合肥大學
HEFEI UNIVERSITY

Programming with Python

22. Alternativen mit `if`

Thomas Weise (汤卫思)
tweise@hfuu.edu.cn

Institute of Applied Optimization (IAO)
School of Artificial Intelligence and Big Data
Hefei University
Hefei, Anhui, China

应用优化研究所
人工智能与大数据学院
合肥大学
中国安徽省合肥市

Version: 2025-09-05

Programming with Python

Dies ist ein Kurs über das Programmieren mit der Programmiersprache Python an der Universität Hefei (合肥大学).

Die Webseite mit dem Lehrmaterial dieses Kurses ist https://thomasweise.github.io/programmingWithPython (siehe auch den QR-Kode unten rechts). Dort können Sie das Kursbuch (in Englisch) und diese Slides finden. Das Repository mit den Beispielprogrammen in Python finden Sie unter https://github.com/thomasWeise/programmingWithPythonCode.

Outline

Einleitung

Einleitung

- Mit den Dingen, die wir bisher gelernt haben, können wir einfache lineare Programme implementieren.

Einleitung

- Mit den Dingen, die wir bisher gelernt haben, können wir einfache lineare Programme implementieren.
- Diese Programme führen Befehl nach Befehl aus, genau in der Reihenfolge, in der wie diese aufgeschrieben haben.

Einleitung

- Mit den Dingen, die wir bisher gelernt haben, können wir einfache lineare Programme implementieren.
- Diese Programme führen Befehl nach Befehl aus, genau in der Reihenfolge, in der wie diese aufgeschrieben haben.
- Wenn wir einen Befehl zweimal ausführen wollen, dann müssen wir ihn zweimal hinschreiben.

Einleitung

- Mit den Dingen, die wir bisher gelernt haben, können wir einfache lineare Programme implementieren.
- Diese Programme führen Befehl nach Befehl aus, genau in der Reihenfolge, in der wie diese aufgeschrieben haben.
- Wenn wir einen Befehl zweimal ausführen wollen, dann müssen wir ihn zweimal hinschreiben.
- Unsere Programme können ihr Verhalten nicht basierend auf den Eingabedaten ändern.

Einleitung

- Mit den Dingen, die wir bisher gelernt haben, können wir einfache lineare Programme implementieren.
- Diese Programme führen Befehl nach Befehl aus, genau in der Reihenfolge, in der wie diese aufgeschrieben haben.
- Wenn wir einen Befehl zweimal ausführen wollen, dann müssen wir ihn zweimal hinschreiben.
- Unsere Programme können ihr Verhalten nicht basierend auf den Eingabedaten ändern.
- Sie können nicht Befehl $\mathcal{A}$ ausführen, wenn eine Variable einen bestimmten Wert hat, und sonst Befehl $\mathcal{B}$.

Einleitung

- Mit den Dingen, die wir bisher gelernt haben, können wir einfache lineare Programme implementieren.
- Diese Programme führen Befehl nach Befehl aus, genau in der Reihenfolge, in der wie diese aufgeschrieben haben.
- Wenn wir einen Befehl zweimal ausführen wollen, dann müssen wir ihn zweimal hinschreiben.
- Unsere Programme können ihr Verhalten nicht basierend auf den Eingabedaten ändern.
- Sie können nicht Befehl $\mathcal{A}$ ausführen, wenn eine Variable einen bestimmten Wert hat, und sonst Befehl $\mathcal{B}$.
- Sie können weder verzweigen noch Schritte wiederholen.

Einleitung

- Mit den Dingen, die wir bisher gelernt haben, können wir einfache lineare Programme implementieren.
- Diese Programme führen Befehl nach Befehl aus, genau in der Reihenfolge, in der wie diese aufgeschrieben haben.
- Wenn wir einen Befehl zweimal ausführen wollen, dann müssen wir ihn zweimal hinschreiben.
- Unsere Programme können ihr Verhalten nicht basierend auf den Eingabedaten ändern.
- Sie können nicht Befehl $\mathcal{A}$ ausführen, wenn eine Variable einen bestimmten Wert hat, und sonst Befehl $\mathcal{B}$.
- Sie können weder verzweigen noch Schritte wiederholen.
- In den folgenden Einheiten werden wir lernen, wie man verzweigt und wiederholt.

Einleitung

- Mit den Dingen, die wir bisher gelernt haben, können wir einfache lineare Programme implementieren.
- Diese Programme führen Befehl nach Befehl aus, genau in der Reihenfolge, in der wie diese aufgeschrieben haben.
- Wenn wir einen Befehl zweimal ausführen wollen, dann müssen wir ihn zweimal hinschreiben.
- Unsere Programme können ihr Verhalten nicht basierend auf den Eingabedaten ändern.
- Sie können nicht Befehl $\mathcal{A}$ ausführen, wenn eine Variable einen bestimmten Wert hat, und sonst Befehl $\mathcal{B}$.
- Sie können weder verzweigen noch Schritte wiederholen.
- In den folgenden Einheiten werden wir lernen, wie man verzweigt und wiederholt.
- Wir lernen Befehle, die den Kontrollfluss ändern.

Alternativen

- Alternativen erlauben uns, ein Stück Kode *nur dann* auszuführen, wenn ein bestimmter Boolescher Ausdruck `True` ergibt.

Alternativen

- Alternativen erlauben uns, ein Stück Kode *nur dann* auszuführen, wenn ein bestimmter Boolescher Ausdruck `True` ergibt.
- Somit können unsere Programme Entscheidungen treffen.

Alternativen

- Alternativen erlauben uns, ein Stück Kode *nur dann* auszuführen, wenn ein bestimmter Boolescher Ausdruck `True` ergibt.
- Somit können unsere Programme Entscheidungen treffen.
- Sie können einen Befehl in einer Situation $\mathcal{A}$ ausführen und einen Befehl $\mathcal{B}$ in einer anderen Situation.

Alternativen

- Alternativen erlauben uns, ein Stück Kode *nur dann* auszuführen, wenn ein bestimmter Boolescher Ausdruck `True` ergibt.
- Somit können unsere Programme Entscheidungen treffen.
- Sie können einen Befehl in einer Situation $\mathcal{A}$ ausführen und einen Befehl $\mathcal{B}$ in einer anderen Situation.
- Alternativen sind somit die fundamentalsten Kontrollflussstatements.

Einfache Alternative mit `if`

Einfache Alternative mit `if`

- Eine einfache Alternative ist ein Stück Programkode das eine Menge von Befehlen ausführt wenn eine Kondition zutrifft.

Einfache Alternative mit `if`

- Eine einfache Alternative ist ein Stück Programkode das eine Menge von Befehlen ausführt wenn eine Kondition zutrifft.
- Die Kondition ist ein einfacher Boolescher Ausdruck, wie wir ihn bereits in Einheit 10 (`bool`) diskutiert haben.

Einfache Alternative mit `if`

- Eine einfache Alternative ist ein Stück Programkode das eine Menge von Befehlen ausführt wenn eine Kondition zutrifft.
- Die Kondition ist ein einfacher Boolescher Ausdruck, wie wir ihn bereits in Einheit 10 (`bool`) diskutiert haben.
- Die Syntax dafür ist sehr einfach[17]:

```
"""The syntax of an "if"-statement in Python."""

if booleanExpression:
    conditional statement 1  # Only executed if booleanExpression
    conditional statement 2  # evaluates to True.
    # ...

normal statement 1  # Always executed, regardless of the result of
normal statement 2  # booleanExpression.
# ...
```

Einfache Alternative mit `if`

- Die erste Zeile beginnt mit `if` gefolgt von einem Booleschen Ausdruck, gefolgt von einem Doppelpunkt (`:`).

```
"""The syntax of an "if"-statement in Python."""

if booleanExpression:
    conditional statement 1  # Only executed if booleanExpression
    conditional statement 2  # evaluates to True.
    # ...

normal statement 1  # Always executed, regardless of the result of
normal statement 2  # booleanExpression.
# ...
```

Einfache Alternative mit `if`

- Die erste Zeile beginnt mit `if` gefolgt von einem Booleschen Ausdruck, gefolgt von einem Doppelpunkt (`:`).
- Nur wenn der Boolesche Ausdruck `True` ergibt, dann wird der *eingerückte* Block von Kode unter dem `if` ausgeführt.

```
"""The syntax of an "if"-statement in Python."""

if booleanExpression:
    conditional statement 1  # Only executed if booleanExpression
    conditional statement 2  # evaluates to True.
    # ...

normal statement 1  # Always executed, regardless of the result of
normal statement 2  # booleanExpression.
# ...
```

Einfache Alternative mit `if`

- Die erste Zeile beginnt mit `if` gefolgt von einem Booleschen Ausdruck, gefolgt von einem Doppelpunkt (`:`).
- Nur wenn der Boolesche Ausdruck `True` ergibt, dann wird der *eingerückte* Block von Kode unter dem `if` ausgeführt.

