Inhaltsverzeichnis

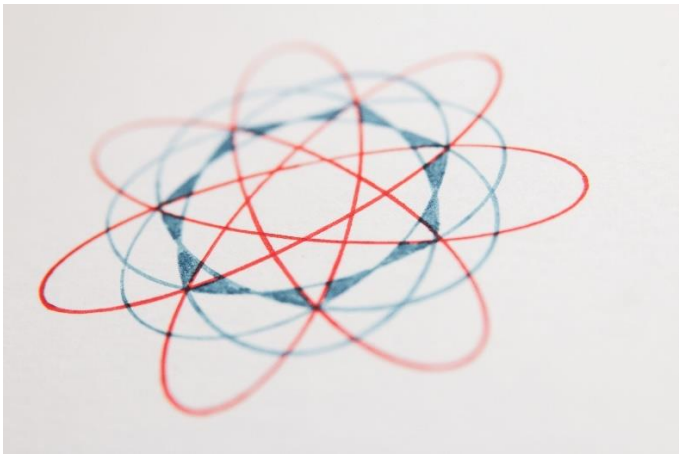
1. Vorwort

Seit längerem trage ich mich mit der Absicht, mich in Python einzuarbeiten. Das hat keinen speziellen Grund, es reizt mich einfach, eine neue Sprache auszuprobieren. Von Python hatte ich gehört, aber keine konkreten Ideen, was man damit machen kann.

Nachdem ich mir viele Video-Tutorials angeschaut habe, glaube ich jetzt soweit zu sein, dass ich die ersten Stolperer und Gehversuche aufschreiben kann, ohne mich als komplett Ahnungslosen aussehen zu lassen.

Um mit einer neuen Sprache zu beginnen, ist es nach meiner Erfahrung am besten, erst einmal „drauflos zu hacken“, um die Grundzüge der Syntax zu verstehen und sie anzuwenden. Wenn man große Projekte macht, helfen einem selbstverständlich die fundamentalen Software Architektur Konzepte wie Model-View-Controller, am Anfang vernachlässige ich das bewusst, die Dateien sind ja nur wenige Zeilen lang.

Einer meiner Neffen hat riesigen Spaß daran, Bilder mit dem Spirographen zu malen. Das sind verschiedene Zeichenschablonen, die mit Zahnrädern ineinandergreifen und mit denen man dann Formen und zeichnen kann, Ihr kennt das sicherlich, das sieht dann in etwa so aus:



Das Bild habe ich via unsplasch.com von Glen Carrie bezogen.

Solche Bilder kann man auch mit Python erstellen, und das will ich hier machen.

Wie in jedem Projekt, bitte ich Euch auch hier, falls mir Fehler unterlaufen sein sollten, bitte teilt es mir via Mail mit, eine falsche Dokumentation ist für mich schlimmer, als keine Dokumentation.

Was gibt es sonst noch zu sagen?

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 hier primär darum, die Grundzüge der Syntax zu verstehen und sie anzuwenden.

Wir werden also Python installieren und die ersten Erfahrungen mit der Sprache machen. Das Zeichnen übernimmt das package `turtle` für uns, auch das schauen wir uns an.

Wir werden Bilder erstellen, die denen aus dem Spirographen ähneln und werden im Anschluss „das Haus des Nikolaus“ ausgeben lassen.

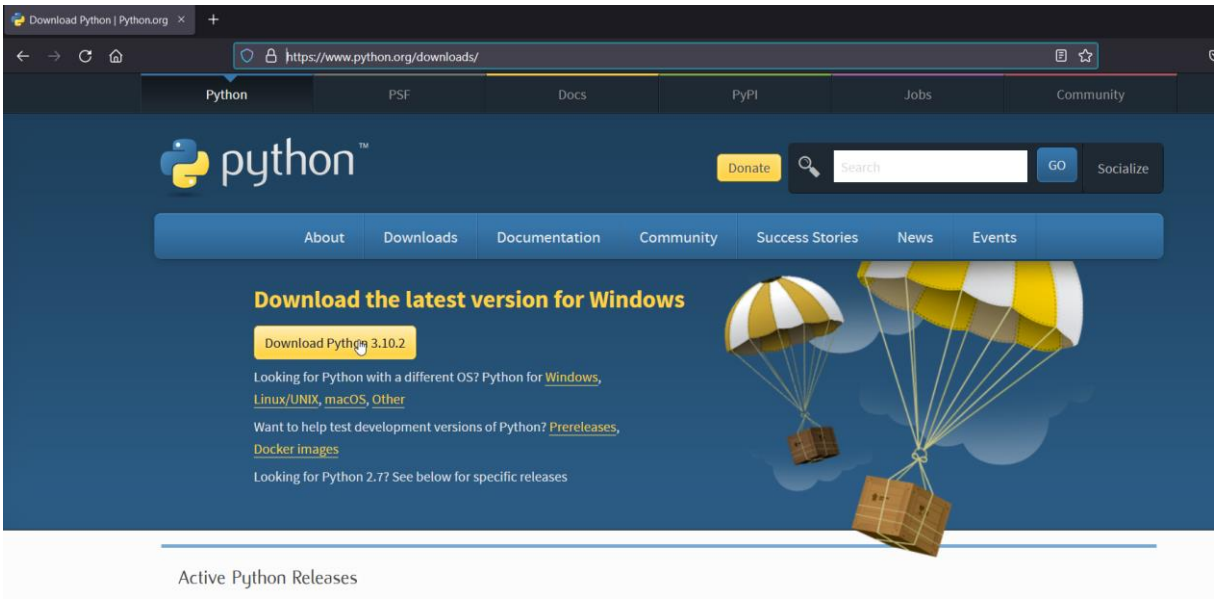
Die ersten Aktivitäten erfolgen in der mitgelieferten IDE „IDLE“ später werde ich mich nach einer anderen IDE umsehen. Für den Anfang taugt IDLE aber.

Bis sich die Gemeinschaft der Open-Sourcler in 2002 an die Entwicklung eines freien Compilers gemacht hat, wäre es und nicht möglich gewesen, dieses Projekt hier zu starten. Daher haben wir bei den Vorarbeiten noch einiges zu tun, aber das kriegen wir hin. Auf geht’s!

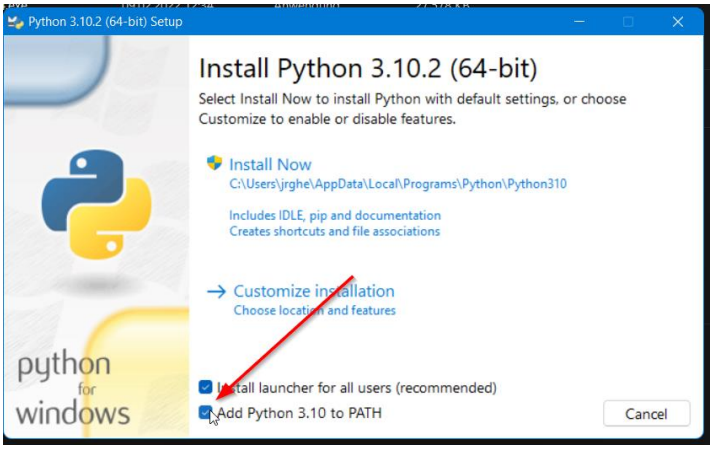
3. Vorarbeiten

Dieses Kapitel wird relativ kurz, dann was brauchen, ist bei der Installation von Python schon mit an Bord.

Wir gehen also auf https://www.python.org/downloads/ und holen uns – in meinem Fall für Windows, die neueste Verion:



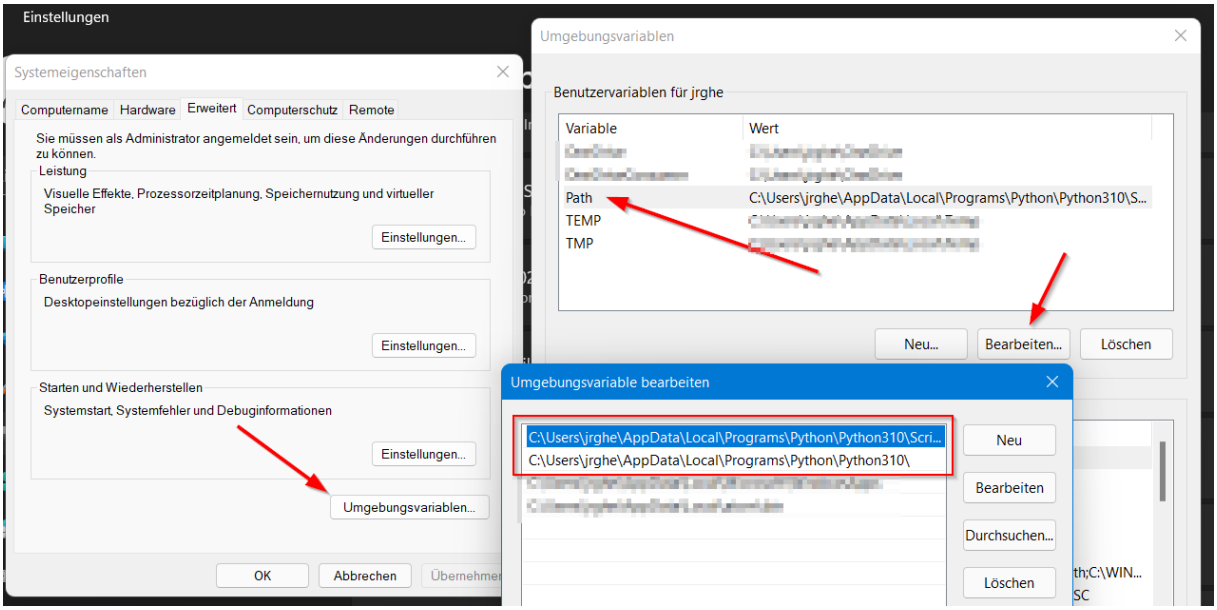
Zum Zeitpunkt meiner Installation ist der Version 3.10.2 aktuell. Nach dem Download den Installer ausführen – hier bitte aufpassen! Wichtig ist, dass das Käschen ganz unten („Add .. to PATH“) angeklickt ist:



Damit wird die Umgebungsvariable in den Pfad eingetragen. Andernfalls muss das von Hand geschehen, sonst ist Python nicht nutzbar.

