Programmieren und Problemlösen

Funktionen und lokale Variablen

Dennis Komm

Repetition – Kontrollanweisungen

Kontrollfluss

Kontrollfluss – `if`

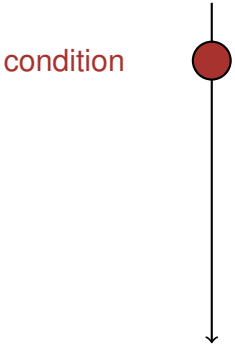
Reihenfolge der (wiederholten) Ausführung von Anweisungen

- Grundsätzlich von oben nach unten...

Kontrollfluss – `if`

Reihenfolge der (wiederholten) Ausführung von Anweisungen

- Grundsätzlich von oben nach unten...
- ...ausser in Auswahl- und Kontrollanweisungen

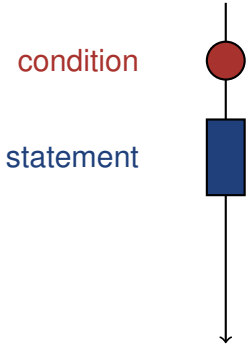



```
if condition:
    statement
```

Kontrollfluss – `if`

Reihenfolge der (wiederholten) Ausführung von Anweisungen

- Grundsätzlich von oben nach unten...
- ...ausser in Auswahl- und Kontrollanweisungen

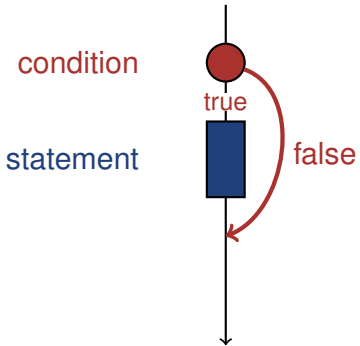



```
if condition:
    statement
```

Kontrollfluss – `if`

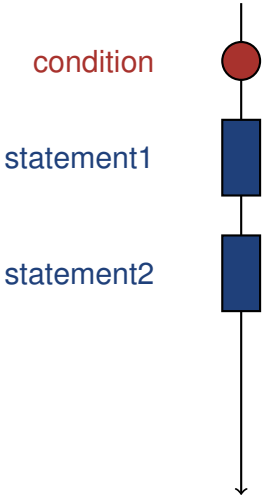
Reihenfolge der (wiederholten) Ausführung von Anweisungen

- Grundsätzlich von oben nach unten. . .
- . . . ausser in Auswahl- und Kontrollanweisungen



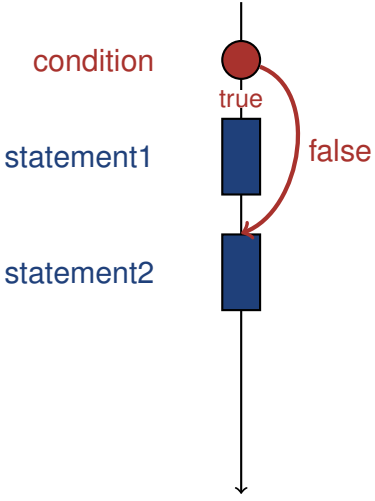

```
if condition:
    statement
```

Kontrollfluss – `if-else`



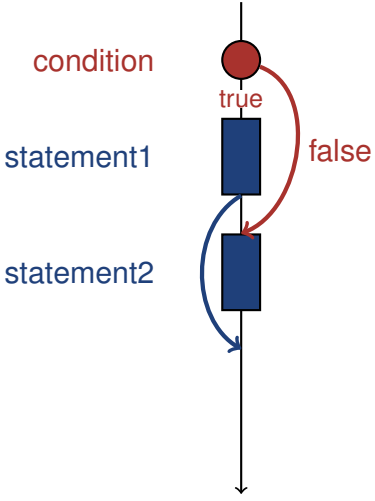

```
if condition:
    statement1
else:
    statement2
```

Kontrollfluss – `if-else`




```
if condition:
    statement1
else:
    statement2
```

Kontrollfluss – `if-else`

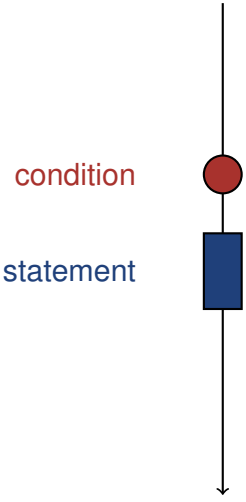



```
if condition:
    statement1
else:
    statement2
```

