Tuples, collections and pattern matching

Tuples

```
(1, 2)
("hello", "world")
("hello", 2)
```

HÖGSKOLAN I SKÖVDE

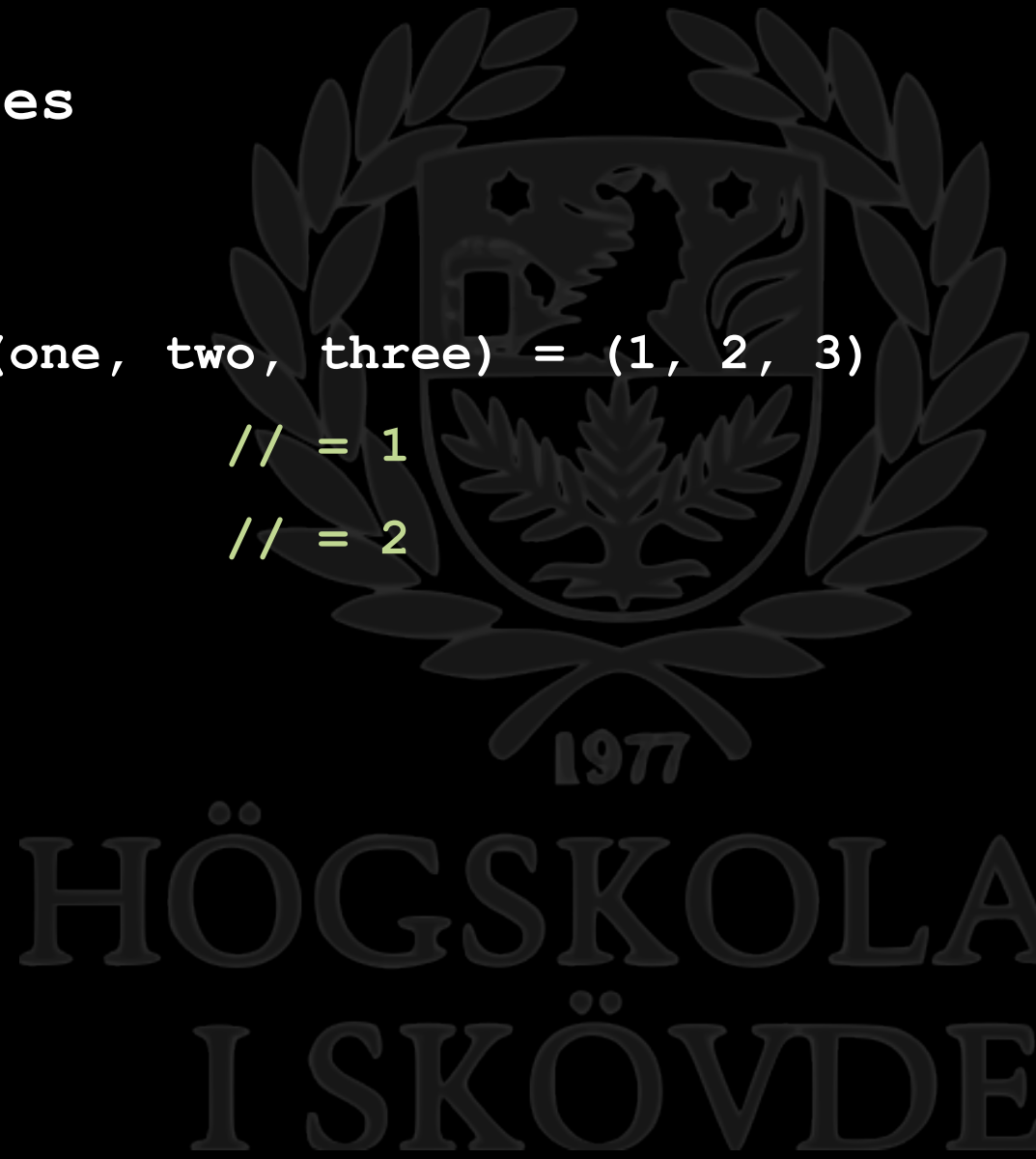
Tuples

```
val t1 = (1, 2)
val t2 = ("hello", "world")
t1._1          // = 1
t2._2          // = "world"
```

