LU07a - Erweiterung der if-Anweisung

If..In.. Anweisung

Eine `if <keyword> in <list>:`-Anweisung kann verwendet werden um:

1. Zu überprüfen ob das <keyword> bestandteil eines `string` ist

```
if 'aul' in 'grault':                # Truthy
    print('yes')

yes
```

2. Zu überprüfen ob das <keyword> dem Element in einer Collection entspricht

```
if 'baz' in ['foo', 'bar', 'baz']:  # Falsy
    print('yes')

yes
```

Bedingte Ausdrücke (Pythons ternärer Operator)

Python unterstützt eine zusätzliche Entscheidungseinheit, den sogenannten bedingten Ausdruck. (An verschiedenen Stellen in der Python-Dokumentation wird er auch als bedingter Operator oder ternärer Operator bezeichnet.)

In seiner einfachsten Form sieht die Syntax des bedingten Ausdrucks wie folgt aus:

```
<expr1> if <conditional_expr> else <expr2>
```

Sie unterscheidet sich von den bisher gezeigten Formen der if-Anweisung, da sie keine Kontrollstruktur ist, die den Ablauf der Programmausführung steuert. Sie verhält sich eher wie ein Operator, der einen Ausdruck definiert. Im obigen Beispiel wird <conditional_expr> zuerst ausgewertet. Wenn er wahr ist, wird der Ausdruck als <expr1> ausgewertet. Wenn er falsch ist, wird der Ausdruck zu <expr2> ausgewertet.

Beachten Sie die nicht offensichtliche Reihenfolge: Der mittlere Ausdruck wird zuerst ausgewertet, und auf der Grundlage dieses Ergebnisses wird einer der Ausdrücke an den Enden zurückgegeben.

Hier sind einige Beispiele, die hoffentlich zur Verdeutlichung beitragen:

```
raining = False
print("Let's go to the", 'beach' if not
raining else 'library')

Let's go to the beach

raining = True
print("Let's go to the", 'beach' if not
raining else 'library')

Let's go to the library

age = 12
s = 'minor' if age < 21 else 'adult'
print(s)

minor

'yes' if ('qux' in ['foo', 'bar', 'baz'])
else 'no'

no
```

Hinweis: Der bedingte Ausdruck in Python ähnelt der Syntax <conditional_expr> ? <expr1> : <expr2> Syntax, die von vielen anderen Sprachen verwendet wird - C, Perl und Java, um nur einige zu nennen. Tatsächlich wird der ?: Operator in diesen Sprachen gemeinhin als ternärer Operator bezeichnet, was wahrscheinlich der Grund dafür ist, dass Pythons bedingter Ausdruck manchmal als ternärer Python-Operator bezeichnet wird.

Sie können in PEP 308 sehen, dass der <conditional_expr> ? <expr1> : <expr2> für Python in Betracht gezogen, aber letztlich zugunsten der oben gezeigten Syntax verworfen wurde.

Ein häufiger Verwendungszweck des bedingten Ausdrucks ist die Auswahl der Variablenzuweisung. Nehmen wir zum Beispiel an, Sie möchten die größere von zwei Zahlen ermitteln. Natürlich gibt es eine eingebaute Funktion, max(), die genau das tut (und mehr), die Sie verwenden könnten. Aber nehmen wir an, Sie wollen Ihren eigenen Code von Grund auf schreiben.

Sie könnten eine standardmäßige if-Anweisung mit einer else-Klausel verwenden:

```
if a > b:
     m = a
 else:
     m = b
```

Aber ein bedingter Ausdruck ist kürzer und wohl auch besser lesbar:

```
m = a if a > b else b
```

Denken Sie daran, dass sich der bedingte Ausdruck syntaktisch wie ein Ausdruck verhält. Er kann als Teil eines längeren Ausdrucks verwendet werden. Der bedingte Ausdruck hat einen niedrigeren Vorrang als praktisch alle anderen Operatoren, so dass Klammern erforderlich sind, um ihn selbst zu gruppieren.

