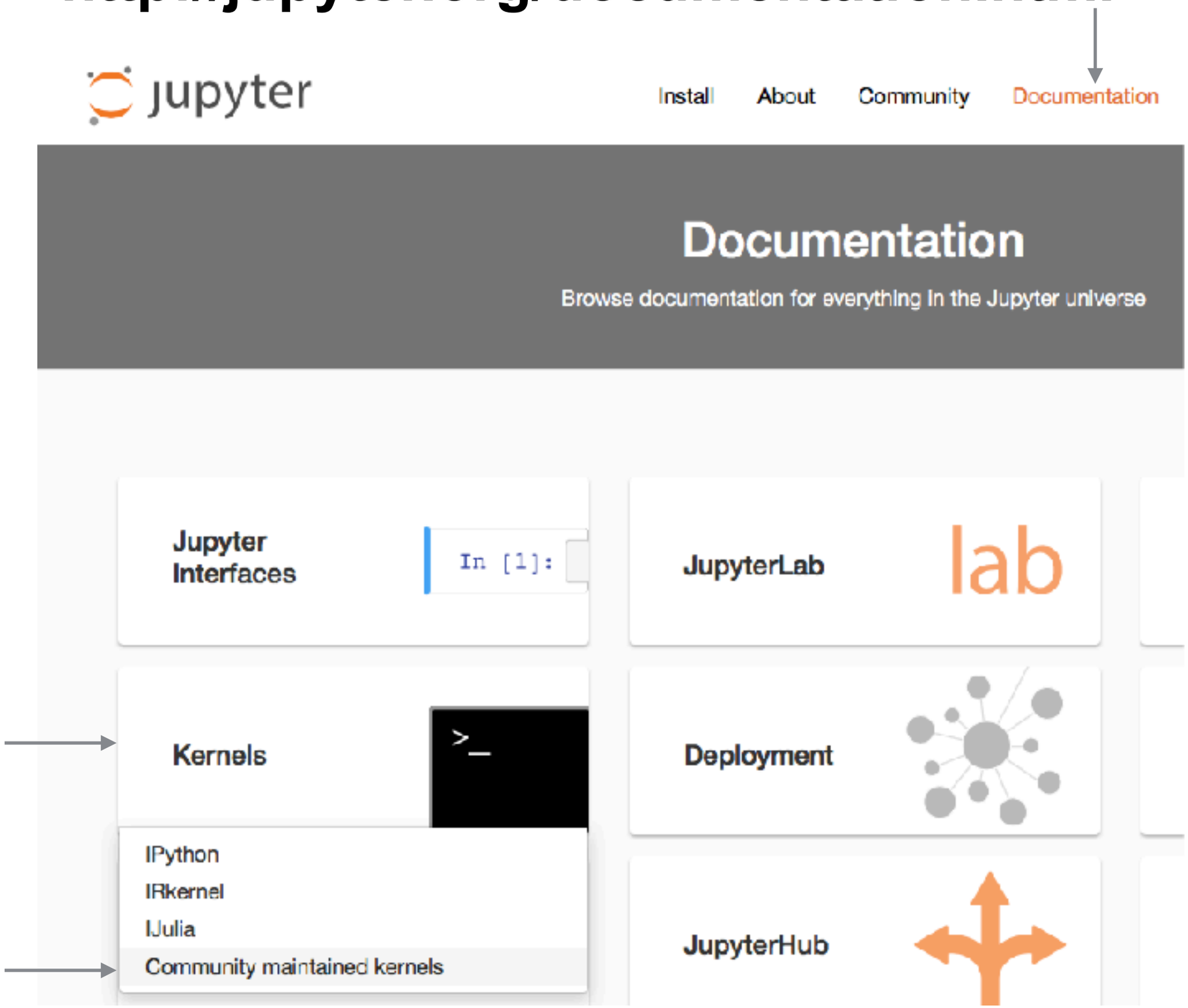
CS 696 Intro to Big Data: Tools and Methods
Fall Semester, 2017
Doc 3 Scala
Aug 30, 2017

Jupyter Notebooks

Mix Text and Code

Can execute code

http://jupyter.org/documentation.html



SageMath	Jupyter 4	Any	many	
pari_jupyter	Jupyter 4	2.8	Cython	
IFSharp	IPython 2.0	F#		Features
gopherlab	Jupyter 4.1, JupyterLab	Go >= 1.6	ZeroMQ (4.x)	examples
Gophernotes	Jupyter 4	Go >= 1.4	zeromq 2.2.x	examples
IGo		Go >= 1.4		
IScala		Scala		
Jupyter-scala	IPython>=3.0	Scala>=2.10		example
IErlang	IPython 2.3	Erlang	rebar	
		Torch 7		

