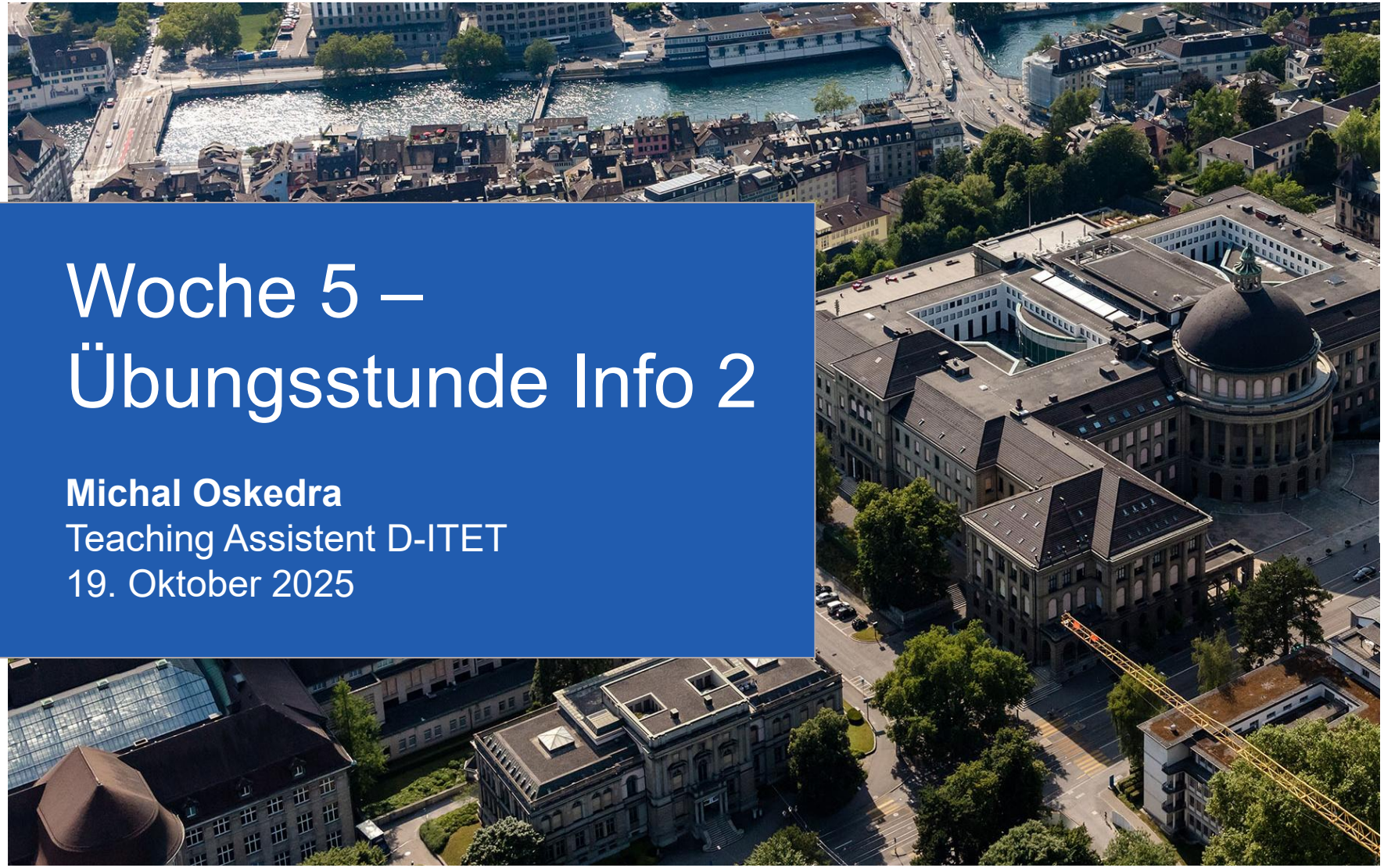
ETH zürich

Info2 - ML J 34.1 14-16

WhatsApp group



Woche 5 – Übungsstunde Info 2

Michal Oskedra
Teaching Assistent D-ITET
19. Oktober 2025

Webseite:
moskedra-web.ethz.ch

01 – Theorie

01.1 Funktionen Höherer Ordnung

01.2 Funktoren

01.3 Lambdas

Definition

Funktionen Höherer Ordnung:
Funktionen, die im Argument eine Funktion beinhalten oder eine Funktion zurückgibt

