Übung 1 - Einstieg

Programmiervorkurs 2025

1 Python Installieren

Um die Aufgaben dieses Programmiervorkurses bearbeiten zu können, benötigt man eine (halbwegs aktuelle) Python-Installation.
Als Code-Editor ist jeder übliche Texteditor geeignet, es benötigt keine Entwicklungsumgebung (IDE), von welchen wir im Rahmen dieses Vorkurses eher abraten, um einen größeren Lerneffekt zu erzielen.
Im Moodle haben wir dafür die Anleitung "Python Installieren für Dummies" hinterlegt.

Auf den Poolrechnern ist Python bereits installiert, dort muss der Befehl **python3** verwendet werden.

2 Theoriefragen

Bitte ordne folgende Aussagen als wahr oder falsch ein:

wahr	falsch	Frage
☐	☐	In Python ergibt der Ausdruck `5 / 2` den Wert `2.5`.
☐	☐	`5y` ist ein valider Variablenname.
☐	☐	`"2" + 5` wird ohne Fehler ausgeführt.
☐	☐	`float(int(str(int("3"))))` ist ein valider Ausdruck.
☐	☐	Es macht einen Unterschied, ob ich Tabs oder Spaces für die Einrückung verwende.

wahr	*falsch*	*Frage*
☒	☐	*In Python ergibt der Ausdruck `5 / 2` den Wert `2.5`.*
☐	☒	*`5y` ist ein valider Variablenname.*
☐	☒	*`"2" + 5` wird ohne Fehler ausgeführt.*
☒	☐	*`float(int(str(int("3"))))` ist ein valider Ausdruck.*
☒	☐	*Es macht einen Unterschied, ob ich Tabs oder Spaces für die Einrückung verwende.*

Bei Einrückungen ist es an sich egal, ob man in einer Datei nur Tabs oder Spaces verwendet. Allerdings darf man diese nicht mischen, was insbesondere dann relevant ist, wenn man Code zusammen mit anderen Leuten schreibt.

3 Ausdrücke

3.1 Code-Verständnis

Überlege für jeden der folgenden Ausdrücke, ob sie korrekten Python-Code darstellen und von welchem Datentyp sie in diesem Fall sind.

- 2
- 10.0
- true

- 'Lorem'
- True
- "ipsum"

> - *korrekt,* `int`
> - *korrekt,* `float`
> - *falsch, richtig wäre* `True`
>
> - *korrekt,* `str`
> - *korrekt,* `bool`
> - *korrekt,* `str`

3.2 Mathe 0 für Informatiker

Wenn Programme nur das zurückgeben würden, was man hinschreibt, hätte man noch lange kein vollständigen Computer. Deswegen möchte man auch Berechnungen durchführen können. Überlege dir, was die folgenden Ausdrücke ergeben, und für welche Rechnenoperation die verwendeten Symbole stehen. Beachte dabei die Präzedenz der einzelnen Operatoren. Lasse dir auch immer den Datentyp des Ergebnisses ausgeben.

- `16 + 26`
- `12 * 123 // 28 - 15`
- `5 * 3 + 6 * 6`
- `13.75 + 28.67`
- `2**6 + 2**3 + 2**1`
- `5.4 * 7.7`

> - *42*
> - *37*
> - *51*
> - *42.42*
> - *74*
> - *41.58*

4 Konvertierung

4.1 Grundlagen

Führe folgende Konvertierungen durch und notiere, was die Ergebnisse sind und welche davon nicht funktionieren:

1. 3.17 -> int
2. 'a' -> int
3. 5 -> float
4. '5' -> int
5. 3.1415 -> string
6. 5 -> bool
7. 'false' -> bool

1. `int(3.17)` → `3`
2. Fehler: `ValueError: invalid literal for int() with base 10: 'a'`
3. `float(5)` → `5.0`
4. `int('5')` → `5`
5. `str(3.1415)` → `'3.1415'`
6. `bool(5)` → `True`
7. `bool('false')` → `True`

4.2 Überkomplizierte Eingabe

Im Folgenden soll eine Ganzzahl unter Nutzung der Stringkonstante `"1"` und der Zahlenkonstante `0` sowie der Konvertierungsfunktionen `int()` und `str()` ausgewertet werden. Arithmetische und String-Operationen sind erlaubt, allerdings keine weiteren Konstanten wie z.B. 2 oder ”19”.
Beispiel: Mit `"1"+ "1"+ str(0)` bekommt man das Ergebnis `"110"`. Dieses kann wiederum in einen Integer umgewandelt werden, um damit weiter zu rechnen.
Kombiniere auf diese Weise die Konstanten, sodass sich am Ende die Zahl `219` ergibt.

