11. EREIGNISPROGRAMMIERUNG (EVENTS)

DU LERNST HIER...

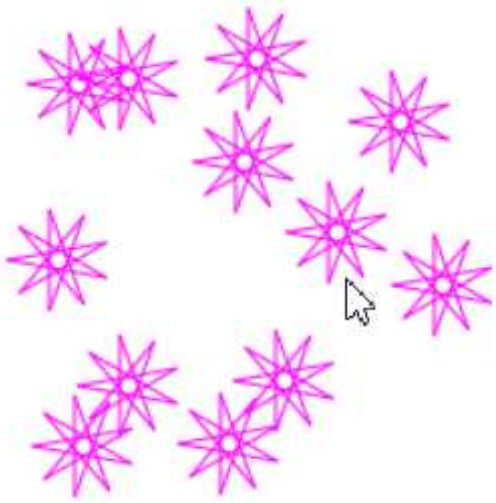
dass die ereignisgesteuerte Programmierung ein wichtiges Programmierkonzept ist. Im Gegensatz zu einem normalen, sequentiell ablaufenden Programm wird dabei eine Funktion, genannt Callbackfunktion oder kurz Callback, beim Auftreten eines Event ausgeführt,, beispielsweise wenn der Benutzer mit der Maus klicket oder eine Tastaturtaste drückt..

MUSTERBEISPIELE ZU DEN MAUSEVENTS

Im ersten Beispiel werden per Mausklick Sterne gezeichnet.

Die Programmiertechnik ist die Folgende: Du schreibst eine Callbackfunktion **onMouseHit(x, y)** (der Name ist frei wählbar). Dort legst du fest, was die Turtle beim Mausklick tun soll. Hier soll sie an der Stelle des Mausklicks, also am Punkt x, y einen Stern zeichnen.

Damit das System weiss, welche Funktion es bei einem Mausevent aufrufen soll, gibst du deren Name als benannten Parameter von **makeTurtle()** an. Damit das Zeichnen schneller erfolgt, versteckst du die Turtle.



Programm: **[► Online-Editor]**

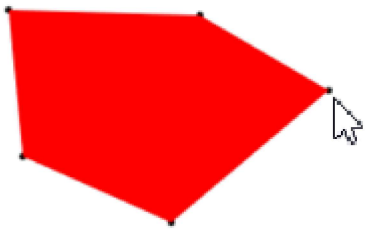
```
from gturtle import *

def drawStar():
    repeat 9:
        forward(40)
        right(160)

def onMouseHit(x, y):
    setPos(x, y)
    drawStar()

makeTurtle(mouseHit = onMouseHit)
hideTurtle()
setPenColor("magenta")
```

► In Zwischenablage kopieren

Typisch für die Eventprogrammierung ist, dass das Hauptprogramm nach setPenColor() zu Ende läuft, der Callback aber immer noch aktiv ist.

Im nächsten Beispiel willst du mit Mausklicks ein Polygon zeichnen. Dabei gibst du zuerst mit einem **Inputdialog** die Anzahl der Ecken des Polygons vor. Danach wählst du per Mausklick die Positionen der Eckpunkte. Bei jedem Klick erscheint ein kleiner Markierungspunkt und nach dem letzten Klick wird das gefüllte Polygon gezeichnet.

Für das Zählen der Mausklicks benötigst du eine Variable **n**, die in der **Callbackfunktion** nach jedem Mausklick um 1 erhöht wird.



Die Variable n ist eine **globale Variable**, da sie einerseits im Hauptprogramm und andererseits in der Callbackfunktion verwendet wird. Damit du den Wert einer globalen Variablen in einer Funktion ändern kannst, musst du sie mit dem Schlüsselwort **global** versehen.