Gute Praxis

Blöcke von Kode werden mit vier Leerzeichen eingerückt[31].

Einfache Alternative mit `if`

- Die erste Zeile beginnt mit `if` gefolgt von einem Booleschen Ausdruck, gefolgt von einem Doppelpunkt (`:`).
- Nur wenn der Boolesche Ausdruck `True` ergibt, dann wird der *eingerückte* Block von Kode unter dem `if` ausgeführt.
- Jede konditionelle Zeile Kode des `if`-Statements ist mit vier Leerzeichen eingerückt.

```
1  """The syntax of an "if"-statement in Python."""
2
3  if booleanExpression:
4      conditional statement 1  # Only executed if booleanExpression
5      conditional statement 2  # evaluates to True.
6      # ...
7
8  normal statement 1  # Always executed, regardless of the result of
9  normal statement 2  # booleanExpression.
10 # ...
```

Einfache Alternative mit `if`

- Die erste Zeile beginnt mit `if` gefolgt von einem Booleschen Ausdruck, gefolgt von einem Doppelpunkt (`:`).
- Nur wenn der Boolesche Ausdruck `True` ergibt, dann wird der *eingerückte* Block von Kode unter dem `if` ausgeführt.
- Jede konditionelle Zeile Kode des `if`-Statements ist mit vier Leerzeichen eingerückt.
- Nach dem eingerückten Block folgt normaler Programmkode.

```
"""The syntax of an "if"-statement in Python."""

if booleanExpression:
    conditional statement 1  # Only executed if booleanExpression
    conditional statement 2  # evaluates to True.
    # ...

normal statement 1  # Always executed, regardless of the result of
normal statement 2  # booleanExpression.
# ...
```

Einfache Alternative mit `if`

- Nur wenn der Boolesche Ausdruck `True` ergibt, dann wird der *eingerückte* Block von Kode unter dem `if` ausgeführt.
- Jede konditionelle Zeile Kode des `if`-Statements ist mit vier Leerzeichen eingerückt.
- Nach dem eingerückten Block folgt normaler Programmkode.
- Dieser Kode ist nicht eingerückt und wird ausgeführt, egal was der Boolesche Ausdruck im `if` ergeben hat.

```
"""The syntax of an "if"-statement in Python."""

if booleanExpression:
    conditional statement 1  # Only executed if booleanExpression
    conditional statement 2  # evaluates to True.
    # ...

normal statement 1  # Always executed, regardless of the result of
normal statement 2  # booleanExpression.
# ...
```

Beispiel

- Benutzen wir doch mal ein `if` um zu prüfen, ob ein Jahr ein Schaltjahr ist.

```python
"""
An example of using the `if` statement.

A year is a leap year if it is divisible by 4 but not divisible by 100
or if it is divisible by 400.
We can use the modulo operator `%` to check this.
`a % 100` will be `0` if and only `a` is a multiple of `100`.
The Boolean statements can be combined with `and` and `or` and grouped
using parentheses.
"""

year: int = 2024  # The year 2024 should be a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line will be executed.

year = 1900       # The year 1900 is not a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line is never reached.

print("End of year checking.")  # This text is always printed.
```

↓ `python3 if_example.py` ↓

```
2024 is a leap year.
End of year checking.
```

Beispiel

- Benutzen wir doch mal ein `if` um zu prüfen, ob ein Jahr ein Schaltjahr ist.
- Nach dem Gregorianischen Kalender ist ein Jahr y ein Schaltjahr wenn y durch 4 geteilt werden kann, aber nicht durch 100; oder wenn es durch 400 geteilt werden kann[23].

```python
"""
An example of using the `if` statement.

A year is a leap year if it is divisible by 4 but not divisible by 100
or if it is divisible by 400.
We can use the modulo operator `%` to check this.
`a % 100` will be `0` if and only `a` is a multiple of `100`.
The Boolean statements can be combined with `and` and `or` and grouped
using parentheses.
"""

year: int = 2024  # The year 2024 should be a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line will be executed.

year = 1900       # The year 1900 is not a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line is never reached.

print("End of year checking.")  # This text is always printed.
```

↓ `python3 if_example.py` ↓

```
2024 is a leap year.
End of year checking.
```

Beispiel

- Benutzen wir doch mal ein `if` um zu prüfen, ob ein Jahr ein Schaltjahr ist.
- Nach dem Gregorianischen Kalender ist ein Jahr y ein Schaltjahr wenn y durch 4 geteilt werden kann, aber nicht durch 100; oder wenn es durch 400 geteilt werden kann[23].
- Zuerst speichern wir das Jahr, was wir untersuchen wollen, in einer Variable `year`.

```
"""
An example of using the `if` statement.

A year is a leap year if it is divisible by 4 but not divisible by 100
or if it is divisible by 400.
We can use the modulo operator `%` to check this.
`a % 100` will be `0` if and only `a` is a multiple of `100`.
The Boolean statements can be combined with `and` and `or` and grouped
using parentheses.
"""

year: int = 2024  # The year 2024 should be a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line will be executed.

year = 1900       # The year 1900 is not a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line is never reached.

print("End of year checking.")  # This text is always printed.
```

↓ `python3 if_example.py` ↓

```
2024 is a leap year.
End of year checking.
```

Beispiel

- Benutzen wir doch mal ein `if` um zu prüfen, ob ein Jahr ein Schaltjahr ist.
- Nach dem Gregorianischen Kalender ist ein Jahr y ein Schaltjahr wenn y durch 4 geteilt werden kann, aber nicht durch 100; oder wenn es durch 400 geteilt werden kann[23].
- Zuerst speichern wir das Jahr, was wir untersuchen wollen, in einer Variable `year`.
- Aus der 7. Einheit erinnern Sie sich vielleicht noch an den Modulo-Operator `%`.

```python
"""
An example of using the `if` statement.

A year is a leap year if it is divisible by 4 but not divisible by 100
or if it is divisible by 400.
We can use the modulo operator `%` to check this.
`a % 100` will be `0` if and only `a` is a multiple of `100`.
The Boolean statements can be combined with `and` and `or` and grouped
using parentheses.
"""

year: int = 2024  # The year 2024 should be a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line will be executed.

year = 1900       # The year 1900 is not a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line is never reached.

print("End of year checking.")  # This text is always printed.
```

↓ `python3 if_example.py` ↓

```
2024 is a leap year.
End of year checking.
```

Beispiel

- Benutzen wir doch mal ein `if` um zu prüfen, ob ein Jahr ein Schaltjahr ist.
- Nach dem Gregorianischen Kalender ist ein Jahr y ein Schaltjahr wenn y durch 4 geteilt werden kann, aber nicht durch 100; oder wenn es durch 400 geteilt werden kann[23].
- Zuerst speichern wir das Jahr, was wir untersuchen wollen, in einer Variable `year`.
- Wenn wir den Modulo-Operator auf zwei Werte `a` und `b` anwenden, liefert er uns den Rest der Division `a // b`.

```python
"""
An example of using the `if` statement.

A year is a leap year if it is divisible by 4 but not divisible by 100
or if it is divisible by 400.
We can use the modulo operator `%` to check this.
`a % 100` will be `0` if and only `a` is a multiple of `100`.
The Boolean statements can be combined with `and` and `or` and grouped
using parentheses.
"""

year: int = 2024  # The year 2024 should be a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line will be executed.

year = 1900       # The year 1900 is not a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line is never reached.

print("End of year checking.")  # This text is always printed.
```

↓ `python3 if_example.py` ↓

```
2024 is a leap year.
End of year checking.
```

Beispiel

- Benutzen wir doch mal ein `if` um zu prüfen, ob ein Jahr ein Schaltjahr ist.
- Nach dem Gregorianischen Kalender ist ein Jahr y ein Schaltjahr wenn y durch 4 geteilt werden kann, aber nicht durch 100; oder wenn es durch 400 geteilt werden kann[23].
- Zuerst speichern wir das Jahr, was wir untersuchen wollen, in einer Variable `year`.
- Wenn wir den Modulo-Operator auf zwei Werte `a` und `b` anwenden, liefert er uns den Rest der Division `a // b`.
- Wenn der Rest 0 ist, dann ist `a` durch `b` teilbar.

```python
"""
An example of using the `if` statement.

A year is a leap year if it is divisible by 4 but not divisible by 100
or if it is divisible by 400.
We can use the modulo operator `%` to check this.
`a % 100` will be `0` if and only `a` is a multiple of `100`.
The Boolean statements can be combined with `and` and `or` and grouped
using parentheses.
"""

year: int = 2024  # The year 2024 should be a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line will be executed.

year = 1900       # The year 1900 is not a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line is never reached.

print("End of year checking.")  # This text is always printed.
```

