

The logo of Högskolan i Skövde is a circular emblem. It features a central shield with a stylized tree, a book, and a sun. The shield is surrounded by a laurel wreath. Below the wreath, the year '1977' is inscribed.

Functional data structures 1

HÖGSKOLAN
I SKÖVDE

Functional data structures

A combination of functional programming with OOP.

This means we create data structures (e.g. collections) which are immutable.

HÖGSKOLAN
I SKÖVDE

MyList

```
val a = List(1, 2, 3)
```

a.head

a.tail



1977

HÖGSKOLAN I SKÖVDE

MyList

```
class MyList (val head: Int, val tail: ???)
```



HÖGSKOLAN
I SKÖVDE

MyList

```
abstract class MyList
```

```
class NonEmpty (val head: Int, val tail: MyList) extends MyList
```

```
val a = new NonEmpty(1, ???)
```

The logo of Högskolan i Skövde is a circular emblem. It features a central shield with a crown on top. The shield is divided into four quadrants: the top-left contains a star, the top-right contains a lion, the bottom-left contains a castle tower, and the bottom-right contains a tree. The shield is surrounded by a laurel wreath. Below the shield, the year '1977' is inscribed.

HÖGSKOLAN
I SKÖVDE

MyList

```
abstract class MyList  
class NonEmpty (val head: Int, val tail: MyList) extends MyList  
object Empty extends MyList  
  
val a = new NonEmpty(1, Empty)
```

The logo of the University of Skövde is centered in the background. It features a shield with a crown on top, flanked by two lions. The shield is surrounded by a laurel wreath. Below the shield is the year '1977'.

HÖGSKOLAN
I SKÖVDE

MyList

```
abstract class MyList
class NonEmpty (val head: Int, val tail: MyList) extends MyList
object Empty extends MyList

val a = new NonEmpty(1, Empty)
val b = new NonEmpty(1, new NonEmpty(2, Empty))
b.tail.head // = ???
```

HÖGSKOLAN
I SKÖVDE

MyList

```
abstract class MyList { def head: Int, def tail: MyList }  
class NonEmpty (val head: Int, val tail: MyList) extends MyList  
object Empty extends MyList {  
  def head = ???  
  def tail = ???  
}
```

HÖGSKOLAN
I SKÖVDE

MyList

```
abstract class MyList { def head: Int, def tail: MyList }  
class NonEmpty (val head: Int, val tail: MyList) extends MyList  
object Empty extends MyList {  
    def head = throw new Exception("Empty has no head")  
    def tail = throw new Exception("Empty has no tail")  
}
```

1977
HÖGSKOLAN
I SKÖVDE

MyList

```
abstract class MyList {  
  def head: Int  
  def tail: MyList  
  def :: (a: Int): MyList  
}
```



HÖGSKOLAN
I SKÖVDE

MyList

```
class NonEmpty (val head: Int, val tail: MyList) extends MyList {  
    def :: (a: Int): MyList = ???  
}  
  
object Empty extends MyList {  
    def head = throw new Exception("Empty has no head")  
    def tail = throw new Exception("Empty has no tail")  
    def :: (a: Int): MyList = ???  
}
```

HOGSKOLAN
I SKÖVDE

MyList

```
class NonEmpty (val head: Int, val tail: MyList) extends MyList {  
  def :: (a: Int): MyList = new NonEmpty(a, this)  
}  
  
object Empty extends MyList {  
  def head = throw new Exception("Empty list")  
  def tail = throw new Exception("Empty list")  
  def :: (a: Int): MyList = new NonEmpty(a, Empty)  
}
```

HOGSKOLAN
I SKÖVDE

Option pattern

```
abstract class MyList { ...  
    def find (f: Int => Boolean): Int  
}  
  
class NonEmpty (val head: Int, val tail: MyList) extends MyList { ...  
    def find (f: Int => Boolean): Int = ???  
}
```

