Objekt-Funktional
Mutable & Immutable

Objekt-Funktional
Motivation

- **bisher:** rein funktional in Scala programmiert
- **in Scala:** auch objektorientierte Programmierung möglich
- **Übergang von rein funktional -> objekt-funktional am**
- **Bsp.:** Gleichheit zweier Objekte
- **funktional:** `equals(xs,ys)`
- **objekt-orientiert:** `xs.equals(ys) //Punkt kann weg`

Objekt-Funktional
rein funktional -> objekt-funktional

- **primitive Regel:**
  Wähle Parameter mit aktuellem Objekt und ziehe diesen vor Punkt.
- Gibt auch Sprachen, bei denen aktuelles Objekt nicht vor Punkt, sondern als 1. Argument, z.B.: **Ada95**
- **auch für Higher-order Funktionen, z.B.** `filter`
- **funktional:** `List[Int] x (Int => Boolean) => List[Int]`
- **objekt-funktional:**
  - Instanzmethode der Klasse `List[Int]`
  - mit Typ `(Int => Boolean) => List[Int]`

Objekt-Funktional
Rein Funktional -> Objekt-Funktional am Beispiel

- **Wdh.:** `quicksort`
- **funktional:**

```
def sortFun(xs: List[Int]): List[Int] = {
  if (length(xs) <= 1) xs
  else {  List.concat(
            sortFun(filter(xs,xs(1) >)),
                    filter(xs,xs(1) ==),
            sortFun(filter(xs,xs(1) <)))    }
}
```

- **objekt-funktional:**

```
def sortOOFun(xs: List[Int]): List[Int] = {
  if (xs.length <= 1) xs
  else {  List.concat(
            sortOOFun(xs filter(xs(1) >)),
                      xs filter(xs(1) ==),
            sortOOFun(xs filter(xs(1) <)))    }
}
```


FH AACHEN UNIVERSITY OF APPLIED SCIENCES

Mutable/Immutable
Definition und OO-Standard

- **Def.: :** (Wikipedia)
  In object-oriented and functional programming, an **immutable object** (unchangeable[1] object) is an object whose state cannot be modified after it is created.[2] This is in contrast to a **mutable object** (changeable object), which can be modified after it is created.

- **Standard in OO: Mutable**
  - Operiere (Verändere) auf Objekten
  - Listenaufbau in Java

```
void vorhängen(Object obj) {
        Node newHead = new Node(obj);
        newHead.next = this.head;
        this.head = newHead;
}
```

Mutable/Immutable
Standard in FuPS

- **Standard in FuPS: Immutable**
  - Erzeuge neues Objekt und operiere darauf
  - Verändere altes nicht
  - Listenaufbau in Java als immutable

```
Liste vorhängen(Object obj) {
        Liste ml = this.clone();
        Node newHead = new Node(obj);
        newHead.next = ml.head;
        ml.head = newHead;
        return ml;
}
```

Mutable/Immutable
in Scala

- **Scala: Objekt-Funktional**
  -> unterstützt beides
  - Mutable: `var currentValue = 1  // Veränderbar`
  - Immutable: `val maxValue = 10  // Konstant`
- **Collection-Klassen wie** `List, Map` **Immutable**
  - jedes Mal Kopie erzeugt
  - sehr ineffizient
  - aber effizient implementiert durch Wiederverwendung von bereits verwendeten Elementen
- **Vorteile von Immutable Objekten**
  - Keine Seiteneffekte, insbesondere bei Multicores: Thread-Safe
  - einfacher zu programmieren, testen und parallelisieren
  - Interner Zustand konsistent, auch bei Exceptions
  - Referenzen cachebar, da nicht veränderlich