

# Control Flow mit Python

---

Wir wollen mehr Kontrolle haben!

1.

Einführung

Was bedeutet  
Control Flow  
überhaupt

# Passwort-Überprüfung

---

Es folgt ein kurzes Beispiel, welches wiederholte Eingaben verwendet und Bedingungen überprüft

# Boolescher Datentyp

- Datentyp mit nur zwei möglichen Werten: **Wahr** und **Falsch**

```
light_is_on = True  
door_is_open = False
```



# Vergleichsoperatoren

Ausdruck	Beschreibung
$x == y$	Wird als Wahr ausgewertet, wenn x gleich y ist; sonst Falsch
$x != y$	Wird als Wahr ausgewertet, wenn x nicht dasselbe wie y ist; sonst Falsch
$x < y$	Wird als Wahr ausgewertet, wenn x kleiner als y ist; sonst Falsch
$x > y$	...
$x >= y$	...
$x <= y$	...

# Logische Operatoren

Logical And: and	
Expression	Value
True and True	True
True and False	False
False and True	False
False and False	False

# Logische Operatoren

Logical Or: or	
Expression	Value
True or True	True
True or False	True
False or True	True
False or False	False

# Logische Operatoren

Logical Negation: not	
Expression	Value
not True	False
not False	True



# Aktualisierte Operatorenreihenfolge

Reihenfolge	Operator	Operation	Beispiel
5	*	Multiplikation	3 * 4
6	-	Subtraktion	10 - 8
7	+	Addition	100 + 200
8	<, <=, >, >=, !=, ==	Vergleich	3 > 2
9	not	Negation	x is not y
10	and	Logical And	x == 2 and y == 3
11	or	Logical Or	x == 2 or y == 3

# Quiz: Bedingungen Teil 1 - Aufwärmen

Zu welchem Wert werden die Ausdrücke A bis D jeweils ausgewertet?

**A**     `True and False`

**B**     `True or False`

**C**     `not True`

**D**     `10 == 12`

# Quiz: Bedingungen Teil 2 - Variablen

Welchen Wert hat die Variable k in den Skripten A und B jeweils?

**A**

```
x, y = 10, 3  
k = x > y and y%2 == 0
```

**B**

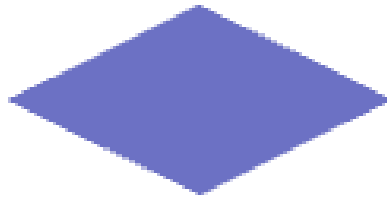
```
r, s = 100, "test"  
k = s == "Test" or r >= 100
```

2.

