1. Pattern Matching

Matches Squence of data.

Syntax **var match case =>**

Example (Start the command line with “cmd” and write “scala” to start scala repl)

Example 1: Simple pattern matching

```
scala> def matchPattern(x: Int) = x match {
     | case 1 => "One"
     | case 2 => "Two"
     | case _ => "None of above"
     | }
matchPattern: (x: Int)String


scala> println(matchPattern(1))
One

scala> println(matchPattern(2))
Two

scala> println(matchPattern(4))
None of above
```

Example 2: Any data type for input parameter and output parameter

```
scala> def matchPattern(x:Any) = x match {
     | case 1 => "One"
     | case 2 => "Two"
     | case "Three" => 3
     | case "Four" => 4
     | case _ => "None of Above"
     | }
matchPattern: (x: Any)Any

scala> matchPattern(1)
res4: Any = One

scala> matchPattern(2)
res5: Any = Two

scala> matchPattern(3)
res6: Any = None of Above

scala> matchPattern(Three)
<console>:9: error: not found: value Three
              matchPattern(Three)
                           ^

scala> matchPattern("Three")
res8: Any = 3

scala> matchPattern("Four")
res9: Any = 4

scala> matchPattern("four")
```

```
res10: Any = None of Above
```

Example 3: Using Case class

- **Default for arguments/parameter is val (immutable)**
- **It generates some functions/methods automatically when you define class as a case class. It includes methods like equals, hashCode, toString**

```
scala> case class Car(name:String, cost:Int)
defined class Car

scala> val mercedes = new Car("Mercedes", 500000)
mercedes: Car = Car(Mercedes,500000)

scala> val bmw = new Car("BMW", 700000)
bmw: Car = Car(BMW,700000)

scala> val jaguar = new Car("Jaguar", 1000000)
jaguar: Car = Car(Jaguar,1000000)

scala> for (car <- List(mercedes,bmw,jaguar) ) {
     | car match {
     | case Car("Mercedes",500000) => println("Car is Mercedes, Congrats!")
     | case Car("BMW",700000) => println("Car is BMW, Waow!")
     | case Car(name,cost) => println("Car is" +name + "Cost is " + cost + "Thats Awesome!!!")
     | }
     | }
Car is Mercedes, Congrats!
Car is BMW, Waow!
Car isJaguarCost is 1000000Thats Awesome!!!

scala> val mercedesa = new Car("Mercedes" , 550000 )
mercedesa: Car = Car(Mercedes,550000)
scala>  for (car <- List(mercedes,bmw,jaguar,mercedesa)){
     |  car match {
     | case Car("Mercedes",500000) => println("Car is Mercedes, Congrats!")
     | case Car("BMW",700000) => println("Car is BMW, Waow!")
     |  case Car(name,cost) => println("Car is" +name + "Cost is " + cost + "Thats Awesome!!!")
     | }
     | }
Car is Mercedes, Congrats!
Car is BMW, Waow!
Car isJaguarCost is 1000000Thats Awesome!!!
Car isMercedesCost is 550000Thats Awesome!!!
```

2. Regular Expression

Regular Expression in Scala is adapted from Java.
Java Regular Expression is also adopted from Perl.

- **import scala.util.matching.Regex**

- **You have to create an object of class Regex**

  val pattern = new Regex(“Whatever you want to match”)
  OR
  val pattern = “whatevet you want to match”.r

