Einführung in die Computerlinguistik – Einführung in Python (2)

Dozentin: Wiebke Petersen

11. Foliensatz

Kurzwiederholung der letzten Sitzung

- **Textausgabe** mit `print("text")`
- **Textausgabe** mit Platzhaltern: `print("%s") % ("text")`
  (alternativ `"%i"` oder `"%f"` für Integer bzw. Floats
- **Texteingabe** mit `raw_input()` und `raw_input("Your name: ")`
- **Variablenzuweisung** mit `variable = value`
- **Rechnen** mit `+ - * / % **`, z. B. `area = 3.14 * 2 * radius`
- **Stringkonkatenation** mit `"+"`, z. B. `"Haus" + "bau"`
- Kontrollstruktur **if/else** für Verzweigungen:
  `if (1): ... else: ...`
- **while**-Schleife für wiederholtes Ausführen eines Programmteils:
  `while (1): ...`
- Achten Sie immer auf Zeilenumbrüche und die korrekte Einrückung!

Häufige Fehler bisher

- Fehlende Klammern: `area = (3.14 * 2 * radius`
- Falsche Einrückung:

```
1 if (radius < 0):
2   print "Radius must be positive!"
3   exit()
4 else:
5 area = 3.14 * 2 * radius
```

- Falsche Datentypen:

```
1 # Typkonvertierung mittels float(...) notwendig
2 radius = raw_input("Please enter the circle radius: ")
3 area = 3.14 * 2 * radius
```

```
1 # Vergleich mit einem String notwendig, also "42" statt 42
2 if (password == 42):
```

Dateien lesen und schreiben (Variante 1)

- Öffnet die Datei mit Hilfe eines Blocks
- Nach dem Ende des Blocks wird die Datei automatisch geschlossen

Dateien lesen

Gibt die Datei "input.txt" aus dem aktuellen Verzeichnis Zeile für Zeile aus

```
1 with open("input.txt", "r") as file:
2   for line in file:
3     print line,
```

Dateien schreiben

Kopiert den Inhalt aus "input.txt" nach "output.txt"

```
1 with open("input.txt", "r") as input:
2   with open("output.txt", "w") as output:
3     for line in input:
4       output.write(line)
```

Dateien lesen und schreiben (Variante 2)

- Öffnet die Datei und speichert eine Referenz in einer Variablen
- Nachteil: Die Datei muss von Hand geschlossen werden

Dateien lesen

```
1 file = open("input.txt", "r")
2 for line in file:
3   print line
4 file.close()
```

Dateien schreiben

```
1 input = open("input.txt", "r")
2 output = open("output.txt", "w")
3 for line in input:
4   output.write(line)
5 input.close()
6 output.close()
```

Suchen mit regulären Ausdrücken in Python

```
1 import re
2 password = raw_input("Please enter password: ")
3 while (not re.search(r"^42$", password)):
4   print "Wrong password! Access denied."
5   password = raw_input("Please enter password: ")
6 print "Correct password! Welcome."
```

- Um reguläre Ausdrücke nutzen zu können, müssen wir die Bibliothek "re" laden: `import re`
- Die Suche erfolgt mittels der Funktion `re.search(regex, input)`
- Best practice: reguläre Ausdrücke als sog. "raw string" eingegeben: `r"raw string"`
- Nebenbei: Wahrheitswerte lassen sich mit `and`, `or` und `not` verknüpfen
- Übung: Schreiben Sie ein Programm, dass Passwörter akzeptiert, die mindestens drei Großbuchstaben enthalten

Groß- und Kleinschreibung

- Mittels sog. Flags (Schalter) können wir das Verhalten von regulären Ausdrücken in Python beeinflussen
- Ein wichtiger Schalter ist `re.IGNORECASE`, oder kürzer, `re.I`: er sorgt dafür, dass der reguläre Ausdruck keinen Wert auf die Groß- und Kleinschreibung legt:

```
1 # Trifft nicht
2 re.search(r"hans", "Hans ist im Haus.")
3 # Trifft
4 re.search(r"hans", "Hans ist im Haus.", re.I)
```

- Alternativ können wir den Schalter auch im regulären Ausdruck selbst setzen: `re.search(r"(?i)a", "A")` trifft, da "(?i)" `re.I` entspricht

Gruppieren und Speichern (1)

- Runde Klammern in einem regulären Ausdruck dienen nicht nur zur Veroderung, sondern auch zur Gruppierung und Speicherung
- Der Ausdruck "^(true|false)$" trifft sowohl auf den String "true", als auch auf "false", und merkt sich gleichzeitig, welcher Wert angetroffen wurde
- Die gespeicherten Werte lassen sich in Python über den Rückgabewert der Funktion `re.search()` abrufen:

```
1 match = re.search(r"(true|false)", "true false true")
2 print match.group(1) # true
```

