Einführung in die Computerlinguistik
Übung 2

Laura Kallmeyer

Sommersemester 2014, Heinrich-Heine-Universität Düsseldorf

Offizielle Python Seite, auf der man jede Menge Dokumentation findet: `http://www.python.org/`

Link zu einem Python Tutorial: `http://docs.python.org/tutorial/`

Das im Kurs verwendete Python findet man als Icon im Menü Programmierung (CL Active Python).

Die Dateien, die im Verzeichnis `X/PC-Pools/Transfer/Python/EinfCL` stehen, können direkt in den Python Interpreter geladen werden.

1. **Einführung: Funktionen und erste Konntrollstrukturen**

   Rufen Sie in dem Fenster, das sich nach Klicken auf das Icon öffnet, den python Interpreter auf:

   `python`

   Hier ein paar Dinge zum Ausprobieren:

```
>>> 1+2
3
>>> "Hello World"
'Hello World'
>>> print "Hello World"
Hello World
```

   Definition und Aufruf einer kleinen Funktion:

```
>>> def start():
...     print "Hello World"
...
>>> start()
Hello World
```

   Definition einer Funktion mit `def`, gefolgt von dem Namen, den man der Funktion geben möchte. Der Name ist gefolgt von einer Liste der Argumentvariablen in Klammern.

   Syntax: `def Funktionsname(Variable1, Variable2, ...):`

   Das was innerhalb der Funktion, also immer wenn man diese aufruft, ausgeführt werden soll, steht in einem eingerückten Block, der in der nächsten Zeile beginnt. Befehle, die zu einem Block gehören, müssen immer gleich eingerückt werden!

   Weitere Strings und Funktionen:

```
>>> string1='Hello World'
>>> string1
'Hello World'
>>> def pprint(string):
...     print("this is the string:"),
...     print(string)
...
>>> pprint(string1)
this is the string:
```

```
Hello World
>>> def alternative_print(x):
...     if x:
...             print("What a nice day!")
...     else:
...             print("Better stay in bed today")
...
>>> alternative_print(1)
What a nice day!
>>> alternative_print(0)
Better stay in bed today
```

Der Befehl `string1='Hello World'` macht mehrere Dinge gleichzeitig: Er führt eine neue Variable mit Namen `string1` ein und weist dieser einen Wert zu. Wertzuweisungen werden mit dem Gleichheitszeichen `=` gemacht, Gleichheitsabfragen dagegen mit `==`:

```
>>> anzahl = 10
>>> anzahl == 9+1
True
```

Der Befehl `print` gibt sein Argument (einen String) aus und beginnt anschließend eine neue Zeile, es sei denn, sein Aufruf ist von einem Komma gefolgt. In diesem Fall wird eine anschließende weitere Ausgabe in dieselbe Zeile geschrieben.

Die letzte Funktion enthielt schon eine einfache Kontrollstruktur, mit der das Ausführen einzelner Blöcke an Bedingungen geknüpft werden kann, nämlich ein `wenn Bedingung, dann Block1, sonst Block2`. Die python Syntax hierfür ist

```
if Bedingung:
   Block1
else:
   Block2
```

Wichtig: `if` und `else` sind auf einer Einrückungsstufe und ebenso die beiden eingebetteten Blöcke.

Aufgabe: Definieren Sie eine Funktion `diff2(n)`, die als Argument eine Zahl, bezeichnet durch die Variable `n`, nimmt, und, falls `n > 10`, den Wert `n+2` zurückgibt, und sonst `n-2`. Einen Wert zurückgeben kann man mit `return(...)`.

Im folgenden Beispiel kommt eine weitere Kontrollstruktur, nämlich `solange Bedingung gilt, führe Block aus`. Hierfür ist die Syntax:

```
while Bedingung:
   Block
```

Hier muss man natürlich dafür sorgen, dass beim Ausführen von `Block` etwas verändert wird, was in der Bedingung abgefragt wird, so dass die Bedingung irgendwann aufhört, wahr zu sein. Andernfalls hat man eine Endlosschleife.

Beispiel: Fibonaccizahlen. (Die ersten beiden sind 1 und 1, jede weitere ist die Summe der beiden vorherigen.)

```
>>># Fibonacci series:
... a,b = 1,1
>>> while a < 10:
...     print a
...     a,b = b, a+b
```

```
...
1
1
2
3
5
8
>>>
```

Der Ausdruck `a,b = ...` fasst zwei Wertzuweisungen zusammen: `a,b = 1,1` steht für das Nacheinanderausführen von `a=1` und `b=1`. Dabei liegen beiden Wertzuweisungen auf der rechten Seite die alten Werte zugrunde. In `a,b = b, a+b` wird also `b` auf die Summe der alten Werte von `a` und `b` gesetzt.