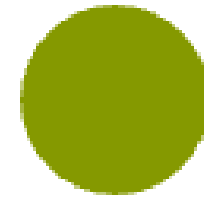
# Flussdiagramme

Etwas  
grafisches zu  
Beginn

# Programm-Flussdiagramm



Branching



Start/Stop



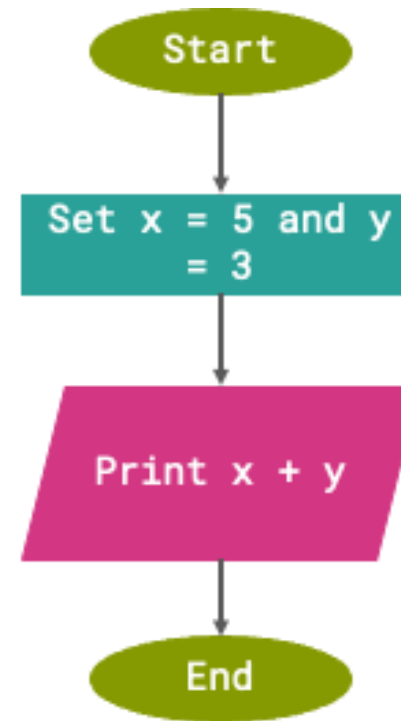
Operation /  
Statement



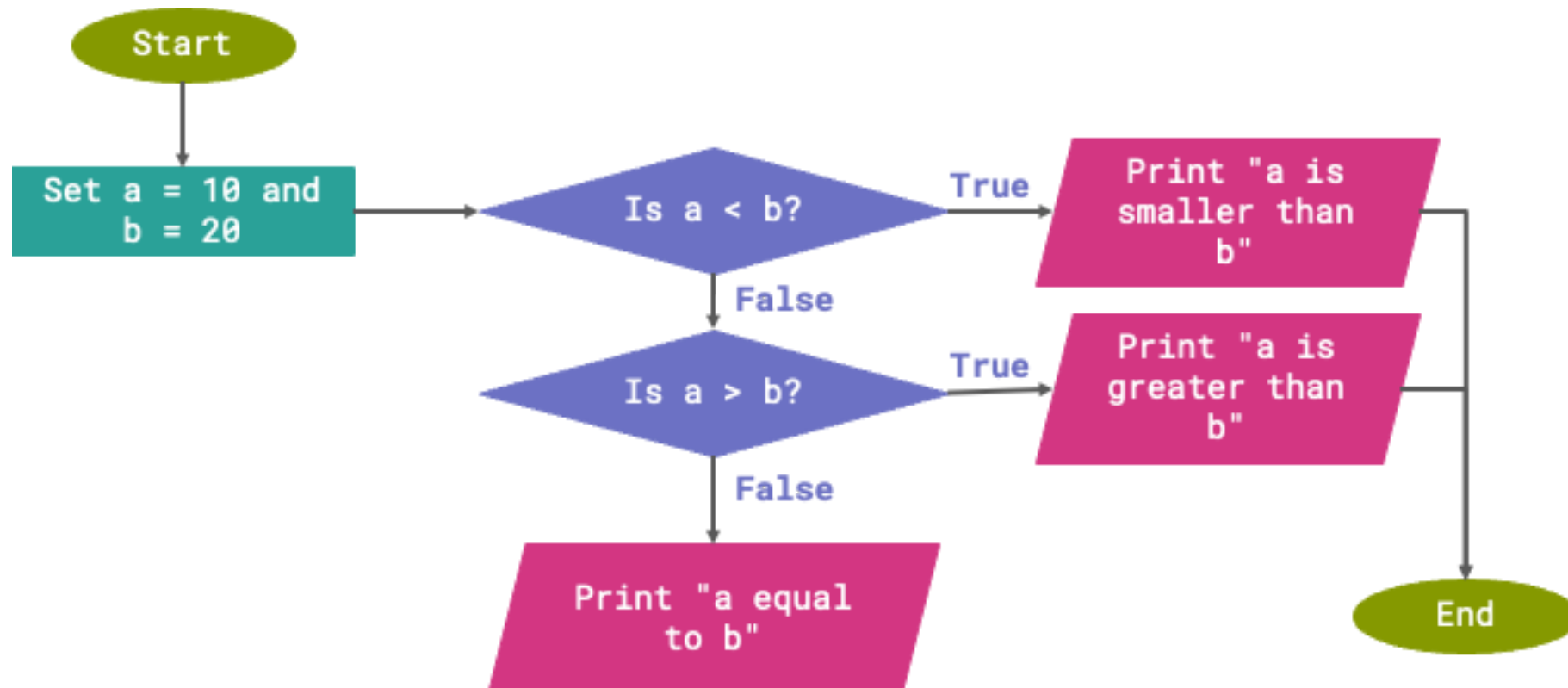
Input/Output

# Programm-Flussdiagramm

```
x, y = 5, 3  
print(x + y)
```



# Bedingte Anweisungen



# 3.

## if-elif-else

Bedingte  
Anweisungen



# Bedingte Anweisungen



```
1 a, b = 10, 20
2
3 if a < b:
4     print("a is smaller than b")
5 elif a > b:
6     print("a is greater than b")
7 else:
8     print("a is equal to b")
```

# Bedingte Anweisungen

---

- if-Anweisung (mit Block) kann allein stehen
- elif- und else-Anweisungen müssen immer mit einer if-Anweisung verwendet werden
- 0 oder eine beliebige Anzahl von elif-Anweisungen können verwendet werden
- die else-Anweisung ist optional

# Bedingte Anweisungen

- Anweisungsblock: Ein zusammenhängender Abschnitt, der aus einer oder mehreren Anweisungen besteht
- Python verwendet Einrückung (Space- oder Tab-Taste) zur Definition eines Blocks

```
if a < b:  
    print("a is smaller than b")  
    print("Still belongs to Block 2")  
else:  
    print("This is a new block")
```

# Quiz: Bedingte Anweisungen

- Was wird auf der Konsole ausgegeben, wenn der Code ausgeführt wird?