1977
HÖGSKOLAN
I SKÖVDE

Tuples

```
val (one, two, three) = (1, 2, 3)
one           // = 1
two           // = 2
```



Vectors

```scala
val a: Vector[Int] = Vector(1, 2, 3)
a.head        // = 1
a.tail        // = Vector(2, 3)
a(2)          // = 3
```

1977
HÖGSKOLAN
I SKÖVDE

Array

```scala
val a: Array[Int] = Array(1, 2, 3)
a.head          // = 1
a.tail          // = Array(2, 3)
a(2)            // = 3
a(2) = 5
a(2)            // = 5
```

1977
HÖGSKOLAN
I SKÖVDE

Lists

```scala
val a: List[Int] = List(1, 2, 3)
a.head       // = 1
a.tail       // = List(2, 3)
```

1977
HÖGSKOLAN
I SKÖVDE

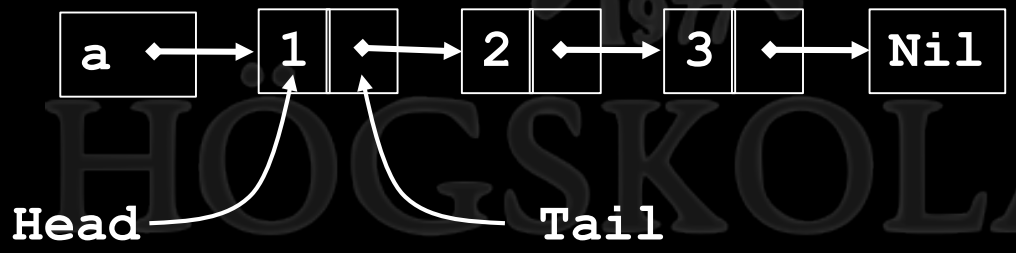
Lists

```scala
val a: List[Int] = 1 :: 2 :: 3 :: Nil
a.head       // = 1
a.tail       // = List(2, 3)
```

1977
HÖGSKOLAN
I SKÖVDE

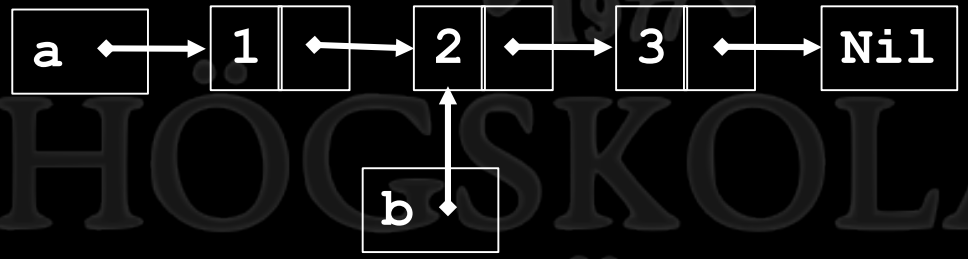
Lists

```
val a: List[Int] = 1 :: 2 :: 3 :: Nil
```



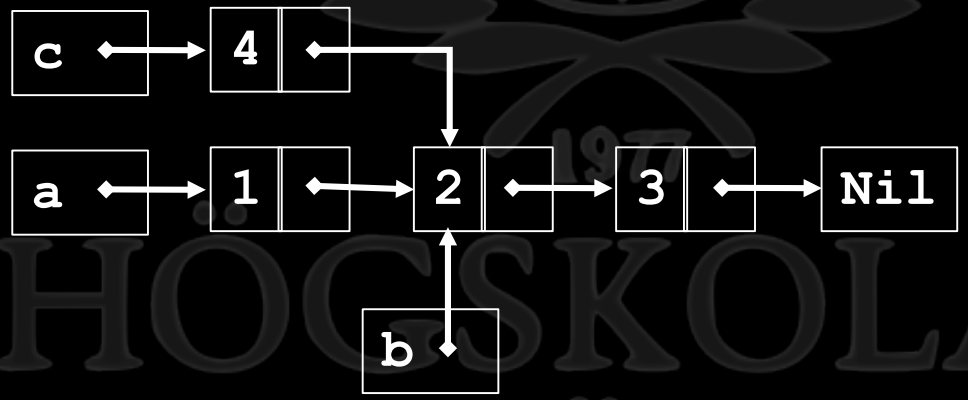
Lists

```
val a: List[Int] = 1 :: 2 :: 3 :: Nil
val b = a.tail
```



