12. LISTEN

DU LERNST HIER...

wie du in einer Liste oder in einem Tupel beliebig viele Werte speichern kannst und wie du auf diese Werte zugreifst.

WOZU BRAUCHT MAN LISTEN?

Statt mehrere Werte in verschiedenen Variable *a, b, c* einzeln abzuspeichern, kannst du die Listen oder Tuppels verwenden. Die Werte (Elemente) einer Liste werden in eckigen Klammern, beim Tuppel in runden Klammern aufgezählt. Beispielsweise:

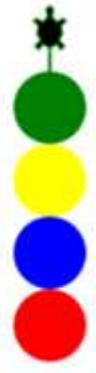
Liste mit Farben:
colors = ["red", "blue", "yellow", "green"]

Liste mit Zahlen:
nb = [4, 2, 6, 3, 7]

Tuppel mit x, y- Koordinaten eines Punktes p:
p(10, 20)

Auf die einzelnen Werte greifst du mit einem Index zu. Das erste Element hat immer den Index 0, das zweite Index 1 usw. Beispielsweise liefert *collors[2]* die Farbe "yellow" oder *p[0]* die Zahl 10. Der Unterschied zwischen Listen und Tupels besteht darin, dass Listen während der Programmausführung verändert werden können, Tupel hingegen nicht. O▶

MUSTERBEISPIELE



Beispiel 1: Liste mit for-Schleife durchlaufen
Die Turtle soll gefüllte Kreise in den Farben rot, blau, gelb und grün zeichnen.

Du speicherst die Farben in der Liste **colors** und verwendest eine**for-Schleife**, um die Farben der Reihe nach auszuwählen.

Dazu gibt es zwei Möglichkeiten.

Du durchläufst die Liste mit einem Index *i.* Das erste Element der Liste hat Index 0. Ist *n* die Anzahl der Listenelemente, (man sagt auch die **Länge der Liste)** so muss der Index i von 0 bis *n-1* laufen. Die Länge n der Liste kannst du mit der eingebauten Funktion **n = len(colors)**bestimmen.

Programm: **[▶ Online-Editor] [▶ WebTigerJython]**

```
from gturtle import *

makeTurtle()
colors = ["red", "blue", "yellow", "green"]
n = len(colors)

for i in range(n):
```

```
    c = colors[i]
    setPenColor(c)
    dot(40)
    forward(40)
```

► In Zwischenablage kopieren

Es gibt eine zweite, kürzere Art, die Liste mit einer **for-Schleife** zu durchlaufen, bei der du keinen Index brauchst. Mit der Schreibweise

```
for c in colors:
```

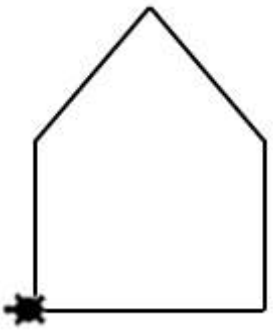
kannst du ein Element um das andere aus der Liste *colors* holen.

Programm: **[► Online-Editor]** **[► WebTigerJython]**

```
from gturtle import *

makeTurtle()
colors = ["red", "blue", "yellow", "green"]

for c in colors:
    setPenColor(c)
    dot(40)
    forward(40)
```

► In Zwischenablage kopieren

Beispiel 2: Tuppels als Liste-Elemente



Ausgehend von der linken unteren Ecke zeichnest du ein Haus, indem du eine Liste **house** mit den Koordinaten der Eckpunkte durchgehst und diese der Reihe nach mit dem Befehl **moveTo(x, y)** verbindest.

Programm: **[► Online-Editor]** **[► WebTigerJython]**

```
from gturtle import*

makeTurtle()
house = [(0, 60), (50, 110), (100, 60), (100, 0), (0, 0)]
for (x, y) in house:
    moveTo(x, y)
```

► In Zwischenablage kopieren

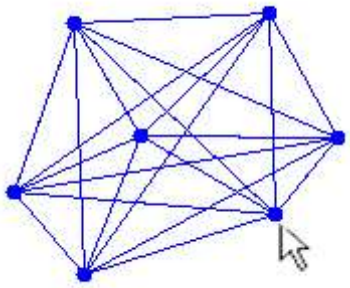
Beispiel 3: Eine dynamische Punkt-Liste

Bisher hast du die Listen im Programmcode definiert. Du kannst aber eine Liste auch erst zu Laufzeit während der Programmausführung erstellen (man sagt auch "dynamisch").