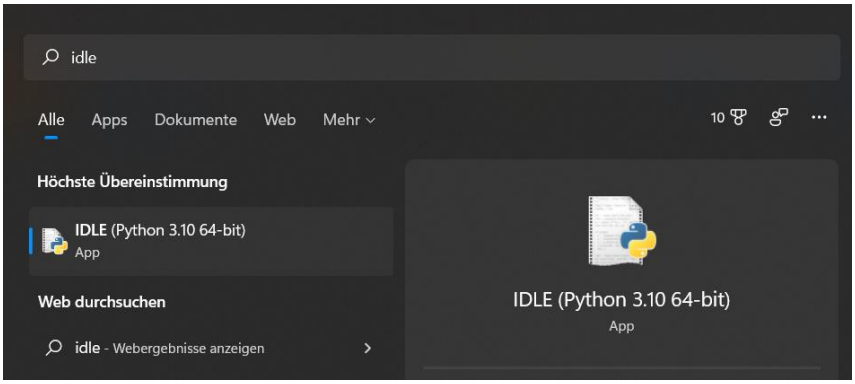
Nach der Installation können wir das in der Systemeinstellung, dort nach „Variabklen“ suchen und dann „Systemumgebungsvariablen“ auswählen.

Die Anzeige sollte dann so aussehen:

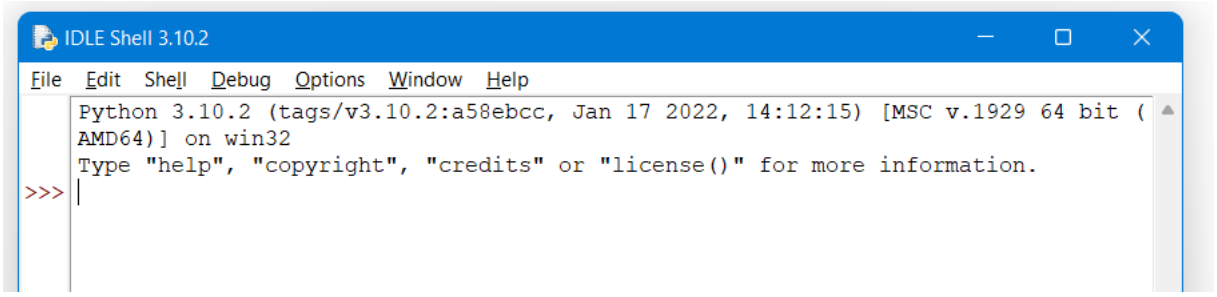


Das war es dann auch schon mit der Installation.

Zum Testen suchen wir in der Startleiste nach „IDLE:

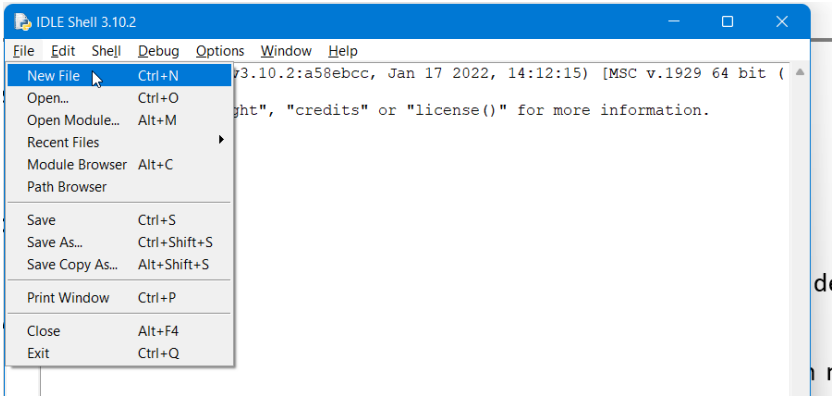


Damit sollte sich ein Fenster üffnen:

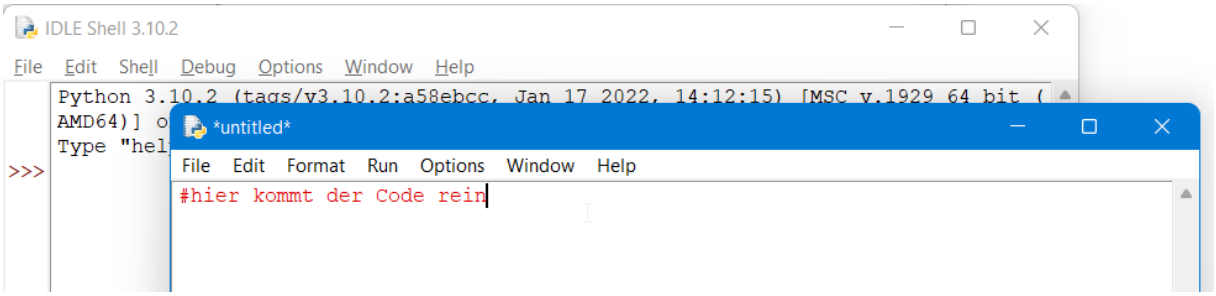


Das ist eine Shell ähnlich cmd.

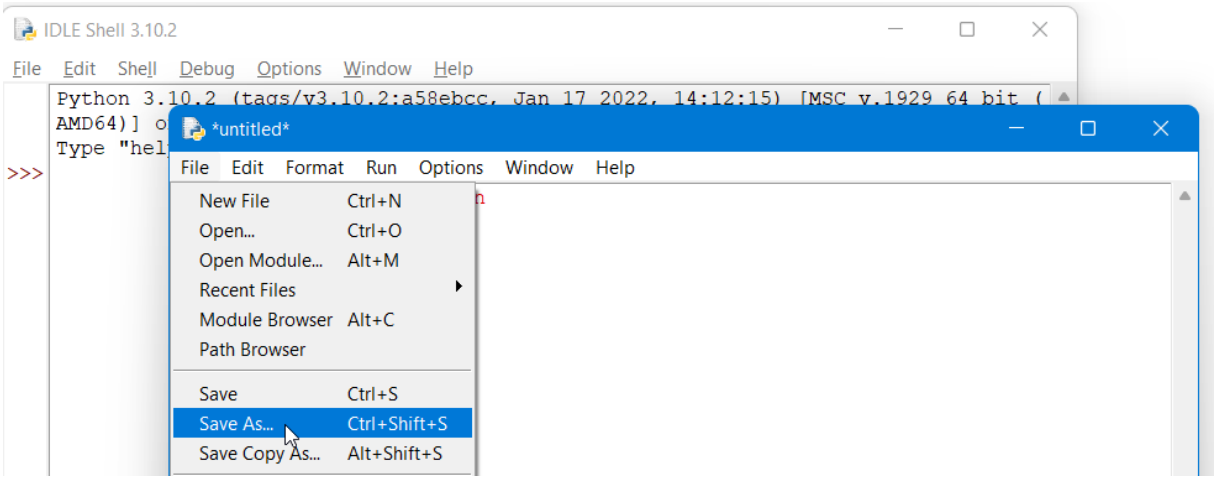
Über File/New File machen wir ein neues Dokument auf:



Es öffnet sich ein weiteres Fenster, in dem wir den Sourcecode ablegen. :



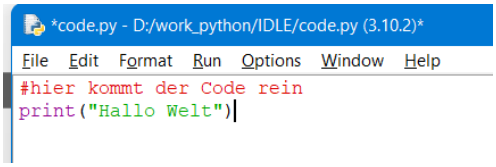
Mit File/Save as… speichern wir das Dokument:



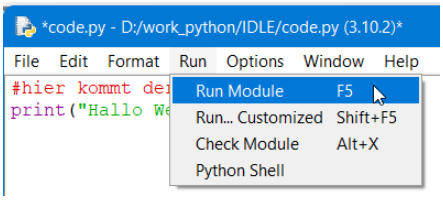
Dateiendung ist „.py“, ich nehme hier mal code.py:

:

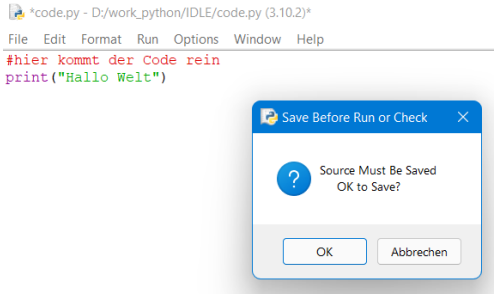
Auch wir wollen natürlich mit einem „Hello-World“-Programm starten, die Funktion zur Ausgabe auf der Kommandozeile ist die Print-Funktion, Syntax ist denkbar einfach **`print()`**. In den Klammern kommt der auszugebende Wert. Da es sich bei uns um einen String handelt, müssen wir das in Anführungsstriche setzen, also **`print("Hallo Welt")`**.



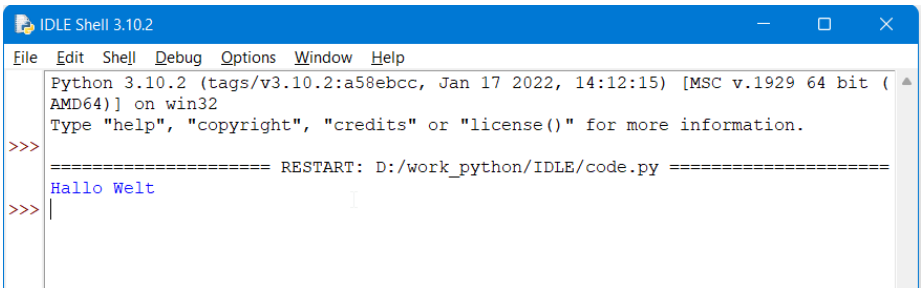
Um das Programm auszuführen, müssen wir es erst speichern, dann über Run/Run Module oder Funktionstaste F5 starten:



Ist die Änderung noch nicht gespeichert, werden wir darauf hingewiesen:



Jetzt sollte wirde das Schell-Fenster hochkommen, und die Ausgabe sollte so aussehen:

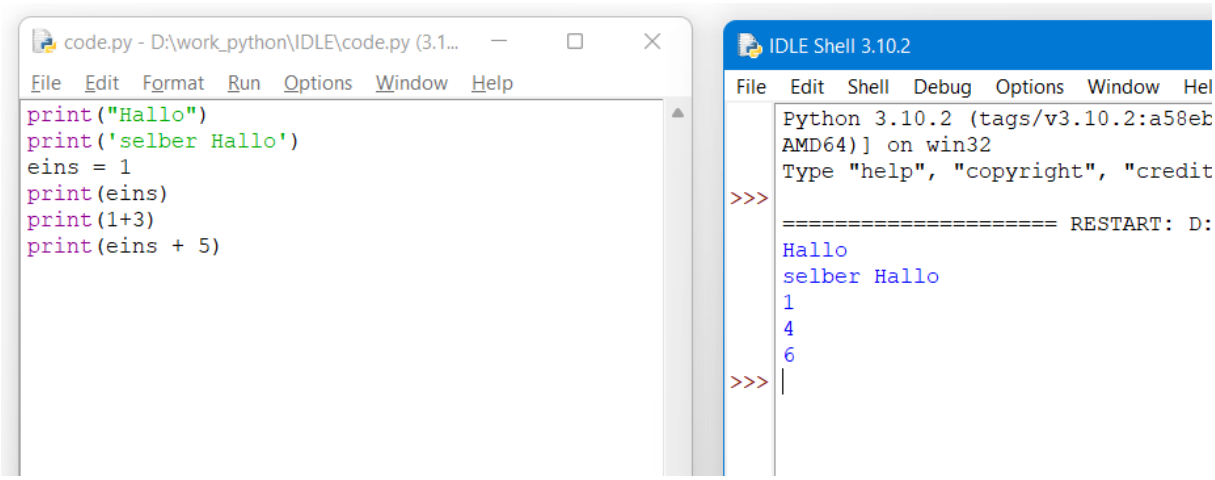


Bei Euch auch – prima, das war’s dann schon, wir können gleich loslegen!

