Blender - Lektion S2
Einstieg in Python

Blender V 2.78 – Skript V 1.0

Auch diese Lektion stammt aus der Feder von Johannes Klatt. Ich wünsche erneuten Lerngenuss bei der Lektüre.

Was tun, wenn das, was man von Blender an Features geliefert bekommt, einmal nicht ausreicht?

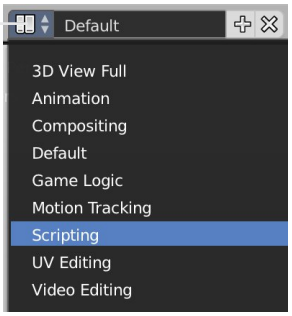
In dieser Lektion geht es um Python, die Programmiersprache, in der die Oberfläche von Blender geschrieben ist. Das macht Python zu einer Art Allzweckwaffe, da mit ihr ALLES, was das Programm zu bieten hat, ansteuern kann.

Autor und Konzept: Johannes Klatt, Franz-Ludwig-Gymnasium Bamberg, Computergrafikgruppe (CoGra-FLG)
Layout und Bilder: Uwe Gleiß
Kontakt über: cogra-flg@web.de

Umgang mit der Konsole

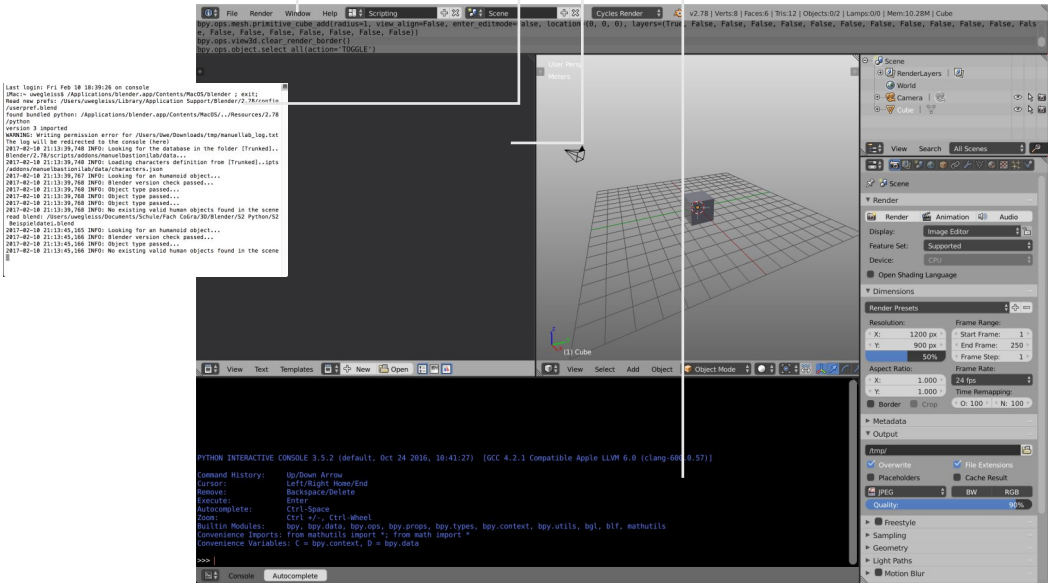
Vorbereitungen

Schalten Sie die Ansicht auf „Scripting“.



Neben den bekannten Frames gibt es zwei neue: eine Python-Konsole unten und den Texteditor links oben. Außerdem hat sich der Info-Frame verändert: Über den bekannten Knöpfen befindet eine neuer Bereich, der die zuletzt ausgeführten Python-Befehle anzeigt.[1]

Zusätzlich werden Sie noch die Systemkonsole benötigen. Diese können sie im Info-Frame unter *Window - Toggle System Console* an– und abschalten.[1] Versuchen Sie aber nie, die Systemkonsole wie ein normales Fenster über das rote X zu schließen, denn dann schließt sich Blender gleich mit.



[1] Unter OSX (Apple) müssen Sie Blender anders starten als gewohnt, um die Systemkonsole zu erhalten: Rechtsklick auf das Programmsymbol – Paketinhalt anzeigen. Dann unter Contents – MacOS die dortige Datei blender starten.