Kontrollfluss – `while`

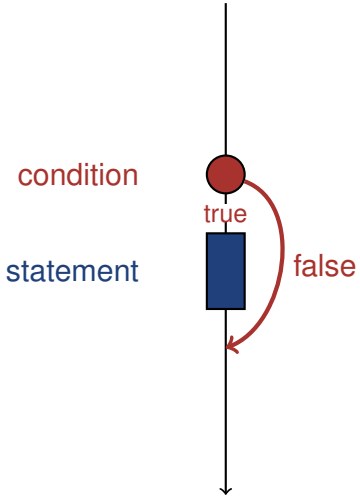
`while` *condition*:
statement

Kontrollfluss – `while`



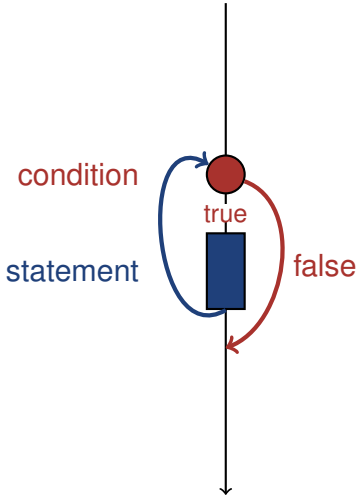
`while` *condition*:
statement

Kontrollfluss – `while`



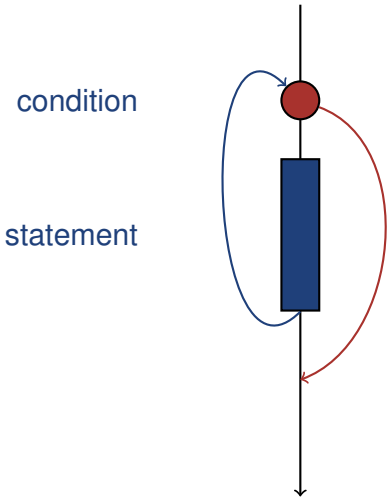
`while` *condition*:
statement

Kontrollfluss – `while`

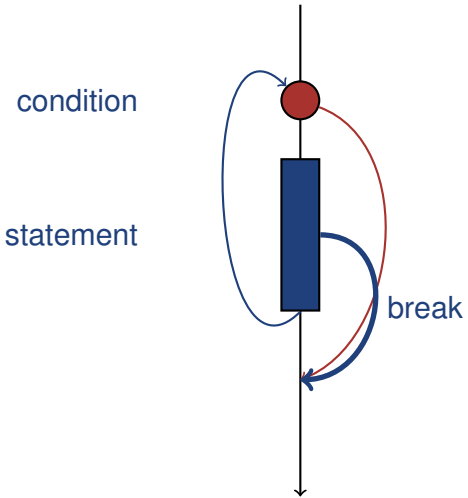


`while` *condition*:
statement

Kontrollfluss `break` in `while`-Schleife



Kontrollfluss `break` in `while`-Schleife



Funktionen

Funktionen

Bisher...

- Ein Algorithmus pro Datei
- Anweisungen werden sequentiell abgearbeitet
- Verwendung von Schleifen und Kontrollstrukturen

Funktionen

Bisher...

- Ein Algorithmus pro Datei
- Anweisungen werden sequentiell abgearbeitet
- Verwendung von Schleifen und Kontrollstrukturen

Gruppiere zusammenhängenden Code als **Funktion**

Funktionen

Bisher...

- Ein Algorithmus pro Datei
- Anweisungen werden sequentiell abgearbeitet
- Verwendung von Schleifen und Kontrollstrukturen

Gruppiere zusammenhängenden Code als **Funktion**

```
def begruessung():
    date = "18. Maerz 2021"
    print("Hallo", username, "!")
    print("Willkommen zur Vorlesung am", date)

begruessung()
```

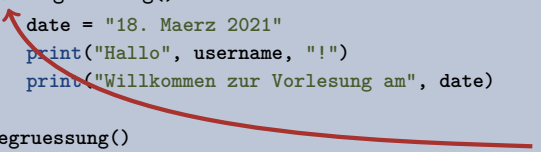
Funktionen

Bisher...

- Ein Algorithmus pro Datei
- Anweisungen werden sequentiell abgearbeitet
- Verwendung von Schleifen und Kontrollstrukturen

Gruppiere zusammenhängenden Code als **Funktion**

```
def begruessung():
    date = "18. Maerz 2021"
    print("Hallo", username, "!")
    print("Willkommen zur Vorlesung am", date)

begruessung()
```



Funktionen

Bisher...

- Ein Algorithmus pro Datei
- Anweisungen werden sequentiell abgearbeitet
- Verwendung von Schleifen und Kontrollstrukturen

Gruppiere zusammenhängenden Code als **Funktion**

```
def begruessung():
    date = "18. Maerz 2021"
    print("Hallo", username, "!")
    print("Willkommen zur Vorlesung am", date)

begruessung()
```

Optionale Parameterliste

Analogie zu natürlichen Sprachen

- Python „versteht“ gewisse Wörter
- Diese heissen **Schlüsselwörter**: `def`, `if`, `while` etc.
- Grundstock an **Funktionen**: `print()`, `range()`, `input()` etc.

Analogie zu natürlichen Sprachen

- Python „versteht“ gewisse Wörter
- Diese heissen **Schlüsselwörter**: `def`, `if`, `while` etc.
- Grundstock an **Funktionen**: `print()`, `range()`, `input()` etc.

`def f():` $\Longleftrightarrow$ Python „lernt“ neues Wort `f`

Analogie zu natürlichen Sprachen

- Python „versteht“ gewisse Wörter
- Diese heissen **Schlüsselwörter**: `def`, `if`, `while` etc.
- Grundstock an **Funktionen**: `print()`, `range()`, `input()` etc.