4. Projekt „Spirograph in Python“

Bevor es losgeht, hier noch ein paar einleitende Worte.

In Python ist sehr viel sehr intuitiv und wenn man von anderen, wesentlich restriktiveren Sprachen kommt, ist Python vielleicht gewöhnungsbedürftig. Wenn ich mir nachfolgenden Code anschaue, bin ich zumindest erstaunt:

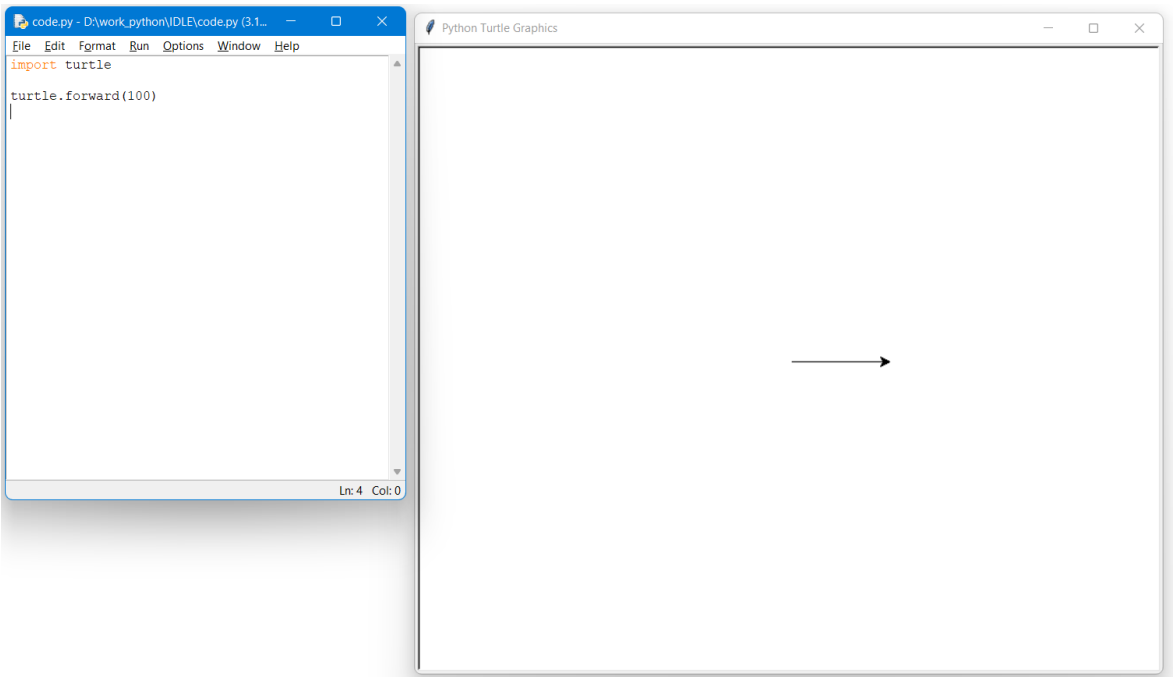


In anderen Sprachen wäre so etwas nicht möglich. Aber legen wir doch mal richtig los.

Um mit der Bibliothek turtle zu arbeiten, müssen wir sie zuerst importieren. Die kleinstmögliche Code-Abfolge, die ich gefunden habe ist:

```
1 import turtle
2
3 turtle.forward(100)
```

Das Ergebnis ist, es geht ein neues Fenster auf in dem ein Strich mit einer Länge von 100 Pixeln gezogen wird:



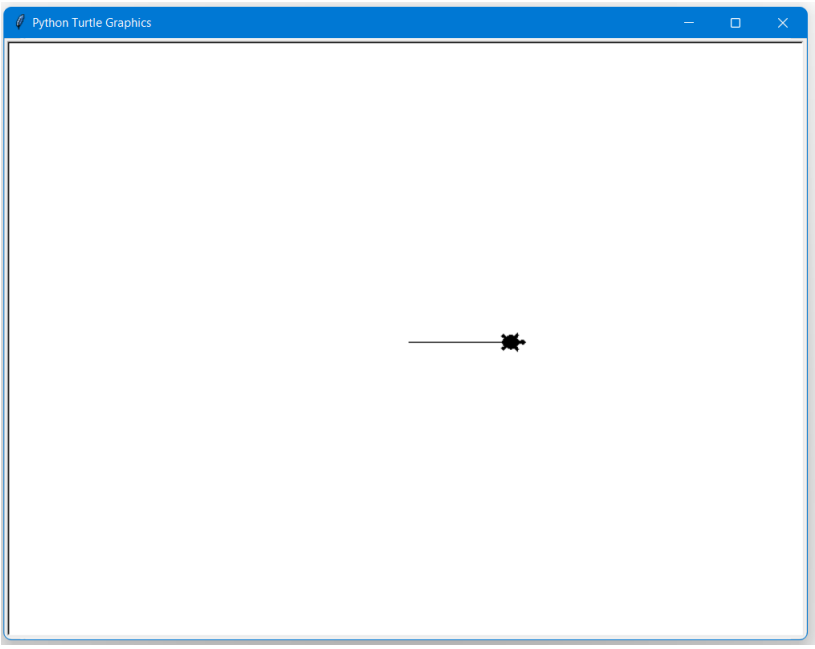
Die erste Erweiterung (in grün) ist wie folgt:

```
 1 import turtle
 2
 3 screen = turtle.Screen()
 4 screen.setup(800,600)
 5
 6 turtle.shape("turtle")
 7
 8 turtle.forward(100)
 9
10 turtle.done()
```

In Zeile 3 wird eine Variable für die Anzeige definiert, Zeile 4 weisen wir ihr die Werte 800 und 600 zu, was bedeutet, dass der Bildschirm 800 Pixel breit (also x-Achse) und 600 Pixel hoch (also y-Achse) ist.

Die Form des Pfeils (shape) können wir auch ändern, mit Turtle wird aus dem Pfeil eine Schildkröte – lustiges Feature.

Wenn wir das jetzt laufen lassen sehen wir:



Weitere Spielereien werden gleich folgen. Zum Nachlesen sei an dieser Stelle auf die Dokumentation auf der Seite https://docs.python.org/3/library/turtle.html hingewiesen. Hier bekommt man alle notwendigen Informationen für die Steuerung der Schildkröte.

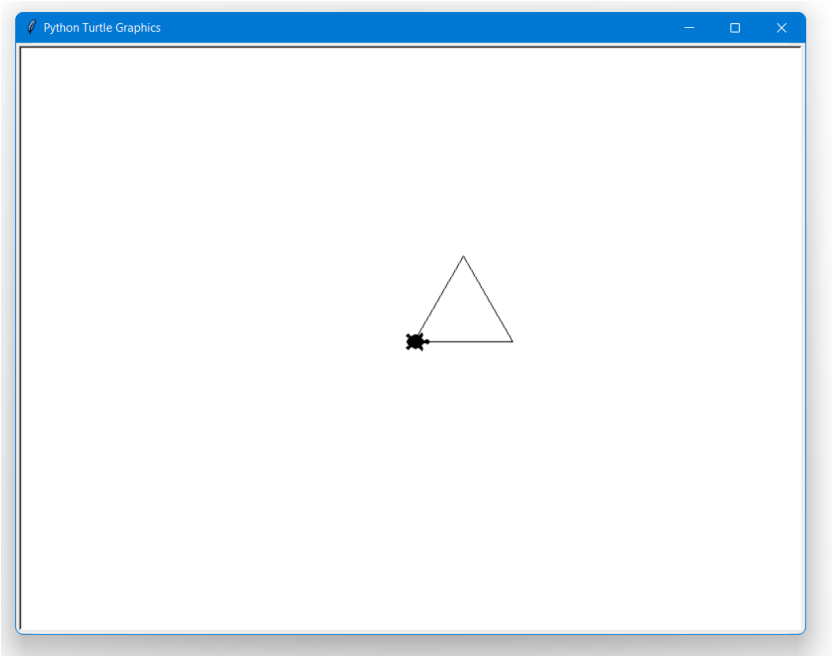
Nachdem die Schildkröte also diesen Weg von 100 Pixel zurückgelegt hat, stoppt sie. Wir können ihr jetzt weitere Anweisungen geben, zum Beispiel, dass sie sich nach links oder rechts drehen soll, dabei geben wir ihr die Gradzahl der Drehung mit.

Versuchen wir als nächstes ein Dreieck zu zeichnen. Die Schildkröte läuft die ersten 100 Pixel und muss sich dann drehen. Soll sie nach oben weiter gehen, muss sie sich nach links drehen. Für ein gleichseitiges Dreieck gilt, alle Kanten sind gleich lang und jeder Innenwinkel hat 60°.

Der Code dafür ist also:

```
 1 import turtle
 2
 3 screen = turtle.Screen()
 4 screen.setup(800,600)
 5
 6 turtle.shape("turtle")
 7
 8 turtle.forward(100)
 9 turtle.left(120)
10 turtle.forward(100)
11 turtle.left(120)
12 turtle.forward(100)
13 turtle.left(120)
14
15 turtle.done()
```

Zeilen 9, 11 und 13 dreht sich die Schildkröte also nach links für 120° (180° – 60°). Das Bild sollte dann so aussehen:



Okay, sieht schonmal ganz gut aus.

Zeilen 8 – 13 wiederholen sich aber, da können wir eine Schleife einsetzen.

In Python gibt es for- oder while-Schleifen, ich fange hier mal mit einer for-Schleife an. Syntax ist

```
1 for i in range(<wert>):
2     <mach was in der Schleife>
```

Das war es, kein weiteres Hochzählen von i, das macht die Schleife automatisch. Cool, oder?

Wichtig hier die Einrückung in der 2. Zeile. Der Compiler interpretiert alles, was in den nachfolgend eingerückten Zeilen steht als der Schleife zugehörig. Der nächste Eintrag beginnend in Position 1 befindet sich damit dann schon außerhalb der Klammer.

Also neuer Code für uns:

```
 1 import turtle
 2
 3 screen = turtle.Screen()
 4 screen.setup(800,600)
 5
 6 turtle.shape("turtle")
 7
 8 for i in range(3):
 9     turtle.forward(100)
10     turtle.left(120)
11
12 turtle.done()
```

Gleiches Ergebnis, richtig?

