Scala Cheat Sheet

Topic	Description	Examples
Comments	Help you understand and document your code.	//single line comment /* Multi-line comment ya da ya da */
Types	Any value needs to have a clearly specified type.	• The value 2 is of type Int • The value 2.43 is of type Double • The value true is of type Boolean • The value 'a' is of type Char • The value “a” is of type String
Variables	Variables hold values or references to where values are stored in memory. Variables can be immutable (val) or immutable (val). The scope of variables depends upon the scope in which they are defined.	val x: Int = 5 var y: Int = 6 y = 7//reassingment
Assignment Operator	Equals sign sets the variable on the left to the value of the expression on the right	y = 7
Maths Operators	Just as you'd expect...	Addition (+) Multiplication (*) Subtraction (-) Division (/) Modulus(%)
Comparison Operators	All good, but pay heed to equality operator (two equals signs in a row)	< (less than) > (greater than) <= (less than or equal to) >= (greater than or equal to) == (test for equality)

Logical Operators	Not (!)
	
	And (&&)
	
	Or (||)
		if (!playing)//if not playing is true
		
		if (alive && playing)//both must be true
		
		if (alive || playing)//if alive or playing are true

Functions	Enables you to break your code up into logical functional modules.	//a function that takes no parameters and returns nothing
		def printHi(): Unit = println(“Hi”)
		
		//or multi-line version
		def printHi(): Unit = {
		println(“Hi”)
		println(“There”)
		}
		
		//a function that takes parameters but returns no value
		def print(message: String): Unit = {
		println(message)
		}
		
		//a function that takes parameters and returns a value
		def percent(score: Double, total: Double): Double {
		(score/total)*100
		}
		
		//a function call
		val p: Double = percent(30, 40)

Conditional Statement - If	Handle logical branching – if some condition is true then do something, if not then do something else. Conditions can be embedded.	if (someConditionEqualsTrue) {
		doSomething
		} else {
		doSomethingElse
		}
		//OR
		if (someConditionEqualsTrue) {
		doSomething
		} else if (someOtherConditionEqualsTrue){
		doSomethingElse
		} else {
		doSomethingElseAgain
		}

Conditional	Another way of handling logical branching. You	someVariable match {

Statement – Match Case	indicate the variable that you are checking. Then each case statement indicates possible values for the variable and then something to happen should that condition be true.	case someValue => doSomething case someOtherValue => doSomethingElse case someOtherValue => doSomethingElseAgain case _ => catchAnyOtherConditions }

Arrays	Store sets of things. The first item in an array is at index 0. The last item is at the length of the array – 1.	//declare and assign values
		val birds: Array[String] = Array(“hen”, “duck”, “emu”)
		
		//declare and then assign values
		val birds:Array[String] = new Array[String](3)
		birds(0) = “hen”
		birds(1) = “duck”
		birds(2) = “emu”
		
		val firstItem: String = birds(0)
		val lastItem: String = birds(birds.length-1)

While Loop	Basic looping mechanism – only use for main game loop. Need to make sure that the specified condition (here 'playing') can be set to false otherwise the loop never ends (infinite loop).	while(playing) {
		doWhatever
		}

For Loop	More sophisticated looping mechanism. The for keyword is followed by a parenthetical statement. On the right side of this statement is a variable that takes on each value in the sequence indicated on the right. I read the '<-” as an arrow (put the value at right into the variable at left).	//looping from start index to finish index
		for(i<-0 to 99) println(i)
		
		//looping through an array
		for(bird<-birds)println(bird)
		
		//note if you have multiple statements needed for each //loop iterations, the write:
		for(bird<-birds) {
		whatever
		whatever
		}

For Each	Another more advanced looping mechanism. Works with arrays and other types of sequences. You indicate a function that should run for each item in a sequence.	val a: Array[String] = Array("b", "c", "d")
		
		//print out all the values
		a.foreach(println)
		
		//longer winded means of doing the same
		a.foreach(arg: String) => println(arg)
		
		//or if you have predefined function (f)
		a.foreach(f)

Import Classes	Import classes to use in your program. You have to provide the full package path.	//import specific class
		import package.subpackage.ClassName
		
		//import all classes in subpackage
		import package.subpackage._

Map	Store key-value pairs	import scala.collection.mutable.Map val collection: Map[String, Boolean] = Map("key" -> false, "dagger" -> true, "ring" -> false) println(collection("key"))//prints false
Random	Import the Random class. Create a new instance of Random. Establish a maximum value. Calculate a random value between 0 and one short of the maximum value.	import scala.util.Random val random: Random = new Random() val maxNum: Int = 6 val ranNum: Int = random.nextInt(maxNum+1)

User Input	Whenever you read the Console input the program waits until you enter something and hit return.	//get the input character
		val inputChar: Char = Console.readChar()
		
		//get the input String on a single line
		val inputString: String = Console.readLine().trim

Classes	Classes contain fields (attribute variables) and methods (functions).	//create class
		Class Dog (var name: String) {
		def bark(): Unit = println(“woof, woof!”)
		}
		//create an instance of Dog and test
		val dog: Dog = new Dog(“Fido”)
		dog.name //prints out “Fido”
		dog.bark //prints out “woof, woof!”)