`def f():` $\Longleftrightarrow$ Python „lernt“ neues Wort `f`

Aus dem Duden

Kühl·schrank, der
Mit einer Kältemaschine ausgestatteter schrankartiger Behälter zum Kühlen oder Frischhalten von Lebensmitteln

Analogie zu natürlichen Sprachen

```
  def begruessung():
    date = "18. Maerz 2021"
    print("Hallo", username, "!")
    print("Willkommen zur Vorlesung am", date)
```

```
username = input("Geben Sie Ihren Benutzernamen ein:")
if username == "leafr" or username == "skamp" or username == "dkomm":
    begruessung()
    ...
else:
    print("Benutzer nicht bekannt.")
...
```

Analogie zu natürlichen Sprachen

```
  def begruessung():
    date = "18. Maerz 2021"
    print("Hallo", username, "!")
    print("Willkommen zur Vorlesung am", date)
```

```
username = input("Geben Sie Ihren Benutzernamen ein:")
if username == "leafr" or username == "skamp" or username == "dkomm":
    begruessung()
    ...
else:
    print("Benutzer nicht bekannt.")
...
```

Analogie zu natürlichen Sprachen

```
  def begruessung():
    date = "18. Maerz 2021"
    print("Hallo", username, "!")
    print("Willkommen zur Vorlesung am", date)
```

```
username = input("Geben Sie Ihren Benutzernamen ein:")
if username == "leafr" or username == "skamp" or username == "dkomm":
    begruessung()



    ...
else:
    print("Benutzer nicht bekannt.")
...
```

Analogie zu natürlichen Sprachen

```
  def begruessung():
    date = "18. Maerz 2021"
    print("Hallo", username, "!")
    print("Willkommen zur Vorlesung am", date)
```

```
username = input("Geben Sie Ihren Benutzernamen ein:")
if username == "leafr" or username == "skamp" or username == "dkomm":
    date = "18. Maerz 2021"
    print("Hallo", username, "!")
    print("Willkommen zur Vorlesung am", date)
    ...
else:
    print("Benutzer nicht bekannt.")
...
```

Analogie zu mathematischen Funktionen

$$f(x) = 2 \cdot x + 1$$

Analogie zu mathematischen Funktionen

$$f(x) = 2 \cdot x + 1$$

Funktionen in Python

- **Parameter** `x` wird Funktion übergeben
- **Wert** wird zurückgegeben mit `return`

Analogie zu mathematischen Funktionen

$$f(x) = 2 \cdot x + 1$$

Funktionen in Python

- **Parameter** `x` wird Funktion übergeben
- **Wert** wird zurückgegeben mit `return`

```
def f(x):
    y = 2 * x + 1
    return y
```

Analogie zu mathematischen Funktionen

$$f(x) = 2 \cdot x + 1$$

Funktionen in Python

- **Parameter** `x` wird Funktion übergeben
- **Wert** wird zurückgegeben mit `return`

```
def f(x):
    y = 2 * x + 1
    return y
```

```
def f(x):
    return 2 * x + 1
```

Analogie zu mathematischen Funktionen

$$f(x) = 2 \cdot x + 1$$

Funktionen in Python

- **Parameter** `x` wird Funktion übergeben
- **Wert** wird zurückgegeben mit `return`

```
def f(x):
    y = 2 * x + 1
    return y
```

```
def f(x):
    return 2 * x + 1
```

- `return` ohne Argument zum Beenden des Funktionsaufrufs

Analogie zu mathematischen Funktionen

```
def f(x):
    return 2 * x + 1
```

Analogie zu mathematischen Funktionen

```
def f(x):
    return 2 * x + 1
```

Durch die Verwendung von `return` repräsentiert der Funktionsaufruf den entsprechenden Wert

Analogie zu mathematischen Funktionen

```
def f(x):
    return 2 * x + 1
```

Durch die Verwendung von `return` repräsentiert der Funktionsaufruf den entsprechenden Wert

- `print(f(5))` erzeugt Ausgabe `11`

Analogie zu mathematischen Funktionen

```
def f(x):
    return 2 * x + 1
```

Durch die Verwendung von `return` repräsentiert der Funktionsaufruf den entsprechenden Wert

- `print(f(5))` erzeugt Ausgabe `11`
- `z = f(6)` weist `z` den Wert `13` zu
- `z = 3 * f(2) + f(4)` weist `z` den Wert `24` zu

Analogie zu mathematischen Funktionen

```
def f(x):
    return 2 * x + 1
```

Durch die Verwendung von `return` repräsentiert der Funktionsaufruf den entsprechenden Wert

- `print(f(5))` erzeugt Ausgabe `11`
- `z = f(6)` weist `z` den Wert `13` zu
- `z = 3 * f(2) + f(4)` weist `z` den Wert `24` zu
- `b = (f(10) > 20)` weist `b` den Booleschen Wert `True` zu

Funktionen mit Parametern

```
def checkuser(givenname):
    validnames = [ "heinj", "sarstein", "spiasko", "celich", "sommerda", "fiscmanu" ]
    if givenname in validnames:
        return True
    else:
        return False
```

```
username = input("Geben Sie Ihren Benutzernamen ein:")

if checkuser(username) == True:
    print("Willkommen", username)
    password = input("Geben Sie Ihr Passwort ein:")
    ...
else:
    print("Benutzername nicht gefunden.")
```