Funktionssignaturen

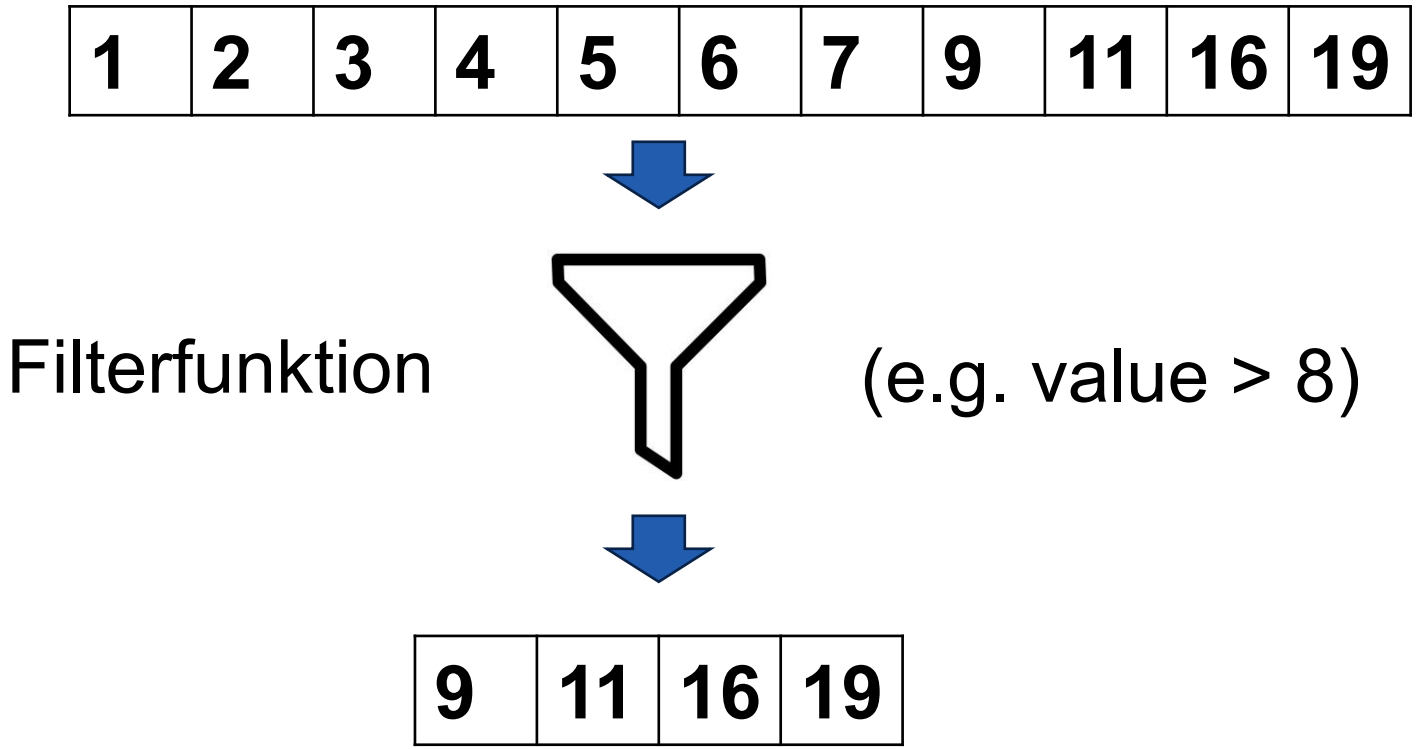
- $f_1 : A \to \text{int}$ Funktion von irgendeinem Typ zu einem Integer
- $f_2 : A \times A \times A \to \text{bool}$ Funktion von drei A's zu einem Boolean
- $f_3 : A \times (A \to B) \to B$ "higher-order function" (mit zwei Argumenten)
- $f_4 : \texttt{vec<A>} \times (A \to B) \to \texttt{vec<B>}$ higher-order function mit Vektoren
- $f_5 : (A \times A \to B) \times A \to ((A \to B) \to \text{bool})$ nimmt und gibt zurück eine Funktion

