



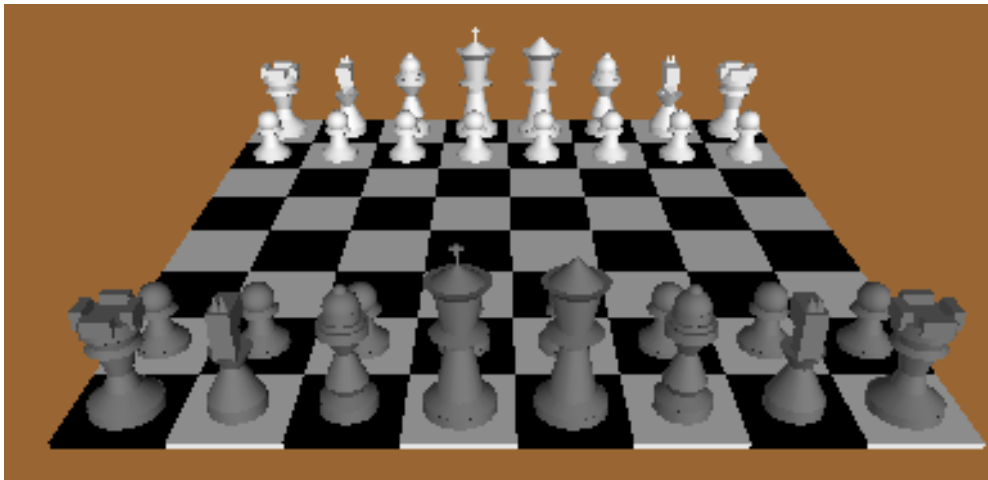
# Vpython – 3D-Programmierung

TDI



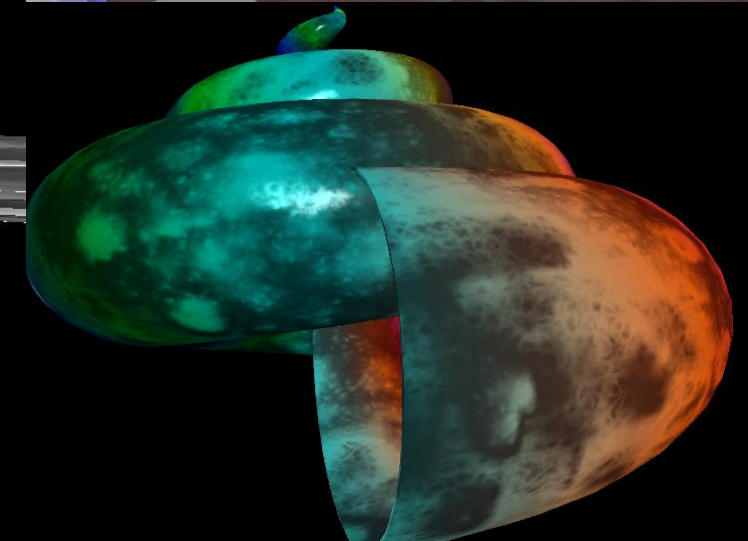
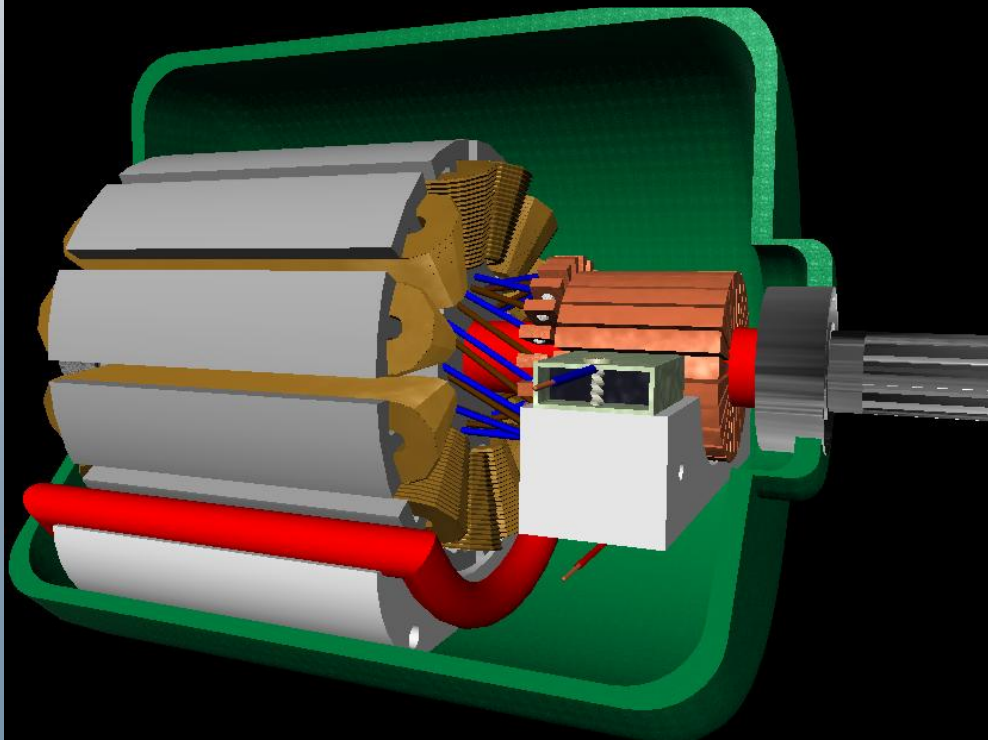
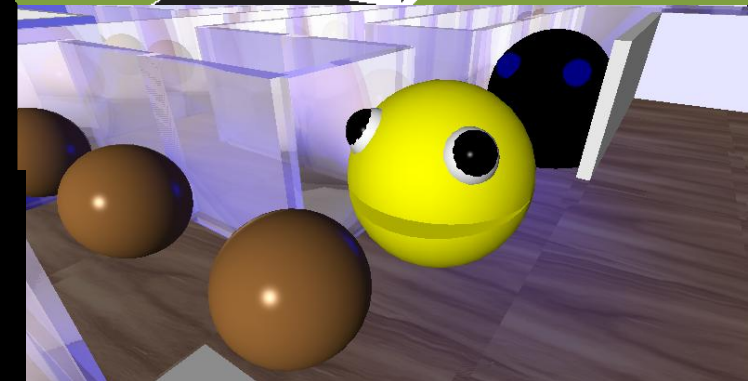
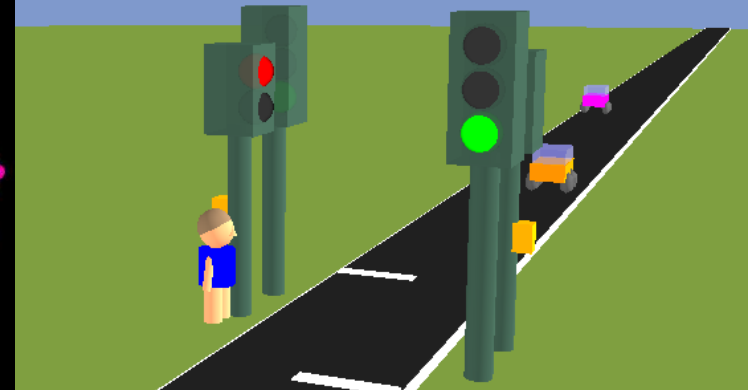
