Datentypen

Referenzen	**Alias** für bestehende Variable

Referenzen können nur Variablen ihrens zugrundeliegenden Typs referenzieren. Sonst gibt es einen Fehler.

Ausserdem können Referenzen nur mit L-Werten initialisiert werden (also Werten mit einer Adresse im Speicher).

```
// Usage
int a = 3;
int& b = a; // reference to a
std::cout << b << "\n"; // Output: 3
a = 18;
std::cout << b << "\n"; // Output: 18
b = 25;
std::cout << a << "\n"; // Output: 25

// Issues
int& c = 3; // Error: 3 has no address in memory
bool d = false;
int& e = d; // Error: d is bool, e wants to reference an int
```

Funktionen

Deklaration	**Erweiterung des Gültigkeitsbereiches** der Funktion

Eine Funktion kann im Programm nur in Zeilen verwendet werden, welche nach der ersten Deklaration der Funktion kommen. Die Definition der Funktion ist immer gleichzeitig auch eine Deklaration.

Bei Deklarationen kann man `const` bei den Argumenten auch weglassen (sollte dies wenn möglich aber nicht machen).

(...)

(...)

```
void B (const int i); // separate declaration

void A (const int i) {
    if (i <= 0) return; // stop calls
    std::cout << "A";
    B(i/2); // use of B although its definition happens below
}

void B (const int i) {
    if (i <= 0) return; // stop calls
    std::cout << "B";
    A(i/2);
}
```

Standardbibliothek

std::pow	**Potenzieren**
Erfordert: `#include <cmath>`	
`double a = std::pow(2.5, 2); // Computes 2.5 ^ 2 == 6.25`	

std::sqrt	**Quadratwurzel**
Erfordert: `#include <cmath>` IEEE 754 garantiert, dass der (mathematisch) exakte Wert auf die nächste repräsentierbare Zahl gerundet wird.	
`double a = std::sqrt(14.0625); // Result: 3.75`	

std::abs	**Absolutbetrag**
Erfordert: `#include <cmath>`	
`double a = std::abs(-3.5); // Result: 3.5`	

Generell

namespace	**Katalogisierung** von Befehlen
Mit namespaces kann man Funktionen, Typen, etc. katalogisieren (z.B. bezüglich Projekten). Beispielsweise werden viele der "offiziellen" Funktionen dieser Vorlesung im `namespace ifmp` zusammengefasst. So können Sie diese Funktionen einfach von Ihren eigenen Funktionen unterscheiden.	

```
namespace ifmp { // namespace called ifmp

    // POST: "Hi" was written to the terminal
    void output_func () { // this function is in namespace ifmp
        std::cout << "Hi";
    }
}

int main () {
    ifmp::output_func(); // use output_func from namespace ifmp
    ifmp::integer a = 2147483647; // also in namespace ifmp
    return 0;
}
```