Python zur Lösung von AvA Übung 1

VON MORITZ FEY

Übersicht

- Einführung
- Socket-Schnittstellen
- Datei-Zugriff
- Aufbau der Nachrichten und Serialisierung
- Besonderheiten von Python
- Realisierung des Programms

Einführung - Geschichte

- Entwicklung seit Anfang der 1990er
- Chefentwickler Guido van Rossum
- Name ist Anspielung auf Monty Python, nicht die Schlange
- Version 1.0 erschien 1994
- Version 2.0 erschien 2000 → 2.7.10
- Version 3.0 erschien 2008 → 3.5.0

Einführung

→ Python 2.7.x

„Pseudo-Code der lauffähig ist.“

Code-Beispiel:

```
print "vor der Schleife"
for i in range(1,5):
    print i
print "nach der Schleife"
```

```
vor der Schleife
1
2
3
4
nach der Schleife
```

Einführung

- Python ist multiparadigmatisch (objektorientiert, strukturiert, Ansätze funktionaler Programmierung)
- Dynamische Typisierung
- Whitespace zur Strukturierung des Codes
- Sehr umfangreiche Standard-Bibliothek
- Zahlreiche (~69.000) Pakete im Python Package Index (https://pypi.python.org/pypi)

Einführung

- Interpreter-Sprache
- Versch. Interpreter für verschiedene Betriebssysteme verfügbar
- Mit Cython auch schnelle Binär-Module
- Exzellente Online-Dokumentation der Programmiersprache
- Referenzzählung für Speicherverwaltung

Interaktive Shell


```
C:\Users\mofey>python
Python 2.7.10 (default, May 23 2015, 09:40:32) [MSC v.1500 32 bit (Intel)] on wi
n32
Type "help", "copyright", "credits" or "license" for more information.
>>> import random
>>> random.random()
0.400793327498231 16
>>> import time
>>> time.localtime()
time.struct_time(tm_year=2015, tm_mon=11, tm_mday=12, tm_hour=19, tm_min=12, tm_
sec=14, tm_wday=3, tm_yday=316, tm_isdst=0)
>>>
```

Version 2 vs. Version 3

- aktuell zwei unterstützte Versionen
- print Statement mit print() Methode ersetzt
- Änderungen „unter der Haube"
- In Python 3 standardmäßig alle Strings Unicode
- Anfänglich viele Libraries nicht mit Python 3 kompatibel
- Python 2 erhält nur noch Sicherheitsupdates

UDP-Server


```
def run_server(self):
    buffer_size = 1024
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.bind((self.address, int(self.port)))

    print "my neighbours are: %s" %(self.get_list_of_hosts_as_string(self.neighbours))

    while self.server_running:
        try:
            m, addr = s.recvfrom(buffer_size)
            msg = Message.decode_message(m)

            if msg.msgtype == Message.MSG_TYPE_APP:
                self.handle_application_message(msg)

            if msg.msgtype == Message.MSG_TYPE_CONTROL:
                self.handle_control_message(msg)
        except socket.timeout:
            pass
```

UDP-Client


```
def send_msg_to_specific_neighbour(self, msg, node_id):

    for host in self.neighbours:
        if host.node_id == node_id:
            s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            s.sendto(Message.encode_message(msg), (host.address, int(host.port)))
            self.print_message_to_cmd(msg, False, host.node_id)
```

Lambda-Ausdrücke und Filter

- Lambda Ausdrücke und Filter auf Listen

```
neighbour_nodes = filter(lambda node: node.node_id in neighbours, host_list)
```

vs.

```
neighbour_nodes = []
for node in host_list:
    if node.node_id in neighbours:
        neighbour_nodes.append(node)
```

Datei-Zugriff

- Lesen einer Datei in Python

```
with open("/etc/motd", "r") as f:
    for line in f:
        print "length of line: %i" %(len(line))
        print line
```