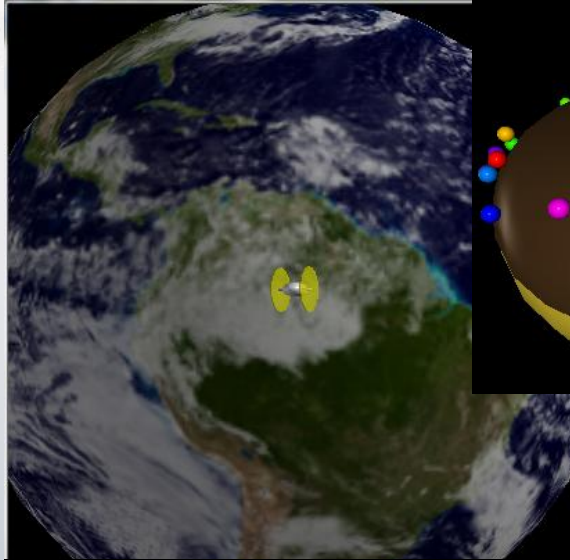
Juli, 2014

- Vpython ist eine Erweiterung zu Python
- bietet Möglichkeit, einfach 3D-Objekte zu erstellen
- selbst erstellte Objekte, z. B. mit Blender, können eingebunden werden