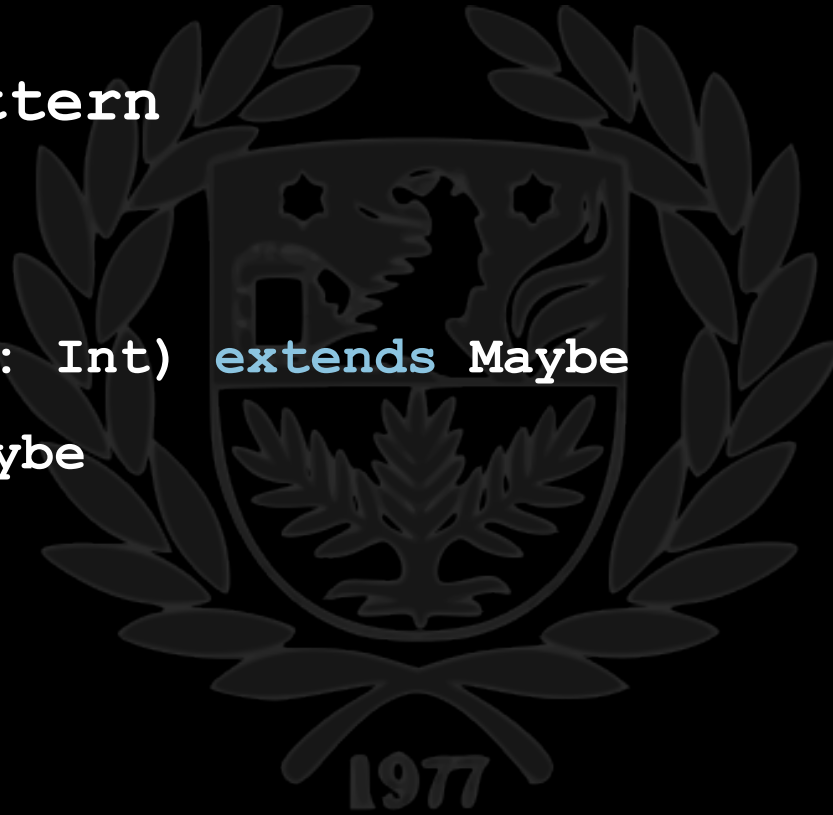
1977
HÖGSKOLAN
I SKÖVDE

Option pattern

```
abstract class MyList { ...  
    def find (f: Int => Boolean): Maybe  
}  
  
class NonEmpty (val head: Int, val tail: MyList) extends MyList { ...  
    def find (f: Int => Boolean): Maybe = ???  
}  
  
object Empty extends MyList { ...  
    def find (f: Int => Boolean): Maybe = ???  
}
```

Option pattern

```
trait Maybe  
class Yes (val value: Int) extends Maybe  
object No extends Maybe
```



HÖGSKOLAN
I SKÖVDE

Option pattern

```
trait Maybe
class Yes (val value: Int) extends Maybe
object No extends Maybe

val res = 1.find(_ % 2 == 0) match {
  case obj: Yes => s"Even number ${obj.value} found"
  case No => "No even numbers found"
}
```

HOGSKOLAN
I SKÖVDE

Option pattern

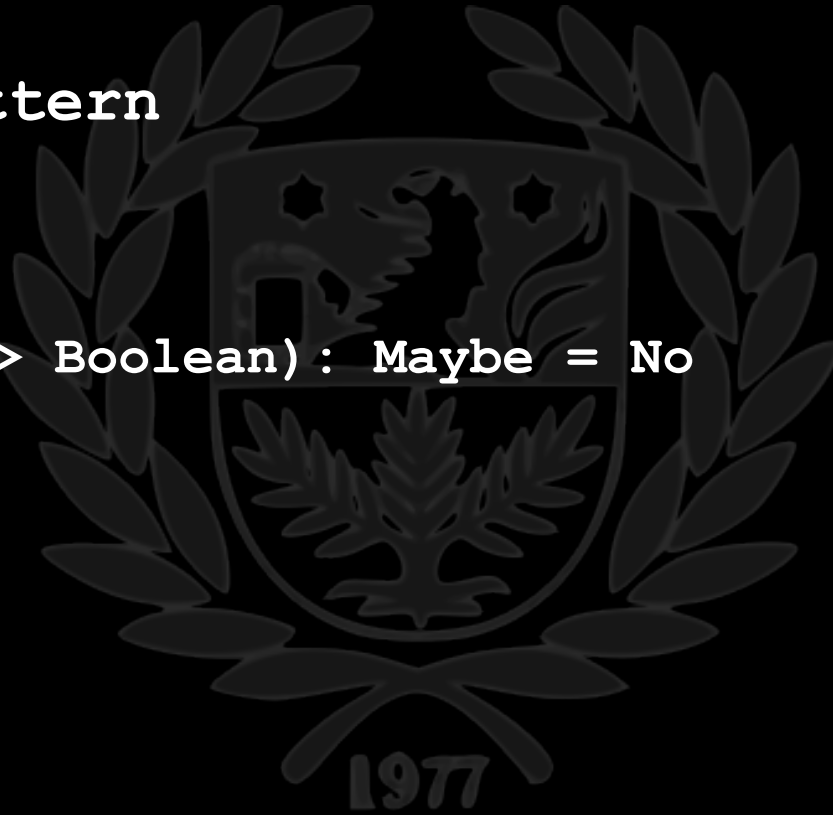
```
class NonEmpty (...) {  
  def find (f: Int => Boolean): Maybe = {  
    if (f(head)) new Yes(head)  
    else tail.find(f)  
  }  
}
```