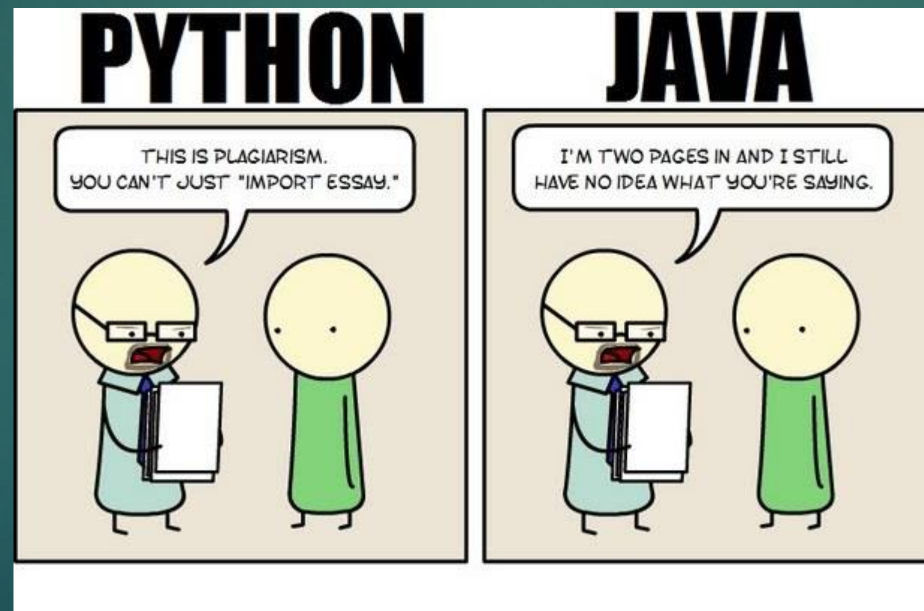
```
with open(node_list_file) as file:
    content = file.readlines()

for line in content:
    new_node = self.get_node_from_line(line)
    if new_node.node_id == node_id:
        address = new_node.address
        port = new_node.port
    else:
        nodelist.append(new_node)
return address, port, nodelist
```

Aufbau der Nachrichten

- Datentyp mittels „namedtuple“ aus dem „collections“-Modul erstellt.

```
Message = collections.namedtuple('Message', ['msgtype', 'task', 'message', 'source'], verbose=False)
```



Quelle: http://pythonthusiast.pythonblogs.com/gallery/230/python-vs-java-726367-copy.jpg

Aufbau der Nachrichten

- **msgtype**: „app“ oder „control“
- **task**: Untertyp für msgtype, verschiedene „app“ Nachrichten für Teilaufgaben
- **message**: eigentliche Nachricht als String
- **source**: ID des Knoten, der die Nachricht verschickt hat

Serialisierung

- Serialisierung der Nachrichten mittels TOML (Tom‘s Obvoius, Minimal Language)
- Eigentlich als Format für Konfigurationsdateien gedacht, eignet sich jedoch für die Serialisierung von Objekten
- Alternative zu JSON oder YAML
- Besser lesbar für Menschen
- Implementierung für Python verfügbar

Serialisierung


```
[[comments]]
author = "Nate"
text = "Great Article!"

[[comments]]
author = "Anonymous"
text = "Love it!"
```

```
{
    "comments" : [
        {
            "author" : "Nate",
            "text" : Great Article!"
        },
        {
            "author" : "Anonymous",
            "text" : Love It!"
        }
    ]
}
```