# Weitere Beispiele





# Installation

**Download unter [vpython.org](http://vpython.org)**

**Versionen:**

<b>Python 2.7.x</b>	<b>abwärtskompatibel, nicht weiterentwickelt weit verbreitet Visual bis aktuell Version 6</b>
<b>Python 3.x</b>	<b>geringe Syntaxunterschiede vollständige Unicode-Unterstützung Visual nur bis 5.7.4 2to3.py (Skript, um 2er- in 3er- Skripte zu überführen, liegt 3er-Version bei)</b>



# Weshalb Python?

- Schlank, schön, einfache Syntax => auch für Realschule geeignet
- immer mehr im Kommen
- dynamisch, aber streng typisiert
- „glue-language“: einfache Vermittlung zwischen anderen Sprachen (C++, Java, ...)
- Erweiterung durch zahlreiche Module => ermöglicht nahezu jeden Einsatzzweck
- Unterstützt gängige Programmierparadigmen:
  - rein algorithmisch
  - objektorientiert
  - funktional

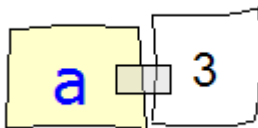


# Philosophie

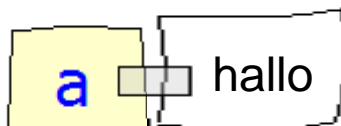
- Interaktive, interpretierte Programmiersprache
- Variablen haben keinen Typ, dieser wird automatisch zugewiesen

Python:

`a = 3`



`a = „hallo“`



gängige Programmiersprachen:



`int a;`



`a = 3;`



# Syntax

- „looks like executable pseudo code“

z. B. `if 3 < x < 5:`                      oder  
      `for i in f.objects:`

- Keine Strichpunkte, geschweifte Klammern o.ä.
- Am Ende einer Anweisung steht ein Doppelpunkt
- Blöcke werden durch Einrückungen („whitespaces“) gekennzeichnet
- Groß- und Kleinschreibung wird unterschieden



# Datentypen

## Einfache Datentypen

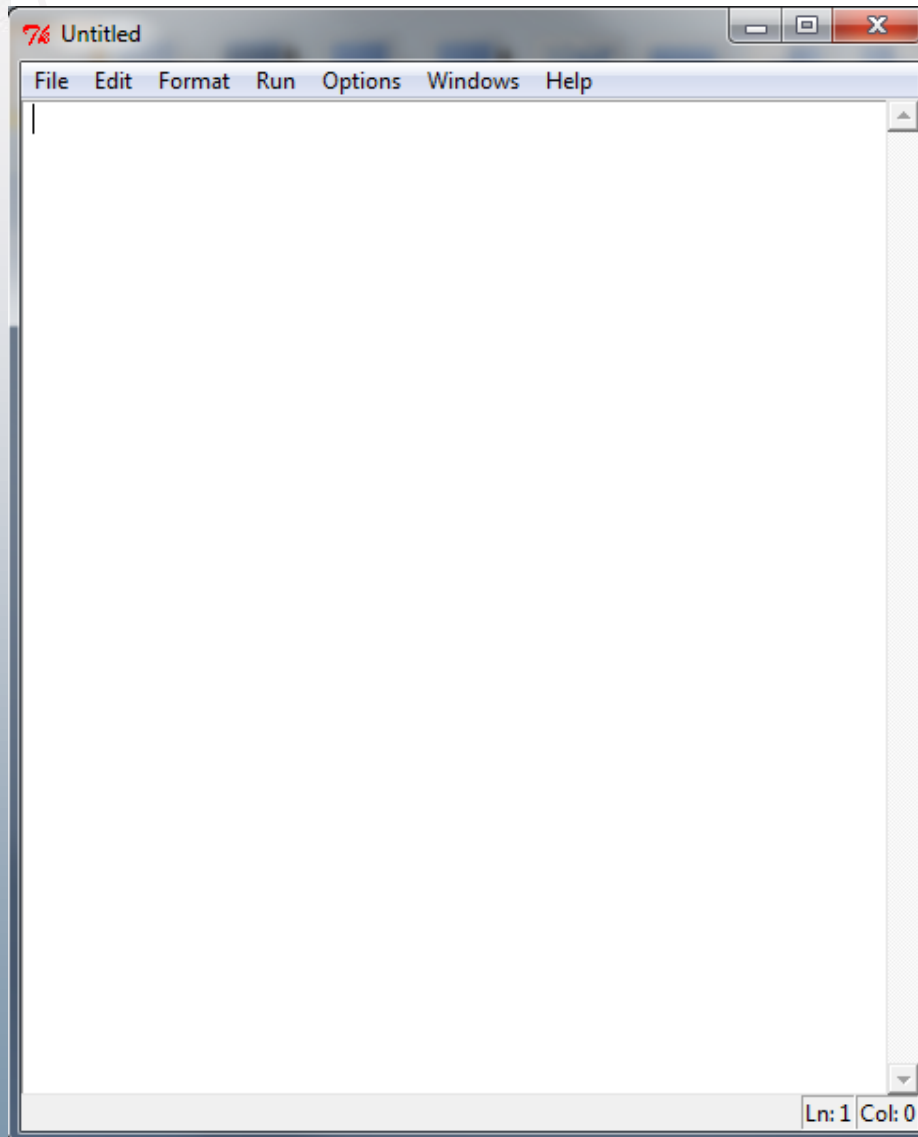
- int(eger): z.B. 2  
*Länge unbegrenzt*
- float: z.B. 3.141...
- string: z.B. „Auto“
- bool: z.B. True <> False  
0 <> 1, „A“ <> ““

