If Else

Ein if/else-Statement führt Code in Abhängigkeit von einer Bedingung aus. Diese Bedingung ist auf der Basisebene immer ein boolean-Wert. In Python werden die if/else-Statements mit einem Doppelpunkt beendet.

In [27]:

```
# boolean
# True / False
# 1 / 0

bool = False

if bool:
    print("if")
else:
    print("else")
```

```
else
```

In der oberen Zelle haben wir der Variablen `bool` einen Wert zugewiesen. Abhängig von diesem Wert wird entweder *if* oder *else* ausgegeben. Der if-Block wird ausgeführt, wenn `bool` den Wert `True` enthält, sonst wird der else-Block ausgeführt.

Als Block werden in diesem Falle alle Zeilen Code bezeichnet, die vom if- oder vom else-Statement abhängen. In Python werden diese Blöcke durch Einrückung gekennzeichnet. Bei dem Beispiel unten werden die Texte *im if Block* und *zweite Zeile* nur ausgegeben, wenn `bool` den Wert `True` hat, der Text *nicht im if Block* hingegen wird immer ausgegeben. Weil die Code-Zeile 4 nicht mehr eingerückt ist, befindet sie sich nicht im if-Block.

In [5]:

```
if bool:
    print("im if Block")
    print("zweite Zeile")
print("nicht im if Block")
```

```
nicht im if Block
```

Dasselbe gilt auch für else-Blöcke. Auch hier ist die Einrückung wichtig!

Als Bedingung für das if-Statement kann auch ein Vergleich verwendet werden. Ein Vergleich wird mit einem doppelten Gleichheitszeichen `==` geschrieben. In dem Beispiel wird *if* ausgegeben, wenn `a` gleich `5` ist. Sonst wird *else* ausgegeben.

In [28]:

```
1 a = int(input())
2
3 if a == 5:
4     print("if")
5 else:
6     print("else")
```

```
5
if
```

Ein Vergleich kann auch verneint werden. Dies wird mit einem `!=` geschrieben. In dem Beispiel wird if ausgegeben, wenn `a` ungleich `5` ist.

In [29]:

```
1 a = int(input())
2
3 if a != 5:
4     print("if")
5 else:
6     print("else")
```

```
4
if
```

Diese Vergleiche können mit beliebigen Datentypen durchgeführt werden. Intern interpretiert Python das aber immer als boolean-Wert. Das heißt, der Vergleich `a == 5` wird als `True` oder `False` interpretiert, je nachdem welchen Wert `a` hat. In dem Beispiel wird der Variable `bool` das Ergebnis des Vergleichs `a == 5` zugewiesen und mit der `print()` -Funktion ausgegeben. Wie wir sehen, ist bool `False` , also tatsächlich ein boolean-Wert.

In [32]:

```
1 a = 4
2 bool = (a == 5)  # Klammern dienen nur der Übersicht, sind aber nicht notwendig
3 print(bool)
```

```
False
```

Wenn mehrere Bedingungen nacheinander geprüft werden, kann ein `elif` -Statement verwendet werden. `elif` setzt sich zusammen aus `else if` , zu deutsch *sonst wenn*. Python prüft elif-Blöcke nur, wenn vorher noch keine Bedingung `True` ergeben hat. Das heißt im Umkehrschluss, wenn eine Bedingung erfüllt wird, sie also `True` ist, werden alle folgenden elif-Blöcke (und else-Blöcke) übersprungen, ohne dass deren Bedingung geprüft wird. Dies ist im Beispiel ersichtlich: wenn für `a` der Wert `5` eingegeben wird, stimmen sowohl die Bedingung in Zeile 3 als auch die Bedingung in Zeile 5. Trotzdem wird nur *Block 1* ausgegeben, weil die Bedingung in Zeile 5 nicht geprüft wird.
Wenn statt der `elif` -Statements mehrere `if` -Statements verwendet werden, wird jede Bedingung geprüft. Deshalb werden sowohl *Block 4* als auch *Block 5* ausgegeben. Dementsprechend muss Python aber auch etwas mehr arbeiten, was den Code unter Umständen verlangsamen kann.

In [36]:

```
a = int(input())

if a == 5:
    print("Block 1")
elif a == 5:
    print("Block 2")
elif a == 3:
    print("Block 3")
else:
    print("else")

if a == 5:
    print("Block 4")
if a == 5:
    print("Block 5")
if a == 3:
    print("Block 6")
```

```
5
Block 1
Block 4
Block 5
```

Jedes `if` -Statement beginnt also eine neue Folge von `if` -> `elif` -> `else` -Statements, wobei die Anzahl der `elif` -Statements unbegrenzt ist.

Eine weitere Möglichkeit für Bedingungen bietet der `in` -Operator. Dieser kann unter anderem auf Strings angewendet werden. Dabei wird geprüft, ob ein String ein Teil eines anderen Strings ist. Die Verneinung wird mit dem `not` -Operator durchgeführt.

In [43]:

```
s = "dies ist ein Text"

if "Te" in s:
    print("if")
else:
    print("else")
```

```
if
```

In [44]:

```
t = "ein anderer Text"

if "Te" not in t:
    print("if")
else:
    print("else")
```

```
else
```

not zur Negierung von booleans

Der `not` -Operator kann nicht nur in Verbindung mit `in` verwendet werden, sonder auch um booleans generell zu verneinen. Wenn etwas `not True` ist, dann ist es `False` und umgekehrt genauso.

In [46]:

```
bool = not True
print(bool)
```

```
False
```

Da wir wissen, dass jede Bedingung bei einem `if` -Statement als boolean-Wert interpretiert wird, kann auch hier `not` zur Verneinung verwendet werden.

In [48]:

```
a = 5

if not a == 5:  # wenn nicht a==5, also wenn a!=5
    print("if")
else:
    print("else")
```

```
else
```

In [50]:

```
s = "string"

if not 'x' in s:  # wenn nicht 'x' in s, also wenn 'x' nicht in s
    print("if")
else:
    print("else")
```

```
if
```

Diese Schreibweisen sind also auch möglich, aber unüblich.