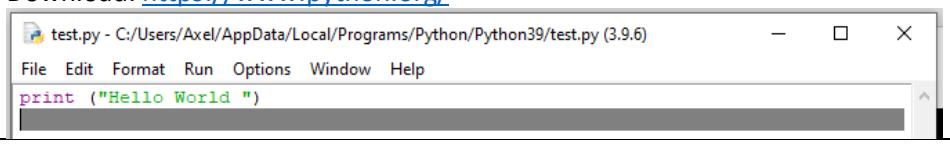
	Grundlagen
Entwicklungsumgebung IDLE	Die Abkürzung IDLE steht für „Integrated Development and Learning Environment“ Download: https://www.python.org/ 
Kommentar	# Mit Hashtag ist die Zeile ein Kommentar
Textausgabe	print ("Hello World ") #Textausgabe print(7) # Ausgabe einer Zahl

Zahlen	print(7) # Ganzzahl (z.B. 7)
	print(3.4) # Gleitkommazahlen (z.B. 3.4)

String	print("Axel "); print('Chobe ') #Zeichenkette ausgeben
	print('Der " PC " ist schwarz') # Ausgabe von Hochkomma (Shift #)
	print("""Jetzt kann der Text #Ausgabe von mehrzeiligem Text
	geteilt werden """)

Operatoren	print(8 + 4); print(7-4); print(8 * 4); print(8 / 4) #hier auch Kommazahlen print("Hallo " + " Welt ") #Strings werden verkettet
Variablen	age = 27 #Variablenname muss mit Buchstabe oder Unterstrich beginnen print(age) # oder vname = “Axel”

Datentypen (Basis)	int_variable = 10 # Int (Integer) = Ganzzahlen
	float_variable = 7.456 # Float = Gleitkommazahlen
	string_variable_1 = "Hallo Axel " # STR (String) = Zeichenkettw

Datenty bool	print(1 < 5); Ausgabe – True (oder False) # Darstellung von Wahrheitswerten
input	userN = input ("Wie ist Dein Name? ") # Eingabe wir in userN gespeichert print("Hallo " + userN) #Achtung: Jede Eingabe wird als String gespeichert!
type	zahl1 = input("Zahl ") # ermittelt den Datentyp print(type(zahl1)) # Ausgabe <class ‚str’>

typecast	wert1 = input("Zahl1 ") # einlesen Zahl1 als String
	wert2 = input("Zahl2 ") # einlesen Zahl2 als String
	print (wert1 + wert2) # Ausgabe falsch; hier Stringverkettung
	print (int(wert1) + int(wert2)) # Ausgabe richtig, hier vorher Umwandlung
	#für Kommazahlen den typecast float benutzen: wert1 = float(input(„zweite Zahl“))

Zuweisungsoperator	a = 1 # weist Werte von rechts nach links zu
	print(a) # Ausgabe 1
	a = a+1 # Wert in der Zuweisung um 1 erhöhen
	a += 1 # andere Schreibweise für Erhöhung
	print(a) # Ausgabe 3

Vergleichsoperatoren	print(3 < 4) # Ergebnis wahr -> Ausgabe True print(3 == 4) # Ergebnis falsch -> Ausgabe False # < kleiner als; > größer als; <= kleiner gleich als; >= größer gleich als; # == gleich; != ungleich;
	Kontrollstrukturen

If Anweisung (Bedingung)
if
elif
else
	age = int(input("Alter: ")) # Ergebnis „str“ wird sofort in „int“ umgewandelt
	if age < 18: # durch Doppelpunkt Einrückung der nächsten Zeile
	print("unter 18") # für alle Anweisungen der if-Abfrage
	elif age == 18: # für mehrere Bedingungen
	print("genau 18")
	else: # alternative Ausgabe wenn age > 18
	print("über 18")

Logische Operatoren
Verschachtelung von if-Abfragen or, and, not
	wert1 = int(input("erste Zahl "))
	wert2 = int(input("zweite Zahl "))
	if wert1 == 3 and wert2 == 15: # and -> und
	print("Im Lotto gewonnen ") # or -> oder
	else: # not -> nicht
	print("leider verloren ")

while Schleife
 	
arbeitet solange das
Ergebnis True ist
	durchgang = 1 # Variable Wert zuweisen
	while durchgang < 3: # Kopf prüft ob durchgang <3
	  print(durchgang) # Körper
	  durchgang = durchgang + 1 # Körper durchgang wird um 1 erhöht
	print("Schleife Ende ") # Schleifenende da Bedingung erreicht
	while True: # Endlosschleife erzeugen Ende mit CTR+C
	  print("Dauerschleife ") # Inhalt der Endlosschleife

for Schleife
Zählerschleife (range)
	print("Schleife beginnt ") # Testausgabe
	for i in range(10): # Beginn d. Schleife, Erhöhung n. jedem Durchlauf
	  print("Schleife Nummer ", i) # print(element)
	print("Schleife beendet ! ") # Schleifenende, da Bedingung erfüllt
	for i in "Hallo Welt ": # auch String möglich
	  print(i) # buchstabenweise Ausgabe
	for i in [1, 3, 5]: # auch Liste möglich
	  print(i) # Ausgabe der Listenwerte
	for i in(-2, 4,2): # -2 (Beginn), 4 (Ende), 2(Schrittweite)
	  print(i) # Ausgabe -2, 0, 2

