LU06a - Selektion

Lernziele

- Die Idee einer Selektion kennen und wissen, wie man ein Programm mit optionalen Operationen durch die Verwendung von bedingten Anweisungen erstellt.
- Sich mit den in Selektionen häufig verwendeten Vergleichs- und logischen Operatoren vertraut machen.
- Die Reihenfolge der Ausführung einer Selektion kennen und wissen, dass das Parsen einer Selektion bei der ersten Bedingung aufhört, deren Aussage als wahr bewertet wird.

Einführung

Unsere Programme sind bisher linear verlaufen. Mit anderen Worten, die Programme wurden von oben nach unten ohne größere Überraschungen oder bedingtes Verhalten ausgeführt. In der Regel möchten wir jedoch bedingte Logik in unsere Programme einbauen. Damit meinen wir Funktionen, die auf die eine oder andere Weise vom Zustand der Programmvariablen abhängig sind.

Um beispielsweise die Ausführung eines Programms auf der Grundlage von Benutzereingaben zu verzweigen, müssen wir eine so genannte Selektion verwenden. Die einfachste Selektion sieht etwa so aus.

```
print('Hello, world!')
if True:
    print('This code is unavoidable!')

Hello, world!
This code is unavoidable!
```

Eine Selektion beginnt mit dem Schlüsselwort `if`, gefolgt von einer Bedingung, die ausgewertet wird, wenn die Selektion erreicht wird. Das Ergebnis der Auswertung ist ein boolescher Wert. Oben wurde keine Auswertung vorgenommen. Stattdessen wurde in der bedingten Anweisung explizit ein boolescher Wert (True) verwendet.

Auf die Bedingung folgt ein Block, der darunter eingerückt ist. Der Quellcode innerhalb des Blocks wird ausgeführt, wenn der Ausdruck innerhalb der Klammern den Wert True ergibt.

Schauen wir uns ein Beispiel an, bei dem wir in der bedingten Anweisung Zahlen vergleichen.

```
number = 11
if (number > 10):
    print('The number was greater than 10')

The number was greater than 10
```

Wenn der Ausdruck in der Selektion als `wahr` bewertet wird, wird die Ausführung des Programms zu dem durch die Selektion definierten Block fortgesetzt. Im obigen Beispiel lautet die Bedingung „wenn die Zahl in der Variablen größer als 10 ist“. Ist der Ausdruck hingegen `falsch`, wird die Anweisung nach dem eingerückten Codeblock ausgeführt.

Code-Einrückung und Blockanweisungen

Ein Codeblock ist ein Abschnitt mit einer bestimmten Einrückungsebene von links.

Die meisten unserer Programme enthalten einen wiederkehrenden Ausschnitt def main():, der einen Block einleitet, wobei der Code innerhalb der Funktion darunter eingerückt ist.

Blöcke definieren die Struktur eines Programms und seine Grenzen. Eine Selektion markiert ebenfalls den Beginn eines neuen Codeblocks.

```
# Look at the indentation below, which marks
the blocks
def main(): #start block 1
    number = 10 #inside block 1
    if (number > 5): #start block 2
        print('This is greater than 5!')
#inside block 2

if __name__ == '__main__':
    main() #outside block 1 and 2 but inside
the if block
```

Neben der Festlegung der Programmstruktur und -funktionalität haben Blockanweisungen auch Auswirkungen auf die Lesbarkeit eines Programms. Code, der sich innerhalb eines Blocks befindet, wird eingerückt. So wird beispielsweise jeglicher Quellcode innerhalb des Blocks einer Selektion tiefer eingerückt als der if-Befehl, der die Selektion eingeleitet hat. Wenn der Block endet, endet auch die Einrückung.

Python hat explizite Richtlinien, wie Code eingerückt werden sollte, die in den PEP 8 Richtlinien nachgelesen werden können. Sie lauten im Wesentlichen:

- Verwenden Sie 4 Leerzeichen pro Einrückungsebene.
- Verwenden Sie Leerzeichen anstelle von Tabulatoren. (PyCharm übersetzt ihren Tabulator automatisch in 4 Leerzeichen)

Das folgende Beispiel ist falsch eingerückt und führt zu einem Fehler.

```
if (number > 10):
number = 9
```

Das folgende Beispiel ist korrekt eingerückt.

```
if (number > 10):
    number = 9
```

Unser Programmcode muss auch in den Übungen immer richtig eingerückt sein. Wenn die Einrückung nicht korrekt ist, wird die Entwicklungsumgebung die Lösung nicht akzeptieren und nicht ausführen.

