1 Einführung in Python 3

- Anfang der 90er Jahre von Guido van Rossum in Amsterdam entwickelt
- seit 2008 Python 3
- Ziel: Einfachheit und Übersichtlichkeit
- nur wenige **Schlüsselwörter**, die nicht für eigene Bezeichner verwendet werden dürfen
- dynamische Verwaltung der Datentypen
- The Zen of Python
- überschaubare, leicht erweiterbare Standardbibliothek

1 The Zen of Python

Ziele und Regeln, 2004, Tim Peters ☺

- Beautiful is better than ugly.	Schön ist besser als hässlich.
- Explicit is better than implicit.	Explizit ist besser als implizit.
- Simple is better than complex.	Einfach ist besser als kompliziert.
- Complex is better than complicated.	Komplex ist besser als undurchschaubar.
- Flat is better than nested.	Flach ist besser als verschachelt.
- Sparse is better than dense.	Spärlich ist besser als beschränkt.
- Readability counts.	Lesbarkeit zählt.
- Special cases aren't special enough to break the rules.	Spezialfälle sind nicht speziell genug, als dass sie die Regeln sprengen dürften.
- Although practicality beats purity.	Obwohl die praktische Anwendbarkeit die Reinheit übertrifft.
- Errors should never pass silently.	Fehler sollten nie schweigend verlaufen.
- Unless explicitly silenced.	Außer man hat sie explizit zum Schweigen gebracht.
- In the face of ambiguity, refuse the temptation to guess.	Im Angesicht der Mehrdeutigkeit, widersage der Versuchung zu raten.
- There should be one– and preferably only one –obvious way to do it.	Es sollten einen — und bevorzugt genau einen — offensichtlichen Weg geben, es zu tun.
- Although that way may not be obvious at first unless you're Dutch.	Obwohl dieser Weg auf den ersten Blick nicht offensichtlich erscheinen mag, außer man ist Holländer.
- Now is better than never.	Jetzt ist besser als nie.
- Although never is often better than *right* now.	Obwohl nie oft besser ist als JETZT SOFORT.
- If the implementation is hard to explain, it's a bad idea.	Wenn die Implementierung schwer zu erklären ist, ist es eine schlechte Idee.
- If the implementation is easy to explain, it may be a good idea.	Wenn die Implementierung einfach zu erklären ist, könnte es eine gute Idee sein.
- Namespaces are one honking great idea – let's do more of those!	Namensräume sind eine glänzende Idee — lasst uns mehr davon machen!

2 Schlüsselwörter

`and`	`continue`	`finally`	`is`	`raise`
`as`	`def`	`for`	`lambda`	`return`
`assert`	`del`	`from`	`None`	`True`
`async`	`elif`	`global`	`nonlocal`	`try`
`await`	`else`	`if`	`not`	`while`
`break`	`except`	`import`	`or`	`with`
`class`	`False`	`in`	`pass`	`yield`

3 Standardbibliothek

Hier einige Pakete der Standarsbibliothek, die zeigen, wie umfangreich die Anwendungsgebiete von Python sind und die unter jeder Entwicklungsumgebung verfügbar sein sollten.

`math, cmath`	mathematische Funktionen (komplexe Zahlen)
`random`	Zufallszahlen oder zufälliges Auswählen aus Mengen
`decimal`	präzise Repräsentatation von Dezimalzahlen
`date, datetime`	Zeit- und Datumsrechnung
`tkinter`	grafische Benutzeroberfläche
`copy`	tiefe Kopien
`cmd`	Kommandozeilenintrepreter
`getpass`	Einlesen von Passwörtern
`webbrowser`	öffnet Standardbrowser
`struct`	Interpretieren von Binärdateien
`hashlib`	Hash-Wert-Berechnungen
`re`	reguläre Ausdrücke
`os, sys`	Schnittstelle zu Betriebssystem (nicht `import from`) und Dateisystem, Zugriff auf Laufzeitumgebung
`shutil`	kopieren und entfernen von Dateien
`ctypes`	Einbinden von C oder C++ Bibliotheken
`setuptool`	Schreiben von Extensions (setzt C-Compiler voraus)
`tempfile`	temporäre Dateien
`concurrent.futures`	Parallelisierung
`threading, multiprocessing`	
`gzip u.a.`	Kompression
`xml`	Einlesen und Schreiben von XML-Dateien
`sqlite3`	Schnittstelle zu Datenbanken
`pickle`	Serialisieren von Objekten
`csv`	Umgang mit csv-Dateien verschiedenster Formate
`socket`	Netzwerkkommunikation
`ftplib`	FTP-Server
`smtplib, poplib, imaplib,email`	Email
`telnet`	Kommunikatiotion mit entfernten Rechner
`pprint, logging`	debuggen, loggen
`doctest, unitest`	automatisiertes Testen
`timeit,cProfile`	Laufzeitmessung
`trace`	Überdeckungsanalyse
⋮	⋮

