UCSC Extension Classes

Search this site

G+1 0

UCSC Extension Classes > Introduction to Spark with Scala >

Introduction To Scala

Start up spark-shell before trying out the following Scala examples

- cd into spark-1.6.0-bin-hadoop2.6
- ./bin/spark-shell
- To exit the shell enter exit

Few hints about working with the REPL

- :help - to display the supported commands
- :history - show the history
- :exit - exit the REPL
- hit TAB key to see available variables
- Code completion: type sc. then hit TAB key

Basic Types

```
// mutable variables
var counter:Int = 10
var d = 0.0
var f = 0.3f

// immutable variables
val msg = "Hello Scala"
println(msg)

// string interpolation
s"Greeting: $msg"

val ? = scala.math.Pi

println(?)
```

Defining functions

```
def hello(name:String) : String = { "Hello " +
name }

def hello1() = { "Hi there!" }
def hello2() = "Hi there!"
def hello3 = "Hi there!"

def max(a:Int, b:Int) : Int = if (a > b) a else
b

max(4,6)
max(8,3)
```

Function literal

```
(x: Int, y: Int) => x + y
val sum = (x: Int, y: Int) => x + y
sum(1,70)
val prod = (x: Int, y: Int) => x * y

def doIt(msg:String, x:Int, y:Int, f: (Int,
Int) => Int) = {
  print(msg + f(x,y))
}

doIt("sum: ", 1, 80, sum)

doIt("prod: ", 2, 33, prod)
```

Tuple

```
val pair = ("Scala", 1)

println(pair._1)  ==> "Scala"
println(pair._2)  ==> 1
```

Class

```
// constructor with two private instance
variables
class Movie(name:String, year:Int)

// With two getter methods
class Movie(val name:String, val year:Int)
val m1 = new Movie("100 days", 2010)
M1.name
m1.year

// With two getter and setter methods
class Movie(var name:String, var year:Int)
val m2 = new Movie("100 days", 2010)
m.name = "100 Hours"
```

Case Class - Scala class on steroid - implicitly come with implementations of methods toString, equals, hashCode, accessor methods only.

```
case class Movie(name:String, year:Int)

val m = Movie("100 days", 2010)
m.toString  ==> Movie(100 days,2010)

println(m.name + "  " + m.year)
```

Pattern Matching

```
def errorMsg(n:Int) = n match {
   case 1 => println("Not a problem")
   case 2 => println("You may want to double
check")
   case 3 => println("System is shutting down")
}

def range(n:Int) = n match {
  case lessThan10 if (lessThan10 <= 10) =>
println("0 .. 10")
  case lessThan50 if (lessThan50 <= 50) =>
println("11 .. 50")
  case _ => println("> 50")
}
range(8)  ==> "0 .. 10"
range(25) ==> "11 .. 50"

======================================
   case matching with Case class
======================================

abstract class Shape
case class Rectangle(h:Int, w:Int) extends
Shape
case class Circle(r:Int) extends Shape

def area(s:Shape) = s match {
  case Rectangle(h,w) => h * w
  case Circle(r) => r * r * 3.14
}

println(area(Rectangle(4,5)))

println(area(Circle(5)))
```

Array

```
val myArray = Array(1,2,3,4);

myArray(0)                              ==> 1
myArray(0) = myArray(1) + 1;
myArray(0)                              ==> 3


myArray.foreach(a => print(a + " "))
myArray.foreach(println)

// iterating with index
for (i <- 0 until myArray.length)
  println(myArray(i))

// iterating without index
for (v <- myArray)
  println(v)

def validLength(m:Array[String]) : Boolean =
 {

   if (m.length == 3) true else false

}

val movie = Array("Lucy", "2014")

validLength(movie)
```

List

```
val l = List(1,2,3,4);
l.foreach(println)
l.foreach(x => println(x + " "))

println(l.head)  ==> 1
println(l.tail)  ==> List(2,3,4)
println(l.last)  ==> 4
println(l.init)  ==> List(1,2,3)

val table: List[List[Int]] = List (
       List(1,0,0),
       List(0,1,0),
       List(0,0,1)
)

val list = List(2,3,4);

// cons operator – prepend a new element to the
beginning
```

```
val m = 1::list

// appending
val n = list :+ 5

// to find out whether a list is empty or not
println("empty list? " + m.isEmpty)

// take the first n elements
list.take(2) ==> List(2,3)

// drop the first n elements
list.drop(2) ==> List(4)
```

