Inhaltsverzeichnis

1. Vorwort

Python hat es mir jetzt echt angetan.

Nach dem ersten Projekt mit dem Turtle-Paket habe ich mir diverse Videos und Tutorials angesehen, Python ist so vielseitig, ich mache damit jetzt definitiv weiter.

Aus meiner Sicht ist einer der ersten Prüfsteine einer Programmiersprache immer die Entwicklung einer Benutzeroberfläche. Das ist meistens entweder sehr komplex oder es sieht nicht gut aus.

Deshalb bin ich in diesem Projekt eine GUI in Python angegangen.

Eine Projektidee hat sich auch schnellgefunden – einer meiner Neffen ist jetzt im Alter, das Einmaleins auswendig lernen zu müssen. Da die Motivation dazu wohl eher überschaubar ist, habe ich versucht, mit einem kleinen Spiel für mehr Abwechslung zu sorgen. Die Oberfläche ist mit Absicht einfach gestaltet, damit es da keine Ablenkung gibt.

Am Ende der Doku ist noch ein Kapitel der Erstellung einer unter Windows ausführbaren Datei (.exe) gewidmet, sodass das Spiel wie gewohnt mit einem Doppelklick geöffnet werden kann.

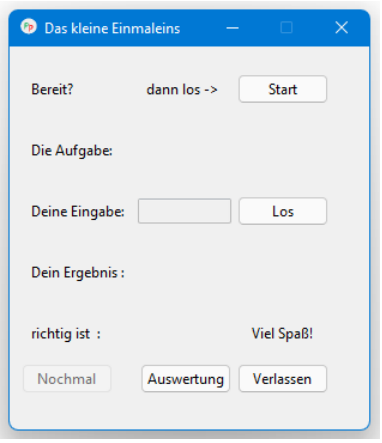
Ich hoffe, das Projekt gefällt Euch und Ihr habt genauso viel Spaß es zu programmieren wie ich es hatte.

Wie gewohnt werde ich in den nachfolgenden Kapiteln sowohl die Installation, als auch die Programmierung selber Schritt für Schritt erklären. Wichtig ist aber, dass Ihr selber ins Rollen kommt. Denn wie immer bei der Programmierung gilt – Probieren geht über Studieren.

2. Das Projekt

Wie im Vorwort beschrieben, geht es in diesem Projekt primär darum, die Grundzüge der GUI-Programmierung in Python zu verstehen und anzuwenden.

Das Spiel sieht so aus:



Nach Aufruf öffnet sich ein eigenes Fenster.

Zu sehen sind 6 Zeilen und 3 Spalten mit festen Werten, Knöpfen und einem Eingabefeld in der Mitte.

Erst mit Drücken des Start-Knopfs wird eine Aufgabe generiert und das Eingabefeld geht auf für die Eingabe des Ergebnisses durch den Spieler. Auch der Nochmal-Knopf wird entsperrt.

Mit Klick auf den Los-Knopf wird die Eingabe mit dem tatsächlichen Ergebnis verglichen. Beide Werte werden angezeigt, bei Übereinstimmung gibt es einen Punkt für die richtigen Antworten, sonst gibt es einen Punkt für die falschen Antworten.

Mit Klick auf den Auswertung-Knopf geht ein neues Fenster auf, in dem die Summen für die richtigen und die falschen Antworten angezeigt werden.

Mit Klick auf den Nochmal-Knopf werden alle bisher gesammelten Ergebnisse zurückgesetzt und eine neue Aufgabe erscheint.

Der Klick auf den Verlassen-Knopf beendet das Programm.

Fehleingaben werden durch Prüfungen abgefangen und Hinweise werden ausgegeben.

Mehr dazu in den einzelnen Abschnitten. Und jetzt – auf geht's!

3. Vorarbeiten

Ein Teil der Vorarbeiten ist auch, die Auswahl einer geeigneten IDE.

Aber was ist ‚geeignet' in diesem Zusammenhang? Die im ersten Projekt benutzte IDLE erfüllt ihren Zweck, ist aber nicht besonders nutzerfreundlich.

Ich habe mir verschiedene IDEs angeschaut und ausprobiert. Nach meinen bisherigen Erfahrungen gibt es drei Gruppen von IDEs. Die einen kommen als leere Editor-Hüllen daher, man muss sich die benötigten Pakete der Reihe nach aus dem Internet ziehen. Dafür sind sie nicht speziell auf Python ausgerichtet, man kann alle möglichen Pakete für die Entwicklung von zum Beispiel C++ oder JavaScript hinzuladen. So eine IDE ist zum Beispiel ‚Atom', die ich bei mir eingerichtet habe.

Die zweite Gruppe sind ‚Bundles', bei denen die Entwicklungskonsole und die Laufzeitumgebung in einem Modul zusammen auftreten. Ein Vertreter ist zum Beispiel ‚Eric' oder ‚TigerJython'.

Die dritte Gruppe versucht einen Mittelweg zu gehen. Also die IDE ist klar auf Python ausgelegt, aber die Laufzeitumgebung kann und muss man zusätzlich installieren. Ein Beispiel dafür ist PyCharm von Jetbrains (https://www.jetbrains.com/pycharm). Es gibt eine kostenlose Community-Version, für die professionelle Version muss man dann zahlen. Auch in der kostenlosen Version ist das ein mächtiges und echt professionelles Tool, für den Anfang aber vielleicht zu verwirrend.

Bei der Suche nach einem Ersatz für IDLE bin ich dann auf ‚Thonny' gestoßen. ‚Thonny' gehört zur zweiten Gruppe und ist daher für den Anfang gut geeignet. Es bietet eine große Zahl von Analysefunktionen um sich wirklich mit dem Code auseinanderzusetzen. Außerdem habe ich das Programm auf dem Raspberry Pi unter Raspberry Pi OS (früher bekannt als ‚Raspbian') gefunden, also perfekt für plattformübergreifende Entwicklung geeignet.

Der Code für dieses Projekt wird mit ‚Thonny' gemacht, daher müssen wir uns die Software zuerst installieren.

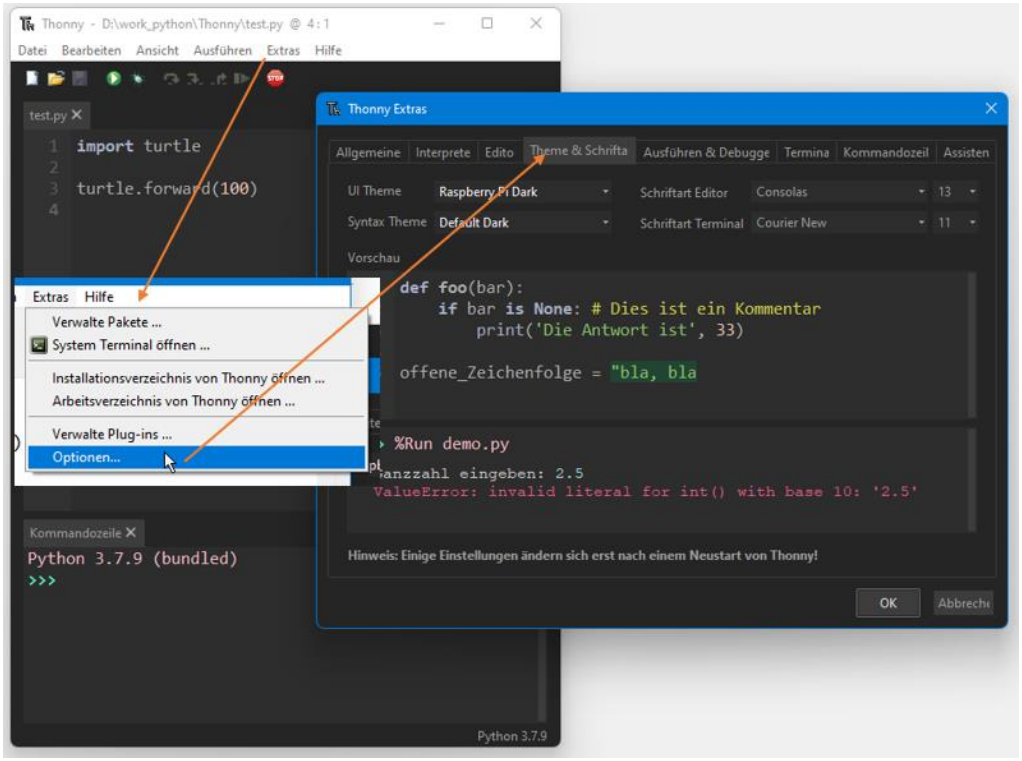
En Download gibt es auf https://thonny.org/ - bei mir wurde die Version 3.3.13 angeboten.

Die Installation selber ist nicht kompliziert, allerdings werden bis zum Start neun Fenster zur Bestätigung abgefragt.

Links ein Screenshot von ‚Thonny', unten zu erkennen ist die Umgebung mit Python 3.7.9 als Bundle.

Als erstes Programm habe ich die ersten Zeilen Code aus dem vorigen Projekt aufgenommen. Zum Ausführen der Anweisungen einfach auf den Pfeil-Knopf drücken, alternativ mit Funktionstaste F5.

Wer einen dunklen Hintergrund bevorzugt, kann das Erscheinungsbild auch über ‚Extras/Optionen…/ Theme&Schriftarten' ändern:



Damit aber genug der Vorarbeiten, los geht's!

4. Projekt „Einmaleins-Spiel"

Um das Projekt etwas zu strukturieren, machen wir uns zuerst Gedanken über den Funktionsumfang, also die Frage nach dem ‚Was'.

Dann gehen wir das ‚Wie' an. Fest steht ja schon, dass wir das mit Python lösen wollen, wir schauen aber nochmal auf die benötigten Pakete.

Im dritten Teil dann kommt die Programmierung dran.

Soweit so gut, gehen wir es an.

4.1. der Funktionsumfang

Das Programm soll dem Spieler die Möglichkeit geben, Einmaleins-Aufgaben zu lösen.

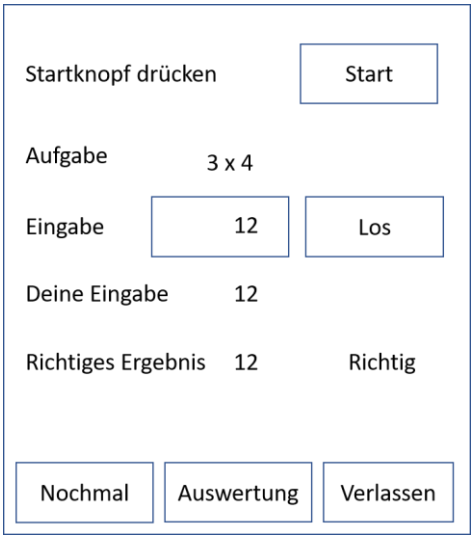
Das System stellt eine Aufgabe und gibt dem Spieler die Möglichkeit einer Eingabe. Dabei sollen nur Ganzzahlen zugelassen werden. Sollte die Eingabe leer sein oder Buchstaben enthalten, ist eine Fehlermeldung auszugeben.

Die Eingabe soll gegen das tatsächliche Ergebnis geprüft werden, beide Werte sollen dem Spieler angezeigt werden. Richtige und falsche Antworten sollen gezählt werden. Am Ende soll der Spieler die Möglichkeit haben, sich seinen Spielstand anzuschauen.