> `int(str(int("1"+ "1")* (int("1")+ int("1")))+ str(0))- int("1")`
> *Man beachte, dass Ausdrücke wie in dieser Aufgabe nicht in echtem Code verwendet werden sollten. Es ist durchaus möglich, dass es alternative Ansätze gibt, welche auf das gleiche Ergebnis kommen.*

5 Variablen

5.1 Fehlersuche

Gegeben ist folgendes Listing:

```
1 meinAlter = 21
2 meinalter = meinalter + 1
3 print(meinAlter)
```

Beim Ausführen der zweiten Zeile wirft Python einen Fehler. Warum? Verbessere das Listing.

> *Die Variable in Zeile 2 müsste `meinAlter` heißen und nicht `meinalter`.*

5.2 Gültigkeit & Konvention

a) `matrikel_nummer`
b) `ÄhmKeineAhnung`
c) `2fancy4python`
d) `_meinAlter`
e) `richtig&gut`
f) `_2Euro`
g) `Variable2`
h) `noch_besser`
i) `#Falsch`

5.2.1 Welche der aufgelisteten Variablennamen sind gültige Variablennamen?

> *Die Antworten c), e) und i) sind nicht gültig.*

5.2.2 Welche entsprechen der Namenskonvention snake_case?

> *Nur die Antworten a) und h) entsprechen der Namenskonvention **snake_case** und sind gültig.*

6 RSA (Bonus)

Solltest du bei dieser Aufgabe Fragen zur verwendeten Mathematik oder Schreibweise haben, frag uns gerne. Für eine RSA-Verschlüsselung werden zwei Primzahlen p, q benötigt. Deren Produkt definiert einen Ring $\mathbb{Z}_n$ mit $n = p \cdot q$. Das heißt, nach jeder Operation wir das Ergebnis mittels Modulo in das Intervall $[0, n)$ gebracht.

Wenn man nun etwas verschlüsseln/entschlüsseln möchte, benötigt man zunächst $N = (n) = (p-1) \cdot (q-1)$. Außerdem wird ein $0 < e < N$ gebraucht, mit dem der größte gemeinsamen Teiler von e und N 1 ist sowie ein $0 < d < N$ mit $e \cdot d \; mod \; N = 1$.

Da das hier keine Krypto-Vorlesung ist, soll die Berechnung dieser Werte übersprungen werden und es sind folgende Werte vorgegeben: $p = 3$, $q = 11$, $n = 33$, $N = 20$, $e = 3$, $d = 7$. Eine Nachricht m wird nun als $m^e \; mod \; n$ verschlüsselt, ein Chiffretext c als $c^d \; mod \; n$ entschlüsselt.

Verschlüssele mit den vorgegebenen (oder eigenen) Werten nun ein paar Zahlen ($0 \leq x < n$), und probiere, ob du sie wieder entschlüsseln kannst. Du kannst diese auch an euren Sitznachbarn weitergeben.

> *Ein Beispiel wäre die folgende Rechnung:*
>
> ```
> >>> (13 ** 3) % 33
> 19
> >>> (19 ** 7) % 33
> 13
> ```
>
> *Um Strings aus Kleinbuchstaben damit zu verschlüsseln/entschlüsseln, könnte man diese Funktionen verwenden:*
>
> ```
> >>> enc = lambda x: [(ord(c) - ord('a')) ** 3 % 33 for c in x if ord('a') <= ord(c) and
>     ord(c) <= ord('z')]
> >>> dec = lambda x: "".join([chr ((v ** 7 % 33) + ord('a')) for v in x if 0 <= v and v <
>     33])
>
> >>> cipher = enc('lorem')
> >>> message = dec(cipher)
> ```