FS24 CAS PML - Python

4. Kontrollfluss: if-Anweisungen

Vergleich Operationen

- Vergleichoperationen ergeben einen Boolean (*True* oder *False*)
- Vergleichoperatoren sind:
  - kleiner als: *<*
  - kleiner oder gleich: *<=*
  - grösser als: *>*
  - grösser oder gleich: *>=*
  - gleich: *==*
  - ungleich: *!=*
  - gleiches Objekt: *is*
  - negierte Objektgleichheit: *is not*

```
In [1]: print(1 < 2)
True

In [2]: print(1 > 2)
False

In [3]: print(1 == 2)
False

In [4]: print(1 != 2)
True
```

Boolesche Operationen

- Boolean können kombiniert werden mit booleschen Operationen *and* und *or*
  - *bool1 and bool2* ist *True* nur wenn beide *bool1* und *bool2 True* sind. Sonst evaluiert es zu *False*
  - *bool1 or bool2* ist *False* nur wenn beide *bool1* und *bool2 False* sind. Sonst evaluiert es zu *True*

```
True and True   # True
False and False  # False
False and True  # False
True and False  # False
```

```
True or True   # True
False or False  # False
True or False  # True
False or True  # True
```

Boolesche Operationen

- Booleans werden mit *not* invertiert:
  - *not True* ist *False*
  - *not False* ist *True*

```
In [1]: not True and False
Out[1]: False

In [2]: not (True and False)
Out[2]: True
```

- Operatoren Präzedenz:
  - *not* > *and* > *or*

```
In [3]: True or True and False
Out[3]: True

In [4]: (True or True) and False
Out[4]: False
```

Boolesche Operationen

- *0*, *None* und leere Sequenzen evaluieren zu *False:*

```
In [9]: bool(0), bool(None)
Out[9]: (False, False)

In [10]: bool([]), bool("")
Out[10]: (False, False)
```

- Andere Zahlen sowie nicht leere Sequenzen evaluieren zu *True:*

```
In [21]: bool(-1), bool((1,2))
Out[21]: (True, True)

In [22]: bool("0"), bool([0])
Out[22]: (True, True)
```

Boolesche Operationen mit Vergleichen

- Boolesche Operationen können auch mit Vergleichen gebraucht werden:

```
In [40]: print(not 1 > 0)
False

In [41]: print(1 > 0 and False)
False

In [42]: print(1 != 0 or 2 > 1)
True
```

Die *if*-Anweisung

- Die *if*-Anweisung erlaubt einen Code-Block auszuführen nur wenn ein gewisses Ergebnis erfüllt ist.
- Die Syntax ist:

```
if condition1:
    # block executed
    # if condition1 is True
elif condition2:
    # executed if condition1 is False
    # and condition2 is True
else:
    # executed if both condition1
    # and condition2 are False
```

Python Syntax

- Code blocks werden durch Einrückung definiert!
- Normalerweise braucht man 4 Leerzeichen

```
if condition1:
    # block executed
    # if condition1 is True
    if condition2:
        # executed if condition1
        # and condition2 are True
    else:
        # executed if condition1 is True
        # and condition2 False
else:
    # executed if condition1 is False
```

Die *if*-Anweisung

- Beispiel:

```
In [49]: a = 6
    ...: if a > 4:
    ...:     print("a is larger than 4")
    ...: elif a > 2:
    ...:     print("a is larger than 2")
    ...: else:
    ...:     print("a is smaller or equal to 2")
    ...:
a is larger than 4
```

- Führt den ersten *if* oder *elif* block aus für welchen das Ergebnis *True* ist

Die *if*-Anweisung

- Beispiel:

```
In [1]: a = 3
   ...: if a > 4:
   ...:     print("a is larger than 4")
   ...: elif a > 2:
   ...:     print("a is larger than 2")
   ...: else:
   ...:     print("a is smaller or equal to 2")
   ...:
a is larger than 2
```

- Führt den ersten *if* oder *elif* block aus für welchen das Ergebnis *True* ist

Die *if*-Anweisung

- Beispiel:

```
In [48]: a = 1
    ...: if a > 4:
    ...:     print("a is larger than 4")
    ...: elif a > 2:
    ...:     print("a is larger than 2")
    ...: else:
    ...:     print("a is smaller or equal to 2")
    ...:
a is smaller or equal to 2
```

- Führt den ersten *if* oder *elif* block aus für welchen das Ergebnis *True* ist

Die *if*-Anweisung

- Die *else* Anweisung ist fakultativ
- Man kann mehrere oder keine *elif* Anweisungen haben

```
a = 2
if a == 2:
    print("a is 2")
print("let's go on!")
```

```
a = 1
if a == 2:
    print("a is 2")
else:
    print("a is not 2")
```

```
a = 1
if type(a) == str:
    print("a is a string")
elif a > 0:
    print("a is positive")
elif a < 0:
    print("a is negative")
else:
    print("a=0")
```

Reminder

- Comparison operators: >, <, >=, <=, ==, !=
- boolean operators: *and* and *or*
- Operator precedence: *not* > *and* > *or*
- if statement:

```
a = 1
if type(a) == str:
    print("a is a string")
elif a > 0:
    print("a is positive")
elif a < 0:
    print("a is negative")
else:
    print("a=0")
```

Zusätzliche Folien

Boolesche Operationen für andere Typen

- *Boolesche Operationen funktionieren auch mit anderen Typen*

```
In [35]: 0 or 1
Out[35]: 1
```

- *Boolesche Operatoren werden von links nach rechts evaluiert (natürlich mit Präzedenz)*
- Die Auswertung wird abgebrochen sobald das Ergebnis eindeutig ist

```
In [36]: 0 or 1 or 2
Out[36]: 1
```

Boolesche Operationen für andere Typen

- *or* gibt den ersten Wert der *True* evaluiert zurück
- Wenn alle *False* sind, dann den letzten Wert

```
In [54]: 0 or False or []
Out[54]: []
```

```
In [55]: 0 or 1 or True
Out[55]: 1
```

- *and* gibt den ersten Wert der *False* ist, zurück
- Wenn alle *True* sind, dann wird der letzte Wert zurückgegeben

```
True and 0 and False
0
```

```
True and 1 and [0]
[0]
```

all und *any*

- die *all* und *any* Funktionen nehmen eine Sequenz als Parameter
- *all(seq)* gibt *True* zurück wenn alle Elemente von *seq* *True* sind, sonst *False*

```
In [74]: all([1, 2, 3])
Out[74]: True
```

```
In [75]: all([1, False, 3])
Out[75]: False
```

- *any(seq)* gibt *False* zurück wenn alle Elemente von *seq False* sind, sonst *True*

```
In [76]: any([0, False])
Out[76]: False
```

```
In [77]: any([0, False, True])
Out[77]: True
```