`if` und Schleifen: Python – C/C++

Die Hauptunterschiede sind:

1) In C/C++ steht kein Doppelpunkt nach dem `if, while, for`.
2) Bedingungen sind in C/C++ **immer in runde Klammern** zu setzen:

   ```
   if (x > 0) …
   while (i != j) …
   ```

3) Einrückungen sind in C++ gut für die Übersichtlichkeit, sind dem Compiler aber egal.
4) Die ganze `if/while/for` Konstruktion gilt als neues Scope. Definiert man eine Variable **innerhalb** dieses Blockes, **so ist sie nach dem Block nicht mehr sichtbar**.
5) Ein abhängiger Block ist **immer** mit geschwungenen Klammern **{ }** zu klammern, **außer er umfasst nur eine Anweisung**. Viele Programmierer machen auch dann die Klammern (ich meistens nicht). Vergisst man die Klammern, ist das Programm trotzdem syntaktisch korrekt (es gehört nur die erste Zeile zum Block) und funktioniert nicht wie erwartet. Neue Compiler warnen in diesem Fall.

   ```
   if (i > 10)
        i = 10;
        cout << "i wurde auf 10 reduziert\n";   // FEHLER!

   if (i > 10) {
        i = 10;
        cout << "i wurde auf 10 reduziert\n";   // korrekt
   }
   ```

6) **Ein Strichpunkt direkt nach dem if, while, for ist fast immer falsch.** Dieser zählt nämlich als eine (leere) Anweisung und ist damit für den Compiler der komplette abhängige Anweisungsblock:

   ```
   if (i > 10);          // falsch, fälscher am fälschesten!
   {    i = 10;
        cout << "i wurde auf 10 reduziert\n";
   }
   ```

   Der ganze Block in `{ }` hängt **nicht** vom `if` ab (keine Warnung!)

   Wenn man wirklich einmal einen leeren Block haben will, ist die Schreibweise `{}` am deutlichsten:

   ```
   while (*dest++ == *src++)      //  überspringe gleiche Werte
   { }                            // der ganze Block ist wirklich leer
   ```

Sowohl in Python als auch in C++

Python if – C++ if:

Geht in C/C++ sehr ähnlich. Der einzige Unterschied ist eigentlich, dass in C++ das `elif` durch `else if` ersetzt wird (es gibt kein `elif` in C/C++). Man schreibt das `else if` nicht eingerückt genau dorthin, wo man in Python das `elif` schreiben würde.

Python while – C++ while:

Hier gibt es überhaupt keine Unterschiede, selbst die Verwendung von `break` und `continue` ist identisch.

Python for in range – C++ for

Diese Python Konstruktion kann man recht einfach in C++ übersetzen:

Python: `for i in range(start, stop, step):`
C++:

```
     step > 0:      for (auto i{start}; i < end; i += step)
     step < 0:      for (auto i{start}; i > end; i += step)
```

Python for – C++ range-based for: (seit C++11)

Diese Konstruktion gibt es in C++ erst seit C++11, ist aber seither sehr populär geworden. Bei der Definition der Laufvariable ist bei C++ der Typ zu schreiben (am besten `auto`)

Python: `for i in …` C++: **`for (`**`auto i :` …**`)`**

Es gibt allerdings in C++ noch die Möglichkeit, Referenzen als Laufvariablen zu verwenden. Deswegen sieht man in C++ auch folgende Konstruktionen:

```
for (auto& i:  …)          // kann … ändern, wenn  i verändert wird
for (auto&& i: …)          // kann … ändern, wenn  i verändert wird
for (const auto& i: …)     // darf … nicht ändern, da man i nicht ändern darf
```

Nur in C++

`do while` Loop:

Dieser verhält sich wie ein normaler `while`-Loop, nur wird die Bedingung erst am Ende getestet. Diese Schleife wird daher immer mindesten einmal durchlaufen. Sie wird eher selten verwendet:

```
do {
      ++i;
      …
}  while (…);
```

`for`-Loop:

Dieser Schleifentyp existiert es schon seit den Anfängen von C und ist die an der häufigsten verwendeten Schleife in C/C++. Man kann sie nicht mit dem range-based `for` verwechseln, weil der Schleifenkopf **2 Strichpunkte** enthalten muss und **keinen Doppelpunkt** enthält:

```
for (init;  condition;  update)
```

`init` Ein Statement, das vor der ganzen Schleife ausgeführt wird (z.B. `int i = 0`)
`condition` Die Schleifenbedingung ohne eigene Klammer
`update` ein allfälliger Update des Index (z.B. `++i, i += 4`)

Laufe von `a` bis zum nächsten Vielfachen von `1000`:

```
for (int i = a; i % 1000 != 0; ++i)
                …
```

`if` mit *`init`* (C++14) und range based `for` mit *`init`* (C++17)

Dadurch ist auch in diesen Konstruktionen eine *`init`*-Anweisung möglich, die genauso wie im `for`-Loop funktioniert:

```
if (auto i{random_int()} ; i == 1) {
      …
} else if (i == 2) {
      …
} else …
```