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.	

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

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

void B (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
```

`std::min`	**Minimum** zweier Argumente

Erfordert: `#include<algorithm>`

Sonst gibt es noch:
`std::max` **Maximum** zweier Argumente

Wichtig ist, dass beide Argumente vom selben Typ sind. Sonst geht es nicht.

```
double z;
std::cin >> z;
std::cout << std::min(z, 1.0); // min of z and 1

std::cout << std::min(z, 1); // Error: z is double, 1 is int
```

Generell

`namespace`	**Katalogisierung** von Befehlen

Mit `namespace`s 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.

Ausserdem kann man bei grösseren Projekten mit `namespace`s verschiedenste Namenskonflikte verhindern (z.B. bei gleich benannten Funktionen).

```
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
    return 0;
}
```

`assert`	**sofortiges Stoppen** des Programms bei Verletzung einer Bedingung (zu Testzwecken)

Erfordert: `#include<cassert>`

Wenn das fertige Programm veröffentlicht werden soll, kann man die `assert`-Befehle bequem deaktivieren.

```
int a;
int b;
std::cin >> a >> b; // read two int values from user
assert(b != 0); // prevent division by 0
std::cout << a / b << "\n";
```