Python für Linguisten

Dozentin: Wiebke Petersen & Co-Dozent: Valentin Heinz

4. Foliensatz
bedingte Anweisungen
Input, Output

Vergleichsoperatoren und Boolsche Werte

- numerische Vergleichsoperatoren: `1 < 2, 2 > 1, 1 == 1, 1 != 2`
  - kleiner als, Beispiel: `1 < 2`
  - größer als, Beispiel: `2 > 1`
  - gleich, Beispiel: `1 == 1`
  - ungleich, Beispiel: `1 != 2`
  - kleiner-gleich, Beispiel `1 <= 2`
  - größer-gleich, Beispiel `1 >= 2`
- Strings vergleichen: `'this_string'== 't_string'`
  `'this_string'!= 't_string'`
- Typen prüfen: `type(this_var) == str` besser:
  `isinstance(this_var, str)`
- Eine Vergleichsoperation hat als Ergebnis einen boolschen Wert vom Datentype `bool`.
- `True` bedeutet logisch wahr. Beispiel: 2+3==5
- `False` bedeutet logisch falsch. Beispiel: 2<1

bedingte Anweisung: if

- >>> `if` prüft ob eine Bedingung zum boolschen Wert `True` evaluiert wird.
- `if` hat folgende Syntax:

```
if bedingung:
    # Beginn des eingrückten Blocks
    # notwendig: enthält ausführbaren Code
    # wenn nichts getan werden soll, verwende: pass
    pass
```

- Der Code in dem if-Block wird nur ausgeführt, wenn die Bedingung zu `True` evaluiert.

```
if number > 2:
    print(number,"größer 2")
if number < 2:
    print(number,"kleiner 2")
if number == 2:
    print(number,"gleich 2")
if number != 2:
    print(number,"ungleich 2")
```

Bedingte Anweisung ‘if’ mit Default ‘else’

- `if` wird meistens zusammen mit `else` verwendet
- `else` ist die Defaultanweisung. Sie wird ausgeführt, falls die `if`-Bedingung nicht zutrifft

```
if number > 2:
    print(number,"größer 2")
else:
    print(number,"kleiner gleich 2")
```

Syntactic Sugar: elif

- Wenn man mehrere `if/else`-Konstruktionen verschachteln möchte, kann man kürzer `elif` verwenden.
- `elif` ist eine nachfolgende Bedingung, die nur geprüft wird, falls `if` nicht zutrifft. `elif` kann mehrmals verwendet werden

```
if number > 2:
    print(number,"größer 2")
else:
    if number < 2:
        print(number,"kleiner 2")
    else:
        print(number,"gleich 2")
```

```
if number > 2:
    print(number,"größer 2")
elif number < 2:
    print(number,"kleiner 2")
else:
    print(number,"gleich 2")
```

Unterschied zwischen if und elif

- Was ist der Unterschied zwischen:

```
if number == 2:
    print(number,"==2")
if number >= 2:
    print(number,">=2")
```

```
if number == 2:
    print(number,"==2")
elif number >= 2:
    print(number,">=2")
```

Hausaufgabe: a- und o-Deklination im Lateinischen

Ziel ist die Entwicklung eines Wortformengenerators für das Lateinische (Aufgabe 3). Die ersten beiden Aufgaben sollen Ihnen bei der Programmierung helfen.

1. Definieren Sie eine Funktion, die zwei Argumente nimmt: 1. den Wortstamm eines lateinischen Nomens, 2. dessen Deklinationsklasse. Die Funktion soll die deklinierten Wortformen auf dem Bildschirm ausgeben.
2. Passen Sie Ihre Funktionsdefinition so an, dass die Stelligkeit der Funktion erhalten bleibt und die Funktion als 1. Argument die Wortform im Nominativ Singular nimmt. Das 2. Argument bleibt unverändert.
3. Passen Sie Ihre Funktionsdefinition so an, dass die Funktion lediglich ein Argument nämlich die Wortform im Nominativ Singular hat und die Deklinationsklasse selbst ermittelt.

Können Sie Ihre Definition so erweitern, dass bei falschem Funktionsaufruf eine entsprechende Fehlermeldung ausgegeben wird?

Hausaufgabe: a- und o-Deklination im Lateinischen, Beispiel In- und Output

Aufgabe 3: Beispielaufruf:

```
>>>print_wordforms(domina)
```

Singular:

Nom	domina
Gen	dominae
Dat	dominae
Acc	dominam
Abl	domina

Plural:

Nom	dominae
Gen	dominarum
Dat	dominis
Acc	dominas
Abl	dominis

```
>>>print_wordforms(dominus)
```

Singular:

Nom	dominus
Gen	domini
Dat	domino
Acc	dominum
Abl	domino

Plural:

Nom	domini
Gen	dominorum
Dat	dominis
Acc	dominos
Abl	dominis

for-Schleifen

- Syntax:

```
for element in datatype:
    print element
```

- führt den Schleifenkörper für alle Elemente in `datatype` aus
- das jeweilige Element ist als Variable `element` im Schleifenkörper verfügbar
- wie immer muss der Schleifenkörper eine Ebene (4 Leerzeichen) eingerückt werden
- dies funktioniert mit Listen, Strings und anderen Datentypen, die wir noch behandeln werden

range

- Syntax:

```
range(von [optional], bis, schrittweite [optional])
```

- Testen Sie die Funktion `range`. Was sind die Argumente, was das Ergebnis?
- `range` wird insbesondere in for-Schleifen eingesetzt:

```
print("Die 13er-Reihe ist:"
for number in range(0,131,13):
    print number
```

