

Scala Syntax and Lab Tasks

We aren't introducing any new concepts of functional programming here. My goal is just to introduce the Scala language and its syntax.

Here are some Resources that you can use as a reference for you

- Scala Facts sheet and Reference guide (Check Canvas)
- [Live book- Functional Programming in Scala](#)
- [Scala-Book](#)
- [Scala Cheat sheet](#)
- [Tutorials Point-Scala Tutorial](#)

The document outline:

The document will include some examples on how to write small programs using Scala syntax

By the end of this lab you should be able to

- Write small program using basics of Scala syntax
- Run code on Scala interpreter
- Use IntelliJ IDE to write small projects using Scala objects
- Use Scala Worksheet to write programs and test it
- Introduce to Higher order functions in Scala

The lab contains 4 tasks to complete and submit on Canvas by Friday March 20th at 11:59 PM

For your most benefits, it is recommended to follow the document in order, and complete the tasks as following:

- **Task 1:** Page 5
- **Task 2:** Page 7
- **Task 3:** Page 8
- **Task 4:** Page 11

First program on Command Line (Using Scala Interpreter)

Assuming you already completed the installation guide available on Canvas

- Now, when you get in the command prompt, type scala:

```
→ ~ scala
Welcome to Scala 2.13.1 (OpenJDK 64-Bit Server VM, Java 13.0.2).
Type in expressions for evaluation. Or try :help.

scala>
```

- Type in expressions for evaluation. Or try :help.

1. scala>

- This will give you the Scala prompt. Here, you can type in expressions to evaluate:

```
scala> println("Hello")
Hello

scala> 2+3
res1: Int = 5
```

- If, however, you want to run your program as a script, **save it as a file with a .scala extension**, and move to that location. Here's the file to use:

```
1. object hello extends App{
2.   println("Hello")
3. }
```

- First, let's quit Scala in the command prompt by pressing Ctrl+C and then pressing y, and then Enter:

1. scala> Terminate batch **job** (Y/N)? y

C:\Users\lifei>

Now, let's get to the Desktop, since that is where we saved the file:

C:\Users\lifei>cd Desktop

Then, to compile the script, run the following command:

C:\Users\lifei\Desktop>scalac hello.scala

Next, to run the script, do the following:

