

## Datentypen

<code>float</code>	Datentyp für <b>Zahlen mit Nachkommastellen</b> (32 Bit)
Literal: ohne Exponent: <code>288.18f</code> , mit Exponent: <code>0.28818e3f</code>	
Der Modulo-Operator <code>%</code> existiert für <code>float</code> nicht.	
<pre>float a = 288.18f; float b = 0.28818e3f / a; // computations work as expected float c; std::cin &gt;&gt; c; // float user input</pre>	

<code>double</code>	größerer Datentyp für <b>Zahlen mit Nachkommastellen</b> (64 Bit)
Literal: ohne Exponent: <code>288.18</code> , mit Exponent: <code>0.28818e3</code>	
Unterschied zu <code>float</code> : <code>double</code> ist genauer (grössere Präzision und grösseres Exponenten-Spektrum), braucht aber mehr Platz im Speicher ( <code>float</code> : 32 Bit, <code>double</code> : 64 Bit). Der Modulo-Operator <code>%</code> existiert für <code>double</code> nicht.	
<pre>double a = 288.18; double b = 0.28818e3 / a; // computations work as expected double c; std::cin &gt;&gt; c; // double user input</pre>	

## Schleifen

<code>while (...) {...}</code>	while-Schleife
<pre>// Compute number of binary digits for input &gt; 0 unsigned int bin_digits = 0; unsigned int input; std::cin &gt;&gt; input; assert(input &gt; 0);  while (input &gt; 0) {     input /= 2;     ++bin_digits; }</pre>	

<code>do {...} while (...);</code>	do-Schleife
<p>Der Unterschied zur <code>while</code>-Schleife ist, dass der <b>Rumpf</b> der <code>do</code>-Schleife mindestens einmal ausgeführt wird. Sie hat ein <code>“;”</code> am Schluss.</p>	
<pre>int input; do {     std::cout &lt;&lt; "Enter negative number: ";     std::cin &gt;&gt; input; } while (input &gt;= 0); std::cout &lt;&lt; "The input was: " &lt;&lt; input &lt;&lt; "\n";</pre>	

## Programmier-Befehle - Woche 4

<code>break</code>	Schleife <b>abbrechen</b>
<pre>double input; int n; std::cin &gt;&gt; input &gt;&gt; n;  // Divide input by n numbers // Stop if 0 is entered. for (int i = 0; i &lt; n; ++i) {     double k;     std::cin &gt;&gt; k;     if (k == 0)         break; // go straight to Output     input /= k; }  // Output std::cout &lt;&lt; input &lt;&lt; " remains\n";</pre>	

<code>continue</code>	zur <b>nächsten Iteration</b> springen
<p>Bei der for-Schleife wird das Inkrement <b>noch ausgeführt</b>.</p>	
<pre>double input; int n; std::cin &gt;&gt; input &gt;&gt; n;  // Divide input by n numbers // Skip entered 0's. for (int i = 0; i &lt; n; ++i) {     double k;     std::cin &gt;&gt; k;     if (k == 0)         continue; // go straight to ++i     input /= k; }  // Output std::cout &lt;&lt; input &lt;&lt; " remains\n";</pre>	

## Andere Kontrollanweisungen

<code>switch</code>	Fallunterscheidung
<p>Wird ein case nicht mit einem <code>break</code> abgeschlossen, so werden die darunter liegenden cases auch noch ausgeführt, bis ein <code>break</code> erreicht wird.</p> <p>Die einzelnen Unterscheidungswerte müssen Konstanten sein.</p>	
<pre>std::cout &lt;&lt; "Behind which door (1,2,3) is the prize?"; int door_number; std::cin &gt;&gt; door_number; switch (door_number) {     case 1:     case 3:         std::cout &lt;&lt; "Wrong choice :-(\n";         break;     case 2:         std::cout &lt;&lt; "You won the prize!\n";         break;     default:         std::cout &lt;&lt; "Error: unknown door number.\n"; }  // User inputs 0 --&gt; Error: unknown door number. // User inputs 1 --&gt; Wrong choice :-(\n // User inputs 2 --&gt; You won the prize!\n // User inputs 3 --&gt; Wrong choice :-(\n</pre>	

## Generell

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

( ... )

(...)

```
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;
}
```