Scala - Command Line

```
[Al pro 13->scala
Welcome to Scala 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_131).
Type in expressions for evaluation. Or try :help.

[scala> 1 + 2
res0: Int = 3

scala>
```

:help

:load

Tab Completion

Scala & Intellij

Intellij

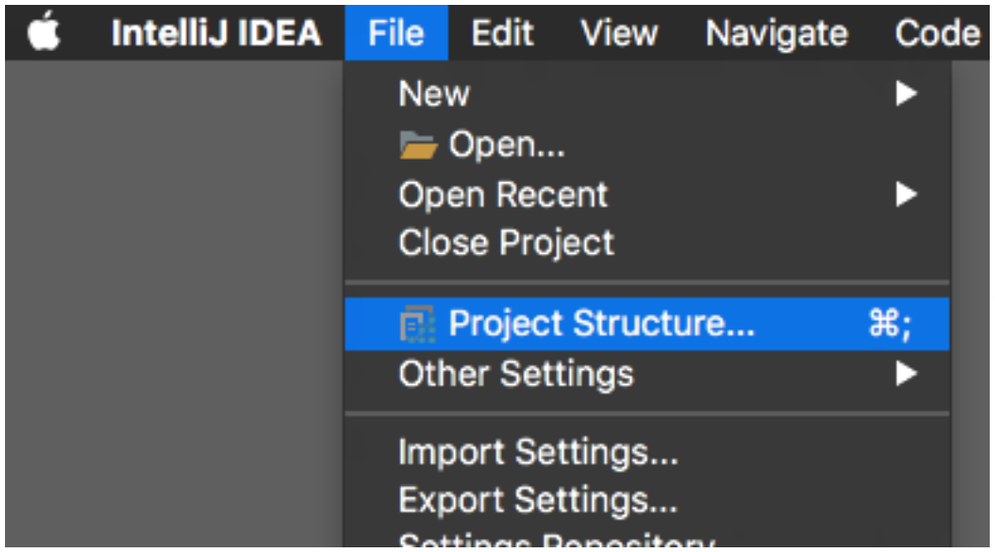
https://www.jetbrains.com/idea/

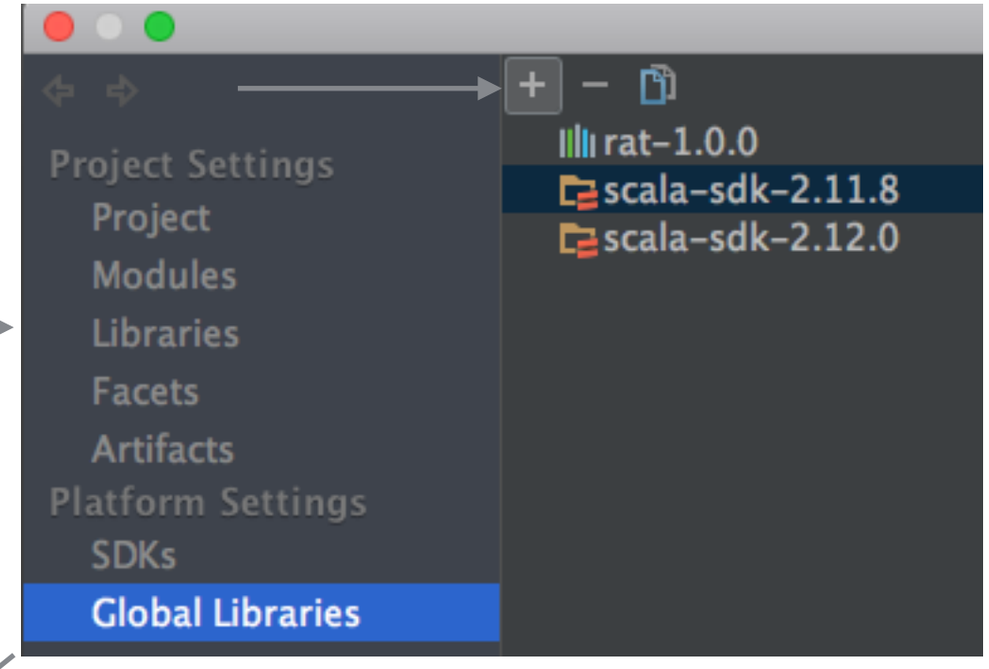
Intellij Plugin

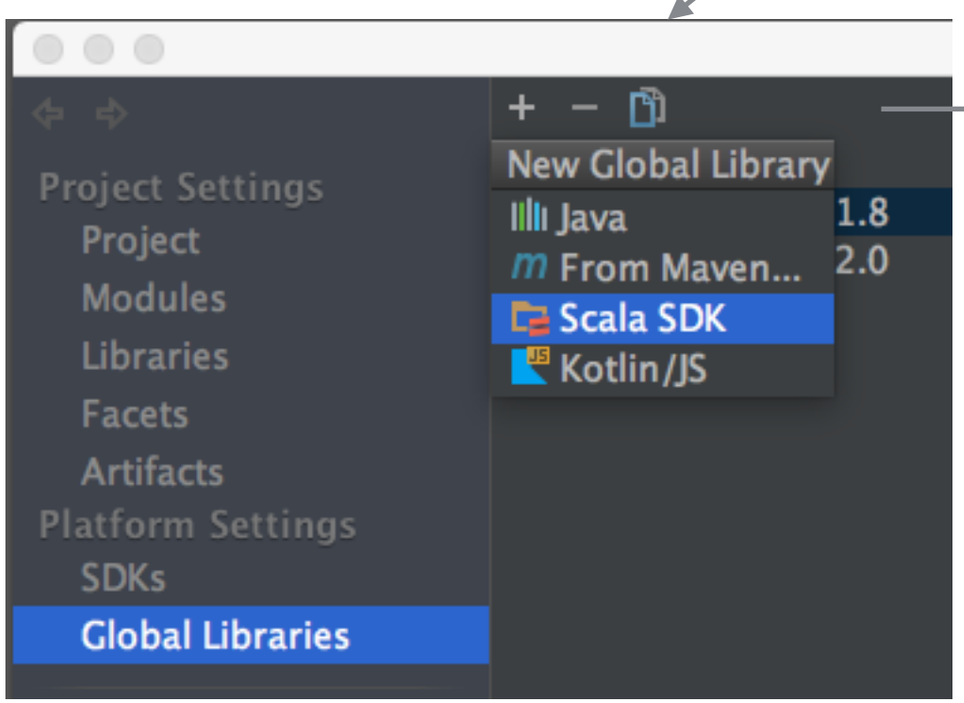
https://www.jetbrains.com/help/idea/enabling-and-disabling-plugins.html