C:\Users\lifei\Desktop>scala hello

Hello

C:\Users\lifei\Desktop>

There you go! All set for your journey with scala

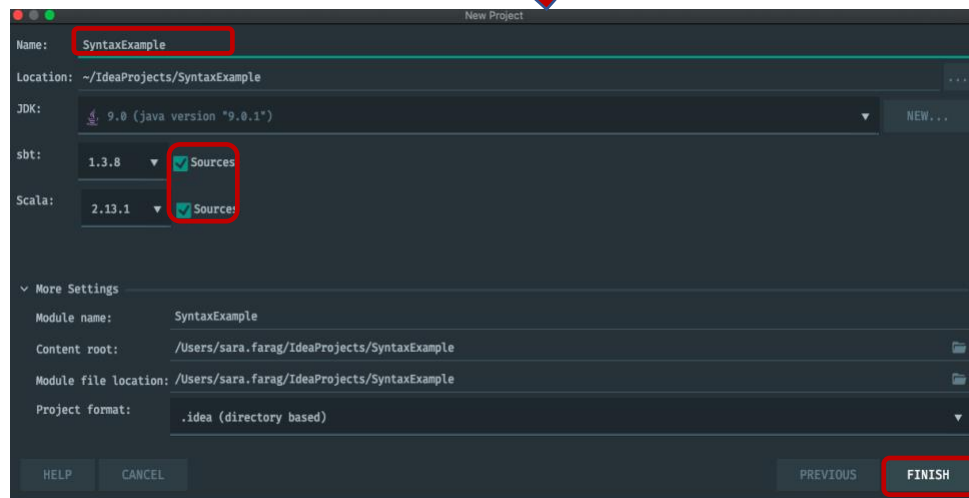
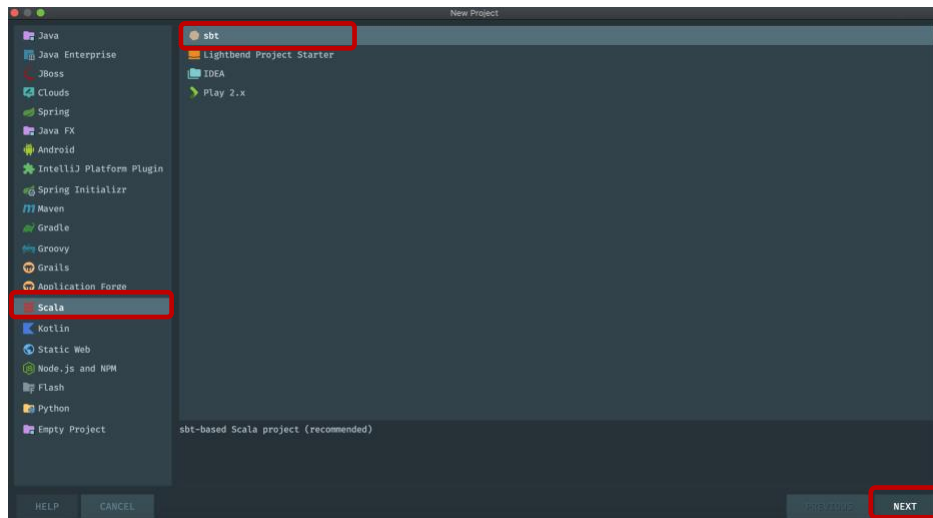
Using IntelliJ IDE

Assuming you already completed the installation guide available on Canvas

- Scala the semicolon is optional. Instead it reads a line of code until it encounters an end line symbol.

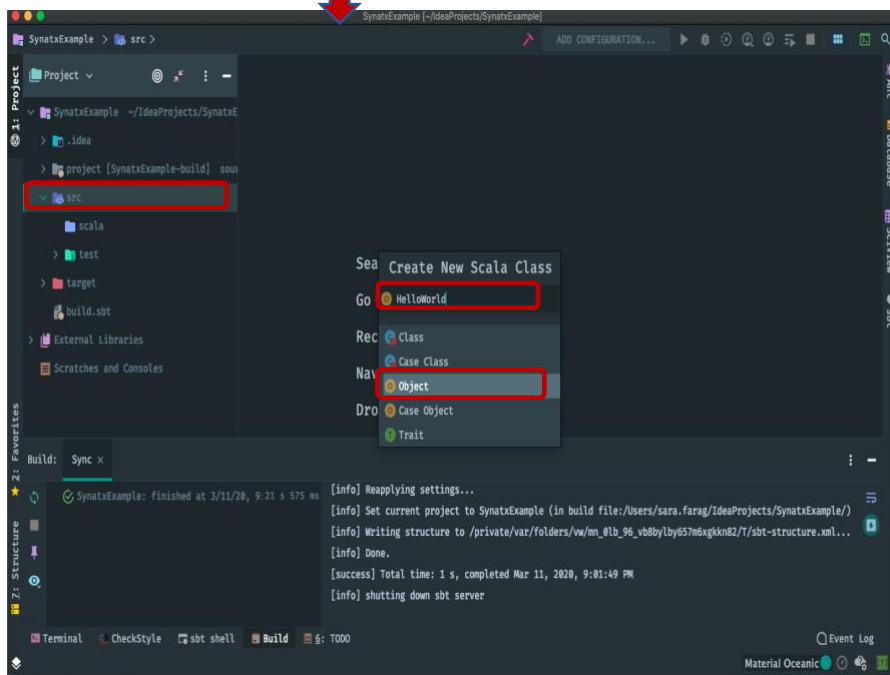
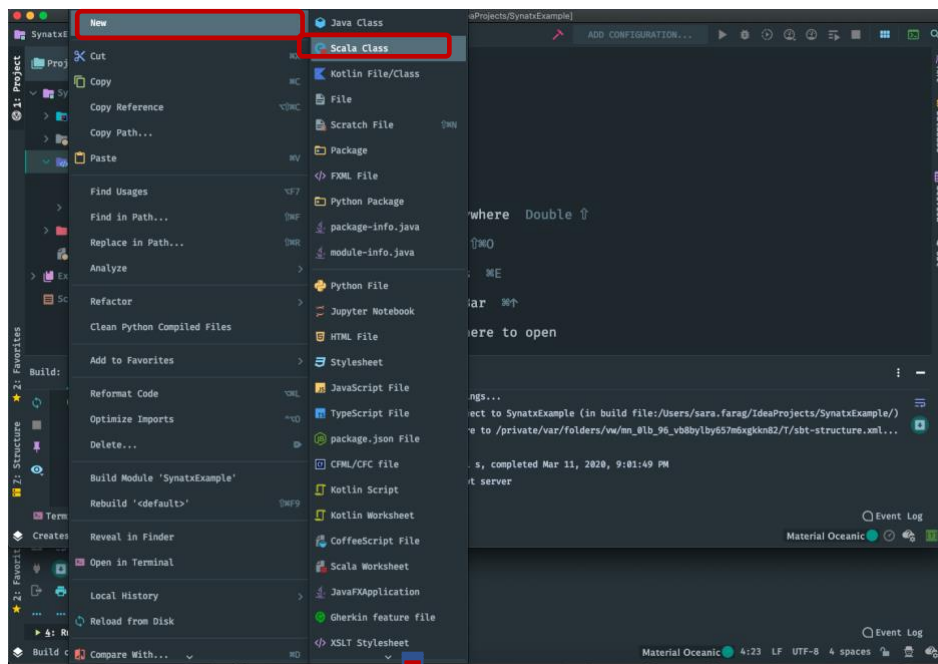
To create a project in IntelliJ

