Universität Potsdam
Institut für Informatik

GdP-Rechnerübung

Aufgabenblatt 6
(zuletzt aktualisiert: 14. November 2022)

Lernziele (zum Abhaken): Nach diesem Aufgabenblatt sollten Sie...

- ☐ mit dem Python-Interpreter umgehen können
- ☐ Variablen anlegen, auslesen und ausgeben können
- ☐ grundlegende Datentypen in Python kennen (Zahlentypen, Strings, Listen)
- ☐ grundlegende Operationen auf Zahlentypen kennen (`+`, `-`, `*`, `/`, `//`, `**`, `%`)
- ☐ grundlegende Operationen & Funktionen auf Sequenzen kennen (`+`, `*`, `len()`, `[]`, `[i:j]`)

10 Der Python-Interpreter

Hinweis: Wenn Sie mit einem privaten Rechner arbeiten, achten Sie bitte unbedingt darauf, Python 3.7 zu installieren (nicht 2.7)!
Verwenden Sie außerdem bitte keine Entwicklungstools (z.B. Eclipse), sondern arbeiten Sie wie in den Aufgaben gewünscht mit dem Terminal!

1. Arbeiten Sie mit dem Terminal!

   Sie haben bereits das Terminal kennen gelernt und können damit umgehen.

   a) Starten Sie den Python-Interpreter, der standardmäßig vom System aufgerufen wird, wenn Sie den Befehl `python` ausführen. Um welche Version von Python handelt es sich?

   Python 2.7.16

   Hinweis: Verlassen Sie den Python-Interpreter mit `STRG + D`!

   b) Um welche Version handelt es sich, wenn Sie stattdessen den Befehl `python3.7` ausführen?

   Python 3.7.3

   c) Erstellen Sie einen Alias, sodass bei der Eingabe von `python` der Python-Interpreter der Version 3.7 gestartet wird.

   alias 'python=python3.7'

   d) Schreiben Sie diesen Alias in die Dateien `~/.bashrc` und `~/.bash_profile`, sodass zukünftig immer `python3.7` gestartet wird.

e) Öffnen Sie eine neue Shell und überprüfen Sie das neue Verhalten.

Hinweis: Sollte weiterhin Python der Version 2.7 gestartet werden, melden Sie sich bitte bei Ihrem Tutor!

2. Der Python-Interpreter lässt sich ähnlich wie eine UNIX-Shell bedienen: Er liest Python-Anweisungen (zeilenweise) ein und führt sie aus. Der Befehl `python` (ohne Argumente) startet den interaktiven Modus.

   Hinweis: Wenn in Aufgabenstellungen Zeilen mit `>>>` beginnen, ist damit gemeint, dass Sie die Zeile in den Python-Interpreter eingeben sollen. Zeilen, die mit einem **$** beginnen, sollen direkt in das Bash-Terminal eingegeben werden.

   a) Starten Sie nun den Python-Interpreter mit dem Befehl `python` in der Konsole und probieren Sie die folgenden Befehle aus!

   ```
   >>> 4*5
   >>> 64 / 16.0
   >>> 64 / 15
   >>> 64 // 15
   >>> 64.0 // 15
   >>> 7 % 3
   >>> 7 * 5-9
   >>> 7 * (5-9)
   >>> -42
   >>> --42
   >>> ---42
   ```

   b) *Variablen* dienen zum Merken von *Werten*, z. B. von Berechnungsergebnissen. Einer Variablen kann mit `<Variablenname> = <Wert>` ein Wert *zugewiesen* werden. Probieren sie die folgenden Befehle aus! In dieser Aufgabe ist ein absichtlicher Fehler eingebaut. Können Sie sich diesen erklären?

   ```
   >>> ganze_zahl = 4 + 2
   >>> float_zahl = 7.3
   >>> string1, string2 = "It's done!", 'Woohoo!'
   >>> ganze_zahl + float_zahl
   >>> neu = ganze_zahl * string1
   >>> neu
   >>> print(neu)
   >>> print(string2 * float_zahl)
   ```

   Was ist beim letzten `print`-Befehl passiert?

   can't multiply sequence by non-int of type 'float'

   Interpretieren Sie die Fehlermeldung!