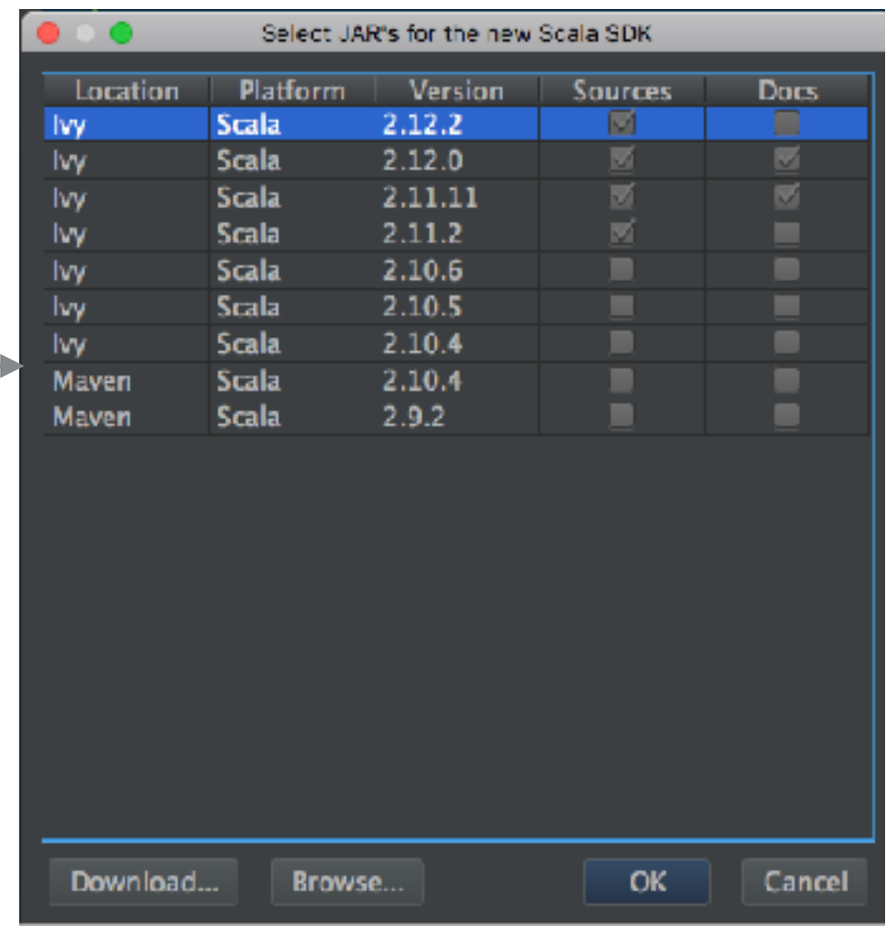
Starting Scala Project

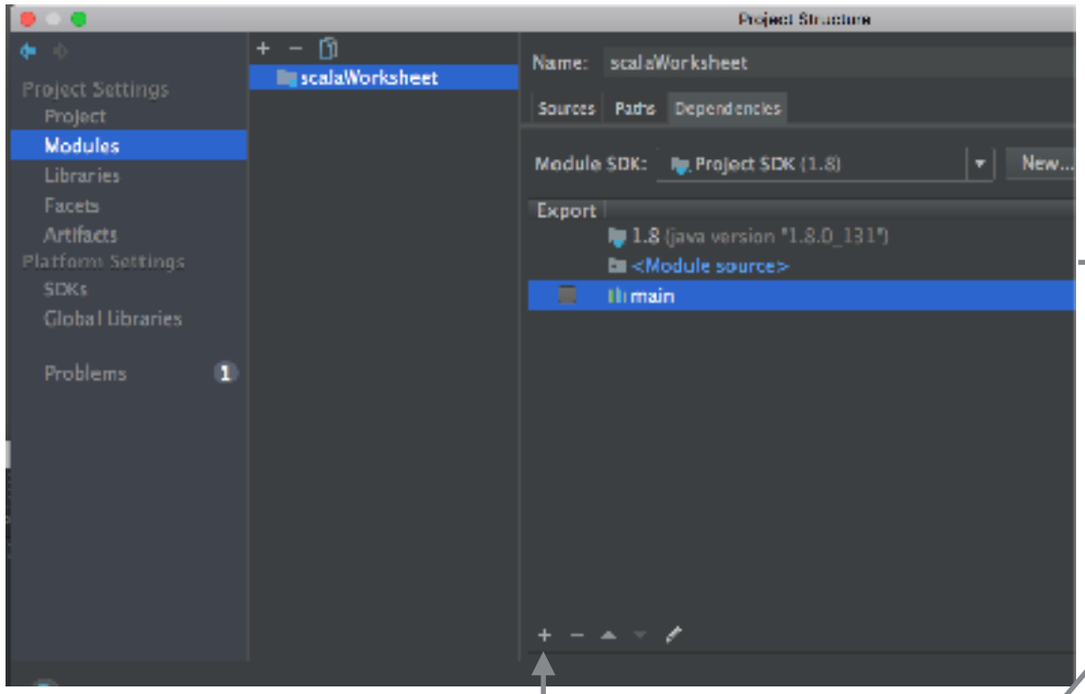
https://www.jetbrains.com/help/idea/creating-and-running-your-scala-application.html

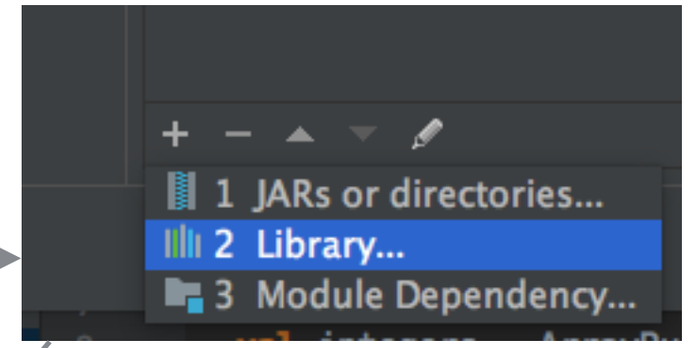

IntelliJ IDEA
File
Edit
View
Navigate
Code
New
Open...
Open Recent
Close Project
Project Structure... ⌘;
Other Settings
Import Settings...
Export Settings...

