vl-02

October 11, 2024

[ ]:
```
%load_ext tutormagic
```

####

1 Programmierung für KI

1.0.1 Wintersemester 2024/25

Prof. Dr. Heiner Giefers

1.0.2 Aus dem Forum: Fehlende Pakete

In [1]:
```
from nose.tools import assert_equal
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[1], line 1
----> 1 from nose.tools import assert_equal

ModuleNotFoundError: No module named 'nose'
```

In [2]:
```
import sys
!{sys.executable} -m pip install nose
```

```
Collecting nose
  Downloading nose-1.3.7-py3-none-any.whl (154 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 154.7/154.7 kB 2.5 MB/s eta 0:00:00:00:01
Installing collected packages: nose
Successfully installed nose-1.3.7
```

In [3]:
```
from nose.tools import assert_equal
```

In [ ]:
```
```

1.0.3 Aus dem Forum: Anwendung von ‘try-except“

```
try:
    # Hier der Code, der einen Fehler hervorrufen kann
    # und der Code, der nur dann ausgeführt werden soll,
    # falls kein Fehler aufgetreten ist.
    # Sobald ein Fehler auftritt, wird dieser Block sofort
    # verlassen und zu einem (passenden) except Block
    # gesprungen
except [Fehlertyp]:
    # Hier der Code, der ausgeführt werden soll, sofern
    # ein Fehler aufgetreten ist. Wird ein Fehlertyp angegeben,
    # so wird dieser Block nur ausgeführt, falls ein Fehler
    # dieser Art aufgetreten ist.
```

1.1 Python Basics

1. Bezeichner und Ausdrücke

- Variablen
- Formatierung und Kommentare
- Ausdrücke
- Boolsche Ausdrücke

1. Kontrollfluss

- Bedingte und alternative Ausführung
- Schleifen (`while` und `for`)
- Funktionen

1. Weiteres

- Dateien, Zeichenketten, …

Variablen Namen

- In Python gibt es keine (klassischen) Variablen
- Objekte können mit einem *Namen* bezeichnet werden
- Objekte haben einen Typ

```
a = 12
b = 12.0
c = "Hallo Welt!"
d = "Hallo" + " Welt!"
```

[ ]:
```
%%tutor --lang python3 --cumulative --heapPrimitives
a = 12
b = 12.0
c = "Hallo Welt!"
d = "Hallo" + " Welt!"
```

1.2 Bezeichner

- Klein- oder Großbuchstaben, Ziffern, oder Unterstriche (_)
- Das erste Zeichen
  - darf keine Ziffer sein
  - sollte kein Unterstrich sein (Namen, die mit Unterstrich(en) beginnen, haben eine spezielle Bedeutung)
- Schlüsselwörter können nicht als Bezeichner gewählt werden
- Funktionen und viele Konstanten sind keine Schlüsselwörter
- Wenn man nicht aufpasst, kann man Namen überdecken (“umdefinieren”)

[ ]:
```
#and = 5
#and + and
xxx = print
xxx("Hello")
```

Es kann auch sinnvoll sein, Standard-Funktionen umzubenennen

[ ]:
```
def myinput(s):
    print(s)
    return 42

i_old = input
input = myinput
x = input("Gib eine Zahl ein: ")
print(x)
input = i_old
```

1.3 Die `print` Anweisung

[ ]:
```
laenge = 5
print("Die Kantenlänge ist", laenge, "cm")
print("Die Kantenlänge ist %d cm" % (laenge))
print(f"Die Kantenlänge ist {laenge} cm")
```

1.3.1 `print` ohne Zeilenumbuch

[ ]:
```
i=1
while(i<10):
    print(i, end=' ')
    i = i + 1
```

[ ]:
```
help(print)
```

1.4 Ausdrücke

- Alles, was sich zu einem Wert evaluieren lässt
- Konstanten, Variablen, Funktionsaufrufe
- Auch kombiniert mit Operatoren

```
4
a
a + 42
sum([1]) + 5
x = a + sum([1]) + 5 # Zuweisung
```

1.5 Boolesche Ausdrücke

- Lassen sich zu `True` oder `False` auswerten
- Werden für *Kontrollfluss* benötigt
- Vergleiche (`>`,`>=`,`<`,`<=`,`==`,`!=`)
- Verknüpfungen (`and`,`or`,`not`)

[ ]:
```
a = 1
x = a < 4
print(x == 1)
```

```
y = 0
print(x > 0 and y > 0)

li = ['a']
i = 1
print(i < len(li) \
      and li[i] == 'b')
```

2 Kontrollfluss

2.1 Bedingte Ausführung

- Die Überprüfung einer *Bedingung* entscheidet, ob ein **Block** von Anweisungen ausgeführt wird
- Python verwendet für Blöcke keine Klammern (so wie viele andere Programmiersprachen) sondern **Einrückungen**
- Achtung: Bitte Leerzeichen verwenden (und **keine** Tabulatoren)!