4 Weitere Module

Darüber hinaus gibt es eine Unzahl weiterer Module, die nachinstalliert werden können, bei einer Entwicklungsumgebung wie *Anaconda* z.T. bereits vorhanden sind.
Unter *https://pypi.python.org* findet man ca. 60 000 Python-Pakete.

`numpy, scipy, mathplotlib`	wissenschaftliches Rechnen, Graphen
`django`	Python als servereitige Programmiersprache im WWW
`Qt, PyQt`	grafische Benutzeroberflächen
`pillow`	Verarbeitung von Rastergrafiken

2 Installation

- unter *http://python.org* Installationsdatei herunterladen, integrierte Entwicklungsumgebung *IDLE* ist bereits enthalten
- Python-Distribution wie *Anaconda* einsetzen
  https://www.continuum.io/downloads/, umfasst auch etliche Erweiterungen, leichtes Nachinstallieren zusätzlicher Module

3 IDLE – Entwicklungsumgebung

Python besitzt eine eingebaute Entwicklungsumgebung namens IDLE (Integrated Development Environment).
Im Konsolenmodus sind die Tastaturkürzel **Alt+n** und **Alt+p** nützlich, um in bereits eingegebenen Befehlen zu blättern. Unter Datei-Neu kann man eine eigene Python-Datei `datei.py` speichern und ausführen.

4 Built-in Eingebaute Funktionen

Die nachfolgenden Funktionen können ohne Einbinden zusätzlicher Module genutzt werden. In der Regel haben sie zusätzliche Parameter in nachfolgenden Klammern `()` .

`#`	Kommentar
`abs`	Berechnet den Betrag einer Zahl.
`all`	Prüft, ob alle Elemente einer Sequenz True ergeben.
`any`	Prüft, ob mindestens ein Element einer Sequenz True ergibt.
`ascii`	Erzeugt einen druckbaren String, der das übergebene Objekt beschreibt. Dabei werden Sonderzeichen maskiert, sodass die Ausgabe nur ASCIIZeichen enthält.
`bin`	Gibt einen String zurück, der die übergebene Ganzzahl als Binärzahl darstellt.
`bool`	Erzeugt einen booleschen Wert.
`bytearray`	Erzeugt eine neue bytearray-Instanz.
`bytes`	Erzeugt eine neue bytes-Instanz.
`callable`	Gibt an, ob eine Instanz aufrufbar ist.
`chr`	Gibt das Zeichen mit einem bestimmten UnicodeCodepoint zurück.
`classmethod`	Erzeugt eine Klassenmethode.
`complex`	Erzeugt eine komplexe Zahl.