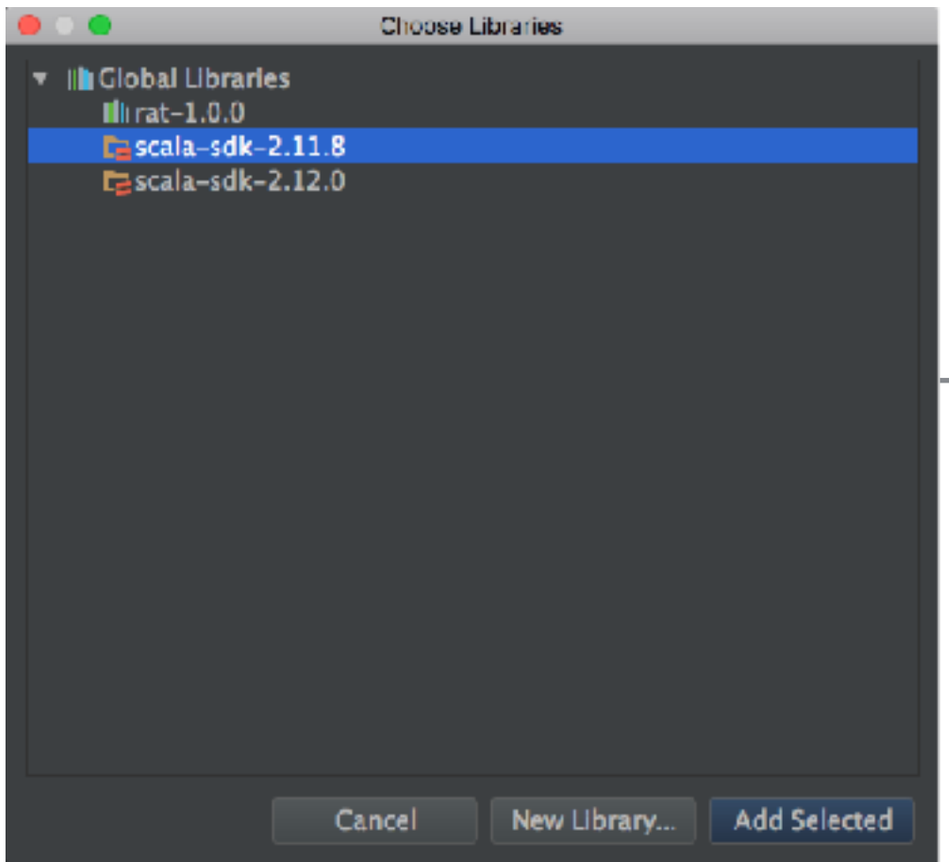

Project Settings
Project
Modules
Libraries
Facets
Artifacts
Platform Settings
SDKs
Global Libraries
rat-1.0.0
scala-sdk-2.11.8
scala-sdk-2.12.0

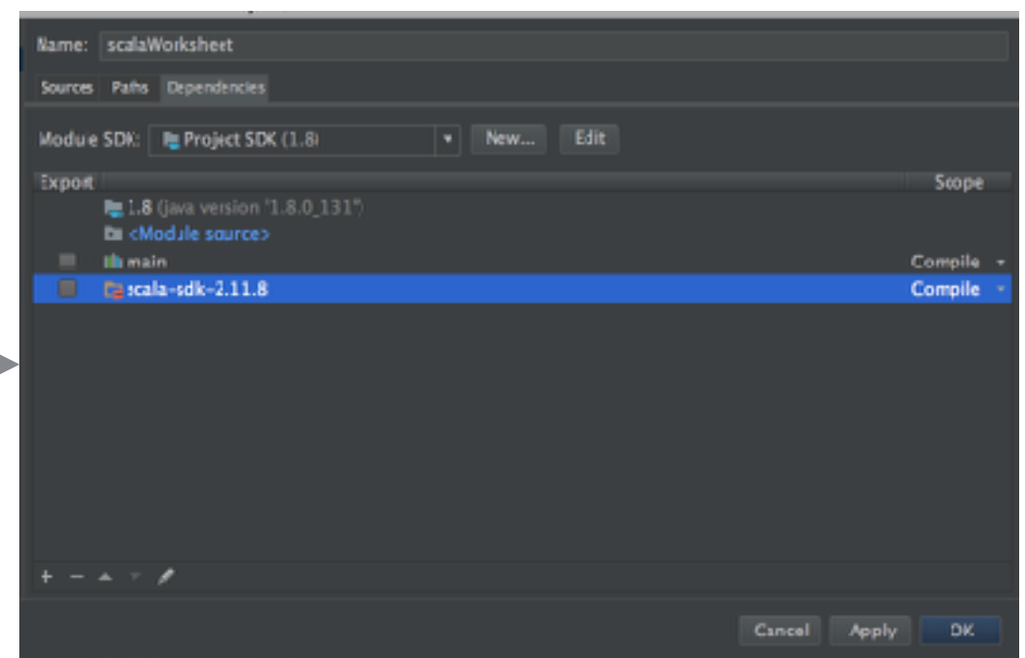

Project Settings
Project
Modules
Libraries
Facets
Artifacts
Platform Settings
SDKs
Global Libraries
New Global Library
Java
From Maven...
Scala SDK
Kotlin/JS
1.8
2.0