Erste Befehle

Schnappen Sie sich die Python-Konsole (nicht die Systemkonsole) und geben Sie eine Zahl ein, bevor Sie mit Enter bestätigen.

```
>>> 3
3
```

Ihre Eingabe

Programmausgabe

Python schreibt also Ihre Zahl in die Konsole. So macht das noch relativ wenig Sinn, aber Python beherrscht auch Mathe:

```
>>> 1 + 2
3
>>> 7 * 8
42
>>> 142.5 / 3
47.5
```

Hierbei steht der Punkt für ein Komma, der Stern für malnehmen und der Schrägstrich für teilen. Wie Sie sehen, lässt sich Python problemlos als Taschenrechner verwenden.

Allerdings reicht es schon im Matheunterricht schnell nicht mehr aus, voneinander unabhängige, einfache Rechenaufgaben zu lösen; man muss sich das Ergebnis auch merken und weiterverarbeiten können. In Python gibt es dafür Variablen. Eine Variable erstellen Sie, indem sie den gewünschten Namen in die Konsole tippen, gefolgt von einem = und dem Wert, den die Variable haben soll, z.B.:

```
>>> Zahl = 2
```

Python wird ihnen nach so einem Befehl keine Ausgabe produzieren. Dafür wird ab jetzt immer, wenn sie den Namen der Variable eingeben, stattdessen ihr Wert verwendet:

```
>>> Zahl
2
>>> Zahl + 3
5
```

[1] Die farbliche Darstellung der Codebeispiele orientiert sich an dem Farbschema Flatty Light. Weiß wurde aber durch Schwarz ersetzt.

Eine Variable können Sie ändern, indem Sie ihren Inhalt mit einem neuen Wert überschreiben:

```
>>> X = 3
>>> X
3
>>> X = 7
>>> X
7
```

Dabei kann in der Anweisung für das Überschreiben auch ein mathematischer Ausdruck oder die Variable selbst vorkommen:

```
>>> Wert = 8
>>> Wert
8
>>> Wert = Wert - 6
>>> Wert
2
```

Skripte

Ein Skript ist eine Abfolge von Befehlen, die hintereinander ablaufen, sobald es ausgeführt wird. In Blender werden Skripte im Texteditor geschrieben. Die Benutzerschnittstelle beim Umgang mit Skripten bildet die Systemkonsole. Um sie anzusteuern gibt es zwei Befehle:

print()

Erstellen Sie durch Drücken auf *New* ein neues Skript und schreiben Sie einige print-Anweisungen untereinander. Die Variable, deren Wert oder die Zahl, die in die Systemkonsole geschrieben werden soll, kommt in die Klammer. Schreiben Sie z.B. das folgende Skript:

```
print(28)
Var = 9
print(Var)
```

Nach Start durch *Run Script* gestartet haben, erscheint in der Systemkonsole die Ausgabe:

```
28
9
```

Oft will man nicht nur Zahlen, sondern auch Text als Ausgabe erhalten. Zwar kann man einer Variable einen Text zuweisen,[1] aber eine wirkliche Lösung ist das nicht. Deshalb kann in die Klammern der print–Anweisung in Anführungszeichen jeder beliebige Text geschrieben werden. Anführungszeichen sorgen dafür, dass Python das Geschriebene direkt in der Konsole ausgibt, anstatt es als Variablennamen zu interpretieren. Außerdem können in die Klammer mehrere Textstücke, Variablen oder Zahlen durch Kommas getrennt kombiniert werden. Diese werden dann auch hintereinander geschrieben. Die Anweisung

```
print("Text" , 123 , "Cogra")
```

führt beispielsweise zu folgendem Ergebnis:

```
Text 123 Cogra
```

[1] Der Text, der der Variable als Wert zugewiesen wird, muss in Anführungszeichen gesetzt werden. Es ist möglich, zwei Textvariablen zu addieren - ihr Inhalt wird aneinandergehängt.

int(input())

Wenn man statt der Python–Konsole ein Skript verwendet, kann man die darin verwendeten Zahlenwerte nicht während der Ausführung bearbeiten, was in vielen Fällen aber nötig ist.

