

## Unterrichts-Ablauf:

Dauer	Thema	Schueler Aktivitaet
3 Minuten	Einfuehrung in das Thema mit "01-intro.py"	Zuhoeren
10 Minuten	Vorstellung von Gueltigkeitsbereichen mit "02-scopes/"	Zuhoeren und wenn noetig Fragen stellen
10 Minuten	Erklaerung des 'return' Befehls mit "02-scopes/04-hoch.png" und "./03-loesung.py"	Zuhoeren und wenn noetig Fragen stellen
15-20 Minuten	Experimentieren mit "./04-example/01-beispiel.py"	Verschiedende Anwendungen von Gueltigkeitsbereichen und dem 'return' Befehl ausprobieren
10 Minuten	Vorstellung eines echten Beispiels anhand von "./04-example/example.py"	Zuhoeren und wenn noetig Fragen stellen
≈ 10 Minuten	Zeitpuffer fuer moeglichen Verzug	

# Code:

## 01-intro.py:

```
from turtle import *

def rect(laenge):
    for ii in range(2):
        forward(laenge)
        right(90)
        forward(laenge)
        right(90)

def dach(laenge):
    for jj in range(3):
        forward(laenge)
        right(120)

def change_pos(laenge, back):
    if back:
        left(30)
        backward(laenge)
    else:
        forward(laenge)
        right(30)

def haus(laenge):
    rect(laenge)
    change_pos(laenge, back=False)
    dach(laenge)
    change_pos(laenge, back=True)

def strasse(laenge, num_hauss):
    for kk in range(num_hauss):
        haus(laenge)
```

```
        right(90)
        forward(laenge)
        left(90)

        laenge += 20
        laenge += 20

def main():
    left(90)
    speed(100000)

    laenge = 100
    laenge = 50

    strasse(laenge, 5)

    #haus(laenge)
    done()
if __name__ == "__main__":
    main()
```

## 02-scopes/

### 01-scopes.py

```
x = 7

def outer():
    y = 5

    def inner():
        z = 6

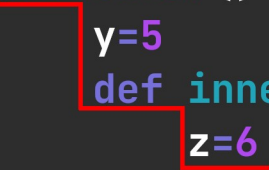
        print("x=", x)
        print("y=", y)
        print("z=", z)
    inner()

    print("x=", x)
    print("y=", y)
    print("z=", z)
outer()

print("x=", x)
print("y=", y)
print("z=", z)
```

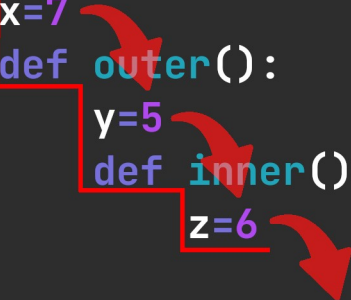
02-treppe.png

```
x=7
def outer():
    y=5
    def inner():
        z=6
```

A diagram showing nested function definitions. A red bracket on the left side groups the lines 'def outer():', 'y=5', and 'def inner():'. A second red bracket is nested inside the 'outer' function, grouping the lines 'def inner():' and 'z=6'. This illustrates the scope levels for the variables x, y, and z.

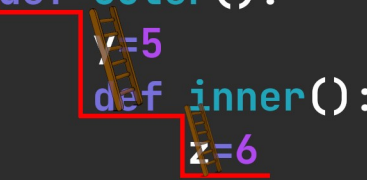
03-runter.png

```
x=7
def outer():
    y=5
    def inner():
        z=6
```

A diagram showing nested function definitions. Red arrows indicate the return flow: one arrow points from the end of the 'inner' function block to the end of the 'outer' function block, and another arrow points from the end of the 'outer' function block to the line 'def outer():'. This illustrates the return path when a function is called and returns.

## 04-hoch.png

```
x=7
def outer():
    y=5
    def inner():
        z=6
```

A diagram illustrating nested function scopes. It shows three levels of indentation: the global scope (x=7), the outer function scope (y=5), and the inner function scope (z=6). Red brackets and ladders indicate the scope boundaries and the flow of execution between them.

## ./03-loesung.py

```
from turtle import *

def rect(laenge):
    for ii in range(2):
        forward(laenge)
        right(90)
        forward(laenge)
        right(90)

def dach(laenge):
    for jj in range(3):
        forward(laenge)
        right(120)

def change_pos(laenge, back):
```

```
    if back:
        left(30)
        backward(laenge)
    else:
        forward(laenge)
        right(30)

def haus(laenge):
    rect(laenge)
    change_pos(laenge, back=False)
    dach(laenge)
    change_pos(laenge, back=True)

def strasse(laenge, num_hauss):
    for kk in range(num_hauss):
        haus(laenge)

        right(90)
        forward(laenge)
        left(90)

        laenge += 20
        laenge += 20

    return laenge

def main():
    left(90)
    speed(100000)

    laenge = 100
    laenge = 50

    laenge = strasse(laenge, 5)

    haus(laenge)
    done()
```

```
if __name__ == "__main__":  
    main()
```

## 04-example/

### 01-bleispiel.py

```
num1 = 5  
num2 = 6  
  
def addition():  
    solution = num1 + num2  
    print(solution)  
  
    return solution  
  
print(addition())
```

### example.py

```
import os.path  
import time  
  
from forex_python.converter import CurrencyRates  
from forex_python.bitcoin import BtcConverter  
  
from openpyxl import Workbook, load_workbook  
from datetime import date, datetime  
  
def save_to_xlsx(BTC_USD, BTC_EUR, USD_EUR):  
    workbook = load_workbook('data.xlsx')  
    sheet = workbook.active  
  
    date = str(datetime.now())
```



```
    row = (date, BTC_USD, BTC_EUR, USD_EUR)
    sheet.append(row)

    workbook.save(filename="data.xlsx")

def setup_xlsx():
    workbook = Workbook()
    sheet = workbook.active

    sheet["A1"] = "date"
    sheet["B1"] = "1 BTC in USD"
    sheet["C1"] = "1 BTC in EUR"
    sheet["D1"] = "1 USD in EUR"

    workbook.save(filename="data.xlsx")

def usd_in_eur():
    c = CurrencyRates()
    EUR = c.get_rate('USD', 'EUR') #convert USD to EURO
    return(EUR)

def btc_in_usd():
    b = BtcConverter()
    USD = b.get_latest_price('USD') #convert BTC to USD
    return(USD)

def btc_in_eur():
    b = BtcConverter()
    EUR = b.get_latest_price('EUR') #convert BTC to EUR
    return(EUR)

def main():
    if os.path.isfile('data.xlsx'):
        pass
    else:
        setup_xlsx()
```

```
while True:

    BTC_USD = round(btc_in_usd(), 2)
    USD_EUR = round(usd_in_eur(), 2)
    BTC_EUR = round(btc_in_eur(), 2)

    save_to_xlsx(BTC_USD, BTC_EUR, USD_EUR)

    # Sleep for 24 hours
    # (For testing purposes you can set it to a lower number)
    time.sleep(24 * 60 * 60)

if __name__ == '__main__':
    main()
```