Select JAR's for the new Scala SDK
Location Platform Version Sources Docs
Ivy Scala 2.12.2
Ivy Scala 2.12.0
Ivy Scala 2.11.11
Ivy Scala 2.11.2
Ivy Scala 2.10.6
Ivy Scala 2.10.5
Ivy Scala 2.10.4
Maven Scala 2.10.4
Maven Scala 2.9.2
Download...
Browse...
OK
Cancel


Project Structure
Project Settings
Project
Modules
Libraries
Facets
Artifacts
Platform Settings
SDKs
Global Libraries
Problems
1
scalaWorksheet
Name: scalaWorksheet
Sources
Paths
Dependencies
Module SDK: Project SDK (1.8)
New...
Export
1.8 (java version "1.8.0_131")
<Module source>
main


1 JARs or directories...
2 Library...
3 Module Dependency...


Choose Libraries
Global Libraries
rat-1.0.0
scala-sdk-2.11.8
scala-sdk-2.12.0
Cancel
New Library...
Add Selected


Name: scalaWorksheet
Sources
Paths
Dependencies
Module SDK: Project SDK (1.8)
New...
Edit
Export
Scope
1.8 (java version "1.8.0_131")
<Module source>
main
Compile
scala-sdk-2.11.8
Compile
Cancel
Apply
OK

Functional Programming

Use functions to compute values

No side-effect allowed

Functions are values

Immutable data structures

Higher order functions
  map, reduce, filter

Hadoop
  Distributed file system (DFS)
  map & reduce

Spark
  Hadoop DFS
  map, reduce, filter + lot more

You have to be able to solve problems using map & reduce

Scala

http://www.scala-lang.org/

Current Version 2.12.2

Functional Programming
Object-Oriented Programming
Scripting
Really strongly typed
Implicit types
Interpreter & compiler

Runs on top of Java
Compiles to Java byte code
Scala can call Java code
Java can call Scala code