Dieses Problem löst die input–Anweisung. Mit ihr lässt sich mitten im Ablauf des Skripts eine Eingabe des Benutzers einlesen. int(input()) lässt sich überall in ein Skript einsetzen, wo normalerweise eine Zahl hingehören würde. Wird dann die entsprechende Codezeile erreicht, so wird der Benutzer aufgefordert, eine Zahl in die Systemkonsole einzugeben. Sobald dies geschehen ist läuft das Skript weiter und verwendet an der Stelle, an der der Befehl steht, die eingegebene Zahl.[1]

Die Klammern funktionieren wie schon bei print: was in sie hineingeschrieben wird, landet in der Systemkonsole, allerdings diesmal als Eingabeaufforderung.

```
Eingabe = int(input("Geben Sie eine Zahl ein:"))
print("Ihre Eingabe war:" , Eingabe)
```

Dieses Skript produziert in der Systemkonsole erst einmal folgende Eingabeaufforderung:

```
Geben Sie eine Zahl ein:
```

Tippt der Benutzer eine Zahl, hier 1024, ein und bestätigt mit Enter, geht es weiter:

```
Geben Sie eine Zahl ein: 1024
Ihre Eingabe war: 1024
```

[1] Eigentlich liest input() die Eingabe aus und int() konvertiert sie in eine ganze Zahl. Benötigen Sie Kommazahlen, dann können Sie int() durch float() ersetzen.

Bedingungen

Struktur

Es kommt oft vor, dass eine bestimmte Befehlskette mehrmals hintereinander ausgeführt werden soll, und dass das "wie oft" dabei ansteuerbar sein soll. Diesen Zweck erfüllen Bedingungen und Schleifen. Generell ist ihr Aufbau immer gleich:

```
Anweisung außerhalb der Bedingung
Kopfzeile
      Anweisung
           .
           .
           .
      Anweisung
Anweisung außerhalb der Bedingung
```

Die Kopfzeile legt die Art der Bedingung fest und enthält auch die Bedingung selbst. Den Abschluss bildet ein Doppelpunkt. Anschließend folgt, durch einen Tabulator eingerückt, der Schleifen-, bzw. Anweisungskörper. In dieser Lektion werden nur die grundlegenden Funktionen der beiden wichtigsten Typen, der *while*–Schleife und der *if*–Anweisung, behandelt.

Beide verwenden als Bedingung eine (Un)gleichung. Im Fall der while-Schleife wird, solange die Bedingung stimmt, der Inhalt der Schleife immer wieder ausgeführt. Stimmt sie nicht mehr, so geht es mit der nächsten Anweisung unterhalb der Schleife weiter.

Die if–Anweisung funktioniert ganz ähnlich: Wenn die Bedingung erfüllt ist wird der Anweisungskörper ausgeführt, ansonsten übersprungen.

Die Form der Kopfzeile ist also:

```
while/if ABC == XYZ:
```

wobei ABC und XYZ jeweils Zahlen, Variablen oder mathematische Ausdrücke sein können. Die beiden Gleichheitszeichen legen fest, dass der Inhalt der Schleife ausgeführt wird, wenn ABC und XYZ gleich sind.[1] Will man stattdessen eine Ungleichung formulieren, kann man <, >, <= oder >= verwenden. Außerdem gibt es != für „ist nicht gleich".

Es können auch zwei Bedingungen festgelegt werden. Diese werden dann hintereinandergeschrieben und durch einen logischen Operator voneinander getrennt, z.B.:

```
if A = 1 and B >= 4:
```

Der Anweisungskörper wird in diesem Fall also abgearbeitet, wenn A genau 1 ist und zugleich B mindestens 4.

Die if–Anweisung kann zusätzlich durch einen *else–Block* ergänzt werden:

```
Anweisung
if Bedingung:
      Anweisung
           .
           .
      Anweisung
else:
      Anweisung
           .
           .
      Anweisung
Anweisung
```

Alles, was nach dem Schlüsselwort else eingerückt steht, wird nur dann ausgeführt, wenn der Inhalt der zugehörigen if–Anweisung nicht ausgeführt wurde. Umgekehrt wird der Else-Block übersprungen, wenn der Inhalt der if–Anweisung ausgeführt wurde.

[1] Die Verdoppelung soll Verwechslungen mit dem Gleichheitszeichen, das den Wert einer Variable festlegt, vorbeugen.

