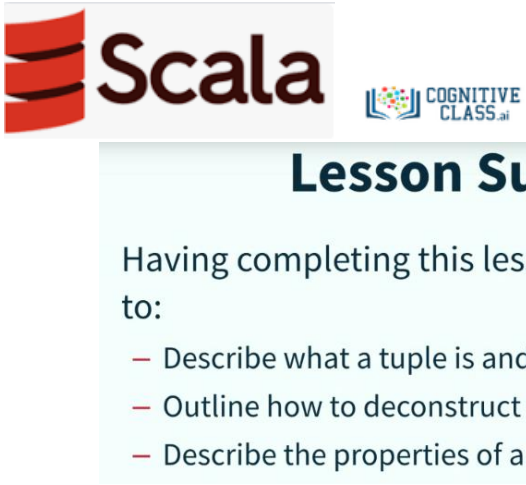
```
scala> res0.zip(res1)
res2: scala.collection.immutable.IndexedSeq[(Int, Char)] =
  Vector((1,a), (2,b), (3,c), (4,d), (5,e))

scala> res2.toMap
res3: scala.collection.immutable.Map[Int,Char] =
  Map(5 -> e, 1 -> a, 2 -> b, 3 -> c, 4 -> d)
```

Maps

```
scala> Map(1 -> "a", 2 -> "b", 3 -> "c")
res0: scala.collection.immutable.Map[Int,String] =
  Map(1 -> a, 2 -> b, 3 -> c)

scala> res0(1)
res1: String = a

scala> res0(4)
java.util.NoSuchElementException: key not found: 4
  at scala.collection.MapLike$class.default(MapLike.scala:228)
  at scala.collection.AbstractMap.default(Map.scala:59)
  at scala.collection.MapLike$class.apply(MapLike.scala:141)
  at scala.collection.AbstractMap.apply(Map.scala:59)
  ... 33 elided
```

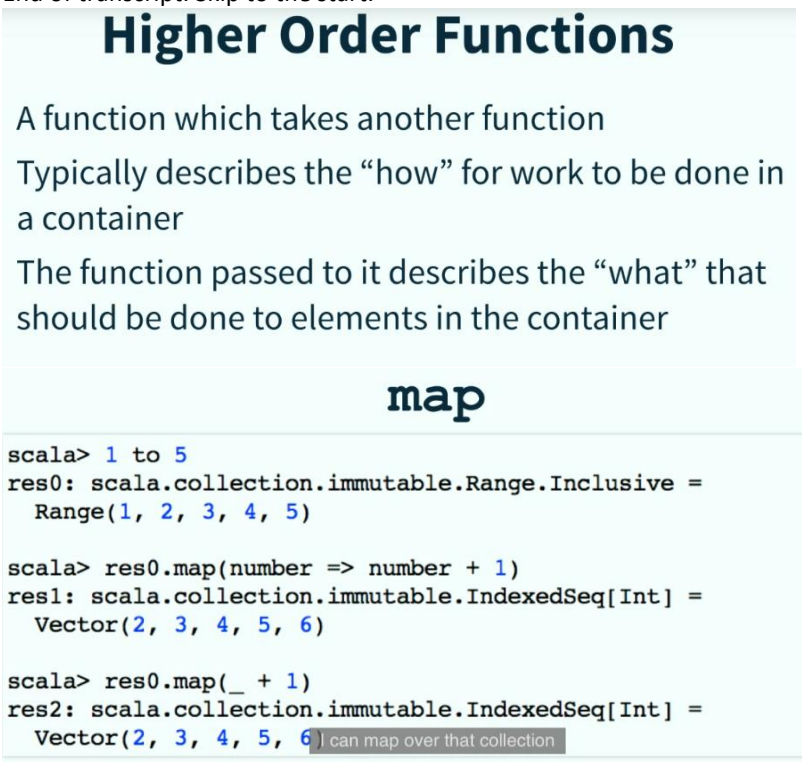
Maps

```
scala> val map = Map(1 -> "a", 2 -> "b")
map: Map[Int,String] = Map(1 -> a, 2 -> b)

scala> map(1)
res0: String = a

scala> map.get(9)
res1: Option[String] = None

scala> map.getOrElse(1, "z")
res2: String = a

scala> map.getOrElse(9, "z")
res3: String = z
```



kip to the end.
welcome two tuples and maps after completing this lesson we should be able to describe what a two boys and how they are used some people call them tuples out my how to deconstruct to polls and describe the properties of a map to polls are loose aggregation of values into a single container and they can have up to twenty two values in scholar they're always use when you see parentheses wrapping data without pointing to a specific kind of type so here's an example of a two point I could create a tuple to which is going to have to values inside of it and provide the two values I want to place into it i could also create another 2.2 with different values inside of it both ends as opposed to an end and strength another syntex I can use what you see at the bottom is sooo not specify the tuple to type instead allow the Scala compiler to figure out this is a tuple too because I'm wrapping paper ends around n number of values if I put five values in here it would have resulted in the tuba 520 can be accessed using a one based access server each value which compared to an array is zero-based might be a little confusing to post can be deconstructed into names bound to each value in a tuple so for example I create a two-point instance here that I called to pull and I said it equaled 24 values inside of parentheses this results in a two-point stance with the types of the values represented inside of the tuple and then I use an underscore three to access the third value out of the four which gives me back the two I can deconstruct that to fall by saying that I have a bowel and have parentheses for every value inside of the tube and I give it a name that I want to buy into that value so whenever I create these first second third fourth values that is now the name bound to the individual values inside of the two pole to pole Tues are frequently called a pair and they have a unique syntax for the values when you create them again I could just wrapping paper ends the values that I want to place into the pair or I could use the arrow between them when I say to give hype in greater than to represent arrow of a key to a value in the pair I can only do this repairs as the syntactic sugar for that arrow is going to try to aggregate values into pairs as you'll see in the last example if I try to do this with three values it's going to create a purse pair which is then pared to another value where there is a pair of a pair inside we saw in an earlier session that the unemployment that in case class companion objects will bring back the values inside of that class so here we have that case class time with hours and minutes inside of it and we create an instance of that time if we then use the on applying the companion object we received back an option of values which are held in a to pull maps are another collection type they represent a grouping of data by key to value which are held inside of this collection as to poor countries it allows you to index values by specific key for fast access and common implementations you may see include HashMap or tree maps I can create a range of values using the 125 syntax and I can create a range of characters as well by using a to G one of the things I can do to show the relationship of two poles to maps is that I can then take the 125 values and sip them to those letters which is going

to pair them up just like a zipper where the one is paired to a Tues pared to be etcetera
as long as both the ranges we use values that can be paired when one of them runs out the zipping stops I can then call to map on the resulting collection which was a vector repairs to create a key-value relationship inside of a map from those two pools if I have a map I can created again using the comp Canyon object apply and giving individual entries as two pairs and then I use the index the key value for the instance apply lock up if it finds that key then it returns the value associated with that key however if it does not find that key it throws an exception because this is an opinionated look up with the expectation that the value should be found for that key if we do not want to use an opinionated look up we can instead use to get method in this life we see that we have a map created with two pairs of 12 in a in a tutu Abby if I look up one directly I know I'll find that value but I'm not sure I'm going to find the value 49 and if I do a get on the map it returns an option of the resulting value which is in this case none because nine is not a key in our map I can also use get or else which we saw on the option before which allows us to provide a default value for a key lookup in case it's not there if I use the one key I get back to value expect if I used the nine key I get the default values my response
having completed this lesson he should be able to describe what a two boys and how they are used outline how to deconstruct two pools and describe the properties of a map
End of transcript. Skip to the start.



flatMap

```
scala> List("Scala", "Python", "R")
res0: List[String] = List(Scala, Python, R)

scala> res0.map(lang => lang + "#")
res1: List[String] = List(Scala#, Python#, R#)

scala> res0.flatMap(lang => lang + "#")
res2: List[Char] =
  List(S, c, a, l, a, #, P, y, t, h, o, n, #, R, #)
```

filter

```
scala> List("Scala", "Python", "R", "SQL")
res0: List[String] = List(Scala, Python, R, SQL)

scala> res0.filter(lang => lang.contains("S"))
res1: List[String] = List(Scala, SQL)
```

foreach

```
scala> List(1, 2)
res0: List[Int] = List(1, 2)

scala> res0.map(println)
1
2
res1: List[Unit] = List((), ())

scala> res0.foreach(println)
1
2
```

Instead, I could use the

forall

```
scala> List("Scala", "Simple", "Stellar")
res0: List[String] = List(Scala, Simple, Stellar)

scala> res0.forall(lang => lang.contains("S"))
res1: Boolean = true

scala> res0.forall(lang => lang.contains("a"))
res2: Boolean = false
```

reduce

```
scala> 1 to 5
res0: scala.collection.immutable.Range.Inclusive =
  Range(1, 2, 3, 4, 5)

scala> res0.reduce((acc, cur) => acc + cur)
res1: Int = 15

scala> res0.reduce(_ + _)
res2: Int = 15

scala> List[Int]().reduce((acc, cur) => acc + cur)
java.lang.UnsupportedOperationException: empty.reduceLeft
  ... 37 elided
```

