





- Dieses Dokument enthält Aufgaben für den Zeitraum **Dienstag, 02.03.2021 – Dienstag, 9.03.2021**. Die Lösungen der Aufgaben werden in der darauffolgenden Stunde besprochen. (Per Video- oder Audiochat??)
- Die Aufgaben sind als tägliche Fingerübung gedacht. Zu jeder Aufgabe steht das vorgeschlagene Zeitpensum im Titel. So könnt ihr euch testen wie gut ihr schon seid.
- Kontaktiere mich bei Fragen oder sonstigen Anliegen unter:
ralf.dorn@hhgym.de.
- **Hinweis:** Das war scheinbar noch nicht jedem klar. Ihr könnt Einrückungen mit  statt mit  erzeugen. Das entspricht standardmäßig einem *whitespace* von vier Leerzeichen und schont eure Finger.

Aufgabe 1: Collatz-Algorithmus

(Di.+Mi.)

Vielleicht kennt ihr schon das sogenannte *Collatz-Algorithmus* (Na klar. Hatten wir doch letzte Woche.). Diese Funktion soll für irgendein $n \in \mathbb{N}$ das Folgende tun: Wenn n gerade ist, ersetzt man es durch $n/2$, wenn n ungerade ist, nimmt man $3n + 1$. Dieses Vorgehen wiederholt man für die erhaltene Zahl solange sie größer als eins ist.

Eine mögliche Funktion, die n und jede folgende Zahl auf dem Bildschirm ausgibt, zeigt der unten stehende Code.

```
1 def collatz(n):
2     while n > 1:
3         print(n)
4         if n%2 == 0:
5             n = n/2
6         else:
7             n = 3*n + 1
```

Für $n = 3$ liefert die Funktion diese Ausgabe:

```
8 >>> collatz(3)
9 10
10 5.0
11 16.0
12 8.0
13 4.0
14 2.0
15 1.0
```

Nun zu den Aufgaben:

- (a) Gib eine Kurzschreibweise für Zeile 5 an.
- (b) Die Funktion gibt in Python 3 ab einem gewissen Rechenschritt (Zeilen 10 – 15) Dezimalzahlen aus, es steht zum Beispiel 5,0 statt 5. Es findet also ein *implizites Casting* von einer Ganzzahl `int` zu einer Fließkommazahl `float` statt. Welche Stelle im Code ruft diese Typumwandlung vor? Tipp: Welche Rechenoperation kann aus ganzen Zahlen Dezimalzahlen machen? Finde heraus, welcher *Operator* (wie `+-*/`) verwendet werden muss, damit keine `float`-Werte entstehen.
- (c) Um zu überprüfen, ob n gerade ist, haben wir im Code verwendet, dass die Division von n durch zwei keinen Rest ergibt. Formuliere mit dem neuen Operator der vorhergehenden Aufgabe eine alternative Äquivalenz für „ n ist gerade“.
- (d) Erweitere die Funktion so, dass nach dem Verlassen der Schleife im Format „Schritte: X“ ausgegeben wird, wie viele Rechenschritte durchlaufen wurden. Für $n = 3$ wären das zum Beispiel sechs.
- (e) Die Kontrollstruktur (Zeilen 4 – 7) weist n in jedem Falle einen neuen Wert zu. Schreibe die Zuweisung eleganter innerhalb einer Zeile.
- (f) Ich habe für euch diesen Lösungsvorschlag anzubieten:

```

16 | def collatz_falsch_1(n):
17 |     while n > 1:
18 |         if n%2 == 0:
19 |             print(n/2)
20 |         else:
21 |             print(3*n + 1)

```

Für $n \neq 1$ resultiert das Programm in einer Endlosschleife. Was ist falsch? Welcher im Struktogramm abgebildete Schritt wurde unterschlagen?

Hinweis am Rande: Wenn dein Programm in einer Endlosschleife feststeckt, rufst du in der Shell mit `Strg C` eine `KeyboardInterrupt`-Ausnahme hervor. Das Programm wird „sauber“ abgebrochen und muss nicht durch das Schließen der Shell „gekillt“ werden.

- (g) Auch das habe ich schon gesehen:

```

22 | def collatz_falsch_2(n):
23 |     while n > 1:
24 |         if n%2 == 0:
25 |             return n/2
26 |         else:
27 |             return 3*n + 1

```

Es wird nichts auf dem Bildschirm ausgegeben, weil `print` nicht benutzt wurde. Aber auch die `while`-Schleife wird beim Aufrufen sofort abgebrochen. Warum? Rufe `collatz_falsch_2(n)` für verschiedene n in der Shell auf und erkläre die Beobachtung.

Wer hiermit fertig ist, soll möglichst frei von Hausaufgaben in Ruhe seine Freizeit genießen.

Aufgabe 2: Schweine-Latein

(Mi.+Do.)