Eine Wiederholung des Spiels muss möglich sein, ohne das Spiel erneut starten zu müssen.

4.2. der Planung der Umsetzung

Jetzt zum ‚Wie'. Machen wir uns zuerst Gedanken um das Aussehen unserer App.



Links sehen wir eine Skizze, wie das Aussehen der App sein könnte, wenn wir uns an den fachlichen Anforderungen orientieren.

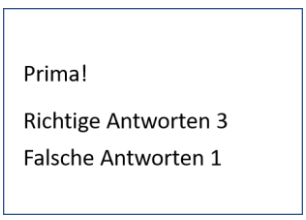
Die Anwendung hat einen Start-Knopf, bei Klick darauf wird eine Aufgabe ermittelt und angezeigt.

In der Mitte hat der Spieler die Möglichkeit, seine Eingabe zu tätigen.

Bei Klick auf den Los-Knopf wird

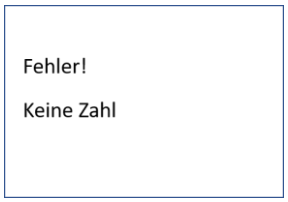
- die Eingabe mit dem Ergebnis verglichen
- beide Werte werden angezeigt
- ‚Richtig' oder ‚Falsch' wird angezeigt
- Eine neue Aufgabe wird ermittelt und angezeigt

Wenn der Spieler keine Lust mehr hat, kann er auf den Auswertung-Knopf klicken.



Dann bekommt er in einem separaten Fenster sein erzieltes Ergebnis angezeigt.

Bei Klick auf den Nochmal-Knopf wird das Spiel erneut gestartet, alle bisher erzielten Ergebnisse werden zurückgesetzt.



Wurde im Eingabe-Feld etwas anderes als eine Ganzzahl eingegeben, oder ist das Feld bei Klick auf den Los-Knopf leer, erfolgt die Anzeige eines Fehlerfensters.

Bei Klick auf den Verlassen-Knopf schließt sich die Anwendung ohne weiteren Hinweis.

Soweit zum Aussehen der Anwendung.

Für die Gestaltung der GUI stehen in Python diverse Pakete zur Verfügung. Für den Anfang sind wir wohl bei `Tkinter` ganz gut aufgehoben. Sobald ich meine Kenntnisse und Fertigkeiten erweitert habe, werde ich mir andere Pakete anschauen, zum Beispiel `Qt` oder auch `Kivy`, das scheint mir da ganz modern zu sein.

Bei der Auswahl der Pakete zeigt sich wieder die Schwierigkeit mit der richtigen Paketauswahl. Eigentlich muss man alle mal ausprobiert haben, denn jedes hat seine Vor- und Nachteile und ist für die eine Aufgabenstellung besser geeignet, als für eine andere.

Also, wie gesagt, wir bleiben erst einmal bei `Tkinter`.

Darüber hinaus benötigen wir für die Generierung von Zufallszahlen das Paket `random`, das müssen wir auch importieren.

Achtung Spoiler: ganz wichtig das explizite Importieren von `tk.messagebox`, sonst werden die Pop-Up-Fenster in der exe nicht angezeigt.

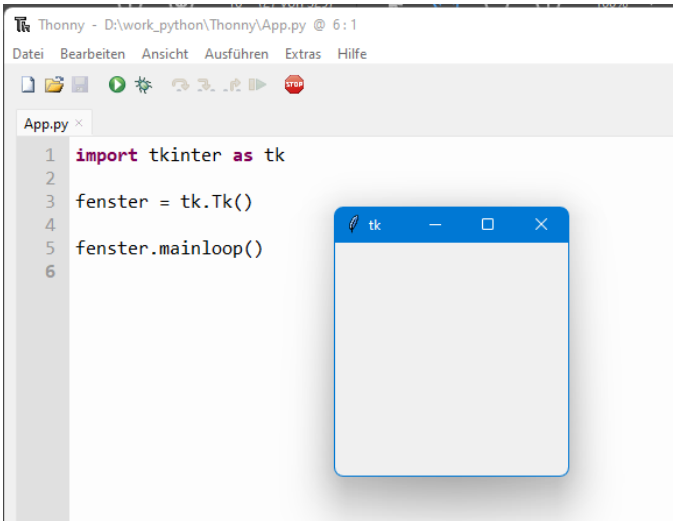
Aber alles zu seiner Zeit, jetzt geht es erstmal los.

4.3. das Coden

Für den Einstieg in das Thema `Tkinter` habe ich mir im Netz diverse Beispiele angeschaut und versucht zu adaptieren, was wir für unser Projekt brauchen.

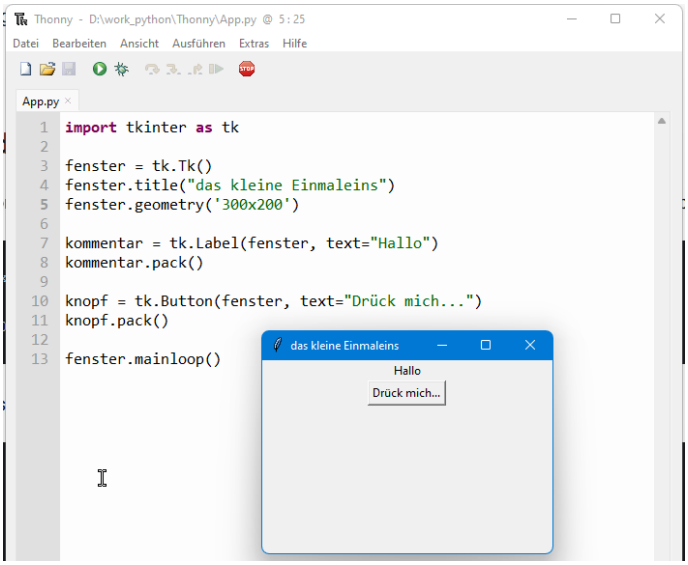
4.3.1. Vorgeplänkel

Die kleinste Einheit zur Erzeugung eines Fensters in `Tkinter` ist:



Alles, was im Fenster angezeigt werden soll, muss zwischen der Definition in Zeile 3 und dem `mainloop()` in Zeile 5 angeordnet werden.

Als sinnvolle erste Ergänzung hier noch ein paar zusätzliche Zeilen Code.



Was ist zu sehen?

In Zeile 4 wird die Überschrift gesetzt.

Zeile 5 gibt die Größe des Fensters an.

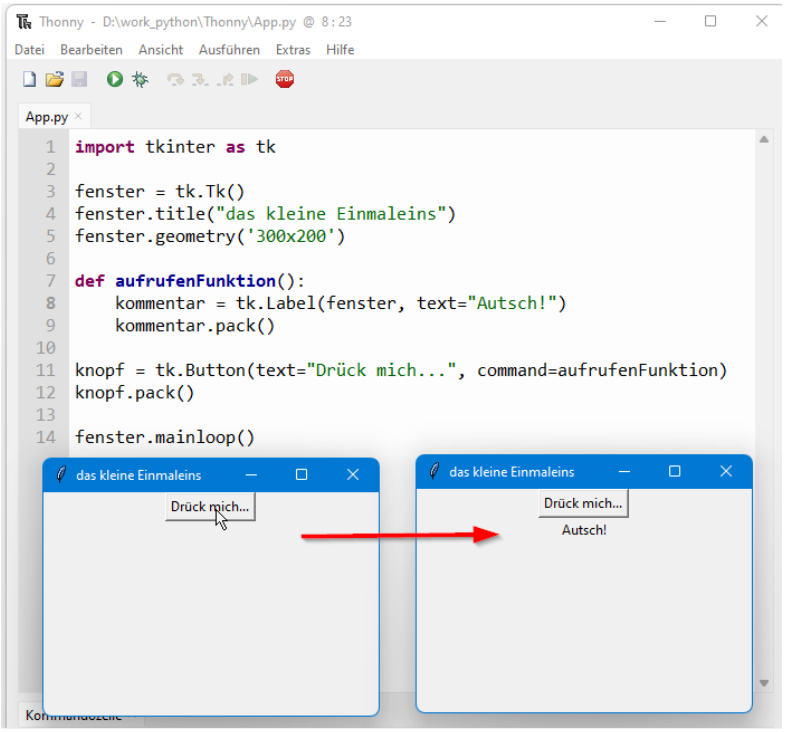
Zeile 7 ist die Definition eines Kommentars (zukünftig werde ich den englischen Begriff ‚Label' verwenden, das geht mir leichter von der Hand).

Zeile 9 ist die Definition eines Knopfs. Auch hier werde ich zukünftig lieber von ‚Button' schreiben.

Label und Button werden als ‚Widgets' bezeichnet. Das ist ein Kunstwort aus ‚Window' und ‚Gadget' und bezeichnet die Komponenten wie Button und Label.

Mit der Funktion `pack()` in den Zeilen 8 und 10 werden die Widgets im Fenster platziert.

Einem Button kann der Aufruf einer Funktion mitgegeben werden, das ist sehr praktisch und wir werden das später nutzen.



4.3.2. Die Programmstruktur

Da wir in der Anwendung verschiedene Operationen ausführen müssen, ist für uns ein etwas anderer Ansatz besser geeignet.

Wir nutzen das Class-Konzept und erstellen uns eine eigene Klasse für das Fenster.

```
 1 import tkinter as tk
 2 from tkinter import ttk
 3
 4 class Einmaleins(ttk.Frame):
 5     def __init__(self, container):
 6         super().__init__(container)
 7
 8
 9 class App(tk.Tk):
10     def __init__(self):
11         super().__init__()
12
13         self.title('Das kleine Einmaleins')
14         self.geometry('400x300')
15         self.resizable(False, False)
16
17
18 if __name__ == "__main__":
19     app = App()
20     Einmaleins(app)
21     app.mainloop()
```

Gehen wir es kurz durch.

Der Programmstart erfolgt in Zeile 18 mit der `if`-Anweisung. In Zeile 19 erzeugen wir eine Instanz der Klasse App, die in Zeile 9 definiert ist. Eine Klasse wird wie eine Funktion definiert, das Schlüsselwort `class` statt `def` zeigt an, dass es sich jetzt um ein Klasse handelt.

In Zeile 20 erzeugen wir eine Instanz von Frame, das ist das Top-Level Widget von `Tkinter` in dem alle anderen Widgets platziert werden.

Interessant noch Zeile 2, `ttk` hat die moderneren Widgets im Vergleich zu `tk`, die Definition geht dann nicht über

```
button = tk.Button(fenster, text= 'Blabla')
```

sondern über

```
button = ttk.Button(fenster, text= 'Blabla')
```

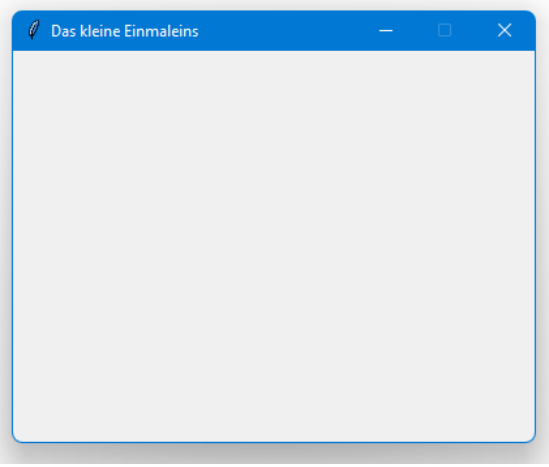
Unsere Aktivitäten beschränken sich im Wesentlichen auf die Klasse `Einmaleins` ab der Zeile 4.