Aufgabe: Definieren Sie eine Funktion `fib(n)`, die die `n`-te Fibonacci Zahl zurückgibt. Man benötigt innerhalb der `while`-Schleife entweder einen Zähler, der bei jedem Durchlauf inkrementiert wird, oder man muss die Argumentvariable runterzählen. Dies kann man mit dem Operator `+=` oder (im Falle des Runterzählens) `-=` machen: Ist z.B. `count` der Zähler, so kann man ihn mit `count+=1` um `1` erhöhen.

```
>>> def fib(n):
...     a,b = 1,1
...     while ...
```

2. **Listen, Strings und Mengen**

   Eine Liste kann man in eckigen Klammern angeben:

```
>>> mylist = [1,2,3,2,4,1,2,5]
>>> mylist
[1, 2, 3, 2, 4, 1, 2, 5]
```

   Mit dem Operator + kann man Listen und Strings konkatenieren.

```
>>> l1 = [1, 2]
>>> l2 = [3, 4]
>>> l1+l2
[1, 2, 3, 4]
>>> s1 = 'abc'
>>> s2 = 'deff'
>>> s1+s2
'abcdeff'
```

   Die Elemente in Strings und Listen haben Indizes, über die auf sie zugegriffen werden kann:

```
>>> l1[0]
1
>>> s2[1]
'e'
```

   Mit `set()` macht man aus einer Liste eine Menge, doppelte Elemente werden eliminiert. Mit `list()` wird aus der Menge wiederum eine Liste.

```
>>> set(mylist)
set([1, 2, 3, 4, 5])
>>> list(set([1,2,3,2,4,1]))
[1, 2, 3, 4]
```

Listen, Strings und Mengen sind iterierbar. Sie können mit `for` durchlaufen werden.

```
>>> for x in mylist:
...     print x,
1 2 3 2 4 1 2 5
```

Die Gleichheit zweier Elemente wird mit `==` überprüft (im Gegensatz zu `=`, das für Wertzuweisungen verwendet wird):

```
>>> list1=[1,2,3,2,1,3,4]
>>> list2=[1,2,3,4,2]
>>> list1==list2
False
>>> set(list1)==set(list2)
True
```

Binäre Operationen auf Mengen:

```
>>> set1 = set([1, 2, 3, 4])
>>> set2 = set([1, 3, 4, 6])
>>> set1.difference(set2)
set([2])
>>> set2.difference(set1)
set([6])
>>> set1.intersection(set2)
set([1, 3, 4])
>>> set1.union(set2)
set([1, 2, 3, 4, 6])
```

Es gibt auch abkürzende Symbole für diese Operationen:

```
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a                                  # unique letters in a
set(['a', 'r', 'b', 'c', 'd'])
>>> a - b                              # letters in a but not in b
set(['r', 'd', 'b'])
>>> a | b                              # letters in either a or b
set(['a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'])
>>> a & b                              # letters in both a and b
set(['a', 'c'])
>>> a ^ b                              # letters in a or b but not both
set(['r', 'd', 'b', 'm', 'z', 'l'])
```

3. **Module in separaten Dateien**

   Erzeugen Sie im Verzeichnis `X/PC-Pools/Transfer/Python/EinfCL` unter einem eindeutigen Namen ⟨myName⟩.py Ihrer Wahl eine separate Datei, in die Sie folgende Definition schreiben:

```
# liefert die Menge der 2-fachen der Elementes
# des Schnitts von set1 und set2
def double_intersect(set1,set2):
     result = set([])
     for x in set1:
         if x in set2:
             result = result.union(set([2 * x]))
     return result
```

Hier gibt es zwei neue Dinge: Kommentare werden durch ein vorhergehendes `#` gekennzeichnet. Und wir sehen hier eine weitere Kontrollstruktur, die `for`-Schleife. Eine solche Schleife durchläuft alle Elemente einer iterierbaren Datenstruktur. Diese kann z.B. eine Menge, eine Liste oder ein String sein. Die Syntax ist

```
for Variable in Struktur:
   Block
```

Die Wirkung ist, dass für alle Elemente aus unserem iterierbaren Element `Struktur` der `Block` durchlaufen wird, wobei in jedem Durchlauf die Variable `Variable` gerade das jeweilige Element als Wert zugewiesen bekommt.

Sie können Ihr Modul ⟨myName⟩ im Interpreter wie folgt laden:

```
>>> import <myName>
```

Danach können die Funktionen aus diesem Modul benutzt werden:

```
>>> s1 = set([1,2,3,4,2])
>>> s1
set([1, 2, 3, 4])
>>> s2=set([1,3,5])
>>> s2
set([1, 3, 5])
>>> <myName>.double_intersect(s1,s2)
set([2, 6])
```

Aufgabe: Definieren Sie in Ihrem Modul eine Variante der obigen Definition, die den Schnittoperator `&` und den Vereinigungsoperator `|` verwendet.

Schreiben Sie die Definitionen in Ihre Datei, und laden Sie das Modul anschließend erneut mit

```
>>> reload(<myName>)
<module '<myName>' from '<myName>.py'>
```

Jetzt müssten Sie Ihre Funktionen benutzen können.

Aufgabe: Definieren Sie außerdem eine Funktion `extract_between(set,n,m)`, die aus einer Menge set1 von natürlichen Zahlen alle Zahlen heraussucht, die zwischen n und m liegen.

Nach `reload(<myName>)` könnne Sie diese Funktion benutzen:

```
>>> <myName>.extract_between(s1,2,5)
set([2, 3, 4])
```

Aufgabe: Definieren Sie als letzes noch eine Funktion, die einen englischen String als Argument nimmt und die Anzahl Vokale in diesem String zurückgibt. Auch für Strings und ihre Zeichen kann man die Relation `in` verwenden, man kann also hier eine Schleife über alle `x in string` durchlaufen, wobei `string` die Variable für den Eingabestring ist.

4. Zum Schluss verlassen Sie den python Interpreter mit `exit()`.