Beispiel

Ziel ist es, ein Skript zu schreiben, das die Kreiszahl π mithilfe des Walis'schen Produkts annähert. Diese Formel besagt:

$$\frac{\pi}{2} = \frac{2}{1} \cdot \frac{2}{3} \cdot \frac{4}{3} \cdot \frac{4}{5} \cdot \frac{6}{5} \cdot \frac{6}{7} \cdot \frac{8}{7} \cdot \frac{8}{9} \cdot \ldots$$

Das ist eine Multiplikation von unendlich vielen Brüchen. Für das Skript muss als Erstes vom Benutzer abgefragt werden, wie viele Durchläufe des Algorithmus erwünscht sind:

```
Gewünscht = int(input("Wie viele Durchläufe?"))
```

Als nächstes erstellen wir eine Variable, die später die bereits durchgeführten Durchläufe zählt, sowie je eine weitere Variable für Nenner und Zähler des Bruchs, und eine letzte für das Zwischenergebnis:

```
Durchgeführt = 0
Nenner = 1
Zähler = 2
Ergebnis = 2
```

Das Skript soll nur dann ausgeführt werden, wenn die Eingabe einen sinnvollen Wert hat, d.h. größer als 0 ist:

```
if Gewünscht <= 0:
      print("Ungültige Eingabe")
      print("Skript wird beendet")
```

Ist die Eingabe ungültig, so werden die Befehle im if–Block ausgeführt, d.h. der Benutzer wird über seine ungültige Eingabe informiert. Ist die Eingabe in Ordnung, so wird stattdessen der nun folgende else–Block ausgeführt, in dem eine Schleife abläuft:

```
else:
      while Durchgeführt < Gewünscht:
            pass
            Durchgeführt = Durchgeführt + 1
```

Die Anweisung *pass* macht ganz einfach gar nichts; sie ist ein Platzhalter, der nur für die Entwicklungsphase gedacht ist und in fertigen Skripten nicht vorkommen sollte.

Die while–Schleife führt nun diese Anweisung so oft aus, bis *Durchgeführt* genauso groß ist wie *Gewünscht*. Da *Durchgeführt* nach jedem Durchgang um 1 vergrößert wird, wird der Schleifeninhalt so oft abgearbeitet, wie mit *Gewünscht* eingestellt wurde.

Zeit, die pass-Anweisung durch die Berechnung zu ersetzen:[1]

```
Nenner = Nenner + 2
Ergebnis = Ergebnis * Zähler
Ergebnis = Ergebnis / Nenner
Zähler = Zähler + 2
Ergebnis = Ergebnis * Zähler
Ergebnis = Ergebnis / Nenner
```

Zuerst wird der Nenner um 2 erhöht. Dann wird der neu entstandene Bruch in 2 Schritten mit dem bereits vorhandenen Zwischenergebnis malgenommen. Als nächstes wird der Zähler um 2 erhöht und der neue Bruch wieder mit dem Zwischenergebnis malgenommen. Das wird dank der Schleife so oft wiederholt, wie der Benutzer eingegeben hat; ein Durchlauf der while–Schleife bezieht sich also auf jeweils 2 Brüche in der Gleichung.

Jetzt fehlt nur noch der Abschluss. Die nächsten Codezeilen (immer noch im else–Block) lauten:

```
pi = Ergebnis * 2
print("Pi =" , pi)
```

[1] Achten Sie darauf, dass dieser Block ebenso wie zuvor die pass–Anweisung eingerückt ist. Sonst erkennt Python nicht, dass er Teil der Schleife ist.

Das vollständige Skript sieht so aus:

```
Gewünscht = int(input("Wie viele Durchläufe?"))
Durchgeführt = 0
Nenner = 1
Zähler = 2
Ergebnis = 2

if Gewünscht <= 0:
      print("Ungültige Eingabe")
      print("Skript wird beendet")
else:
      while Durchgeführt < Gewünscht:
            Nenner = Nenner + 2
            Ergebnis = Ergebnis * Zähler
            Ergebnis = Ergebnis / Nenner
            Zähler = Zähler + 2
            Ergebnis = Ergebnis * Zähler
            Ergebnis = Ergebnis / Nenner

            Durchgeführt = Durchgeführt + 1

      pi = Ergebnis * 2
      print("Pi =" , pi)
```

Wird das Skript ausgeführt, dürfte die Ausgabe in der Systemkonsole in etwa so aussehen: (Die Anzahl der Durchläufe mag übertrieben erscheinen, aber ihr Rechner steckt das problemlos weg)

```
Wie viele Durchläufe? 10000
Pi = 3.14167118653445
```

