ETH zürich
Departement Informatik

Programmieren und Problemlösen

Dr. Dennis Komm
https://lec.inf.ethz.ch/ppl/
16. April 2021

Übungsblatt 2
Lösungsvorschläge

Aufgabe 1

Schreiben Sie ein Programm, das das folgende Muster ausgibt:

```
( 1 0 )
( 2 0 ) ( 2 1 )
( 3 0 ) ( 3 1 ) ( 3 2 )
( 4 0 ) ( 4 1 ) ( 4 2 ) ( 4 3 )
( 5 0 ) ( 5 1 ) ( 5 2 ) ( 5 3 ) ( 5 4 )
( 6 0 ) ( 6 1 ) ( 6 2 ) ( 6 3 ) ( 6 4 ) ( 6 5 )
( 7 0 ) ( 7 1 ) ( 7 2 ) ( 7 3 ) ( 7 4 ) ( 7 5 ) ( 7 6 )
( 8 0 ) ( 8 1 ) ( 8 2 ) ( 8 3 ) ( 8 4 ) ( 8 5 ) ( 8 6 ) ( 8 7 )
( 9 0 ) ( 9 1 ) ( 9 2 ) ( 9 3 ) ( 9 4 ) ( 9 5 ) ( 9 6 ) ( 9 7 ) ( 9 8 )
```

Lösung

```
for i in range(0, 10):
    for j in range (0, 10):
        if i > j:
            print("(", i, j, ")", end = " ")
    print()
```

Aufgabe 2

Was ist die Ausgabe des folgenden Programms?

```
a = 5
b = 10
print(a == b)
print(a != b)
print(True == (a == b))
print(True == (a != b))
print((True == (a != b)) or (True == (a == b)))
print(True and False and True)
```

```
print(True and False and False or True)
print((True and False) and (False or True))
print(True and (False and False) or True)
```

Lösung

```
False
True
False
True
True
False
True
False
True
```

Entscheidend für diese Aufgabe ist es, die Präzedenzen der Operatoren `and`, `or`, `==` und `!=` zu kennen.

Man kann diese zum Beispiel unter https://www.programiz.com/python-programming/precedence-associativity nachschlagen.

Aufgabe 3

Setzen Sie in der folgenden Berechnung alle Klammern, um die Präzedenzen der verwendeten Operatoren zu verdeutlichen.

Tipp: Sie können die Präzedenzen der Operatoren zum Beispiel auf https://www.programiz.com/python-programming/precedence-associativity finden.

```
e = True
f = False
g = - 2 ** 8 / 9 + 20 <= 250 and not e or f
```

Lösung

```
e = True
f = False
h = (((((- (2 ** 8)) / 9) + 20) <= 250) and (not e)) or f
```

Aufgabe 4

Schreiben Sie den Funktionskörper einer Funktion, die eine Zahl `n` als Eingabe erhält und alle Zahlen zwischen 0 und `n` ausgibt, welche durch 7 teilbar sind. Verwenden Sie dazu eine `while`-Schleife.

```
def meine_funktion(n):
    # Ihr Code
```

Beispiel: Wird `meine_funktion(49)` aufgerufen, soll Folgendes ausgegeben werden:

```
0 7 14 21 28 35 42 49
```

Lösung

```
def meine_funktion(n):
    j = 0
    while j <= n:
        if j % 7 == 0:
            print(j, end=" ")
        j += 1
```

Aufgabe 5

Schreiben Sie den Funktionskörper einer Funktion, die zwei Zahlen `x` und `y` als Eingabe erhält und die kleinste Zahl findet, die sowohl durch `x` als auch durch `y` teilbar ist. Die Funktion soll diese Zahl zurückgeben.