↓ `python3 if_example.py` ↓

```
2024 is a leap year.
End of year checking.
```

Beispiel

- Nach dem Gregorianischen Kalender ist ein Jahr y ein Schaltjahr wenn y durch 4 geteilt werden kann, aber nicht durch 100; oder wenn es durch 400 geteilt werden kann[23].
- Zuerst speichern wir das Jahr, was wir untersuchen wollen, in einer Variable `year`.
- Wenn wir den Modulo-Operator auf zwei Werte `a` und `b` anwenden, liefert er uns den Rest der Division `a // b`.
- Wenn der Rest 0 ist, dann ist `a` durch `b` teilbar.
- Zum Beispiel `10 % 5 == 0` aber `10 % 6 == 4` und `10 % 3 == 1`.

```
"""
An example of using the `if` statement.

A year is a leap year if it is divisible by 4 but not divisible by 100
or if it is divisible by 400.
We can use the modulo operator `%` to check this.
`a % 100` will be `0` if and only `a` is a multiple of `100`.
The Boolean statements can be combined with `and` and `or` and grouped
using parentheses.
"""

year: int = 2024  # The year 2024 should be a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line will be executed.

year = 1900       # The year 1900 is not a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line is never reached.

print("End of year checking.")  # This text is always printed.
```

↓ `python3 if_example.py` ↓

```
2024 is a leap year.
End of year checking.
```

Beispiel

- Zuerst speichern wir das Jahr, was wir untersuchen wollen, in einer Variable `year`.
- Wenn wir den Modulo-Operator auf zwei Werte `a` und `b` anwenden, liefert er uns den Rest der Division `a // b`.
- Wenn der Rest 0 ist, dann ist `a` durch `b` teilbar.
- Zum Beispiel `10 % 5 == 0` aber `10 % 6 == 4` und `10 % 3 == 1`.
- Um zu prüfen, ob `year` durch 4 teilbar ist, brauchen wir nur zu prüfen ob `(year %4) == 0`.

```python
"""
An example of using the `if` statement.

A year is a leap year if it is divisible by 4 but not divisible by 100
or if it is divisible by 400.
We can use the modulo operator `%` to check this.
`a % 100` will be `0` if and only `a` is a multiple of `100`.
The Boolean statements can be combined with `and` and `or` and grouped
using parentheses.
"""

year: int = 2024  # The year 2024 should be a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line will be executed.

year = 1900       # The year 1900 is not a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line is never reached.

print("End of year checking.")  # This text is always printed.
```

↓ `python3 if_example.py` ↓

```
2024 is a leap year.
End of year checking.
```

Beispiel

- Wenn wir den Modulo-Operator auf zwei Werte `a` und `b` anwenden, liefert er uns den Rest der Division `a // b`.
- Wenn der Rest 0 ist, dann ist `a` durch `b` teilbar.
- Zum Beispiel `10 % 5 == 0` aber `10 % 6 == 4` und `10 % 3 == 1`.
- Um zu prüfen, ob `year` durch 4 teilbar ist, brauchen wir nur zu prüfen ob `(year %4) == 0`.
- `year` soll nicht durch 100 teilbar sein, also prüfen wir auch `(year %100) != 0` (ein Rest ungleich 0 bedeutet "nicht teilbar").

```
"""
An example of using the `if` statement.

A year is a leap year if it is divisible by 4 but not divisible by 100
or if it is divisible by 400.
We can use the modulo operator `%` to check this.
`a % 100` will be `0` if and only `a` is a multiple of `100`.
The Boolean statements can be combined with `and` and `or` and grouped
using parentheses.
"""

year: int = 2024  # The year 2024 should be a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line will be executed.

year = 1900       # The year 1900 is not a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line is never reached.

print("End of year checking.")  # This text is always printed.
```

↓ `python3 if_example.py` ↓

```
2024 is a leap year.
End of year checking.
```

Beispiel

- Wenn der Rest 0 ist, dann ist `a` durch `b` teilbar.
- Zum Beispiel `10 % 5 == 0` aber `10 % 6 == 4` und `10 % 3 == 1`.
- Um zu prüfen, ob `year` durch 4 teilbar ist, brauchen wir nur zu prüfen ob `(year %4) == 0`.
- `year` soll nicht durch 100 teilbar sein, also prüfen wir auch `(year %100) != 0` (ein Rest ungleich 0 bedeutet "nicht teilbar").
- Wir müssen jetzt beide Bedingungen mit `and` kombinieren und schreiben `((year %4) == 0) and ((year %100) != 0)`.

```python
"""
An example of using the `if` statement.

A year is a leap year if it is divisible by 4 but not divisible by 100
or if it is divisible by 400.
We can use the modulo operator `%` to check this.
`a % 100` will be `0` if and only `a` is a multiple of `100`.
The Boolean statements can be combined with `and` and `or` and grouped
using parentheses.
"""

year: int = 2024  # The year 2024 should be a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line will be executed.

year = 1900       # The year 1900 is not a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line is never reached.

print("End of year checking.")  # This text is always printed.
```

↓ `python3 if_example.py` ↓

```
2024 is a leap year.
End of year checking.
```

Beispiel

- Um zu prüfen, ob `year` durch 4 teilbar ist, brauchen wir nur zu prüfen ob `(year %4) == 0`.
- `year` soll nicht durch 100 teilbar sein, also prüfen wir auch `(year %100) != 0` (ein Rest ungleich 0 bedeutet "nicht teilbar").
- Wir müssen jetzt beide Bedingungen mit `and` kombinieren und schreiben `((year %4) == 0) and ((year %100) != 0)`.
- Das Ergebnis ist nur dann `True`, wenn beide Bedingungen `True` sind, also wenn das Jahr durch 4 aber nicht durch 100 teilbar ist.

```python
"""
An example of using the `if` statement.

A year is a leap year if it is divisible by 4 but not divisible by 100
or if it is divisible by 400.
We can use the modulo operator `%` to check this.
`a % 100` will be `0` if and only `a` is a multiple of `100`.
The Boolean statements can be combined with `and` and `or` and grouped
using parentheses.
"""

year: int = 2024  # The year 2024 should be a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line will be executed.

year = 1900       # The year 1900 is not a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line is never reached.

print("End of year checking.")  # This text is always printed.
```

↓ `python3 if_example.py` ↓

```
2024 is a leap year.
End of year checking.
```

Beispiel

- `year` soll nicht durch 100 teilbar sein, also prüfen wir auch `(year %100) != 0` (ein Rest ungleich 0 bedeutet “nicht teilbar”).
- Wir müssen jetzt beide Bedingungen mit `and` kombinieren und schreiben `((year %4) == 0) and ((year %100)!= 0)`.
- Das Ergebnis ist nur dann `True`, wenn beide Bedingungen `True` sind, also wenn das Jahr durch 4 aber nicht durch 100 teilbar ist.
- Wir benutzen Klammern um de Ausdruck einfacher lesbar zu machen.

```
"""
An example of using the `if` statement.

A year is a leap year if it is divisible by 4 but not divisible by 100
or if it is divisible by 400.
We can use the modulo operator `%` to check this.
`a % 100` will be `0` if and only `a` is a multiple of `100`.
The Boolean statements can be combined with `and` and `or` and grouped
using parentheses.
"""

year: int = 2024  # The year 2024 should be a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line will be executed.

year = 1900       # The year 1900 is not a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line is never reached.

print("End of year checking.")  # This text is always printed.
```

↓ `python3 if_example.py` ↓

```
2024 is a leap year.
End of year checking.
```

Beispiel

- Wir müssen jetzt beide Bedingungen mit `and` kombinieren und schreiben `((year %4) == 0) and ((year %100)!= 0)`.
- Das Ergebnis ist nur dann `True`, wenn beide Bedingungen `True` sind, also wenn das Jahr durch 4 aber nicht durch 100 teilbar ist.
- Wir benutzen Klammern um de Ausdruck einfacher lesbar zu machen.
- Wenn diese kombinierte Bedingung `False` ist, dann könnte das Jahr immer noch ein Schaltjahr sein, nämlich wenn es durch 400 teilbar ist.

```python
"""
An example of using the `if` statement.

A year is a leap year if it is divisible by 4 but not divisible by 100
or if it is divisible by 400.
We can use the modulo operator `%` to check this.
`a % 100` will be `0` if and only `a` is a multiple of `100`.
The Boolean statements can be combined with `and` and `or` and grouped
using parentheses.
"""

year: int = 2024  # The year 2024 should be a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line will be executed.

year = 1900       # The year 1900 is not a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line is never reached.

print("End of year checking.")  # This text is always printed.
```