Funktionen mit Parametern

```
def checkuser(givenname):
    validnames = [ "heinj", "sarstein", "spiasko", "celich", "sommerda", "fiscmanu" ]
    if givenname in validnames:
        return True
    else:
        return False
```

```
username = input("Geben Sie Ihren Benutzernamen ein:")

if checkuser(username) == True:
    print("Willkommen", username)
    password = input("Geben Sie Ihr Passwort ein:")
    ...
else:
    print("Benutzername nicht gefunden.")
```

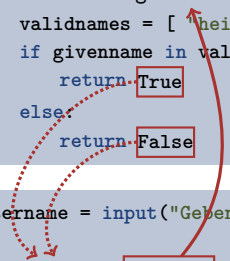
Funktionen mit Parametern

```
def checkuser(givenname):
    validnames = [ "heinj", "sarstein", "spiasko", "celich", "sommerda", "fiscmanu" ]
    if givenname in validnames:
        return True
    else:
        return False
```

```
username = input("Geben Sie Ihren Benutzernamen ein:")

if checkuser(username) == True:
    print("Willkommen", username)
    password = input("Geben Sie Ihr Passwort ein:")
    ...
else:
    print("Benutzername nicht gefunden.")
```

Funktionen mit Parametern

```
def checkuser(givenname):
    validnames = [ "heinj", "sarstein", "spiasko", "celich", "sommerda", "fiscmanu" ]
    if givenname in validnames:
        return True
    else:
        return False

username = input("Geben Sie Ihren Benutzernamen ein:")

if checkuser(username) == True:
    print("Willkommen", username)
    password = input("Geben Sie Ihr Passwort ein:")
    ...
else:
    print("Benutzername nicht gefunden.")
```



Funktionen mit Parametern

```
username = input("Geben Sie Ihren Benutzernamen ein:")
```

Funktionen mit Parametern

```
username = input("Geben Sie Ihren Benutzernamen ein:")
```

```
Geben Sie Ihren Benutzernamen ein:
```

Funktionen mit Parametern

```
username = input("Geben Sie Ihren Benutzernamen ein:")
```

```
Geben Sie Ihren Benutzernamen ein: dkomm
```

Funktionen mit Parametern

```
username = input("Geben Sie Ihren Benutzernamen ein:")
```

```
Geben Sie Ihren Benutzernamen ein: dkomm
```

```
username = dkomm
```

Funktionen mit Parametern

```
username = input("Geben Sie Ihren Benutzernamen ein:")
```

```
Geben Sie Ihren Benutzernamen ein: dkomm
```

```
username = dkomm
```

```
if checkuser(username) == True:
```

Funktionen mit Parametern

```
username = input("Geben Sie Ihren Benutzernamen ein:")
```

```
Geben Sie Ihren Benutzernamen ein: dkomm
```

```
username = dkomm
```

```
if checkuser(dkomm) == True:
```

Funktionen mit Parametern

```
username = input("Geben Sie Ihren Benutzernamen ein:")
```

```
Geben Sie Ihren Benutzernamen ein: dkomm
```

```
username = dkomm
```

```
if checkuser(dkomm) == True:
```

```
def checkuser(givenname):
    validnames = [ "heinj", "sarstein", "spiasko", "celich", "sommerda", "fiscmanu" ]
    if givenname in validnames:
        return True
    else:
        return False
```

Funktionen mit Parametern

```
username = input("Geben Sie Ihren Benutzernamen ein:")
```

```
Geben Sie Ihren Benutzernamen ein: dkomm
```

```
username = dkomm
```

```
if checkuser(dkomm) == True:
```

```
def checkuser(dkomm):
    validnames = [ "heinj", "sarstein", "spiasko", "celich", "sommerda", "fiscmanu" ]
    if givenname in validnames:
        return True
    else:
        return False
```

Funktionen mit Parametern

```
username = input("Geben Sie Ihren Benutzernamen ein:")
```

```
Geben Sie Ihren Benutzernamen ein: dkomm
```

```
username = dkomm
```

```
if checkuser(dkomm) == True:
```

```
def checkuser(dkomm):
    validnames = [ "heinj", "sarstein", "spiasko", "celich", "sommerda", "fiscmanu" ]
    if dkomm in validnames:
        return True
    else:
        return False
```

Funktionen mit Parametern

```
username = input("Geben Sie Ihren Benutzernamen ein:")
```

```
Geben Sie Ihren Benutzernamen ein: dkomm
```

```
username = dkomm
```

```
if checkuser(dkomm) == True:
```

```
def checkuser(dkomm):
    validnames = [ "heinj", "sarstein", "spiasko", "celich", "sommerda", "fiscmanu" ]
    if dkomm in validnames:
        return True
    else:
        return False
```