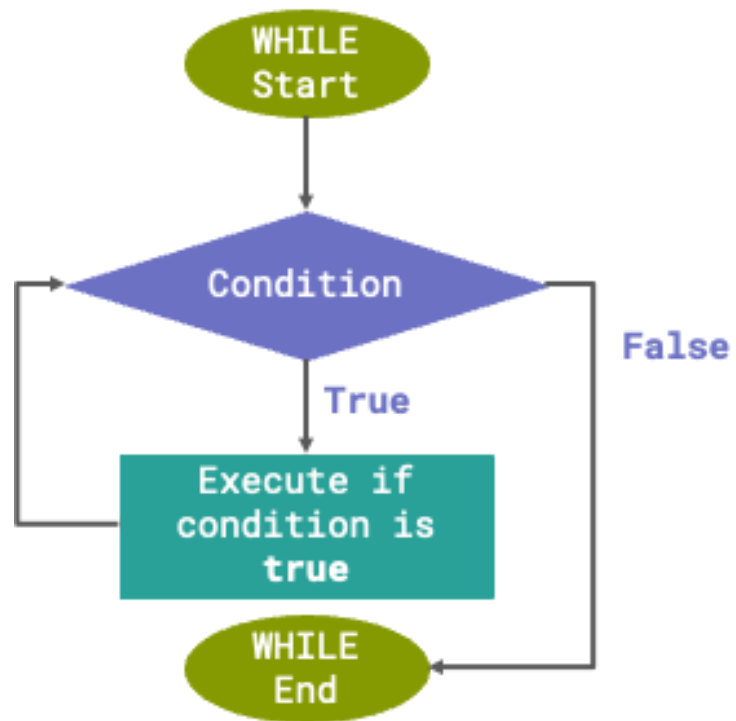
```
1 x = 100
2 if x > 0:
3     if x < 100:
4         print('A')
5     elif x > 30 and x % 2 == 0:
6         print('B')
7     else:
8         print('C')
9 print('D')
```

4.

while Schleife

Eine Erste  
Iteration

# while Schleife

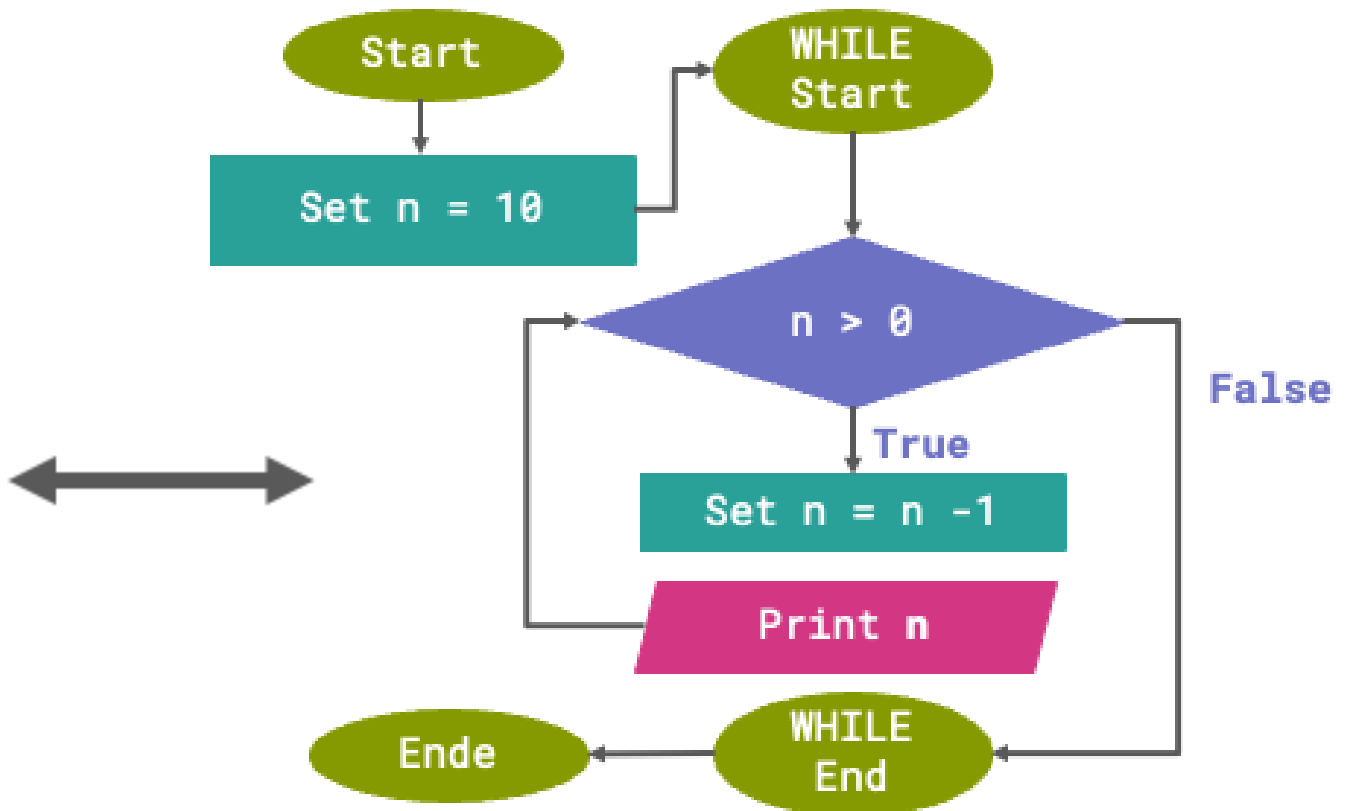


```
while <Condition>:  
    <Execute if condition is true>
```