Lists

```
val a: List[Int] = 1 :: 2 :: 3 :: Nil
val b = a.tail
val c = 4 :: b
```



Recursion with lists

```
def sum(l: List[Int]): Int = {
  if (l.isEmpty) 0
  else l.head + sum(l.tail)
}
```

1977
HÖGSKOLAN
I SKÖVDE

Pattern matching: constant patterns

```
val num: Int = …
val str = num match {
  case 0 => "zero"
  case 1 => "one"
  case 2 => "two"
  case _ => "something else"
}
```

Wildcard

HÖGSKOLAN I SKÖVDE 1977

Pattern matching: constant patterns

```
val eng: String = …
val swe = eng match {
  case "Hello" => "Hej"
  case "Good bye" => "Hej då"
  case "Not too much, not too little" => "Lagom"
  case _ => "Not in dictionary"
}
```

HÖGSKOLAN I SKÖVDE

Pattern matching: constant patterns

```scala
def describe(x: Any): String = x match {
  case true => "true"
  case Nil => "empty list"
  case 5 => "five"
  case '5' => "five"
  case _ => "Something else"
}
```

HÖGSKOLAN I SKÖVDE 1977

Pattern matching: typed patterns

```
def describe(x: Any): String = x match {
  case a: Int => a + " is an integer"
  case a: Double => a + " is a double"
  case a: Boolean => a + "is a boolean"
  case a: String => a + " is a string"
  case _ => "unknown type"
}
```

HÖGSKOLAN I SKÖVDE

Pattern matching: guards

```
def describe(x: Any): String = x match {
  case n: Int if n == 0 => "zero"
  case n: Int if n < 0 => "negative"
  case n: Int if n > 0 => "positive"
  case _ => "not an int"
}
```

1977
HÖGSKOLAN I SKÖVDE

Pattern matching: tuple patterns

```
val t: (Int, Int, Int) = …
val sum = t match {
  case (a, b, c) => a + b + c
}
```

1977
HÖGSKOLAN
I SKÖVDE

Pattern matching: tuple patterns

```
val t: (Int, Int, Int) = …
val sum = t match {
  case (a, b, _) => a + b
}
```

1977
HÖGSKOLAN
I SKÖVDE

Pattern matching: tuple patterns

```
def describe(t:(Boolean, Boolean)): String = t match {
  case (true, true) => "Both true"
  case (true, false) => "First true"
  case (false, true) => "Second true"
  case (false, false) => "None true"
}
```

HÖGSKOLAN I SKÖVDE

Pattern matching: sequence patterns

```
val ns: List[Int] = …
val sum = ns match {
  case List() => 0
  case List(a) => a
  case List(a, b) => a + b
  case List(a, b, c) => a + b + c
}
```

1977 HÖGSKOLAN I SKÖVDE

Pattern matching: sequence patterns

```
val ns: List[Int] = …
val sum = ns match {
  case Nil => 0
  case a :: Nil => a
  case a :: b :: Nil => a + b
  case a :: b :: c :: Nil => a + b + c
}
```

HÖGSKOLAN I SKÖVDE

Recursion with lists

```
def sum(l: List[Int]): Int = {
  if (l.isEmpty) 0
  else l.head + sum(l.tail)
}
```

1977
HÖGSKOLAN
I SKÖVDE

Recursion with lists: with pattern match

```
def sum(l: List[Int]): Int = l match {
  case Nil => 0
  case h :: t => h + sum(t)
}
```

1977
HÖGSKOLAN
I SKÖVDE

Recursion with lists: with pattern match

```
val sum:(List[Int] => Int) = {
  case Nil => 0
  case h :: t => h + sum(t)
}
```