   Es kann eine sequenze von nicht int type Vaiablen multipliziert werden

c) Schließen Sie nun den Interpreter und starten Sie ihn erneut! Führen Sie den Befehl `print(neu)` erneut aus! Was stellen Sie fest?

die Variable neu ist nicht mehr deffiniert

11 Variablen und Datentypen

Jeder Wert, der in einer Variablen gespeichert ist, kann einem konkreten *Datentyp* zugeordnet werden.

1. Welche verschiedenen Datentypen kennt Python? Finden Sie es mit der `type`-Funktion heraus!

```
>>> type(4)
```
Ausgabe: int

```
>>> type(4.2)
```
Ausgabe: float

```
>>> type("Hallo Welt!")
```
Ausgabe: str

```
>>> type([1, 2, 3])
```
Ausgabe: list

```
>>> type(True)
```
Ausgabe: bool

2. Arbeiten Sie zunächst mit **Zahlentypen**.

   a) Was ist Ihrer Meinung nach die Ausgabe dieser Codezeilen? Überlegen Sie zunächst und probieren Sie es anschließend aus!

```
>>> var1 = float(5)
>>> var2 = var1 + 2
>>> var1 = 6.0
>>> print(var1)            Ausgabe: 6.0
>>> print(var2)            Ausgabe: 7.0
>>> print(var1 + var2)     Ausgabe: 13.0
>>> var1 = int(var1)
>>> print(hex(var1))       Ausgabe: 0x6
>>> var3 = 0o21
>>> print(var3)            Ausgabe: 17
>>> print(bin(var3))       Ausgabe: 0b10001
>>> var2 = "Hallo"
>>> print(var1, var2)      Ausgabe: 6 Hallo
```

   b) Wozu dient die Funktion...

      i. `float()`? cast var to float

      ii. `int()`? cast var to int

      iii. `hex()`? cast var to hex hex representaion of var

      iv. `bin()`? cast var to bin binary representaion of var

      Falls Sie unsicher sind, überprüfen Sie Ihre Annahmen noch einmal im Python-Interpreter!

   c) Arbeiten Sie mit arithmetischen Operationen auf Zahlen! Betrachten Sie folgende Zuweisung:

```
>>> zahl =  1 + 2 - 3 * 4 / 5.0
```

      i. Welchen Wert hat die Variable `zahl` nach der Zuweisung?

      0,55

      Überprüfen Sie Ihre Annahme anschließend im Interpreter!

      0.6000000000000001

ii. In welcher Reihenfolge arbeitet Python die Operatoren ab?

A. *Hinweis:* Operatoren können linksassoziativ oder rechtsassoziativ sein. Das nachfolgende Beispiel zeigt den Unterschied:

```
>>> (5 - 2) - 1 # linksassoziative Klammerung
```

Ergebnis: 2

```
>>> 5 - (2 - 1) # rechtsassoziative Klammerung
```

Ergebnis: 4

B. Arbeitet der Operator - in Python nun links- oder rechtsassoziativ?

```
>>> 5 - 2 - 1   # Pythons Arbeitsweise
```

2 linksassoziativ

iii. Python unterstützt auch Kurzschreibweisen:

```
>>> y=4
>>> y//=3
>>> y*=6
>>> y-=2
>>> print(y)
```

Welche Ausgabe erwarten Sie? Überprüfen Sie!

Erwartete Ausgabe: 4 -> 1 -> 6 -> 4

Tatsächliche Ausgabe: 4

d) Der in der Vorlesung besprochene Modulo-Operator wird in Python durch % repräsentiert.

i. Mit welchem Wert ist die Variable `rest` jetzt belegt?

```
>>> rest = 11 % 3
```

2

ii. Schreiben Sie alle möglichen Reste y für $x \bmod 3 = y$ auf.