01 – Theorie
01.1 Funktionen Höherer Ordnung
01.2 Funktoren
01.3 Lambdas
01.4 Closures

Definition

Funktoren:
Funktionen mit einem gespeicherten Zustand

Beispiel: Filter design



Funktoren

```
template <typename T, typename Function>
void filter(const T& collection, Function f);
```

Funktion Höherer Ordnung

```
class GreaterThan{
  int value; // state
  public:
  GreaterThan(int x):value{x}{}

  bool operator() (int par) const {
    return par > value;
  }
};
```

Funktor:
Eine Funktion mit einem Zustand (`int value` in diesem Fall)

```
std::vector<int> a {1,2,3,4,5,6,7,9,11,16,19};
int value=8;
filter(a,GreaterThan(value)); // 9,11,16,19
```

Aufruf

Funktoren

```
template <typename T, typename Function>
void filter(const T& collection, Function f);
```

Funktion Höherer Ordnung

funktioniert auch mit Templates:

```
class GreaterThan{
    T value;
public:
    GreaterThan(T x):value{x}{}

    bool operator() (T par) const{
        return par > value;
    }
};
```

Funktor:
Eine Funktion mit einem Zustand (`int value` in diesem Fall)

```
std::vector<int> a {1,2,3,4,5,6,7,9,11,16,19};
int value=8;
filter(a,GreaterThan(value)); // 9,11,16,19
```

Aufruf

01 – Theorie
01.1 Funktionen Höherer Ordnung
01.2 Funktoren
01.3 Lambdas
01.4 Closures

Lambdas

Manchmal wollen wir eine Funktion in einer Zeile definieren…

Lambdas

Manchmal wollen wir eine Funktion in einer Zeile definieren…

```
std::vector<int> a {1,2,3,4,5,6,7,9,11,16,19};
int value=8;

filter(a, [value](int x) {return x > value;} );
```

Lambda = Definition der Funktion auf einer Zeile

Lambdas

Manchmal wollen wir eine Funktion in einer Zeile definieren…

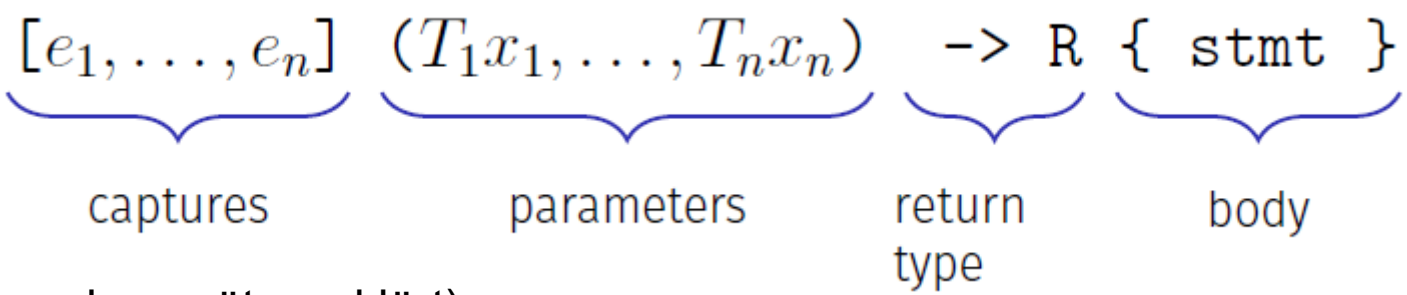
```
std::vector<int> a {1,2,3,4,5,6,7,9,11,16,19};
int value=8;

filter(a, [value](int x) {return x > value;} );
```

Definition einer Funktion auf einer Zeile

…macht genau das gleiche wie die Code von Slide 8!

