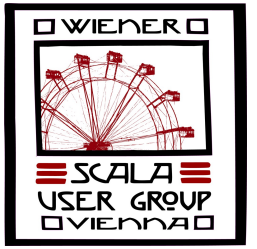
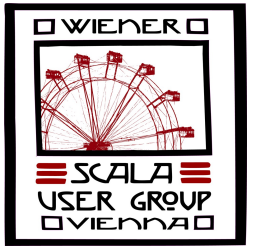
Parser Combinators easy to use ???



Define what you want to parse

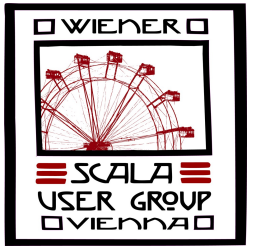
```
a = (1 2)
b =hallo
hallo=b
c = 123456
```



```
a = (1 2)
b =hallo
hallo=b
c = 123456
```

What will be the result of parsing ?

```scala
case class Prop(name: String, value: Value)

trait Value

case class SingleVal(value: String) extends Value

case class ListVal(value: List[String]) extends Value
```



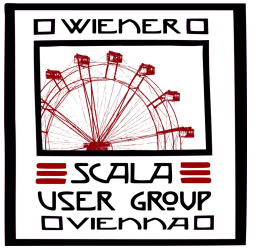
```
a = (1 2)
b =hallo
hallo=b
c = 123456
```

```scala
case class Prop(name: String, value: Value)

trait Value

case class SingleVal(value: String) extends Value

case class ListVal(value: List[String]) extends Value
```

# What does the BNF look like

BNF: Backus-Naur Form

```
props     ::= [prop]
prop      ::= name '=' value
value     ::= listVal | singleVal
listVal   ::= '(' [aVal] ')'
singleVal ::= aVal
aVal      ::= '\w+'
name      ::= '\w+'
```



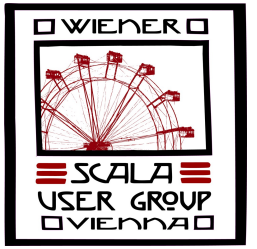
```
a = (1 2)
b =hallo
hallo=b
c = 123456
```

```
props     ::= [prop]
prop      ::= name '=' value
value     ::= listVal | singleVal
listVal   ::= '(' [aVal] ')'
singleVal ::= aVal
aVal      ::= '\w+'
name      ::= '\w+'
```

```scala
case class Prop(name: String, value: Value)

trait Value

case class SingleVal(value: String) extends Value

case class ListVal(value: List[String]) extends Value
```

# BNF-Rules to Parser Combinators

```scala
def props: Parser[List[Prop]]    = rep1(prop)                ^^ {case v => v}

def prop: Parser[Prop]           = name ~ "=" ~ value        ^^ {case n ~ _ ~ v => Prop(n, v)}

def value: Parser[Value]         = listVal | singleVal       ^^ {case v => v }

def listVal: Parser[ListVal]     = "(" ~ rep1(aVal) ~ ")"    ^^ {case _ ~ v ~ _ => ListVal(v)}

def singleVal: Parser[SingleVal] = aVal                      ^^ {case v => SingleVal(v)}

def aVal : Parser[String]        = """\w+""".r               ^^ {case v => v}

def name: Parser[String]         = """\w+""".r               ^^ {case v => v}
```



```
a = (1 2)
b =hallo
hallo=b
c = 123456
```

```
props     ::= [prop]
prop      ::= name '=' value
value     ::= listVal | singleVal
listVal   ::= '(' [aVal] ')'
singleVal ::= aVal
aVal      ::= '\w+'
name      ::= '\w+'
```

```scala
case class Prop(name: String, value: Value)

trait Value

case class SingleVal(value: String) extends Value

case class ListVal(value: List[String]) extends Value
```

```scala
def props: Parser[List[Prop]]    = rep1(prop)                ^^ {case v => v}
def prop: Parser[Prop]           = name ~ "=" ~ value        ^^ {case n ~ _ ~ v => Prop(n, v)}
def value: Parser[Value]         = listVal | singleVal       ^^ {case v => v }
def listVal: Parser[ListVal]     = "(" ~ rep1(aVal) ~ ")"    ^^ {case _ ~ v ~ _ => ListVal(v)}
def singleVal: Parser[SingleVal] = aVal                      ^^ {case v => SingleVal(v)}
def aVal : Parser[String]        = """\w+""".r               ^^ {case v => v}
def name: Parser[String]         = """\w+""".r               ^^ {case v => v}
```

# How to call it

```scala
def parse(in: String): List[Prop] = {
    parseAll(props, in) match {
        case Success(re, _) => re
        case failure: NoSuccess => {
            val msg = "Could not parse grammar. %s" format (failure)
            throw new IllegalStateException(msg)
        }
    }
}
```

Not to forget: extend scala.util.parsing.combinator.RegexParsers