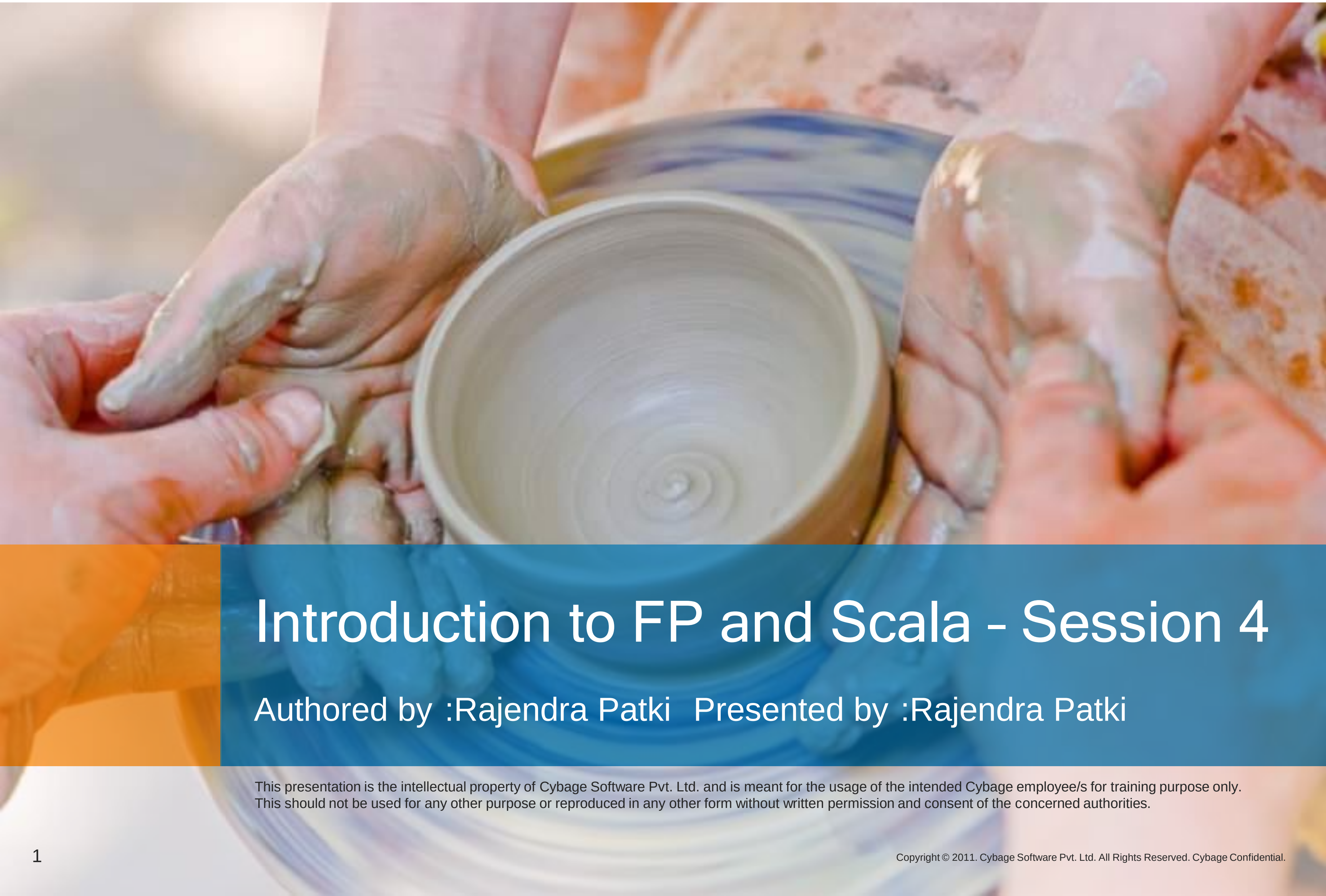

Introduction to FP and Scala - Session 4

Authored by :Rajendra Patki Presented by :Rajendra Patki

Revision

- Class & Objects
  - Getters & setters
  - Access modifier
  - Singleton & Factory
  - Inheritance
- Pattern matching
  - Wild card
  - Constant
  - Variable
  - Sequence

Agenda

- Pattern matching continued
- Case classes
- Option
- Function literals and values
- Partially applied function

Kinds of pattern 4

- Requirement

Write a function that takes an object and returns it's size depending on the actual type.

```
int getSize (Object o) {
    int size = 0;

    if (o instanceof String){
        String s = (String) o;
        size = s.length();
    }
     else if (o instanceof List) {
        List l = (List) o;
        size = l.size();
    }
    else if (o instanceof StringBuffer) {
        StringBuffer sb = (StringBuffer) o;
        size = sb.length();
    }
    else
      size = -1;
return size;
```

- Typed Pattern

```
def size (x : Any) : Int = x match {
    case s : String => s.length
    case l : List[_] => l.size
    case m : Map[_, _] => m.size
    case sb : StringBuffer => sb.length
    case _  => -1
  }
```

Kinds of pattern 5

- **Requirement**

Select a list using sequence pattern and display the contents of the list.

- Variable binding

  It performs pattern match as normal and if pattern succeeds then it sets the variable to the matched object just as with simple variable pattern

```
def listMatch1 (list : List[Int]) = list match {
     case l @ List(0,1,_,_) => println ("List " +  l)
     case _ => println("No valid list")
   }
```

Kinds of pattern 6

- Requirement

You want to add qualifying logic to a case statement in a match expression, i.e. if that pattern matches you want evaluate some additional criteria.

- Pattern Guard

```
def matchGurd (num : Int) = num match {
   case x if x == 1 => println("one, a lonely number")
   case x if (x == 2 || x == 3) => println(x)
   case _ => println("some other value")
   }
```

Kind of pattern : Summary

Pattern	Comment
Wild card	Matches any object, denoted with (_)
Constant	Matches any literal value
Variable	Matches any object, variable is bound
Sequence	Selects a particular sequence
Typed	Selected based of type, variable is bound
Variable binding	Does pattern matching normally and variable is bound with selected pattern
Pattern guard	Selects a pattern after evaluating condition

Case classes

- Case classes are created with adding “case” modifier

```
Ex : case class CaseTest (arg1:Int, arg2:Int)
```

- Case modifier adds syntactic conveniences
  - All the parameter list implicitly get val prefix
  - Factory method with name of the class
  - Natural implementation of methods toString, hashCode and equals
  - Adds copy method for making modified copies

Pattern matching with Case classes

Requirement

- Build an expression evaluator to support
  - Addition of two numbers
  - Multiplication of two numbers
  - Set expression as Negative

Pattern matching with Case classes

```
abstract class Expression
case class Const(value : Int) extends Expression
case class Add(left : Expression, right : Expression) extends Expression
case class Multiply(left : Expression, right : Expression) extends Expression
case class Negative(expr : Expression) extends Expression

def eval(expression : Expression) : Int = expression match {
  case Const(cst) => cst
  case Add(left, right) => eval(left) + eval(right)
  case Multiply(left, right) => eval(left) * eval(right)
  case Negative(expr) => - eval(expr)
}
```

- It is called as constructor pattern

Assignment - 2 (Pattern matching)

1) Write parseArgument function with following

if argument is

    -h or -help call displayHelp ()

    -v or -version call displayVersion ()

else call displayError()

2) Write a function to use regular expression using pattern matching

3) Write a function to calculate factorial of a number using pattern matching.

4) Write a function to calculate sum of all elements in List[Int] using pattern matching

Any Questions?

Thank you!