In this case, I create a range of values from one to five

fold, foldLeft, foldRight

```
scala> 1 to 5
res0: scala.collection.immutable.Range.Inclusive =
  Range(1, 2, 3, 4, 5)

scala> res0.foldLeft(0){ case (acc, cur) => acc + cur}
res1: Int = 15

scala> List[Int]().foldLeft(0){ case (acc, cur) => acc + cur}
res2: Int = 0
```

product

```
scala> 1 to 5
res0: scala.collection.immutable.Range.Inclusive =
  Range(1, 2, 3, 4, 5)

scala> res0.product
res1: Int = 120
```

exists

```
scala> 1 to 5
res0: scala.collection.immutable.Range.Inclusive =
  Range(1, 2, 3, 4, 5)

scala> res0.exists(num => num == 3)
res1: Boolean = true

scala> res0.exists(num => num == 6)
res2: Boolean = false
```

find

```
scala> 1 to 5
res0: scala.collection.immutable.Range.Inclusive =
  Range(1, 2, 3, 4, 5)

scala> res0.find(num => num == 3)
res1: Option[Int] = Some(3)

scala> res0.find(num => num == 6)
res2: Option[Int] = None
```

groupBy

```
scala> 1 to 5
res0: scala.collection.immutable.Range.Inclusive =
  Range(1, 2, 3, 4, 5)

scala> res0.groupBy(num => num % 2)
res1: Map[Int,scala.collection.immutable.IndexedSeq[Int]] =
  Map(1 -> Vector(1, 3, 5), 0 -> Vector(2, 4))
```

Lesson Summary

Having completing this lesson, you should be able to:

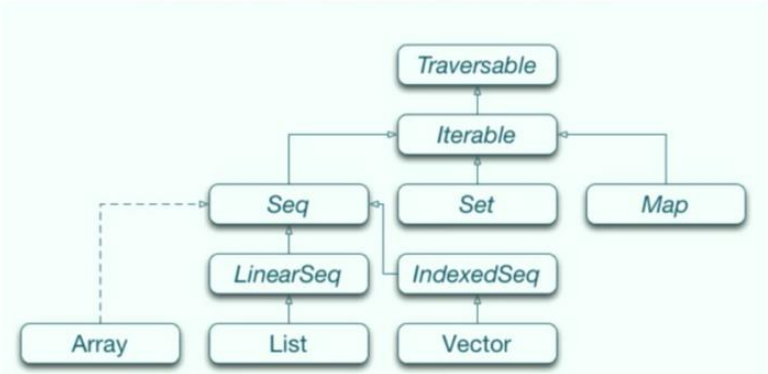
- Describe the application of functions to data
- Outline basic usages of higher order functions in Scala

Start of transcript. Skip to the end.
welcome to higher-order functions after completing this lesson you should be
able to describe the application of functions to data and outline basic
usages of higher-order functions in Scala higher-order function is a
function which takes another function as an argument that is passed to the
higher-order functions of the can perform work on the container it
typically describes how work will be done on the container and the function
is passed to a described what you want to do to the data that exists inside of
the container if I have a collection of values 125 in a range I can map over
that collection by saying for every number in the collection had 12 each
value and returns to me a new collection of values with every single one of them
added by one there's another syntex we can use at the bottom of the slide where
instead of giving a name for every value that is being used in the function
instead use the placeholder underscored represent the value this is a more
success syntax however it's not quite as readable we also have the higher-order
function flat map and what flag map will do is apply the function and then
flatten it down one level in this case we have a list of strings and each
string and scholars actually a sequence of characters so the list is a list of
sequence of characters and two collections are in play if I map over
the list and a panda hash to the end of every string I get a new list with each
string and a half at the end of each one if I plat map instead it's going to
perform the same transformation where the hash is appended to the end of every
string and that's going to flatten one level so the sequence of characters is
now broken apart and we end up with a list of characters another higher order
functions filter if we have a list of Scala Python are and ask you well as
strength we can return a subset of these values by applying a predicate
a function that returns a true or false condition in this case I apply the
filter higher-order function with a function that says only return those
strings that contain the capital letter S and in this case I end up with a
subset of the original list of only skyline SQL each of the previous
higher-order functions resulted in a new collection being created for the
transformation that was performed but there isn't something we want to do in
every case we have another higher order functions called for each which allows
us to apply the function in effect for way but not return about you if I have a
list of values and IMAP over that list and provide a function such as print
land which is going to merely print out the values inside of the collection I
end up printing the values but then I have a result that I don't need and it's
a waste of memory resources on the JVM instead I could use the for each
higher-order functions and apply the print function to it the sprinting out
the values and not creating the extra values I don't need a similarly named
higher-order function is for all but in this case we're going to look at the
values inside of our container and see if any meat that condition again this
one takes a predicate much like filter did if I have a list of values in this
case Scala simple and stellar and I say for all do all of the values and cited
this contain the capital letter S and I get a result of true if I provide a
value that is not in every one of these strains I return false
another well-known higher-order function in the functional programming world is
reduce and this is a powerful concept we see used in the MapReduce world for
transforming data to make sense of it in this case I create a range of values
from one to five and I want to reduce it such that I add the values together but
what's really happening here as I haven't accumulated value for the
application of the function each time for each value and I end up with a value
15 this is very simple 22 however it has some drawbacks
first of all you could use the underscore syntax here for placeholder
but you could use it twice and it doesn't make it quite as clear what

you're doing
secondly if you have an empty list as we see at the bottom of the slide and we've tried to perform the exact same reduction we get an unsupported operation exception because we have an empty list and we're trying to reduce it this isn't behavior that you want to see in production the way to get around that is to use the more general fold functions again I create the values 125 and I say that I'm going to fold left in this case because i dont have an unending stream of data that I'm going to work with I'm going to give a initial value 0 and I'm gonna perform the exact same transformation where I merely adding the values together based upon an accumulated value you'll notice that I have a case key word here don't worry about that for now we're going to talk about what that means later on however if I do have an operation on an empty list were a try to fold left in this case I end up with a value of zero and not an exception and other well-known function is to provide the product of a calculation of numbers if I have a collection of values 125 and I use the product function on it is going to multiply them together one by one so I multiply one times to get 22 times three to get 66 times for to get twenty four 24 times five to get 120 there's also the exists higher-order function if I have values from 125 I can pass in a predicate which tells me whether or not something exists within the collection and this will short-circuit when it binds the first one that meets this condition I can do exist asking is there a value 43 inside of my range and it will return true if I ask for six it will return false
similar higher-order function is called find and if I had the values 125 and I say find the value if I have three in my product it is going to return an option of about you because find is making the lack of assertion that I know the value is there if it is not found it returns none
another higher order functions we saw earlier on his group by I can have my values 125 and then do a modulus function on them to separate them out by those that are even in those that are odd I end up with a map of key the remainder of 12 those values that were awed and remainder 02 those values that were even finally we have the take while and drop while higher order functions in this case I have my values from 125 I can you stick while to return a collection that only represents the values that meet a predicate in this case are the numbers less than three and I end up with the range of values of only one into if I use dropped while it's going to remove those that do not meet this predicate up until it finds one that does and this case I say the numbers have to be greater than three because I'm making the assertion to drop everything until I find about you that is not less than three and in this case I end up with a range of 345
having completed this lesson we should be able to describe the application of functions to data and outline basic usages of higher-order functions in Scala
End of transcript. Skip to the start.

>>Lab

The following diagram summarises the Scala collections hierarchy



4.2 Sequences and Sets

By the end of this section you should be able to:

- Describe the various kinds of sequence collections and their properties
- Outline the desirable properties of the Vector collection type
- Describe the features of set collections

Arrays:

Here are some properties of arrays:

- Arrays are an ordered sequence of data that is fixed in size
- Arrays are fast on the JVM
- Array values are contiguous in memory
- Arrays are indexed by position
- Array are mutable

You can create an array using the following command:

```
val FirstArray = Array(1,2,3,4,5)
FirstArray
```

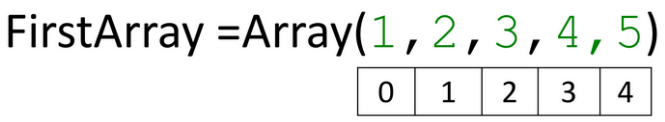
```
FirstArray = Array(1, 2, 3, 4, 5)
Array(1, 2, 3, 4, 5)
```

You can create an array using the following command:

```
val FirstArray = Array(1,2,3,4,5)
FirstArray
```

```
FirstArray = Array(1, 2, 3, 4, 5)
Array(1, 2, 3, 4, 5)
```

Arrays are zero-based as shown in the following figure:



Array with equivalent indexes

You can access the first value of an array as follows :

```
FirstArray(0)
```

```
Name: Compile Error
Message: <console>:26: error: not found: value FirstArray
       FirstArray(0)
       ^

StackTrace:
```

and set the last value

```
FirstArray(4) = 5
```

```
Name: Compile Error
Message: <console>:26: error: not found: value FirstArray
       FirstArray(4) = 5
       ^
```

Arrays are mutable, as a result you can change them. For example, you can change the 5-th element to 4 and print the result:

```
FirstArray(4) = 100000
FirstArray.foreach(println)
```

```
Name: Compile Error
Message: <console>:28: error: not found: value FirstArray
       FirstArray.foreach(println)
       ^
<console>:25: error: not found: value FirstArray
       FirstArray(4) = 100000
       ^

StackTrace:
```

