# Scala cont'

Types, Loops, Strings, Reading Files

# Scala Types

- All values in Scala are objects
  - Objects contain variables and methods
  - No primitive values in Scala
- We'll start with the following types:
  - Int
  - Long
  - Double
  - Boolean
  - Unit
  - String

# Int

- A whole number
- 32 bit representation
  - 31-bit number and one sign bit
- -2147483648 to 2147483647
  - Values outside this range will **overflow**
    - Or underflow
  - Overflow values will "wrap around"

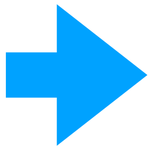
```
val a: Int = 2147483647
println(a + 1)
```

**-2147483648**

# Long

- A whole number (Like Int)
- 64 bit representation
  - 63-bit number and one sign bit
- -9223372036854775808 to 9223372036854775807
- Useful when you expect values that would overflow an Int

```
val a: Long = 2147483647
println(a + 1)
```

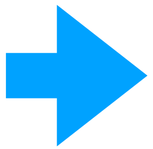


**2147483648**

# Integer Division

- When dividing two Ints/Longs the result is always an Int/Long
- Decimal portion is removed
  - Effectively returns the floor of the result

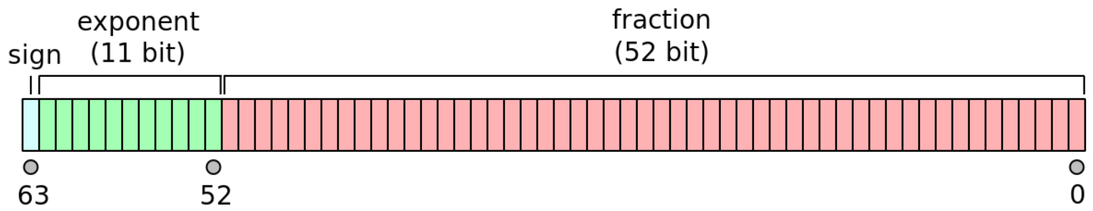
```
val ageInMonths: Int = 245
val monthsPerYear: Int = 12
val ageInYears = ageInMonths/monthsPerYear
println(ageInYears)
```



**20**

# Double

- Number with a whole number and a decimal portion
- 64-bit representation
- Values are truncated to fit in 64 bits
  - Loss of precision!




**https://en.wikipedia.org/wiki/Double-precision_floating-point_format**

# Double

- Values are represented in binary
  - Ex. 0.11 == 1/2 + 1/4 == 3/4
- In decimal we have values that cannot be stored without truncation
  - Ex. 1/3 != 0.33333333333333333333
- Values such as 0.1 cannot be represented as a sum of powers of 2
  - 0.1 (base 10) != 0.0001100110011001100110011001100110011001100110011001 (base 2)
  - But this the best we can do with Double representations

# Double

- We need to be aware of this truncation in our programs
- In the code below, c == 0.3 is false!

```
val b: Double = 0.1
val c: Double = b * 3
println(c)
```

**0.30000000000000004**

# Boolean and Unit

- Boolean
  - true or false
- Unit
  - Nothing
  - Used to indicate a method/function that does not return a value
  - Ex: main and println both return Unit

# String

- A sequence of characters (type Char)
- Declared with double quotes
  - val s: String = "valid String literal"
- Many useful methods. Examples:
  - .startsWith(String) - check if this String starts with the given String
  - .length() - number of characters in this String
  - .split(String) - Separates this String by the given String

# Scala Type Conversions

```scala
package example

object Types {

  def main(args: Array[String]): Unit = {

    // Declaring variable
    var anInt: Int = 10
    var aDouble: Double = 5.8
    var aBoolean: Boolean = true
    var aString: String = "6.3"

    // Converting variable types
    var anotherDouble: Double = aString.toDouble
    var anotherString: String = anInt.toString

    // Truncates the decimal. anotherInt == 5
    var anotherInt: Int = aDouble.toInt
  }

}
```

Use the "to<Type>" to convert between certain types

# For Loop

# For Loop

```
for(<variable_name> <- <data_structure>){
  <loop_body>
}
```

Reads:

"for variable_name in data_structure, execute loop_body"

# For Loop

```
package example

object Loop {

  def printOneTo(n: Int): Unit = {
    for(i <- 1 to n){
      println("i == " + i)
    }
  }

  def printOneToAlternate(n: Int): Unit = {
    val numbers: Range = 1 to n
    for (i <- numbers) {
      println("i == " + i)
    }
  }

  def main(args: Array[String]): Unit = {
    printOneTo(10)
  }

}
```

