Hauptunterschiede zwischen Python und C++

Worin liegen die wesentlichen Unterschiede zwischen Python und C++, wo muss man auch als etwas erfahrener Python-Programmierer aufpassen:

1) C++ ist eine kompilierte Sprache, d.h. einfach ein paar Statements ausführen funktioniert in C/C++ gar nicht. Man muss ein korrektes und vollständiges Programm erstellen, dieses kompilieren und erst dann kann man es ausführen.
2) Der Compiler darf ihr Programm optimieren, d.h. er darf ihren Code so umformen, dass das Programm schneller läuft aber dasselbe Ergebnis liefert. Da die nötige Programmanalyse mehr Zeit erfordert, macht der Compiler das nur, wenn Sie ihn dazu auffordern (Option `-O` oder `-O3`). Typische Aktionen sind z.B. das Wegoptimieren von Funktionsaufrufen oder das Loop-Unrolling:

```
int square(int x) { return x*x; }
int x = square(i);        ->  int x = i*i;

for (i = 0; i < 3; ++i)          // dieser Loop wird unrolled
     x[i] = 0;            -> x[0] = 0; x[1] = 0; x[2] = 0;
```

3) C/C++ erlaubt ausführbaren Code nur innerhalb von Funktionen. Das gesamte Programm besteht daher aus einer nichtleeren Menge von Funktionen. Genau eine davon muss **`main()`** heißen und ist der Startpunkt des ganzen Programms. Das Programm endet, wenn die Funktion `main()` abgearbeitet oder anderweitig beendet wird.
4) C/C++ ist eine **statisch stark** typisierte Sprache, d.h. für jedes Objekt **muss bei dessen Definition** der Datentyp angegeben werden. Dieser Typ kann **nicht mehr verändert** werden. Auch Funktionsargumente und -ergebnisse müssen eine Typangabe haben.

   Python:

```
n = 4            # kein Typ angegeben, dadurch ist n jetzt ein int
n = "haha"       # kein Typ angegeben, dadurch ist n jetzt ein str
```

   C/C++ vor 2011:

```
int n = 4;       // der Typ muss angegeben werden, n ist ein int
n = "haha";      // Fehler: ein int kann keinen String speichern
```

   C++ ab 2011 kann **`auto`** als Typangabe verwenden, allerdings kann auch hier der Typ nach der Definition nicht mehr verändert werden:

```
auto n = 4;      // n ist ein int
n = "haha";      // Fehler: ein int kann keinen String speichern
```

Eine mehrfache Definition eines Namens im selben Scope ist ebenfalls nicht möglich:

```
auto n = 4;          // n ist ein int
auto n = "haha";     // Fehler: verbotene Redeklaration
```

Namen dürfen in verschiedenen Scopes neu definiert werden, bezeichnen dann aber **unterschiedliche** Objekte:

```
auto n = 4;                // dieses n ist ein int
{                          // ein neuer Scope wird erzeugt
    n = "haha";            // Fehler: dieses n ist immer noch ein int
    auto n = "haha";       // ok: hier entsteht ein neues n als String
    n = 4;                 // Fehler: dieses neue n ist ein String

}                          // der neue Scope endet hier, der String n wird zerstört
n = 7;                     // ok, dieses n ist vom Typ int

int sum(int a, int b)      // Argument-Typen und Ergebnistyp sind nötig
```

5) Einrückungen und Zeilenvorschübe haben in C/C++ keine semantische Bedeutung. Wer will, kann das gesamte C++ Programm in einer einzigen Zeile schreiben. Das Ende von Anweisungen wird durch einen Strichpunkt gekennzeichnet, Anweisungsblöcke (Schleifen etc.) durch Klammerung mit `{}`. **Korrekte Einrückungen verbessern aber die Lesbarkeit des Programmes enorm.**

6) Es gibt in der C++-Bibliothek auch Datentypen, die analog zu einer Liste oder einem Tupel beliebig viele Objekte speichern können, allerdings müssen diese Objekte **alle vom selben Typ** sein!

7) Die eingebauten Datentypen von C/C++ orientieren sich an den Datentypen der CPUs. Diese kennen eigentlich nur ganze Zahlen (mit oder ohne Vorzeichen und mit verschiedenen Bitanzahlen) sowie die Gleitkommazahlen der IEEE. Ein zusätzlicher Datentyp ist die Speicheradresse (= Pointer), der kein eigentlicher Datentyp ist, sondern nur eine Stelle im Computerspeicher (=RAM) adressiert.

8) C und C++ haben einen sehr kleinen Sprachkern. Alles darüber Hinausgehende ist in Bibliotheken ausgelagert, über die der Compiler gar nichts weiß. Um solche Bibliotheksfunktionen richtig zu verwenden, müssen diese dem Compiler „vorgestellt“ werden. Das geschieht derzeit noch ausschließlich durch das Inkludieren von einigen (oder sehr vielen) Headerdateien, in denen diese Bibliotheksinhalte deklariert sind. Erst in C++20 wurde ein Modulsystem vorgestellt, welches so ähnlich wie die Python-Module funktionieren wird.

9) Ein Programm in C/C++ kann auf beliebig viele Dateien aufgeteilt werden (sog. Translation Units), die auch einzeln kompiliert werden können. Jede Translation Unit muss alle verwendeten Bibliotheken inkludieren und auch die Bestandteile von den anderen Translation Units kennen, die verwendet werden. Dies erreicht man durch zusätzliche selbstgeschriebene Headerdateien, die die Inhalte einer oder mehrerer Translation Units deklarieren:

```
utility.cpp     # eine Translation Unit mit einigen Funktionen, Datentypen ...
utility.h       # Headerdatei dazu, die die Deklarationen enthält
```

Hello World

Das folgende ist das klassische Hello-World-Programm in C++:

```
/*  mit Slash Stern starten
     wir (wie in Python mittels """) einen
          mehrzeiligen Kommentar, der
               hier beim Stern Slash endet  */

// ein einzeiliger Kommentar (wie # in Python), keine Division

#include <iostream> // Headerdatei mit (u.a.) der Deklaration von std::cout

int main()          // das Hauptprogramm gibt IMMER einen int zurück
{                   // mit dieser Klammer beginnt main()

     std::cout << "Hello World\n";

}                   // mit dieser Klammer endet main()
```

`main()` ist ein Spezialfall: Endet diese bei }, wird automatisch 0 zurückgegeben. Keine andere Funktion hat diese spezielle Eigenschaft, d.h. alle anderen Funktionen mit Ergebnis **müssen imme**r einen expliziten Rückgabewert vereinbaren! Ich empfehle daher, auch in `main()` eine explizite `return` Anweisung zu machen. Der Rückgabewert ist ein Fehlercode, der dem Betriebssystem verrät, ob das Programm erfolgreich ausgeführt wurde:

```
return 0; // das Programm hat funktioniert
return 1; // das Programm hat nicht funktioniert (Werte ungleich 1 sind Fehler)
```

Wenn Sie die Headerdatei **cstdlib** inkludieren, können Sie statt dieser Rückgabecodes auch ausdrucksstarke Namen verwenden:

```
#include <cstdlib>
…
int main()
{ …
      return EXIT_SUCCESS;       # gib 0 zurück
      return EXIT_FAILURE;       # gib 1 zurück
```