Once you create an array, you can not change the size, for example, if you try to add a 6-th element you will get an error, try running the following line of code:

```
FirstArray(5)
```

```
Name: Compile Error
Message: <console>:26: error: not found: value FirstArray
       FirstArray(5)
       ^
```

Question 4.1:

Create an Array A such that: A[0]:"A",A[1]:1,A[2]:"B",A[3]:2 .

```
val A=Array("A",1,"B",2)
```

```
A = Array(A, 1, B, 2)
Array(A, 1, B, 2)
```

Question 4.2:

The second element of the array "A" has a value of 1, convert it to a value of "one" .

```
a(1)="one"
print(A)
```

```
Name: Unknown Error
Message: <console>:27: error: not found: value a
       a(1)="one"
```

Lists:

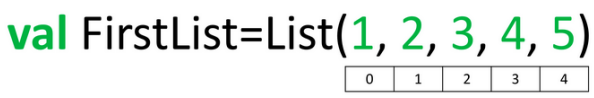
Here are some properties of lists:

- Lists are implemented pairs of head and tail.
- Theoretically limitless in size
- Look ups are fast when accessing the head of the list. Random access is less performant because the list has to be traversed before an element can be retreived.

```
val FirstList = List(1,2,3,4,5)
FirstList
```

```
FirstList = List(1, 2, 3, 4, 5)
List(1, 2, 3, 4, 5)
```

Lists are zero-based as shown in the following figure:



List with equivalent indexes

you can access the first, second or third element as follows:

```
println("1st element", FirstList(0))
println("second element", FirstList(1))
println("third element ", FirstList(2))
```

```
(1st element,1)
(second element,2)
(third element ,3)
```

Lists are immutable as a result you can't change them

```
FirstList(0) = "A"
```

```
Name: Compile Error
Message: <console>:28: error: value update is not a member of List[Int]
       FirstList(0) = "A"
       ^

StackTrace:
```

you can create a new list and add a value to a list by appending with a colon and an addition sign

```
val SecondList = FirstList :+ 6
SecondList
```

```
SecondList = List(1, 2, 3, 4, 5, 6)
List(1, 2, 3, 4, 5, 6)
```

you can prepend the item on a list as follows

```
val ThirdList = 0 +: SecondList
ThirdList
```

```
ThirdList = List(0, 1, 2, 3, 4, 5, 6)
List(0, 1, 2, 3, 4, 5, 6)
```

As lists are sequences, you can add duplicate values

```
val LastList = List("A","A",1,1,"B",2,2)
LastList
```

```
LastList = List(A, A, 1, 1, B, 2, 2)
List(A, A, 1, 1, B, 2, 2)
```

As illustrated by the above cell, as lists are sequences they can have duplicate values. We can remove duplicates and assign them to other values using the distinct method.

```
val DistinctList = LastList.distinct
DistinctList
```

```
DistinctList = List(A, 1, B, 2)
List(A, 1, B, 2)
```

Question 4.4:

Create the list with "WrongAlphabet" the elements B,C,D

```
val WrongAlphabet=List("B","C","D")
```

```
WrongAlphabet = List(B, C, D)
List(B, C, D)
```

Question 4.5:

Prepend the letter "A" and append the letter "E" and assign it the variable "Alphabet"

```
val Alphabet="A"+:WrongAlphabet:+"E"
```

```
Alphabet = List(A, B, C, D, E)
List(A, B, C, D, E)
```

Vector

Vectors are similar to lists but have properties that allow you to access the values faster. Some of the properties can be summarised as followed:

- A linked list of 32 elements arrays
- T1.15 billion possible elements
- Indexed by hashing

```
val FistVector = Vector(1,2,3,4,5)
FistVector
```

```
FistVector = Vector(1, 2, 3, 4, 5)
Vector(1, 2, 3, 4, 5)
```

you can access the first, second and third element as follows:

```
println("1st element: " + FistVector(0))
println("second element: " + FistVector(1))
println("third element: " + FistVector(2))
```

```
1st element: 1
second element: 2
third element: 3
```

vectors are immutable, uncomment the following code to see what happens when you try to change an element of a vector

you can prepend an item

```
val ThirdVector = 0 +: FistVector
ThirdVector
```

```
ThirdVector = Vector(0, 1, 2, 3, 4, 5)
Vector(0, 1, 2, 3, 4, 5)
```

Set

Sets are a" bag of data," were no duplicates are permitted. The order is not guaranteed inside of the set. We can create a set as follows:

```
val FirstSet = Set(0,1,2,3,4)
FirstSet
```

```
FirstSet = Set(0, 1, 2, 3, 4)
Set(0, 1, 2, 3, 4)
```

This set as the elements 0,1,2,3,4 you can insert an element into a set using the "+" as follows and assign it to the second set "SecondSet":

```
val SecondSet = FirstSet + 5
SecondSet
```

```
SecondSet = Set(0, 5, 1, 2, 3, 4)
Set(0, 5, 1, 2, 3, 4)
```

As a result, the set now contains a 5, the elements 0,1,2,3,4,5. As there are no duplicates in a set we can add another 5 to the set and assign it to the variable "ThirdSet" :

```
val ThirdSet = SecondSet + 5
ThirdSet
```

```
ThirdSet = Set(0, 5, 1, 2, 3, 4)
Set(0, 5, 1, 2, 3, 4)
```

The two sets have the exact same number of elements, we can verify this using the familiar equality syntax:

```
ThirdSet == SecondSet
```

```
true
```

Question 4.6:

Create a new set " Set3" from "Set2" the has all the same elements as "Set1"and verify they are equal

```
val Set1=Set("A","B","C")
val Set2=Set("A","B")
val Set3=Set2+"C"
Set3==Set1
```

```
Set1 = Set(A, B, C)
Set2 = Set(A, B)
Set3 = Set(A, B, C)
true
```

4.3 Options

By the end of this section you should be able to:

- describe the relevance of option in the Scala type syst
- outline how to use option inside of your types

consider the option of a string

```
val Name = Option("Jamie")
Name
```

```
Name = Some(Jamie)
Some(Jamie)
```

I can get the value inside the wrapping

```
Name.get
```

```
Jamie
```

Its better practice to use the "getOrElse()" method, if the element is not present it will return the option, in this case, "Jamie". Consider if we look for Bob, as Bob is not in the string the value Jamie is returned.

```
Name.getOrElse("Bob")
```

```
Jamie
```

4.4 Tuples and Maps

By the end of this section you should be able to:

- to describe what is a tuple and how they are used
- Outline how to deconstruct tuples
- Describe the properties of map

Tuples

Tuples can be used to wrap different types of data ; they can have up to 22 values. Typically, tuples are created with parentheses. You can also create tuples explicitly by class that guarantess how many elements a tuple will have. Here are two examples of how to generate a tuple of 2 elements:

```
Tuple2(1,"a")
```

```
(1,a)
```

```
(1,"a")
```

```
(1,a)
```

Tuples can be accessed using a 1-based accessor; we can access the first two elements as follows:

```
val fist_tuple=(1,"a")
println("First index:",fist_tuple._1)
println("Second index:",fist_tuple._2)
```

```
(First index:,1)
(Second index:,a)
fist_tuple = (1,a)
```

```
(1,a)
```

COGNITIVE CLASS.ai

Question 4.7:

Create a tuple with for elements: 1, "a",3,"b" just using parenthesis

```
val mytuple=(1,"a",3,"b")
```

```
mytuple = (1,a,3,b)
(1,a,3,b)
```

tuples can be deconstructed into names bound to each value in a tuple

```
val tuple = (1,"a",2,"b")
```

```
tuple = (1,a,2,b)
(1,a,2,b)
```

You can deconstruct a tuple, simply assign the tuple to another tuple with the variables you would like to bind each element too. For example, you can bind the first, second, third and fourth element of the variable "tuple" to the variable `first`, `second`, `third` and `forth` as follows:

```
val (first, second, third, fourth) = tuple
```

```
first = 1
second = a
third = 2
fourth = b
b
```

the variable take on the values in the tuple accordingly

```
first
```

```
1
```

```
second
```

```
a
```

```
third
```

```
2
```

```
fourth
```

the process is illustrated in the following figure

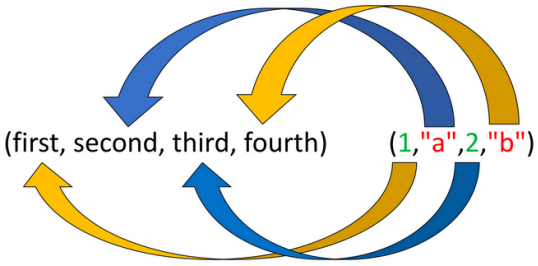


illustration of deconstructing a tuple

Tuples with two elements are frequently called a pair and they have a unique syntax for values. You can create the Tuple with two elements the standard way:

illustration of deconstructing a tuple

Tuples with two elements are frequently called a pair and they have a unique syntax for values. You can create the Tuple with two elements the standard way:

```
(1,"a")
```

```
(1,a)
```

You can also use the "->" as follows to create a tuple:

```
2 -> "b"
```

```
(2,b)
```

You can use this rocket symbol "->" to nest tuples that consist of two elements:

```
3 -> "c" -> 4
```

```
((3,c),4)
```

you can repeat the process

```
0 -> 0 -> 1 -> 2 -> 3 -> 4
```

```
(((((0,0),1),2),3),4)
```

Maps

Did you know? IBM Watson Studio lets you build and deploy an AI solution, using the best of open source and IBM software and giving your team a single environment to work in. Learn more here.