**Output:**

**i == 1**
**i == 2**
**i == 3**
**i == 4**
**i == 5**
**i == 6**
**i == 7**
**i == 8**
**i == 9**
**i == 10**

"1 to n" creates a Range of integers that can be iterated over with a for loop
-Similar to range(n) in Python

# For Loop + String Example

```scala
package example

object StringSplitter {

  def computePercentTrue(line: String): Double = {
    val splits: Array[String] = line.split(";")
    var totalCount: Double = 0.0
    var trueCount: Double = 0.0
    for (value <- splits) {
      val valueAsBoolean: Boolean = value.toBoolean
      if (valueAsBoolean) {
        trueCount += 1.0
      }
      totalCount += 1.0
    }
    trueCount / totalCount
  }

  def main(args: Array[String]): Unit = {
    val testInput = "true;false;true;true;true"
    val percentTrue = computePercentTrue(testInput) // expecting 0.8
    println("Percentage true == " + percentTrue)
  }

}
```

Given a String containing boolean values separated by semicolons, return the percentage of values that are true

# For Loop + String Example

```
package example

object StringSplitter {

  def computePercentTrue(line: String): Double = {
    val splits: Array[String] = line.split(";")
    var totalCount: Double = 0.0
    var trueCount: Double = 0.0
    for (value <- splits) {
      val valueAsBoolean: Boolean = value.toBoolean
      if (valueAsBoolean) {
        trueCount += 1.0
      }
      totalCount += 1.0
    }
    trueCount / totalCount
  }

  def main(args: Array[String]): Unit = {
    val testInput = "true;false;true;true;true"
    val percentTrue = computePercentTrue(testInput) // expecting 0.8
    println("Percentage true == " + percentTrue)
  }

}
```

Split the String on semicolons
-Returns an Array of Strings

# For Loop + String Example

```
package example

object StringSplitter {

  def computePercentTrue(line: String): Double = {
    val splits: Array[String] = line.split(";")
    var totalCount: Double = 0.0
    var trueCount: Double = 0.0
    for (value <- splits) {
      val valueAsBoolean: Boolean = value.toBoolean
      if (valueAsBoolean) {
        trueCount += 1.0
      }
      totalCount += 1.0
    }
    trueCount / totalCount
  }

  def main(args: Array[String]): Unit = {
    val testInput = "true;false;true;true;true"
    val percentTrue = computePercentTrue(testInput) // expecting 0.8
    println("Percentage true == " + percentTrue)
  }

}
```

Iterate over each value

# For Loop + String Example

```
package example

object StringSplitter {

  def computePercentTrue(line: String): Double = {
    val splits: Array[String] = line.split(";")
    var totalCount: Double = 0.0
    var trueCount: Double = 0.0
    for (value <- splits) {
      val valueAsBoolean: Boolean = value.toBoolean
      if (valueAsBoolean) {
        trueCount += 1.0
      }
      totalCount += 1.0
    }
    trueCount / totalCount
  }

  def main(args: Array[String]): Unit = {
    val testInput = "true;false;true;true;true"
    val percentTrue = computePercentTrue(testInput) // expecting 0.8
    println("Percentage true == " + percentTrue)
  }

}
```

Convert the Strings to Booleans

# For Loop + String Example

```
package example

object StringSplitter {

  def computePercentTrue(line: String): Double = {
    val splits: Array[String] = line.split(";")
    var totalCount: Double = 0.0
    var trueCount: Double = 0.0
    for (value <- splits) {
      val valueAsBoolean: Boolean = value.toBoolean
      if (valueAsBoolean) {
        trueCount += 1.0
      }
      totalCount += 1.0
    }
    trueCount / totalCount
  }

  def main(args: Array[String]): Unit = {
    val testInput = "true;false;true;true;true"
    val percentTrue = computePercentTrue(testInput) // expecting 0.8
    println("Percentage true == " + percentTrue)
  }

}
```