1. Open IntelliJ IDE → Create new Scala Project → Call it SyntaxExample



2. Now you right click on src folder → New → Scala Class.
If you can't see the Scala Class option make sure to check the Add New Scala Class Option – Not Available.pdf (Available on Canvas) to know how to fix this issue
3. Choose object → Give it a name HelloWorld

// Scala has no equivalent to Java's *static* keyword, and an *object* is often used in Scala where you might use a class with static members in Java.



Let's have a look on our first code example and get to learn about the syntax

On line one, we declare an object called HelloWorld

open curly bracket to indicate the start of our code.

define main
method "def
word is
lower case"

Print Statement

Multiple line
comment and // for
single line

In two lines
this is Ok

Call function1

```
1  object HelloWorld {
2  def main(args: Array[String]): Unit = {
3      //this is my first Scala program
4      println("Hello, world!")
5      /*
6       * declare variables
7       */
8      var firstName:String = "Peggy"
9      val age:Int = 21
10     println(firstName + ", is \n"+age+" years old")
11     var sum = 10 +
12         25
13     println("Sum is = " +sum )
14     def function1:Unit = {
15         println("This is function1")
16     }
17     function1
18 }
19
20 }
```

Unit return type: equivalent to void in java

The data type and the variable name
reversed compared to a Java
method, there's no return type in
front of the word main

I have **var** and **val**. They both declare
variables; the difference will be that **var**
allows the variable *to change*. It is a
mutable variable. **Val** is *immutable*.

Remember, *the colon* after the variable
name indicates *the return type*, but it is
optionl

Task 1:

- Try the SyntaxExample yourself, run the project, record the output from the program.
- Submit HelloWorld.scala and text file that includes the output

Note:

if you get an error that there is no main class, go into the Run Configuration and make sure that the main class specified says Hello world.