Lambda Syntax



(werden später erklärt)

Beispiel:

```
auto f = [](){std::cout << "Hello World";};

f(x); // Hello World
```

Lambda Beispiel

```
auto f = [](->void {std::cout << "Hello World";};
f(x) // Hello World
```

Geht es kürzer?

Lambda Beispiel

```
auto f = [](->void {std::cout << "Hello World";};
f(x) // Hello World
```

Geht es kürzer?

- Rückgabetyp kann inferiert werden, wenn kein oder nur ein return:[7]

  ```
  [](){std::cout << "Hello World";}
  ```

- Keine Parameter und kein expliziter Rückgabetyp ⇒ () kann weggelassen werden

  ```
  []{std::cout << "Hello World";}
  ```

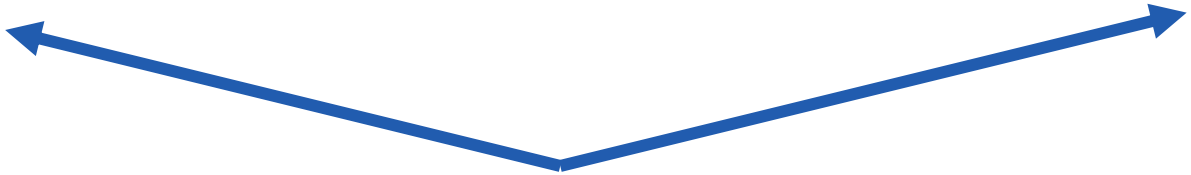
Weitere Beispiele

```
[](int x, int y) {std::cout << x * y;} (4,5);
```

Output: 20

```
int k = 8;
auto f = [](int& v) {v += v;};
f(k);
std::cout << k;
```

```
int k = 8;
auto f = [](int v) {v += v;};
f(k);
std::cout << k;
```



Was ist der Output dieser Lambdas?

Weitere Beispiele

```
[](int x, int y) {std::cout << x * y;} (4,5);
```

Output: 20

```
int k = 8;
auto f = [](int& v) {v += v;};
f(k);
std::cout << k;
```

Output: 16

```
int k = 8;
auto f = [](int v) {v += v;};
f(k);
std::cout << k;
```

Output: 8

Nützliche Funktion für die Serie

```
template< class InputIt, class UnaryFunc >
UnaryFunc for_each( InputIt first, InputIt last, UnaryFunc f );
```

Applies the given invocable object `f` to each element in the target range `[ first , last )`. If `f` returns a result, the result is ignored.

Parameters

`first, last` - the pair of iterators defining the target range

`f` - function object, to be applied to the elements

The signature of the function should be equivalent to the following:

```
void fun(const Type &a);
```

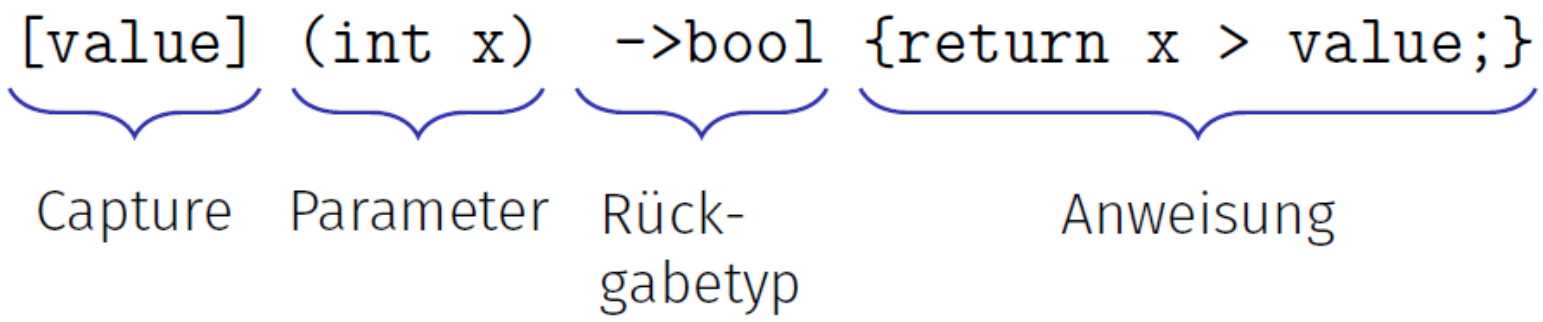
The signature does not need to have `const &`.
The type `Type` must be such that an object of type `InputIt` can be dereferenced and then implicitly converted to `Type`.

