```python
# Einstieg Funktionen

# ----- Hilfsfunktion für Aufgabenüberschrift
# ----- (nicht Teil der Übung; dient lediglich der Übersicht)
aufgaben_nr = 0


def aufgabe(text=''):
    global aufgaben_nr
    aufgaben_nr += 1

    ueberschrift = f' Aufgabe {aufgaben_nr} '
    if text:
        ueberschrift += f'({text}) '
    print(f"\n{ueberschrift:=^80}\n")


# Schreiben Sie eine Funktion quadrat, die eine Zahl x als Parameter erwartet
# und die das Quadrat der Zahl als Rückgabewert liefert
# Testen Sie das Programm mit x=10 und geben Sie das Ergebnis aus.
aufgabe()


def quadrat(x):
    return x**2

x = 10
e = quadrat(x)
print(e)

# Erweitern Sie die Funktion so, dass sie eine Fehlermeldung ausgibt,
# falls kein Wert übergeben wurde; in diesem Fall soll die Funktion
# als Rückgabewert None liefern
aufgabe()


def quadrat(x=None):
    if x is None:
        print("FEHLER: Argument x fehlt")
        return_wert = None
    else:
        return_wert = x**2
    return return_wert

print(quadrat(12))
print(quadrat())

# Alternative (?) (1/2)
def quadrat(x=None):
    if x is None:
        print("FEHLER: Argument x fehlt")
    else:
        x = x**2
    return x


# Einfache Alternative (2/2) ohne Fehlermeldung
def quadrat(x=None):
    return x**2 if x is not None else None

print(quadrat(12))
print(quadrat())

# Schreiben Sie eine Funktion addition(). Der Funktion sollen mindestens
# zwei Argumente übergeben werden. Die Funktion soll die Summe der übergebenen
# Werte (beliebig viele) zurückliefern.
# Geben Sie am Anfang der Funktion die übergebenen Argumente aus, damit Sie
# die Form der Übergabe der einzelnen Parametertypen sehen können.
# Testen Sie die Funktion mit den Aufrufen: addition(3,4), addition(4,5,6,7,8)
aufgabe()

```

```
72
73  def addition(a, b, *weitere):
74      print(a, b, weitere)  # nur zur Verdeutlichung
75      summe = a + b
76      for zahl in weitere:
77          summe += zahl
78
79      # Alternative zur Summenbildung mit Funktion sum
80      summe = a + b + sum(weitere)
81
82      # weitere Alternative zur Summenbildung mit sum
83      summe = sum(weitere, a+b)  # s. Dokumentation der Fkt. sum
84
85      return summe
86
87  print(addition(3, 4))  # = 7
88  print(addition(4, 5, 6, 7, 8))  # = 30
89
90  # einfache Variante
91  def addition(a, b, *weitere):
92      return sum(weitere, a+b)
93
94  print(addition(3, 4))  # = 7
95  print(addition(4, 5, 6, 7, 8))  # = 30
96
97
98  # weitere einfache Variante
99  def addition(*args):
100     return sum(args) if len(args) >= 2 else None
101
102 print(addition())
103 print(addition(3, 4))  # = 7
104 print(addition(4, 5, 6, 7, 8))  # = 30
105
106 # Gegeben sei eine quadratische Funktion der Form f(x) = a*x^2 + b*x + c.
107 # Implementieren Sie die Funktion quad_func(x, a, b, c) entsprechend und
108 # testen Sie die Funktion mit den Werten a=1,b=2,c=3 und x=[0;10].
109 aufgabe()
110
111
112 def quad_func(x, a, b, c):
113     return a*x**2 + b*x + c
114
115
116 for x in range(11):
117     print(quad_func(x, 1, 2, 3))
118
119
120 # Erweitern Sie quad_func so, dass a,b und c optional werden und den
121 # Standardwert 1 bekommen. Testen Sie das Programm für x =[0;10]
122 aufgabe()
123
124
125 def quad_func(x, a=1, b=1, c=1):
126     return a*x**2 + b*x + c
127
128 for x in range(11):
129     print(quad_func(x))
130
131 # Wie sieht der Funktionsaufruf für x=10 aus, wenn Sie nur den Parameter
132 # a auf 5 setzen wollen (b und c sollen den Standardwert bekommen)?
133 # Wie sieht der Aufruf aus, wenn Sie nur den Parameter c auf 5 ändern wollen?