Funktionen mit Parametern

```
username = input("Geben Sie Ihren Benutzernamen ein:")
```

```
Geben Sie Ihren Benutzernamen ein: dkomm
```

```
username = dkomm
```

```
if checkuser(dkomm) == True:
```

```
def checkuser(dkomm):
    validnames = [ "heinj", "sarstein", "spiasko", "celich", "sommerda", "fiscmanu" ]
    if dkomm in validnames:
        return True
    else:
        return False
```

```
if checkuser(dkomm) == True:
```

Funktionen mit Parametern

```
username = input("Geben Sie Ihren Benutzernamen ein:")
```

```
Geben Sie Ihren Benutzernamen ein: dkomm
```

```
username = dkomm
```

```
if checkuser(dkomm) == True:
```

```
def checkuser(dkomm):
    validnames = [ "heinj", "sarstein", "spiasko", "celich", "sommerda", "fiscmanu" ]
    if dkomm in validnames:
        return True
    else:
        return False
```

```
if False == True:
```

Funktionen mit Parametern

```
username = input("Geben Sie Ihren Benutzernamen ein:")
```

```
Geben Sie Ihren Benutzernamen ein: dkomm
```

```
username = dkomm
```

```
if checkuser(dkomm) == True:
```

```
def checkuser(dkomm):
    validnames = [ "heinj", "sarstein", "spiasko", "celich", "sommerda", "fiscmanu" ]
    if dkomm in validnames:
        return True
    else:
        return False
```

```
if False == True:
```

```
Benutzername nicht gefunden.
```

Definition von Funktionen

Funktion muss definiert werden, **bevor** sie verwendet werden kann

Definition von Funktionen

Funktion muss definiert werden, **bevor** sie verwendet werden kann

```
def f(x):
    return 2 * x + 1

print(f(2))
```

funktioniert, aber nicht. . .

Definition von Funktionen

Funktion muss definiert werden, **bevor** sie verwendet werden kann

```
def f(x):
    return 2 * x + 1

print(f(2))
```

funktioniert, aber nicht...

```
print(f(2))

def f(x):
    return 2 * x + 1
```

Definition von Funktionen

Funktion muss definiert werden, **bevor** sie verwendet werden kann

```
def f(x):
    return 2 * x + 1

print(f(2))
```

funktioniert, aber nicht...

```
print(f(2))

def f(x):
    return 2 * x + 1
```

```
NameError: name 'f' is not defined
```

Funktionen

Beispiel – Keksrechner

Beispiel – Keksrechner

```
kinder = int(input("Anzahl Kinder:"))
kekse = int(input("Anzahl Kekse:"))

print("Jedes Kind kriegt", kekse // kinder, "Kekse")
print("Papa kriegt", kekse % kinder, "Kekse")
```

Beispiel – Keksrechner

```
kinder = int(input("Anzahl Kinder:"))
kekse = int(input("Anzahl Kekse:"))

print("Jedes Kind kriegt", kekse // kinder, "Kekse")
print("Papa kriegt", kekse % kinder, "Kekse")
```

Wir wollen sicherstellen, dass `kinder` positiv ist und jedes Kind mindestens einen Keks kriegt

Keksrechner – Eingabeprüfung

Aus . . .

```
kinder = int(input("Anzahl Kinder:"))
```

Keksrechner – Eingabeprüfung

Aus . . .

```
kinder = int(input("Anzahl Kinder:"))
```

. . . wird demnach

Keksrechner – Eingabeprüfung

Aus . . .

```
kinder = int(input("Anzahl Kinder:"))
```

. . . wird demnach

```
while True:
    kinder = int(input("Anzahl Kinder:"))
    if kinder >= 1:
        break
    else:
        print("Wert muss mindestens 1 sein")
```

Keksrechner – Eingabeprüfung

Aus . . .

```
kinder = int(input("Anzahl Kinder:"))
```

. . . wird demnach

```
while True:
    kinder = int(input("Anzahl Kinder:"))
    if kinder >= 1:
        break
    else:
        print("Wert muss mindestens 1 sein")
```

Analog dazu müssen wir prüfen, dass `kekse >= kinder` ist

Keksrechner – Es wird unübersichtlich

```
while True:
    kinder = int(input("Anzahl Kinder:"))
    if kinder >= 1:
        break
    else:
        print("Wert muss mindestens 1 sein")

while True:
    kekse = int(input("Anzahl Kekse:"))
    if kekse >= kinder:
        break
    else:
        print("Wert muss mindestens", kinder, "sein")

print("Jedes Kind kriegt", kekse // kinder, "Kekse")
print("Papa kriegt", kekse % kinder, "Kekse")
```

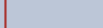
Keksrechner – Es wird unübersichtlich

```
while True:
    kinder = int(input("Anzahl Kinder:"))
    if kinder >= 1:
        break
    else:
        print("Wert muss mindestens 1 sein")

while True:
    kekse = int(input("Anzahl Kekse:"))
    if kekse >= kinder:
        break
    else:
        print("Wert muss mindestens", kinder, "sein")

print("Jedes Kind kriegt", kekse // kinder, "Kekse")
print("Papa kriegt", kekse % kinder, "Kekse")
```