Programm: **[► Online-Editor]**

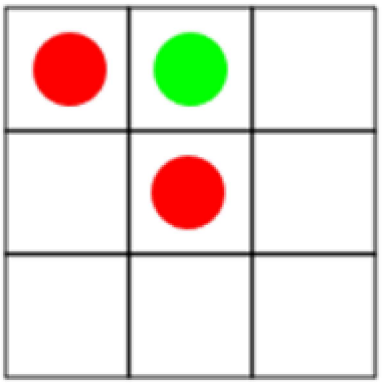
```
from gturtle import *

def drawPoint(x, y):
    global n
    setPos(x, y)
    if n == 0:
        startPath()
    dot(5)
    n = n + 1
    if n == nbCorners:
        fillPath()

n = 0
makeTurtle(mouseHit = drawPoint)
hideTurtle()
nbCorners = inputInt("Gib die Eckenzahl ein:")
setFillColor("red")
```

► In Zwischenablage kopieren

Game Tic-Tac-Toe (light)

Tic-Tac-Toe ist ein bekanntes Spiel, bei dem zwei Spieler abwechslungsweise Kreuze oder Kreise in die Felder eines 3x3-Gitters setzen mit dem Ziel, als erster drei eigene Zeichen in einer horizontalen, vertikalen Linie oder einer der Diagonalen zu haben.

Hier zeichnest du bei statt Kreuze und Kringel einen je nach Spieler einen roten oder grünen Punkt.



Um die Farbe umzuschalten, führst du eine Variable **player**ein, den Spieler identifiziert. Diese musst du in der Callbackfunktion **drawDot(x, y)** wieder als **global**bezeichnen.

Programm: **[► Online-Editor]**

```
from gturtle import *

def square():
    repeat 4:
        forward(50)
        right(90)
```

```
def drawBoard():
    for x in range(3):
        for y in range(3):
            setPos(50 * x, 50 * y)
            square()

def drawDot(x, y):
    global player
    if player == 1:
        setPenColor("red")
        player = 2
    elif player == 2:
        setPenColor("lime")
        player = 1
    setPos(x, y)
    dot(30)

player = 1
makeTurtle(mouseHit = drawDot)
hideTurtle()
drawBoard()
```

► In Zwischenablage kopieren

Es wäre interessant, das Spiel so zu erweitern, dass es von selbst merkt, wenn der ein Spieler gewonnen hat bzw. wenn das Spiel unentschieden ist.

MERKE DIR...

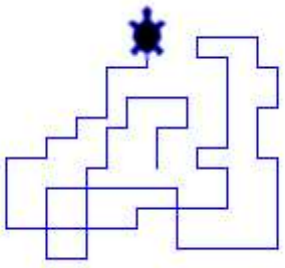
Die eventgesteuerte Programmiertechnik ist daran zu erkennen, dass die Funktionen **onMouseHit(x, y)** , **drawPoint(x, y)** , bzw. **drawDot(x, y)** nirgends von deinem Programm aufgerufen werden. Du gibst dem System den Funktionsnamen an, damit es sie beim Auftreten eines Ereignisses von sich aus aufruft. Solche Funktionen nennt man **Callbackfunktionen** oder kurz **Callbacks**.

Eine Variable, die sowohl im Hauptprogramm wie in einer Funktion verwendet wird, ist eine **globale Variable**. Damit du ihren Wert in der Funktion ändern kannst, muss du sie mit dem Schlüsselwort **global** versehen. Eine Variable die hingegen nur in einer Funktion verwendet wird, nennt man eine **lokale Variable**.

MUSTERBEISPIEL ZU DEN TASTATUREVENTS

Bei den Tastatur-Events wird bei einem Tastendruck ein Event ausgelöst und dabei eine Callbackfunktion aufgerufen, in der du definierst, was beim Tastendruck geschehen soll. Der Callback erhält beim Aufruf durch das System die Information, welche Taste gedrückt wurde. Die Callbackfunktionen musst du im Hauptprogramm registrieren.

In deinem Beispiel verwendest du die Cursortasten, um die Turtle zu steuern.
registerKey('up', moveUp) registriert die Callbackfunktion **moveUp**, die beim Drücken der Cursortaste *Pfeil nach oben* aufgerufen wird. Die Callbacks sind nur aktiv, falls das Turtlefenster den Fokus erhält. Du musst dazu nach dem Programmstart mit einem Mausklick in das Turtlefenster klicken.



In deinem Programm willst du mit den 4 Cursortasten die Turtle im Turtlefenster herumbewegen.