↓ `python3 if_example.py` ↓

```
2024 is a leap year.
End of year checking.
```

Beispiel

- Das Ergebnis ist nur dann `True`, wenn beide Bedingungen `True` sind, also wenn das Jahr durch 4 aber nicht durch 100 teilbar ist.
- Wir benutzen Klammern um de Ausdruck einfacher lesbar zu machen.
- Wenn diese kombinierte Bedingung `False` ist, dann könnte das Jahr immer noch ein Schaltjahr sein, nämlich wenn es durch 400 teilbar ist.
- Die letzte Bedingung ist also, zu prüfen ob `(year %400) == 0`.

```python
"""
An example of using the `if` statement.

A year is a leap year if it is divisible by 4 but not divisible by 100
or if it is divisible by 400.
We can use the modulo operator `%` to check this.
`a % 100` will be `0` if and only `a` is a multiple of `100`.
The Boolean statements can be combined with `and` and `or` and grouped
using parentheses.
"""

year: int = 2024  # The year 2024 should be a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line will be executed.

year = 1900       # The year 1900 is not a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line is never reached.

print("End of year checking.")  # This text is always printed.
```

↓ `python3 if_example.py` ↓

```
2024 is a leap year.
End of year checking.
```

Beispiel

- Wir benutzen Klammern um de Ausdruck einfacher lesbar zu machen.
- Wenn diese kombinierte Bedingung `False` ist, dann könnte das Jahr immer noch ein Schaltjahr sein, nämlich wenn es durch 400 teilbar ist.
- Die letzte Bedingung ist also, zu prüfen ob `(year %400) == 0`.
- Wir fügen diese Bedingung mit `or` zu unserem Ausdruck hinzu, weil es ausreicht, wenn entweder unser erster Teilausdruck *oder* diese Bedigung wahr ist.

```
"""
An example of using the `if` statement.

A year is a leap year if it is divisible by 4 but not divisible by 100
or if it is divisible by 400.
We can use the modulo operator `%` to check this.
`a % 100` will be `0` if and only `a` is a multiple of `100`.
The Boolean statements can be combined with `and` and `or` and grouped
using parentheses.
"""

year: int = 2024  # The year 2024 should be a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line will be executed.

year = 1900       # The year 1900 is not a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line is never reached.

print("End of year checking.")  # This text is always printed.
```

↓ `python3 if_example.py` ↓

```
2024 is a leap year.
End of year checking.
```

Beispiel

- Wenn diese kombinierte Bedingung `False` ist, dann könnte das Jahr immer noch ein Schaltjahr sein, nämlich wenn es durch 400 teilbar ist.
- Die letzte Bedingung ist also, zu prüfen ob `(year %400) == 0`.
- Wir fügen diese Bedingung mit `or` zu unserem Ausdruck hinzu, weil es ausreicht, wenn entweder unser erster Teilausdruck *oder* diese Bedigung wahr ist.
- Wir bekommen den Ausdruck
  `(((year %4) == 0)`
  `and ((year %100) != 0))`
  `or ((year %400) == 0)`.

```
"""
An example of using the `if` statement.

A year is a leap year if it is divisible by 4 but not divisible by 100
or if it is divisible by 400.
We can use the modulo operator `%` to check this.
`a % 100` will be `0` if and only `a` is a multiple of `100`.
The Boolean statements can be combined with `and` and `or` and grouped
using parentheses.
"""

year: int = 2024  # The year 2024 should be a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line will be executed.

year = 1900       # The year 1900 is not a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line is never reached.

print("End of year checking.")  # This text is always printed.
```

↓ `python3 if_example.py` ↓

```
2024 is a leap year.
End of year checking.
```

Beispiel

- Die letzte Bedingung ist also, zu prüfen ob `(year %400) == 0`.
- Wir fügen diese Bedingung mit `or` zu unserem Ausdruck hinzu, weil es ausreicht, wenn entweder unser erster Teilausdruck *oder* diese Bedigung wahr ist.
- Wir bekommen den Ausdruck
  `(((year %4) == 0)`
  `and ((year %100) != 0))`
  `or ((year %400) == 0)`.
- Wir nutzen wieder Klammern um die Teilausdrücke schon für die Lesbarkeit zu gruppieren.

```
"""
An example of using the `if` statement.

A year is a leap year if it is divisible by 4 but not divisible by 100
or if it is divisible by 400.
We can use the modulo operator `%` to check this.
`a % 100` will be `0` if and only `a` is a multiple of `100`.
The Boolean statements can be combined with `and` and `or` and grouped
using parentheses.
"""

year: int = 2024  # The year 2024 should be a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line will be executed.

year = 1900       # The year 1900 is not a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line is never reached.

print("End of year checking.")  # This text is always printed.
```

↓ `python3 if_example.py` ↓

```
2024 is a leap year.
End of year checking.
```

Beispiel

- Wir fügen diese Bedingung mit `or` zu unserem Ausdruck hinzu, weil es ausreicht, wenn entweder unser erster Teilausdruck *oder* diese Bedigung wahr ist.
- Wir bekommen den Ausdruck `(((year %4) == 0) and ((year %100) != 0)) or ((year %400) == 0)`.
- Wir nutzen wieder Klammern um die Teilausdrücke schon für die Lesbarkeit zu gruppieren.
- Alles, was wir jetzt noch machen müssen, ist ein `if` vor den Ausdruck und ein `:` dahinter zu schreiben, und schon haben wir eine Alternative konstruiert.

```
"""
An example of using the `if` statement.

A year is a leap year if it is divisible by 4 but not divisible by 100
or if it is divisible by 400.
We can use the modulo operator `%` to check this.
`a % 100` will be `0` if and only `a` is a multiple of `100`.
The Boolean statements can be combined with `and` and `or` and grouped
using parentheses.
"""

year: int = 2024  # The year 2024 should be a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line will be executed.

year = 1900       # The year 1900 is not a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line is never reached.

print("End of year checking.")  # This text is always printed.
```

↓ `python3 if_example.py` ↓

```
2024 is a leap year.
End of year checking.
```

Beispiel

- Wir bekommen den Ausdruck `(((year %4) == 0) and ((year %100) != 0)) or ((year %400) == 0)`.
- Wir nutzen wieder Klammern um die Teilausdrücke schon für die Lesbarkeit zu gruppieren.
- Alles, was wir jetzt noch machen müssen, ist ein `if` vor den Ausdruck und ein `:` dahinter zu schreiben, und schon haben wir eine Alternative konstruiert.
- Wenn der Ausdruck `True` ergibt, dann drucken wir `f"{year}is a leap year."`.

```
"""
An example of using the `if` statement.

A year is a leap year if it is divisible by 4 but not divisible by 100
or if it is divisible by 400.
We can use the modulo operator `%` to check this.
`a % 100` will be `0` if and only `a` is a multiple of `100`.
The Boolean statements can be combined with `and` and `or` and grouped
using parentheses.
"""

year: int = 2024  # The year 2024 should be a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line will be executed.

year = 1900       # The year 1900 is not a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line is never reached.

print("End of year checking.")  # This text is always printed.
```

↓ `python3 if_example.py` ↓

```
2024 is a leap year.
End of year checking.
```

Beispiel

- Wir nutzen wieder Klammern um die Teilausdrücke schon für die Lesbarkeit zu gruppieren.
- Alles, was wir jetzt noch machen müssen, ist ein `if` vor den Ausdruck und ein `:` dahinter zu schreiben, und schon haben wir eine Alternative konstruiert.
- Wenn der Ausdruck `True` ergibt, dann drucken wir `f"{year}is a leap year."`.
- Dafür müssen wir das `print` Kommando mit vier Leerzeichen einrücken.

```
"""
An example of using the `if` statement.

A year is a leap year if it is divisible by 4 but not divisible by 100
or if it is divisible by 400.
We can use the modulo operator `%` to check this.
`a % 100` will be `0` if and only `a` is a multiple of `100`.
The Boolean statements can be combined with `and` and `or` and grouped
using parentheses.
"""

year: int = 2024  # The year 2024 should be a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line will be executed.

year = 1900       # The year 1900 is not a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line is never reached.

print("End of year checking.")  # This text is always printed.
```

↓ `python3 if_example.py` ↓

```
2024 is a leap year.
End of year checking.
```

Beispiel

- Wir nutzen wieder Klammern um die Teilausdrücke schon für die Lesbarkeit zu gruppieren.
- Alles, was wir jetzt noch machen müssen, ist ein `if` vor den Ausdruck und ein `:` dahinter zu schreiben, und schon haben wir eine Alternative konstruiert.
- Wenn der Ausdruck `True` ergibt, dann drucken wir `f"{year}is a leap year."`.
- Dafür müssen wir das `print` Kommando mit vier Leerzeichen einrücken.
- Wir kopieren diesen Kode zweimal und prüfen die Jahre `2024` und `1900`.

```
"""
An example of using the `if` statement.

A year is a leap year if it is divisible by 4 but not divisible by 100
or if it is divisible by 400.
We can use the modulo operator `%` to check this.
`a % 100` will be `0` if and only `a` is a multiple of `100`.
The Boolean statements can be combined with `and` and `or` and grouped
using parentheses.
"""

year: int = 2024  # The year 2024 should be a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line will be executed.

year = 1900       # The year 1900 is not a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line is never reached.

print("End of year checking.")  # This text is always printed.
```