Lösen Sie diese Aufgabe, ohne eine Multiplikation zu verwenden.

```
def meine_zweite_funktion(x, y):
    # Ihr Code
```

Lösung

```
def meine_zweite_funktion(x, y):
    i = 1
    while True:
        if i % x == 0 and i % y == 0:
            return i
        i += 1
```

Diese Funktion inkrementiert eine Zahl `i` in einer `while`-Schleife. In jedem Durchgang wird geprüft, ob `i` sowohl durch `x` als auch durch `y` teilbar ist. Falls dies der Fall ist, wird `i` mit `return i` zurückgegeben und der Funktionsaufruf beendet.

Beachten Sie, dass die Funktion für keine Eingabe unendlich lange ausgeführt wird, da die Bedingung in Zeile 4 spätestens für $\texttt{i} = \texttt{x} \cdot \texttt{y}$ erfüllt ist.

Aufgabe 6

In der folgenden Funktion ist ein Logikfehler passiert. Finden Sie den Fehler und korrigieren Sie die Funktion.

```
def falsche_funktion(a, b):
    if a and b != 1:
        print("Weder a noch b sind gleich 1.")
    else:
        print("Entweder a oder b oder sogar beide sind gleich 1.")
```

Lösung

```
def korrekte_funktion(a, b):
    if a != 1 and b != 1:
        print("Weder a noch b sind gleich 1.")
    else:
        print("Entweder a oder b oder sogar beide sind gleich 1.")
```

Diese Funktion soll offensichtlich überprüfen, ob beide Zahlen nicht eins sind. Da `!=` eine höhere Präzedenz hat als `and` wird im gegebenen Programm nur geprüft ob `b != 1` ist und dann dieses Resultat mit `a` verundet.

Aufgabe 7

Erklären Sie, weshalb das folgende Programm die gezeigte Ausgabe produziert.

```
a = 100
b = 200

def fun1():
    print(a, b)

def fun2():
    a = 2
    print(a, b)

def fun3():
    a = 6
    b = 10
    print(a, b)

fun1()
fun2()
fun3()
```

Ausgabe:

```
100 200
2 200
6 10
```

Lösung

In `fun1()` werden die beiden globalen Variaben `a` und `b` ausgegeben. In `fun2()` wird die globale Variable `a` durch eine lokale Variable `a` überschattet, während `b` weiter die globale Variable bezeichnet. In `fun3()` bezeichnen wiederum beide Variablen `a` und `b` lokale Variablen.

Aufgabe 8

Erklären Sie, weshalb `fun4()` nicht ausführbar ist, aber `fun5()` funktioniert.

```
a = 100
b = 200

def fun4():
    a += 1
    b += 1
    print(a, b)

def fun5():
    a = 0
    b = 10
    a += 1
    b += 1
    print(a, b)
```

Lösung

Die Idee von `fun4()` ist vermutlich, die globalen Variablen `a` and `b` zu überschreiben. Leider weiss `fun4()` nicht, dass `a` und `b` keine lokalen Variablen darstellen sollen. Da `a += 1` äquivalent ist zu `a = a + 1` wird hier von `a` gelesen bevor `a` einen Wert zugewiesen bekommt. Es kommt also zu einem Fehler. Dasselbe gilt für `b`. Falls Sie diese Funktion allerdings laufen lassen, werden Sie nur den Fehler für `a` sehen, da dieser zuerst auftritt.

In `fun5()` wird den lokalen Variablen erst ein Wert zugewiesen, bevor von ihnen gelesen wird. Es kommt somit zu keinem Fehler.

Möchte man, dass `fun4()` die globalen Variablen verändert, kann man `fun4()` wie folgt anpassen:

(Die `print()`-Anweisungen werden hier nur verwendet, um zu demonstrieren, dass sich der Wert der globalen Variablen ändert.)

```
a = 100
b = 200
# Ausgeben der globalen Variablen
print(a, b)

def fun4():
    global a
    global b
    a += 1
    b += 1
    print(a, b)

# Aufrufen der Funktion fun4()
fun4()

# Erneutes Ausgeben der globalen Variablen
print(a, b)
```

Dies würde dann Folgendes ausgeben:

```
100 200
101 201
101 201
```