Noch eine Randnotiz:

Hier zeigt sich der prozedurale Ansatz von Python. Die *Definition* der Klassen ist eigentlich unabhängig von deren *Verwendung*. In anderen Sprachen ist es egal wo welcher Code-Block steht. In Python ist die Reihenfolge wichtig. Würden wir die Zeilen 18 bis 21 vor die Zeile 4 kopieren, würde das Programm nicht laufen und eine Fehlermeldung ausgeben.

4.3.3. Die Funktion `grid()`

Zurück zu unserem Programm – wenn wir den Code ausführen, sollten wir nur ein einfaches Fenster sehen:



Nicht weiter verwunderlich, wir haben ja noch nichts mitgegeben.

Wenn wir die Zeichnung aus Kapitel 4.2 noch einmal hernehmen, die Inhalte ein bisschen nach links und rechts verschieben und ein Gitter drüberlegen, erkennen wir einen Tabellenstruktur, die 6 Zeilen und 3 Spalten hat.

	Spalte 1	Spalte 2	Spalte 3
Zeile 1	Startknopf drücken		Start
Zeile 2	Aufgabe	3 x 4	
Zeile 3	Eingabe	12	Los
Zeile 4	Deine Eingabe	12	
Zeile 5	Richtiges Ergebnis	12	Richtig
Zeile 6	Nochmal	Auswertung	Verlassen

Für die Abbildung einer Tabelle bietet `Tkinter` die Funktion `grid()` an. Der Aufbau ist

```
<widget>.grid(row=<nummer>, column=<nummer>)
```

Zu beachten hier, es geht weder in der Zeile noch in der Spalte mit ‚1' los, sondern mit ‚0'!

Wir haben pro Zelle in der Tabelle immer 2 Zeilen Code, die *Deklaration* und die *Positionierung* mit `grid()`.

Machen wir das gemeinsam für die Zelle in Zeile 1, Spalte 1. Dort haben wir ein Widget vom Typ ‚Label'. Die **Deklaration** ist also

```
label_z1_s1 = ttk.Label(fenster, text= 'Startknopf drücken')
```

Die Positionierung ist

```
self.label_z1_s1.grid(row=0, column=0)
```

Prinzip ist klar geworden, oder?

Wir haben also 18 Zellen, die wir deklarieren und positionieren müssen. Also 36 Zeilen Code nur für die einzelnen Zellen.

Ich habe da mal was vorbereitet:

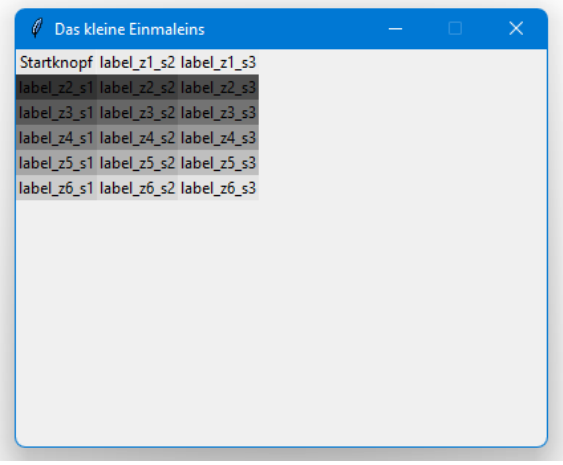
```
import tkinter as tk
from tkinter import ttk

class Einmaleins(ttk.Frame):
    def __init__(self, container):
        super().__init__(container)
        # Zeile 1
        self.label_z1_s1 = ttk.Label(self, text= 'Startknopf', background='grey91')
        self.label_z1_s1.grid(row=0, column=0)
        self.label_z1_s2 = ttk.Label(self, text= 'label_z1_s2', background='grey92')
        self.label_z1_s2.grid(row=0, column=1)
        self.label_z1_s3 = ttk.Label(self, text= 'label_z1_s3', background='grey93')
        self.label_z1_s3.grid(row=0, column=2)
        # Zeile 2
        self.label_z2_s1 = ttk.Label(self, text= 'label_z2_s1', background='grey20')
        self.label_z2_s1.grid(row=1, column=0)
        self.label_z2_s2 = ttk.Label(self, text= 'label_z2_s2', background='grey25')
        self.label_z2_s2.grid(row=1, column=1)
        self.label_z2_s3 = ttk.Label(self, text= 'label_z2_s3', background='grey30')
        self.label_z2_s3.grid(row=1, column=2)
        # Zeile 3
        self.label_z3_s1 = ttk.Label(self, text= 'label_z3_s1', background='grey35')
        self.label_z3_s1.grid(row=2, column=0)
        self.label_z3_s2 = ttk.Label(self, text= 'label_z3_s2', background='grey40')
        self.label_z3_s2.grid(row=2, column=1)
        self.label_z3_s3 = ttk.Label(self, text= 'label_z3_s3', background='grey45')
        self.label_z3_s3.grid(row=2, column=2)
        # Zeile 4
        self.label_z4_s1 = ttk.Label(self, text= 'label_z4_s1', background='grey50')
        self.label_z4_s1.grid(row=3, column=0)
        self.label_z4_s2 = ttk.Label(self, text= 'label_z4_s2', background='grey55')
        self.label_z4_s2.grid(row=3, column=1)
        self.label_z4_s3 = ttk.Label(self, text= 'label_z4_s3', background='grey60')
        self.label_z4_s3.grid(row=3, column=2)
        # Zeile 5
        self.label_z5_s1 = ttk.Label(self, text= 'label_z5_s1', background='grey65')
        self.label_z5_s1.grid(row=4, column=0)
        self.label_z5_s2 = ttk.Label(self, text= 'label_z5_s2', background='grey70')
        self.label_z5_s2.grid(row=4, column=1)
        self.label_z5_s3 = ttk.Label(self, text= 'label_z5_s3', background='grey75')
        self.label_z5_s3.grid(row=4, column=2)
        # Zeile 6
        self.label_z6_s1 = ttk.Label(self, text= 'label_z6_s1', background='grey80')
        self.label_z6_s1.grid(row=5, column=0)
        self.label_z6_s2 = ttk.Label(self, text= 'label_z6_s2', background='grey85')
        self.label_z6_s2.grid(row=5, column=1)
        self.label_z6_s3 = ttk.Label(self, text= 'label_z6_s3', background='grey90')
        self.label_z6_s3.grid(row=5, column=2)
        # alles zur Anzeige bringen
        self.grid()

```

```
52 class App(tk.Tk):
53     def __init__(self):
54         super().__init__()
55 
56         self.title('Das kleine Einmaleins')
57         self.geometry('400x300')
58         self.resizable(False, False)
59 
60 
61 if __name__ == "__main__":
62     app = App()
63     Einmaleins(app)
64     app.mainloop()
65 
```

Der grüne Code erklärt sich von selbst, oder? Das Ergebnis ist wenig überraschend:



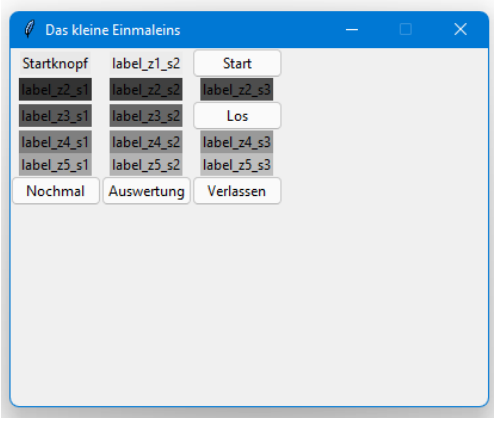
Nicht sehr überraschend, zugegeben.

Die nächste Iteration sind die 5 Button. Wir tauschen die Label gegen Button und löschen die `background`-Anweisung. Exemplarisch hier nur der Code für die Zeile 6:

```
41         …
42         # Zeile 6
43         self.button_nochmal = ttk.Button(self, text= 'Nochmal')
44         self.button_nochmal.grid(row=5, column=0)
45         self.button_auswertung = ttk.Button(self, text= 'Auswertung')
46         self.button_auswertung.grid(row=5, column=1)
47         self.button_verlassen = ttk.Button(self, text= 'Verlassen')
48         self.button_verlassen.grid(row=5, column=2)
49         # alles zur Anzeige bringen
50         self.grid()
51 
52 …
```

Tipp: Nennt die Widgets so, dass Ihr sie eindeutig zuordnen könnt, sonst verwirrt Ihr Euch später nur selber. Und lieber etwas längere Namen wählen, als ‚`b1`'…

Das Ergebnis sollte dann so aussehen:

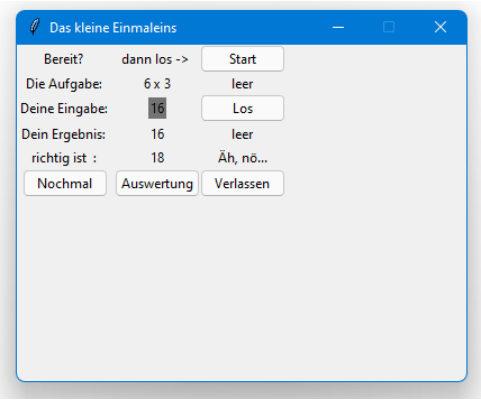


Jetzt ersetzen wir die Dummy-Label durch echten Text und fixe Werte, sodass wir sehen, wie das nachher aussehen soll. Der Code für die Klasse `Einmaleins` sieht dann so aus:

```
3  …
4  class Einmaleins(ttk.Frame):
5      def __init__(self, container):
6          super().__init__(container)
7          # Zeile 1
8          self.label_bereit = tk.Label(self, text= 'Bereit? ')
9          self.label_bereit.grid(row=0, column=0)
10         self.label_dannLos = tk.Label(self, text= 'dann los -> ')
11         self.label_dannLos.grid(row=0, column=1)
12         self.button_start = ttk.Button(self, text= 'Start')
13         self.button_start.grid(row=0, column=2)
14         # Zeile 2
15         self.label_dieAufgabe = ttk.Label(self, text= 'Die Aufgabe: ')
16         self.label_dieAufgabe.grid(row=1, column=0)
17         self.label_aufgabe = ttk.Label(self, text= '6 x 3')
18         self.label_aufgabe.grid(row=1, column=1)
19         self.label_z2_s3 = ttk.Label(self, text= 'leer')
20         self.label_z2_s3.grid(row=1, column=2)
21         # Zeile 3
22         self.label_deineEingabe = tk.Label(self, text= 'Deine Eingabe: ')
23         self.label_deineEingabe.grid(row=2, column=0)
24         self.label_z4_s2 = ttk.Label(self, text= '16', background='grey45')
25         self.label_z4_s2.grid(row=2, column=1)
26         self.button_los = ttk.Button(self, text= 'Los')
27         self.button_los.grid(row=2, column=2)
28         # Zeile 4
29         self.label_deineEingabe = tk.Label(self, text= 'Dein Ergebnis: ')
30         self.label_deineEingabe.grid(row=3, column=0)
31         self.label_z4_s2 = ttk.Label(self, text= '16')
32         self.label_z4_s2.grid(row=3, column=1)
33         self.label_z4_s3 = ttk.Label(self, text= 'leer')
34         self.label_z4_s3.grid(row=3, column=2)
35         # Zeile 5
36         self.label_deinErgebnis = tk.Label(self, text= 'richtig ist  : ')
37         self.label_deinErgebnis.grid(row=4, column=0)
38         self.label_z5_s2 = ttk.Label(self, text= '18')
39         self.label_z5_s2.grid(row=4, column=1)
40         self.label_erfolg = ttk.Label(self, text= 'Äh, nö...')
41         self.label_erfolg.grid(row=4, column=2)
```

```
42          # Zeile 6
43          self.button_nochmal = ttk.Button(self, text= 'Nochmal')
44          self.button_nochmal.grid(row=5, column=0)
45          self.button_auswertung = ttk.Button(self, text= 'Auswertung')
46          self.button_auswertung.grid(row=5, column=1)
47          self.button_verlassen = ttk.Button(self, text= 'Verlassen')
48          self.button_verlassen.grid(row=5, column=2)
49          # alles zur Anzeige bringen
50          self.grid()
51  …
```