Anzahl Kinder einlesen und prüfen

Keksrechner – Es wird unübersichtlich

```
while True:
    kinder = int(input("Anzahl Kinder:"))
    if kinder >= 1:
        break
    else:
        print("Wert muss mindestens 1 sein")

while True:
    kekse = int(input("Anzahl Kekse:"))
    if kekse >= kinder:
        break
    else:
        print("Wert muss mindestens", kinder, "sein")

print("Jedes Kind kriegt", kekse // kinder, "Kekse")
print("Papa kriegt", kekse % kinder, "Kekse")
```



Anzahl Kekse einlesen und prüfen

Keksrechner – Erkenntnisse

- Die beiden Code-Fragmente sind **fast identisch**

Keksrechner – Erkenntnisse

- Die beiden Code-Fragmente sind **fast identisch**
- Folgende Aspekte sind unterschiedlich:
  - Der Prompt, also `"Kinder:"` vs. `"Kekse:"`
  - Das Minimum, also `1` vs. `kinder`

Keksrechner – Erkenntnisse

- Die beiden Code-Fragmente sind **fast identisch**
- Folgende Aspekte sind unterschiedlich:
  - Der Prompt, also `"Kinder:"` vs. `"Kekse:"`
  - Das Minimum, also `1` vs. `kinder`
- Wir können das Code-Fragment in eine Funktion auslagern und somit **wiederverwenden**

Keksrechner – Erkenntnisse

- Die beiden Code-Fragmente sind **fast identisch**
- Folgende Aspekte sind unterschiedlich:
  - Der Prompt, also `"Kinder:"` vs. `"Kekse:"`
  - Das Minimum, also `1` vs. `kinder`
- Wir können das Code-Fragment in eine Funktion auslagern und somit **wiederverwenden**
- Dabei müssen wir die unterschiedlichen Aspekte **parametrisieren**

Übung – Keksrechner

Schreiben Sie eine Funktion, die

- zwei Parameter `prompt` und `minimum` erhält
- den User bittet, eine Zahl einzugeben
- die Eingabe per `return` zurückgibt, wenn sie mindestens `minimum` ist
- sonst erneut um eine Eingabe bittet

Verwenden Sie Ihre Funktion im Keksrechner

Übung – Keksrechner

```
def checkinput(prompt, minimum):
    while True:
        x = int(input(prompt))
        if x >= minimum:
            return x
        else:
            print("Wert muss mindestens", minimum, "sein")


kinder = checkinput("Anzahl Kinder:", 1)
kekse = checkinput("Anzahl Kekse:", kinder)

print("Jedes Kind kriegt", kekse // kinder, "Kekse")
print("Papa kriegt", kekse % kinder, "Kekse")
```

Funktionen

Scope und Lifetime von Variablen

Lokale Variablen

Parameter einer Funktion sind nur innerhalb der Funktion gültig

```
def f(x):
    return x + 5

print(x)
```

Lokale Variablen

Parameter einer Funktion sind nur innerhalb der Funktion gültig

```
def f(x):
    return x + 5

print(x)
```

```
NameError: name 'x' is not defined
```

Lokale Variablen

Dasselbe gilt für Variablen, die in einer Funktion definiert werden

```
def f(x):
    y = 5
    return x + y

print(y)
```

Lokale Variablen

Dasselbe gilt für Variablen, die in einer Funktion definiert werden

```
def f(x):
    y = 5
    return x + y

print(y)
```

```
NameError: name 'y' is not defined
```

Lokale Variablen

- Solche Variablen (Parameter) heissen **lokale Variablen**
- Ausserhalb einer Funktion definierte Variablen heissen **globale Variablen**
- Gültigkeitsbereich einer Variablen heisst **Scope**
- Zeit, in der Variable existiert, heisst ihre **Lifetime**

Lokale Variablen

- Solche Variablen (Parameter) heissen **lokale Variablen**
- Ausserhalb einer Funktion definierte Variablen heissen **globale Variablen**
- Gültigkeitsbereich einer Variablen heisst **Scope**
- Zeit, in der Variable existiert, heisst ihre **Lifetime**

```
def f(x):
    if x < 0:
        return -100
    y = x + 1
    if y < 10:
        y += 10
    else:
        y -= 20
    return y
```

Lokale Variablen

- Solche Variablen (Parameter) heissen **lokale Variablen**
- Ausserhalb einer Funktion definierte Variablen heissen **globale Variablen**
- Gültigkeitsbereich einer Variablen heisst **Scope**
- Zeit, in der Variable existiert, heisst ihre **Lifetime**

Scope von x

```
def f(x):
    if x < 0:
        return -100
    y = x + 1
    if y < 10:
        y += 10
    else:
        y -= 20
    return y
```

Lokale Variablen

- Solche Variablen (Parameter) heissen **lokale Variablen**
- Ausserhalb einer Funktion definierte Variablen heissen **globale Variablen**
- Gültigkeitsbereich einer Variablen heisst **Scope**
- Zeit, in der Variable existiert, heisst ihre **Lifetime**

Scope von x

```
def f(x):
    if x < 0:
        return -100
    y = x + 1
    if y < 10:
        y += 10
    else:
        y -= 20
    return y
```

```
def f(x):
    if x < 0:
        return -100
    y = x + 1
    if y < 10:
        y += 10
    else:
        y -= 20
    return y
```