Im folgenden Beispiel bindet der Operator + stärker als der bedingte Ausdruck, so dass 1 + x und y + 2 zuerst ausgewertet werden, gefolgt von dem bedingten Ausdruck. Die Klammern im zweiten Fall sind unnötig und ändern das Ergebnis nicht:

```
x = 40
y = 40

z = 1 + x if x > y else y + 2
print(z)

42

z = (1 + x) if x > y else (y + 2)
print(z)

42
```

Wenn Sie möchten, dass der bedingte Ausdruck zuerst ausgewertet wird, müssen Sie ihn mit gruppierenden Klammern umgeben. Im nächsten Beispiel wird `(x if x > y else y)` zuerst ausgewertet. Das Ergebnis ist `y`, das `40` ist, also wird `z` mit `1 + 40 + 2 = 43` belegt:

```
x = 40
y = 40

z = 1 + (x if x > y else y) + 2
```

```
print(z)

43
```

Wenn Sie einen bedingten Ausdruck als Teil eines größeren Ausdrucks verwenden, ist es wahrscheinlich eine gute Idee, gruppierende Klammern zur Verdeutlichung zu verwenden, auch wenn sie nicht benötigt werden.

Einzeilige if-Anweisungen

Es ist üblich, if `<expr>` in eine Zeile und `<statement>` eingerückt in die folgende Zeile zu schreiben, etwa so:

```
if <expr>:
    <statement>
```

Es ist jedoch zulässig, eine komplette if-Anweisung in eine Zeile zu schreiben. Die folgende Anweisung ist funktional gleichwertig mit dem obigen Beispiel:

```
if <expr>: <statement>
```

Es kann sogar mehr als ein `<statement>` in der gleichen Zeile stehen, getrennt durch Semikolons:

```
if <expr>: <statement_1>; <statement_2>; ...;
<statement_n>
```

Das Semikolon, das die `<statements>` trennt, hat eine höhere Priorität als der Doppelpunkt nach `<expr>` - im Computerjargon sagt man, das Semikolon bindet fester als der Doppelpunkt. Die `<statements>` werden also als eine Folge von Anweisungen behandelt, und entweder werden alle ausgeführt, oder keine:

```
x = 2
```

```
if x == 1: print('foo'); print('bar');
print('baz')
elif x == 2: print('qux'); print('quux')
else: print('corge'); print('grault')

qux
quux

x = 3
if x == 1: print('foo'); print('bar');
print('baz')
elif x == 2: print('qux'); print('quux')
else: print('corge'); print('grault')

corge
grault
```

Obwohl all dies funktioniert und der Interpreter es zulässt, wird im Allgemeinen davon abgeraten, da es zu einer schlechten Lesbarkeit führt, insbesondere bei komplexen if-Anweisungen. PEP 8 rät ausdrücklich davon ab.

Wie üblich ist es eine Frage des Geschmacks. Die meisten Leute würden das folgende Beispiel optisch ansprechender und auf den ersten Blick leichter zu verstehen finden als das obige Beispiel:

```
x = 3
if x == 1:
     print('foo')
     print('bar')
     print('baz')
elif x == 2:
     print('qux')
     print('quux')
else:
     print('corge')
     print('grault')

corge
grault
```

Wenn eine if-Anweisung jedoch einfach genug ist, kann es sinnvoll sein, alles in eine Zeile zu packen. So etwas würde wahrscheinlich niemanden zu sehr aufregen:

```
debugging = True  # Set to True to turn
debugging on.

    .
    .
    .

if debugging: print('About to call function
foo()')
foo()
```

M319-F1G, M319-F3G

© Kevin Maurizi

From:
https://wiki.bzz.ch/ - **BZZ - Modulwiki**

Permanent link:
https://wiki.bzz.ch/modul/archiv/m319python/learningunits/lu07/lu07a-erweiterteif

Last update: **2024/03/28 14:07**