Das Ergebnis sollte dann so aussehen:



Was jetzt noch fehlt ist die Eingabe. Das Widget dafür heißt `Entry()` und hat folgenden Aufbau:

```
entry = tk.Entry(fenster)
```

Also ganz einfach. Da wir mit der Eingabe arbeiten wollen, brauchen wir jetzt noch eine Variable, in der der Wert der Eingabe gespeichert werden soll. Die Syntax für einen String ist

```
eingabe = tk.StringVar()
```

Da wir hier eine Zahl eingeben wollen, könnten wir auch eine Integer-Variable einsetzen:

```
eingabe = tk.IntVar()
```

Da laut fachlicher Anforderung aber eine Fehlermeldung ausgegeben werden soll, können wir erst einmal eine String-Variable nutzen und den Inhalt dann später abgleichen.

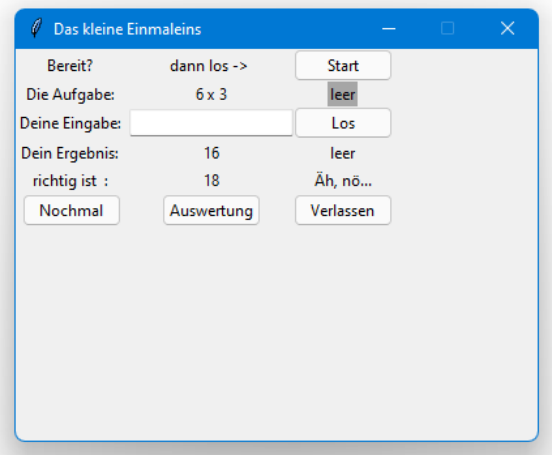
Damit das Widget und die Variable zusammenfinden, müssen wir das in der Deklaration des Widgets mit angeben:

```
entry = tk.Entry(fenster, textvarialble=eingabe)
```

Für die Zeile 3 sieht der neue Code dann so aus:

```
20  …
21          # Zeile 3
22          self.label_deineEingabe = tk.Label(self, text= 'Deine Eingabe: ')
23          self.label_deineEingabe.grid(row=2, column=0)
24          self.eingabe = tk.StringVar()
25          self.eingabefeld = ttk.Entry(self, textvariable=self.eingabe)
26          self.eingabefeld.grid(row=2, column=1)
27          self.button_los = ttk.Button(self, text= 'Los')
28          self.button_los.grid(row=2, column=2)
29  …
```

Das neue Aussehen ist dann



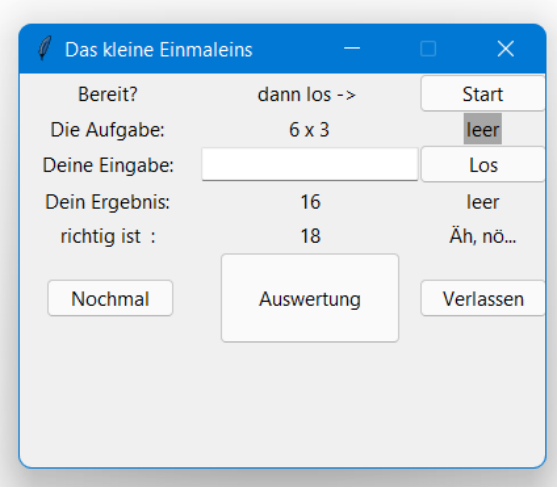
Jetzt haben wir alle Komponenten zusammen. Das Aussehen ist aber noch nicht wirklich schön. Zwischen den Zeilen und den Spalten sollte ein bisschen mehr Abstand sein, außerdem sollten die Texte in der Spalte 1 linksbündig sein.

In `grid()` stehen uns die Pärchen `padx` und `pady` bzw. `ipadx` und `ipady` zur Verfügung.

Schauen wir uns das anhand der Zeile 6 einmal genauer an.

```
41          …
42          # Zeile 6
43          self.button_nochmal = ttk.Button(self, text= 'Nochmal')
44          self.button_nochmal.grid(row=5, column=0, padx=20, pady=20)
45          self.button_auswertung = ttk.Button(self, text= 'Auswertung')
46          self.button_auswertung.grid(row=5, column=1)
47          self.button_verlassen = ttk.Button(self, text= 'Verlassen')
48          self.button_verlassen.grid(row=5, column=2, ipadx=20, ipady=20)
49          # alles zur Anzeige bringen
50          self.grid()
51
52  …
```

Das Ergebnis ist dann:



Was ist zu sehen? `padx` wirkt sich auf die *horizontale* Positionierung des Widgets *innerhalb der grid-Zelle* aus. Links und rechts neben dem Button ‚Nochmal' sind jetzt 20 Pixel Platz.

Die Anweisung `pady` wirkt sich auf die *vertikale* Positionierung des Widgets *innerhalb der grid-Zelle* aus. Ober- und unterhalb vom Button sind jetzt 20 Pixel Platz.

Der Zusatz ‚i' in `ipadx` bedeutet ‚intern' und bezieht sich auf das Widget selber.

Zu sehen am Beispiel des Buttons ‚Auswertung', der ist jetzt rechts und links sowie unter- und oberhalb des Textes um 20 Pixel gewachsen.

Die Positionierung des Widgets innerhalb der `grid`-Zelle kann mit der Anweisung `sticky` und dem ersten Buchstaben der englischen Bezeichnung der Himmelsrichtung angegeben werden.

Also `sticky='w'` für ‚west' würde das Widget linksbündig einstellen, `sticky='e'` für ‚east' dementsprechend rechtsbündig. Zentriert ist `sticky='nswe'`

Am Beispiel der Zeile 1 hier der entsprechende Code-Ausschnitt:

```
6 …
7         # Zeile 1
8         self.label_bereit = tk.Label(self, text= 'Bereit? ')
9         self.label_bereit.grid(row=0, column=0, sticky='w', padx=5, ipady=15)
10        self.label_dannLos = tk.Label(self, text= 'dann los -> ')
11        self.label_dannLos.grid(row=0, column=1, sticky='e', padx=5)
12        self.button_start = ttk.Button(self, text= 'Start')
13        self.button_start.grid(row=0, column=2, sticky='e', padx=5)
14        # Zeile 2
15 …
```

Auch das erklärt sich von selbst, oder?

Damit die Proportionen stimmen, habe ich das gesamte Fenster von der Größe her noch einmal angepasst:

```
48 …
49 class App(tk.Tk):
50     def __init__(self):
51         super().__init__()
52 
53         self.title('Das kleine Einmaleins')
54         self.geometry('300x320')
55         self.resizable(False, False)
56 …
57 
```

In manchen Durchläufen habe ich bemerkt, dass das Fenster mit der Angabe von `300x320` zu klein war, warum auch immer sich das erst später ausgewirkt hat. Dann bin ich auf `360x380` gegangen.

Da ich auch die leeren Label gelöscht habe und auch noch Anpassungen an den Namen gemacht habe, hier zum Abschluss des Themas `grid()` noch einmal der gesamte bisher erstellte Code in voller Länge:

```
1  import tkinter as tk
2  from tkinter import ttk
3  
4  class Einmaleins(ttk.Frame):
5      def __init__(self, container):
6          super().__init__(container)
7          # Zeile 1
8          self.label_bereit = tk.Label(self, text= 'Bereit? ')
9          self.label_bereit.grid(row=0, column=0, sticky='w', padx=5, ipady=15)
10         self.label_dannLos = tk.Label(self, text= 'dann los -> ')
11         self.label_dannLos.grid(row=0, column=1, sticky='e', padx=5)
12         self.button_start = ttk.Button(self, text= 'Start')
13         self.button_start.grid(row=0, column=2, sticky='e', padx=5)
14         # Zeile 2
15         self.label_dieAufgabe = ttk.Label(self, text= 'Die Aufgabe: ')
16         self.label_dieAufgabe.grid(row=1, column=0, sticky='w', padx=5, ipady=15)
17         self.label_aufgabe = ttk.Label(self, text= '6 x 3')
18         self.label_aufgabe.grid(row=1, column=1, sticky='e', padx=5)
```

```
19          # Zeile 3
20          self.label_deineEingabe = tk.Label(self, text= 'Deine Eingabe: ')
21          self.label_deineEingabe.grid(row=2, column=0, sticky='w', padx=5, ipady=15)
22          self.eingabe = tk.StringVar()
23          self.eingabefeld = ttk.Entry(self, width=5, justify=tk.RIGHT, textvariable=self.eingabe)
24          self.eingabefeld.grid(row=2, column=1, sticky='e', padx=5)
25          self.button_los = ttk.Button(self, text= 'Los')
26          self.button_los.grid(row=2, column=2, sticky='e', padx=5)
27          # Zeile 4
28          self.label_deineEingabe = tk.Label(self, text= 'Dein Ergebnis: ')
29          self.label_deineEingabe.grid(row=3, column=0, sticky='w', padx=5, ipady=15)
30          self.label_deinErgebnis = ttk.Label(self, text= '16')
31          self.label_deinErgebnis.grid(row=3, column=1, sticky='e', padx=5)
32          # Zeile 5
33          self.label_richtigIst = tk.Label(self, text= 'richtig ist  : ')
34          self.label_richtigIst.grid(row=4, column=0, sticky='w', padx=5, ipady=15)
35          self.label_richtigesErgebnis = ttk.Label(self, text= '18')
36          self.label_richtigesErgebnis.grid(row=4, column=1, sticky='e', padx=5)
37          self.label_erfolg = ttk.Label(self, text= 'Äh, nö...')
38          self.label_erfolg.grid(row=4, column=2, sticky='e', padx=5)
39          # Zeile 6
40          self.button_nochmal = ttk.Button(self, text= 'Nochmal')
41          self.button_nochmal.grid(row=5, column=0, sticky='w', padx=5)
42          self.button_auswertung = ttk.Button(self, text= 'Auswertung')
43          self.button_auswertung.grid(row=5, column=1, sticky='e', padx=5)
44          self.button_verlassen = ttk.Button(self, text= 'Verlassen')
45          self.button_verlassen.grid(row=5, column=2, sticky='e', padx=5)
46          # alles zur Anzeige bringen
47          self.grid(padx=10, pady=10, sticky='nswe')
48
49  class App(tk.Tk):
50      def __init__(self):
51          super().__init__()
52
53          self.title('Das kleine Einmaleins')
54          self.geometry('300x320')
55          self.resizable(False, False)
56
57
58  if __name__ == "__main__":
59      app = App()
60      Einmaleins(app)
61      app.mainloop()
62
```

Als nächstes gehen wir die Programmierung der Button an.

