IWGS1_WS1920_Zettel6

July 7, 2020

1 Informatische Werkzeuge in den Geistes- und Sozialwissenschaften I

1.1 Hausaufgabe 6 (Unicode, 100 Punkte)

Erschienen: 23.11.2019
Abgabe bis: 01.12.2019

Bitte laden Sie Ihre Notebooks bis 23:59 Uhr am Abgabetag in Ihrer Übungsgruppe bei StudOn hoch.

Wenn Ihnen einige der hier verwendeten Konzepte unbekannt sind oder Sie nicht wissen, wie Sie fortfahren sollen, können Sie die Vorlesungsunterlagen zu Rate ziehen oder jederzeit Fragen im Forum stellen, im Tutorium nachfragen, oder Ihrem Tutor eine Mail schreiben.

1.1.1 Aufgabe 6.1 (Unicode Umwandlung, 10 Punkte)

Je mehr Sie programmieren, desto mehr werden Sie bemerken, wie nützlich es ist, einzelne Code-Abschnitte in Funktionen zu kapseln, die Sie wieder und wieder verwenden können, idealerweise auch in Umständen, die Sie ursprünglich gar nicht bedacht hatten. Diese Aufgabe ist eine weitere "Fingerübung", um mit Funktionen bekannter zu werden.

Das Szenario ist, dass Sie frisch von Unicode erfahren haben, aber bemerken, dass Sie nicht in allen Situationen auf die `"\Uxxxxxxxx`-Notation zurück fallen können oder wollen, zum Beispiel bei User Input. Was Sie gebrauchen könnten ist eine Funktion, die sowohl mit Strings als auch mit ganzen Zahlen umgehen kann und einen Unicode-Charakter daraus machen kann. Die Funktion `chr` kann dies für Zahlen bereits, aber kann nicht mit Strings umgehen.

[3]:
```
# 1F601 (Hexadezimal) == 128513 (Dezimal)

print(chr(128513))   # Funktioniert!
# print(chr("1F601")) # Funktioniert nicht!
```

Sie sollten sich aus vergangenen Übungen bereits darüber im Klaren sein, dass Sie mit `int()` Strings in Zahlen umwandeln können. Normalerweise wird dabei von Python die Basis 10 angenommen. Wenn Sie allerdings eine andere Basis (z.B. 16) benutzen wollen, können Sie diese als zweites Argument übergeben, wie in folgendem Beispiel:

[2]:
```
print(chr(128513))
print(chr(int("1F601",16)))
```

Für die Unterscheidung zwischen ganzen Zahlen (= Integers) und Strings im Rahmen dieser Aufgabe stehen Ihnen diese vorgegebenen Funktionen zur Verfügung, die angeben, ob ihr Argument dem entsprechenden Typ angehört.

[3]:
```
# Gibt "True" zurück, wenn val eine ganze Zahl ist, sonst "False".
def isInt(val):
    return str(type(val)) == "<class 'int'>"

# Gibt "True" zurück, wenn val ein String ist, sonst "False".
def isStr(val):
    return str(type(val)) == "<class 'str'>"
```

Aufgabe: Schreiben Sie in der nächsten Zeile eine Python-Funktion mit dem Namen `unicode`, die *entweder* eine Zahl *oder* einen String als Parameter nimmt und einen String zurück gibt. Was auch immer übergeben wird, soll entsprechend behandelt werden (Sie können dafür die Funktionen `isInt()` und `isStr()` oben verwenden). *Integers* (= ganze Zahlen) sollen direkt mit `chr` in Unicode umgewandelt werden. Strings sollen erst in eine ganze Zahl (immer Basis 16) konvertiert und *dann* mit `chr` in Unicode umgewandelt werden. Wenn weder ein String noch ein Integer übergeben wurde, soll eine Fehlermeldung ausgegeben werden. Sie können davon ausgehen, dass alle Strings, die eingegeben werden tatsächlich Zahlen sind, Eingaben wie `unicode("falsch")` müssen Sie *nicht* abfangen.

Testen Sie Ihren Code!