  r – is a method/function that is defined in a Regex class and it does nothing but calls the constructor

```
scala> import scala.util.matching.Regex    // Always import this first
import scala.util.matching.Regex

scala> val pattern = new Regex("Hello")    // Using the constructor for class Regex
pattern: scala.util.matching.Regex = Hello

scala> val stringToFind = "Hello How are you? Hello Again"    // String where you want to
search the pattern
stringToFind: String = Hello How are you? Hello Again
              ^

scala> pattern findFirstIn stringToFind              // Syntax to find the pattern in a
given String. findFirstIn is the method which will only find the 1st instance of pattern
res16: Option[String] = Some(Hello)                     // if you search for something
that is not found, it will give you as None

scala> pattern findAllIn stringToFind                // findAllIn - will return all the
strings matching with pattern
res17: scala.util.matching.Regex.MatchIterator = non-empty iterator

scala> (pattern findAllIn stringToFind).mkString(", ")
res18: String = Hello, Hello

scala>

scala> val pattern = "Hello".r           // used the method r instead of using new Regex
pattern: scala.util.matching.Regex = Hello

scala> (pattern findAllIn stringToFind).mkString(", ")
res19: String = Hello, Hello

scala>

scala> var stringToFind = "My name is Harish and age is 10 and i study in standard 7"
stringToFind: String = My name is Harish and age is 10 and i study in standard 7
```

```
scala> val pattern = "[0-9]+".r       // find all between 0 to 9 with 1 or more instance
pattern: scala.util.matching.Regex = [0-9]+

scala> (pattern findAllIn stringToFind).mkString(", ")
res20: String = 10, 7

scala> (pattern findAllIn stringToFind).toArray
res21: Array[String] = Array(10, 7)

scala>

scala>

scala> val stringToFind = "Hello How are you? hello Again"
stringToFind: String = Hello How are you? hello Again

scala> val pattern = "(H|h)ello".r
pattern: scala.util.matching.Regex = (H|h)ello   // Find both Hello and hello.

scala> (pattern findAllIn stringToFind).toArray
res22: Array[String] = Array(Hello, hello)
```

A. Using Regular expression with **getOrElse** Function

```
scala> val pattern = "Hellooooooo".r  // Trying to search for Hellooooooo
pattern: scala.util.matching.Regex = Hellooooooo

scala> val stringToFind = "Hello How are you? hello Again"
stringToFind: String = Hello How are you? hello Again

scala> pattern findFirstIn stringToFind getOrElse("No Match Found") // it will go in else as
pattern not found
res29: String = No Match Found

scala> val pattern = "Hello".r
pattern: scala.util.matching.Regex = Hello

scala> pattern findFirstIn stringToFind getOrElse("No Match Found") // it will get the value as
pattern was found
res30: String = Hello
```

B. Using regular expression with forEach

```
scala> val pattern = "(H|h)ello".r
pattern: scala.util.matching.Regex = (H|h)ello

scala> val stringToFind = "Hello How are you? hello Again"
stringToFind: String = Hello How are you? hello Again

scala> pattern findAllIn stringToFind foreach(d=>print(d))
Hellohello

scala> pattern findAllIn stringToFind foreach(d=>println(d))
Hello
Hello
```

Options for Regular Expression Forming

SUBEXPRESSION	MATCHES

SUBEXPRESSION	MATCHES
^	It is used to match starting point of the line.
$	It is used to match terminating point of the line.
.	It is used to match any one character excluding the newline.
[…]	It is used to match any one character within the brackets.
[^…]	It is used to match any one character which is not in the brackets.
\\A	It is used to match starting point of the intact string.
\\z	It is used to match terminating point of the intact string.
\\Z	It is used to match end of the whole string excluding the new line, if it exists.
re*	It is utilized to match zero or more appearances of the foregoing expressions.
re+	It is used to match one or more of the foregoing expressions.
re?	It is used to match zero or one appearance of the foregoing expression.
re{ n}	It is used to matches precisely n number of appearances of the foregoing expression.
re{ n, }	It is used to match n or more appearances of the foregoing expression.
re{ n, m}	It is used to match at least n and at most m appearances of the foregoing expression.
q|r	It is utilized to match either q or r.
(re)	It is utilized to group the Regular expressions and recollects the text that are matched.
(?: re)	It also groups the regular expressions but does not recollects the matched text.
(?> re)	It is utilized to match self-reliant pattern in absence of backtracking.
\\w	It is used to match characters of the word.
\\W	It is used to match characters of the non-word.
\\s	It is utilized to match white spaces which are analogous to [\t\n\r\f].
\\S	It is used to match non-white spaces.
\\d	It is used to match the digits i.e, [0-9].
\\D	It is used to match non-digits.
\\G	It is used to match the point where the endmost match overs.
\\n	It is used for back-reference to occupy group number *n*.
\\b	It is used to match the word frontiers when it is out of the brackets and matches the backspace when it is in the brackets.
\\B	It is used to match non-word frontiers.
\\n, \\t, etc.	It is used to match the newlines, tabs, etc.
\\Q	It is used to escape (quote) each of the characters till \\E.
\\E	It is used in ends quoting starting with \\Q.