List - high order methods

```
val n = List(1,2,3,4)
val s = List("LNKD", "GOOG", "AAPL")
val p = List(265.69, 511.78, 108.49)

var product = 1;
n.foreach(product *= _)  ==> 24

n.filter(_ % 2 != 0)     ==> List(1,3)
n.partition(_ % 2 != 0)  ==> (List(1,3),
List(2,4))

n.find(_ % 2 != 0)       ==> Some(1)
n.find(_ < 0)            ==> None

p.takeWhile(_ > 200.00)  ==> List(265.69,
511.78)
p.dropWhile(_ > 200.00)  ==> List(108.49)

p.span(_ > 200.00)       ==> (List(265.69,
511.78),List(108.49))

val n = List(1,2,3,4)
val s = List("LNKD", "GOOG", "AAPL")


n.map(_ + 1)             ==> List(2,3,4,5)
s.flatMap(_.toList)      ==>
List(L,N,K,D,G,O,O,G,A,A,P,L)

n.reduce((a,b)           ==> { a + b} )  ==> 10
n.reduce(_ + _)

n.contains(3)            ==> true
```

List - pattern matching

```
val n = List(1,2,3,4)
val s = List("LNKD", "GOOG", "AAPL")

def sum(xs: List[Int]) : Int = xs match {
  case Nil => 0
  case x :: ys => x + sum(ys)
}
```

Merging two sorted lists

Java

```
public static void merge(int []a, int []b, int
[]c) {
  int cursorA = 0,cursorB = 0, cursorC = 0;
  int sizeA = a.length; int sizeB = b.length;

  while(cursorA < sizeA && cursorB < sizeB) {
    if (a[cursorA] < b[cursorB]) {
      c[cursorC++] = a[cursorA++];
    } else {
      c[cursorC++] = b[cursorB++];
    }

    while(cursorA < sizeA) {
       c[cursorC++] = a[cursorA++];
    }

    while(cursorB < sizeB) {
       c[cursorC++] = b[cursorB++];
    }
  }
}
```

Challenge #1 – Write a function to calculate the length of a given List using recursion

```
def length(xs: List[Int]) : Int = xs match {

}

val xs = List(1,2,3,4)
println(length(xs)) // expect 4
```

Challenge #2 – Write a function to reverse the elements in the list

```
val xs = List(1,2,3,4)
def reverse(xs: List[Int]) : List[Int] = xs
match {

}

println(reverse(xs))   // expect List(4,3,2,1)
```

Challenge #3

Write a function to validate whether parentheses are balance

Your function should return **true** for the following strings

- (if (zero? x) max (/ 1 x))
- I told him (that it’s not (yet) done). (But he wasn’t listening)

Your function should return **false** for the following strings

- :-)
- ())(

```
def balance(chars: List[Char]): Boolean {
}
```

The following methods are useful for this challenge

- chars.isEmpty
- chars.head
- chars.tail

Hint: you can define an inner function if you need to pass extra parameters to your function.

To convert a String to List[Char] ==> "ucsc school".toList

Extra credit: write another implementation that uses pattern match with list extraction pattern

Challenge #4

Given a list of numbers with duplicate values – write a function to remove the duplicates.

Hint – think recursively, leverage pattern matching and declare all the possible cases your function needs to handle. Use List.contains method and list append operator :+

```
val dups = List(1,2,3,4,6,3,2,7,9,4)


def removeDups(xs : List[Int]) : List[Int] = xs
match {

}

removeDups(dups).sorted  // expect
List(1,2,3,4,6,7,9
```
)

Challenge #5

Write two functions for performing run length encoding and decoding:

encoding("WWWWWWWWWWWWBWWWWWWWWWWWWBBBWWWWWWWW") ==> "12W1B12W3B8W"
decoding("12W1B12W3B8W")
==> "WWWWWWWWWWWWBWWWWWWWWWWWWBBBWWWWWWWW"

Comments

Commenting disabled due to a network error. Please reload the page.

You do not have permission to add comments.