Programm:**[► Online-Editor]**

```
from gturtle import *

def moveLeft():
    setHeading(270)
    forward(5)

def moveRight():
    setHeading(90)
    forward(5)

def moveUp():
    setHeading(0)
    forward(5)

def moveDown():
    setHeading(180)
    forward(5)

makeTurtle()
setPenColor("blue")
registerKey('up', moveUp)
registerKey('down', moveDown)
registerKey('left', moveLeft)
registerKey('right', moveRight)
```

► **In Zwischenablage kopieren**

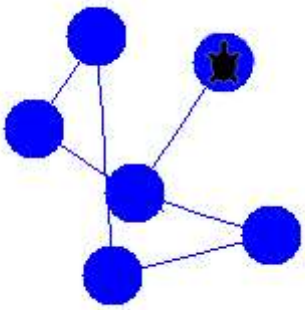
MERKE DIR...

Bei den Tastaturevents wird beim Drücken einer Tastaturtaste eine Callbackfunktion aufgerufen, die du mit dem Befehl **registerKey(key, callback)** registrierst. Als key-Bezeichner kannst du ausser den Kleinbuchstaben "a",..,"z" auch die Cursortasten "up", "down", "left" und "right" verwenden.

ZUM SELBST LÖSEN

1. Die Turtle bewegt sich bei jedem Mausklick an die Position des Mauscursors und zeichnet dort einen gefüllten Kreis.

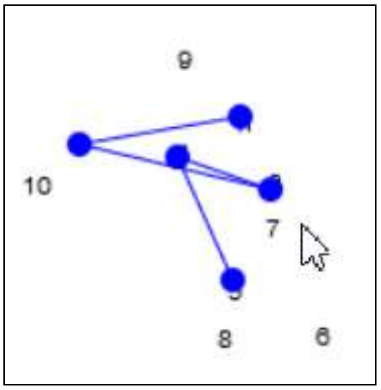
   Anleitung: Verwende den Befehl *moveTo(x, y)*, damit auch die Verbindungslinien sichtbar sind.



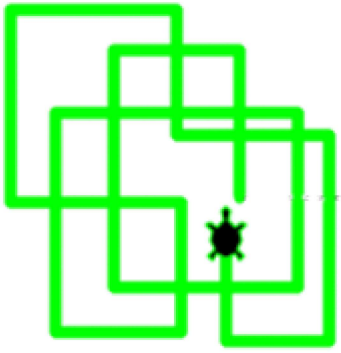
2. Bei jedem Mausklick zeichnet die versteckte Turtle einen gefüllten Stern entstehen. Mit clear("darkblue") färbst du den Hintergrund.

3. Programmiere ein Reaktionsspiel. Dabei muss der Spieler die zufällig positionierten Zahlen 1 - 20 möglichst schnell per Mausklick der Reihe nach verbinden. Hinweis: Für die zufälligen Positionen verwendest du am einfachsten l

setRandomPos(400, 400) .und schreibst die Zahlen mit *label(n)*aus. Der Spieler muss die Zahlen nur ungefähr mit treffen.



4. Snake light:
   Ändere das Programm im Beispiel *Tastaturevents* so, dass sich die Turtle ständig vorwärts bewegt und du mit den Cursortasten nur die Richtung änderst, so dass die Turtle nicht über den Rand des Fensters hinaus läuft.



5*. Erstelle ein **Nim-Spiel** mit 15 roten Steinen. Jeder Spieler kann mit Mausklicks 1, 2 oder 3 Steine entfernen und muss dann jeweils zur Bestätigung auf den gränen Kreis klicken. Dann kommt der andere Spieler zum Zug. Wer den letzten Stein entfernt, hat verloren.

Hinweis: Du kannst die Farbe an der Position des Mausklicks mit *getPixelColorStr()* abfragen. Als ergebnis erhälst du die Farbe als Hex-String (für eine rote Farbe "#ff0000", für eine grüne *#00ff00"). Du merkst also, ob der Benutzer auf einen roten Stein oder auf den grünen Button geklickt hat. Am einfachsten kannst einen Stein wegnehmen, indem du ihn mit einem etwas grössen Dot weiss übermalst.