↓ `python3 if_example.py` ↓

```
2024 is a leap year.
End of year checking.
```

Beispiel

- Alles, was wir jetzt noch machen müssen, ist ein `if` vor den Ausdruck und ein `:` dahinter zu schreiben, und schon haben wir eine Alternative konstruiert.
- Wenn der Ausdruck `True` ergibt, dann drucken wir `f"{year}is a leap year."`.
- Dafür müssen wir das `print` Kommando mit vier Leerzeichen einrücken.
- Wir kopieren diesen Kode zweimal und prüfen die Jahre `2024` und `1900`.
- Das zweite `if` beginnt mit der selben Einrückung wie das erste `if`.

```
"""
An example of using the `if` statement.

A year is a leap year if it is divisible by 4 but not divisible by 100
or if it is divisible by 400.
We can use the modulo operator `%` to check this.
`a % 100` will be `0` if and only `a` is a multiple of `100`.
The Boolean statements can be combined with `and` and `or` and grouped
using parentheses.
"""

year: int = 2024  # The year 2024 should be a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line will be executed.

year = 1900       # The year 1900 is not a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line is never reached.

print("End of year checking.")  # This text is always printed.
```

↓ `python3 if_example.py` ↓

```
2024 is a leap year.
End of year checking.
```

Beispiel

- Wenn der Ausdruck `True` ergibt, dann drucken wir `f"{year}is a leap year."`.
- Dafür müssen wir das `print` Kommando mit vier Leerzeichen einrücken.
- Wir kopieren diesen Kode zweimal und prüfen die Jahre `2024` und `1900`.
- Das zweite `if` beginnt mit der selben Einrückung wie das erste `if`.
- Es wir ausgeführt, gleichgültit wie die erste Alternative ausgegangen ist.

```
"""
An example of using the `if` statement.

A year is a leap year if it is divisible by 4 but not divisible by 100
or if it is divisible by 400.
We can use the modulo operator `%` to check this.
`a % 100` will be `0` if and only `a` is a multiple of `100`.
The Boolean statements can be combined with `and` and `or` and grouped
using parentheses.
"""

year: int = 2024  # The year 2024 should be a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line will be executed.

year = 1900       # The year 1900 is not a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line is never reached.

print("End of year checking.")  # This text is always printed.
```

↓ `python3 if_example.py` ↓

```
2024 is a leap year.
End of year checking.
```

Beispiel

- Dafür müssen wir das `print` Kommando mit vier Leerzeichen einrücken.
- Wir kopieren diesen Kode zweimal und prüfen die Jahre `2024` und `1900`.
- Das zweite `if` beginnt mit der selben Einrückung wie das erste `if`.
- Es wir ausgeführt, gleichgültit wie die erste Alternative ausgegangen ist.
- Am Ende geben wir noch `End of year checking.` auf der Konsole aus.

```python
"""
An example of using the `if` statement.

A year is a leap year if it is divisible by 4 but not divisible by 100
or if it is divisible by 400.
We can use the modulo operator `%` to check this.
`a % 100` will be `0` if and only `a` is a multiple of `100`.
The Boolean statements can be combined with `and` and `or` and grouped
using parentheses.
"""

year: int = 2024  # The year 2024 should be a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line will be executed.

year = 1900       # The year 1900 is not a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line is never reached.

print("End of year checking.")  # This text is always printed.
```

↓ `python3 if_example.py` ↓

```
2024 is a leap year.
End of year checking.
```

Beispiel

- Wir kopieren diesen Kode zweimal und prüfen die Jahre 2024 und 1900.
- Das zweite `if` beginnt mit der selben Einrückung wie das erste `if`.
- Es wir ausgeführt, gleichgültit wie die erste Alternative ausgegangen ist.
- Am Ende geben wir noch `End of year checking.` auf der Konsole aus.
- Diese letzte Zeile ist auch nicht eingerückt, wird also auf jeden Fall ausgeführt.

```
"""
An example of using the `if` statement.

A year is a leap year if it is divisible by 4 but not divisible by 100
or if it is divisible by 400.
We can use the modulo operator `%` to check this.
`a % 100` will be `0` if and only `a` is a multiple of `100`.
The Boolean statements can be combined with `and` and `or` and grouped
using parentheses.
"""

year: int = 2024  # The year 2024 should be a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line will be executed.

year = 1900       # The year 1900 is not a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line is never reached.

print("End of year checking.")  # This text is always printed.
```

↓ `python3 if_example.py` ↓

```
2024 is a leap year.
End of year checking.
```

Beispiel

- Wir kopieren diesen Kode zweimal und prüfen die Jahre 2024 und 1900.
- Das zweite `if` beginnt mit der selben Einrückung wie das erste `if`.
- Es wir ausgeführt, gleichgültit wie die erste Alternative ausgegangen ist.
- Am Ende geben wir noch `End of year checking.` auf der Konsole aus.
- Diese letzte Zeile ist auch nicht eingerückt, wird also auf jeden Fall ausgeführt.
- Wir sehen, das 2024 ein Schaltjahr war, 1900 aber nicht.

```
"""
An example of using the `if` statement.

A year is a leap year if it is divisible by 4 but not divisible by 100
or if it is divisible by 400.
We can use the modulo operator `%` to check this.
`a % 100` will be `0` if and only `a` is a multiple of `100`.
The Boolean statements can be combined with `and` and `or` and grouped
using parentheses.
"""

year: int = 2024  # The year 2024 should be a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line will be executed.

year = 1900       # The year 1900 is not a leap year.
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line is never reached.

print("End of year checking.")  # This text is always printed.
```

↓ `python3 if_example.py` ↓

```
2024 is a leap year.
End of year checking.
```

Kombinierte Alternative mit `if-else`

Kombinierte Alternative mit `if-else`

- Die Möglichkeit, eine Aktion auszuführen, wenn eine Bedingung wahr ist, ist schonmal gut.

Kombinierte Alternative mit `if-else`

- Die Möglichkeit, eine Aktion auszuführen, wenn eine Bedingung wahr ist, ist schonmal gut.
- Oftmals wollen wir aber eine Aktion ausführen, wenn die Bedingung wahr ist und eine andere Aktion, wenn die Bedingung falsch ist.

Kombinierte Alternative mit `if-else`

- Die Möglichkeit, eine Aktion auszuführen, wenn eine Bedingung wahr ist, ist schonmal gut.
- Oftmals wollen wir aber eine Aktion ausführen, wenn die Bedingung wahr ist und eine andere Aktion, wenn die Bedingung falsch ist.
- Technisch gesehen können wir das auch jetzt schon.

Kombinierte Alternative mit `if-else`

- Die Möglichkeit, eine Aktion auszuführen, wenn eine Bedingung wahr ist, ist schonmal gut.
- Oftmals wollen wir aber eine Aktion ausführen, wenn die Bedingung wahr ist und eine andere Aktion, wenn die Bedingung falsch ist.
- Technisch gesehen können wir das auch jetzt schon.
- Wir brauchen einfach nur zwei Alternativen mit der selben Bedingung `if` zu definieren, wobei wir bei der zweiten einfach `not` vor die Bedigung schreiben.

Kombinierte Alternative mit `if-else`

- Die Möglichkeit, eine Aktion auszuführen, wenn eine Bedingung wahr ist, ist schonmal gut.
- Oftmals wollen wir aber eine Aktion ausführen, wenn die Bedingung wahr ist und eine andere Aktion, wenn die Bedingung falsch ist.
- Technisch gesehen können wir das auch jetzt schon.
- Wir brauchen einfach nur zwei Alternativen mit der selben Bedingung `if` zu definieren, wobei wir bei der zweiten einfach `not` vor die Bedigung schreiben.
- Natürlich geht das auch einfacher: mit `if...else`[17].