Maps group data from key to a value, you can use the key to access the value.

We can create a range of numbers and a range of characters as follows:

```
val Numbers=1 to 5
Numbers
```

```
Numbers = Range(1, 2, 3, 4, 5)
Range(1, 2, 3, 4, 5)
```

```
val Letters= 'a' to 'e'
Letters
```

```
Letters = NumericRange(a, b, c, d, e)
NumericRange(a, b, c, d, e)
```

We can use the zip function to place both "Numbers" and "Letter" into a new vector, each element of the vector contains a tuple. The tuple has one element from "Numbers" and one from "Letters".

```
val LettertoNumber = Letters.zip(Numbers)
 LettertoNumber
```

```
LettertoNumber = Vector((a,1), (b,2), (c,3), (d,4), (e,5))
Vector((a,1), (b,2), (c,3), (d,4), (e,5))
```

We can use the function "toMap" to create a key value relation between those tuples

```
val MapLettertoNumber = LettertoNumber.toMap
```

```
MapLettertoNumber = Map(e -> 5, a -> 1, b -> 2, c -> 3, d -> 4)
Map(e -> 5, a -> 1, b -> 2, c -> 3, d -> 4)
```

we can use the key to look up the value as follows:

```
println("for key a", MapLettertoNumber('a'))
println("for key b", MapLettertoNumber('b'))
println("for key c", MapLettertoNumber('c'))
println("for key d", MapLettertoNumber('d'))
println("for key e", MapLettertoNumber('e'))
```

```
(for key a,1)
(for key b,2)
(for key c,3)
(for key d,4)
(for key e,5)
```

The process is summarized in the following figure after the function zip is applied, we use the `toMap` function. We represent the value `MapLettertoNumber` as a table where the keys are represented in the first column, and the values are shown in the second column.

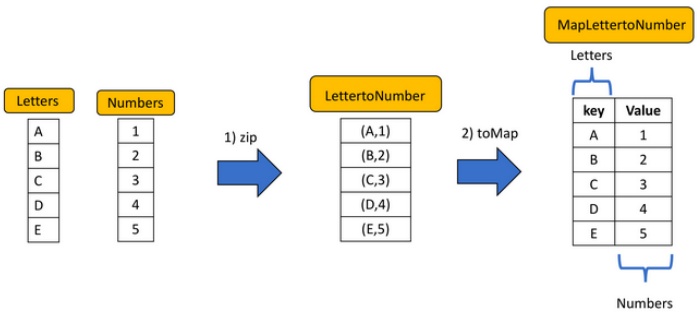


illustration of deconstructing a tuple

If we try to use a key that does not exist we get an exception:

```
MapLettertoNumber('z')
```

```
Name: java.util.NoSuchElementException
Message: key not found: z
StackTrace:   at scala.collection.MapLike$class.default(MapLike.scala:228)
  at scala.collection.AbstractMap.default(Map.scala:59)
  at scala.collection.MapLike$class.apply(MapLike.scala:141)
  at scala.collection.AbstractMap.apply(Map.scala:59)
```

If your not sure if the key exists you can use the "get" method. It will return an `Option` of a value at the specified key. If the value is in the Map, you will get a `Some` of a value. If the value is not in the map, you will get a `None`.

```
MapLettertoNumber.get('z')
```

```
lastException: Throwable = null
None
```

`getOrElse` that allows us to provide a default option, in this case the default is the key is not in the Map.

```
MapLettertoNumber.getOrElse('z',"not here" )
```

```
not here
```

If the key exists it returns the actual value

```
MapLettertoNumber.getOrElse('b',"not here" )
```

```
2
```

4.5 Higher Order Functions

Map

If I have a collection of values 1 to 10 in the value "myNums"

```
val myNums = 1 to 10
myNums
```

```
myNums = Range(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
Range(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

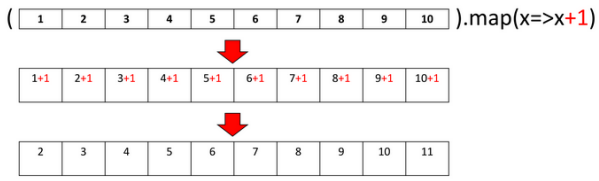
You can map over the collection by adding 1 to each value and returning the result to a new collection "NewmyNums". The values has a 1 added to it

```
val NewmyNums = myNums.map(n=>n+1)
NewmyNums
```

```
NewmyNums = Vector(2, 3, 4, 5, 6, 7, 8, 9, 10, 11)
Vector(2, 3, 4, 5, 6, 7, 8, 9, 10, 11)
```

The process is demonstrated in the following figure.

The process is demonstrated in the following figure.



Example of a map in function

Question 4.8:

Use the map function to to multiply every number in the list by 2

```
val SomeNumbers=List(1,10,100)
SomeNumbers
```

```
SomeNumbers = List(1, 10, 100)
List(1, 10, 100)
```

```
SomeNumbers.map(x=>x*2)
```

```
List(2, 20, 200)
```

The function "contains" returns "true" if the string contains a sub-string, consider the string "Python" contains "Py" as follows

```
"Python".contains("Py")
```

```
true
```

The function returns a `True` if the function does contain the sub-string, otherwise we get a `False`. For example :

```
"Python".contains("Sc")
```

```
false
```

We can apply the function to each element in the list. Consider the following list:

```
val ProgrammingLanguages=List("Scala","Python","R","SQL")
ProgrammingLanguages
```

```
ProgrammingLanguages = List(Scala, Python, R, SQL)
List(Scala, Python, R, SQL)
```

We can see if each element contains an "S" and return a new list that contains a True if the corresponding element contains "S" or else it will return a false

```
ProgrammingLanguages.map(s => s.contains("S"))
```

```
List(true, false, false, true)
```

Flat Map

The function "flatmap" will apply the function and then flatten it down one level. In this case, we have a list of strings; each string is a sequence of characters.

```
val ProgrammingLanguages=List("Scala","Python","R","SQL")
ProgrammingLanguages
```

```
ProgrammingLanguages = List(Scala, Python, R, SQL)
List(Scala, Python, R, SQL)
```

We can use the function "map" to add a hash to each element in the list:

```
ProgrammingLanguages.map(lang=>lang+"#")
```

```
List(Scala#, Python#, R#, SQL#)
```

If we apply a "flatMap", it will perform the map, but it's going to flatten one level so the sequence of characters is now broken apart and we end up with a list of characters:

```
ProgrammingLanguages.flatMap(lang=>lang+"#")
```

```
List(S, c, a, l, a, #, P, y, t, h, o, n, #, R, #, S, Q, L, #)
```

Alternatively, consider this example where we use `flatMap` to flatten a list of lists.

```
val lst = List(1,2,3)
lst.map(i => List(i))
```

```
lst = List(1, 2, 3)
List(List(1), List(2), List(3))
```

```
lst.flatMap(i => List(i))
```

```
List(1, 2, 3)
```

Filter

The function filter allows you to return a subset of the values by applying a predicate function. The predicate function returns a true or false condition if the condition is false the elements will not be returned.

```
val ProgrammingLanguages = List("Scala","Python","R","SQL")
ProgrammingLanguages
```

```
ProgrammingLanguages = List(Scala, Python, R, SQL)
List(Scala, Python, R, SQL)
```

We apply the predicate function "contains" with an argument of "S". If a string in the list does not contain an "S" it will be filtered out. The result is the new list only contains the string "Scala" and "SQL".

```
val ProgrammingLanguagesWithS = ProgrammingLanguages.filter(s => s.contains("S"))
ProgrammingLanguagesWithS
```

```
ProgrammingLanguagesWithS = List(Scala, SQL)
List(Scala, SQL)
```

Question 4.10:

Use the filter function to filter out any string in the list "List("AB","BC","BC","DB")" with the letter "A"

```
List("AB","BC","BC","DB").filter(x=>x.contains("A"))
```

```
List(AB)
```

Foreach

Higher-order functions result in a new collection being created for the transformation that was performed; we do not want to do this in every case. We have another higher order function called "for each" which allows us to apply the function. The advantage is that this function does not return a list of values.

Let's say we use "map" to apply the "println" function:

```
ProgrammingLanguages.map(println)
```

```
Scala
Python
R
SQL
List((), (), (), ())
```

The function prints each element in the list, but it returns a new list "List((), (), (), ())" this is a waste of memory. We can use the function "foreach" instead.