Listen [ ]
können mehrere
Variablen speichern
Tupel ( ) wie Liste
nur nicht veränderbar
	wert = [3, 17, 23, 57] # definieren einer Liste
	print(wert[0]) # Ausgabe des ersten Listenwertes
	wert[0]=6 # ändern des ersten Listenwertes
	print(wert[0]) # Ausgabe des geänderten 1. Listenwertes
	wert.append(12) # Liste erweitern um Wert 12 am Ende
	print(wert[-1]) # Ausgabe des letzten Listenwertes(-2 Vorletzte…)

zusätzliche interne Module (Bibliotheken)(Pakete)	
Zufall	import random # Modul random importieren print(random.randint(0,10)) # Ausgebe einer Zahl zw. 0 und 10
Mathematik	import math # Modul math importieren print(math.sqrt(81)) #Aufruf über vollen Namen, hier Wurzel 81 oder from math import sin, pi #Import einzelner Funktionen/Methoden print(sin(10)) #Aufruf ohne math

Time	import time # Modul time importieren
	zeitD = time.strftime("%d.%m.%Y ") # Zuweisung von Tag, Monat, Jahr
	print(zeitD) # Ausgabe
	zeitT = time.strftime("%H:%M:%S ") # Zuweisung von Stunde, Minute, Sekunde
	print(zeitT) # Ausgabe

Zusätzliche externe Module einbinden	
	Die entsprechende py-Datei in das Verzeichnis kopieren: Benutzer/ Name/ AppData/ Local/ Programms/ Python/ Python39/ Lib
Funktionen	
Funktion (Unterprogramm)	def ansage(): # Erzeugung einer Funktion, Einleitung mit def print("Hallo Benutzer ") # Inhalt automatisch eingerückt ansage () # Aufruf der Funktion

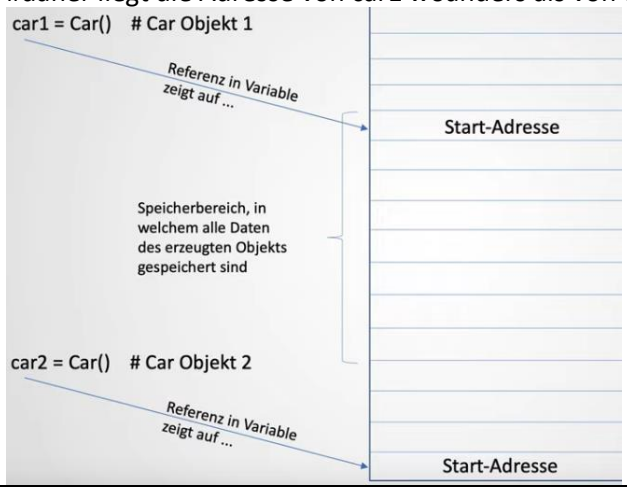
Funktion mit Parametern	def ansage(name): # Parameter einer Funktion sind nur innerhalb
	  print("hallo " + name) # der Funktion gültig!
	ansage("User ") #Aufruf der Funktion mit Parameterübergabe od
	def plus(wert1, wert2): # Übergabe von mehreren Parametern
	  print(int(wert1) + int(wert2)) # Ausgabe der Summe beider Parameter
	plus(3,4) # Aufruf der Funktion mit zwei Parametern

Funktion mit Rückgabe	def maximum(a, b): # Erzeugung einer Funktion
	  if a < b: # Ist a Kleiner b
	  return b # dann Rückgabe Wert b
	  else: # sonst
	  return a # Rückgabe Wert a
	print(maximum(4,6)) # Aufruf d. Funktion und Ausgabe d. Ergebnisses
	#Jede Funktion gibt einen Wert zurück (none; Datentyp: NonType)

	Objektorientierung (Objekte modellieren bzw. eigene Datentypen bauen)

Klasse erstellen
Bauplan mit
eigenem Datentyp
	class Car: #Schlüsselwort und Bezeichner
	  def __init__(self): # Intitfunktion, in Klasse heißt es Methoden
	  self.car_typ = None # z.B.Attribut für Hersteller #erkennbar an (self)
	  self.horse_power = None # z.B. für PS (None –noch kein Wert zugewiesen)
	  self.color = None # z.B. für Farbe

Objekt (Instanz)	Car() # Objekt wird erzeugt, Rückgabe ist die Referenz auf das Objekt
	car1 = Car() # welches in einer Variable gespeichert wird (weiter car2 = Car())
	print(car1.car_typ) #Ausgabe ist None, weil kein Wert vorhanden ist
	car1.car_typ = "Citroen " # Zuweisen eines Wertes für Hersteller
	print(car1.car_typ) # Ausgabe -> Citroen

Self Parameter	#beim Erzeugen eines Objektes wird Speicherplatz reserviert (z.B.Car()) #in der Variable (z.B.car1) wird nur die Speicheradresse gespeichert (Referenz) print(car1) #Ausgabe -> <__main__.Car object at 0x7fd42820d2b0> #self liefert die Referenz des aktuellen ausführenden Objektes zurück #daher liegt die Adresse von car1 woanders als von car2 

Methoden in Klassen
(Funktionen der Objekte)
	class Car: #Schlüsselwort und Bezeichner
	  def __init__(self): # Intitfunktion, in Klassen heißt es Methoden
	  self.x_position = 5 #zusätzliches Attribut für die kommende Funktion
	  self.y_position = 5 #zusätzliches Attribut für die kommende Funktion
	  def drive(self, x, y): #eine Funktion des späteren Objektes
	  self.x_position += x # Erhöhung der x Position um übergebenen Wert
	  self.y_position += y # Erhöhung der y Position um übergebenen Wert
	car1 = Car() # Erzeugung eines Objektes aus der Klasse
	print(car1.x_position) #Ausgabe -> 5 (Kontrolle der Position)
	print(car1.y_position) #Ausgabe -> 5 (Kontrolle der Position)
	car1.drive(5, 10) #Aufruf der Funktion um Position zu verändern)
	print(car1.x_position) #Ausgabe -> 10 (Kontrolle der Position)
	print(car1.y_position) #Ausgabe -> 15 (Kontrolle der Position)