[ ]:
```
x = 1
doit = "yes"
if x>0 and doit=="yes":
    x = 0
    res = 42
print(x)
```

2.2 Alternativen

[ ]:
```
a = 1 ; b = 1
if(a<b):
    print("Kleiner")
else:
    print("Groesser oder Gleich")
```

2.3 Alternativen

[ ]:
```
a = 1 ; b = 1
if(a<b):
    print("Kleiner")
elif(a>b):
    print("Groesser")
else:
    print("Gleich")
```

2.4 Ab Python 3.10 gibt es ein `match-case`-Konstrukt

[ ]:
```
radius = breite = hoehe = 2
#form = "Kreis"
form = input("Gib eine geometrische Figur ein: ")
match form:
    case "Kreis":
        print("Die Fläche ist", 3.14159 * radius * radius)
    case "Rechteck":
        print("Die Fläche ist", breite * hoehe)
    case "Dreieck":
        print("Die Fläche ist", 0.5 * breite * hoehe)
    case _:
        print("Unbekannte Form!")
```

2.5 Schleifen

- Python kennt 2 Arten von Schleifen
- `while`Solange eine Bedingung gilt, führe einen bestimmten Anweisungsblock aus
- `for`Für alle Elemente eine Folge, führe einen bestimmten Anweisungsblock aus
- Schleifen können *geschachtelt* sein

[ ]:
```
potenz = 1
while(potenz<1000):
    print(potenz)
    potenz *= 2
```

[ ]:
```
liste = [1, 'Zwei', 3.0]

print("Liste: ", end='')
for elem in liste:
    print(elem, end=' ')

print("\n\nrange: ", end='')
for i in range(10):
    print(i, end=' ')
```

2.6 Funktionen

Die Anweisung def verbindet einen (ggf. neuen) Namen (für eine Funktion) mit einem Block (von Anweisungen)

- Der Name *referenziert* diesen Anweisungsblock
- Namen sind Bezeichner und folgen den bekannten Syntaxregeln
- Funktionen können in Funktionen definiert werden
- Funktionen können dynamisch erzeugt werden

2.7 Funktionsaufruf

- Funktionsnamen mit `()`
- Argumente übergeben
- Eine Funktion besitzt *Parameter*
- Beim Aufruf übergibt man *Argumente* an die Funktion
- Default Parameter
- Optionale Parameter

[ ]:
```
def print_params(a,b,c):
    print(a)
    print(b)
    print(c)

print(1,2,3)
```

2.8 Aufrufsemantik

- Die Aufrufsemantik von Python ist Pass-by-assignment
- Die Parameter zeigen auf dieselben Objekte wie die Argumente
- Ändert sich auch der Wert eines Arguments (ausserhalb der Funktion), wenn man den Parameter innerhalb der Funktion ändert?
- Antwort: Das liegt daran, ob der Typ des Arguments ein *veränderbarer* Typ oder ein *nicht-veränderbarer* Typ ist

[ ]:
```
def foo(arg):
    arg += 1
    print(arg)
```

[ ]:
```
x = 1
foo(x)
print(x)
```

[ ]:
```
%%tutor --lang python3 --cumulative --heapPrimitives
def foo(arg):
    arg += [3,4]
    print(arg)

x = [1,2]
foo(x)
print(x)
```

2.9 Namensräume

Name, an den innerhalb einer Funktion zugewiesen wird, ist lokal zu dieser Funktion - Existiert nur, solange Anweisungen dieser Funktion ausgeführt werden - Werte sind nach dem Ende der Funktion *verloren*

[ ]:
```
a = "globales a"
def f():
    a = "lokales a"
    print(a)

f()
print(a)
```

2.10 Rückgabewert

- Mechanismus, um einen Wert aus dem Funktionsaufruf an die aufrufenden Funktion zu übergeben
- Geht nicht mit normalem Namen für Werte (die sind ja nach der Funktion verloren)
- Wert mit `return` übergeben

[ ]:
```
def my_pow(a,b):
    return a**b

zwei_hoch_acht = my_pow(2,8)
zwei_hoch_acht
```

2.11 Zeichenketten und Dateien

- Die wichtigste Methode für Strings ist:

[ ]:
```
help(str)
```

- **str ist ein nicht-veränderbarer, sequentieller Datentyp**

[ ]:
```
s = "Hallo Welt!"
s[6]
```

[ ]:

- Textersetzung mit `replace`
- Umwandlung Groß-/Kleinschreibung mit `lower` und `upper`

[ ]:
```
'Übergröße'.lower().replace('ü','ue').\
    replace('ö','oe').replace('ß','ss').upper()
```

- Mehrere Ersetzungen gleichzeitig mit `translate` und `makerans`

[ ]:
```
de_table = str.maketrans({'Ä': 'Ae', 'Ö': 'Oe', 'Ü': 'Ue',
                          'ä': 'ae', 'ö': 'oe', 'ü': 'ue', 'ß': 'ss'})
'Übergröße'.translate(de_table)
```

2.12 Dateien

- Öffnen von Datein mit `open` im Modus *read* (`r`), *write* (`w`) oder *append* (`a`)
- Schließen mit `file.close()`
- `file.read()` ließt den gesamten Inhalt von `file`
- Zeilenweises Durchlaufen mit `for`
- Zugriff auf Dateien kann fehlschlagen. Daher immer `try-catch` oder *Context Manager* (`with`-Statement) verwenden

[ ]:
```
try:
    f = open('faust-utf8.txt','r')
    i = 1
    for line in f:
        print(f"{i:6d}: ",line,end='')
        if i==50:
            break
        i += 1
    f.close()
except:
    print('Datei konnte nicht geoeffnet werden')
```

[ ]:
```
f = open('faust-utf8.txt','r')
f_neu = open("absatz.txt", 'w')
i = 1
for line in f:
    if i>=1359 and i<=1361:
        f_neu.write(line)
    i += 1
    if i>1361: break;
f.close()
```

```
f_neu.close()
```

[ ]:
```
f = open('absatz.txt','r')
for line in f:
    print(line.replace('ü','ue'), end='')
f.close()
```

2.13 Zum 3. Termin

- Listen, Tupel und Dictionaries
- Wenn wir dort sind, können wir viele sinnvolle Programme schreiben:-)