```
ProgrammingLanguages.foreach(println)
```

```
Scala
Python
R
SQL
```

As you can see using "foreach" does not return a new list.

Forall

A similar named higher-order function is "forall". In this case, we're going to look at the values inside of our container and see if any meat that condition by using a predicate function

Consider the following list, each string in the list contains an "A" :

```
val ABC=List("A","AB","ABC")
```

```
ABC = List(A, AB, ABC)
List(A, AB, ABC)
```

If we apply the function "contains("A")" using the "forall" we get a True as each element contains a "A"

```
ABC.forall(s => s.contains("A"))
```

```
true
```

If we apply the function "contains("B")" using the "forall" we get a False as only 2 of the three elments contain a "B"

```
ABC.forall(s => s.contains("B"))
```

```
false
```

Reduce

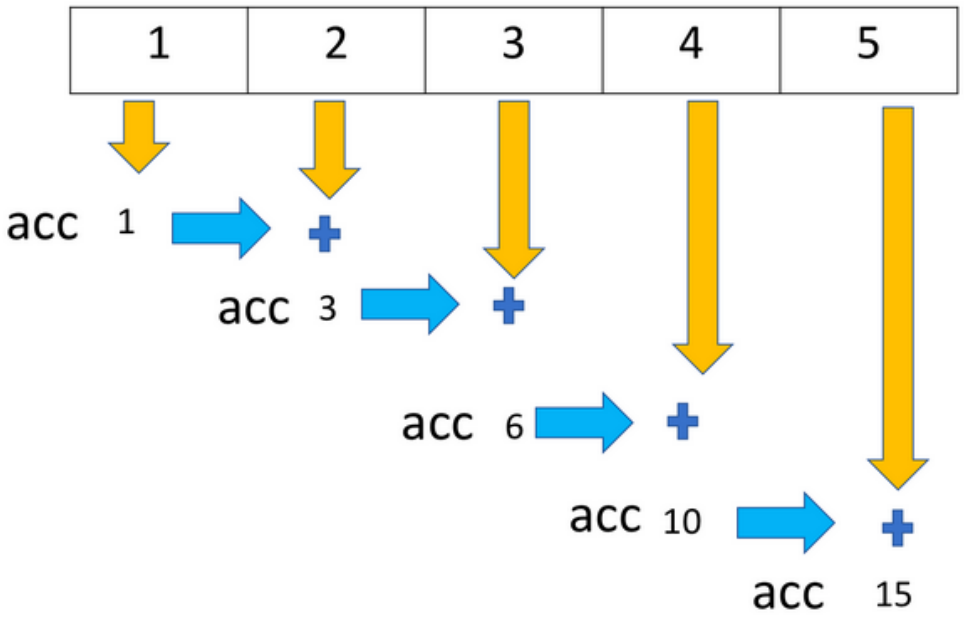
Another well-known higher-order function in the functional programming world is reduced. This is a powerful concept we see used in the Map Reduce world for transforming data,let's say we would li to perform a summation on a the myNums of values from one to five.

```
val myNums= 1 to 5
myNums
```

```
myNums = Range(1, 2, 3, 4, 5)
Range(1, 2, 3, 4, 5)
```

The process works by adding the current element "cur" to an accumulator "acc".

```
myNums.reduce((acc,cur) => acc+cur)
```



Example of the reduce in function

we can square the value before we add it.

```
myNums.reduce((acc,cur) => acc+cur*cur)
```

```
55
```

we can also square the accumulator value for each value

```
myNums.reduce((acc,cur) => acc*acc+cur)
```

```
21909
```

```
myNums.reduce((acc,cur) => acc+2*cur)
```

If you have an empty list and try to perform the same reduction, we get an unsupported operation exception because we have an empty list and we're trying to reduce it.

Try running this command : List[Int]().reduce((acc,cur) => acc+cur)

To get around this we use the more general fold functions again we create the values 1 to 5. We will use the function "foldLeft()". We are going to give an initial value 0 i.e foldLeft(0).

```
val myNums= 1 to 5
myNums.foldLeft(0){case (acc,cur) => acc+cur}
```

```
myNums = Range(1, 2, 3, 4, 5)
15
```

If we apply the operation to an empty List we get a zero.

```
List[Int]().foldLeft(0){case (acc,cur) => acc+cur}
```

```
0
```

Product

The product function provides the product of a sequence of numbers, for example, we can calculate the product of all the integers from 1 to 5.

```
myNums.product
```

Exists

There's also the `exists` higher-order function. We can pass in a predicate. If the condition is met it will return a `true`. In the next cell, we use an anonymous function `num => num == 3` in combination with `exists` to check if any of the elements in `myNums` are equal to `3`.

```
myNums.exists(num => num==3)
```

```
true
```

If we would like to see if any of the numbers in the sequence is equal to 1000 we can apply the following:

```
myNums.exists(num => num == 1000)
```

```
false
```

A similar higher-order function is called find if we had the values from 1 to 5 and applied the function its going to return the option of a value of the option,in this case 3

```
myNums.find(x => x == 3)
```

```
Some(3)
```

If the value is not found it returns none

```
myNums.find(x=>x==100)
```

group By

Another higher-order function is group by. We can have values from 1 to 5 ; we can group them into odd and evens. We simply apply the modules function and placed into a map, where the key is the remainder, and the value is the number.

```
val NewMap=myNums.groupBy(num=>num%2)
NewMap
```

```
NewMap = Map(1 -> Vector(1, 3, 5), 0 -> Vector(2, 4))
Map(1 -> Vector(1, 3, 5), 0 -> Vector(2, 4))
```

```
NewMap(0)
```

```
Vector(2, 4)
```

```
NewMap(1)
```

```
Vector(1, 3, 5)
```

takeWhile and dropWhile

Finally, we have the **takeWhile** and **dropWhile** higher order functions. For example we have the values from 1 to 5 we can return a collection that only represents the values that meet a condition using the **takeWhile** function, in this case, the numbers less than three

```
myNums.takeWhile(num=>num<3)
```

```
Range(1, 2)
```

We can drop the values the meet the condition that meet a predicate using the **dropWhile**

```
myNums.dropWhile(num=>num<3)
```

```
Range(3, 4, 5)
```

Lesson 5 - Idiomatic Scala

Learning objectives

5.1. For expressions
After completing this lesson, you should be able to:
● Understand the relationship betwen for expressions and higher order functions
● Describe the usage of for expressions

5.2. Pattern Matching
After completing this lesson, you should be able to:
● Describe how to use pattern matching to handle different values in different ways
● Outline how case classes and ADTs help in pattern matching
● Illustrate how to extract values from tuples

5.3. Handling Options
After completing this lesson, you should be able to:
● Describe how to pattern match on an Option
● Outline how to use higher order functions on an option to avoid nullchecking
● Illustrate how to use for comprehensions to work with Option

5.4. Handling Failures
After completing this lesson, you should be able to:
● Describe the usage and importance of a Try
● Describe how to pattern match on a Try
● Outline how to use higher order functions on a Try
● Illustrate how to use for comprehensions to work with Try

5.5. Handling Futures
After completing this lesson, you should be able to:
● Describe how to use Futures to perform work asynchronously
● Describe how to pattern match on the result of a Future
● Outline how to use higher order functions on the result of a Future
● Illustrate how to use for comprehensions to work with Futures

Composing HOFs

```
scala> val myNums = 1 to 3
myNums: Range.Inclusive = Range(1, 2, 3)

scala> myNums.map(i => (1 to i).map(j => i * j))
res0: IndexedSeq[IndexedSeq[Int]] =
  Vector(Vector(1), Vector(2, 4), Vector(3, 6, 9))

scala> myNums.flatMap(i => (1 to i).map(j => i * j))
res1: IndexedSeq[Int] = Vector(1, 2, 4, 3, 6, 9)
```

For Expressions

```
scala> val myNums = 1 to 3
myNums: Range.Inclusive = Range(1, 2, 3)

scala> for {
     |   i <- myNums
     |   j <- 1 to i
     | } yield i * j
res0: IndexedSeq[Int] = Vector(1, 2, 4, 3, 6, 9)
```

Syntax

- Must start with the **`for`** keyword
- Must have generators, using the **`<-`** arrow
- The **`yield`** keyword dictates whether or not a new value is returned

Syntax

Syntactic sugar over **`map`**, **`flatMap`**, **`withFilter`** and **`foreach`**

Higher Order Functions have rules

- If I **`map`** over a List, I will get a List
- The first generator of a for expression follows the same rule

Can have guard conditions to apply filters

Filtering

```
scala> val myNums = 1 to 3
myNums: scala.collection.immutable.Range.Inclusive =
  Range(1, 2, 3)

scala> for {
     | i <- myNums if i % 2 == 1
     | j <- 1 to i
     | } yield i * j
res0: scala.collection.immutable.IndexedSeq[Int] =
  Vector(1, 3, 6, 9)
```

Effectful Usages