Quelle: http://npf.io/2014/08/intro-to-toml/

Serialisierung

- Im Fall der Nachrichten in meinem Programm recht minimalistisch

msgtype=“app“

task=“spreadRumor“

message=“filthyRumor“

source=“1“

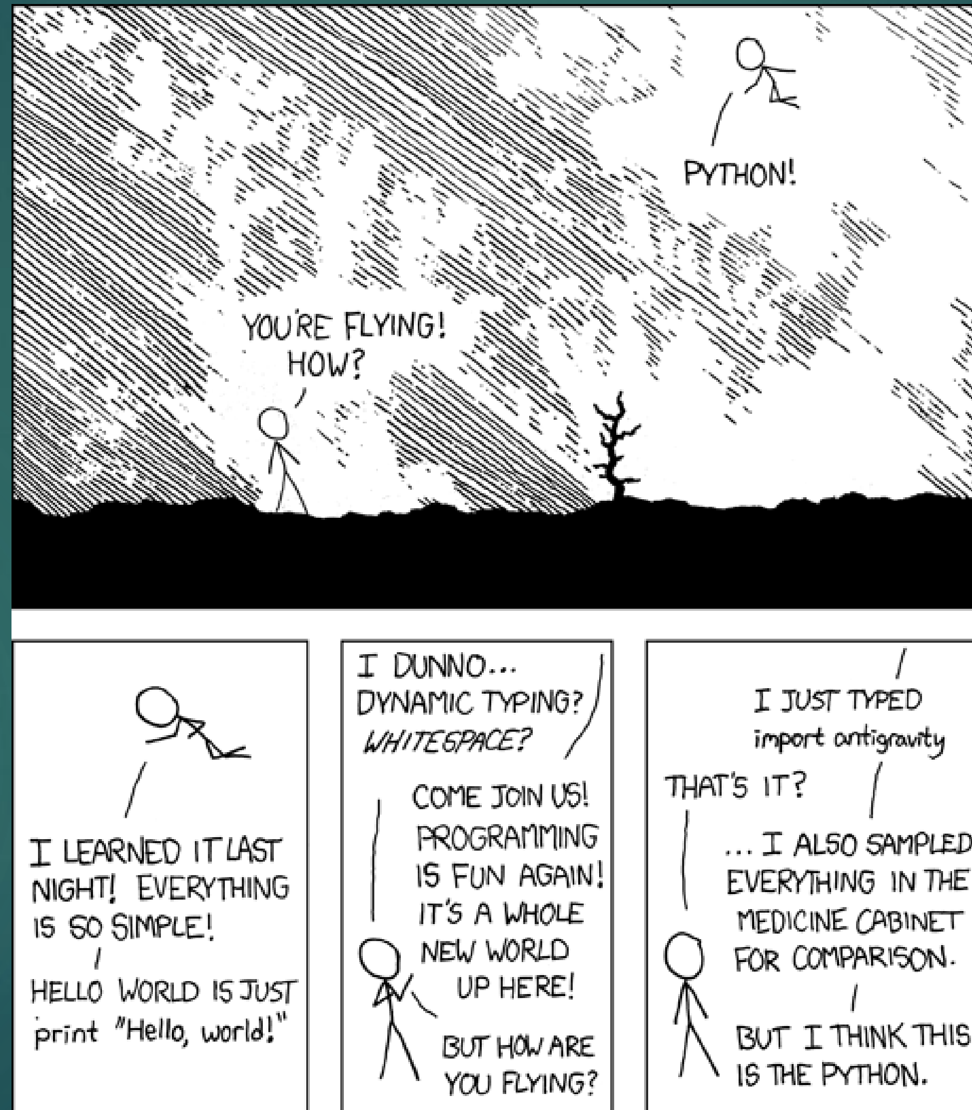
Besonderheiten

- Mehrfachverzweigung durch if-elif-else
- and, or, not statt &&, | |, ! → Code eher wie natürliche Aussage
- Statt Arrays Listen → auch gemischte Typen [4, ‘h‘, 3, ‘e‘]
- beliebig große Integer
- mehrere return-Werte in einer Methode

Realisierung des Programms

- Node.py als zentrale Quell-Datei mit der Implementierung des Knotens als Klasse
- nodeStarter.py parst Kommandozeilenargumente und startet den Knoten
- graphGen.py erstellt Graphen

Fragen?



Quelle: https://pbs.twimg.com/media/B38J1JTIUAAW1NS.jpg