# while Schleife



```
1 n = 10
2
3 while n > 0:
4     n = n - 1
5     print(n)
```



# break Ausdruck

```
1 n = 10
2 while True:
3     if n == 37:
4         print("Found the number 37")
5         break
6     n = n + 1
```

Der break Ausdruck kann nur innerhalb einer while- und/oder for-Schleife verwendet werden



# Endlose Schleifen



```
1 n = 0
2 while n < 10:
3     print("Hallo!")
```

Die Bedingung in der while-Schleife muss irgendwann zu False ausgewertet werden, oder die while-Schleife muss irgendwann durch break beendet werden, sonst gerät man in eine **Endlosschleife**

# Quiz: while Schleife

Welche Zahlen bzw. wie viele werden ausgegeben, wenn der folgende Code ausgeführt wird?

```
1 n = 10
2
3 while True:
4     n = n - 1
5
6     if n == 1:
7         break
8
9     if n % 2 != 0 and n != 7:
10        print(n)
```

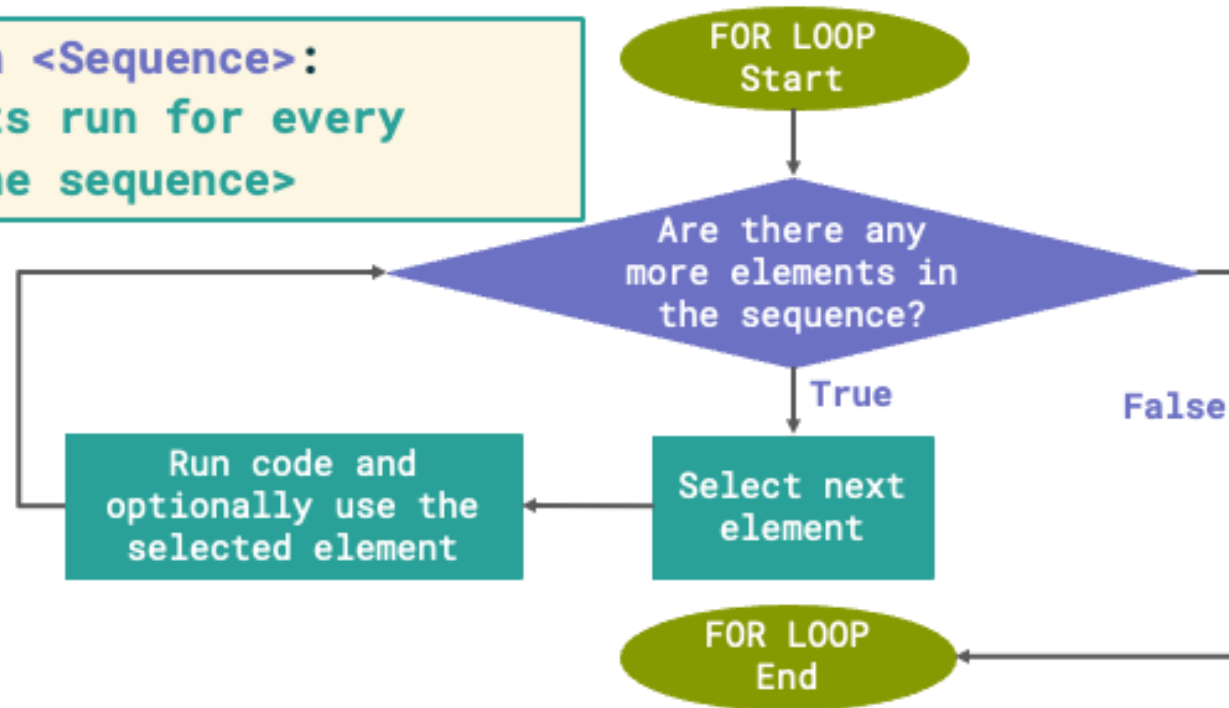
5.