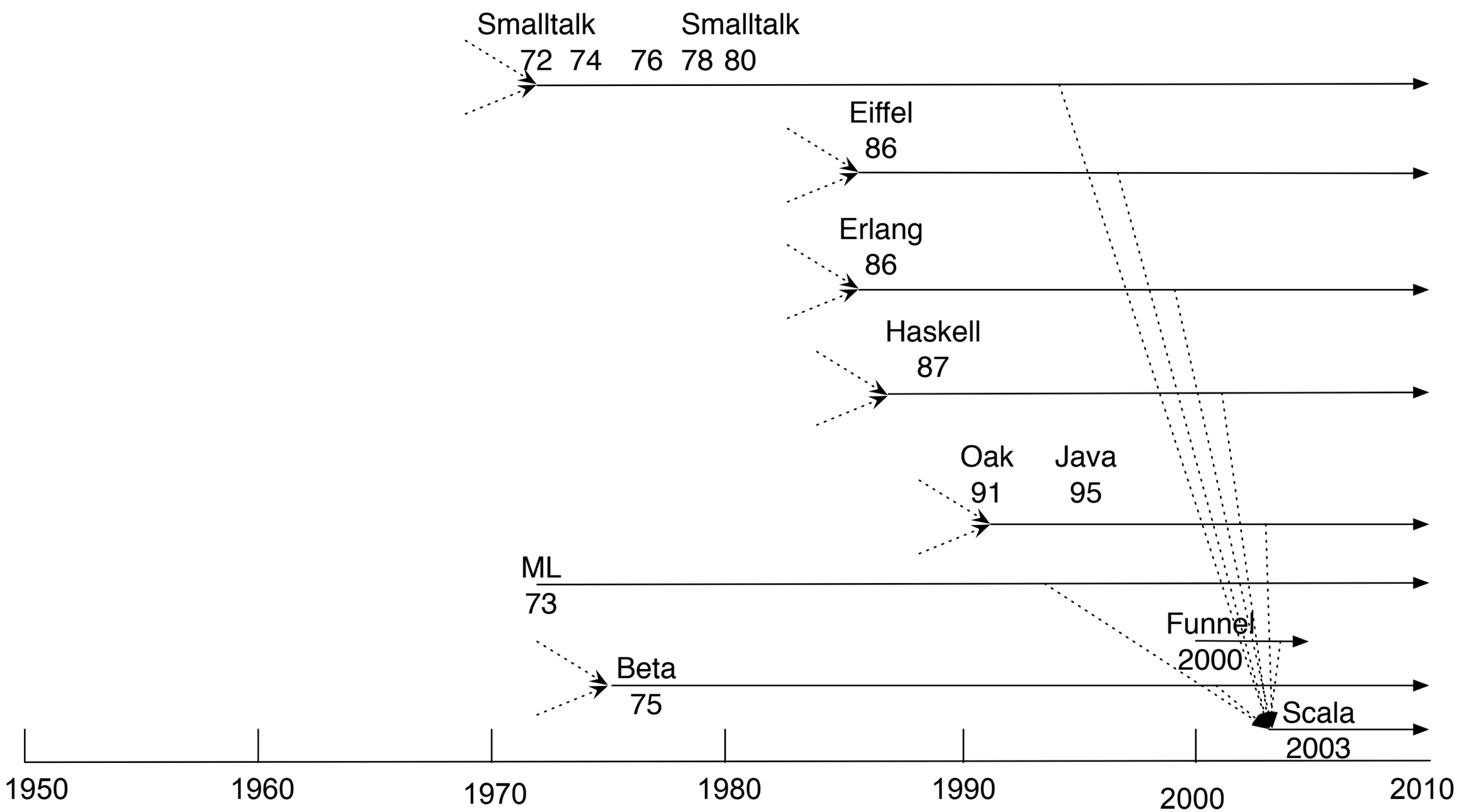
Scala & Spark

Scala 2.12.x supports uses Java 8 lambdas

Spark

- Does not work with Scala 2.12
- Use Scala 2.11

Scala Influences



Basic Types

Boolean
Byte
Char
Double
Float
Int
Long
Short
String

Symbol

BigInt (java.math.BigInteger)
BigDecimal (java.math.BigDecimal)

range is same as Java's types

Literals

```
val hex = 0x5
val oct = 035
val decimal = 35
val long = 35L
var double = 3.5
double = 1.23e2
```

```
val float = 1.234F
val char = 'Q'
val unicode = '\u0044'
val boolean = true
val string = "Hi mom"
val symbol = 'whatIsThis
```

Implicit Conversions

```
scala> var y: Int = 'c'
y: Int = 99

scala> var w:Char = 123
w: Char = {

scala> var x: Long = 23
x: Long = 23
```

```
scala> var w:Char = 123L
<console>:4: error: type mismatch;
 found   : Long(123L)
 required: Char
      var w:Char = 123L
                   ^

scala> var x: Int = 23L
<console>:4: error: type mismatch;
 found   : Long(23L)
 required: Int
      var x: Int = 23L
                   ^
```

Double & RichDouble

Implicit conversion from Double -> RichDouble

So can call RichDouble methods on Double

RichDouble methods are not shown in autocomplete list

```
val x = 2.3

x.getClass()
x.byteValue()
```

Same with Int, Float etc