`delattr`	Löscht ein bestimmtes Attribut einer Instanz.
`dict`	Erzeugt ein Dictionary.
`dir`	Gibt eine Liste aller Attribute eines Objekts zurück.
`divmod`	Gibt ein Tupel mit dem Ergebnis einer Ganzzahldivision und dem Rest zurück.
`divmod(a, b)`	ist äquivalent zu `(a // b, a % b)`
`enumerate`	Gibt einen Aufzählungsiterator für die übergebene Sequenz zurück.
`eval`	Wertet einen Python-Ausdruck aus.
`exec`	Wertet einen Python-Ausdruck aus.
`filter`	Ermöglicht es, bestimmte Elemente einer Liste herauszufiltern.
`float`	Erzeugt eine Gleitkommazahl.
`format`	Formatiert einen Wert mit der angegebenen Formatangabe.
`frozenset`	Erzeugt eine unveränderliche Menge.
`getattr`	Gibt ein bestimmtes Attribut einer Instanz zurück.
`globals`	Gibt ein Dictionary mit allen Referenzen des globalen Namensraums zurück.
`hasattr`	Überprüft, ob eine Instanz über ein bestimmtes Attribut verfügt.
`hash`	Gibt den Hash-Wert einer Instanz zurück.
`help`	Startet die eingebaute interaktive Hilfe von Python.
`hex`	Gibt den Hexadezimalwert einer ganzen Zahl in Form eines Strings zurück.
`id`	Gibt die Identität einer Instanz zurück.
`import`	Bindet ein Modul oder Paket ein.
`input`	Liest einen String von der Tastatur ein.
`int`	Erzeugt eine ganze Zahl.
`isinstance`	Prüft, ob ein Objekt Instanz einer bestimmten Klasse ist.
`issubclass`	Prüft, ob eine Klasse von einer bestimmten Basisklasse erbt.
`iter`	Erzeugt ein Iterator-Objekt.
`len`	Gibt die Länge einer bestimmten Instanz zurück.
`list`	Erzeugt eine Liste.
`locals`	Gibt ein Dictionary zurück, das alle Referenzen des lokalen Namensraums enthält.
`map`	Wendet eine Funktion auf jedes Element einer Liste an.
`max`	Gibt das größte Element einer Sequenz zurück.
`min`	Gibt das kleinste Element einer Sequenz zurück.
`next`	Gibt das nächste Element des übergebenen Iterators zurück.
`oct`	Gibt den Oktalwert einer ganzen Zahl in Form eines Strings zurück.
`open`	Erzeugt ein Dateiobjekt.
`ord`	Gibt den Unicode-Code eines bestimmten Zeichens zurück.
`pow`	Führt eine Potenzoperation durch.
`print`	Gibt die übergebenen Objekte auf dem Bildschirm in anderen Ausgabegeräten aus.
`property`	Erzeugt ein Managed Attribute.
`hline range`	Erzeugt einen Iterator, mit dem gezählt werden kann.
`repr`	Gibt eine String-Repräsentation einer Instanz zurück.
`reversed`	Erzeugt einen Iterator, der ein iterierbares Objekt rückwärts durchläuft.
`round`	Rundet eine Zahl.
`set`	Erzeugt ein Set.
`setattr`	Setzt ein bestimmtes Attribut einer Instanz auf einen bestimmten Wert.

`sorted`	Sortiert ein iterierbares Objekt.
`staticmethod`	Erzeugt eine statische Methode.
`str`	Erzeugt einen String.
`sum`	Gibt die Summe aller Elemente einer Sequenz zurück.
`tuple`	Erzeugt ein Tupel.
`type`	Gibt den Datentyp einer Instanz zurück.
`vars`	Gibt das Dictionary `x.__dict__` zurück, wenn eine Instanz x übergeben wird. Ohne Argument ist `vars` äquivalent zu `locals`.
`zip`	Fasst mehrere Sequenzen zu Tupeln zusammen, um sie beispielsweise mit einer for-Schleife zu durchlaufen.

5 Python - Turtle

1 Einführung

Das Turtle-Module lässt eine programmierbare Schildkröte über die Zeichenfläche laufen und bei abgesenktem Stift zeichnen.

Man kann direkt im IDLE die Programmschritte eingeben. Dann sieht man sofort was passiert.

```
>>> from turtle import *
>>> forward(100)
```

Im IDLE kann man auch ein neues Fenster öffnen, die Datei unter `name.py` abspeichern und dann das Programm schreiben.
Mit `F5` wird das Programm gestartet.

```
from turtle import *    # bzw. import turtle

t1 = Turtle()           # dann t1 = turtle.Pen(shape='turtle')
t1.shape('turtle')
t1.color("red","blue")
t1.fd(100)
def quadrat():
    for i in range(4):
        fd(seite)
        rt(90)
seite=100
quadrat()
mainloop()   # Endlosschleife, hier nicht zwingend
```

2 Turtle-Befehle

Bewegung und Zeichnen

`forward(40)`	`fd(40)`	40 Pixel vorwärts
`back(30)`	`bk(30)`	30 Pixel zurück
`right(90)`	`rt(90)`	dreht die Turtle um 90° weiter nach rechts
`left(60)`	`lt(60)`	dreht die Turtle um 60° weiter nach links
`circle(30)`		zeichnet einen Kreisbogen links mit dem Radius 30 Pixel
`circle(20,90)`		zeichnet einen Kreisbogen nach links mit einem Radius von 20 Pixel und einem Winkel von 90°
`dot(3,?red?)`		zeichnet einen Punkt mit dem Durchmesser 3 Pixel und der Farbe rot an der aktuellen Position
`goto(100,200)`	`setpos(100,200)`	setzt die Turtle auf den Punkt (100|200) und zeichnet
`setheading(270)`	`seth(270)`	dreht die Turtle auf 270° (West), unabhängig von dem vorher eingestellten Winkel
`pu()`	`up()`	der Zeichenstift wird hochgenommen. (die Turtle bewegt sich danach ohne zu zeichnen)