Okay, wir machen jetzt an das Ende jeder Linie noch einen Kreis. Das geht denkbar einfach mit:

```
turtle.circle(<radius>)
```

Ich entscheide mich zunächst mal für einen Radius von 10 Pixeln:

```
 1 import turtle
 2
 3 screen = turtle.Screen()
 4 screen.setup(800,600)
 5
 6 turtle.shape("turtle")
 7
 8 for i in range(3):
 9     turtle.forward(100)
10     turtle.left(120)
11     turtle.circle(10)
12
13 turtle.done()
```

Spielt ein bisschen rum mit den Parametern, damit Ihr ein Gefühl dafür bekommt.

Okay, weiter im Text.

Um jetzt die Form des Dreiecks aufzubrechen, müssen wir den Radius der Links-Drehung verändern. Dann haben wir „nur“ eine Linie, an deren Ende ein Kreis ist. Ändern wir den Code wie folgt:

```
 1 import turtle
 2
 3 screen = turtle.Screen()
 4 screen.setup(800,600)
 5
 6 turtle.shape("turtle")
 7 turtle.speed(0)
 8
 9 for i in range(30):
10     turtle.forward(100)
11     turtle.left(111)
12     turtle.circle(10)
13
14 screen.exitonclick()
```

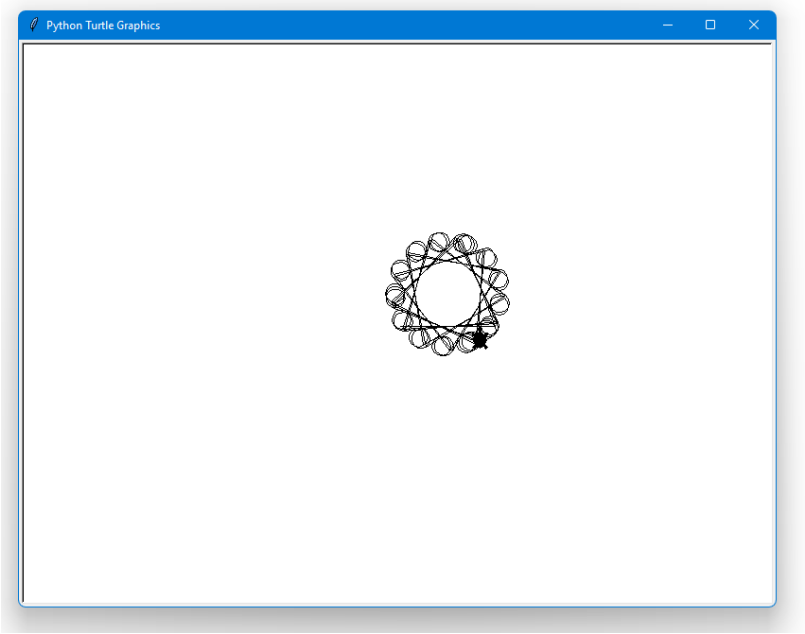
Was bedeutet das? Zeile 7 sagt, dass wir die Schildkröte so schnell wie möglich machen, da wir die Arbeitsweise ja nun kennen. Die Skala geht da von 1 für langsam bis 10 für schnell, 0 toppt das aber, es bedeutet „sehr schnell“.

In Zeile 9 sagen wir jetzt, dass wir den Schleifendurchlauf nicht mehr 3-mal, sondern 30-mal machen wollen.

Zeile 11 setzte den Radius auf 111°. Spielt auch damit rum.

Zeile 14 finde ich cool, das gibt uns die Möglichkeit, ach der Verarbeitung irgendwo im Bild zu klicken und das Fenster geht zu.

Was sehen wir nun?



Sehr schön, das sieht doch schon gut aus.

Aber alles noch sehr statisch, da ist noch keine Dynamik drin. Bevor wir damit loslegen, ändern wir den Code so, dass wir in der Schleife mit Variablen arbeiten – unser erstes „Refactoring“ sozusagen…

```
 1 import turtle
 2
 3 screen = turtle.Screen()
 4 screen.setup(800,600)
 5
 6 turtle.shape("turtle")
 7 turtle.speed(0)
 8
 9 laenge_linie = 100
10 drehung = 111
11 radius_kreis = 10
12
13 for i in range(30):
14     turtle.forward(laenge_linie)
15     turtle.left(drehung)
16     turtle.circle(radius_kreis)
17
18 screen.exitonclick()
```

Zeilen 9 bis 11 sind unsere Variablen, die wir mit den Werten vorbelegt haben, die in der Schleife enthalten waren. Dafür haben wir die Werte in Zeile 14 bis 16 durch die Variablen ersetzt. Da wir sonst keine Änderungen gemacht haben, ist das Bild identisch zur Vorversion. Richtig? Gut.

Der nächste Schritt bringt uns zu unserer ersten eigenen Funktion. Die Syntax ist

```
1 def name_der_funktion(Parameter_1, …, Parameter_n):
2     <mach was in der Funktion>
```

Erklärt sich eigentlich selbst, oder? Die Parameter in den Klammern sind optional, man muss nicht immer was mitgeben. In unserem Fall wollen wir das Zeichnen in eine Funktion auslagern, daher müssen wir die 3 Parameter `laenge_linie, drehung, radius_kreis` mitgeben, sonst funktioniert es ja nicht.

Der neue Code sieht dann so aus:

```
 1 import turtle
 2
 3 screen = turtle.Screen()
 4 screen.setup(800,600)
 5
 6 turtle.shape("turtle")
 7 turtle.speed(0)
 8
 9 laenge_linie = 100
10 drehung = 111
11 radius_kreis = 10
12
13 def zeichnen(laenge_linie, drehung, radius_kreis):
14     turtle.forward(laenge_linie)
15     turtle.left(drehung)
16     turtle.circle(radius_kreis)
17
18 for i in range(30):
19     zeichnen(laenge_linie, drehung, radius_kreis)
20
21 screen.exitonclick()
```

Zeilen 13 bis 16 beheimaten nun unsere Funktion, die wir in der Schleife jetzt 30-mal rufen.

Gedanklich haben wir uns damit komplett vom Zeichnen eines Dreiecks verabschiedet, das ist Euch aufgefallen, richtig?

Und weil das so ist, können wir jetzt mit den einzelnen Parametern spielen.

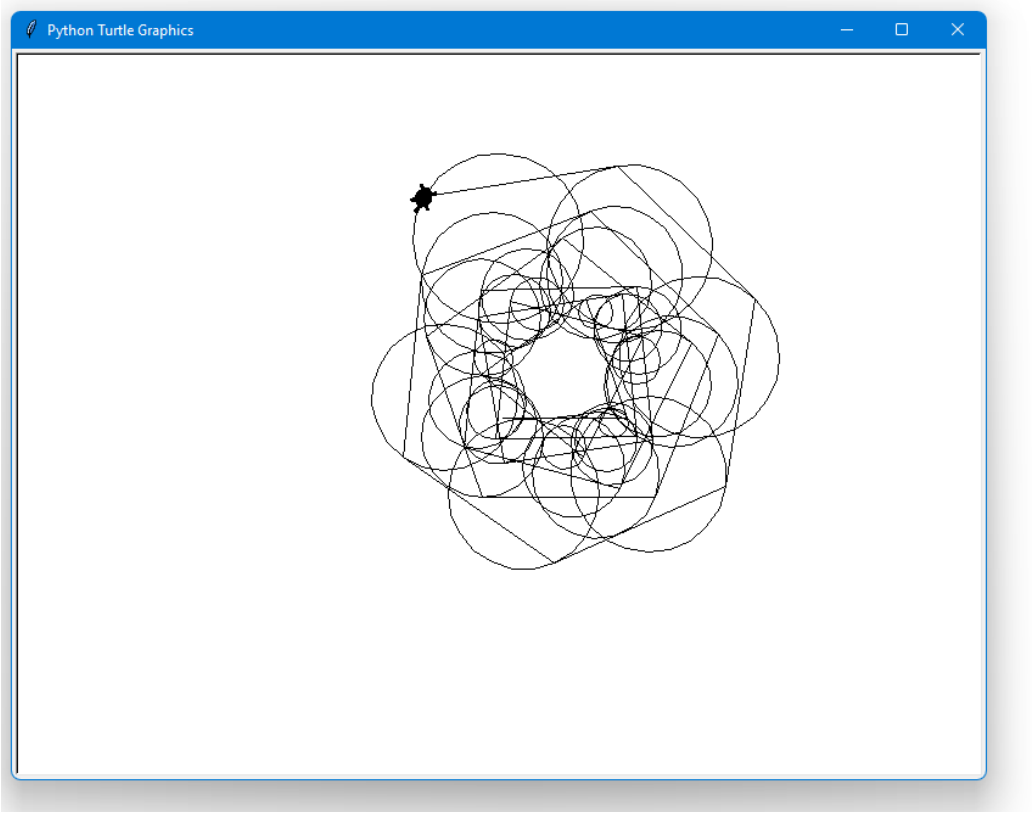
Die nächsten Änderungen machen wir jetzt ausschließlich in der Schleife. Ich mache mal einen Vorschlag:

```
17 …
18 for i in range(30):
19     laenge_linie = laenge_linie+2
20     drehung -= 2
21     radius_kreis += 2
22
23     zeichnen(laenge_linie, drehung, radius_kreis)
24 …
```

Zeilen 19 bis 21 zeigen die Veränderung. Da wir 30-mal die Schleife durchlaufen, wird bei jedem Durchlauf der Wert geändert. Damit sind kein Strich und kein Kreis so wie der vorherige.

Ich habe übrigens die lange Schreibweise in Zeile 19 absichtlich gelassen, beides geht, die kurze Zuweisung wir in Zeilen 20 und 21 oder die lange Variante. Mir persönlich ist die kurze Schreibweise lieber. Das ist aber Geschmackssache.

Also, wie sieht das Ergebnis aus?