for Schleife

Eine zweite  
Iteration

# for Schleife

```
for <Element> in <Sequence>:  
    <Code that gets run for every  
    element in the sequence>
```



# for Schleife

---



```
1 for n in [1, 2, 3]:  
2     print(n)
```

Die Python for-Schleife wird immer auf Sequenzen oder andere iterierbare Objekte angewendet

Sequenzen: list, tuple und range

# continue Ausdruck

```
1 for n in [1, 2, 3, 4, 5]:  
2     if n == 2:  
3         continue  
4  
5     print(n)
```

Der continue Ausdruck kann nur innerhalb einer while- und/oder for-Schleife verwendet werden

# Quiz: continue Ausdruck



```
1 for n in range(100):  
2     if n%2 == 0:  
3         continue  
4  
5     print(n)
```

Was bewirkt das obere Skript?

# Übung: Einen Counter implementieren

Füllen Sie im folgenden Code die Lücken ( \_\_ ) so aus, dass die Zahlen von 1 bis einschliesslich 20 in aufsteigender Reihenfolge ausgegeben werden. Längere Lücken können aus mehreren Ausdrücken bestehen.

```
k = ____  
  
while k ____:  
    print(k)  
    k = ____
```

Testen Sie Ihren Code anschliessend in PyCharm.



# Übung: Bedingungen definieren

Definieren Sie die folgenden Bedingungen als Python-Ausdrücke:

1. Die Zeichenfolge **s** muss mit "**Knb423Lo12**" übereinstimmen
2. **x** muss entweder grösser als 100 oder kleiner als 20 und durch 3 teilbar sein
3. **y** muss zwischen 10 und 20 liegen und darf nicht durch 7 teilbar sein

# Übung: Grösste Nummer

Schreiben Sie ein Skript **largest\_number.py**, das den Benutzer auffordert, drei ganze Zahlen einzugeben.

Nach der Eingabe der Zahlen sollte das Skript ausgeben, welche Zahl die größte ist.

Wenn der Benutzer zum Beispiel 100, 14 und 81 eingibt, sollte das Skript **100** auf der Konsole ausgeben.

# Übung: Advanced Calculator

Schreiben Sie ein Skript **advanced\_calculator.py**, das zwei Ganzzahlen und einen Operator (+, -, / oder \*) einliest.

Geben Sie das Ergebnis je nach Operator auf dem Bildschirm aus. Wenn der eingegebene Operator nicht unterstützt wird, geben Sie stattdessen eine **Fehlermeldung** aus.

# Übung

Schreiben Sie ein Skript `simple_calculator.py`, das den Benutzer auffordert, zwei Zahlen einzugeben. Das Skript sollte die folgende Zeichenfolge ausgeben, wobei `<SUM>` die Summe der beiden eingegebenen Zahlen ist:

```
The result is <SUM>
```

# Ende

Das war alles für dieses Kapitel

---