Else und Else-If

else

Wenn der Ausdruck innerhalb der Klammern der bedingten Anweisung den Wert `false` ergibt, wird die Ausführung des Codes auf die Anweisung nach dem Einrückungsblock verschoben. Dies ist nicht immer erwünscht, und in der Regel wollen wir eine alternative Option für den Fall schaffen, dass der bedingte Ausdruck als `falsch` ausgewertet wird.

Dies kann mit Hilfe des else-Befehls geschehen, der zusammen mit dem `if`-Befehl verwendet wird.

```
number = 4

if (number > 5):
    print('Your number is greater than
five!')
else:
    print('Your number is five or less!')

Your number is five or less!
```

Wenn für eine Selektion ein `else`-Zweig angegeben wurde, wird der durch den `else`-Zweig definierte Block für den Fall ausgeführt, dass die Bedingung der Selektion falsch ist. Der `else`-Befehl steht in

der gleichen Zeile wie der durch den `if`-Befehl definierte Block.

elif

Bei mehreren Bedingungen verwenden wir den `elif`-Befehl. Der Befehl `elif`, kurz für else if, ist wie else, aber mit einer zusätzlichen Bedingung. `elif` folgt der if-Bedingung, und sie kann mehrmals vorkommen.

```
number = 3

if (number == 1):
    print('The number is one')
elif (number == 2):
    print('The given number is two')
elif (number == 3):
    print('The number must be three!')
else:
    print('Something else!')

The number must be three!
```

Lesen wir das obige Beispiel vor: 'Wenn die Zahl eins ist, dann drucke „The number is one“, wenn die Zahl zwei ist, dann drucke „The given number is two“, wenn die Zahl drei ist, dann drucke „The number must be three!“. Andernfalls drucke „Something else!“'

Selektionen mit einer booleschen Variable

Der Wert, der zwischen den Klammern der bedingten Anweisung steht, sollte nach der Auswertung vom Typ Boolean sein. Variablen vom Typ Boolean sind entweder `True` oder `False`.

```
is_it_true = True
if (is_it_true):
    print('Pretty wild!')

Pretty wild!
```

Vergleichsoperatoren können auch außerhalb von Bedingungen verwendet werden. In diesen Fällen wird der boolesche Wert, der sich aus dem Vergleich ergibt, zur späteren Verwendung in einer booleschen Variablen gespeichert.

```
first = 1
second = 3
is_greater = first > second
```

Im obigen Beispiel enthält die boolesche Variable `is_greater` jetzt den booleschen Wert `False`. Wir können das vorherige Beispiel erweitern, indem wir eine Selektion hinzufügen.

```
first = 1
second = 3
is_less_than = first < second

if (is_less_than):
    print('1 is less than 3!')
```

Der obige Code wurde bis zu dem Punkt ausgeführt, an dem die Variablen des Programms erstellt und mit Werten belegt wurden. Die Variable `is_less_than` hat den Wert `True`. Der nächste Schritt in der Ausführung ist der Vergleich `if (is_less_than)` - der Wert für die Variable `is_less_than` wird in ihrem Container gefunden, und das Programm gibt schließlich aus:

```
1 is less than 3!
```

Der Modulo-Operator ist ein etwas weniger gebräuchlicher Operator, der aber sehr praktisch ist, wenn man z. B. die Teilbarkeit einer Zahl überprüfen will. Das Symbol für den Modulo-Operator ist %.

```
remainder = 7 % 2
print(remainder)  # prints 1
print(5 % 3)  # prints 2
print(7 % 4)  # prints 3
print(8 % 4)  # prints 0
print(1 % 2)  # prints 1
```