1977
HÖGSKOLAN
I SKÖVDE

Built-in functions

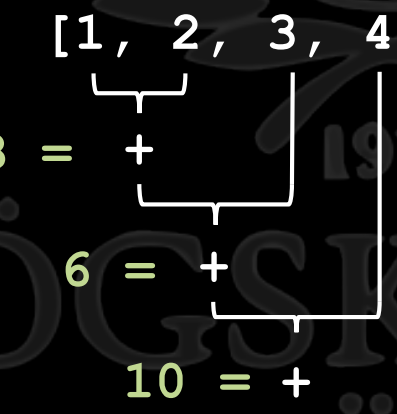
```scala
val a = List(1, 2, 3, 4)
a.filter(x => x % 2 == 0) // = List(2, 4)
a.map(x => x + x) // = List(2, 4, 6, 4)
a.reduce((a, b) => a + b)     // = 10
a.fold(10)((a, b) => a + b)   // = 20
a.zip(List(5, 6, 7, 8)) // = List((1, 5), (2, 6), (3, 7), (4, 8))
```

scala-lang.org/api/current/scala/List.html

Built-in functions: reduce

```
List(1, 2, 3, 4).reduce((a, b) => a + b)
```

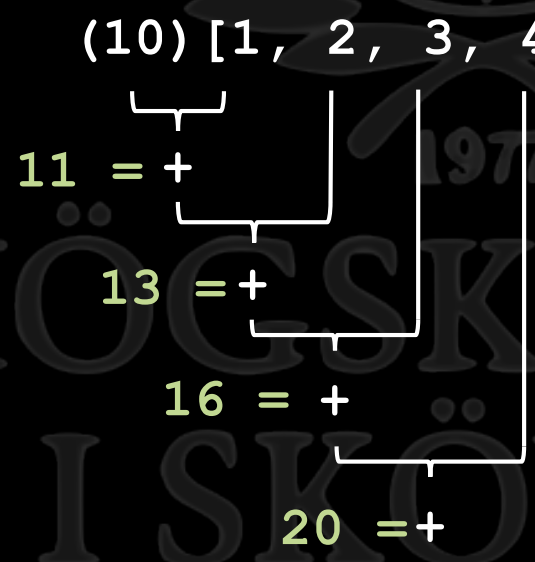
```
(((1 + 2) + 3) + 4)
```



Built-in functions: fold

```
List(1, 2, 3, 4).fold(10)((a, b) => a + b)
```

```
((((0 + 1) + 2) + 3) + 4)
```



Built-in functions: foldLeft

```
List(1, 2, 3, 4).foldLeft("-->")((a, b) => a + b)

((("-->" + 1) + 2) + 3) + 4)
// = "-->1234"
```

1977
HÖGSKOLAN
I SKÖVDE

Built-in functions: foldRight

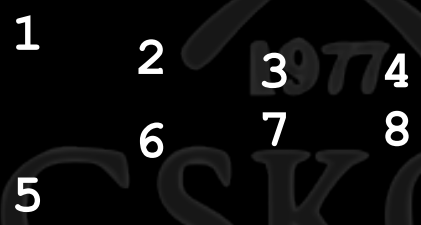
```
List(1, 2, 3, 4).foldRight("-->")((a, b) => a + b)

          (1 + (2 + (3 + (4 + "-->"))))
          // = "1234-->"
```

1977
HÖGSKOLAN
I SKÖVDE

Built-in functions: zip

```
val a = List(1, 2, 3, 4)
val b = List(5, 6, 7, 8)
a.zip(b) // = List((1, 5), (2, 6), (3, 7), (4, 8))
```

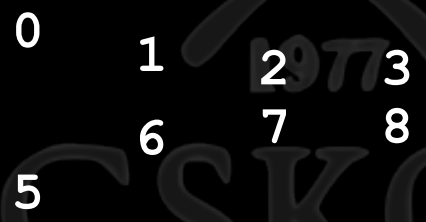


HÖGSKOLAN I SKÖVDE

Built-in functions: zipWithIndex

```
val a = List(5, 6, 7, 8)

a.zipWithIndex // = List((5, 0), (6, 1), (7, 2), (8, 3))
```



HÖGSKOLAN I SKÖVDE