[4]:
```
# Ihr Code hier! (Doppelklick zum Editieren, Shift + Enter um die Zelle␣
↪auszuführen)
```

1.1.2 Aufgabe 6.2 (Unicode Hieroglyphen, 40 Punkte)

Nachdem Beatrice Beispiel überall davon rumerzählt hat, wie talentiert und hilfsbereit Sie sind, kommen jetzt auch häufiger Kolleg*innen aus anderen Fachbereichen auf Sie zu. So auch heute der Ihnen höchstens flüchtig bekannte Ädgar Ägyptologe. Dieser erzählt Ihnen, dass er für seine Arbeit häufig Unicodezeichen für ägyptische Hieroglyphen benötigt. Bisher hat er jeweils aufwändig nach den einzelnen Zeichen gegoogelt und sie per Hand kopiert und in seinen Text eingefügt. Diese Vorgehensweise nimmt allerdings gerade bei längeren Texten viel Zeit in Anspruch. Und das kann fatal sein, wie er Ihnen versucht mit dem ZDF-Sketch "*Ente, Auge, Zickzack*" zu verdeutlichen.

Er stellt Ihnen ein Dictionary mit den wichtigsten Hieroglyphen bereits zur Verfügung, fragt Sie allerdings danach, ob Sie ihm ein Programm schreiben können, mit dem die Eingabe von längeren Hieroglyphentexten schneller passieren kann als mit der bisherigen Google/Copy/Paste-Methode.

[5]:
```
# Ein Dictionary mit den relevanten Hieroglyphen.
# Dieses ist (sobald diese Zelle ausgeführt wurde) auch in nachfolgenden Zellen␣
↪verfügbar.
```

```
# Wenn sich Ihr Code über "hieroglyphen is not defined" beschwert, müssen Sie␣
 ↪diese Zelle nochmal ausführen.

hieroglyphen = {
    "ente"     : "\U00013170",
    "auge"     : "\U00013079",
    "zickzack" : "\U00013216",
    "schlange" : "\U00013199",
    "stern"    : "\U000131FC",
    "eule"     : "\U00013153",
    "ankh"     : "\U00002625",
    "spirale"  : "\U00013362" # und so weiter und so fort...
}
```

Potentiell hilfreich, wenn auch nicht unbedingt notwendig dafür (allerdings auf jeden Fall gut zu wissen) sind die Methoden `d.keys()` und/oder `d.values()`, die für ein Dictionary `d` entsprechend entweder alle Schlüssel oder alle Werte als zurück geben. Um sie als Liste zu erhalten, die leicht weiter benutzt werden kann, müssen Sie auch die `list()`-Funktion verwenden, wie unten gezeigt.

[6]:
```
print(list(hieroglyphen.keys()))
print(list(hieroglyphen.values()))
```

```
['ankh', 'auge', 'stern', 'zickzack', 'ente', 'spirale', 'eule', 'schlange']
[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
```

Hinweis: Wenn Sie im Output der letzten Zelle nicht die entsprechenden Hieroglyphen (sondern z.B. nur leere Quadrate) sehen, liegt das wahrscheinlich daran, dass Ihr Rechner und/oder Ihr Browser keine Schriftart benutzt, die die Symbole darstellen kann. Wir wissen, dass das z.B. unter Windows 8.1 mit Chrome auftreten kann. Wenn dieses Problem bei Ihnen auftritt und hartnäckig bleibt, kontaktieren Sie bitte Ihren Tutor.

Aufgabe: Schreiben Sie ein Python-Programm, dass (ähnlich zu Aufabe 4.3 und 5.2) den Menschen wieder und wieder nach Input fragt und dabei einen String in einer Variable aufbaut. Dieser String soll nur aus Hieroglyphen bestehen, aber der Input sollen die entsprechenden deutsche Worte sein (In anderen Worten: Ist der Input ein *Schlüssel* im obigen Dictionary, soll an den String der entsprechende *Wert* im Dictionary angehängt werden.). Ist der Input exakt `":ende"`, so soll die Schleife beendet werden und das finale Ergebnis ausgegeben werden. Ist der Input etwas anderes, so soll eine Fehlermeldung ausgegeben und weiter nachgefragt werden.

Testen Sie Ihren Code!