Wenn wir wissen wollen, ob die vom Benutzer angegebene Zahl durch vierhundert teilbar ist, prüfen wir, ob der Rest Null ist, nachdem wir den Modulo der Zahl und vierhundert gebildet haben.

```
number = int(input())
remainder = number % 400
```

```
if (remainder == 0):
    print('The number ' + str(number) + ' is
divisible by four hundred.')
else:
    print('The number ' + str(number) + ' is
not divisible by four hundred.')
```

Da das Modulo eine Operation wie andere Berechnungen ist, kann es Teil eines Ausdrucks in einer Selektion sein.

```
number = int(input())

if (number % 400 == 0):
    print('The number ' + str(number) + ' is
divisible by four hundred.')
else:
    print('The number ' + str(number) + ' is
not divisible by four hundred.')
```

Vergleichsoperatoren

Sie kennen die Vergleichsoperatoren bereits aus der LU04c - Selektion

Operator	Die Bedingung ist erfüllt, wenn …
==	… die beiden Werte gleich sind.
!=	… die beiden Werte nicht gleich sind.
>	… der erste Wert grösser als der zweite Wert ist.
>=	… der erste Wert grösser oder gleich wie der zweite Wert ist.
<	… der erste Wert kleiner als der zweite Wert ist.
<=	… der erste Wert kleiner oder gleich wie der zweite Wert ist.

Logische Operatoren

Der Ausdruck einer bedingten Aussage kann aus mehreren Teilen bestehen, in denen die logischen Operatoren `and`, `or` und `not` verwendet werden.

- Ein Ausdruck, der aus zwei Ausdrücken besteht, die mit dem Und-Operator kombiniert werden, ist wahr, wenn beide kombinierten Ausdrücke wahr sind.
- Ein Ausdruck, der aus zwei Ausdrücken besteht, die mit dem Oder-Operator kombiniert werden, ist wahr, wenn entweder einer oder beide der kombinierten Ausdrücke als wahr gewertet werden.

- Logische Operatoren werden nicht verwendet, um den booleschen Wert von wahr nach falsch oder von falsch nach wahr zu ändern.

Im nächsten Beispiel kombinieren wir zwei einzelne Bedingungen mit `and`. Der Code wird verwendet, um zu prüfen, ob die Zahl in der Variablen größer oder gleich 5 und kleiner oder gleich 10 ist. Mit anderen Worten, ob sie innerhalb des Bereichs von 5-10 liegt:

```
print('Is the number within the range 5-10:
')
number = 7

if (number >= 4 and number <= 10):
    print('It is! :)')
else:
    print('It is not :(')

Is the number within the range 5-10:
It is! :)
```

Im nächsten Schritt werden zwei Bedingungen mit dem `or`-Operator angegeben: Ist die Zahl kleiner als Null oder größer als 100. Die Bedingung ist erfüllt, wenn die Zahl eine der beiden Bedingungen erfüllt:

```
print('Is the number less than 0 or greater
than 100')
number = 145

if (number < 0 or number > 100):
    print('It is! :)')
else:
    print('It is not :(')

Is the number less than 0 or greater than 100
It is! :)
```

In diesem Beispiel vertauschen wir das Ergebnis des Ausdrucks Zahl > 4 mit `not`, d. h. dem `not`-Operator. Der `not`-Operator ist so geschrieben, dass der zu kippende Ausdruck in Klammern eingeschlossen ist und der `not`-Operator vor den Klammern steht.

```
number = 7

if (not (number > 4)):
    print('The number is not greater than
4.')
```

```
else:
    print('The number is greater than or
equal to 4.')

The number is greater than or equal to 4.
```

Ausführungsreihenfolge von Selektionen

Die Vergleiche werden von oben nach unten ausgeführt. Wenn die Ausführung ein `if` erreicht, deren Bedingung wahr ist, wird ihr Block ausgeführt und der Vergleich beendet.

```
number = 5

if (number == 0):
    print('The number is zero.')
elif (number > 0):
    print('The number is greater than zero.')
elif (number > 2):
    print('The number is greater than two.')
else:
    print('The number is less than zero.')

The number is greater than zero.
```

Im obigen Beispiel wird die Zeichenfolge „The number is greater than zero.“ ausgegeben, auch wenn die Bedingung `Zahl > 2` wahr ist. Der Vergleich stoppt bei der ersten Bedingung, die als wahr ausgewertet wird.