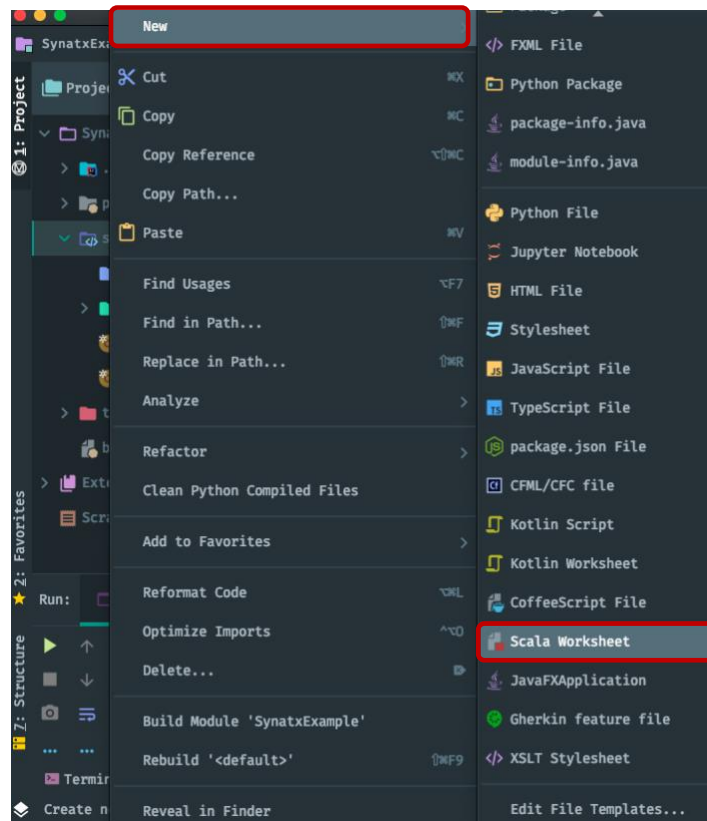
Scala Worksheet

A really nice feature of Scala is that it provides several ways to code and test our programs. One option is using an IDE where we can create what they call a Scala worksheet.

A worksheet in the Scala IDE automatically invokes the Scala interpreter and is going to provide us with realtime feedback.

Let's get started. We're going to start by creating a new project.

I'll go to **File > New > Scala Project** and I'm going to call my project **WorksheetExample**. Then **click Finish**. Now if I expand worksheet example you'll see an src package. **Right-click the src → New**, and all the way down towards the bottom you should have **Scala worksheet**. I'm going to call this worksheet1 and click **Finish**.



Why Scala Worksheet?

Because what's nice about a Scala worksheet is that as we type in statements inside of our worksheet on the right-hand side it prints out the output. Every time we save it, it will automatically compile, call the interpreter to make sure everything's good and print out the output on the right-hand side. So, the `println` statement is printing out the words Welcome to the Scala worksheet.

This format is great for testing our code in an independent environment then we can use this code inside of a Scala program.

Here is your first example to try worksheets:

```
1 object worksheet1 {
2   println("Welcome to the Scala worksheet")    //> Welcome to the Scala worksheet
3   var x = 5                                    //> x : Int = 5
4   val str = "Scala Rocks!"                    //> str : String = Scala Rocks!
5   var odds = List(1,3,5,7,9)                  //> odds : List[Int] = List(1, 3, 5, 7, 9)
6   2 to 10                                     //> res0: scala.collection.immutable.Range.Inclusive = Range(2, 3, 4,
7   //| , 9, 10)
8   var y = true                                //> y : Boolean = true
9   val pi = 3.14159                            //> pi : Double = 3.14159
10  def add(a:Int, b:Int):Int = a + b            //> add: (a: Int, b: Int)Int
11  add(4,9)                                     //> res1: Int = 13
12 }
```


Task 2:

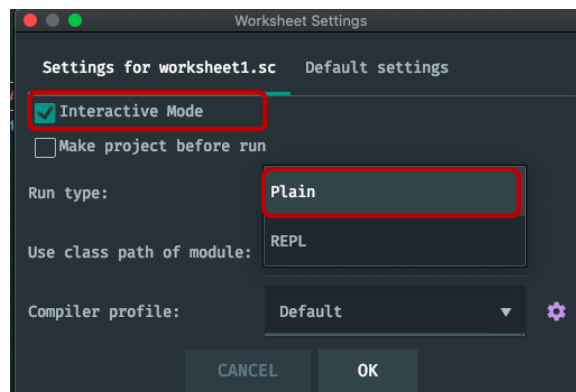
- Try worksheet1 example yourself.
- Submit worksheet.sc

After you complete your task, to evaluate you need to click on the run green arrow.
This should be the output

```
1 object worksheet1 {
2   println("Welcome to the Scala worksheet") //> Welcome to the Scala worksheet
3   var x = 5                                //> x : Int = 5
4   val str = "Scala Rocks!"                 //> str : String = Scala Rocks!
5   var odds = List(1,3,5,7,9)               //> odds : List[Int] = List(1, 3, 5, 7, 9)
6   2 to 10                                 //> res0: scala.collection.immutable.Range.Inclusive = Range 2 to 10
7   //| , 9, 10)
8   var y = true                             //> y : Boolean = true
9   val pi = 3.14159                         //> pi : Double = 3.14159
10  def add(a:Int, b:Int):Int = a + b         //> add: (a: Int, b: Int) => Int
11  add(4,9)                                 //> res1: Int = 13
12 }
```

Welcome to the Scala worksheet res0: Unit = ()
x: Int = 5
str: String = Scala Rocks!
odds: List[Int] = List(1, 3, 5, 7, 9)
res1: scala.collection.immutable.Range.Inclusive = Range 2 to 10
y: Boolean = true
pi: Double = 3.14159
add: add[(val a: Int, val b: Int) => Int]
res2: Int = 13

If you don't have the same as please change the worksheet settings to be as highlighted below
Click in the  icon and then



Scala Challenge

Are you ready to try some Scala programming?

Task 3

- Create a new Project in your Scala IDE,
- Create a new Worksheet in this project and call it addsubtract.
- add two functions, call one add and call the other one subtracts.
- Each function will take two arguments and each one shall return one value.
 - Add will return the sum of two numbers and
 - subtract will return the difference of two numbers.
- Make sure you test it out by adding some statements that call your new functions.
 - add, the statements add with the values 10, 20
 - and call the subtract function, given that the values 10, 20.
 - add some of your own test cases too, maybe add 10 and 10 and subtract 10 minus 10 to see if you get zero.
- Submit **addsubtract.sc**

Note: When you're coding you won't actually see anything happen until you save the worksheet, or Run

You can refer to the Scala reference guide.pdf or any of the online resources that I suggested for you to help with the syntax

Variables in Scala

- Variables are named memory locations that are used to store values.
- When you declare a variable memory is reserved to hold the value of this variable.
 - Depending on the variable data type, whether it's an integer or a string, the compiler decides the amount of memory to allocate for each value.
- Variables are defined with two keywords, either **val** or **var**.
- A variable, defined with the keyword **val**, stands *for a constant value* and cannot be changed.
- If you use the keyword **var**, that means *that variable can change the data that it holds later on in your program*.
- A variable can be declared *with or without a data type*. The compiler will infer the data type if it is omitted. If you omit the data type you must initialize that value when you declare it.
- The syntax for declaring a variable is the keyword val or var, depending on how you want to use that variable