134 aufgabe()
135
136 print(quad_func(10, 5))    # a = 5 (b und c bekommen Standardwert zugewiesen)
137 print(quad_func(10, c=5))  # c = 5 (a und b bekommen Standardwert zugewiesen)
138
139 # Speichern Sie die Werte für die Parameter a=2, b=3 und c=4 in einem Tupel
140 # mit der Bezeichnung parameter. Rufen Sie die Funktion für x=10 und die
141 # gerade genannten Werte der parameter auf.
142 aufgabe()
```

```
143
144 parameter = 2, 3, 4
145 print(quad_func(10, *parameter))
146
147 # Wie würden die Definition der Variable parameter sowie der Funktionsaufruf
148 # aussehen, wenn Sie die parameter in Form eines Dictionary speichern würden?
149 aufgabe()
150
151 # Definition der Parameter
152 parameter = {'a': 2, 'b': 3, 'c': 4}
153 # Alternative:
154 parameter = dict(a=2, b=3, c=4)
155
156 print(quad_func(10, **parameter))
157
158
159 # Erweitern Sie die Funktion addition() aus Aufgabe 3, so dass die Addition
160 # nur dann ausgeführt wird, wenn alle übergebenen Argumente entweder ganze
161 # Zahlen und/oder Fließkommazahlen sind. Bei der Übergabe anderer Datentypen
162 # soll eine Fehlermeldung ausgegeben und der Rückgabewert None zurückgeliefert
163 # werden.
164
165 aufgabe()
166
167
168 # Einfache Variante: schrittweise Überprüfung auf Gültigkeit der Datentypen
169 def addition(a, b, *weitere):
170     summe = None
171     print("Argumente: a={}, b={}, weitere={}".format(a, b, weitere))
172     if (type(a) == int or type(a) == float) and\
173        (type(b) == int or type(b) == float):
174             # a und b haben korrekte Datentypen; nun die weiteren Typen prüfen
175             for element in weitere:
176                 if not (type(element) == int or type(element) == float):
177                     # falscher Datentyp entdeckt; Abbruch der Überprüfung
178                     break
179             else:
180                 # alle Typen korrekt; Summe kann berechnet werden
181                 summe = a + b + sum(weitere)
182     if summe is None:
183         print("FEHLER: Verwendung unerlaubter Datentypen")
184     return summe
185
186 print(addition(3, 4))
187 print(addition(3, 4.0))
188 print(addition(1, 2, 3.0, 4j))
189
190
191 # Alternative, schönere Varianten
192
193 # Einfache Variante: schrittweise Überprüfung auf Gültigkeit der Datentypen
194 def addition(a, b, *weitere):
195     print("Argumente: a={}, b={}, weitere={}".format(a, b, weitere))
196     for element in (a, b, *weitere):  # '*weitere' möglich ab Python 3.5.2
197         if type(element) not in (int, float):
198             # falscher Datentyp entdeckt; Abbruch der Überprüfung
199             print("Falscher Datentyp: {} ({})".format(type(element), element))
200             summe = None
201             break
202     else:
203         # alle Typen korrekt; Summe kann berechnet werden
204         summe = sum(weitere, a+b)
205
206     return summe
207 #
208 # print(addition(3, 4))
209 # print(addition(3, 4.0))
210 # print(addition(1, 2, 3.0, 4j))
211
212
213 # Alternative mit Mengen (alle verwendeten Datentypen in einem set speichern
```

```
# und anschließend prüfen, ob es neben int und float weitere Typen enthält)
# (zur Erinnering an das Thema von Übungsblatt 6: Mengen)
def addition(a, b, *weitere):
    print("Argumente: a={}, b={}, weitere={}".format(a, b, weitere))
    typen = {type(a), type(b)} | {type(e) for e in weitere}
    print("Datentypen der Argumente:", typen)
    if typen <= {int, float}:
        summe = sum(weitere, a+b)
    else:
        print("(1) FEHLER: Verwendung unerlaubter Datentypen")
        summe = None

    return summe

print(addition(1, 2, 3.0, 4j))


# Alternative mit Generator und der Funktion 'all'
# Diese Variante bricht die Überprüfung der Argument-Typen direkt ab, sobald
# ein nicht erlaubter Datentyp gefunden wird.
# Hinweis: statt Verwendung eines Generators mit all((...for...in...)) würde
# auch die normale List Comprehension mit all([...for...in...]) funktionieren
def addition(a, b, *weitere):
    if all(type(e) in (int, float) for e in (a, b, *weitere)):
        return sum(weitere, a+b)
    else:
        print("(2) FEHLER: Verwendung unerlaubter Datentypen")
        return None


print(addition(3, 4))
print(addition(3, 4.0))
print(addition(1, 2, 3.0, 4j))

```