Type Inference

```
val a: String = "cat"

val b = “cat"        // Type inference

val c: Int = 5 * 2
val d = 5 * 2

val e = 10.2
```

val, var

```
val x = 5 * 2          // Type inference
x = 2              // Error val is read only

var y = 10.2
y = 3.4

var z: Double = y * x
z.toInt                //Base types are objects
```

Any

```
var x: Any = 5
x = "cat"
x = 1.23
x = true
x = 'c'
```

```
var y: Int = 5
y = "cat"       //Compile error
```

Ranges

```
val x = 10

x.to(14)          Range 10 to 14
x to(14)
x to 14

x.to(16,2)        Range 10 to 16 by 2
x to 16 by 2

'a' to ’n'        NumericRange a to n

for (k <- 1 to 7 by 2)
  println(k)
                  1
                  3
                  5
                  7
```

Imports & Blocks

```
val x = 10
val x0 = 5
val y = 6
val y0 = 5
import scala.math.sqrt

val distance = {
  val dx = x - x0;
  val dy = y - y0;
  sqrt(dx * dx + dy * dy)
}
```

```
import scala.math._
```

If

```
var x = 10                                  x: Int = 10

val first = if (x>0) 1 else -1              first: Int = 1

val second = if (x>0) 1 else “false"        second: Any = 1

x = -2

val third = if (x>0) 1                      third: AnyVal = ()

val fourth = if (x<0) 1                     fourth: AnyVal = 1
```