Aus https://en.cppreference.com/cpp/algorithm/for_each

01 – Theorie
01.1 Funktionen Höherer Ordnung
01.2 Funktoren
01.3 Lambdas
01.4 Closures

Closures

Closure ist ein Lambda, die ihre Umgebung speichert.



Capture definiert, welche Elemente gespeichert werden:

- `[]` kein Zugriff auf Kontext
- `[x]` `x` ist Kopiert (und `const`)
- `[&x]` `x` ist referenzierbar
- `[x, &y]` `x` ist kopiert, `y` ist referenziert
- `[&]` alle nötigen Variablen werden automatisch referenziert
- `[=]` alle nötigen Variablen werden automatisch kopiert
- `[&, x]` alle nötigen Variablen werden referenziert, nur `x` wird kopiert
- `[=, &x]` alle nötigen Variablen werden kopiert, nur `x` wird referenziert

Closures Beispiel

```
std::vector<int> a {1,2,3,4,5,6,7,9,11,16,19};

int s=0;

std::for_each(a.begin(), a.end(), [&s] (int x) {s += x;} );
s=0; //hier würde die Variable s auf 0 gesetzt
std::cout << s << std::endl; //capture speichert die variabeln zum Zeit
                             der Definition
```

Closure mit gespeicherten Variable s

02 – Quiz

Quiz Frage 1

Ein Funktor ist ein Objekt, das wie eine Funktion aufgerufen werden kann, da es den Operator () überlädt.

a) Wahr
b) Falsch

Quiz Frage 1

Ein Funktor ist ein Objekt, das wie eine Funktion aufgerufen werden kann, da es den Operator () überlädt.

a) Wahr

b) Falsch

Ein Funktor ist eine Funktion, die in einem `class`-Objekt gespeichert wird

Beispiel:

```
class GreaterThan {
  int value;
public:
  GreaterThan(int v): value(v) {}
  bool operator()(int x) const { return x > value; }
};
```

Quiz Frage 2

Was ist eine Funktion höherer Ordnung?

A) Eine Funktion, die nur auf Klassen arbeitet
B) Eine Funktion mit mehr als zwei Parametern
C) Eine virtuelle Funktion
D) Eine Funktion, die andere Funktionen als Argumente nimmt oder zurückgibt

Quiz Frage 2

Was ist eine Funktion höherer Ordnung?

A) Eine Funktion, die nur auf Klassen arbeitet
B) Eine Funktion mit mehr als zwei Parametern
C) Eine virtuelle Funktion
D) Eine Funktion, die andere Funktionen als Argumente nimmt oder zurückgibt

Funktionen höherer Ordnung nehmen andere Funktionen als Parameter oder geben sie zurück.
Beispiele sind `filter`, `std::for_each` oder `std::sort`, die Funktoren oder Lambdas als Argumente erhalten

Quiz Frage 3

Welche der folgenden Capture-Listen kopiert alle Variablen aus dem aktuellen Kontext?

A) `[&]`
B) `[=]`
C) `[this]`
D) `[]`

Quiz Frage 3

Welche der folgenden Capture-Listen kopiert alle Variablen aus dem aktuellen Kontext?