Val (or var) < name> : data Type

- The colon datatype is optional if you initialize the variable at the time that you define it.
- Scala *naming standards* usually have *variables starting with a lowercase letter*. *The names can include special symbols* but *the underscore is strongly discouraged* since it also is *used as a wildcard symbol in Scala*.
- All *variable data types are classes*, so we have an int class, a double class, a string class.
 - Because all variable data types are classes that means any time we declare a variable that has a type of one of these classes it's going to be an object, so we might have a variable x that's of type int, x is an object of type integer, so this means that all variables are instances of classes and they're all called objects.
 - Variables are used to represent fields, method parameters and local variables. Fields are variables that belong to an object, these fields are automatically accessible to every method and function in the object.
 - Method parameters or arguments are used as a connection between the calling program and the method. They allow us to pass values between the two.
- Local variables are variables declared inside a method. Local variables are only accessible from inside the method, but the objects you create may share the variable by returning them from a method

```
1  object variables {
2      println("Welcome to the Scala worksheet")    //> Welcome to the Scala worksheet
3      var greeting, message:String = null         //> greeting : String = null
4      //| message : String = null
5
6      val list1 = List(1, 2, 3)                    //> list1 : List[Int] = List(1, 2, 3)
7      val list2 = List(1, 2, 3.0)                  //> list2 : List[Double] = List(1.0, 2.0, 3.0)
8      val list3 = List(1, 2, true)                 //> list3 : List[AnyVal] = List(1, 2, true)
9      val list4 = List(1, 2, true, "Peggy")         //> list4 : List[Any] = List(1, 2, true, Peggy)
10
11     var a = 10.toString()                        //> a : String = 10
12     var b = 10.to(20)                             //> b : scala.collection.immutable.Range.Inclusive = Range(10, 11, 12, 13, 14,
13     //| 15, 16, 17, 18, 19, 20)
14     var c = 10.*(11)                             //> c : Int = 21
15     var d = 10 to 20                             //> d : scala.collection.immutable.Range.Inclusive = Range(10, 11, 12, 13, 14,
16     //| 15, 16, 17, 18, 19, 20)
17     var e = 10 + 11                               //> e : Int = 21
18     var f = 97.toChar                             //> f : Char = a
19     var g = 85.97.toInt                           //> g : Int = 85
20
21     a += " ten"
22     println(a)                                    //> 10 ten
23 }
```

Here is a worksheet example to show some of the different variables that you can use in Scala ([try it yourself](#))

Loops in Scala

- Syntax of while loop: while (expr is true) {...}
- Syntax of do/while loop: do {...} while (expr is true)
- Syntax for a for loop: for (i <- 1 to 10){...}
- Syntax for a for comprehension: for(I <- 1 to 10) yield.....

Here is a worksheet to show some of the loops syntax (try it yourself)

```
1  object loopsPractice {
2      var x = 10                                ///< x : Int = 10
3      /*while(x ≥ 0) {
4          println(x)
5          x -= 1 //same as x = x-1
6      }
7      do {
8          println(x)
9          x -= 1
10     } while(x ≥ 0 )
11     for(x ← 10 to 0 by -1) {
12         println(x)
13     }
14     println("BlastOff!")
15     var y = for(num ← 1 to 10) yield num + 1 */
16     var word = "Monday"                        ///< word : String = Monday
17     for(x ← 0 until word.length) { println(word(x))}
18     ///< M
19     ///< o
20     ///< n
21     ///< d
22     ///< a
23     ///< y
24
25 }
```