Kombinierte Alternative mit `if-else`

- Natürlich geht das auch einfacher: mit `if...else`[17].

```
"""The syntax of an "if-else"-statement in Python."""

if booleanExpression:
    conditional statement 1  # Only executed if booleanExpression
    conditional statement 2  # evaluates to True.
    # ...
else:
    alternative statement 1  # Only executed if booleanExpression
    alternative statement 2  # evaluates to False.
    # ...

normal statement 1  # Always executed, regardless of the result of
normal statement 2  # booleanExpression.
```

Kombinierte Alternative mit `if-else`

- Natürlich geht das auch einfacher: mit `if...else`[17].
- Das `if...else`-Statement beginnt genau wie ein normales `if`-Statement.

```
"""The syntax of an "if-else"-statement in Python."""

if booleanExpression:
    conditional statement 1  # Only executed if booleanExpression
    conditional statement 2  # evaluates to True.
    # ...
else:
    alternative statement 1  # Only executed if booleanExpression
    alternative statement 2  # evaluates to False.
    # ...

normal statement 1  # Always executed, regardless of the result of
normal statement 2  # booleanExpression.
```

Kombinierte Alternative mit `if-else`

- Natürlich geht das auch einfacher: mit `if...else`[17].
- Das `if...else`-Statement beginnt genau wie ein normales `if`-Statement.
- `if`, gefolgt von einem Booleschen Ausdruck, gefolgt vom Doppelpunkt (`:`) stellt die Bedingung dar.

```
"""The syntax of an "if-else"-statement in Python."""

if booleanExpression:
    conditional statement 1  # Only executed if booleanExpression
    conditional statement 2  # evaluates to True.
    # ...
else:
    alternative statement 1  # Only executed if booleanExpression
    alternative statement 2  # evaluates to False.
    # ...

normal statement 1  # Always executed, regardless of the result of
normal statement 2  # booleanExpression.
```

Kombinierte Alternative mit `if-else`

- Natürlich geht das auch einfacher: mit `if...else`[17].
- Das `if...else`-Statement beginnt genau wie ein normales `if`-Statement.
- `if`, gefolgt von einem Booleschen Ausdruck, gefolgt vom Doppelpunkt (`:`) stellt die Bedingung dar.
- Der folgende Block, wo jedes Statement mit vier Leerzeichen eingerückt ist, wird nur ausgeführt, wenn der Boolesche Ausdruck `True` ergibt.

```
"""The syntax of an "if-else"-statement in Python."""

if booleanExpression:
    conditional statement 1  # Only executed if booleanExpression
    conditional statement 2  # evaluates to True.
    # ...
else:
    alternative statement 1  # Only executed if booleanExpression
    alternative statement 2  # evaluates to False.
    # ...

normal statement 1  # Always executed, regardless of the result of
normal statement 2  # booleanExpression.
```

Kombinierte Alternative mit `if-else`

- Das `if...else`-Statement beginnt genau wie ein normales `if`-Statement.
- `if`, gefolgt von einem Booleschen Ausdruck, gefolgt vom Doppelpunkt (`:`) stellt die Bedingung dar.
- Der folgende Block, wo jedes Statement mit vier Leerzeichen eingerückt ist, wird nur ausgeführt, wenn der Boolesche Ausdruck `True` ergibt.
- Dann folgt die `else:` Zeile.

```
"""The syntax of an "if-else"-statement in Python."""

if booleanExpression:
    conditional statement 1  # Only executed if booleanExpression
    conditional statement 2  # evaluates to True.
    # ...
else:
    alternative statement 1  # Only executed if booleanExpression
    alternative statement 2  # evaluates to False.
    # ...

normal statement 1  # Always executed, regardless of the result of
normal statement 2  # booleanExpression.
```

Kombinierte Alternative mit `if-else`

- `if`, gefolgt von einem Booleschen Ausdruck, gefolgt vom Doppelpunkt (`:`) stellt die Bedingung dar.
- Der folgende Block, wo jedes Statement mit vier Leerzeichen eingerückt ist, wird nur ausgeführt, wenn der Boolesche Ausdruck `True` ergibt.
- Dann folgt die `else:` Zeile.
- Diese ist *nicht* eingerückt und befindet sich in der selben Ebene wie das `if`.

```
"""The syntax of an "if-else"-statement in Python."""

if booleanExpression:
    conditional statement 1  # Only executed if booleanExpression
    conditional statement 2  # evaluates to True.
    # ...
else:
    alternative statement 1  # Only executed if booleanExpression
    alternative statement 2  # evaluates to False.
    # ...

normal statement 1  # Always executed, regardless of the result of
normal statement 2  # booleanExpression.
```

Kombinierte Alternative mit `if-else`

- Der folgende Block, wo jedes Statement mit vier Leerzeichen eingerückt ist, wird nur ausgeführt, wenn der Boolesche Ausdruck `True` ergibt.
- Dann folgt die `else:` Zeile.
- Diese ist *nicht* eingerückt und befindet sich in der selben Ebene wie das `if`.
- Danach kommt wieder ein ein mit vier Leerzeichen eingerückter Block.

```
"""The syntax of an "if-else"-statement in Python."""

if booleanExpression:
    conditional statement 1  # Only executed if booleanExpression
    conditional statement 2  # evaluates to True.
    # ...
else:
    alternative statement 1  # Only executed if booleanExpression
    alternative statement 2  # evaluates to False.
    # ...

normal statement 1  # Always executed, regardless of the result of
normal statement 2  # booleanExpression.
```

Kombinierte Alternative mit `if-else`

- Dann folgt die `else:` Zeile.
- Diese ist *nicht* eingerückt und befindet sich in der selben Ebene wie das `if`.
- Danach kommt wieder ein ein mit vier Leerzeichen eingerückter Block.
- Dieser Block wird nur dann ausgeführt, wenn der Boolesche Ausdruck im `if` `False` ergibt.

```
"""The syntax of an "if-else"-statement in Python."""

if booleanExpression:
    conditional statement 1  # Only executed if booleanExpression
    conditional statement 2  # evaluates to True.
    # ...
else:
    alternative statement 1  # Only executed if booleanExpression
    alternative statement 2  # evaluates to False.
    # ...

normal statement 1  # Always executed, regardless of the result of
normal statement 2  # booleanExpression.
```

Beispiel

- Wir passen unser Schaltjahr-Program an, so dass es auch Text ausgibt, wenn das Jahr kein Schaltjahr ist.

```python
"""An example of using the `if-else` statement."""

year: int = 2024  # the year 2024 should be a leap year
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line will be executed.
    print("yes, it really is.")       # This line as well.
else:  # If we get here, it's not a leap year.
    print(f"{year} is not a leap year.")  # This line is never reached.
    print("Believe you me, it indeed is not.")    # also never reached.

year = 1900  # the year 1900 is not a leap year
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line is never reached.
    print("yes, it really is.")       # This line is never reached.
else:  # If we get here, it's not a leap year..
    print(f"{year} is not a leap year.")  # This line will be executed.
    print("Believe you me, it indeed is not.")  # This line too.

print("End of year checking.")  # This text is always printed.
```

↓ python3 if_else_example.py ↓

```
2024 is a leap year.
yes, it really is.
1900 is not a leap year.
Believe you me, it indeed is not.
End of year checking.
```

Beispiel

- Wir passen unser Schaltjahr-Program an, so dass es auch Text ausgibt, wenn das Jahr kein Schaltjahr ist.
- Das können wir mit dem neuen `else`-Zweig machen.

```python
"""An example of using the `if-else` statement."""

year: int = 2024  # the year 2024 should be a leap year
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line will be executed.
    print("yes, it really is.")       # This line as well.
else:  # If we get here, it's not a leap year.
    print(f"{year} is not a leap year.")  # This line is never reached.
    print("Believe you me, it indeed is not.")    # also never reached.

year = 1900  # the year 1900 is not a leap year
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line is never reached.
    print("yes, it really is.")       # This line is never reached.
else:  # If we get here, it's not a leap year..
    print(f"{year} is not a leap year.")  # This line will be executed.
    print("Believe you me, it indeed is not.")  # This line too.

print("End of year checking.")  # This text is always printed.
```

↓ `python3 if_else_example.py` ↓

```
2024 is a leap year.
yes, it really is.
1900 is not a leap year.
Believe you me, it indeed is not.
End of year checking.
```