A) `[&]`
B) [=]
C) `[this]`
D) `[]`

[=] kopiert alle im Lambda verwendeten Variablen per Wert.
[&] würde alle per Referenz übergeben.
[] erlaubt keinen Zugriff auf äussere Variablen.

Quiz Frage 4

Welche Ausgabe erzeugt folgender Code?

```
int v = 42;
auto f = [=]() { std::cout << v; };
v = 7;
f();
```

A) 7
B) 42
C) Compilerfehler
D) Unbestimmt

Quiz Frage 4

Welche Ausgabe erzeugt folgender Code?

```
int v = 42;
auto f = [=]() { std::cout << v; };
v = 7;
f();
```

A) 7
B) 42
C) Compilerfehler
D) Unbestimmt

Bei `[=]` werden Variablen zum Zeitpunkt der Definition kopiert.Das Lambda merkt sich den alten Wert (42) — spätere Änderungen an `v` wirken sich nicht aus.

Quiz Frage 5

Welche Syntax erzeugt eine minimale Lambda-Funktion, die „Hello World“ ausgibt?

A) `[]{std::cout << "Hello World";};`

B) `[]->void(std::cout << "Hello World");`

C) `[=]{std::cout << "Hello World";}`

D) `lambda(){std::cout << "Hello World";}`

Quiz Frage 5

Welche Syntax erzeugt eine minimale Lambda-Funktion, die „Hello World“ ausgibt?

A) **`[]{std::cout << "Hello World";};`**

B) `[]->void(std::cout << "Hello World");`

C) `[=]{std::cout << "Hello World";}`

D) `lambda(){std::cout << "Hello World";}`

Ein Lambda besteht aus drei Teilen: `[capture](parameter) -> return { body }`.

Wenn keine Parameter oder Rückgabetypen nötig sind, reichen:

```
[]{std::cout << "Hello World";};
```

Quiz Frage 6

Welche Ausgabe liefert der folgende Code-Snippet?

```
int s = 0;
std::for_each(a.begin(), a.end(), [&s](int x){ s += x; });
```

A) s bleibt unverändert
B) s wird um die Summe aller Elemente von a erhöht
C) s wird auf 0 gesetzt
D) Kompilierungsfehler

Quiz Frage 6

Welche Ausgabe liefert der folgende Code-Snippet?

```
int s = 0;
std::for_each(a.begin(), a.end(), [&s](int x){ s += x; });
```

A) s bleibt unverändert
B) s wird um die Summe aller Elemente von a erhöht
C) s wird auf 0 gesetzt
D) Kompilierungsfehler

Da `s` per Referenz (`&s`) eingefangen wird, kann das Lambda die Variable direkt verändern.
Mit `std::for_each` wird jedes Element x wird zu s addiert, sodass am Ende die Gesamtsumme entsteht.

Quiz Frage 7

Aufgabe: Definiere eine Funktion mit der Signatur
$f2 : A \times A \rightarrow bool$

```
template <typename A>
A eq(A a1, A a2) {
    return a1 == a2;
}
```

```
template <typename A>
bool eq(int a1, int a2) {
    return a1 == a2;
}
```

```
template <typename A>
bool eq(A a1, A a2) {
    return a1 + a2;
}
```

```
template <typename A>
bool eq(A a1, A a2) {
return a1 == a2;
}
```

Quiz Frage 7

Aufgabe: Definiere eine Funktion mit der Signatur
f2 : A × A → bool

```
template <typename A>
A eq(A a1, A a2) {
    return a1 == a2;
}
```

Returns `A`, not `bool`

```
template <typename A>
bool eq(int a1, int a2) {
    return a1 == a2;
}
```

Works only for `int`, not generic

```
template <typename A>
bool eq(A a1, A a2) {
    return a1 + a2;
}
```

Adds instead of comparing

```
template <typename A>
bool eq(A a1, A a2) {
return a1 == a2;
}
```

Correct!