0, 1, 2

e) Noch ein Operator ist `**` (das doppelte Mal-Zeichen). Was bewirkt er? Probieren Sie die folgenden Zeilen aus und geben Sie eine kurze Antwort.

```
>>> square = 10 ** 2
>>> cube = 2 ** 3
>>> KiByte = 2 ** 10
```

x^y = x ** y

3. **Strings** sind Sequenzen (Folgen) von Zeichen. Strings werden von einfachen (`'`), doppelten (`"`) oder dreifachen doppelten Anführungszeichen (`"""`) umgeben.

   a) Probieren Sie das Arbeiten mit Strings aus:

```
>>> mystring = 'hello'
>>> print(mystring)
>>> RNB = "R 'n' B steht für Rhythm and Blues"
>>> print(RNB)
>>> helloworld = mystring + " " + "world"
>>> print(helloworld)
>>> gruss = """Hallo,
... wie geht's?"""
>>> print(gruss)
```

   b) Definieren Sie zwei Variablen, denen Sie Ihren Vor- bzw. Nachnamen als String zuweisen. Erzeugen Sie dann eine Ausgabe der Form *Nachname, Vorname* (z. B. `"Muster, Max"`). Geben Sie die Befehle dafür an:

```
>>> Nachname = "neunert"
>>> Vorname = "Joshua"
>>> print(Nachname + ",", Vorname)
```

   c) Wie Sie bereits in Aufgabe 2b) gesehen haben, kann Python auch den `*`-Operator in Kombination mit Strings benutzen. Probieren Sie aus!

```
>>> nonsens = "bla" * 5
```

   Wert von `nonsens`: blablablablabla

   d) Was bewirkt die Funktion `len()`?

```
>>> astring = "hallihallo !"
>>> print(len(astring))
```

   12

4. **Listen** sind spezielle Variablen, in denen mehrere Werte in einer festen Reihenfolge gespeichert werden können. Arbeiten Sie mit **Listen**:

   a) Welchen Wert besitzt `mylist`?

   ```
   >>> mylist = [7, 1, 5]
   >>> mylist.append(3)
   >>> print(mylist)
   >>> mylist.sort()
   >>> print(mylist)
   ```

   b) Welche Ausgabe erwarten Sie von folgendem Code? Notieren Sie!

   ```
   >>> even_numbers = [2,4,6,8]
   >>> odd_numbers = [1,3,5,7]
   >>> all_numbers = odd_numbers + even_numbers
   ```

   c) Welchen Wert hat `all_numbers` tatsächlich? Prüfen Sie es nach!

   d) Auf Listenelemente kann mit dem Zugriffsoperator `[ ]` zugegriffen werden. In den eckigen Klammern gibt man dann die Position des Elements in der Liste (beginnend bei 0) an. So steht zum Beispiel `mylist[0]` für das nullte Element gleich am Anfang der Liste. Probieren Sie anhand des folgenden Beispiels Zugriffe auf Listenelemente aus!

   ```
   >>> mylist = ["eins", 2, [3, 4], "fuenf"]
   >>> print(mylist[2])
   >>> mylist[0] = 1
   >>> print(mylist)
   ```

      i. Welchen Wert erwarten Sie als Ausgabe? _____

      ii. Prüfen Sie nach! Welcher Wert wird ausgegeben? _____

      iii. Von welchem Datentyp ist der ausgegebene Wert? _____

      iv. Überprüfen Sie! Ist `"eins" in mylist`? _____

      v. Wozu dient also der Operator `in`?

e) Sequenzen (wie Listen oder Strings) können mit dem Slice-Operator `[i:j]` zerteilt werden. Dabei werden die Elemente mit Index `i` bis `j-1` extrahiert (d.h. die untere Grenze `i` ist inklusiv und die obere Grenze `j` ist exklusiv).

Arbeiten Sie mit den Slice-Operationen. Welche Werte entstehen jeweils?

```
>>> S = "Slicing funktioniert auf Listen und Strings"
```

`>>> S[25:]` Ergebnis: _____

`>>> S[:-12]` Ergebnis: _____

`>>> S[8:20]` Ergebnis: _____

`>>> S[:]` Ergebnis: _____

`>>> S[::2]` Ergebnis: _____

`>>> S[19:6:-1]` Ergebnis: _____

Weiterführende nützliche Links

- Die offizielle Python-Dokumentation
  `http://docs.python.org/`
- Deutsches Python-Tutorial
  `https://py-tutorial-de.readthedocs.io/de/python-3.3/`
- Interaktives englischsprachiges Tutorial
  `https://runestone.academy/ns/books/published/thinkcspy/index.html`
- Wikibooks: Python Programming (engl.)
  `https://en.wikibooks.org/wiki/Python_Programming`