Beispiel

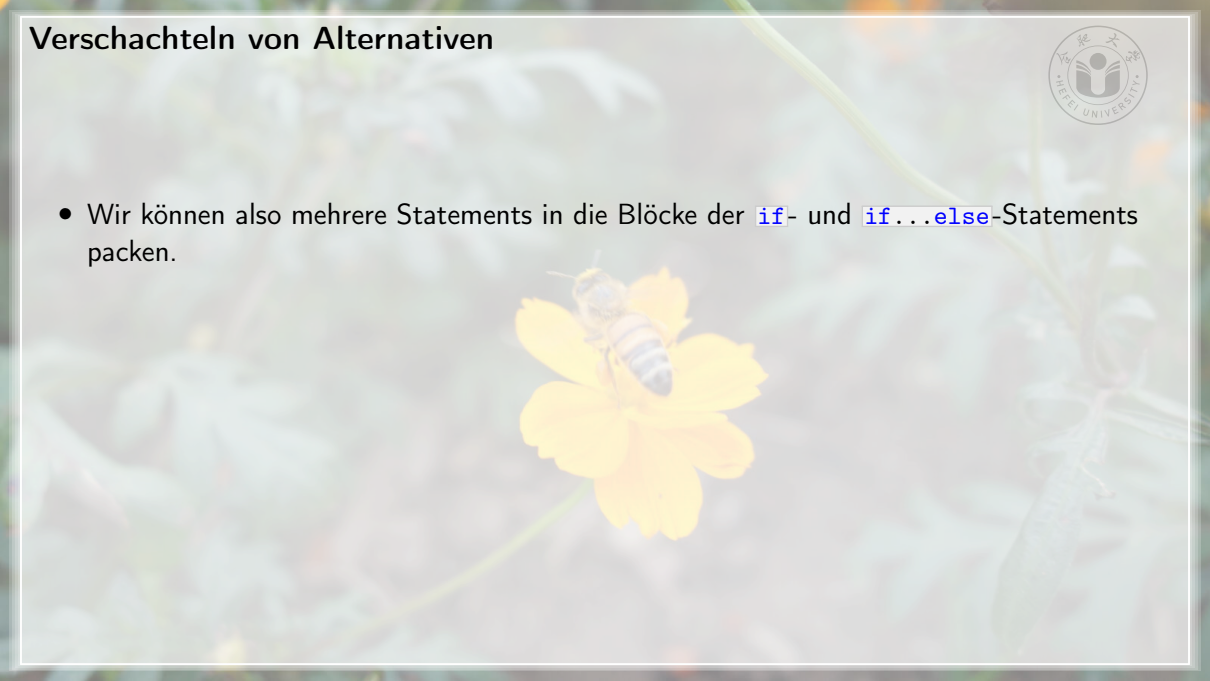
- Wir passen unser Schaltjahr-Program an, so dass es auch Text ausgibt, wenn das Jahr kein Schaltjahr ist.
- Das können wir mit dem neuen `else`-Zweig machen.
- Wir zeigen auch, dass mehrere Statements sowohl im `if`- als auch im `else`-Zweig platziert werden können.

```
"""An example of using the `if-else` statement."""

year: int = 2024  # the year 2024 should be a leap year
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line will be executed.
    print("yes, it really is.")       # This line as well.
else:  # If we get here, it's not a leap year.
    print(f"{year} is not a leap year.")  # This line is never reached.
    print("Believe you me, it indeed is not.")    # also never reached.

year = 1900  # the year 1900 is not a leap year
if (((year % 4) == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(f"{year} is a leap year.")  # This line is never reached.
    print("yes, it really is.")       # This line is never reached.
else:  # If we get here, it's not a leap year..
    print(f"{year} is not a leap year.")  # This line will be executed.
    print("Believe you me, it indeed is not.")  # This line too.

print("End of year checking.")  # This text is always printed.
```

↓ `python3 if_else_example.py` ↓

```
2024 is a leap year.
yes, it really is.
1900 is not a leap year.
Believe you me, it indeed is not.
End of year checking.
```

Verschachteln von Alternativen



- Wir können also mehrere Statements in die Blöcke der `if`- und `if...else`-Statements packen.

Verschachteln von Alternativen

- Wir können also mehrere Statements in die Blöcke der `if`- und `if...else`-Statements packen.
- Moment.

Verschachteln von Alternativen

- Wir können also mehrere Statements in die Blöcke der `if`- und `if...else`-Statements packen.
- Moment. “*`if`- und `if...else`-Statements*” ...

Verschachteln von Alternativen

- Wir können also mehrere Statements in die Blöcke der `if`- und `if...else`-Statements packen.
- Moment. "*`if`- und `if...else`-Statements*" ... das sind auch Statements?

Verschachteln von Alternativen

- Wir können also mehrere Statements in die Blöcke der `if`- und `if...else`-Statements packen.
- Moment. "*`if`- und `if...else`-Statements*" ... das sind auch Statements?
- Können wir also `if`- und `if...else`-Statements in die Kodeblöcke von `if`- und `if...else`-Statements packen?

Verschachteln von Alternativen

- Wir können also mehrere Statements in die Blöcke der `if`- und `if...else`-Statements packen.
- Moment. "*`if`- und `if...else`-Statements*" ... das sind auch Statements?
- Können wir also `if`- und `if...else`-Statements in die Kodeblöcke von `if`- und `if...else`-Statements packen?
- Können wir Alternativen verschachteln??

Verschachteln von Alternativen

- Wir können also mehrere Statements in die Blöcke der `if`- und `if...else`-Statements packen.
- Moment. "*`if`- und `if...else`-Statements*" ... das sind auch Statements?
- Können wir also `if`- und `if...else`-Statements in die Kodeblöcke von `if`- und `if...else`-Statements packen?
- Können wir Alternativen verschachteln??
- Ja, können wir.

Beispiel

- Probieren wir das mal aus in unserem Programm `if_else_nested.py`.

```python
"""An example of using the nested `if-else` statements."""

a: int = 13  # the first number
b: int = 7  # the second number
c: int = 9  # the third number

if a > b:       # This means that a > b
    if a > c:  # This means that a > b and a > c.
        print(f"{a} is the greatest number.")
    else:       # This means that a > b and c >= a.
        print(f"{c} is the greatest number.")
else:           # This means that b >= a.
    if b > c:  # This means that b >= a and b > c.
        print(f"{b} is the greatest number.")
    else:       # This means that b >= a and c >= b.
        print(f"{c} is the greatest number.")

# Some side information: The max and min function are very useful.
print(f"The maximum is {max(a, b, c)}.")  # max: get maximum of a
    ↪ sequence
print(f"The minimum is {min(a, b, c)}.")  # min: get minimum of a
    ↪ sequence
```

↓ python3 if_else_nested.py ↓

```
13 is the greatest number.
The maximum is 13.
The minimum is 7.
```

Beispiel

- Probieren wir das mal aus in unserem Programm `if_else_nested.py`.
- Wir haben drei Zahlen in den Variablen `a`, `b`, and `c`.

```python
"""An example of using the nested `if-else` statements."""

a: int = 13  # the first number
b: int = 7  # the second number
c: int = 9  # the third number

if a > b:       # This means that a > b
    if a > c:  # This means that a > b and a > c.
        print(f"{a} is the greatest number.")
    else:       # This means that a > b and c >= a.
        print(f"{c} is the greatest number.")
else:           # This means that b >= a.
    if b > c:  # This means that b >= a and b > c.
        print(f"{b} is the greatest number.")
    else:       # This means that b >= a and c >= b.
        print(f"{c} is the greatest number.")

# Some side information: The max and min function are very useful.
print(f"The maximum is {max(a, b, c)}.")  # max: get maximum of a
    ↪ sequence
print(f"The minimum is {min(a, b, c)}.")  # min: get minimum of a
    ↪ sequence
```

↓ python3 if_else_nested.py ↓

```
13 is the greatest number.
The maximum is 13.
The minimum is 7.
```

Beispiel

- Probieren wir das mal aus in unserem Programm `if_else_nested.py`.
- Wir haben drei Zahlen in den Variablen `a`, `b`, and `c`.
- Wir wollen das Maximum dieser drei Zahlen herausfinden, also die größte Zahl, die entweder in `a`, `b`, oder `c` gespeichert ist.

```python
"""An example of using the nested `if-else` statements."""

a: int = 13  # the first number
b: int = 7  # the second number
c: int = 9  # the third number

if a > b:       # This means that a > b
    if a > c:  # This means that a > b and a > c.
        print(f"{a} is the greatest number.")
    else:       # This means that a > b and c >= a.
        print(f"{c} is the greatest number.")
else:           # This means that b >= a.
    if b > c:  # This means that b >= a and b > c.
        print(f"{b} is the greatest number.")
    else:       # This means that b >= a and c >= b.
        print(f"{c} is the greatest number.")

# Some side information: The max and min function are very useful.
print(f"The maximum is {max(a, b, c)}.")  # max: get maximum of a
    ↪ sequence
print(f"The minimum is {min(a, b, c)}.")  # min: get minimum of a
    ↪ sequence
```

↓ python3 if_else_nested.py ↓

```
13 is the greatest number.
The maximum is 13.
The minimum is 7.
```

Beispiel

- Probieren wir das mal aus in unserem Programm `if_else_nested.py`.
- Wir haben drei Zahlen in den Variablen `a`, `b`, and `c`.
- Wir wollen das Maximum dieser drei Zahlen herausfinden, also die größte Zahl, die entweder in `a`, `b`, oder `c` gespeichert ist.
- Wir prüfen erstmal ob `a > b`.

```python
"""An example of using the nested `if-else` statements."""

a: int = 13  # the first number
b: int = 7  # the second number
c: int = 9  # the third number

if a > b:       # This means that a > b
    if a > c:  # This means that a > b and a > c.
        print(f"{a} is the greatest number.")
    else:       # This means that a > b and c >= a.
        print(f"{c} is the greatest number.")
else:           # This means that b >= a.
    if b > c:  # This means that b >= a and b > c.
        print(f"{b} is the greatest number.")
    else:       # This means that b >= a and c >= b.
        print(f"{c} is the greatest number.")

# Some side information: The max and min function are very useful.
print(f"The maximum is {max(a, b, c)}.")  # max: get maximum of a
    ↪ sequence
print(f"The minimum is {min(a, b, c)}.")  # min: get minimum of a
    ↪ sequence
```