Schick. Sehr schick. Für den Anfang…

Da geht noch was. Okay, nächste Änderung:

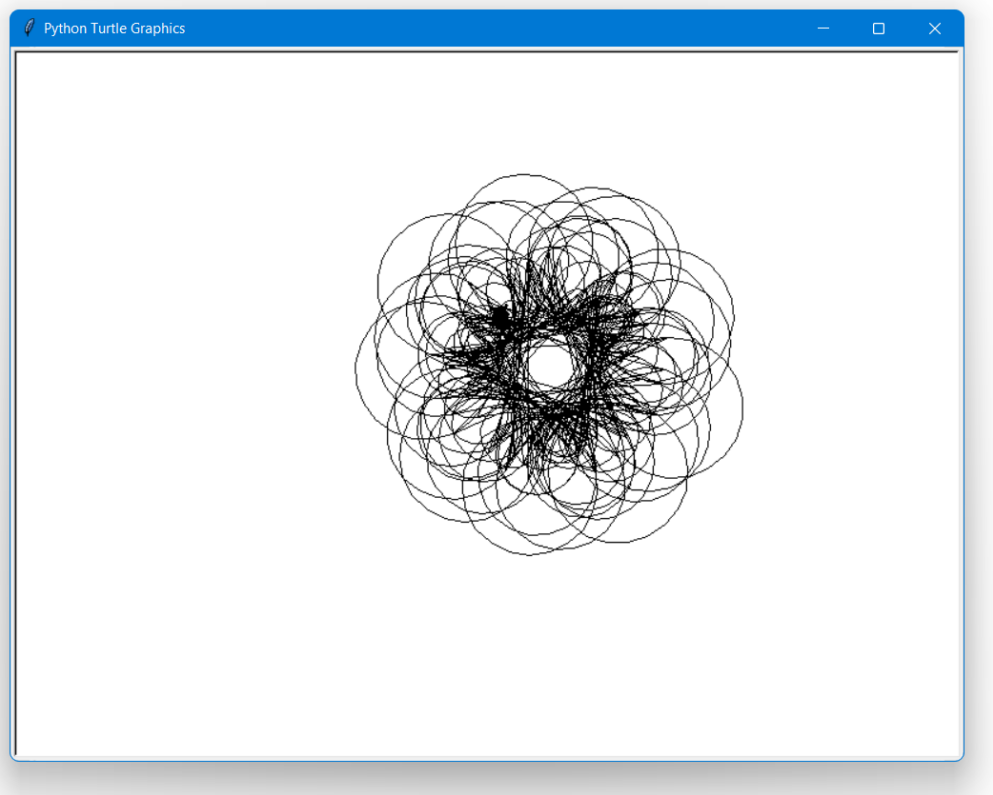
```
17 …
18 for i in range(100):
19     if i < 50:
20         radius_kreis += 1
21         laenge_linie += 2
22         drehung += 2
23     else:
24         radius_kreis -= 1
25         laenge_linie -= 2
26         drehung -= 2
27 
28     zeichnen(laenge_linie, drehung, radius_kreis)
29 …
```

Aha, das gute „wenn – dann“ ist dazugekommen. Aber der Reihe nach:

In Zeile 18 erhöhen wir den Durchlauf mal auf 100. Also wenn der Index des Schleifendurchlaufs kleiner als 50 ist (Zeile 19), vergrößern sich die Werte konstant.

Andernfalls, nämlich ab dem Zeitpunkt, dass der Index 50 ist, verkleinern sich die Werte wieder konstant.

Wie sieht das aus?



Oh ja, da ist schon richtig Musik drin, das gefällt mir.

Aber kann man das nicht farbig gestalten? Ihr ahnt es, klar, das geht.

Die Linie ist der „`pen`“, die Änderung der Linienfarbe dementsprechend

```
turtle.pencolor(<Name_der_Farbe_oder_HEX_Nummern_der_Farbe>)
```

Mit `<Name_der_Farbe>` ist ein String wie „`green`“ oder „`red`“ gemeint.

Mit `<HEX_Nummer_der_Farbe>` könnte für rot „`#FF0000`“ angegeben werden

Wer sich damit richtig auseinandersetzen möchte, dem sei die nachfolgende Seite empfohlen:
https://www.wikipython.com/tkinter-ttk-tix/summary-information/colors/

Wollen wir nur die Füllfarbe ändern, geht das über ist „`fillcolor`“

```
turtle. fillcolor(<Farbe>)
```

Es gibt noch 2 weitere Möglichkeiten:

`turtle.color(<Farbe>)` - setzt Linie und Füllung zusammen auf die gleiche Farbe,

`turtle.color((<Farbe_der_Linie>, (<Farbe_der_Füllung>)` - macht das in einem Aufwasch, zu beachten ist das Komma zwischen den beiden Farbangaben. Cool, oder?

Ach, und den Hintergrund können wir natürlich auch farblich absetzen, das geht mit

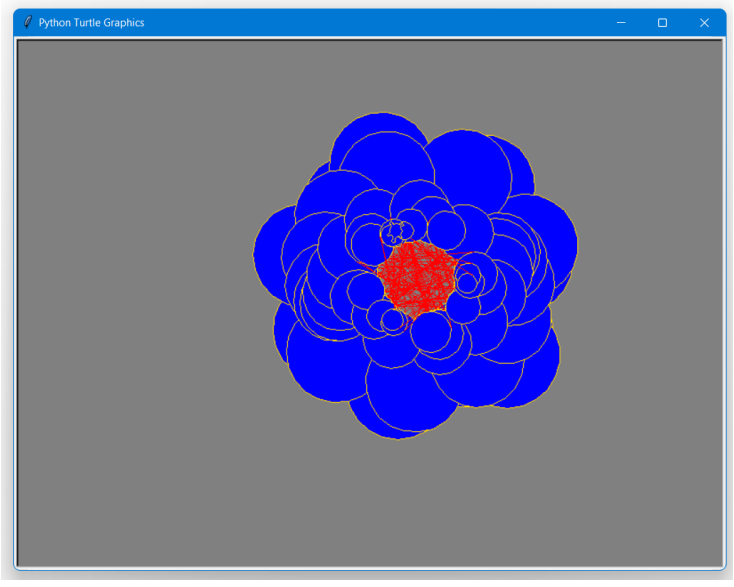
```
turtle.bgcolor(<Farbe>)
```

Damit haben wir farbtechnisch alles zusammen, gehen wir es also an.

Der Hintergrund soll grau sein, die Linie rot und die Kreise goldig, Füllung der Kreise ist blau. Bei mir sieht der Code dann so aus:

```
1  import turtle
2
3  screen = turtle.Screen()
4  screen.setup(800,600)
5
6  turtle.shape("turtle")
7  turtle.speed(0)
8  turtle.bgcolor("gray")
9
10 laenge_linie = 100
11 drehung = 111
12 radius_kreis = 10
13
14 def zeichnen(laenge_linie, drehung, radius_kreis):
15     turtle.color("red")
16     turtle.forward(laenge_linie)
17     turtle.left(drehung)
18     turtle.color("gold","blue")
19     turtle.begin_fill()
20     turtle.circle(radius_kreis)
21     turtle.end_fill()
22
23 for i in range(100):
24     if i < 50:
25         radius_kreis += 1
26         laenge_linie += 2
27         drehung += 2
28     else:
29         radius_kreis -= 1
30         laenge_linie -= 2
31         drehung -= 2
32
33     zeichnen(laenge_linie, drehung, radius_kreis)
34
35 screen.exitonclick()
```

Und die Ausgabe?



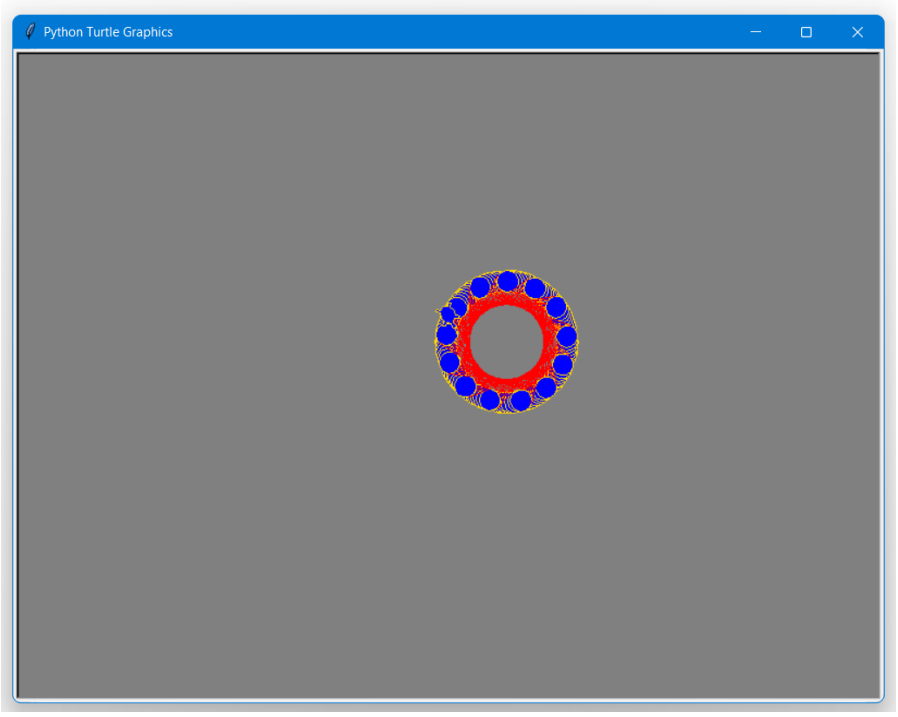
Najaaa,... schön bunt.

Aber so wie die Werke aus den Spirographen ist das nicht gerade…

Okay, gehen wir nochmal ein paar Schritte zurück. Wir lassen die Farben drin, spielen jetzt aber nicht mit den Größen. Daher nehmen wir den ganzen if-Zweig einfach raus:

```
1  import turtle
2
3  screen = turtle.Screen()
4  screen.setup(800,600)
5
6  turtle.shape("turtle")
7  turtle.speed(0)
8  turtle.bgcolor("gray")
9
10 laenge_linie = 100
11 drehung = 111
12 radius_kreis = 10
13
14 def zeichnen(laenge_linie, drehung, radius_kreis):
15     turtle.color("red")
16     turtle.forward(laenge_linie)
17     turtle.left(drehung)
18     turtle.color("gold","blue")
19     turtle.begin_fill()
20     turtle.circle(radius_kreis)
21     turtle.end_fill()
22
23 for i in range(100):
24
25     zeichnen(laenge_linie, drehung, radius_kreis)
26
27 screen.exitonclick()
```

Wie sieht das dann aus?

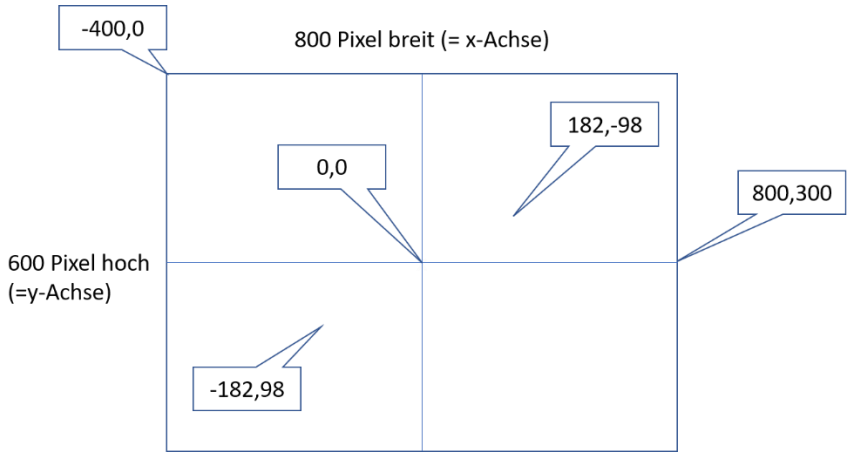


Ja, das kommt schon eher hin.