4.3.4. Der Start-Button

Wir erweitern jetzt die Klasse `Einmaleins` um diverse Funktionen. Als erstes legen wir die Schritte des Start-Button fest.

Was sind die Arbeitsschritte?

Bei Klick auf den Start-Button

- lassen wir zunächst eine Aufgabe generieren
- diese Aufgabe wird dann in der App in Zeile 2 Spalte 2 angezeigt

Um das zu erreichen, müssen wir uns erst anschauen, wie wir eine Aufgabe generieren können.

Im letzten Tutorial haben wir mit dem Import von `random` für die Möglichkeit gesorgt, uns per Zufall die Zahl für die Länge einer Seite generieren zu lassen. Das machen wir jetzt wieder. Also erweitern wir den `import` um dieses Paket:

```
1  import tkinter as tk
2  from tkinter import ttk
3  from random import *
4
5  faktor_1 = 0
6  faktor_2 = 0
7
8  class Einmaleins(ttk.Frame):
9      def __init__(self, container):
10         super().__init__(container)
11         # Zeile 1
12         self.label_bereit = tk.Label(self, text= 'Bereit? ')
13         self.label_bereit.grid(row=0, column=0, sticky='w', padx=5, ipady=15)
14         self.label_dannLos = tk.Label(self, text= 'dann los -> ')
15         self.label_dannLos.grid(row=0, column=1, sticky='e', padx=5)
16         self.button_start = ttk.Button(self, text= 'Start', command=self.startenSpiel)
17         self.button_start.grid(row=0, column=2, sticky='e', padx=5)
18         # Zeile 2
19         self.label_dieAufgabe = ttk.Label(self, text= 'Die Aufgabe: ')
20         self.label_dieAufgabe.grid(row=1, column=0, sticky='w', padx=5, ipady=15)
21         self.label_aufgabe = ttk.Label(self, text= '')
22         self.label_aufgabe.grid(row=1, column=1, sticky='e', padx=5)
23         # Zeile 3
24 …
```

Für die Aufnahme der beiden Faktoren richten wir die beiden Variablen `faktor_1` und `faktor_2` ein (Zeilen 5 und 6).

In Zeile 16 geben wir dem Button das Kommando zum Aufruf der Funktion `startenSpiel()` mit.

Zeile 21 löschen wir den vordefinierten Text mit der Aufgabe, das wollen wir ja gleich richtig anzeigen.

Die Funktion `startenSpiel()` selbst deklarieren wir nach dem ganzen `grid`-Geraffel:

```
49 …
50          # alles zur Anzeige bringen
51          self.grid(padx=10, pady=10, sticky='nswe')
52
53      def generierenAufgabe(self):
54          print('in generierenAufgabe')
55          global faktor_1
56          faktor_1 = randint(3,9)
57          global faktor_2
58          faktor_2 = randint(3,9)
59          print('faktor_1 = ', faktor_1, 'faktor_2 = ', faktor_2)
60          aufgabe = (faktor_1, 'x', faktor_2)
61          self.label_aufgabe.config(text=aufgabe)
62
63      def startenSpiel(self):
64          print('in starten')
65          self.generierenAufgabe()
66
67
68  class App(tk.Tk):
69      def __init__(self):
70          super().__init__()
71  …
```

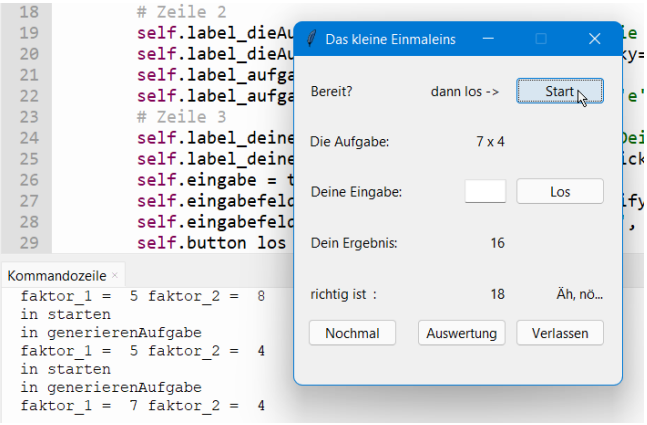
Wenn der Los-Button gedrückt wird, springt die Verarbeitung in Zeile 63. Dort wird die `print()`-Funktion aufgerufen (was absolute Geschmackssache ist. Ich mache das immer gerne am Anfang einer Funktion, dann kann ich den Programmfluss kontrollieren. Sobald ich sicher bin, dass das richtig läuft, nehme ich die Statements wieder raus).

Da wir das Generieren der Aufgabe auch später noch separat brauchen, habe ich den Teil gleich in eine eigene Funktion ab Zeile 53 ausgelagert.

Zeile 55 geben wir an, dass wir etwas mit der *globale* Variable `faktor_1` vorhaben. Was das ist kommt dann in Zeile 56, nämlich das zufällige Besorgen einer Zahl vom Typ Integer (also einer Ganzzahl) die im Wertenbereich zwischen 3 und 9 liegt. Würden wir hier nicht das Schlüsselwort `global` setzen, würde eine *lokale* Variable erzeugt werden. Wir müssen aber von überall darauf zugreifen können.

In den Zeilen 57 und 58 passiert das gleiche für `faktor_2`.

Das print-Statement in Zeile 59 soll uns dann die beiden Faktoren in der Kommandozeile anzeigen.



In Zeile 60 bauen wir die Aufgabe zusammen, die wir mit der `config()`-Funktion des `label`-Widgets in Zeile 61 ändern.

Prüfen wir das, indem wir die Änderungen laufen lassen.

Damit haben wir jetzt die Möglichkeit, wann immer wir auf den Start-Button klicken, eine neue Aufgabe zu generieren und diese anzuzeigen.

Somit sind die Anforderungen aus dem Anfang des Kapitels erfüllt.

4.3.5. Der Nochmal-Button

Was soll der Nochmal-Button bewirken? In der fachlichen Anforderung stand, dass das Spiel erneut gestartet werden können muss, ohne es zu schließen und dann erneut zu öffnen.

Demnach ist der Funktionsumfang ähnlich wie der des Start-Button. Allerdings sollen auch die bis dahin erreichten Punkte gelöscht werden. Also haben wir folgenden Ablauf:

- Setze die erreichten Punkte wieder auf 0
- Generiere eine neue Aufgabe
- Zeige diese in der App in Zeile 2 Spalte 2 an

Darin sind wir inzwischen Profis. Die Deklaration der beiden Zahlen kommt nach oben:

```
1  import tkinter as tk
2  from tkinter import ttk
3  from random import *
4
5  faktor_1 = 0
6  faktor_2 = 0
7  richtige_antworten = 0
8  falsche_antworten = 0
9
10 class Einmaleins(ttk.Frame):
11     def __init__(self, container):
12         super().__init__(container)
13 …
```

Die Verbindung zwischen Button und neuer Funktion erfolgt in der Deklaration des Buttons:

```
44 …
45         # Zeile 6
46         self.button_nochmal = ttk.Button(self, text= 'Nochmal', command=self.nochmalSpielen)
47         self.button_nochmal.grid(row=5, column=0, sticky='w', padx=5)
48 …
```

Und die Funktion selbst ist dann auch nicht mehr überraschend:

```
68 …
69     def nochmalSpielen(self):
70         print('in nochmalSpielen')
71         global richtige_antworten
72         richtige_antworten = 0
73         global falsche_antworten
74         falsche_antworten = 0
75         self.generierenAufgabe()
76
77 …
```

Auch der Code dieser Funktion folgt dem gleichen Muster wie der für den Start-Button. Und die Auslagerung für die Generierung der Aufgabe hat sich hier schon ausbezahlt.

Damit verhalten sich die beiden Button ‚Start' und ‚Nochmal' zumindest *optisch* gleich, bei Klick wird immer eine neue Aufgabe generiert, die dann auch angezeigt wird. Da wir bisher noch keine Punkte vergeben haben, sehen wir das Zurücksetzen noch nicht.

4.3.6. Der Auswertung-Button

Schauen wir zunächst wieder auf den Funktionsumfang.

In der fachlichen Anforderung stand, dass der Spieler die Möglichkeit haben soll die Anzahl der richtigen und falschen Antworten einzusehen.

Wir definieren dafür wieder eine eigene Funktion, der Aufruf kommt dann wieder an den Button:

```
44 …
45         # Zeile 6
46         self.button_nochmal = ttk.Button(self, text= 'Nochmal', command=self.nochmalSpielen)
47         self.button_nochmal.grid(row=5, column=0, sticky='w', padx=5)
48         self.button_auswertung = ttk.Button(self, text= 'Auswertung',
49                                             command=self.ergebnisAnzeigen)
50         self.button_auswertung.grid(row=5, column=1, sticky='e', padx=5)
51         self.button_verlassen = ttk.Button(self, text= 'Verlassen')
52         self.button_verlassen.grid(row=5, column=2, sticky='e', padx=5)
53 …
```

Die Funktion selber sieht so aus:

```
77 …
78     def ergebnisAnzeigen(self):
79         global richtige_antworten
80         str_richtige_Antworten = str(richtige_antworten)
81         global falsche_antworten
82         str_falsche_Antworten = str(falsche_antworten)
83         if richtige_antworten > falsche_antworten:
84             erfolg = 'Gut gemacht!'
85         else:
86             erfolg = 'Das kannst Du besser!'
87 
88         ausgabe = ('\n' + erfolg + '\n\n\nRichtige Antworten: '
89                    + str_richtige_Antworten + '\nFalsche Antworten: '
90                    + str_falsche_Antworten)
91         tk.messagebox.showinfo('Ergebnis', ausgabe)
92 …
```

Diesmal fange ich von unten an.

In Zeile 91 sehen wir den Aufbau zum Aufruf eines Pop-Up-Fensters. Es handelt sich in diesem Fall um ein Info-Fenster, es besteht aus zwei Teilen, der Überschrift ‚`Ergebnis`‘ vor dem Komma und der Variablen ‚`ausgabe`‘ nach dem Komma. Dies muss eine Variable vom Typ ‚`String`‘ sein.

Für die Messagebox brauchen wir noch einen Import:

```
from tkinter.messagebox import showinfo
```

Die Variable ‚`ausgabe`‘ wird in Zeile 88 zusammengebaut.

De Variable ‚`erfolg`‘ wird in Abhängigkeit zu den erreichten Punkten gesetzt. Dazu wird in Zeile 83 geprüft, ob die Anzahl der richtigen Antworten größer ist, als die der falschen Antworten. Trifft das zu, wird ein ‚`Gut gemacht!`‘ in Erfolg gespeichert, im anderen ein ‚`das kannst Du besser!`‘.