```
for (n <- 1 to 3) println(n)
(1 to 3).foreach(n => println(n))
```

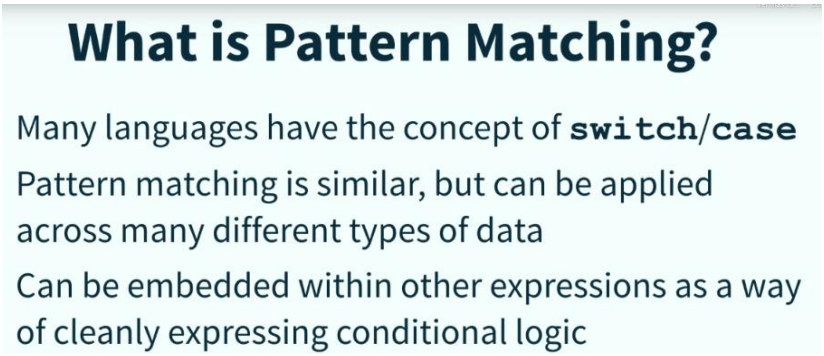
Lesson Summary

Having completing this lesson, you should be able to:

- Understand the relationship between for expressions and higher order functions
- Describe the usage of for expressions

tart of transcript. Skip to the end.
welcome to for expressions after completing this lesson he should be able to understand the relationship between for expressions and higher-order functions and described the usage of for expressions composition is difficult trying to put together multiple higher-order functions into a single expression to transform data can be very hard to do for expressions are syntactic sugar from the Scala compiler that simplifies the work of coding a multi-stage transformation consider trying to put multiple higher-order functions together in this case I've got a range of values it's simply one two and three and I'm going to map over that data and then say for each value in that collection I'm going to loop again I'm going to perform another map so the first time I go through this my I would equal one and then in my internal iteration I have one to one then I'm going to map a single time we're both I NJ or one and I'm going to multiply them together to yield the first value then I'm gonna come back and I'm going to try and make my eye the next value in this case it's going to be too and then I'm going to go to the sub loop where I say 12 to map the first value we won and so in this case I will multiply times J two times one resulting in a value of two then I move my J to the next value to and where I was too and jay is to a yield of value for at this point I popped back out to the external iteration and I move I did three and I iterate through the sub loop 12 13 again this year olds the values of 369 the resulting collection I get is not what I expect however I don't end up with a vector of values altogether but instead with collections inside of that because I'm mapping over one collection creating another collection so I have to now change this to flat map the external iteration to make sure that I flattened

out the internal vectors does she owning the values
wanted in the first place a vector of 124 36 and 94 expressions greatly
simplify the syntax again I have my numbers which are values 123 and I'm
just going to say that I have a for statement and inside of that where I
each value in my moms where J is each value in 12 I in the sub iteration yield
I times J and it takes care of all the details or whether this was flat map map
filtering you name it does yielding the vector that I wanted in the first place
the code is much simpler syntax for for expression is as follows they must start
with the four key word and they must have generators inside them using that
leftward arrow generators get the values out of the containers upon which the
poor comprehension they're working the yield keyword dictates whether or not a
new value is returned as a result of the iteration so for expressions are
syntactic sugar over the map
flat map with filter and for each higher-order functions we used with
filter instead of filter due to laziness higher-order functions have rules if I
map over a list I'm going to get a list in responses what has yielded and poor
comprehension to have to follow the exact same rules
first generator and a poor expression has to follow that rule yielding a list
if the first generator was on a list we can also apply guard conditions in the
form of filters in our for expressions as an example again I have my noms which
is a range of values 123 and I'm going to perform the exact same for expression
I did last time however in this case I'm putting a guard condition with an if
statement on the first line I still have my generator for each I but I'm saying
only give myself a value of I if I mod the value by two
has an odd number once again I have my collection my noms which is a range of
values from 123 I have a poor expression just like I did before but in this case
I added in its statement and I'm asking that if I mod the value by 22 result in
a value of 1 dozen indicating that this is an odd value I will only get value in
the poor expression if it meets this condition otherwise it will pop out as a
result I end up with a smaller yielded vector that does not include the results
of having performed any calculation where I was equal to 20 can also
definitions and side of our for expressions in this case my for is going
to save for each time in a collection of times I have an hour's value them
calculating based upon whether or not the hours are greater than 12 and thus
meaning that this is an afternoon value then you know the hours given that these
could be twenty four-hour representations with a p.m. if they
don't want to YouTube value and instead went to apply a function is going to be
effective tools such as printing out the data I can omit the yield keyword and
instead it merely apply the function to the data as it is being iterated this
translates into a porridge as we saw in the higher-order functions session
having completed this lesson should be able to understand the relationship
between for expressions in higher-order functions and described the usage of
ports pressions
End of transcript. Skip to the start.



The `match` Keyword

```
def isCustomer(someValue: Any): Boolean = {
  someValue match {
    case cust: Customer => true
    case _ => false
  }
}
```

Usage

```
scala> case class Customer(first: String = "",
                           last: String = "")
defined class Customer

scala> Customer("Martin", "Odersky")
res0: Customer = Customer(Martin,Odersky)

scala> isCustomer(res0)
res1: Boolean = true

scala> isCustomer("Martin Odersky")
res2: Boolean = false
```

Exhaustiveness

When you see the **`case`** keyword, pattern matching is in play

Case classes and ADTs provide compile-time exhaustiveness checking that all possible conditions have been have been met

Pattern Matching Tuple Values

```
scala> val tuple = (1, "a", 2, "b")
tuple: (Int, String, Int, String) = (1,a,2,b)

scala> tuple._3
res0: Int = 2

scala> val (first, second, third, fourth) = tuple
first: Int = 1
second: String = a
third: Int = 2
fourth: String = b
```

and assigning them to

Lesson Summary

Having completing this lesson, you should be able to:

- Describe how to use pattern matching to handle different values in different ways
- Outline how case classes and ADTs help in pattern matching
- Illustrate how to extract values from tuples

t of transcript. Skip to the end.
welcome to pattern matching after completing this lesson you should be
able to describe how to use pattern matching to handle different values in
different ways
outline how case classes in algebraic datatypes helping pattern matching and
illustrate how to extract value from two poles so what does pattern matching many
languages have the concept of switch case but pattern matching is much more
powerful construct allows you to match on data and make conditional statements
based upon what is found
here's the simple example of a method that includes a pattern match the method
is called is customer and a value can be passed in of any type if it is a
customer those found a boolean is returned
of whether or not it was customers either true or false
we look at the some value parameter and match on it using the match keyword and
we have an open curly brace with cases inside the cases represented different
conditions that could be represented in this case we are buying a customer type
to a name cust and then saying if it is a customer return true if anything else
is found false is returned and we're using the underscore placeholder we
could have also instead of an underscore used an axe or any combination of
letters that would have been bound to the name but in this case we don't need
it so it's very easy to use the underscore placeholder so let's look at
how this would be used if I have a case class customer with the first and last
name and then create an instance of their customer the name Martin order ski
I passed the instance of the customer to the is customer method and it returns
true to me it is a customer instance regardless of what the values are inside
of it I can then passing a string that happens to have the same values as the
name I passed into the customer but since it is not a customer tight the
pattern match returns false pattern matching is very flexible you can match
on many different kinds of values such as literals like a strength
value and must match exactly in order to return the value that you place on the
right hand side of that arrow we can use guard conditions like we saw in for
expressions to provide extra specificity to each one of the conditions in the
pattern match we can match on only some parts of a value such as I could have
had my is customer check to see whether the last name inside of the customer was
older ski when I have more specific cases they must come first and more
general cases must come after them
the compiler will warn you if a more general case is listed in the pattern
match before the more specific case and if you use the underscore symbol name
with no tie both match on anything found the key to pattern matching is exhausted
missus when you see the case keyword anywhere in Scala pattern matching is in
play
case classes and algebraic datatypes provide compile time exhaustive list
checking to make sure that all possible conditions have been met inside of the
pattern match this way you can make sure compile time that all conditions
possible have been meant we saw earlier in our tuple section that we could
extract the values from a tuple this is actually done through pattern matching
under the hood and we had that too poor to poor values inside of it and then we
asked it to be placed into four different named values as we see at the
bottom of the slide and pattern matching is actually extracting the values in
assigning them to those individual names if I have my collection of values from
one to five and I perform the reduce function we noticed that I could extract
the to pull into the names ACC Amcor without having to use a case keyword
that is due to the structure of the reduce higher-order function on the
other hand fold left has a different structure and in order for us to extract
the values of ACC incur we must explicitly put the case keyword and

rapid and curly braces
having completed this lesson you should be able to describe how to use pattern matching
to handle different values in different ways outline how case classes in algebraic datatypes help in pattern matching and illustrate how to extract values from two pools
End of transcript. Skip to the start.

Pattern Matching an Option

```
def getMiddleName(value: Option[String]): String = {
  value match {
    case Some(middleName) => middleName
    case None => "No middle name"
  }
}
```

Pattern Matching an Option

```
scala> case class Customer(first: String = "",
                           middle: Option[String] = None,
                           last: String = "")

scala> val martin = Customer("Martin", last = "Odersky")
martin: Customer = Customer(Martin,None,Odersky)

scala> getMiddleName(martin.middle)
res0: String = No middle name
```

HOFs and Option

```
scala> Option("Martin")
res0: Option[String] = Some(Martin)

scala> res0.map(name => println("Yay, " + name))
Yay, Martin
res1: Option[Unit] = Some(())

scala> res0.foreach(name => println("Yay, " + name))
Yay, Martin

scala> None.foreach(name => println("Yay, " + name))
```

For Expressions and Option