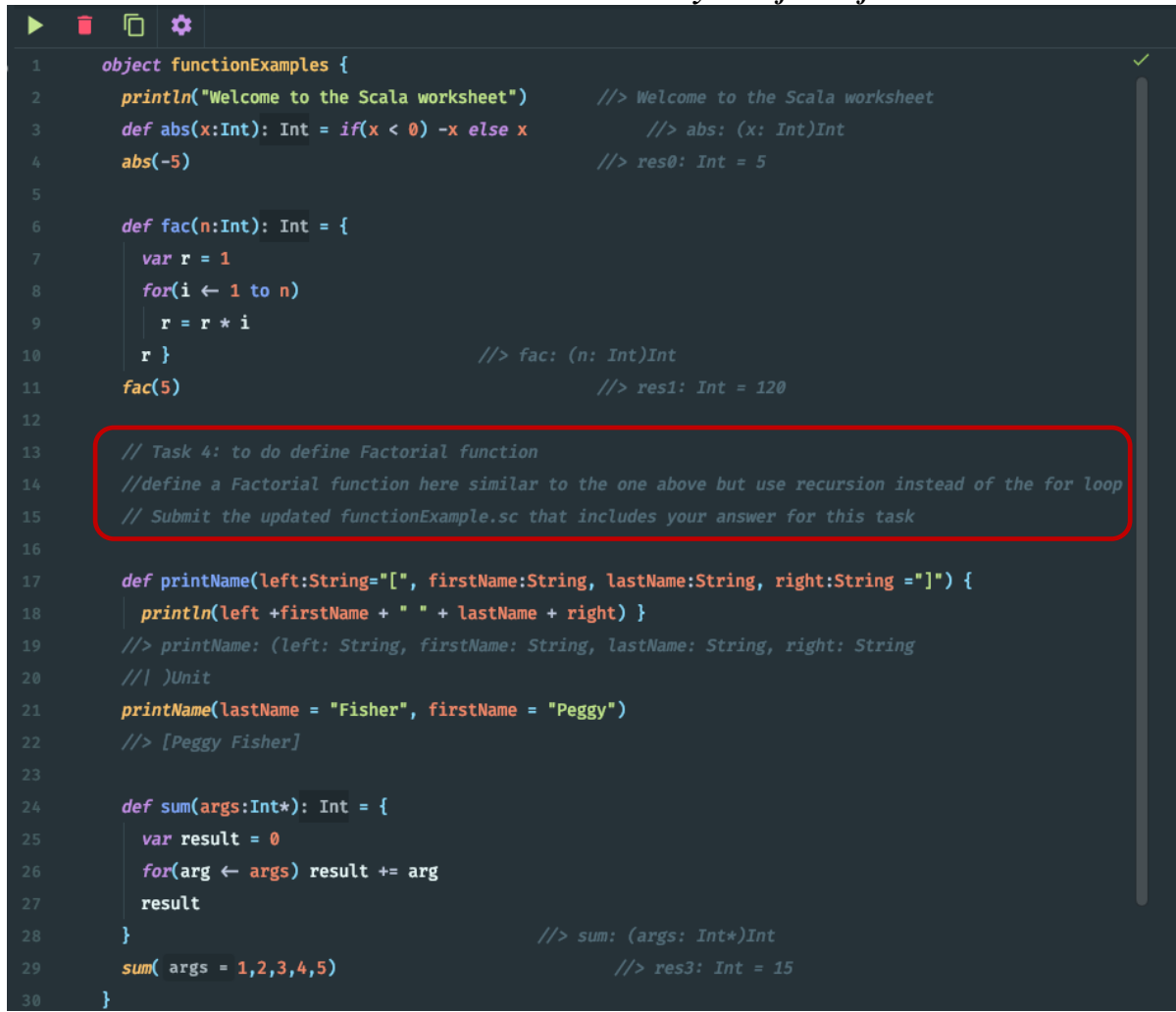
// Some of the code is commented to test one type of the loops at a time in your worksheet.

Functions in Scala

You define function using

Refer to our first example in the document as well as the Scala Reference Guide.pdf as well as any other resources for the syntax.

Here is a worksheet to show how you define a function



```
1  object functionExamples {
2      println("Welcome to the Scala worksheet")    //> Welcome to the Scala worksheet
3      def abs(x:Int): Int = if(x < 0) -x else x      //> abs: (x: Int)Int
4      abs(-5)                                       //> res0: Int = 5
5
6      def fac(n:Int): Int = {
7          var r = 1
8          for(i ← 1 to n)
9              r = r * i
10         r }                                       //> fac: (n: Int)Int
11         fac(5)                                   //> res1: Int = 120
12
13         // Task 4: to do define Factorial function
14         //define a Factorial function here similar to the one above but use recursion instead of the for loop
15         // Submit the updated functionExample.sc that includes your answer for this task
16
17         def printName(left:String="[" , firstName:String, lastName:String, right:String ="]") {
18             println(left +firstName + " " + lastName + right) }
19         //> printName: (left: String, firstName: String, lastName: String, right: String
20         //| )Unit
21         printName(lastName = "Fisher", firstName = "Peggy")
22         //> [Peggy Fisher]
23
24         def sum(args:Int*): Int = {
25             var result = 0
26             for(arg ← args) result += arg
27             result
28         }                                       //> sum: (args: Int*)Int
29         sum( args = 1,2,3,4,5)                 //> res3: Int = 15
30     }
```