The logo of Högskolan i Skövde is a circular emblem. It features a central shield with a crown on top. The shield is divided into four quadrants: the top-left contains a star, the top-right contains a lion, the bottom-left contains a castle tower, and the bottom-right contains a tree. The shield is surrounded by a laurel wreath. Below the wreath, the year '1977' is inscribed.

HÖGSKOLAN
I SKÖVDE

Option pattern

```
object Empty {  
  def find (f: Int => Boolean): Maybe = No  
}
```



HÖGSKOLAN
I SKÖVDE

Either pattern

```
def pascal(r: Int, c: Int): Int =  
  if (r == c || c == 1) 1  
  else pascal(r - 1, c - 1) + pascal(r - 1, c)
```

```
pascal(5, 4)      // = 4
```

	1	2	3	4	5	6
1	1					
2	1	1				
3	1	2	1			
4	1	3	3	1		
5	1	4	6	4	1	
6	1	5	10	10	5	1

Either pattern

```
def pascal(r: Int, c: Int): Int =  
  if (r == c || c == 1) 1  
  else pascal(r - 1, c - 1) + pascal(r - 1, c)
```

`pascal(4, 5)` `// = ???`

	1	2	3	4	5	6
1	1					
2	1	1				
3	1	2	1			
4	1	3	3	1		
5	1	4	6	4	1	
6	1	5	10	10	5	1

Either pattern

```
def pascal(r: Int, c: Int): ??? =  
  if (c > r) ???  
  else if (r == c || c == 1) 1  
  else pascal(r - 1, c - 1) + pascal(r - 1, c)
```

```
pascal(4, 5)    // = ???
```

	1	2	3	4	5	6
1	1					
2	1	1				
3	1	2	1			
4	1	3	3	1		
5	1	4	6	4	1	
6	1	5	10	10	5	1

Either pattern

```
trait ThisOrThat
class This(val value: Int) extends ThisOrThat
class That(val value: String) extends ThisOrThat

val res = pascal(5, 4) match {
  case obj: This => s"Value is ${obj.value}"
  case obj: That => obj.value
}
```

HOGSKOLAN
I SKÖVDE

Either pattern

```
def pascal(r: Int, c: Int): ThisOrThat =  
  if (c > r) new That(s"No value at row $r, col $c")  
  else if (r == c || c == 1) new This(1)  
  else (pascal(r - 1, c - 1), pascal(r - 1, c)) match {  
    case (a: This, b: This) => new This(a.value + b.value)  
    case _ => new That("Something went wrong")  
  }
```

HÖGSKOLAN
I SKÖVDE

Case classes

```
case class Person(name: String)
```

```
val jenny = Person("Jenny")
```

```
jenny.name // = Jenny
```

The logo of Högskolan i Skövde is a circular emblem. It features a central shield with a crown on top. The shield is divided into four quadrants: the top-left has a star, the top-right has a lion, the bottom-left has a tree, and the bottom-right has a flower. The shield is surrounded by a laurel wreath. Below the wreath, the year '1977' is inscribed.

HÖGSKOLAN
I SKÖVDE

Case classes

```
case class Person(name: String)
```

```
val jenny = Person("Jenny")
```

```
val otherJenny = Person("Jenny")
```

```
jenny == otherJenny // = true
```

HÖGSKOLAN
I SKÖVDE

Case classes

```
case class Address(street: String, postnr: Int)
case class Person(name: String, addr: Address)

val jenny = Person("Jenny", Address("Hogskolevagen", 54128))
val otherJenny = Person("Jenny", Address("Hogskolevagen", 54128))

jenny == otherJenny // = true
```

HÖGSKOLAN
I SKÖVDE

Case classes

```
case class Address(street: String, postnr: Int)
case class Person(name: String, addr: Address)

val jenny = Person("Jenny", Address("Hogskolevagen", 54128))
val otherJenny = Person("Jenny", Address("Hogskolevagen", 54123))

jenny == otherJenny    // = false
```