Example	Description
.	Match any character except newline
[Rr]uby	Match "Ruby" or "ruby"
rub[ye]	Match "ruby" or "rube"

[aeiou]	Match any one lowercase vowel
[0-9]	Match any digit; same as [0123456789]
[a-z]	Match any lowercase ASCII letter
[A-Z]	Match any uppercase ASCII letter
[a-zA-Z0-9]	Match any of the above
[^aeiou]	Match anything other than a lowercase vowel
[^0-9]	Match anything other than a digit
\\d	Match a digit: [0-9]
\\D	Match a nondigit: [^0-9]
\\s	Match a whitespace character: [\t\r\n\f]
\\S	Match nonwhitespace: [^ \t\r\n\f]
\\w	Match a single word character: [A-Za-z0-9_]
\\W	Match a nonword character: [^A-Za-z0-9_]
ruby?	Match "rub" or "ruby": the y is optional
ruby*	Match "rub" plus 0 or more ys
ruby+	Match "rub" plus 1 or more ys
\\d{3}	Match exactly 3 digits
\\d{3,}	Match 3 or more digits
\\d{3,5}	Match 3, 4, or 5 digits
\\D\\d+	No group: + repeats \\d
(\\D\\d)+/	Grouped: + repeats \\D\d pair
([Rr]uby(,)?)+	Match "Ruby", "Ruby, ruby, ruby", etc.

Intermediate Stage

1. Example 1

```
stringToFind: String = Hello i am Able to do it, abl11 able able0

scala> val pattern = "abl[ae]\\d+".r
pattern: scala.util.matching.Regex = abl[ae]\d+

scala> pattern findAllIn stringToFind toArray
warning: there were 1 feature warning(s); re-run with -feature for details
res38: Array[String] = Array(able0)

scala>

scala> val pattern = "abl[ae]\\d*".r
pattern: scala.util.matching.Regex = abl[ae]\d*

scala> pattern findAllIn stringToFind toArray
warning: there were 1 feature warning(s); re-run with -feature for details
res39: Array[String] = Array(able, able0)

scala> val pattern = "[Aa]bl[ae]\\d*".r
pattern: scala.util.matching.Regex = [Aa]bl[ae]\d*
```

```
scala> pattern findAllIn stringToFind toArray
warning: there were 1 feature warning(s); re-run with -feature for details
res40: Array[String] = Array(Able, able, able0)

scala> val pattern = "(A|a)bl[ae]\\d*".r
pattern: scala.util.matching.Regex = (A|a)bl[ae]\d*

scala> pattern findAllIn stringToFind toArray
warning: there were 1 feature warning(s); re-run with -feature for details
res41: Array[String] = Array(Able, able, able0)
```

2. Example 2

```
scala> val pattern = "(-)?(\\d+)(\\.\\d*)?".r  //listen to video at around 1 hour 40
mins//
pattern: scala.util.matching.Regex = (-)?(\d+)(\.\d*)?

scala> val stringToFind = "-1.5 divide by 5 is 3 is wrong"
stringToFind: String = -1.5 divide by 5 is 3 is wrong

scala> pattern findAllIn stringToFind toArray
warning: there were 1 feature warning(s); re-run with -feature for details
res44: Array[String] = Array(-1.5, 5, 3)

scala>

scala>

scala> val pattern = """(-)?(\d+)(\.\d*)?""".r
pattern: scala.util.matching.Regex = (-)?(\d+)(\.\d*)?

scala> pattern findAllIn stringToFind toArray
warning: there were 1 feature warning(s); re-run with -feature for details
res45: Array[String] = Array(-1.5, 5, 3)
```

3. More difficult example by extracting the value from regular expression

```
scala> val Decimal = """(-)?(\d+)(\.\d*)?""".r
Decimal: scala.util.matching.Regex = (-)?(\d+)(\.\d*)?

scala> val Decimal(sign, integerpart, decimalpart) = "-1.23"
sign: String = -
integerpart: String = 1
decimalpart: String = .23

scala> val stringToFind = "-1.5 divide by 5 is 3 is wrong"
stringToFind: String = -1.5 divide by 5 is 3 is wrong

scala> for (Decimal(sign, integerpart, decimalpart) <- Decimal findAllIn stringToFind)
     | println("Sign is " + sign + "Integer Part is " + integerpart + " Decimal Part is " +
decimalpart)
Sign is -Integer Part is 1 Decimal Part is .5
Sign is nullInteger Part is 5 Decimal Part is null
Sign is nullInteger Part is 3 Decimal Part is null
```