- Schreiben Sie eine Funktion, die für beliebige Integers die entsprechende Multiplikationsreihe ausgibt. Beispiel:

```
>>>reihe(4)
0  mal 4 ist 0
1  mal 4 ist 4
2  mal 4 ist 8
...
10 mal 4 ist 40
```

for-Schleifen: break und continue

- Syntax:

```
datatype = ['kiwi', 'apple', 'salad', 'cherry', 'pineapple']
for element in datatype:
    if element == 'salad':
        continue
    print element
```

- `continue` wird benutzt, um die aktuelle Iteration zu beenden
- `break` wird benutzt, um die Iteration und die Schleife zu beenden

```
for element in datatype:
    if element == 'apple':
        continue
    elif element == 'salad':
        break
    print element
```

while-Schleifen

- Syntax:

```
while boole:
    statement
```

Beispiel:

```
var = 0
while var <= 10:
    print var
    var = var + 1
```

- kann mit `continue` und `break` benutzt werden
- läuft, solange die Bedingung wahr ist, oder `break` ausgeführt wird
- Noch ein Beispiel:

```
number = 2
while number <= 1000000:
    print number
    number = number ** 2
```

Interpreter: Input/Output

- Output bedeutet, Daten auszugeben. Dies kann direkt auf die Standardausgabe erfolgen:
- `print("Ich bin Output")`
- dies ist sinnvoll, für Informationen die den **Nutzer** des Programms direkt (besonders während der Nutzung) betreffen
- Input sind Daten, die mittels des Programms verarbeitet werden sollen und eingegeben werden
- dies kann interaktiv durch die Standardeingabe geschehen
- `this_input = raw_input()`
- Mit Aufforderung:
  `this_input = raw_input("Bitte Daten eingeben:\n")`

Dateien öffnen - Datei-Handle erstellen

- Daten können auch in eine Datei geschrieben werden, etwa um das Ergebnis dauerhaft zu speichern oder Loggingdaten zu erheben
- hierzu muss ein Datei-Handle geöffnet werden:
  `fh = open('dateiname', 'w')`
- `fh` ist eine Konvention nach der man den Datei-Handle benennt (file handle)
- `dateiname` ist der Name einer existierenden oder zu erstellenden Datei
- das zweite Argument, in diesem Fall `w` gibt an, mit welchen Rechten die Datei geöffnet wird; hier eine Auswahl:
- `w` zum schreiben öffnen
- `r` zum lesen öffnen (default)
- `a` nur zusätzliche Daten schreiben

Dateien verarbeiten - Dateihandler nutzen

- wurde ein Handler mit Schreibberechtigung erstellt, kann in die Datei geschrieben werden:
- `fh.write('Ich bin ein String der in dateiname geschrieben wird')`
- wurde ein Handler mit Leseberechtigungen erstellt, kann die Datei eingelesen werden:
- `fh = open('mein_corpus', 'r')`
- `fh.read()` liest den gesamten Inhalt ein
- `fh.readlines()` liest den Inhalt zweilenweise ein und erstellt eine Liste mit Zeilen als Elemente
- die beiden Argumente können auch als `rw` kombiniert werden
- ist man mit der Nutzung der Datei fertig, sollte man die Datei schließen: `fh.close()`

Daten nutzen

- Die eingelesenen Daten sind nun verfügbar und können durch die Variable verwendet werden, in die sie geschrieben wurden:

```
# Schreiben:
fh = open('my_file', 'w')
fh.write('this is input')
fh.close()

# Lesen:
fh = open('my_file', 'r')
input_data = fh.read()
fh.close()

#Ausgeben:
for token in input_data:
    print token
```

Aufgaben: einfacher

- Schreiben Sie eine Funktion, die alle Vokale in einem Wort zählt.
- Schreiben Sie ein Script, das folgendes Spiel implementiert: Eine Zufallszahl zwischen 0 und 40 wird erzeugt (`random(40)`), die der Spieler raten soll. Er erhält Hinweise wie: "Zahl zu hoch", "Zahl zu niedrig'.
- Schreiben Sie ein Script, das einen Text aus einer Datei einliest und ihn Wort für Wort ausgibt.
- Öffnen Sie eine Datei zum Schreiben mit einem Namen Ihrer Wahl (Benennungskonvention und Dateiendung beachten), schreiben Sie einen String in die Datei und schließen Sie die Datei
- Öffnen Sie die gerade erstellte Datei erneut. Schreiben Sie einen anderen String in die Datei und schließen Sie die Datei. Öffnen Sie die Datei erneut und lesen Sie sie aus. Was passierte?

Aufgaben: schwieriger

- Schreiben Sie eine Funktion, die eine gegebene Gleitkommazahl in ihre binäre Repräsentation umwandelt.
- Schreiben Sie eine Funktion, die alle Vokale in einem Text aus einer Datei zählt.