Probleme mit der Ausführungsreihenfolge umgehen

Machen wir uns mit der Ausführungsreihenfolge von Selektionen anhand einer klassischen Programmierübung vertraut.

Schreiben Sie ein Programm, das den Benutzer auffordert, eine Zahl zwischen eins und hundert einzugeben, und das diese Zahl ausgibt. Wenn die Zahl durch drei teilbar ist, gibst du statt der Zahl „Fizz“ aus. Wenn die Zahl durch fünf teilbar ist, dann drucke „Buzz“ anstelle der Zahl. Wenn die Zahl sowohl durch drei als auch durch fünf teilbar ist, dann drucke „FizzBuzz“ anstelle der Zahl.'

Der Programmierer beginnt mit dem Lösen der Aufgabe, indem er die Aufgabenbeschreibung liest und den Code entsprechend der Beschreibung schreibt. Die Bedingungen für die Ausführung werden in der Beschreibung in einer bestimmten Reihenfolge angegeben, und die anfängliche Struktur für das Programm wird auf der Grundlage dieser Reihenfolge gebildet. Die Struktur wird anhand der folgenden Schritte gebildet:

- Schreiben Sie ein Programm, das den Benutzer zur Eingabe einer Zahl auffordert und diese Zahl ausgibt.
- Wenn die Zahl durch drei teilbar ist, wird anstelle der Zahl „Fizz“ gedruckt.
- Wenn die Zahl durch fünf teilbar ist, dann drucke „Buzz“ anstelle der Zahl.
- Wenn die Zahl sowohl durch drei als auch durch fünf teilbar ist, wird anstelle der Zahl „FizzBuzz“ gedruckt.

Wenn-Bedingungen lassen sich leicht mit `if`-, `elif`-, `else`- Anweisungen umsetzen. Der nachstehende Code wurde auf der Grundlage der obigen Schritte geschrieben, funktioniert aber nicht richtig, wie man am Beispiel sehen kann.

```
number = int(input())

if (number % 3 == 0):
    print('Fizz')
elif (number % 5 == 0):
    print('Buzz')
elif (number % 3 == 0 and number % 5 == 0):
    print('FizzBuzz')
else:
    print(number)

User: <3>
Fizz

User: <4>
4

User: <5>
Buzz

User: <15>
Fizz
```

Das Problem mit dem bisherigen Ansatz ist, dass **das Parsen von bedingten Anweisungen bei der ersten Bedingung, die wahr ist, aufhört.** So wird z. B. bei dem Wert 15 die Zeichenfolge „Fizz“ ausgegeben, da die Zahl durch drei teilbar ist (15 % 3 == 0).

Ein Ansatz für die Entwicklung dieses Gedankengangs wäre, zunächst die **anspruchsvollste Bedingung** zu finden und zu implementieren. Danach würden wir die anderen Bedingungen implementieren. Im obigen Beispiel erfordert die Bedingung „wenn die Zahl sowohl durch drei **als auch** durch fünf teilbar ist“, dass zwei Dinge geschehen. Der Gedankengang wäre nun folgender:

1. Schreiben Sie ein Programm, das die Eingaben des Benutzers liest.
2. Wenn die Zahl sowohl durch drei als auch durch fünf teilbar ist, dann gib „FizzBuzz“ anstelle der Zahl aus.
3. Wenn die Zahl durch drei teilbar ist, dann drucke „Fizz“ anstelle der Zahl.
4. Wenn die Zahl durch fünf teilbar ist, wird anstelle der Zahl „Buzz“ gedruckt.
5. Ansonsten gibt das Programm die vom Benutzer angegebene Zahl aus.

Jetzt scheint das Problem gelöst zu sein:

```
number = int(input())

if (number % 3 == 0 and number % 5 == 0):
    print('FizzBuzz')
elif (number % 3 == 0):
    print('Fizz')
elif (number % 5 == 0):
    print('Buzz')
else:
    print(number)

User: <3>
Fizz

User: <4>
4

User: <5>
Buzz

User: <15>
FizzBuzz
```

M319-F3G, M319-F3F, M319-F3E