## Container-Datentypen

- tuple: z.B. (1,3.141,“A“)  
unveränderbar
- liste: z.B. [1,3.141, „A“]  
veränderbar
- dictionary: z.B.  
{'dog':'Hund', 'cat':'Katze'}



# IDLE – integrierte Entwicklungsumgebung

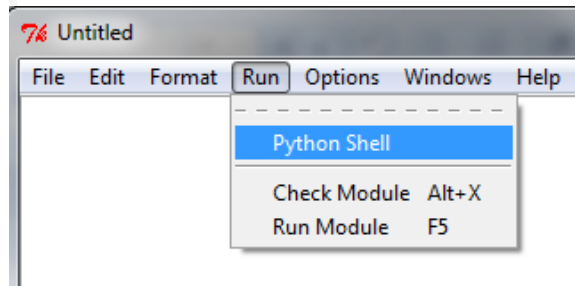


- Konsolen- und Editorfenster
- Syntaxhighlighting
- Codevervollständigung (Tab-Taste)
- Class-/Path-Browser
- Skriptstart mit „F5“





# Python Shell



- automatisch beim Ausführen eines Programms
- manuell über's Menü

## Interaktives Arbeiten -> Eingabe in der Shell:

```
>>>augenzahl = 3
```

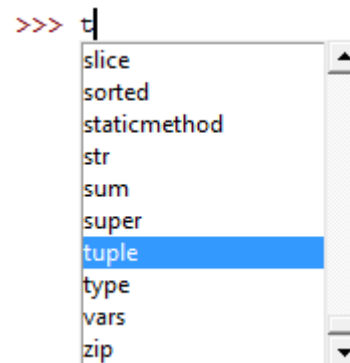
*Frage: welchen Datentyp hat augenzahl?*

```
>>>type(augenzahl)
```

*<class 'int'>*

*mit Codevervollständigung*

```
>>>t (+Tab-Taste)
```



*>>>ty TAB -> type*



# Einfaches Arbeiten in der Shell

`>>> augenzahl * 5`                      15            *(dient als Taschenrechner)*

`>>> augenzahl = augenzahl * 0.5`    *Datentyp augenzahl?*

`>>> type(augenzahl)`                      `<class ,float'>`

`>>> augenzahl`                              1.5

`>>> augenzahl *= 2`                        3.0

`>>> 5 / 2`                                    2.5

*Ganzzahldivision:*

`>>> 5 // 2`                                  2            *(Ergebnis ist int)*

`>>> 6.0 // 2`                                3.0            *(Ergebnis ist float)*

`>>> 5 * 2`                                    10

`>>> 5 ** 2`                                  25            *(Potenzieren)*



# Arbeiten mit Strings (Zeichenketten)

```
>>> einstring = „Zeichenkette“
```

immutable, d. h. einzelne Zeichen  
können nicht verändert werden

```
>>> type(einstring)
```

