SCALA PROGRAMMING 2.12

(Training: 3rd Oct 2023 to 11th Oct 2023)

Introduction

- Scala is a pure Object-Oriented language.
- It's a functional language since every function is a value
- Functions are treated as first class citizens which means we can assign function and return then as a result from function.
- Scala is a statically typed language.
- Scala is compiled to java bytecode, so that it can run on any platform where java is supported.
- Everything is an object in Scala.
- Conciseness and expressiveness: Scala minimize boilerplate code.
- Type inference: no need to mention datatype while declaring variable
  int data=10; //java
  val data=10; //scala

Setup Requirements

jdk 1.8

scala 2.12 and path must be added to env variables

Scala IDE (specific eclipse)

Important Points

- Semicolons are optional in Scala but in few cases where we are having multiple statement at same line. For example,
  // val a=10; val b=20; val c=30
- 'object' keyword - it is used to define a singleton object.
- Name of the source file and object name can be different.

Scala Variables

2 types –

1. Immutable (val keyword is used) - cannot change the value of variable - recommended type.
2. mutable (var keyword is used) - value of variable can be changed)

- Scala encourages immutability as a first-class citizen in the language.

Variable Conventions:

1. should be in lowercase

2. can contain letter and only 2 special characters allowed (_ and $)

3. whitespace not allowed

4. must start with alphabet

5. keywords cannot be used

Lazy Initialization

- Sometimes we need to delay the initialization of variable until at the point where it's consumed by application
- This can be particularly useful in situations where the initialization of a resource is costly or time consuming and u want to avoid unnecessary overhead by initialization it lazily.
- 'lazy' keyword is used. lazy variables will not be initialized until first time it is used. Once it is initialized, the value will be cached and reused for subsequent access.
- Use cases of lazy
  1. improve performance 2. reducing memory usage

Type inference - it allows us to declare variable without mentioning datatype and scala assigns datatype based on declaration.

Scala is statically typed. So, cannot change datatype.

String interpolation is used to print and format the variables.

```
//define any var along with its data type in scala
     val num1: Int = 5

//Implicit casting
     val numberOfPizzas: Short=1
     val minimumPizzasToBuy: Int= numberOfPizzas //implicit casting is taking
place.
     //short to int is allowed but reverse is not possible
```

- all Scala control structures can be used as expressions
- Remember expressions returns a result while statement does not
- The if condition which contains only statements cannot be assigned to any variable.

Match Expression

```
//match expression which is equivalent to switch case
    val choice=0
    var result="Day is:  ";

    val day =
      choice match
      {
      case 0=>"Sunday"
      case 1=>"Monday"
      case 2=>"Tuesday"
      case 3=>"Thursday"
      case _=>"Invalid day” // default value, catch all
      }

    //VVVVIMPNOTE: default has to be last in match expression
    //u cannot write any case after that, it will be treated as unreachable code
    println(result+day)


    //What if we have multiliner code per case
    // Do remember {} are not required to mark it as a block for particular case
    // in scala next case is treated as termination of prev case, or } curly
bracket of entire match exp is tread as
    //closure of current case.
```

```
    //Scenario is, we have 4 cases like 0,1,2,3 and default case
    // in all of these cases we know what was the val in choice variable
    //but we would like to know what value user has entered and because of that
default case has executed.

    println("-----------------------------------------------------")
    val choice1=40

    choice1 match{
      case 0 => println("ZERO")
      case 1 => println("ONE")
      case what => println(s"You gave me :  $what ")
    }


    println("-----------------------------------------------------")

//Scenario: How to handle multiple possible matches on one line
    val choice2=33

    val evenOrOdd =

      choice2 match
      {
      case  1 | 3| 5| 7| 9 => println("odd")
      case  2 |4 |6 |8 |10 => println("even")
      case _               => println("some other number")

      }
```

For Loop

```
//basic syntax
    for( element <- collection)
    {
        code to execute
    }

// a simple for loop from 1 to 5 inclusive
    for( numberOfPizzasToBuy <- 1 to 5)
    {
       println(s"Number of pizzas to buy =  $numberOfPizzasToBuy  ")
    }

    //a simple for loop from 1 to 5 where 5 is not inclusive
    println("-----------------------------------------------------------------")
    for(numberOfPizzasToBuy <- 1 until 5)
    {
       println(s"Number of pizzas to buy =  $numberOfPizzasToBuy  ")
    }

//Filter values using if condition in for loop
//VVVIMPNOTE: if expression inside for loop are referred as guards.

    val pizzaIngredients= List("dough","tomat sauce","mozzarella","bell
pepper","olives", "mushroom")
```

```
// loop thru the list of pizza ingredients and filter out all items for the
mozzarella
     println("-----------------------------------------------------------------")
    for(ingredient  <- pizzaIngredients if ingredient == "mozzarella")
    {
       println(s"Found cheesy ingredient = $ingredient")
    }

//Take above pizza ingresients and filter for toppig elemnets as "olives",
"mushroom" and store it in a var

   var toppingIngredients= for{
      ingredient  <- pizzaIngredients
      if( ingredient == "olives" || ingredient =="mushroom" )
    } yield ingredient

     println("-----------------------------------------------------------------
-")
    println(s"Topping ingredeints= $toppingIngredients")



    //using multiple guards
     println("----------Example of multiple guards-------------------------------
-")
    for{
      i <- 1 to 3
      j <- 'a' to 'c'
      if i==2
      if j=='b'
    }
    {

      println(s"i= $i   , j=$j")
    }


    //for expression
    val ints= List(1,2,3,4,5)
    val abc1 = for( i <- ints) yield i*2
    val abc2 = for( i <- ints) yield ( i*2 )
    val abc3 = for{ i <- ints} yield ( i*2 )
```

While Loop

```
      var numberOfPizzasToBake= 10

      while(numberOfPizzasToBake >  0)
      {
            println(s"Remaining pizzas to bake= $numberOfPizzasToBake")
            numberOfPizzasToBake-=1
      }
```

Do-While Loop

```
      var numberofPizzasBaked=20

      do
      {
        numberofPizzasBaked +=1
```

```
    println(s"Number of pizzas baked is $numberofPizzasBaked")
  }while(numberofPizzasBaked <5)
```

Breakable

```
    var i=0
    breakable{
    while(i <= 10)
    {
      println(i)
      if(i ==3)
        break;
      i= i+1
    }

  }
```

Arrays

Basic Example

```
var size=0

  var array1=  new Array[Int] (5);  //5 50  // 50  10
  //how to init values in array
  array1(0)=10
  array1(1)=10
  array1(2)=10
  array1(3)=10
  array1(4)=10

  println("Array size "+ array1.length);

  for( num <- array1)
    println(num)
```

Array of unknown size

```
//How to create array when we dont know the size in advance???
//Achieved using ArrayBuffer
   var allNames= ArrayBuffer[String]()

   allNames+="Java"
   allNames+=".Net"
   allNames+="python"

   println(allNames)

   //later convert it to normal array if required.
   var newArray= allNames.toArray
```