Count the total number of values and the number that are true

# For Loop + String Example

```
package example

object StringSplitter {

  def computePercentTrue(line: String): Double = {
    val splits: Array[String] = line.split(";")
    var totalCount: Double = 0.0
    var trueCount: Double = 0.0
    for (value <- splits) {
      val valueAsBoolean: Boolean = value.toBoolean
      if (valueAsBoolean) {
        trueCount += 1.0
      }
      totalCount += 1.0
    }
    trueCount / totalCount
  }

  def main(args: Array[String]): Unit = {
    val testInput = "true;false;true;true;true"
    val percentTrue = computePercentTrue(testInput) // expecting 0.8
    println("Percentage true == " + percentTrue)
  }

}
```

Compute the average

-Note: If these values were Ints this would be integer division

# String Splitting

```
def accessSplitsExample(): Unit = {
  val stringToSplit: String = "value1_value2_value3"

  val splits: Array[String] = stringToSplit.split("_")

  // Access the three values
  val firstValue: String = splits(0)
  val secondValue: String = splits(1)
  val thirdValue: String = splits(2)

  println(firstValue)
  println(secondValue)
  println(thirdValue)
}
```

Use (index) to access the value in an Array at a specific index

# Reading Files

# Reading Files

```scala
package example

import scala.io.{BufferedSource, Source}

object FileReader {

  def convertFileToString(filename: String): String = {
    var contents: String = ""
    val file: BufferedSource = Source.fromFile(filename)
    for (line <- file.getLines()){
      contents += line + "\n"
    }
    contents
  }

  def main(args: Array[String]): Unit = {
    val filename = "data/testFile.txt"
    val contents = convertFileToString(filename)
    println(contents)
  }

}
```

Read the contents of a file into a String line-by-line
-Assumes "data/testFile.txt" exists in the project

# Reading Files

```scala
package example

import scala.io.{BufferedSource, Source}

object FileReader {

  def convertFileToString(filename: String): String = {
    var contents: String = ""
    val file: BufferedSource = Source.fromFile(filename)
    for (line <- file.getLines()){
      contents += line + "\n"
    }
    contents
  }

  def main(args: Array[String]): Unit = {
    val filename = "data/testFile.txt"
    val contents = convertFileToString(filename)
    println(contents)
  }

}
```

Import the objects needed from the standard library

# Reading Files

```scala
package example

import scala.io.{BufferedSource, Source}

object FileReader {

  def convertFileToString(filename: String): String = {
    var contents: String = ""
    val file: BufferedSource = Source.fromFile(filename)
    for (line <- file.getLines()){
      contents += line + "\n"
    }
    contents
  }

  def main(args: Array[String]): Unit = {
    val filename = "data/testFile.txt"
    val contents = convertFileToString(filename)
    println(contents)
  }

}
```

Call scala.io.Source.fromFile(filename: String): BufferedSource

# Reading Files

```scala
package example

import scala.io.Source

object FileReader {

  def convertFileToString(filename: String): String = {
    var contents: String = ""
    val file: BufferedSource = Source.fromFile(filename)
    for (line <- file.getLines()){
      contents += line + "\n"
    }
    contents
  }

  def main(args: Array[String]): Unit = {
    val filename = "data/testFile.txt"
    val contents = convertFileToString(filename)
    println(contents)
  }

}
```

Call BufferedSource.getLines() to get the lines in a data structure of Strings

# Reading Files

```scala
package example

import scala.io.Source

object FileReader {

  def convertFileToString(filename: String): String = {
    var contents: String = ""
    val file: BufferedSource = Source.fromFile(filename)
    for (line <- file.getLines()){
      contents += line + "\n"
    }
    contents
  }

  def main(args: Array[String]): Unit = {
    val filename = "data/testFile.txt"
    val contents = convertFileToString(filename)
    println(contents)
  }

}
```

Do whatever you need to do with the content of the file

Note: Creating a String with all the contents of file is only done as an example. Do not create such a String when reading large files

# Project 1 - Task 1

- In the PaleBlueDot object, which is in the pbd package, write a method named "getCountryCode" which:
  - Takes two Strings as parameters representing:
    - The name of a file containing country data. ex. "data/coutries.txt"
    - The name of a country to lookup in this file
  - Returns the 2 character country code as a String of the country name parameter
    - The country code must be all lowercase
    - The country name is not case-sensitive (ex. You code must treat "jaPan" and "JAPAN" as the same country name and return "jp" for both
    - Returns "" if the country is not found in the file

**Sample lines from the countries file**

```
Jamaica#JM
Jordan#JO
Japan#JP
```

Submit a zip file of your project to AutoLab:

File > Export to zip file -OR- tools -> Pack Whole Project

# Reading Files

## Example in IntelliJ