Kommentare

Wie man sich vorstellen kann, werden Skripte ohne Hilfestellung schnell unübersichtlich. Um in diesem Punkt Abhilfe zu schaffen gibt es Kommentare. Diese beeinflussen den Ablauf des Skriptes nicht und dienen nur zur Dokumentation. Ihr Syntax ist ziemlich selbsterklärend:

```
print(A)
#Das hier ist Dank der Raute eine Kommentarzeile

print("Beispiel") #Das geht auch nachgestellt

""" Das ist ein Blockkommentar,
er kann sich auch über
mehrere Zeilen erstrecken."""
```

Außerdem kann man mit Kommentaren zu Testzwecken gerade nicht benötigte Programmteile deaktivieren, indem man zu Beginn jeder Zeile eine Raute setzt, oder ganze Absätze in einen Blockkommentar verwandelt.

Module

Mathe mit Python ist schön und gut, aber wir sind ja Computergrafiker und wollen mit Python Blender ansteuern können! – Genau so wie damit womöglich ein paar Bankangestellte ihre Umbuchungen regeln, ein Spieleentwickler die Netzwerkkommunikation seines Onlinespiels steuern oder eine Firma ihre Produktionsanlage kontrollieren will. Die für all das nötigen Befehle standardmäßig in Python zu integrieren wäre der reinste Wahnsinn; ständig würde man aus Versehen Schlüsselwörter aus einem anderen Bereich verwenden. Deshalb hält Python erst einmal nur grundlegende Funktionen bereit, der Rest muss in Form von Modulen hinzugefügt werden.

Blender eine Modulbibliothek schon an Bord; diese umfasst von Raytracing bis hin zu Netzwerkkommunikation schon so ziemlich alle Module, die man braucht. Sollte trotzdem mal etwas nicht zu finden sein, kann man sich außerdem zusätzliche Module aus dem Internet herunterladen.[1]

Importiert wird ein Modul mit dem Befehl import, gefolgt vom Namen des Moduls. So importiert der Befehl

```
import random
```

das Modul Random, das unter Anderem eine Reihe von Zufallsfunktionen bereitstellt.

Vor Befehlen, die zu einem Modul gehören, stehen immer der Name des Moduls und ein Punkt. Wollen wir jetzt zum Beispiel eine Zufallszahl generieren, müssen wir nicht nur den Befehl

```
randint()
```

(mit den benötigten Parametern in der Klammer) eingeben, sondern es muss heißen:

```
random.randint(1, 10)
```

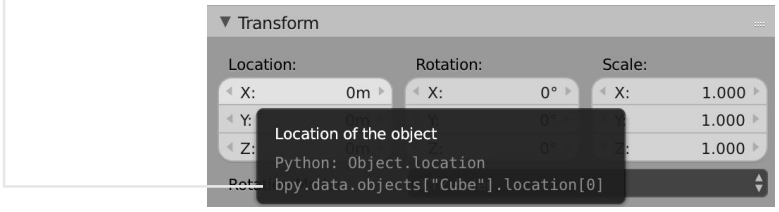
Dieser Befehl spuckt eine Zufallszahl zwischen 1 und 10 aus (natürlich geht das auch mit anderen Zahlen).

[1] Sämtliche Blender–Addons sind verkappte Python–Module.

bpy.data

Dieses Modul ermöglicht den am Anfang erwähnten Zugriff auf alle Parameter in Blender und ist somit wirkich die ultimative Waffe für dieses Programm.

Wenn Sie die Maus über ein beliebiges Zahlenfeld in Blender halten, achten Sie einmal auf die unterste Zeile des Tooltips.



Was auf den ersten Blick fürchterlich kompliziert aussieht, ist in Wirklichkeit der Name der Python-Variable, die zu dem Zahlenfeld gehört, einschließlich des Namens ihres Moduls, bpy.data. Diese lässt sich wie jede andere Variable auch über ein Skript oder die Konsole ansteuern. Man kann mit ihr auch wie mit normalen Variablen rechnen.