Für die Ausgabe der Anzahl der Antworten müssen wir diese zuerst in einen String umwandeln, das nennt sich ‚`casten`‘ und erfolgt in den Zeilen 80 und 82.

Das war es mit dem Auswertung-Button.

4.3.7. Der Verlassen-Button

Die Behandlung geht ganz schnell. Hier ist nur der `command`-Befehl am Button zu platzieren:

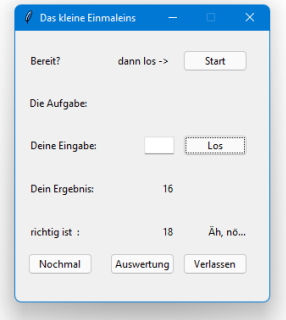
```
44 …
45          # Zeile 6
46          self.button_nochmal = ttk.Button(self, text= 'Nochmal', command=self.nochmalSpielen)
47          self.button_nochmal.grid(row=5, column=0, sticky='w', padx=5)
48          self.button_auswertung = ttk.Button(self, text= 'Auswertung',
49                                              command=self.ergebnisAnzeigen)
50          self.button_auswertung.grid(row=5, column=1, sticky='e', padx=5)
51          self.button_verlassen = ttk.Button(self, text= 'Verlassen', command=app.destroy)
52          self.button_verlassen.grid(row=5, column=2, sticky='e', padx=5)
53 …
```

Cool, oder?

4.3.8. Der Los-Button

Das Beste zum Schluss, der Los-Button hat es wirklich in sich.

Auch hier wieder zuerst der Blick auf den Funktionsumfang. Schauen wir und das bisher erreichte Ergebnis noch einmal an:



Bei Klick auf den Los-Button

- erfolgt das Lesen der Eingabe
- es wird geprüft, ob die Eingabe eventuell leer ist, oder etwas anderes als eine Ganzzahl enthält
    - Ist das der Fall erfolgt die Ausgabe einer Fehlermeldung
- Ist die Eingabe eine Ganzzahl
    - wird die Zahl in der ‚`Dein Ergebnis`'-Zeile 4 in Spalte 2 übernommen
    - es erfolgt die Berechnung des Produkts
    - das Produkt wird in Zeile 5 Spalte 2 angezeigt
    - Eingabe und Produkt werden verglichen
    - Bei Übereinstimmung
        - wird in Zeile 5 Spalte 3 ‚`Prima!`' angezeigt
        - die Zahl in `richtige_antworten` um 1 erhöht
    - gibt es keine Übereinstimmung
        - wird in Zeile 5 Spalte 3 ‚`Äh, nö…!`' angezeigt
        - die Zahl in `falsche_antworten` um 1 erhöht
- eine neue Aufgabe wird generiert
- das Eingabefeld wird geleert

Nicht wenig, für einen einzigen Button, oder? Gehen wir es an.

Auch hier bauen wir uns wieder eine neue Funktion innerhalb der Klasse `Einmaleins()`. Der Aufruf der Funktion geht wieder wie gewohnt im dazugehörigen `grid()`-Statement:

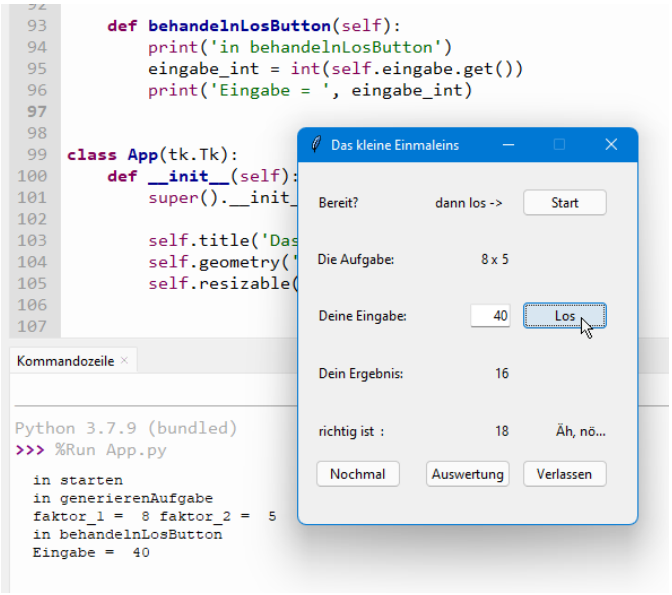
```
24 …
25          # Zeile 3
26          self.label_deineEingabe = tk.Label(self, text= 'Deine Eingabe: ')
27          self.label_deineEingabe.grid(row=2, column=0, sticky='w', padx=5, ipady=15)
28          self.eingabe = tk.StringVar()
29          self.eingabefeld = ttk.Entry(self, width=5, justify=tk.RIGHT, textvariable=self.eingabe)
30          self.eingabefeld.grid(row=2, column=1, sticky='e', padx=5)
31          self.button_los = ttk.Button(self, text= 'Los', command=self.behandelnLosButton)
32          self.button_los.grid(row=2, column=2, sticky='e', padx=5)
33 …
```

Bei der Gelegenheit fällt mir auf, dass ich über die Zeile 29 noch ein paar Worte verlieren sollte. Die Anpassung der Größe des `Entry`-Widgets erfolgt über die `width`-Anweisung, mit `justify=tk.RIGHT` richten wir die Eingabe rechtsbündig aus.

Das Einlesen der Eingabe erfolgt mit der `get()`-Funktion des Entry-Widges

```
eingabe_int = int(self.eingabe.get())
```

Die Eingabe ist ein String, wir brauchen eine Zahl, also müssen wir das erst einmal wieder ‚`casten`', das erfolgt mit der `int()`-Funktion. Das Ergebnis ist dann die Integer-Variable ‚`eingabe_int`'



Der zweite Punkt war die Prüfung der Eingabe auf eine Zahl.

Da wir versuchen, ein Integer zu befüllen, gibt es bei jedem anderen Wert eine Fehlermeldung. Diese ‚exception' können wir ‚fangen', indem wir das `casting` in einen `try`-Block legen.

Die Ausgabe erfolgt dann wieder in einem Pop-Up-Fenster.

Der Code dafür sieht bei mir so aus:

```
 92 …
 93     def behandelnLosButton(self):
 94         print('in behandelnLosButton')
 95         try:
 96             eingabe_int = int(self.eingabe.get())
 97             print('Eingabe = ', eingabe_int)
 98 
 99         except ValueError as error:
100             eingabe_str = self.eingabe.get()
101             if len(eingabe_str) == 0:
102                 fehlertext = ('Du musst schon eine Zahl eingeben...')
103             else:
104                 fehlertext = (eingabe_str + ' ist keine ganze Zahl, oder?')
105                 self.eingabefeld.delete('0', tk.END)
106 
107             tk.messagebox.showerror('Fehler', message=fehlertext)
108 …
```

Es wird also *versucht*, die Eingabe in einen Integer-Wert zu casten. Nur, wenn auch eine ganze Zahl mitgegeben wurde, werden die Anweisungen im `try`-Block ausgeführt. Auch die Eingabe von zum Beispiel ‚2,45' würde zu einem Fehler führen, da diese in eine Variable vom Typ `float` gecastet werden müsste.

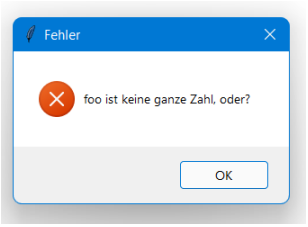
Die Unterscheidung, ob es sich im Fehlerfall um eine leere Eingabe oder etwas anderes handelt, können wir mit der Abfrage auf die Länge des Feldes prüfen. In Zeile 101 ist das zu sehen.

In Zeile 105 wird das Eingabefeld geleert, die Ausgabe der Messagebox erfolgt dann in Zeile 107.

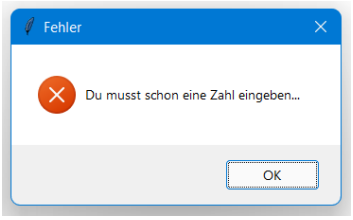
Für die Messagebox brauchen wir noch einen Import:

```
from tkinter.messagebox import showinfo, showerror
```

Das Ergebnis ist dann:



bzw.



Der nächste Punkt ist die Anzeige der Eingabe in Zeile 3, Spalte 2. Das kennen wir schon aus der Anzeige der Aufgabe in der `generierenAufgabe()`-Funktion und geht so:

```
self.label_deinErgebnis.config(text=eingabe_int)
```

Wenden wir uns dem Produkt zu. Die Anweisung ist denkbar einfach:

```
produkt = faktor_1 * faktor_2
```

Das Produkt zeigen wir dann in der Zeile 5 Spalte 2 an

```
self.label_richtigesErgebnis.config(text=produkt)
```

Folgt der Vergleich von Eingabe mit Produkt. Dafür kennen wir das if-else Konstrukt:

```
if eingabe_int == produkt:
```

Der Rest ist dann eigentlich Fleißarbeit.

Bei mir sieht der fertige Code für die Funktion `behandelnLosButton()` dann so aus:

```
 92 …
 93     def behandelnLosButton(self):
 94         print('in behandelnLosButton')
 95         try:
 96             eingabe_int = int(self.eingabe.get())
 97             print('Eingabe = ', eingabe_int)
 98             self.label_deinErgebnis.config(text=eingabe_int)
 99             produkt = faktor_1 * faktor_2
100             print('Produkt = ', produkt)
101             self.label_richtigesErgebnis.config(text=produkt)
102             if eingabe_int == produkt:
103                 global richtige_antworten
104                 richtige_antworten += 1
105                 self.label_erfolg.config(text='Richtig!')
106                 self.eingabefeld.delete('0', tk.END)
107             else:
108                 global falsche_antworten
109                 falsche_antworten += 1
110                 self.label_erfolg.config(text='Äh, nö...')
111                 self.eingabefeld.delete('0', tk.END)
112 
113             self.eingabefeld.delete('0', tk.END)
114             self.generierenAufgabe()
115 
116         except ValueError as error:
117             eingabe_str = self.eingabe.get()
118             if len(eingabe_str) == 0:
119                 fehlertext = ('Du musst schon eine Zahl eingeben...')
120             else:
121                 fehlertext = (eingabe_str + ' ist keine ganze Zahl, oder?')
122                 self.eingabefeld.delete('0', tk.END)
123 
124             tk.messagebox.showerror('Fehler', message=fehlertext)
125 …
```

Das ist soweit alles verständlich, nehme ich an. Wirklich neuer Code ist da auch nicht drin, das ist alles schon bekannte Materie.

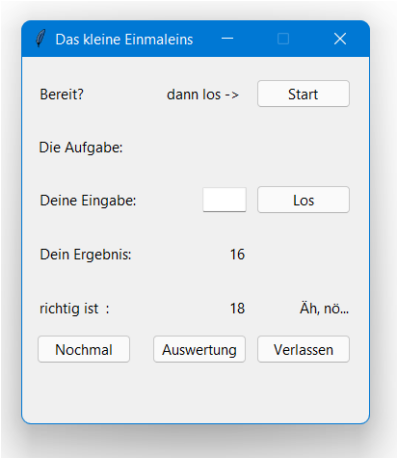
Damit haben wir auch für den Los-Button alle Anforderungen erfüllt, das Programm sollte jetzt funktionieren und darf auch bei Fehlerfassung und wildem ‚Rumgeklicke' nicht abstürzen.