2 Sachen stören mich noch.

Nummer 1 ist, dass das Bild nicht mittig ist. Die Schildkröte startet zwar genau in der Mitte, da wir aber sagen, sie soll mit einer Linie von 100 Pixeln anfangen, ist das Bild sozusagen 50 Pixel zu weit rechts.

Wir können der Schildkröte eine absolute Position zuweisen. Sie bewegt sich in einem Koordinatensystem, dessen Mittelpunt die Position 0,0 hat:



Wenn wir jetzt flexibel sein wollen, was die Länge der Linie angeht, sollten wir die Position auf der x-Achse nicht mit dem fixen Wert „- 50“ angeben, sondern als mathematischen Ausdruck. Wollen wir nur die Position von x setzen, können wir das gezielt mit

```
turtle.setx(<Position>)
```

machen (analog geht natürlich auch `turtle.sety(<Position>)`).

Alternativen sind

```
turtle.setpos(<Position_x>,<Position_y>)
```
oder
```
turtle.goto(<Position_x>,<Position_y>)
```

Bevor wir das jetzt einbauen, noch ein Hinweis. Bei Start fängt die Schildkröte sofort an zu malen, das ist ja ihr Job. Wir wollen das aber nicht, sie soll erst anfangen zu malen, wenn sie die Startposition erreicht haben. Dazu geben wir ihr die Anweisung den Stift zunächst anzuheben, und erst wenn sie am Ziel ist, senken wir den Stift wieder ab. Das geschieht mittels der beiden Funktionen

```
turtle.penup()
```
und
```
turtle.pendown()
```

Es würden auch `turtle.pu()` und `turtle.pd()` oder `turtle.up()` und `turtle.down()` tun. Was Ihr davon nutzt, liegt ganz bei Euch, probiert es einfach aus.

Und was machen wir jetzt zwischen den beiden Anweisungen?

Na, wir wollen die Hälft der Linienlänge auf der x-Achse nach links gehen. Was haltet Ihr von

```
pixel_nach_links = laenge_linie/2
startpunkt = 0-pixel_nach_links
```

Ja, das geht. Alternativ in einer Zeile:

```
turtle.setx((laenge_linie/2)*(-1))
```

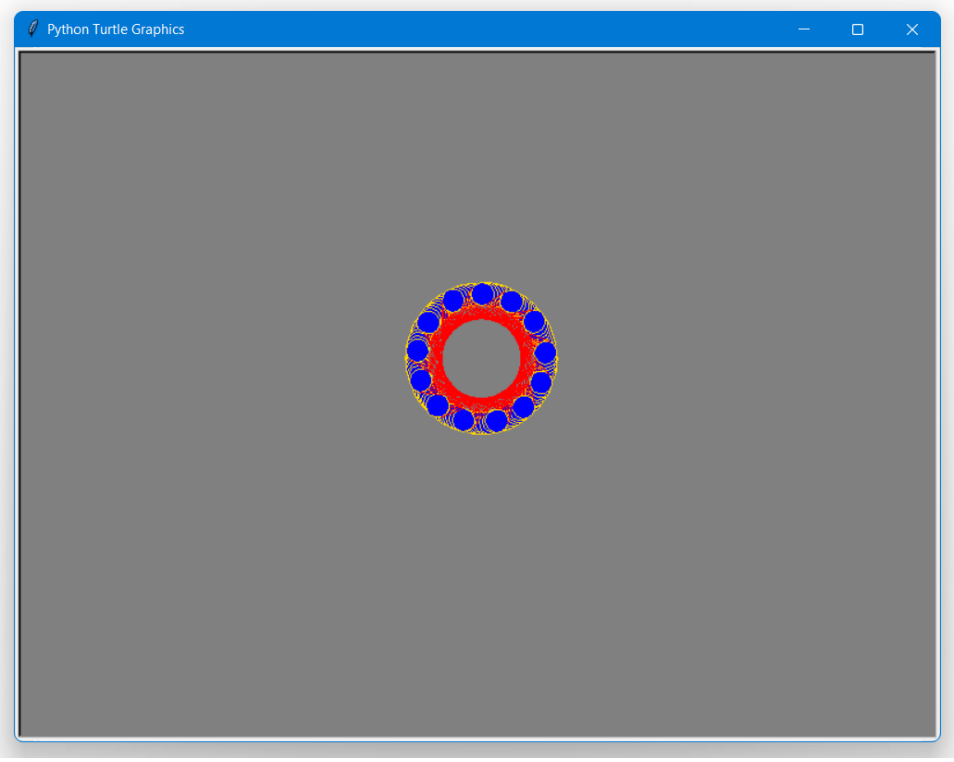
Richtig, niemand mag Angeber…

Bevor wir jetzt den neuen Code einpflegen, noch zur 2. Sache die mich stört – am Ende steht die Schildköte relativ unmotiviert in der Gegend rum, man kann ihr sagen, dass sie sich verstecken soll, das geht mit

```
turtle.hideturtle()
```

Und wenn sie wieder erscheinen soll, geht das mit `turtle.showturtle()`. So einfach kann es sein. Der vollständige neue Code sieht dann so aus:

```
1  import turtle
2
3  screen = turtle.Screen()
4  screen.setup(800,600)
5
6  turtle.shape("turtle")
7  turtle.speed(0)
8  turtle.bgcolor("gray")
9
10 laenge_linie = 100
11 drehung = 111
12 radius_kreis = 10
13
14 turtle.penup()
15 turtle.setx((laenge_linie/2)*(-1))
16 turtle.pendown()
17
18 def zeichnen(laenge_linie, drehung, radius_kreis):
19     turtle.color("red")
20     turtle.forward(laenge_linie)
21     turtle.left(drehung)
22     turtle.color("gold","blue")
23     turtle.begin_fill()
24     turtle.circle(radius_kreis)
25     turtle.end_fill()
26
27 for i in range(100):
28
29     zeichnen(laenge_linie, drehung, radius_kreis)
30
31 turtle.hideturtle()
32 screen.exitonclick()
```



Ja, das gefällt mir.

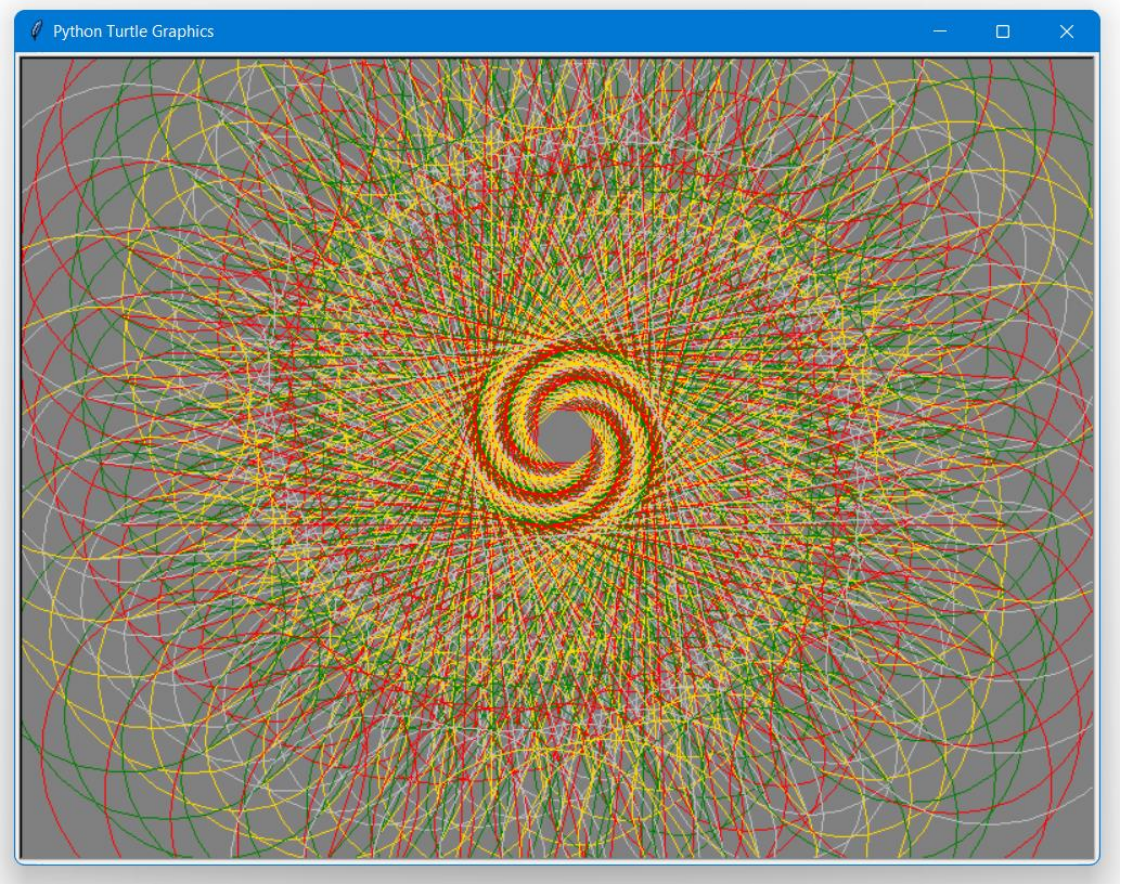
Als nächstes geben wir noch ein bisschen mehr Bewegung in der Sache. Ich definiere jetzt mal die 4 Weihnachtsfarben, rot, grün, silber und gold. Neuer Code in der Funktion und der Schleife:

```
17 …
18 def zeichnen(laenge_linie, drehung, radius_kreis):
19     turtle.forward(laenge_linie)
20     turtle.left(drehung)
21     turtle.circle(radius_kreis)
22 
23 for i in range(100):
24 
25     laenge_linie += 5
26     radius_kreis += 1
27     turtle.color("red")
28     zeichnen(laenge_linie, drehung, radius_kreis)
29     turtle.color("green")
30     zeichnen(laenge_linie, drehung, radius_kreis)
31     turtle.color("gold")
32     zeichnen(laenge_linie, drehung, radius_kreis)
33     turtle.color("silver")
34     zeichnen(laenge_linie, drehung, radius_kreis)
35 
36 turtle.hideturtle()
37 screen.exitonclick()
```

Die Funktion selbst wird wieder reduziert auf das Zeichnen der Linie und des Kreises.

In der Schleife ändern wir die Länge der Linie und den Durchmesser des Kreises, dann übergeben wir je Farbe die aktuellen Werte an die Funktion. Da die Schleife 100 Durchläufe hat, werden 400 Strich-Kreis-Kombinationen gemalt.