`pd()`	`down()`	der Zeichenstift wird heruntergesetzt. Die Turtle zeichnet wieder.
`write('Hallo Welt',font=('Arial',16))`		Schreibt Hallo Welt in Schriftart Arial und Schriftgröße 16 an der aktuellen Position
`hideturtle()`	`ht()`	Turtle wird unsichtbar
`showturtle()`	`st()`	Turtle wird sichtbar
`s1=stamp()`		stempelt den Umriss der Turtle an der aktuellen Position. Stempelname ist s1
`clearstamp(s1)`		löscht Stempel s1
`clearstams(n)`		lösche alle, die ersten, die letzten von n Stempeln. Wenn die Klammer leer ist, werden alle gelöscht. Wenn n > 0 werden die ersten n Stempel, sonst die letzten n Stempel gelöscht

Position und Winkel bestimmen

`position()`	`pos()`	gibt die (x|y)-Koordinaten der Turtle zurück
`towards(x,y)`		gibt den Abstand bis zum Punkt (x|y) zurück
`xcor()`		gibt die x-Koordinate der Turtle zurück
`ycor()`		gibt die y-Koordinate der Turtle zurück
`heading()`		gibt den Winkel der Turtle zurück
`distance(x,y)`		gibt den Abstand der Turtle zum Punkt (x|y) zurück

Farben, Füllen, Aussehen

`width(3)`	`pensize(3)`	setzt Linienbreite auf 3 Pixel
`color('red','yellow')`		die Zeichenfarbe ist rot, die Hintergrundfarbe ist gelb
`fillcolor('blue')`		die Hintergrundfarbe wird auf blau gesetzt.
`begin_fill()`		färbt folgende geschlossene Fläche mit aktueller Hintergrundfarbe aus.
`end_fill()`		färbt vorher geschlossene Fläche mit angegebener Farbe.
`bgcolor('pink')`		die Hintergrundfarbe wird auf pink gesetzt
`bgpic('dateipfad')`		es wird ein Bild als Hintergrund geladen
`shape('turtle')`		legt das Aussehen der Turtle fest ('arrow', 'turtle', 'circle', 'square', 'triangle', 'classic').

Grafikbildschirm

`setworldcoordinates (-10,-10,10,10)`	legt das Koordinatensystem der Zeichenfläche auf die entsprechenden Koordinaten fest. (Gibt u. a. Probleme mit circle(20,90))
`title("Willkommen beim Turtle!")`	Titelzeile im Grafikfenster
`setup(width=200, height=200, startx=0, starty=0)`	Das Grafikfenster wird auf die angegebenen Werte eingestellt.
`bye()`	schließt das Grafikfenster
`clear()`	löscht die Zeichnung der Turtle z. B. `paula.clear()`
`clearscreen()`	löscht den Grafikbildschirm