String Interpolation

```
val name = "Sam"
val age = 21

val hello = f"Hello $name, in 6 months you will be ${age + 0.5}%2.1f years old"

print(hello)
```

```
Hello Sam, in 6 months you will be 21.5 years old
```

```
val noF = "Hello $name, in 6 months you will be ${age + 0.5}%2.1f years old"

print(noF)
```

```
"Hello $name, in 6 months you will be ${age + 0.5}%2.1f years old"
```

Loops

```
for (k <- 1 to 3)
  print(k)
```

```
for (k <- 1.to(3))
  print(k)
```

```
var sum = 0
for (ch <- "ab")
  sum += ch
sum
```

195

'a'.toInt + 'b'.toInt 195

```
for (j <- 1 to 3; k <- 1 to 3 if j < k)
  println(j, k)
```

(1,2)
(1,3)
(2,3)

```
for (j <- 1 to 3; k <- 1 to 3 if j < k)
  println(f"j $j k $k")
```

j 1 k 2
j 1 k 3
j 2 k 3

Loops

```
for (j <- 1 to 3; squared = j*j; k <- 1 to squared if k % 2 == 0 )
  println(f"j $j k $k")
```

```
for (j <- 1 to 3;
      squared = j*j;
       k <- 1 to squared if k % 2 == 0 )
  println(f"j $j k $k")
```

```
j 2 k 2
j 2 k 4
j 3 k 2
j 3 k 4
j 3 k 6
j 3 k 8
```

Yield

```
for (j <- 1 to 4 )
    yield j + 2
```

```
val x = for (j <- 1 to 4 ) yield j + 2
```

Vector(3, 4, 5, 6)

Functions

```
def fahrenheitToCelcius(f: Float): Float = {            //Parameters are all val
  (5* (f-32))/9
}
```

```
def fahrenheitToCelcius(f: Float) = {                   // Type inference on return type
  (5 * (f-32))/9
}
```

Special Syntax for One argument

```
def next(x: Int) = {x + 1}
val a = next(1)
val b = next { 1 }
```

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

data.contains(2)

data. contains(2)

data. contains (2)

data contains (2)

data contains 2

data.contains{ 2 }

data . contains{ 2 }

data  contains{ 2 }

data  contains { 2 }
```

Why the Multiple Syntax?

ScalaTest FlatSpec example

```
class ExampleSpec extends FlatSpec with Matchers {

  "A Stack" should "pop values in last-in-first-out order" in {
    val stack = new Stack[Int]
    stack.push(1)
    stack.push(2)
    stack.pop() should be (2)
    stack.pop() should be (1)
  }
}
```

Why the Multiple Syntax?

"A Stack" should "pop values in last-in-first-out order" in { … }

"A Stack”.should(“pop values in last-in-first-out order”).in(…)

Why the Multiple Syntax?

```
"A Stack”.should(“pop values in last-in-first-out order”).in(
    val stack = new Stack[Int]
    stack.push(1)
    stack.push(2)
    stack.pop() should be (2)
    stack.pop() should be (1)
)
```

```
"A Stack”.should(“pop values in last-in-first-out order”).in {
    val stack = new Stack[Int]
    stack.push(1)
    stack.push(2)
    stack.pop() should be (2)
    stack.pop() should be (1)
}
```

Nesting Functions

```
def average(items: List[Int]) = {
   def sum(items: List[Int]) = {
      var sum = 0
      for (item <- items)
         sum += item
      sum
   }

   if (items.length == 0)
      throw new RuntimeException( "empty list")
   sum(items)/items.length
}
```

Returning Functions

```
def addN(n:Int):(Int => Int) = {
  def adder(k:Int):Int = {
    k + n
  }
  adder
}

val add5 = addN(5)
val add2 = addN(2)

add5(3)                    // 8
add2(3)                    // 5
```

Anonymous Functions

```
var next = (x: Int) => x + 1

val previous = (x: Int) => x - 1

next(4)
previous(3)
```

```
def example(test: (Int => Int)) {
   println( test(4))
}

example (previous)
example (next)
```

```
next = x => x + 2
next(4)
```

Anonymous Functions & Types

```
var next = (x: Int) => x + 1
next = x => x + 2


var badNext = x => x + 2        // Compile Error

var okNext1:(Int => Int) = (x) => x + 1

var okNext2  = (x:Int) => x + 1

var okNext3:(Int => Int) = x => x + 1

var okNext4:(Int => Int) = _ + 1
```

Using Type Inference

```
def addN(n:Int):(Int => Int) = {
  def adder(k:Int):Int = {
    k + n
  }
  adder
}
```

```
def addN(n:Int):(Int => Int) = {
  _ + n
}
```