Wie sieht das Bild jetzt aus?



Sehr schön! Jetzt bin ich begeistert!

Einen Tipp habe ich noch – wir können uns die Ausgabe sichern. Der Befehl dazu lautet

```
screen.getcanvas().postscript(file="<File_Name>.eps")
```

Die Endung EPS steht für „Encapsulated PostScript“. Einmal gespeichert kann man das Bild über freie EPS-Konverter online nach jpg oder png konvertieren lassen.

Das soll es zum Thema Spirograph gewesen sein, Ihr habt sicher tolle Einfälle, was noch alles möglich ist.

Bevor es weitergeht, hier noch der Hinweis auf eine sehr schöne Seite:

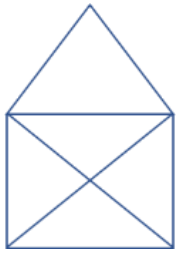
http://csc.columbusstate.edu/carroll/1301/turtleGraphics/cheatSheet.pdf

Zur Sicherheit auf der nächsten Seite noch einmal der letzte Codestand komplett.

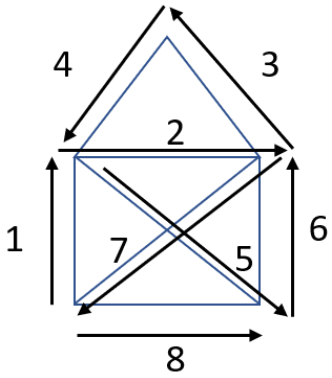
```
1  import turtle
2
3  screen = turtle.Screen()
4  screen.setup(800,600)
5
6  turtle.shape("turtle")
7  turtle.speed(0)
8  turtle.bgcolor("gray")
9
10 laenge_linie = 200
11 drehung = 157
12 radius_kreis = 20
13
14 turtle.penup()
15 turtle.setx((laenge_linie/2)*(-1))
16 turtle.pendown()
17
18 def zeichnen(laenge_linie, drehung, radius_kreis):
19     turtle.forward(laenge_linie)
20     turtle.left(drehung)
21     turtle.circle(radius_kreis)
22
23 for i in range(100):
24
25     laenge_linie += 5
26     radius_kreis += 1
27     turtle.color("red")
28     zeichnen(laenge_linie, drehung, radius_kreis)
29     turtle.color("green")
30     zeichnen(laenge_linie, drehung, radius_kreis)
31     turtle.color("gold")
32     zeichnen(laenge_linie, drehung, radius_kreis)
33     turtle.color("silver")
34     zeichnen(laenge_linie, drehung, radius_kreis)
35
36 turtle.hideturtle()
37
38 screen.getcanvas().postscript(file="ausgabe.eps")
39 screen.exitonclick()
```

5. Projekt „Das Haus des Nikolaus“

Wer kennt es nicht, das Zeichenspiel „das Haus des Nikolaus“. Mit 8 Strichen muss ein Haus gemalt werden, das so aussieht:



Dabei darf der Stift nicht abgesetzt werden und Linien dürfen nicht doppelt gemalt werden. Es gibt diverse Lösungen, ich habe mich mal für die folgende Reihenfolge entschieden:



Was bedeutet das jetzt für unsere Schildkröte? Fangen wir gleich mit den ersten Zeilen Code an:

```
1  import turtle
2
3  screen = turtle.Screen()
4  screen.setup(800,600)
5
6  speed = 0
7  laenge_seite = 100
8
9  def haus_malen(laenge_seite):
10     turtle.left(90)
11     turtle.forward(laenge_seite)
12     turtle.right(90)
13     turtle.forward(laenge_seite)
14
15 haus_malen(laenge_seite)
16
17 screen.exitonclick()
```

Alles nichts Neues für uns, die ersten beiden Striche sind schon in der Funktion definiert.

Jetzt machen wir das Dach. Die Kanten lassen wir gleich lang, das bedeutet, wir haben auch hier wieder ein gleichschenkliges Dreieck.

Aus dem Matheunterricht wissen wir noch, dass die Summe der Winkel in einem Dreieck immer 180° ergeben muss. Bei einem gleichschenkligen Dreieck sind das je Winkel also 60°.

Unsere Kröte muss sich nach links bewegen, also ist der Winkel 120°, nämlich 180° - 60°.

Dann die Pixel für Länge der Seite gehen, nochmal 120° drehen und wieder die Pixel für Länge der Seite gehen.

```
 9 def haus_malen(laenge_seite):
10     turtle.left(90)
11     turtle.forward(laenge_seite)
12     turtle.right(90)
13     turtle.forward(laenge_seite)
14     turtle.left(120)
15     turtle.forward(laenge_seite)
16     turtle.left(120)
17     turtle.forward(laenge_seite)
18
19 haus_malen(laenge_seite)
```

Jetzt wird es kniffelig und ich fühle mich wie in der Schule. Und ganz ehrlich – ich habe auch nachgeschaut…

Die Striche 5 und 7 sind ja länger als die Seitenlänge, in diesem Fall haben wir ja 2 Dreiecke mit jeweils einem 90° Winkel. Das sind die Striche 1 und 8 bzw. 8 und 6 und dann die längeren Seiten 5 und 7.

Wie war das noch?

$a^2 + b^2 = c^2$. Also ist $c = \sqrt{a^2 + b^2}$

Das müssen wir zum Glück nicht selber codieren, dafür gibt es die Bibliothek „math“, die müssen wir importieren und dann können wir c berechnen lassen.

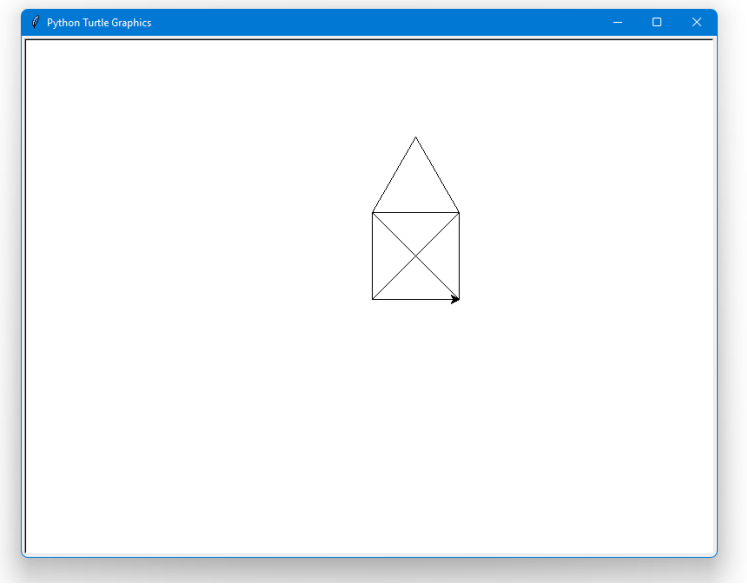
```
 1 import turtle
 2 import math
 3
 4 screen = turtle.Screen()
 5 screen.setup(800,600)
 6
 7 turtle.speed(0)
 8 laenge_seite = 100
 9 laenge_mitte = math.sqrt(2*(laenge_seite*laenge_seite))
10
11 def haus_malen(laenge_seite):
12     turtle.left(90)
13 …
```

Jetzt bleibt nur noch die Drehung vor dem Zeichnen der langen Seite. Wir kommen mit 60° Schräglage an. Von dem Punkt sind es noch einmal 45° also 105°. 180° - 105° sind 75°. Klar, oder? Wenn nicht, nehmt Euch das Geo-Dreieck aus der Schublade links, es kann auch das sein, wo die eine Ecke abgebrochen ist…

Damit sieht die Funktion `haus_malen` bei mir jetzt so aus:

```
10 …
11 def haus_malen(laenge_seite):
12     turtle.left(90)
13     turtle.forward(laenge_seite)
14     turtle.right(90)
15     turtle.forward(laenge_seite)
16     turtle.left(120)
17     turtle.forward(laenge_seite)
18     turtle.left(120)
19     turtle.forward(laenge_seite)
20     turtle.left(75)
21     turtle.forward(laenge_mitte)
22     turtle.left(135)
23     turtle.forward(laenge_seite)
24     turtle.left(135)
25     turtle.forward(laenge_mitte)
26     turtle.left(135)
27     turtle.forward(laenge_seite)
28 …
```

Und, wie ist das Ergebnis? Bei mir sieht es so aus:



Grandios!

Okay, legen wir noch einen drauf, malen wir eine Häuserzeile mit unterschiedlich großen Häusern.

Wie gehen wir vor?

Die unterschiedliche Größe der Häuser entsteht ja durch eine unterschiedliche große Seitenlänge. Wir haben Schleifen kennen gelernt, wir können also den Aufruf der Funktion in eine solche Schleife packen und vor Aufruf die Länge der Seite neu vergeben.

Wie sieht der Code dazu aus?

```
28 …
29 for i in range (10):
30     laenge_seite += 2
31     haus_malen(laenge_seite)
32 
33 screen.exitonclick()
34 …
```

Die Startposition der Schildkröte verlegen wir nach weiter links, damit wir eine schöne Straße gemalt bekommen. Der Screen ist 800 Pixel breit, dann ist unser Startpunkt auf der x-Achse mal bei -380.

Was wir aber auch noch machen müssen, ist, die Berechnung der Länge der mittleren Linie mit in die Funktion zu packen, sonst ist das immer der gleiche Wert, der nur auf die Länge der Startseite passt:

```
 6 …
 7 turtle.speed(0)
 8 laenge_seite = 100
 9 turtle.penup()
10 turtle.setx(-380)
11 turtle.pendown()
12 
13 def haus_malen(laenge_seite):
14     laenge_mitte = math.sqrt(2*(laenge_seite*laenge_seite))
15     turtle.left(90)
16     turtle.forward(laenge_seite)
17 …
```

Mit 10 Wiederholungen läuft die Schildkröte rechts aus dem Bild, das ist natürlich nicht schön, da kümmern wir uns gleich drum.

Zuerst holen wir uns aber noch eine weitere Bibliothek an Bord, nämlich `random`.