Die Geheimsprache „Pig Latin“ ist im englischen Sprachraum bei Kindern beliebt und richtet sich (hier vereinfacht) nach der folgenden Zuordnungsvorschrift:

- Beginnt ein Wort mit einem Konsonanten, wird dieser an das Ende des Wortes verschoben und es wird ein „ay“ angehängt. Aus „Pig Latin“ wird „Igpay Atinlay“.
- Beginnt ein Wort dagegen mit einem Vokal, wird direkt ein „ay“ angehängt. Aus „America“ wird „Americaay“.

Zu den Aufgaben:

- (a) Der folgende Auszug aus der Shell zeigt grundlegende Eigenschaften von Strings in Python.

```
28 >>> a = "Python"
29 >>> a[0]
30 'P'
31 >>> a[1]
32 'y'
33 >>> a[-1]
34 'n'
35 >>> a[-2]
36 'o'
37 >>> a[0:3]
38 'Pyt'
39 >>> a[-3:-1]
40 'ho'
41 >>> a[:]
42 'Python'
43 >>> a.lower()
44 'python'
45 >>> a.upper()
46 'PYTHON'

47 >>> "python".capitalize()
48 'Python'
49 >>> a[0].isupper()
50 True
51 >>> for char in a: char
52
53 'P'
54 'y'
55 't'
56 'h'
57 'o'
58 'n'
59 >>> "Py" in a
60 True
61 >>> "THON" in a
62 False
63 >>> "Pytho" + "\n"
64 'Python'
```

Versuche, alle Schritte nachzuvollziehen. Eine Erläuterung der `str`-Klasse erhältst du, indem du `help(str)` in die Shell eingibst. Ansonsten recherchiere Unbekanntes im Internet.

- (b) Schreibe eine Funktion `pig_latin(word)`, die als Rückgabewert die Übersetzung eines Wortes `word` ins Schweine-Latein besitzt.

Du darfst davon ausgehen, dass der Benutzer nur Wörter eingibt (keine Zahlen o.Ä.), die sich außerdem nicht im *UPPER CASE* befinden. Mache unter dieser Berücksichtigung folgende Ergänzung: Wenn das Wort *Capitalized* ist, also mit einem großen Buchstaben anfängt, soll das auch der Rückgabewert sein.

Beispiel: `pig_latin("word")` → `"ordway"` und `pig_latin("Word")` → `"Ordway"`

Aufgabe 3: Rekursive und iterative Funktionen

(Fr.+Sa.)

Im Unterricht haben wir für $n \in \mathbb{N} \cup \{0\}$ die Fakultät von n nach der Definition $0! := 1$ und $n! := n \cdot (n-1)!$ für $n \geq 1$ in Python umgesetzt und sind zu einem Ergebnis gekommen, das genau diese rekursive Definition benutzt.

```
65 def fac_rec(n):
66     return 1 if n == 0 else n * fac_rec(n-1)
```

Nun ist das nicht die einzige Lösung, $n!$ in Python aufzuschreiben. Eine *iterative* Lösung, in der Zahlen von einer Schleife durchlaufen (*iteriert*) werden, kann man sich anhand der alternativen Definition

$$n! := 1 \cdot 2 \cdot 3 \cdot \dots \cdot (n-1) \cdot n$$

erarbeiten (es folgt daraus $0! = 1$, „leeres Produkt“). Für n starten wir also beim Wert eins und multiplizieren alle Zahlen $i \in [2, 3, \dots, (n-1), n]$ hinzu. Die Variable i heißt *Iterationsvariable* und durchläuft diese geordnete Liste.

Zu den Aufgaben:

- (a) Schreibe eine iterative Funktion `fac_iter(n)`, die für eine Zahl $n \in \mathbb{N} \cup \{0\}$ ihre Fakultät $n!$ zurückgibt.
- (b) Sei $(f_n)_{n \in \mathbb{N}}$ die Zahlenfolge mit der rekursiven Bildungsvorschrift $f_1 := 2$, $f_2 := 4$ und $f_n := 3f_{n-2} + f_{n-1}$ für $n \geq 3$. Es gilt also

$$(f_n)_{n \in \mathbb{N}} = (2, 4, 10, 22, 52, 118, \dots).$$

Schreibe eine rekursive Funktion `f_rec(n)` und eine iterative Funktion `f_iter(n)`, die für $n \in \mathbb{N}$ das Folgeglied f_n zurückgibt.

Lösungsvorschläge können im Lernraum hochgeladen werden, und zwar als Python-Datei im Format

```
67 # Aufgabe 1
68
69 # 1a
70
71 """Erläuternder Text, dafür
72 eignen sich multiline Strings
73 besonders gut!"""
74
75 def irgendein_code(n):
76     pass
77
78 # 1b
79 ...
```

Na dann: Viel Erfolg!

R. Dorn