Lokale Variablen

- Solche Variablen (Parameter) heissen **lokale Variablen**
- Ausserhalb einer Funktion definierte Variablen heissen **globale Variablen**
- Gültigkeitsbereich einer Variablen heisst **Scope**
- Zeit, in der Variable existiert, heisst ihre **Lifetime**

Scope von x

```
def f(x):
    if x < 0:
        return -100
    y = x + 1
    if y < 10:
        y += 10
    else:
        y -= 20
    return y
```

Scope von y

```
def f(x):
    if x < 0:
        return -100
    y = x + 1
    if y < 10:
        y += 10
    else:
        y -= 20
    return y
```

Globale Variablen

Funktionen können auf globale Variablen zugreifen

Globale Variablen

Funktionen können auf globale Variablen zugreifen

```
x = 1

def f():
    y = x + 1
    print(y)
    return

print(x)
f()
print(x)
```

Globale Variablen

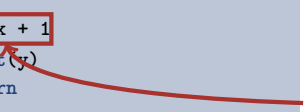
Funktionen können auf globale Variablen zugreifen

```
x = 1

def f():
    y = x + 1
    print(y)
    return

print(x)
f()
print(x)
```

Globale Variablen

Funktionen können auf globale Variablen zugreifen

```
x = 1

def f():
    y = x + 1
    print(y)
    return

print(x)
f()
print(x)
```



Lokale Variable `y` erhält Wert,
der von globaler Variablen `x` abhängt

Globale Variablen

Funktionen können auf globale Variablen zugreifen

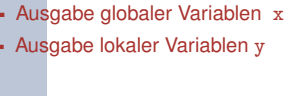
```
x = 1

def f():
    y = x + 1
    print(y)
    return

print(x)
f()
print(x)
```

Ausgabe globaler Variablen `x`

Globale Variablen

Funktionen können auf globale Variablen zugreifen

```
x = 1

def f():
    y = x + 1
    print(y)
    return

print(x)
f()
print(x)
```



Globale Variablen

Funktionen können auf globale Variablen zugreifen

```
x = 1

def f():
    y = x + 1
    print(y)
    return

print(x)
f()
print(x)
```

Ausgabe globaler Variablen `x`

Ausgabe lokaler Variablen `y`

Ausgabe globaler Variablen `x`

Globale Variablen

Funktionen können auf globale Variablen zugreifen

```
x = 1

def f():
    y = x + 1
    print(y)
    return

print(x)
f()
print(x)
```

```
x = 1

def f(y):
    z = x + y
    return z

print(x)
print(f(2))
print(x)
```

Globale Variablen

Funktionen können auf globale Variablen zugreifen

```
x = 1

def f():
    y = x + 1
    print(y)
    return

print(x)
f()
print(x)
```

```
x = 1

def f(y):
    z = x + y
    return z

print(x)
print(f(2))
print(x)
```

Globale Variablen

Funktionen können auf globale Variablen zugreifen

```
x = 1

def f():
    y = x + 1
    print(y)
    return

print(x)
f()
print(x)
```

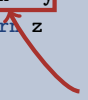
```
x = 1

def f(y):
    z = x + y
    return z

print(x)
print(f(2))
print(x)
```

Globale Variablen

Funktionen können auf globale Variablen zugreifen

```
x = 1

def f():
    y = x + 1
    print(y)
    return

print(x)
f()
print(x)
```

```
x = 1

def f(y):
    z = x + y
    return z

print(x)
print(f(2))
print(x)
```



Lokale Variable `z` erhält Wert, der von globaler Variablen `x` und Parameter `y` abhängt

Globale Variablen

Funktionen können auf globale Variablen zugreifen

```
x = 1

def f():
    y = x + 1
    print(y)
    return

print(x)
f()
print(x)
```

```
x = 1

def f(y):
    z = x + y
    return z

print(x)
print(f(2))
print(x)
```

Ausgabe globaler Variablen x

Globale Variablen

Funktionen können auf globale Variablen zugreifen

```
x = 1

def f():
    y = x + 1
    print(y)
    return

print(x)
f()
print(x)
```

```
x = 1

def f(y):
    z = x + y
    return z

print(x)
print(f(2))
print(x)
```

Ausgabe globaler Variablen `x`

Ausgabe lokaler Variablen `z`

Globale Variablen

Funktionen können auf globale Variablen zugreifen

```
x = 1

def f():
    y = x + 1
    print(y)
    return

print(x)
f()
print(x)
```

```
x = 1

def f(y):
    z = x + y
    return z

print(x)    <-- Ausgabe globaler Variablen x
print(f(2)) <-- Ausgabe lokaler Variablen z
print(x)    <-- Ausgabe globaler Variablen x
```

Lokale Variablen

Lokale und globale Variablen dürfen denselben Namen haben

Lokale Variablen

Lokale und globale Variablen dürfen denselben Namen haben

- **Shadowing**
- Nicht verboten, aber sollte hier möglichst vermieden werden

Lokale Variablen

Lokale und globale Variablen dürfen denselben Namen haben

- **Shadowing**
- Nicht verboten, aber sollte hier möglichst vermieden werden

```
x = 1

def f():
    x = 2
    return x

print(x)
print(f())
print(x)
```