HÖGSKOLAN
I SKÖVDE

Case classes

```
case class Address(street: String, postnr: Int)
case class Person(name: String, addr: Address)

val jenny = Person("Jenny", Address("Hogskolevagen", 54128))
val street = jenny match {
  case Person(_, Address(s, _)) => s
}
```

HÖGSKOLAN
I SKÖVDE

Algebraic data types (ADT)

An ADT "is the sum or union of its data constructors
and each data constructor is the product of its arguments,
hence the name algebraic data type"*

*Chiusano, P., & Bjarnason, R. (2014). Functional programming in Scala. Manning Publications Co..

Algebraic data types (ADT)

A type composed of other types

e.g.

suit = spade | dimond | heart | club

HÖGSKOLAN
I SKÖVDE

Option pattern is an ADT

```
trait Maybe
```

```
class Yes (val value: Int) extends Maybe
```

```
object No extends Maybe
```

The logo of Högskolan i Skövde is a circular emblem. It features a central shield with a crown on top. The shield is divided into four quadrants: top-left has a star, top-right has a star, bottom-left has a castle tower, and bottom-right has a tree. The shield is surrounded by a laurel wreath. Below the wreath is the year '1977'.

HÖGSKOLAN
I SKÖVDE

Option pattern is an ADT

```
sealed trait Maybe
```

```
class Yes (val value: Int) extends Maybe
```

```
object No extends Maybe
```



HÖGSKOLAN
I SKÖVDE

Option pattern is an ADT

```
sealed trait Maybe
class Yes (val value: Int) extends Maybe
object No extends Maybe

val res = 1.find(_ % 2 == 0) match {
  case obj: Yes => s"Even number ${obj.value} found"
  case No => "No even numbers found"
}
```

HOGSKOLAN
I SKÖVDE

Option pattern is an ADT

```
sealed trait Maybe
case class Yes (value: Int) extends Maybe
object No extends Maybe

val res = 1.find(_ % 2 == 0) match {
  case Yes(v) => s"Even number $v found"
  case No => "No even numbers found"
}
```

HOGSKOLAN
I SKÖVDE

MyList is an ADT

```
abstract class MyList { def head: Int; def tail: MyList; ... }  
class NonEmpty (val head: Int, val tail: MyList) extends MyList {  
  def :: (a: Int): MyList = new NonEmpty(a, this)  
}  
object Empty extends MyList {  
  def head = throw new Exception("Empty list")  
  def tail = throw new Exception("Empty list")  
  def :: (a: Int): MyList = new NonEmpty(a, Empty)  
}
```

MyList is an ADT

```
sealed abstract class MyList { def head: Int; def tail: MyList; ... }  
class NonEmpty (val head: Int, val tail: MyList) extends MyList {  
  def :: (a: Int): MyList = new NonEmpty(a, this)  
}  
object Empty extends MyList {  
  def head = throw new Exception("Empty list")  
  def tail = throw new Exception("Empty list")  
  def :: (a: Int): MyList = new NonEmpty(a, Empty)  
}
```

HOGSKOLAN
I SKÖVDE

MyList is an ADT

```
sealed abstract class MyList { def head: Int; def tail: MyList; ... }
case class NonEmpty (val head: Int, val tail: MyList) extends MyList {
  def :: (a: Int): MyList = new NonEmpty(a, this)
}
object Empty extends MyList {
  def head = throw new Exception("Empty list")
  def tail = throw new Exception("Empty list")
  def :: (a: Int): MyList = new NonEmpty(a, Empty)
}
```

HOGSKOLAN
I SKÖVDE

MyList is an ADT

```
val ns = 1 :: 2 :: 3 :: Empty
```

```
def sum(l: MyList): Int = l match {  
  case _: Empty => 0  
  case a: NonEmpty => a.head + sum(a.tail)  
}
```

HÖGSKOLAN
I SKÖVDE

MyList is an ADT

```
val ns = 1 :: 2 :: 3 :: Empty
```

```
def sum(l: MyList): Int = l match {  
  case Empty => 0  
  case NonEmpty(h, t) => h + sum(t)  
}
```

The logo of the University of Skövde is centered in the background. It features a shield with a crown on top, flanked by two lions. The shield is surrounded by a laurel wreath. Below the shield is the year '1977'.

HÖGSKOLAN
I SKÖVDE