## Generell

| { ... } | **Block** |
|---|---|
| Blöcke spielen eine grosse Rolle, wenn es darum geht, wo im Programm eine Variable gültig ist. So ist eine Variable ab ihrer Deklaration bis hin zum Ende des Blocks, in dem sie definiert wurde potentiell gültig. | |

```
int main () {
    unsigned int a;
    std::cin >> a;
    if (a < 4) {
        std::cout << a << " "; // a exists in nested blocks
        int b = 18;
        std::cout << b << " "; // b exists here too
    } else {
        std::cout << b << " "; // Error: b not declared yet
        int b = 11;
    }
    std::cout << a << " "; // a still exists here
    std::cout << b << "\n"; // Error: b does not exist anymore
    return 0;
}
```

## Datentypen

| void | Datentyp für **Funktion ohne Rückgabe**. |
|---|---|
| `void`-Funktionen haben keinen Rückgabewert, aber **sinnvollerweise einen Effekt** (z.B. Textausgabe im Beispiel unten). | |

```
void print_message (const int a, const double b) {
    std::cout << "Your assets:   " << a << "\n"
              << "Your interest: " << b << "\n";
}
```

## Funktionen

| Funktion | **Selbstständiger Codeabschnitt** |
|---|---|

Wichtige Befehle:

**Definition:** `int my_fun (bool arg1, float arg2) {...}`
**Deklaration:** `int my_fun (bool arg1, float arg2);`
**Rückgabe:** `return my_val;`
**Aufruf:** `my_fun(true, 3.75f)`

Der Rückgabewert wird immer zum Rückgabetyp konvertiert.

Wir verwenden Funktionsdeklarationen dazu, um den Gültigkeitsbereich der zugehörigen Funktionsdefinition zu vergrössern (Siehe Beispiel unten).

Bei der Funktionsdeklaration spielt es keine Rolle, ob die Argumente `const` sind oder nicht.

Die Funktionsdefinition ist auch eine Funktionsdeklaration.

Jede Funktion, die nicht den Rückgabetyp `void` hat, muss ein `return` haben.

```
int foo (unsigned int a);

void bar (unsigned int a) {
    std::cout << foo(--a) << "\n";
        // Note: foo is defined below. Thus without the
        //       declaration above, bar would not "know about"
        //       foo.
}

int foo (const unsigned int a) {
    bar(a);
    return a+1;
}

...
```

| `// PRE: ...`<br>`// POST: ...` | **Funktionsbeschreibung** |
|---|---|
| PRE-/POST-Conditions gehören vor **jede** Funktionsdefinition ausser der `main`-Funktion. (In diesen Programmier-Befehlszusammenfassungen werden sie aber manchmal aus Platzgründen weggelassen.) | |
| `// PRE: -`<br>`// POST: return value is a^4`<br>`int power_4 (const unsigned int a) { return a*a*a*a; }`<br><br>`// PRE: a != 0`<br>`// POST: return value is true if and only if a divides 8765`<br>`bool divides_8765 (const int a) { return 8765 % a == 0; }` | |

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