Lokale Variablen

Lokale und globale Variablen dürfen denselben Namen haben

- **Shadowing**
- Nicht verboten, aber sollte hier möglichst vermieden werden

```
x = 1

def f():
    x = 2
    return x

print(x)
print(f())
print(x)
```

Lokale Variablen

Lokale und globale Variablen dürfen denselben Namen haben

- **Shadowing**
- Nicht verboten, aber sollte hier möglichst vermieden werden

```
x = 1

def f():
    x = 2
    return x

print(x)
print(f())
print(x)
```

Lokale Variablen

Lokale und globale Variablen dürfen denselben Namen haben

- **Shadowing**
- Nicht verboten, aber sollte hier möglichst vermieden werden

```
x = 1

def f():
    x = 2
    return x

print(x)
print(f())
print(x)
```

Lokale Variablen

Lokale und globale Variablen dürfen denselben Namen haben

- **Shadowing**
- Nicht verboten, aber sollte hier möglichst vermieden werden

```
x = 1

def f():
    x = 2
    return x

print(x)
print(f())
print(x)
```

Ausgabe globaler Variablen `x`

Lokale Variablen

Lokale und globale Variablen dürfen denselben Namen haben

- **Shadowing**
- Nicht verboten, aber sollte hier möglichst vermieden werden

```
x = 1

def f():
    x = 2
    return x

print(x)
print(f())
print(x)
```

Lokale Variablen

Lokale und globale Variablen dürfen denselben Namen haben

- **Shadowing**
- Nicht verboten, aber sollte hier möglichst vermieden werden

```
x = 1

def f():
    x = 2
    return x

print(x)
print(f())
print(x)
```

Ausgabe globaler Variablen `x`

Ausgabe lokaler Variablen `x`

Ausgabe globaler Variablen `x`

Lokale Variablen

Lokale und globale Variablen dürfen denselben Namen haben

- **Shadowing**
- Nicht verboten, aber sollte hier möglichst vermieden werden

```
x = 1

def f():
    x = 2
    return x

print(x)
print(f())
print(x)
```

```
x = 1

def f(x):
    x = x + 1
    return x

print(x)
print(f(2))
print(x)
```

Lokale Variablen

Lokale und globale Variablen dürfen denselben Namen haben

- **Shadowing**
- Nicht verboten, aber sollte hier möglichst vermieden werden

```
x = 1

def f():
    x = 2
    return x

print(x)
print(f())
print(x)
```

```
x = 1

def f(x):
    x = x + 1
    return x

print(x)
print(f(2))
print(x)
```

Lokale Variablen

Lokale und globale Variablen dürfen denselben Namen haben

- **Shadowing**
- Nicht verboten, aber sollte hier möglichst vermieden werden

```
x = 1

def f():
    x = 2
    return x

print(x)
print(f())
print(x)
```

```
x = 1

def f(x):
    x = x + 1
    return x

print(x)
print(f(2))
print(x)
```

Lokale Variablen

Lokale und globale Variablen dürfen denselben Namen haben

- **Shadowing**
- Nicht verboten, aber sollte hier möglichst vermieden werden

```
x = 1

def f():
    x = 2
    return x

print(x)
print(f())
print(x)
```

```
x = 1

def f(x):
    x = x + 1
    return x

print(x)
print(f(2))
print(x)
```

Parameter `x` erhält neuen Wert, der von seinem aktuellen Wert abhängt

Lokale Variablen

Lokale und globale Variablen dürfen denselben Namen haben

- **Shadowing**
- Nicht verboten, aber sollte hier möglichst vermieden werden

```
x = 1

def f():
    x = 2
    return x

print(x)
print(f())
print(x)
```

```
x = 1

def f(x):
    x = x + 1
    return x

print(x)
print(f(2))
print(x)
```

Parameter `x` wird zurückgegeben

Lokale Variablen

Lokale und globale Variablen dürfen denselben Namen haben

- **Shadowing**
- Nicht verboten, aber sollte hier möglichst vermieden werden

```
x = 1

def f():
    x = 2
    return x

print(x)
print(f())
print(x)
```

```
x = 1

def f(x):
    x = x + 1
    return x

print(x)
print(f(2))
print(x)
```

Ausgabe globaler Variablen `x`

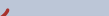
Lokale Variablen

Lokale und globale Variablen dürfen denselben Namen haben

- **Shadowing**
- Nicht verboten, aber sollte hier möglichst vermieden werden

```
x = 1

def f():
    x = 2
    return x

print(x)
print(f())
print(x)
```

```
x = 1

def f(x):
    x = x + 1
    return x

print(x)
print(f(2))
print(x)
```



Lokale Variablen

Lokale und globale Variablen dürfen denselben Namen haben

- **Shadowing**
- Nicht verboten, aber sollte hier möglichst vermieden werden

```
x = 1

def f():
    x = 2
    return x

print(x)
print(f())
print(x)
```

```
x = 1

def f(x):
    x = x + 1
    return x

print(x)    <-- Ausgabe globaler Variablen x
print(f(2)) <-- Ausgabe Parameter x
print(x)    <-- Ausgabe globaler Variablen x
```

Danke für die Aufmerksamkeit