Task 4: to do Notice the Red Square in the figure above

- Define a Factorial function here similar to the one in the worksheet but use recursion instead of the for loop
- Submit the updated functionExample.sc that includes your answer for this task

Higher Order function

Here is an example for how higherOrder function looks like in Scala

In Scala, we not only have regular functions, we also have something called higher-order functions.

Remember, Scala is a functional programming language.

A higher-order function is

- a function that takes another function as a parameter,
- or returns a function.

This is in contrast to a first-order function, which takes only data items in as parameters.

I think they best way to explain this is to look at some examples, so check the Scala worksheet below in your IDE., but first try to read [Closures](#) in Scala and [pattern matching](#) that explains the symbol =>

Example:

It creates a new Scala worksheet, called Order FNS “Higher-order functions”. (See Screenshot below for the code)

So, the example that I want to do, is where one function takes another function as input,

Starting from line 4: I am to create a function that is going to actually double any number that's passed into it. I will call it double

I'll use i and then what happens is this function is going to take i and it's going to multiply it by two, so it's just going to double the value.

Line 5: Now I want to declare my higher-order function, so we're going to call this higherOrder, and it is a function that takes in an integer value, which will be the value that I want to be doubled, and it takes in another function. I have to give that other function a name, so I'm going to call it y, and again this other function is going to be any function, which is what's really different here, any function that takes in an Int and returns an Int.

Line 7: what's going to happen is when I call higherOrder I want to invoke whatever function came in as the parameter y, and I'm going to use the x as my variable. So, I'm going to say I want to take y and apply that function on the value x.

line 16, we declare variable y of type Int, which is equal to 5, we have, right below it, the multiplier function which takes an Int, returns an Int, and finally we call the multiplier function with an integer, it takes that value and multiplies it by the value outside the function which is 5.

```

1  object higherOrderFns {
2      println("Welcome to the Scala worksheet")    //> Welcome to the Scala wor
3
4      val double = (i:Int) => i * 2                //> double : Int => Int = <
5      def higherOrder(x:Int, y:Int=>Int): Int = y(x)    //> higherOrder: (x: In
6
7      higherOrder(6, double)                       //> res0: Int = 12
8
9      val triple = (i:Int) => i * 3                 //> triple : Int => Int = <
10     higherOrder(6, triple)                        //> res1: Int = 18
11
12     def sayHello: String => String = (name:String) => {"Hello" + " " + name}
13     //> sayHello: => String => String
14     var message = sayHello("Peggy")              //> message : String = Hell
15
16     var y = 5                                     //> y : Int = 5
17     val multiplier = (x:Int)> x * y                //> multiplier : Int => Int
18     multiplier(10)                               //> res2: Int = 50
19 }

```

highOrderFns worksheet examples

Line 12 and 13 are example where one function returns another function

In line 12, we declared a function that takes a parameter, and it returns a function of type String.

We are defining a function that returns another function. We are going to define a function that says sayHello, and it is equal to another function that takes in a string called name, and the second function returns hello plus a space, plus whatever name we gave it. Let's look closely at this. So we have a function called sayHello. sayHello actually is equal to another function, so it returns that function when we call sayHello. So what we're going to start by just first calling sayHello and let's give it a value, we'll give it Peggy, okay? So what happens is when I invoke sayHello, it returns that function that we see up there in line 12, but instead of actually returning the function, the function is then invoked at the same time, it substitutes name with Peggy and it'll say, "Hello Peggy". And now we see result two, the string is equal to Hello Peggy.

In Line 13: We could actually assign the call to sayHello, to a variable so I could even do variable message equals and then you could say call sayHello

Higher-order functions are great because they give us the ability to encapsulate an algorithm inside of a function.