Mit `random` können wir den Rechner zufällig ausgewählte Werte aus von uns festgelegten Bereichen aussuchen lassen, die wir dann nutzen können. Für Ganzzahlen ist die Syntax:

```
random.randint(<niedrigster_Wert>, <höchster_Wert>)
```

Wir nutzen das, um uns die Seitenlänge in der Schleife immer wieder neu vergeben zu lassen. Und das Tolle ist, wir können das in den Aufruf der Funktion einbauen. Zuerst der Import:

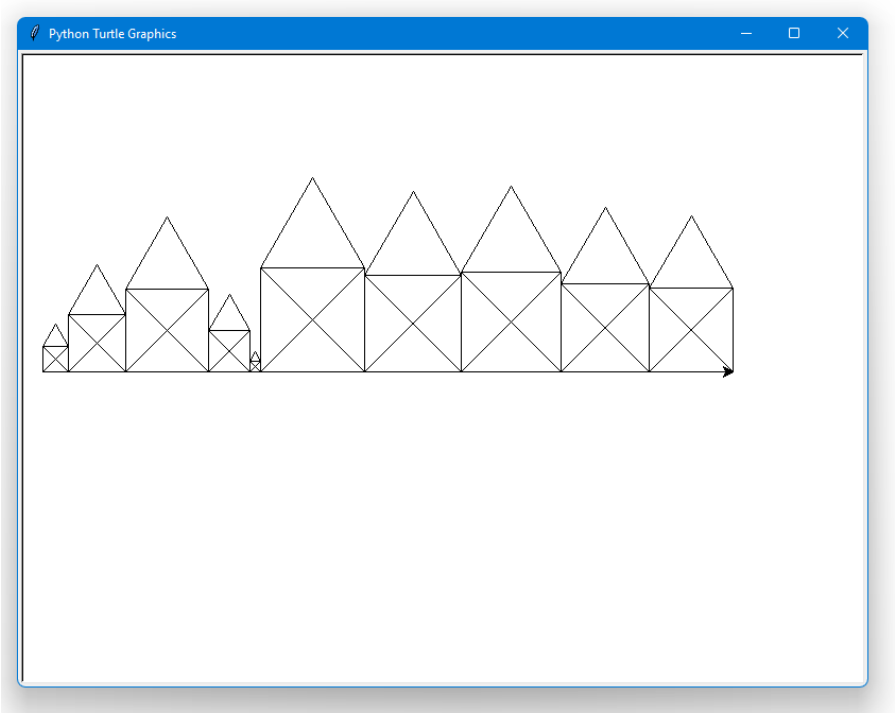
```
1 import turtle
2 import math
3 import random
4 …
```

Einbau in die Funktion, natürlich löschen wir die Zeile `laenge_seite += 2`:

```
32 …
33 for i in range (10):
34     haus_malen(random.randint(10, 100))
35 
36 screen.exitonclick()
37 …
```

Damit holt sich das `ramdom` Zahlen aus dem Bereich zwischen 10 und 100 und übergibt sie der Funktion.

Ab jetzt wird das bei Euch anders aussehen, da wir zufällige Werte nutzen. Der erste Durchlauf bei mir sah so aus:



Lustig.

Bleibt das Problem, dass die Schildkröte aus dem Bild läuft.

Um das zu verhindern, bauen wir den Aufruf rund um die Funktion um. Satt der `for`-Schleife

```
32 …
33 for i in range (10):
34     haus_malen(random.randint(10, 100))
35
36 screen.exitonclick()
37 …
```

setzen wir jetzt eine `while`-Schleife ein. Diese macht etwas solange die Abbruchbedingung nicht erreicht ist.

Schauen wir uns das direkt an unserem Beispiel an:

```
33 …
34 while not ende_erreicht:
35     laenge_seite = random.randint(10, 100)
36     if (laenge_seite + turtle.xcor()) < 400:
37         haus_malen(laenge_seite)
38     else:
39         ende_erreicht == True
```

```
40          break
41
42  screen.exitonclick()
```

Übersetzt so viel wie – mache, solange nicht `ende_erreicht` ist. Wie erreichen wir das? `ende_erreicht` ist eine Variable, ein sogenannter boolscher Wert, hat also nur 2 Zustände `0` und `1` oder `False` und `True`.

Die Deklaration erfolgt oben in Zeile 10:

```
 7  …
 8  turtle.speed(0)
 9  laenge_seite = 100
10  ende_erreicht = False
11
12  turtle.penup()
13  turtle.setx(-380)
14  turtle.pendown()
15  …
```

Also ist beim ersten Mal der die Varible auf `False`, was dazu führt, dass die Schleife betreten wird.

Mit `turtle.xcor()` in Zeile 36 holen wir uns die aktuelle Position der Schildkröte auf der x-Achse. Wenn wir die Länge, die sich der Rechner ausgesucht hat, auf diese Position addieren, dürfen wir nicht über die x-Position 400 hinausauskommen, sonst würden wir den sichtbaren Bereich verlassen.

Wenn also nach der Auswahl das Haus noch „reinpasst“, okay, falls nicht, setzen wir die Abbruchbedingung und verlassen die Schleife mit `break`, womit dann auch die `while`-Schleife verlassen wird.

An dieser Stelle ein Hinweis – versucht doppelte Verneinung zu vermeiden, das führt immer zu Verwirrungen beim Leser. In diesem Fall hätte ich es eigentlich positiv formulieren müssen.

```
 7  …
 8  turtle.speed(0)
 9  laenge_seite = 100
10  ein_haus_geht_noch = True
11  …
```

Der Code würde dann so aussehen:

```
34  …
35  while ein_haus_geht_noch:
36      laenge_seite = random.randint(10, 100)
37      if (laenge_seite + turtle.xcor()) < 400:
38          haus_malen(laenge_seite)
39      else:
40          ein_haus_geht_noch == False
41          break
42
43  screen.exitonclick()
```

Wie es Euch besser gefällt…

Zum Schluss noch eine Übung mit Listen.

Wenn wir jetzt aus verschiedenen Farben auswählen sollen, sind das ja keine Zahlen, sondern Strings wie „green“ oder „red“, das haben wir im ersten Projekt gesehen.

Die zur Wahl stehenden Farben können wir in eine sogenannte `Liste` packen. Eine `Liste` hat folgende Syntax

```
auswahl = [<Objekt_1>, <Objekt_2>, …, <Objekt_n>]
```

Entscheidend sind also hier die eckigen Klammern.

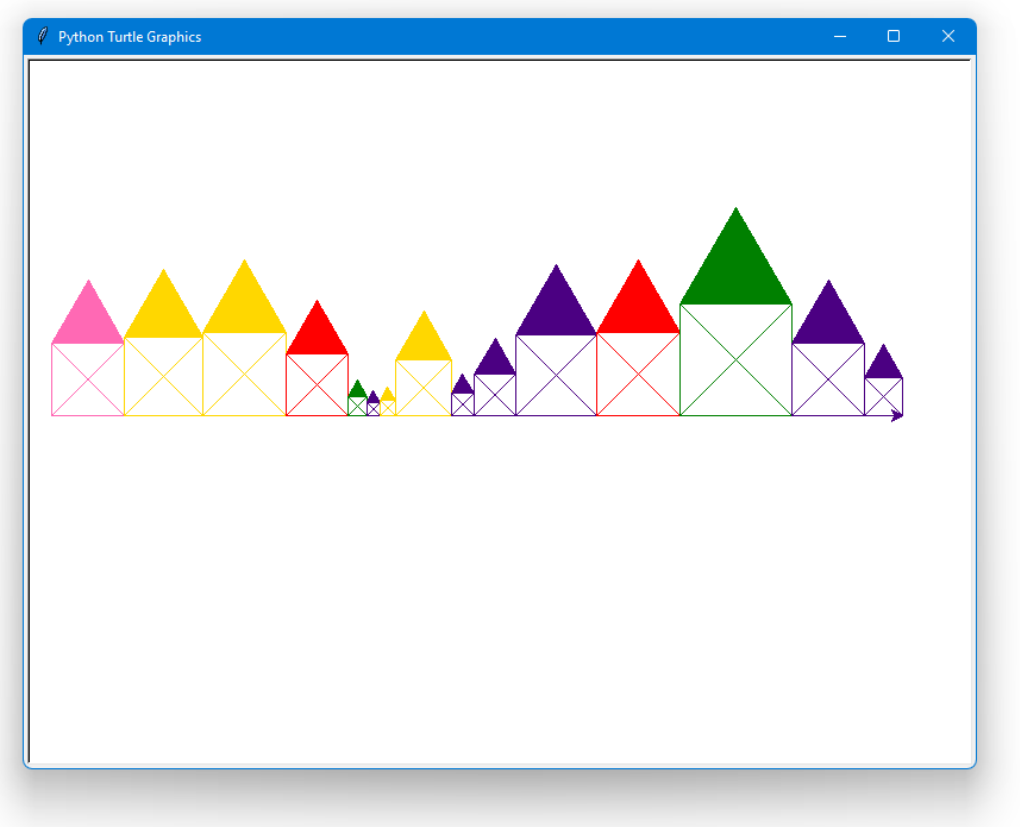
Listen sind sehr mächtige Kostruktionen, damit sollte man sich unbedingt näher beschäftigen. Hier gehe ich nicht weiter darauf ein, vielleicht komme ich in einem späteren Projekt darauf zurück.

Das Setzen der Farbe bauen wir in der Funktion mit ein, ich denke mal, das Dach können wir dann auch gleich mit ausfüllen lassen. Mein Code sieht dann so aus:

```
1  import turtle
2  import math
3  import random
4
5  screen = turtle.Screen()
6  screen.setup(800,600)
7
8  turtle.speed(0)
9  laenge_seite = 100
10 ein_haus_geht_noch = True
11 farben = ["red","green","indigo","hotpink","gold"]
12
13 turtle.penup()
14 turtle.setx(-380)
15 turtle.pendown()
16
17 def haus_malen(laenge_seite):
18     laenge_mitte = math.sqrt(2*(laenge_seite*laenge_seite))
19     turtle.color(random.choice(farben))
20     turtle.left(90)
21     turtle.forward(laenge_seite)
22     turtle.right(90)
23     turtle.forward(laenge_seite)
24     turtle.left(120)
25     turtle.begin_fill()
26     turtle.forward(laenge_seite)
27     turtle.left(120)
28     turtle.forward(laenge_seite)
29     turtle.left(75)
30     turtle.end_fill()
31     turtle.forward(laenge_mitte)
32     turtle.left(135)
33     turtle.forward(laenge_seite)
34     turtle.left(135)
35     turtle.forward(laenge_mitte)
36     turtle.left(135)
37     turtle.forward(laenge_seite)
38
```

```
39 while ein_haus_geht_noch:
40     laenge_seite = random.randint(10, 100)
41     if (laenge_seite + turtle.xcor()) < 400:
42         haus_malen(laenge_seite)
43     else:
44         ein_haus_geht_noch == False
45         break
46
47 screen.exitonclick()
```

Das Ergebnis kann sich sehen lassen:



So, das war es dann auch schon wieder, ich hoffe, es hat Euch Spaß gemacht.

Ich mache mich jetzt an mein erstes Spiel in Python, ich hoffe, Ihr seid wieder mit dabei.

Viel Spaß beim Nachbauen und wenn Euch was wirklich Tolles gelungen ist, schickt mir das Ergebnis gerne per Mail an papa@papa-programmiert.de, ich würde mich freuen.

Viele Grüße, Papa