03 – Programmierübung

Programmierübung: Prominente Higher Order Function

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <functional>
#include <cassert>

template <typename A, typename B>
std::vector<B> map(std::vector<A> as, std::function<B(A)> f) {
  //TODO: Deine Code hier
    return result;
}



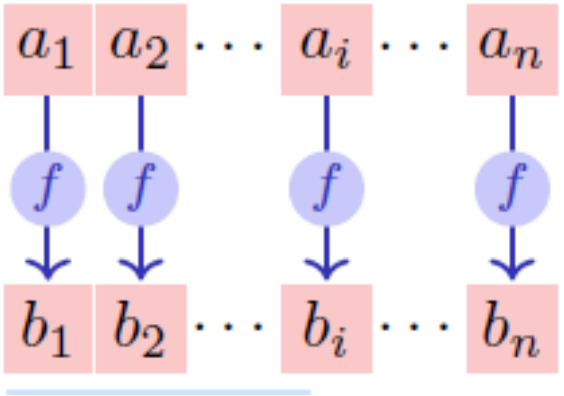
//-----------------TESTCASE, DONT TOUCH--------------------------------

template <typename T>
void printVector(const std::vector<T>& vec, const std::string& name) {
    std::cout << name << ": [ ";
    for (const auto& el : vec) {
        std::cout << el << " ";
    }
    std::cout << "]" << std::endl;
}
```

1. Geh auf CodeExpert → Code Examples → Sandbox Task
2. Implementiere Funktion map mit folgender Signatur:

$$\texttt{vec<A>} \times (A \to B) \to \texttt{vec<B>}$$

Visualisierug:



Code zum Abkopieren:

```
#include <iostream>
#include <vector>
#include <string>
#include <functional>
#include <cassert>

template <typename A, typename B>
std::vector<B> map(std::vector<A> as, std::function<B(A)> f) {
  //TODO: Deine Code hier
    return result;
}



//------------------TESTCASE, DONT TOUCH---------------------------------


template <typename T>
void printVector(const std::vector<T>& vec, const std::string& name) {
    std::cout << name << ": [ ";
    for (const auto& el : vec) {
        std::cout << el << " ";
    }
    std::cout << "]" << std::endl;
}

int main() {
    std::vector<int> input = {1, 2, 3, 4};

    auto result = map<int, std::string>(input, [](int x) {
        return std::to_string(x);
    });

    std::vector<std::string> expected = {"1", "2", "3", "4"};

    printVector(result, "Result");
    printVector(expected, "Expected");

    assert(result == expected);

    std::cout << "Test passed!" << std::endl;

    return 0;
}
```

Programmierübung: Prominente Higher Order Function

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <functional>
#include <cassert>

template <typename A, typename B>
std::vector<B> map(std::vector<A> as, std::function<B(A)> f) {
  //TODO: Deine Code hier
    return result;
}

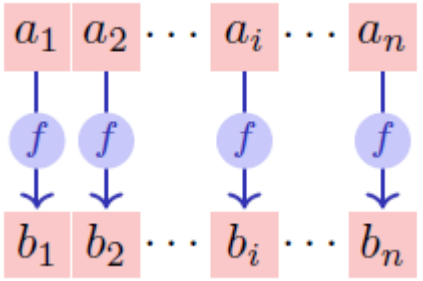


//-----------------TESTCASE, DONT TOUCH--------------------------------

template <typename T>
void printVector(const std::vector<T>& vec, const std::string& name) {
    std::cout << name << ": [ ";
    for (const auto& el : vec) {
        std::cout << el << " ";
    }
    std::cout << "]" << std::endl;
}
```

1. Geh auf CodeExpert → Code Examples → Sandbox Task
2. Implementiere eine Funktion mit Signatur

$$\texttt{vec<A>} \times (A \to B) \to \texttt{vec<B>}$$



Lösung_

```cpp
template <typename A, typename B>
std::vector<B> map(std::vector<A> as, std::function<B(A)> f) {
  std::vector<B> result;
  for (const auto& a : as)
    result.push_back(f(a));
  return result;
}
```

ETH zürich

Michal Oskedra
Teaching Assistent D-ITET
moskedra@ethz.ch