```
scala> val martin = Option("Martin")
martin: Some[String] = Some(Martin)

scala> val jane = Option("Jane")
jane: Some[String] = Some(Jane)

scala> for {
     |   m <- martin
     |   j <- jane
     | } yield (m + " is friends with " + j)
res1: Option[String] = Some(Martin is friends with Jane)
```

Lesson Summary

Having completing this lesson, you should be able to:

- Describe how to pattern match on an Option
- Outline how to use higher order functions on an option to avoid null-checking
- Illustrate how to use for comprehensions to work with Option

t. Skip to the end.
welcome to handling options after completing this lesson we should be able to describe how to pattern match on an option
outline how to use higher order functions on an option to avoid no checking and illustrate how to use for comprehension to also work with option
is very simple to pattern match on options it's a really can only be one of two values this being an algebraic datatype can only be some value or none
in this case we created a method called get middle name is going to take a value that is an option of his strength and return a string as a result of the pattern match we take the value is passed in then we match on it and if we have some value that we assigned to a name called middle name we return that middle name value if we have none we return a string no middle name when we try to use this we first have our case class customer with the first and last names which are required strains and the middle name which is an option of a string with the default value of money if we create a customer instance with only a first and last name and then we called in get middle name that we just created passing in the Martin . Middlefield we would see the return value of no middle name on the other hand if we created case class instance of a customer named James D
dole we then call the get middle name
passing in Jane done middle and we get the value d period we can also use higher order functions with option in this case we create an option of a string Martin and then we map over the option for the value that is inside of it which we call name we are going to print minimal you will be a plus that name
this will only work if there's a value inside of res 0 and that value is not none however because we mapped over an option
an option is returned which is never really used so it's not a good idea this merely map over an option whenever you aren't going to return a value that you need to use later in this case we were instead use for each
and we see that on the next line where we try to apply the for each method to the res 0 and for the name value inside of res 0 we print linen yay + name and this case we do not create a new option as a result if we try to do the same higher-order function for each on a nun instance nothing happens
finally we can use for expressions with options as well if I create an instance of an option of a string Martin and I give it to me Martin and I create an instance of another option with the string Jane Jane I can use them as generators inside of the four expression where AM is retrieved from Martin pulling out the string inside of the option and then jay is retrieved from Jane pulling out the string inside of change it then yields a new strength but because we're operating on an option it is an option of a string and thus some Martin is friends with Jane if we have a nun in any of the options that we are operating on inside of her for expression then the for expression short

circuits at that point and returns none
having completed this lesson we should be able to describe how to pattern match on an option
outline how to use higher order functions on an option to avoid no checking and illustrate how to use poor comprehension to work with option
End of transcript. Skip to the start.

JVM Exceptions

They represent runtime failures for various reasons

- NullPointerException (Runtime)
- ClassCastException (Runtime)
- IOException (Checked)
- When one occurs, control is "thrown" back within a thread stack to whomever "catches" it

Catching an Exception

```
def toInt(s: String): Int =
  try {
    s.toInt
  } catch {
    case _: NumberFormatException => 0
  }
```

Idiomatic Scala and Exceptions

In Scala, we do not believe in this approach, as it represents a possible "side effect"

- We want everything in our code to be pure
- When we interact with libraries or services that may fail, we "wrap" the call in a Try to capture the failure

Wrapping a Call in Try

```
scala> import scala.util.{Try, Success, Failure}
import scala.util.{Try, Success, Failure}

scala> Try("100".toInt)
res0: scala.util.Try[Int] = Success(100)

scala> Try("Martin".toInt)
res1: scala.util.Try[Int] =
  Failure(java.lang.NumberFormatException:
    For input string: "Martin")
```

COGNITIVE CLASS.ai

Pattern Matching on Try

```
scala> import scala.util.{Try, Success, Failure}
import scala.util.{Try, Success, Failure}

scala> def makeInt(s: String): Int = Try(s.toInt) match {
     | case Success(n) => n
     | case Failure(_) => 0
     | }
makeInt: (s: String)Int

scala> makeInt("35")
res2: Int = 35

scala> makeInt("James")
res3: Int = 0
```

Higher Order Functions and Try

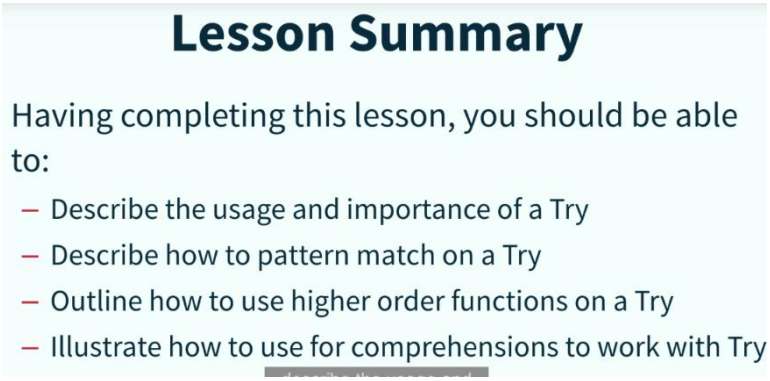
```
scala> import scala.util._
import scala.util._

scala> def getScala: Try[String] = Success("Scala")
getScala: scala.util.Try[String]

scala> val scala = getScala
scala: scala.util.Try[String] = Success(Scala)

scala> scala.map(s => s.reverse)
res0: scala.util.Try[String] = Success(alacS)
```

Higher Order Functions and Try

```
scala> import scala.util._
import scala.util._

scala> def getOuch: Try[String] =
  Failure(new Exception("Ouch"))
getOuch: scala.util.Try[String]

scala> val ouch = getOuch
ouch: scala.util.Try[String] =
  Failure(java.lang.Exception: Ouch)

scala> ouch.map(s => s.reverse)
res0: scala.util.Try[String] =
  Failure(java.lang.Exception: Ouch)
```

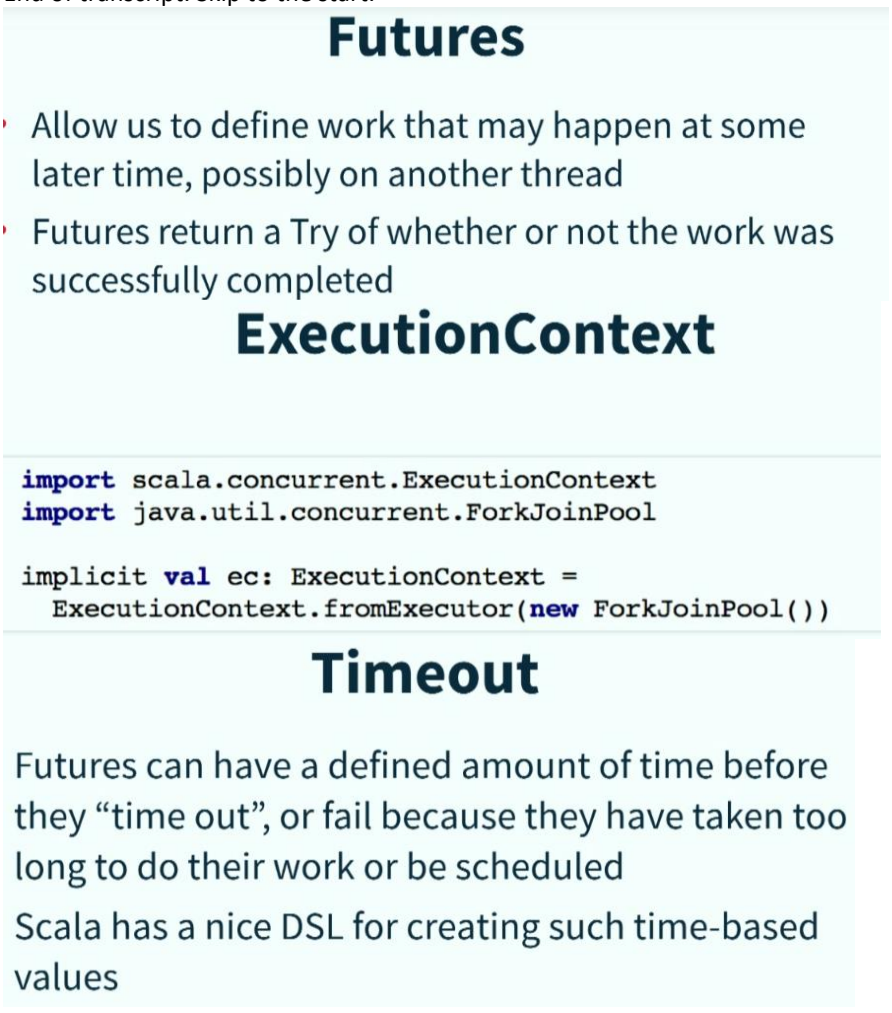
For Expressions and Try