Maus und Tastatur	
`listen()`	setzt den Focus auf das Fenster, um auf Maus und Tastatur zu reagieren.
`onkey(funk,'T')`	funk wird ausgeführt wenn Taste 'T' gedrückt wird
`onkeyrelease(funk,'T')`	funk wird ausgeführt wenn Taste 'T' losgelassen wird
`onkeypress(funk,['T'])`	funk wird ausgeführt solange (irgendeine) Taste gedrückt wird
`onscreenclick(funk)`	funk wird ausgeführt, wenn der Bildschirm angeklickt wird
`onclick(test)`	beim Klicken auf die Turtle wird test(x|y) mit den Mauskoordinaten (x|y) ausgeführt. Die aufgerufene Prozedur muss so aussehen def test(x,y):
`onrelease(test2)`	test2(x|y) wird beim Loslassen mit den Mauskoordinaten (x|y) ausgeführt. def test2(x|y):
`ondrag(test3)`	test3(x|y) wird beim Ziehen mit den Mauskoordinaten (x|y) ausgeführt. def test3(x|y): auch: ondrag(goto)
Variablen u.a.	
`a=2`	weist der Variablen a den Wert 2 zu.
`fd(3*a)`	zeichnet eine Linie mit 6 Einheiten, wenn a gleich 2 ist
`a+=3` oder `a-=3`	vergrößert (verkleinert) den Wert der Variablen a um 3. Wenn a gleich 2 ist, dann wird a=5 bzw. a=-1)
`x=a**3`	weist der Variablen x den Wert a^3 zu. Wenn a=2 dann ist x=8
`z=randint(1,6)`	vorher das Modul random importieren: `from random import *` z bekommt als Wert eine Zufallszahl zwischen 1 und 6 (würfeln)
`y:=3x+2`	es können beliebige Formeln als Zuweisung gewählt werden
`zahl str = textinput('Gib eine Zahl ein:')`	Du kannst in einem Dialog einen beliebigen Text eingeben.
`kommazahl = float(zahl str)`	Wenn der Text aus Ziffern besteht, wird der Text in eine Kommazahl (float) umgewandelt.
`zahl = int(zahlstr)`	Wenn der Text aus Ziffern besteht, wird der Text in eine ganze Zahl (integer) umgewandelt.

An der Eingabeaufforderung kann man sich zu unklaren Befehlen helfen lassen.

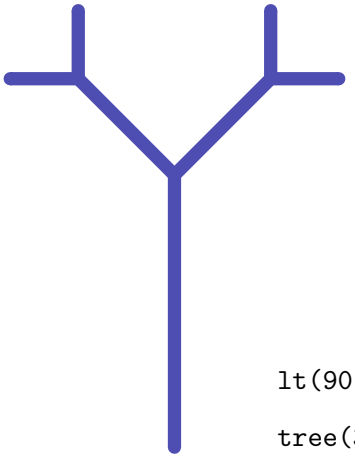
```
>>> help(color)
Help on function color in module turtle:

color(*args) ...
```

6 Python - Rekursion mit der Turtle

So wie in HASKELL kann es auch in PYTHON rekursive Funktionen geben. Als Beispiel nehmen wir uns einen rekursiven Baum. Wie und mit welchen Parametern erfogt nun die Abarbeitung.

```
1  def tree (laenge):
2   if laenge < 1 : return() # terminierender Fall
3    fd (laenge)
4    rt (45)
5    tree (laenge/2) # rekursiver Aufruf
6    lt (90)
7    tree (laenge/2) # rekursiver Aufruf
8    rt (45)
9    back (laenge)
10   return ()
```

`tree(6)`
$6 \not< 1$ `(kein return)`
`fd(6)`
`rt(45)`
`tree(6/2)` $\longrightarrow$ `tree(3)`
$3 \not< 1$ `(kein return)`
`fd(3)`
`rt(45)`
`tree(3/2)` $\longrightarrow$ `tree(1.5)`
$1.5 \not< 1$ `(kein return)`
`fd(1.5)`
`rt(45)`
`tree(1.5/2)` $\longrightarrow$ `tree(0.75)`
$0.75 < 1$
`lt(90)` $\Longleftarrow$ `return()`
`tree(1.5/2)` $\longrightarrow$ `tree(0.75)`
$0.75 < 1$
`rt(45)` $\Longleftarrow$ `return()`
`back(1.5)`
`lt(90)` $\Longleftarrow$ `return()`
`tree(3/2)` $\longrightarrow$ `tree(1.5)`
weiter wie oben, bis `tree(1.2)` vollständig abgearbeitet wurde.
$\vdots$
`rt(45)` $\Longleftarrow$ `return()`
`back(3)`
`lt(90)` $\Longleftarrow$ `return()`
`tree(6/2)` $\longrightarrow$ `tree(3)`
weiter wie oben, bis `tree(3)` vollständig abgearbeitet wurde.
$\vdots$
`rt(45)` $\Longleftarrow$ `return()`
`back(6)`
`return()`



Jeder Funktionsaufruf muss vollständig bis zu einem `return()` abgearbeitet werden.

Man erkennt, bis auf `tree(6)` gibt es immer einen rechten und einen linken Unterbaum.

Aufgaben

1. Analysiere die dargestellten Zeichnungen und programmiere sie rekursiv nach.