- `match.group(1)` beinhaltet den Teil des Strings, auf den die erste Klammer getroffen hat
- `match.group(0)` beinhaltet den gesamten Teil des Strings, der getroffen wurde

Gruppieren und Speichern (2)

- Die Nummer der Gruppen ergibt sich, indem man die öffnenden Klammern von links nach rechts betrachtet
- Die erste Klammer bekommt die Nummer 1, die zweite Klammer die Nummer zwei, usw.
- Dies gilt auch für verschachtelte Klammern. Beispiel:

```
1 match = re.search(r"^(\d\d(\d)).*$", "123abc")
2 print match.group(0) # 123abc
3 print match.group(1) # 123
4 print match.group(2) # 3
```

Gruppieren und Speichern (3)

- Die Methode `re.search(regex, input)` sucht immer nur nach dem ersten Treffer des regulären Ausdrucks innerhalb von input. Beispiel:

```
1 match = re.search(r"\w", "abc123")
2 print match.group(0) # a
```

- Möchte man alle Treffer sehen, so sollte man die Methode `re.finditer(regex, input)` in Kombination mit einer for-Schleife benutzen. Beispiel:

```
1 for match in re.finditer(r"\w", "abc123"):
2   print match.group(0) # a b c 1 2 3
```

Suchen und Ersetzen

- `re.sub(regex, replace, input)` führt eine Suche nach dem regulären Ausdruck `regex` in `input` durch, und ersetzt den getroffenen Teil durch `replace`

```
1 text = "The gas price just went from 0.94 Mark to 1.05 Mark!"
2 text = re.sub(r"Mark", "EUR", text)
3 # "The gas price just went from 0.94 EUR to 1.05 EUR!"
```

- Beim Ersetzen kann über `\g<Gruppennummer>` auf die jeweilige Klammergruppe zugegriffen werden. Beispiel:

```
1 text = "The gas price just went from 0.94 Mark to 1.05 Mark!"
2 text = re.sub(r"Mark", "EUR", text)
3 text = re.sub(r" (\d+\.\d+) (.+) (\d+\.\d+) ", \
4   " \g<3> \g<2> \g<1> ", text)
```

Zusammenfassung bisher

- Dateien öffnen mittels `open(file, "r")` (lesen) oder `open(file, "w")` (schreiben)
- Aus Datei lesen mit `for line in file: ...`. Schreiben mit `file.write(string)`
- `re.search(regex, string)` bzw. `re.finditer(regex, string)` zur Suche nach dem regulären Ausdruck `regex` in `string`
- Flag `re.I` oder `r"(?i)..."` zum Ignorieren der Groß- und Kleinschreibung
- Runde Klammern im regulären Ausdruck zum Speichern des getroffenen Teilstrings: `re.search(r"([a-z]+)", string)`
- Rückgabewert von `re.search()` bzw. `re.finditer()` erlaubt Zugriff auf die gespeicherten Klammerinhalte: `match.group(1)`
- Suchen und Ersetzen mittels `re.sub(regex, replace, input)`
- Zugriff auf Klammerinhalte beim Ersetzen mittels `\g<Gruppennummer>`

Übungseinheit

1. Überlegen Sie sich, wie das Programm `names.py` funktioniert (wählen Sie dazu `names.txt` als Inputdatei) und ändern Sie es so, dass es die Namen in der Form "Nachname, Initial." ausgibt
2. Schreiben Sie ein Programm, das die Zahl der unbestimmten Artikel, die in einem Text vorkommen, zählt

Hausaufgabe (Abgabetermin: 10.01.2011)

1. Schreiben Sie ein Programm, das eine Datei mit Passwörtern einliest (jede Zeile beinhaltet ein Passwort, siehe passwords.txt auf der Homepage) und für jedes Passwort prüft, ob dieses die folgenden vier Merkmale aufweist
   - es ist zwischen 6 und 8 Zeichen lang und
   - es enthält mindest einen Großbuchstaben und
   - es enthält mindestens zwei Ziffern und
   - es endet auf ein Sonderzeichen (verwenden Sie die Zeichenklasse `[.?!;,\\/+\-*%=\"'$&()\[\]#_]`)

   Es muss nicht alles in einem regulären Ausdruck geprüft werden! **Wieviele Passwörter aus der Datei passwords.txt akzeptiert ihr Programm?**
2. Bearbeiten Sie Aufgabe 2 von Folie 13.

- Für einen BN reicht die Bearbeitung von Aufgabe 2
- Senden Sie die Aufgaben bitte per E-Mail an `buecker@phil-fak.uni-duesseldorf.de`. Eine Abgabe in Druckform ist dann nicht nötig.