Wenn Ihr sicher seid, dass alles funktioniert, könnt Ihr den Aufruf der `print()`-Funktionen wieder entfernen.

Den letzten Schliff holen wir uns dann jetzt ab.

4.3.9. Der letzte Schliff

Schauen wir unser bisher erreichtes Ergebnis noch einmal an:



Sieht schon recht hübsch aus.

Als erste Verbesserung nehmen wir mal die Fixwerte ‚`16`', ‚`18`' und ‚`Äh, nö…`' vor.

Für die beiden Zahlen können wir die Textzuweisung `text= '16'` einfach rausnehmen, oder nur die Zahl eliminieren, also `text= ''` schreiben, wie es Euch gefällt. Ich habe mal beide Varianten eingebaut.

‚`Äh, nö…`' ersetzen wir durch ‚`Viel Spaß!`'.

Für die Behandlung von Feldern und Button haben wir die Möglichkeit, diese zu sperren oder zu entsperren, das können wir hier nutzen.

Bei Programmstart soll nur der Start- und der Verlassen-Button offen sein, alle anderen Button und die Eingabe werden gesperrt.

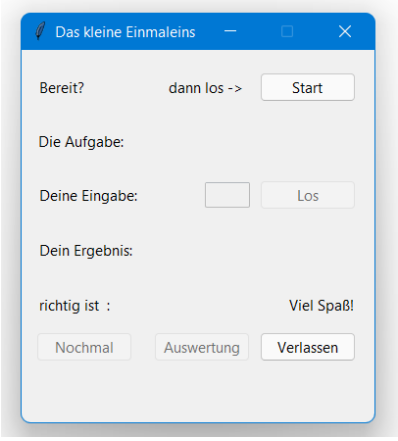
Das *Sperren* erfolgt über die Funktion `config(state='disable')` **nach(!)** der Deklaration des Widgets aber **vor(!)** der Behandlung mit `grid()`. Am Beispiel des Los-Buttons wäre das

```
24  …
25          # Zeile 3
31          …
32          self.button_los = ttk.Button(self, text= 'Los', command=self.behandelnLosButton)
33          self.button_los.config(state='disable')
34          self.button_los.grid(row=2, column=2, sticky='e', padx=5)
35  …
```

Das *Entsperren* erfolgt über die gleiche Funktion, allerdings – wer hätte es gedacht - mit dem Aufruf

```
config(state='enable')
```

Nachdem wir alles gesperrt haben, sollte das Programm bei Start so aussehen:



Sehr gut, alles ist dicht, der Spieler kann keine Fehler machen, er hat nur die Auswahl zwischen ‚`Start`' und ‚`Verlassen`'.

Der Klick auf ‚`Verlassen`' bewirkt keine Änderung im Programm, da bleibt also alles wie es ist.

Bei Klick auf den Start-Button müssen wir jetzt allerdings alles aufmachen. Und da wir ja eine eigene Funktion `startenSpiel()` haben, ist das der Platz für die Erweiterung.

Den Start-Button deaktivieren wir allerdings, für das erneute Spielen haben wir ja den Nochmal-Button.

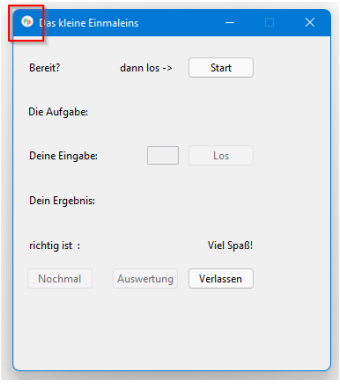
Nach der Anpassung sieht meine `startenSpiel()`-Funktion dann so aus:

```
69 …
70      def startenSpiel(self):
71          print('in starten')
72          self.generierenAufgabe()
73          self.eingabefeld.config(state='enable')
74          self.button_los.config(state='enable')
75          self.button_nochmal.config(state='enable')
76          self.button_auswertung.config(state='enable')
77          self.button_start.config(state='disable')
78
79 …
```

Bei der Verarbeitung des Nochmal-Buttons können wir auch noch eine Verbesserung machen. Wenn der Spieler den Button klickt, sollen natürlich auch das angezeigte Ergebnis und das Produkt gelöscht werden. Das geht wieder über die `config()`-Funktion.

```
75 …
76      def nochmalSpielen(self):
77          global richtige_antworten
78          richtige_antworten = 0
79          global falsche_antworten
80          falsche_antworten = 0
81          self.generierenAufgabe()
82          self.label_deinErgebnis.config(text='')
83          self.label_richtigesErgebnis.config(text='')
84          self.label_erfolg.config(text='')
85 …
```

Einen habe ich noch…



Oben links ist die Stelle, an die ein Icon gesetzt werden kann.

Ich habe das von meiner Homepage genommen und in den Ordner kopiert, in dem auch die Datei App.py liegt.

Pp_.ico

In meinem Fall also ‚D:\work_python\Thonny\Pp_.ico'. Der Befehl zum Einbinden ist auch denkbar einfach und kommt in die Klasse `App()`:

```
130 …
131 class App(tk.Tk):
132     def __init__(self):
133         super().__init__()
134
135         self.title('Das kleine Einmaleins')
136         self.geometry('360x380')
137         self.iconbitmap('d:/work_python/Thonny/Pp_.ico')
138         self.resizable(False, False)
139 …
```

Das war es von meiner Seite, mehr Verbesserungen habe ich nicht. Ich denke, man könnte vielleicht den Hintergrund noch ändern, die Schriftart und Größe, das überlasse ich aber gerne Euch.

4.3.10. Der letzte Stand

An dieser Stelle noch einmal der gesamte Code ‚am Stück'.

```
 1 import tkinter as tk
 2 from tkinter import ttk
 3 from random import *
 4 from tkinter.messagebox import showinfo, showerror
 5
 6 faktor_1 = 0
 7 faktor_2 = 0
 8 richtige_antworten = 0
 9 falsche_antworten = 0
10
11 class Einmaleins(ttk.Frame):
12     def __init__(self, container):
13         super().__init__(container)
14         # Zeile 1
15         self.label_bereit = tk.Label(self, text= 'Bereit? ')
16         self.label_bereit.grid(row=0, column=0, sticky='w', padx=5, ipady=15)
17         self.label_dannLos = tk.Label(self, text= 'dann los -> ')
18         self.label_dannLos.grid(row=0, column=1, sticky='e', padx=5)
19         self.button_start = ttk.Button(self, text= 'Start', command=self.startenSpiel)
20         self.button_start.grid(row=0, column=2, sticky='e', padx=5)
21         # Zeile 2
22         self.label_dieAufgabe = ttk.Label(self, text= 'Die Aufgabe: ')
23         self.label_dieAufgabe.grid(row=1, column=0, sticky='w', padx=5, ipady=15)
24         self.label_aufgabe = ttk.Label(self, text= '')
25         self.label_aufgabe.grid(row=1, column=1, sticky='e', padx=5)
26         # Zeile 3
27         self.label_deineEingabe = tk.Label(self, text= 'Deine Eingabe: ')
28         self.label_deineEingabe.grid(row=2, column=0, sticky='w', padx=5, ipady=15)
29         self.eingabe = tk.StringVar()
30         self.eingabefeld = ttk.Entry(self, width=5, justify=tk.RIGHT, textvariable=self.eingabe)
31         self.eingabefeld.config(state='disable')
32         self.eingabefeld.grid(row=2, column=1, sticky='e', padx=5)
33         self.button_los = ttk.Button(self, text= 'Los', command=self.behandelnLosButton)
34         self.button_los.config(state='disable')
35         self.button_los.grid(row=2, column=2, sticky='e', padx=5)
36         # Zeile 4
37         self.label_deineEingabe = tk.Label(self, text= 'Dein Ergebnis: ')
38         self.label_deineEingabe.grid(row=3, column=0, sticky='w', padx=5, ipady=15)
39         self.label_deinErgebnis = ttk.Label(self, text= '')
40         self.label_deinErgebnis.grid(row=3, column=1, sticky='e', padx=5)
41         # Zeile 5
42         self.label_richtigIst = tk.Label(self, text= 'richtig ist  : ')
43         self.label_richtigIst.grid(row=4, column=0, sticky='w', padx=5, ipady=15)
44         self.label_richtigesErgebnis = ttk.Label(self)
45         self.label_richtigesErgebnis.grid(row=4, column=1, sticky='e', padx=5)
46         self.label_erfolg = ttk.Label(self, text= 'Viel Spaß!')
47         self.label_erfolg.grid(row=4, column=2, sticky='e', padx=5)
48         # Zeile 6
49         self.button_nochmal = ttk.Button(self, text= 'Nochmal', command=self.nochmalSpielen)
50         self.button_nochmal.config(state='disable')
51         self.button_nochmal.grid(row=5, column=0, sticky='w', padx=5)
52         self.button_auswertung = ttk.Button(self, text= 'Auswertung',
```

```
53                                          command=self.ergebnisAnzeigen)
54          self.button_auswertung.grid(row=5, column=1, sticky='e', padx=5)
55          self.button_auswertung.config(state='disable')
56          self.button_verlassen = ttk.Button(self, text= 'Verlassen', command=app.destroy)
57          self.button_verlassen.grid(row=5, column=2, sticky='e', padx=5)
58          # alles zur Anzeige bringen
59          self.grid(padx=10, pady=10, sticky='nswe')
60
61      def generierenAufgabe(self):
62          global faktor_1
63          faktor_1 = randint(3,9)
64          global faktor_2
65          faktor_2 = randint(3,9)
66          aufgabe = (faktor_1, 'x', faktor_2)
67          self.label_aufgabe.config(text=aufgabe)
68
69      def startenSpiel(self):
70          self.generierenAufgabe()
71          self.eingabefeld.config(state='enable')
72          self.button_los.config(state='enable')
73          self.button_nochmal.config(state='enable')
74          self.button_auswertung.config(state='enable')
75          self.button_start.config(state='disable')
76
77      def nochmalSpielen(self):
78          global richtige_antworten
79          richtige_antworten = 0
80          global falsche_antworten
81          falsche_antworten = 0
82          self.generierenAufgabe()
83          self.label_deinErgebnis.config(text='')
84          self.label_richtigesErgebnis.config(text='')
85          self.label_erfolg.config(text='')
86
87      def ergebnisAnzeigen(self):
88          global richtige_antworten
89          str_richtige_Antworten = str(richtige_antworten)
90          global falsche_antworten
91          str_falsche_Antworten = str(falsche_antworten)
92          if richtige_antworten > falsche_antworten:
93              erfolg = 'Gut gemacht!'
94          else:
95              erfolg = 'Das kannst Du besser!'
96
97          ausgabe = ('\n' + erfolg + '\n\n\nRichtige Antworten: '
98                     + str_richtige_Antworten + '\nFalsche Antworten: '
99                     + str_falsche_Antworten)
100         tk.messagebox.showinfo('Ergebnis', ausgabe)
101
102     def behandelnLosButton(self):
103         try:
104             eingabe_int = int(self.eingabe.get())
105             self.label_deinErgebnis.config(text=eingabe_int)
106             produkt = faktor_1 * faktor_2
107             self.label_richtigesErgebnis.config(text=produkt)
108             if eingabe_int == produkt:
109                 global richtige_antworten
```

```
110                 richtige_antworten += 1
111                 self.label_erfolg.config(text='Richtig!')
112                 self.eingabefeld.delete('0', tk.END)
113             else:
114                 global falsche_antworten
115                 falsche_antworten += 1
116                 self.label_erfolg.config(text='Äh, nö...')
117                 self.eingabefeld.delete('0', tk.END)
118 
119             self.eingabefeld.delete('0', tk.END)
120             self.generierenAufgabe()
121 
122         except ValueError as error:
123             eingabe_str = self.eingabe.get()
124             if len(eingabe_str) == 0:
125                 fehlertext = ('Du musst schon eine Zahl eingeben...')
126             else:
127                 fehlertext = (eingabe_str + ' ist keine ganze Zahl, oder?')
128                 self.eingabefeld.delete('0', tk.END)
129 
130             tk.messagebox.showerror('Fehler', message=fehlertext)
131 
132 class App(tk.Tk):
133     def __init__(self):
134         super().__init__()
135 
136         self.title('Das kleine Einmaleins')
137         self.geometry('300x360')
138         self.iconbitmap('d:/work_python/Thonny/Pp_.ico' )
139         self.resizable(False, False)
140 
141 if __name__ == "__main__":
142     app = App()
143     Einmaleins(app)
144     app.mainloop()
145 
146     Einmaleins(app)
147     app.mainloop()
148 
```

Keine 150 Zeilen, das ist ein tolles Ergebnis!