Probieren Sie das einfach mal mit diesem Befehl in der Konsole aus:

```
bpy.data.objects["Cube"].location[0] = 2
```

math

fügt etliche mathematische Werkzeuge und Konstanten hinzu. Wenn Sie genauer wissen wollen welche, dann importieren Sie das Modul in der Konsole und geben dann nur ein

```
math.
```

(mitsamt dem Punkt) und drücken Sie dann *Ctrl+Space*. Dieser Befehl ist die Autovervollständigung. Da math reihenweise Befehle enthällt, zeigt Python alle als lange Liste an. Tippen Sie zusätzlich noch cos und erneut *Ctrl+Space* ein, dann bietet Python nur noch die Möglichkeiten *(* und *h(*, die die Zeile zur Kosinusfunktion oder zur hyperbolischen Kosinusfunktion ergänzen.

bpy.ops

führt Blenderbefehle aus, ist aber deutlich komplizierter als bpy.data. Viele Befehle dieses Moduls funktionieren ohne weitere Voreinstellungen nicht, aber Sie können ja trotzdem mal mit *Ctrl+Space* stöbern. Genauer wird diese Modul in diesem Skript nicht behandelt.

random

kann Zufallszahlen generieren und Wahrscheinlichkeiten berechnen.

bge

Wenn Sie die Blender Game–Engine mit Python steuern wollen, dann kommen Sie an diesem Modul nicht vorbei. Aber Vorsicht: Seine Befehle lassen sich nicht wie sonst in Skripten direkt nutzen. Vielmehr müssen diese Skripte von einem Controller in einer Logic–Schaltung aufgerufen werden.

antigravity

Probieren Sie es selbst!

Anti-Panik-Anweisung

Im Zusammenhang mit Python ist es der Normalfall, dass die ersten paar (oder auch mehr) Versionen eines Skripts nicht funktionieren. Die meisten Fehler entpuppen sich relativ schnell als einfach zu behebende Banalitäten. Die häufigsten Varianten sind:

- Falsche Groß- und Kleinschreibung
- Vergessene Anführungszeichen
- Vergessene Doppelpunkte
- Vergessene Verdoppelung von Gleichheitszeichen

Darüber hinaus liefert die Systemkonsole seit Neuestem bei vielen Fehlermeldungen nicht nur eine Kopie der fehlerhaften Codezeile, sondern zusätzlich auch die genaue Stelle (mittels eines kleinen Pfeils unter der Kopie). Auf diese Weise lassen sich etwa 90% aller Fehler relativ schnell ausmerzen oder zumindest lokalisieren. Also, wenn eines ihrer Skripte mal nicht will: Prüfen Sie unbedingt diese Punkte ab, anstatt es gleich in die Tonne zu hauen!

Und sollten Sie einmal verzweifelt nach einem Befehl suchen: In der Skripting Ansicht von Blender wird im Infobereich ganz oben jeder Befehl angezeigt, den Sie in Blender gerade ausgeführt haben. Wenn Sie beispielsweise einen Würfel einfügen führt das dort zu der Bandwurmzeile:

```
bpy.ops.mesh.primitive_cube_add(radius=1, view_align=False,
enter_editmode=False, location=(0, 0, 0), layers=(True, False,
False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False))
```

Sie können diese markieren, kopieren und direkt in Skript oder Konsole verwenden und mit ihr experimentieren.

Übungsaufgaben

1. Verwenden sie die Konsole, um eine von diesen nervigen, endlos langen Kopfrechenaufgaben aus der 6. Klasse zu lösen!

2. Tun Sie irgendetwas in Blender und wiederholen Sie es dann mittels des passenden Befehls in der Konsole. Spielen Sie dann mit diesem Befehl, indem Sie seine Parameter verändern.[1]

3. Schreiben sie ein Skript, das den Standardwürfel eine vom Benutzer einstellbare Strecke entlang der Diagonalen zwischen X- und Y-Achse verschiebt (im Modul math gibt es eine Anweisung namens sqrt(), die die Wurzel der Zahl oder Variable, die in der Klammer steht, berechnet und sich zusammen mit dem Satz des Pythagoras als nützlich erweisen könnte).

4. Schreiben Sie ein Skript, das die Fakultät einer vom Benutzer einstellbaren Zahl berechnet und das Ergebnis in die Systemkonsole schreibt! (Für die, die es nicht wissen: die Fakultät einer Zahl ist diese Zahl mit allen Zahlen, die zwischen ihr und 0 liegen, multipliziert. Die Fakultät von 4 ist beispielsweise $4 \cdot 3 \cdot 2 \cdot 1 = 24$).

5. Sorgen Sie mittels Kommentaren für eine ordentliche Dokumentation ihrer Skripte!

6. Schreiben Sie ein kreatives Skript!

[1] Machen Sie das auf keinen Fall in einer ungespeicherten Blenderdatei, deren Inhalt Ihnen lieb und teuer ist!