In deinem Programm erzeugst du mit einem Mausklick einen neuen Punkt (x, y) und zeichnest jeweils alle Verbindungslinien von diesem Punkt zu den bestehenden Punkten. Dazu musst du die bestehenden Punkte in einer Liste *points* speichern.

Du erzeugst zuerst mit **points = []** eine leere Liste. Bei jedem Mausklick wird die Funktion *addPoint(x, y)*aufgerufen. Diese zeichnet an der Stelle des Mausklicks einen Punkt und fügt die Koordinaten des Mausklicks in die

Liste *points*. Um den Punkt in die Liste hinzufügen, schreibst du **points.append((x, y))**.



Mit jedem neuen Punkt wird die Funktion **drawLines(x, y)**aufgerufen, die die Verbindungslinien zeichnet. Mit einer for-Schleife gehst du die Liste der bereits vorhandenen Punkte durch und verbindest jedes Element mit dem neu erzeugten Punkt.

Programm: **[► Online-Editor]**

```
from gturtle import *

def drawLines(x, y):
    for (x1, y1) in points:
        setPos(x, y)
        moveTo(x1, y1)

def addPoint(x, y):
    setPos(x, y)
    dot(8)
    drawLines(x, y)
    points.append((x, y))

makeTurtle(mouseHit = addPoint)
hideTurtle()
setPenColor("blue")
points = []
print("Click with the mouse to draw points!")
```

► In Zwischenablage kopieren

MERKE DIR...

In einer Liste kannst du mehrere Werte speichern und über einen Index auf die einzelnen Elemente zugreifen. Beim Aufzählen der Listenelemente verwendest du die eckigen Klammern.

Mit *li.append(x)* fügst du ein neues Element x hinten an der Liste li an.
for n in li: durchläufst du die Liste Element um Element. Eine Liste kann auch dynamisch, während der Programmausführung erzeugt oder modifiziert werden.

Neben den Listen stellt Python zum Speichern von mehreren Werten auch ***Tuppels*** zur Verfügung. Die Elemente eines Tuppels werden in den runden Klammern angegeben. Tuppels werden häufig zum Speichern von Koordinaten (x, y) verwendet.

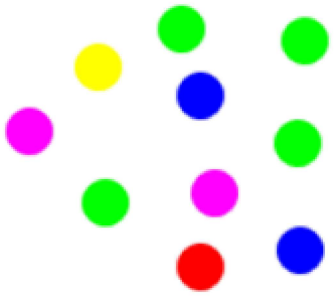
ZUM SELBST LÖSEN

1. Erstelle eine Liste *colors* mit 6 frei gewählten Farben. (**trinket.io/docs/colors**). Zeichne ein 6-Eck, bei dem für jede Seitenfarbe eine Farbe aus deiner Liste colors gewählt wird. Für das Zeichnen der 6-Eck-Seiten wählst du einen breiteren Stift.

2. Mit Hilfe von Zufallszahlen kannst du aus der Liste *colors* zufällig eine Farbe auswählen:

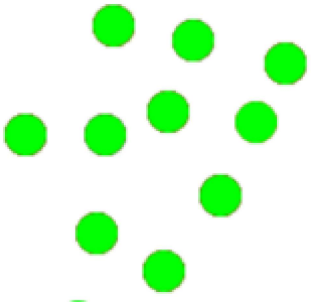
```
from random import randint

colors = ["red", "blue", "yellow", "lime", "magenta"]
i = randint(0, 4)
c = colors[i]
print(c)
```

Schreibe ein Programm so dass bei jedem Mausklick an der Mausposition ein Kreis gezeichnet wird, wobei die Farbe zufällig aus der Liste *colors* gewählt wird.



Wie im Beispiel 3 definierst du ein Callbackfunktion *drawDot()*, die bei jedem Mausklick aufgerufen wird. In dieser Funktion wird die Farbe zufällig gesetzt, die Turtle wird an die Position des Mausklick gesetzt und ein *dot* gezeichnet.

3. Bei jedem Mausklick wird ein neuer Kreis gezeichnet und gleichzeitig alle bereits vorhandene Kreise mit der gleichen, zufällig, aus der Liste *colors* gewählten, Farbe gefärbt.



Nimm das Beispiel 3 als Vorlage. Die Funktion *addPoint(x, y)* wird bei jedem Mausklick aufgerufen. Dabei wird in der Liste points, die zu Beginn leer ist, ein neuer Punkt (x, y) hinzugefügt und anschliessend die Funktion *drawDot()* aufgerufen.

In der Funktion *drawDot()* wählst du eine Farbe und gehst du die Liste *points* durch, um alle Punkte neu zu färben.