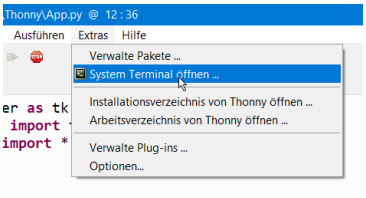
4.4. Die App als eigenständige Anwendung

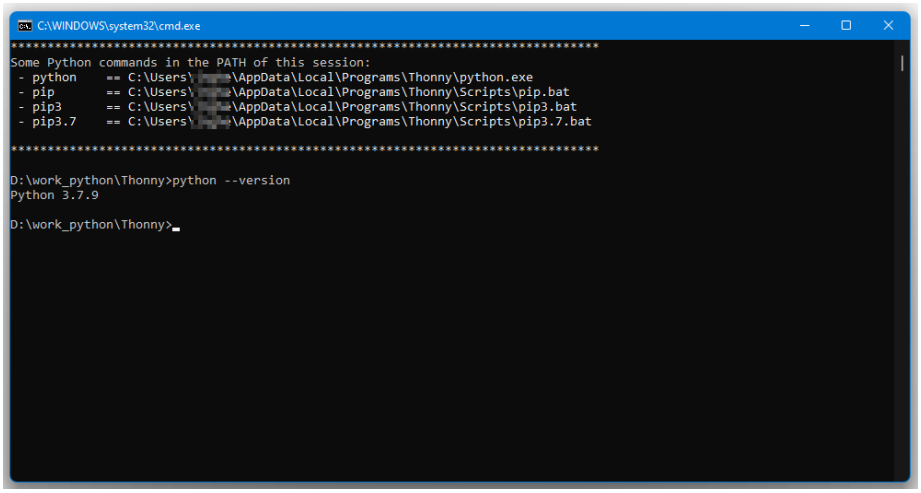
Zum Schluss noch eine letzte Sache – mit relativ wenig Aufwand ist es möglich, die App als unter Windows ausführbare Datei (‚`App.exe`‘) zu erstellen. Die kann man dann mit Doppelklick öffnen, ohne dass man eine Python-Umgebung oder IDE bräuchte.

Das ist nicht der professionellste Weg, aber darum geht es ja hier auch nicht. Wir sind immer noch Anfänger und versuchen uns von Aufgabe zu Aufgabe zu steigern.

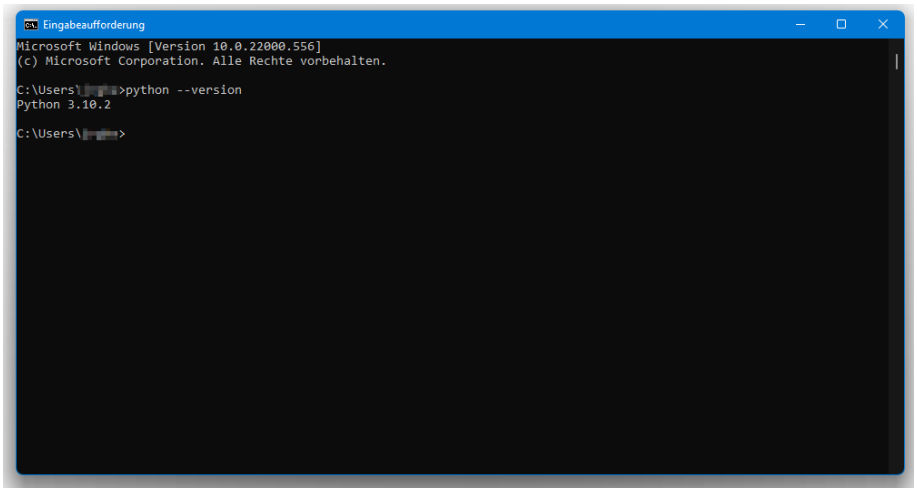
Wie geht das jetzt?

Wir brauchen dazu das Paket/Modul ‚`pyinstaller`‘. Das können wir uns relativ bequem in ‚Thonny‘ über den integrierten Terminal besorgen. Der Aufruf erfolgt über ‚Extras/System Terminal öffnen …‘:





Geöffnet wird dadurch ein Terminal-Fenster ähnlich wie die Eingabeaufforderung (`cmd`) unter Windows. Man sieht sehr schön was ‚bundle‘ bedeutet. ‚Thonny‘ arbeitet mit der Python Version 3.7.2. Das ist allerdings nicht die Version, die wir uns aus dem Internet gezogen haben, das ist 3.10.2:



Wir befinden uns also in einer eigenen, nicht für den ganzen Rechner geltenden virtuellen Umgebung.

Das ist die Arbeitsweise von Python, für jedes Projekt kann eine eigene Umgebung eingestellt oder ausgewählt werden. Die Konfiguration ist aber Experten-Modus, das machen wir später mal.

Was hat das jetzt für uns zu bedeuten?

- Für das Erstellen einer exe-Datei benötigen wir wie beschrieben das Paket ‚`pyinstaller`'
- Die Paketinstallation unter ‚Thonny' erfolgt über das Modul ‚pip'
- ‚pip' ist ein Tool zur Verwaltung von Paketen, also Installation und Deinstallation aber auch für Upgrades
- ‚pip' wird über das Terminal-Fenster aufgerufen

Der Aufruf für die Installation von `pyinstaller` ist

```
pip install pyinstaller
```

Tipp: die 3 Worte kopieren, ins Terminal-Fenster wechseln und einfach die linke Maustaste drücken. Cool, oder?

Mit Enter bestätigen, dann läuft die Installation. Bei mir kommt die Meldung:

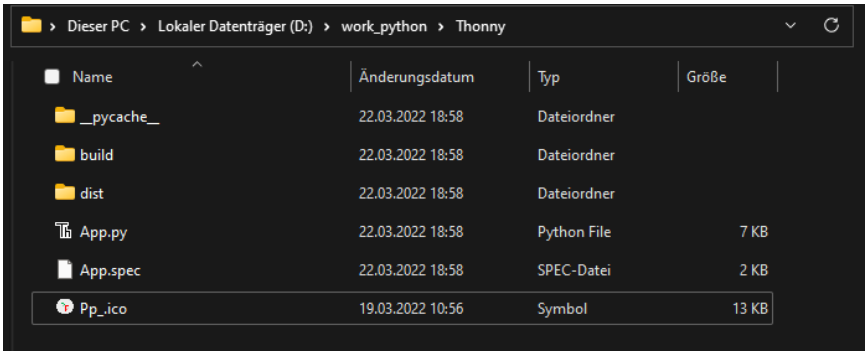
```
D:\work_python\Thonny>pip install pyinstaller
Requirement already satisfied: pyinstaller in c:\users\jrghe\appdata\local\programs\thonny\lib\site-packages (4.10)
Requirement already satisfied: setuptools in c:\users\jrghe\appdata\local\programs\thonny\lib\site-packages (from pyinstaller) (47.1.0)
```

Die Meldung im Fenster sagt, dass pyinstaller bereits installiert ist, das hätten wir also nicht gebraucht. Aber sicher ist sicher…

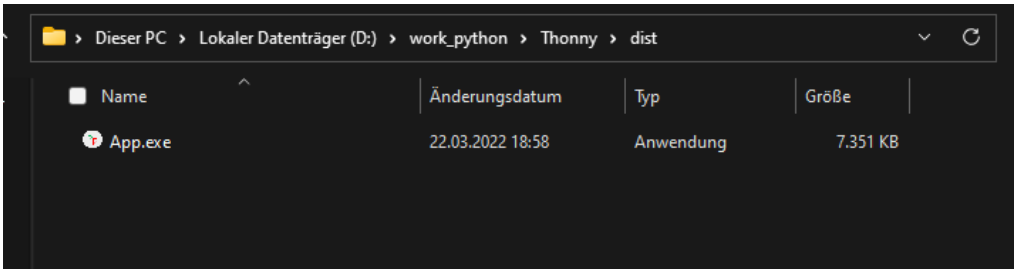
Jetzt passiert die Magie. Mit folgendem Befehl kreieren wir eine App.exe:

```
pyinstaller.exe --onefile -w --icon=Pp_.ico App.py
```

Das dauert jetzt etwas, am Ende hat das Tool ganz viele Ordner und Dateien erzeugt, das sieht man, wenn man in den Explorer schaut



Im Ordner /`dist` findet Ihr dann auch die exe:



Mit der exe könnt Ihr jetzt machen, was Ihr wollt, den Rest der erzeugten Dateien und Ordner braucht Ihr nicht, das könnt Ihr alles löschen.

5. Abschluss

So, damit bin ich auch schon wieder am Ende des Projekts angelangt. Ich hoffe, es hat Euch ebenso viel Spaß gemacht wie mir.

Was gibt es noch zu tun?

Von meiner Seite habe ich keine Anforderungen mehr an das Programm, ich möchte es so schmucklos lassen, wie es ist. Sicher kann man aber noch mit den Schriftarten und -größen, aber auch dem Hintergrund experimentieren, Button anders anordnen, und so weiter, das überlasse ich aber alles Eurer Phantasie.

Neben `Tkinter` gibt es ja auch noch andere GUI-Programme, vielleicht kann man zu Übungszwecken mal eine Portierung nach `Qt` oder `Kivy` ausprobieren.

Ansonsten gilt wie immer, viel Spaß beim Nachbauen und wenn Euch was Tolles einfallen sollte, was man umsetzen könnte, schickt mir gerne eine Mail an papa@papa-programmiert.de, ich würde mich sehr freuen, genau wie auch über eine Rückmeldung zu diesem Dokumet.

Viel Spaß weiterhin beim Coden, bleibt neugierig und hartnäckig.

Viele Grüße, Papa