```
scala> Success("Scala").map(_.reverse)
res0: scala.util.Try[String] = Success(alacS)

scala> for {
     |   language <- Success("Scala")
     |   behavior <- Success("rocks")
     | } yield s"$language $behavior"
res1: scala.util.Try[String] = Success(Scala rocks)
```



ript. Skip to the end.
welcome to handling failures after completing this lesson should be able to describe the usage in importance of the tried type described how to pattern match on a try instance outline how to use higher order functions on a try and illustrate how to use for comprehension to work with the tries well the JVM has this concept of exceptions that represent runtime failures for various reasons you could ever know pointer exception of run time if you tried to access a value that didn't have an instance you can get a class cast exception if you tried to cast a value to a different type that is unacceptable you can also get an IOException if you try to work with any kind of files or data coming from a socket when one occurs control is thrown back within a thread stack to whomever catches it here's an example of how we would wrap work to try and catch something if we pass in a string to this method to end and it is acceptable and has only number of values inside of it then it will work without any problems however if we passed in a string that had letters inside of it and we tried to convert it to an integer using this method it would throw a number format exception in this case we would catch that exception and we would return a zero in Scala we do not believe in this approach as it represents a possible side effect within your code it's something that could happen that isn't documented in the name of the method or in the work you're doing we want everything in our code to be pure purely functional and when we interact with libraries are services that may fail we wrap the call in a try to capture the failure
first of all I need to import the classes that I need to use to represent the Tri type as well as the conditions of success and failure this is another example of an algebraic datatype where there are only two possible conditions if I try to do it to end
method call on a string that says 100 that is successful and I get a success of 100 just like option this is something that wraps the value as a container
if I try to do a try on the string Martin and convert that to an integer that results in a number format exception and instead of a success I get a failure representing the exception itself without going up the thread stack in order to handle the results of a try I can use pattern matching to try and deconstruct the possible value in this case I have my making method which takes his strength is going to try to do that to int method on the string value and that's going to match on the result if it's successful we then bind to the value and in return the end value in the case of a failure I'm not going to pay attention to the particular kind of failure and I represent that is an underscore whatever the failure is I'm just going to return a SERO and we see that in the next two lines when we try to passing valid and invalid values I could alternatively use higher order functions to interpret the value inside of a try in this case I have a method called get scholar which returns a try of a string and I return the string scholar wrapped as a success when I do

that and I called the value I end up with a success of Scala and then apply trying to map over that value it works returning its success of the transformed strength in this case reversed I could also handle failure in this case I have a method called get out and it also returns a try of its strength but in this case returns a failure of some new exception with the string out inside of it i den call this method and put the value into a BALCO out when I map over the algae and I try to reverse the value the function is not applied to a failure and that allows me to ignore failures if I want to or handle them as special cases because we have higher order functions and try we can also use for expressions
map over these tries in a cleaner way in this case I before where I'm going to return about your success Scala and put that into a bound name called language and also have a success of Roxana
gonna put that into a bad name of behavior and i yield a string representation of those values put together using something called string interpolation in Scala this results in the success of the complete value transformed to be one string alone having completed this lesson you should be able to describe the usage in importance of a try type described how to pattern matching a try
outline how to use higher order functions on a try and illustrate how to use for comprehension to work with tries
End of transcript. Skip to the start.



Timeout

```
scala> import scala.concurrent.duration._
import scala.concurrent.duration._

scala> implicit val timeout = 1 second
timeout: scala.concurrent.duration.FiniteDuration = 1 second
```

Required Imports

```
import scala.concurrent.Future
import scala.concurrent.ExecutionContext
import java.util.concurrent.ForkJoinPool
import scala.util.Failure
import scala.util.Success
import scala.concurrent.duration._
```

Wrapping a Call in a Future

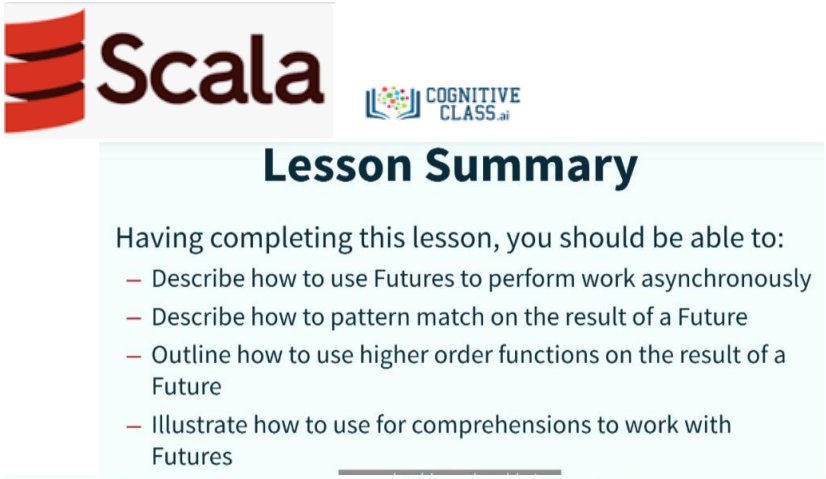
```
val f: Future[Int] = Future {
  inventoryService.getCurrentInventory(1234567L)
}
```

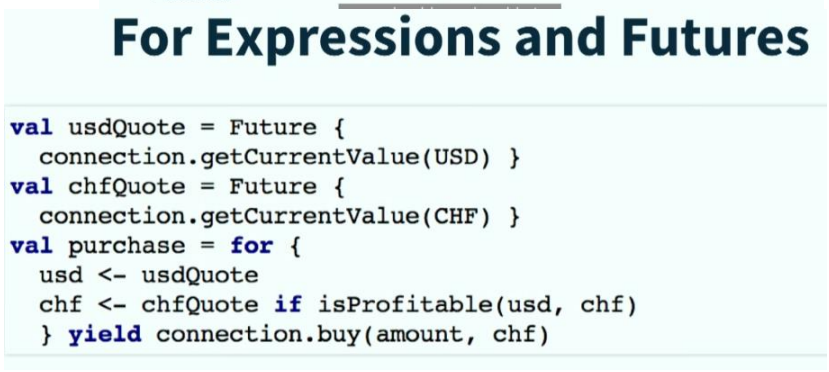
Pattern Matching on Future

```
scala> val f: Future[Int] = Future { 1 + 2 + 3 }
f: scala.concurrent.Future[Int] =
  scala.concurrent.impl.Promise$DefaultPromise@8b96fde

scala> f.onComplete {
     | case Success(i) => println("onComplete Success: " + i)
     | case Failure(f) => println("onComplete Failure: " + f)
     | }

onComplete Success: 6
```

Higher Order Functions and Future

```
scala> val g: Future[Int] = Future { 1 + 2 + 3 }
g: scala.concurrent.Future[Int] = ...

scala> g.map(result => println(Mapping: " + result)
Mapping: 6
```





f transcript. Skip to the end.

welcome to handling futures after completing this lesson we should be able to describe how to use futures to perform work asynchronously describe how to pattern matching the result of a future outline how to use higher order functions on the result of a future and illustrate how to use for comprehension to work with futures futures allow us to define work that may happen at some later time possibly on another thread this is work that will be done asynchronously futures return a try whether or not that work was successfully completed allowing us to leverage what we just saw in the previous session to use a future you must provide a thread pool in the JVM that the future can use to perform the work for example I can use an implicit vow to declare at one time and automatically applied to all usages of a future within a scope so rather than having to provide an execution context to every single one of the future is explicitly define it one time using this implicit Val and then it can be used for all futures inside of that block of code here's an example of how to create an execution context including the import still need we define an implicit Val execution context and we use the job of Fort joined pooled created this will spread work automatically across all of the cores available to our JVM at runtime futures also have a defined time out before they will stop and say that they were not able to do their work by default this is three seconds however you have the ability to override then using a very nice DSL for creating such time base values we import the Skylark concurrent duration . underscore package and all the values inside of it and using that we then say that we're going to use an implicit Val about time out and give it a value of 1 second allowing us to use every future inside of the block where this is defined with a timeout up one second the imports that are required to use futures include this list here this is Khaliq and current future and the execution context and as well the port joined pool that we're going to use to leverage all the course on the box we also have to have the Tri algebraic datatype values for failure and success in scope to create a future you merely define a value that you want to have as a future of some type like an integer or a string or a customer and in this case we're going to get the current inventory number how many items we have a specific book or something like that we're going

to wrap it in a future and call our inventory service saying get us the current inventory number for product with a SKU number of 1234567 we can then pattern match on the results of a future in this case I'm very simple future which is going to return the computation of adding one two and three together if I have a successful completion when I call on complete I'm using callbacks to define behavior such as depression out that I successfully did this work if I had a failure i then say that I had a failure and show what the kind of exception was in this case it successfully completed in shows the value of six I can also use higher order functions so if I can find my future to calculate the value of adding one two and three together I can map over the future and applied this behavior at the time the future completes this behavior is not run immediately is instead defining the behavior that will be applied once that future is completed and in this case the execution context provides the capability and prints out the appropriate value because the future successfully completed however I might have a failure in this case remember the time out that I said before I'm going to make this thread sleep for four thousand milliseconds and then I'm going to try and return value 5 whenever I do that and I try to map over the future it's not going to work it's not going to apply this function because of future returns a failure and you see no result we can also use for expression on futures just because they support the higher-order functions here's an example of where I'm going to call two different services asynchronously first I'm going to ask for the current value of something and U S dollars second I'm going to ask you that value in Swiss francs these are going to take place
asynchronously on other threads and then I can say I have a purchase if I get a quote of a value in USA dollars and I get a quote of a value in Swiss francs but only if the call to is profitable return true in that case I will yield a connection . why we're making the purchase of that amount in Swiss francs having completed this lesson you should now be able to describe how to use futures to perform work asynchronously describe how to pattern matching the result of a future and outline how to use higher order functions on the result of a future is also allows us to use for comprehension and at this point you should be aware of some certain idiomatic usage is of container values in Scala pattern matching higher-order functions and for comprehension
End of transcript. Skip to the start.