[7]:
```
# Ihr Code hier! (Doppelklick zum Editieren, Shift + Enter um die Zelle␣
 ↪auszuführen)
```

1.1.3 Aufgabe 6.3 (Programmerweiterung, 20 Punkte)

Dies ist eine Folgeaufgabe zu 6.2. Nachdem Sie Ihr Programm an Ädgar geschickt haben, erhalten Sie innerhalb einer Stunde eine neue E-Mail. Diese ist voll mit Lob und preist Sie in den höchsten

Tönen. Sie hätten quasi einhändig die Abteilung für Ägyptologie doppelt so effizient gemacht. Kurz hinter diesen süßen Worten folgt jedoch schon der erste Wunsch nach Änderungen. Es wäre hilfreich, wenn Sie außerdem möglich machen könnten, das Nutzer*innen den aktuellen Zustand des Strings betrachten und das letzte eingegebene Zeichen auch wieder löschen können.

Aufgabe: Erweitern Sie Ihren Code aus Aufgabe 6.2 um zwei weitere Eingabemöglichkeiten. Wird `":zeigen"` eingegeben, soll der aktuelle Stand des Strings, der aufgebaut wird, ausgegeben werden. Wird hingegen `":löschen"` eingegeben, so soll *das letzte Zeichen* des Strings entfernt werden. Alle anderen Funktionalitäten sollen gleich bleiben.

Testen Sie Ihren Code!

[8]:
```
# Ihr Code hier! (Doppelklick zum Editieren, Shift + Enter um die Zelle␣
↪auszuführen)
```

1.1.4 Aufgabe 6.3 (Verständnisaufgabe 5, 30 Punkte)

Abermals kommt Ihre gute Freundin, Beatrice Beispiel, auf Sie zu und fragt um Rat. Sie entschuldigt sich nicht dafür, Ihnen zusätzliche Arbeit mit den Ägyptologen gemacht zu haben, und fängt stattdessen sofort an, zu klagen. Sie hat angefangen, sich mit regulären Ausdrücken (regular expressions, oder kurz: RegEx) auseinander zu setzen und merkt aber, dass sie Startschwierigkeiten hat. Sie hat sich zu Testzwecken eine kleine (und korrekte!) Funktion `wieoft(r,k)` geschrieben, die Pythons `re`-Library dafür benutzt, herauszufinden, wie oft der RegEx `r` im Korpus `k` vorkommt. Allerdings stimmen die Werte so überhaupt nicht mit ihren Erwartungen überein. Sie bittet Sie darum, ebenfalls einen Blick drauf zu werfen, weil bald Abgabe ist und sie fürchtet, es nicht alleine zu schaffen.

Aufgabe: Machen Sie sich mit RegExen und dem Python-Code in der nächsten Zelle bekannt. Finden Sie heraus, welche Fehler Beatrice mit ihren RegExen unterlaufen sind. Welche Muster wollte Beatrice finden und welche hat sie tatsächlich gefunden? Schreiben Sie Ihre Erkenntnisse in der übernächsten Zelle auf und geben Sie die RegExe an, die Beatrice stattdessen hätte benutzen sollen.

[1]:
```
# Hier importiere ich die Regular Expression Library
import re

# Meine eigene kleine Funktion, die mir sagt,
# wie oft ein RegEx in einem Korpus gefunden wird.
def wieoft(regex, korpus):
    ergebnisse = re.findall(regex, korpus)
    return len(ergebnisse)

# Das sind meine RegExe, die ich entworfen habe.
# Ich hab notiert, was ich gemeint habe, aber alles ist irgendwie falsch.

r1 = "ecken$"    # Für alle Wörter, die auf "...ecken" enden.
r2 = "ess?el"    # Für sowohl "Esel", als auch "Nessel", also 1 oder 2 "s".
r3 = "(bla)*"    # Für Politikerreden, also beliebig viele "bla"s.
```

```
# Das ist der String, in dem wir suchen.
schnecken = "Wenn Schnecken an Schnecken schlecken, merken sie zu ihrem␣
 ↪Schrecken, dass Schnecken nicht schmecken."
esel = "Esel essen Nesseln nicht, Nesseln essen Esel nicht."
rede = "bla bla bla bla bla bla bla bla bla bla"

# Tests, mit erwarteten Ergebnissen.

# Erwartet: 6, Tatsächlich: 0?
print(wieoft(r1,schnecken))

# Erwartet: 4, Tatsächlich: 2?
print(wieoft(r2,esel))

# Erwartet: 10, Tatsächlich: 20?
print(wieoft(r3,rede))
```

```
0
2
20
```

Ihre Beschreibung und Erläuterungen hier! (Doppelklick zum Editieren, Shift + Enter um die Zelle auszuführen)