*<class 'str'>*

```
>>> einstring[0]
```

*erstes Zeichen einstring[0]  
,Z‘*

```
>>> einstring[0] = ,W‘
```

*ERROR*

```
>>> 3 * einstring
```

*'ZeichenketteZeichenketteZeichenkette'*

```
>>> einstring[-1]
```

*,e‘      erstes Zeichen von hinten*

```
>>> einstring[1:6]
```

*,eiche‘*

```
>>> einstring[:-1]
```

*,Zeichenkett‘  
letzter Buchstabe wird gelöscht*



# Ausgabe (print-Befehl)

```
>>> augen = 2  
>>> nase = 1
```

*Trennung durch „**Komma**“ (gemischte Datentypen, keine Typumwandlung):*

```
>>> print(„ich habe“, augen, „Augen und“, nase, „Nase!“)
```

*Ergebnis:*  
*ich habe 2 Augen und 1 Nase!*

*Trennung durch „**Plus**“ (Typumwandlung erforderlich):*

```
>>> print(„ich habe “ + str(augen) + „ Augen und „ + str(nase) + „ Nase!“)
```

*Ergebnis:*  
*ich habe 2 Augen und 1 Nase!*

**Achtung:** Auf Leerzeichen achten (bei Trennung mit Komma automatisch)



# Quelltext

**Kommentare** beginnen mit **#-Zeichen** oder  
sind zwischen **je 3 Anführungszeichen**

Zusammengehörende **Code-Blöcke** werden **engerückt**

In der Regel 4 Leerzeichen

Beginn immer in gleichen Spalte

keine geschweiften Klammern, Strichpunkte etc.

Am **Ende einer Anweisung** steht ein **Doppelpunkt**,

danach Einrückung

Anweisung können durch Backslash (\) in der **nächsten Zeile** fortgeführt werden



# Wiederholungen

## Zahlenbereiche:

```
>>> range(8)
```

die ersten 8 Zahlen ab 0  
Ergebnis: 0,1,2,3,4,5,6,7

```
>>> range(3,6)
```

alle ganzen Zahlen von 3 bis 6  
(ohne die 6)  
Ergebnis: 3, 4, 5

## Feste Anzahl an Wiederholungen (for-Schleife)

(die Laufvariable i nimmt nacheinander jeden Wert an)

```
for i in range(0,3):  
    print (i)
```

Ausgabe: 0  
1  
2

```
for i in ("Auto", 2, 3.14):  
    print (i, type(i))
```

Ausgabe:  
('Auto', <type 'str'>)  
(2, <type 'int'>)  
(3.14, <type 'float'>)



# Wiederholungen

## Bedingte Wiederholungen (while-Schleife)

```
i = 0
while i < 3:
    print (i)
    i = i + 1
```

Ausgabe: 0  
1  
2

## Weitere Anweisungen in Wiederholungen:

**break**

verlässt diese Schleife sofort und macht nach der Schleife weiter

**continue**

beendet diesen Schleifendurchlauf und macht sofort mit dem nächsten Durchlauf weiter



# Bedingungen

- `if ....`: (einseitige B.)
- `elif ....`: für Mehrfachauswahl)
- `else :` (Alternative)

Aus dem Python-Modul ,random' wird die Methode choice eingebunden

```
from random import choice
```

```
optionen = ['Sonne','Wolken','Regen','Sturm']
```

```
wetter = choice(optionen)
```

```
if wetter == 'Sonne':
```

```
    print('badengehen')
```

```
elif wetter == 'Wolken':
```

```
    print('radeln')
```

```
else:
```

```
    print('Regenschirm einpacken')
```





# Methoden

Methoden beginnen mit dem Schlüsselwort **def**

Optional: Kommentar, der bei Aufruf angezeigt wird, folgt in „

Rückgabewert folgt nach Schlüsselwort **return**

Beispiel:

```
from math import sqrt
```

```
def gibWurzel(zahl=0):
```

```
    „optionaler Kommentar, der angezeigt wird“
```

```
    if zahl < 0:
```

```
        return False
```

```
    else:
```

```
        return sqrt(zahl)
```

Aus dem Python-Modul ‚math‘ wird die Methode ‚sqrt‘ (liefert Wurzel einer Zahl) eingebunden

Eine Methode namens gibWurzel wird erstellt, der Parameter ‚zahl‘ ist mit Null vorbelegt

Der Rückgabewert kann unterschiedlichen Datentyp haben (hier ist er entweder eine Zahl oder ein Boolean-Wert)

Aufruf z. B. `gibWurzel()`

```
(zahl=0)
```

```
optionaler Kommentar, der angezeigt wird
```

`gibWurzel(9)` liefert 3.0



# Das visual-Modul:

**Ausprobieren!**

(unter [vpython.org](http://vpython.org) sehr gute und ausführliche Dokumentation)