↓ python3 if_else_nested.py ↓

```
13 is the greatest number.
The maximum is 13.
The minimum is 7.
```

Beispiel

- Probieren wir das mal aus in unserem Programm `if_else_nested.py`.
- Wir haben drei Zahlen in den Variablen `a`, `b`, and `c`.
- Wir wollen das Maximum dieser drei Zahlen herausfinden, also die größte Zahl, die entweder in `a`, `b`, oder `c` gespeichert ist.
- Wir prüfen erstmal ob `a > b`.
- Wenn das `True` ergibt, dann kann `b` auf keinen Fall die größte Zahl sein. Nur `a` oder `c` kommen in Frage.

```python
"""An example of using the nested `if-else` statements."""

a: int = 13  # the first number
b: int = 7  # the second number
c: int = 9  # the third number

if a > b:       # This means that a > b
    if a > c:  # This means that a > b and a > c.
        print(f"{a} is the greatest number.")
    else:       # This means that a > b and c >= a.
        print(f"{c} is the greatest number.")
else:           # This means that b >= a.
    if b > c:  # This means that b >= a and b > c.
        print(f"{b} is the greatest number.")
    else:       # This means that b >= a and c >= b.
        print(f"{c} is the greatest number.")

# Some side information: The max and min function are very useful.
print(f"The maximum is {max(a, b, c)}.")  # max: get maximum of a
    ↪ sequence
print(f"The minimum is {min(a, b, c)}.")  # min: get minimum of a
    ↪ sequence
```

↓ python3 if_else_nested.py ↓

```
13 is the greatest number.
The maximum is 13.
The minimum is 7.
```

Beispiel

- Wir haben drei Zahlen in den Variablen `a`, `b`, and `c`.
- Wir wollen das Maximum dieser drei Zahlen herausfinden, also die größte Zahl, die entweder in `a`, `b`, oder `c` gespeichert ist.
- Wir prüfen erstmal ob `a > b`.
- Wenn das `True` ergibt, dann kann `b` auf keinen Fall die größte Zahl sein. Nur `a` oder `c` kommen in Frage.
- Im Block des ersten `if`-Statements brauchen wir also nur noch zu prüfen, ob `a > c`.

```python
"""An example of using the nested `if-else` statements."""

a: int = 13  # the first number
b: int = 7  # the second number
c: int = 9  # the third number

if a > b:       # This means that a > b
    if a > c:  # This means that a > b and a > c.
        print(f"{a} is the greatest number.")
    else:       # This means that a > b and c >= a.
        print(f"{c} is the greatest number.")
else:           # This means that b >= a.
    if b > c:  # This means that b >= a and b > c.
        print(f"{b} is the greatest number.")
    else:       # This means that b >= a and c >= b.
        print(f"{c} is the greatest number.")

# Some side information: The max and min function are very useful.
print(f"The maximum is {max(a, b, c)}.")  # max: get maximum of a
    ↪ sequence
print(f"The minimum is {min(a, b, c)}.")  # min: get minimum of a
    ↪ sequence
```

↓ python3 if_else_nested.py ↓

```
13 is the greatest number.
The maximum is 13.
The minimum is 7.
```

Beispiel

- Wir wollen das Maximum dieser drei Zahlen herausfinden, also die größte Zahl, die entweder in `a`, `b`, oder `c` gespeichert ist.
- Wir prüfen erstmal ob `a > b`.
- Wenn das `True` ergibt, dann kann `b` auf keinen Fall die größte Zahl sein. Nur `a` oder `c` kommen in Frage.
- Im Block des ersten `if`-Statements brauchen wir also nur noch zu prüfen, ob `a > c`.
- Beachten Sie, dass das zweite `if`, der dazugehörige Block, das `else`, und dessen Block vier Leerzeichen tiefer eingerückt sind, als das äußere `if`.

```python
"""An example of using the nested `if-else` statements."""

a: int = 13  # the first number
b: int = 7  # the second number
c: int = 9  # the third number

if a > b:       # This means that a > b
    if a > c:  # This means that a > b and a > c.
        print(f"{a} is the greatest number.")
    else:       # This means that a > b and c >= a.
        print(f"{c} is the greatest number.")
else:           # This means that b >= a.
    if b > c:  # This means that b >= a and b > c.
        print(f"{b} is the greatest number.")
    else:       # This means that b >= a and c >= b.
        print(f"{c} is the greatest number.")

# Some side information: The max and min function are very useful.
print(f"The maximum is {max(a, b, c)}.")  # max: get maximum of a
    ↪ sequence
print(f"The minimum is {min(a, b, c)}.")  # min: get minimum of a
    ↪ sequence
```

↓ python3 if_else_nested.py ↓

```
13 is the greatest number.
The maximum is 13.
The minimum is 7.
```

Beispiel

- Wir prüfen erstmal ob `a > b`.
- Wenn das `True` ergibt, dann kann `b` auf keinen Fall die größte Zahl sein. Nur `a` oder `c` kommen in Frage.
- Im Block des ersten `if`-Statements brauchen wir also nur noch zu prüfen, ob `a > c`.
- Beachten Sie, dass das zweite `if`, der dazugehörige Block, das `else`, und dessen Block vier Leerzeichen tiefer eingerückt sind, als das äußere `if`.
- Falls `a > c` wahr ist, dann drucken wir den f-String `f"{a} is the greatest number."` aus.

```python
"""An example of using the nested `if-else` statements."""

a: int = 13  # the first number
b: int = 7  # the second number
c: int = 9  # the third number

if a > b:       # This means that a > b
    if a > c:  # This means that a > b and a > c.
        print(f"{a} is the greatest number.")
    else:       # This means that a > b and c >= a.
        print(f"{c} is the greatest number.")
else:           # This means that b >= a.
    if b > c:  # This means that b >= a and b > c.
        print(f"{b} is the greatest number.")
    else:       # This means that b >= a and c >= b.
        print(f"{c} is the greatest number.")

# Some side information: The max and min function are very useful.
print(f"The maximum is {max(a, b, c)}.")  # max: get maximum of a
    ↪ sequence
print(f"The minimum is {min(a, b, c)}.")  # min: get minimum of a
    ↪ sequence
```

↓ python3 if_else_nested.py ↓

```
13 is the greatest number.
The maximum is 13.
The minimum is 7.
```

Beispiel

- Wenn das `True` ergibt, dann kann `b` auf keinen Fall die größte Zahl sein. Nur `a` oder `c` kommen in Frage.
- Im Block des ersten `if`-Statements brauchen wir also nur noch zu prüfen, ob `a > c`.
- Beachten Sie, dass das zweite `if`, der dazugehörige Block, das `else`, und dessen Block vier Leerzeichen tiefer eingerückt sind, als das äußere `if`.
- Falls `a > c` wahr ist, dann drucken wir den f-String `f"{a} is the greatest number."` aus.
- Wenn es dagegen `False` ist, dann gilt entweder `a < c` oder `a == c`.

```python
"""An example of using the nested `if-else` statements."""

a: int = 13  # the first number
b: int = 7  # the second number
c: int = 9  # the third number

if a > b:       # This means that a > b
    if a > c:  # This means that a > b and a > c.
        print(f"{a} is the greatest number.")
    else:       # This means that a > b and c >= a.
        print(f"{c} is the greatest number.")
else:           # This means that b >= a.
    if b > c:  # This means that b >= a and b > c.
        print(f"{b} is the greatest number.")
    else:       # This means that b >= a and c >= b.
        print(f"{c} is the greatest number.")

# Some side information: The max and min function are very useful.
print(f"The maximum is {max(a, b, c)}.")  # max: get maximum of a
    ↪ sequence
print(f"The minimum is {min(a, b, c)}.")  # min: get minimum of a
    ↪ sequence
```

↓ python3 if_else_nested.py ↓

```
13 is the greatest number.
The maximum is 13.
The minimum is 7.
```

Beispiel

- Im Block des ersten `if`-Statements brauchen wir also nur noch zu prüfen, ob `a > c`.
- Beachten Sie, dass das zweite `if`, der dazugehörige Block, das `else`, und dessen Block vier Leerzeichen tiefer eingerückt sind, als das äußere `if`.
- Falls `a > c` wahr ist, dann drucken wir den f-String `f"{a} is the greatest number."` aus.
- Wenn es dagegen `False` ist, dann gilt entweder `a < c` oder `a == c`.
- So oder so, `f"{c} is the greatest number."` ist richtig.

```python
"""An example of using the nested `if-else` statements."""

a: int = 13  # the first number
b: int = 7  # the second number
c: int = 9  # the third number

if a > b:       # This means that a > b
    if a > c:  # This means that a > b and a > c.
        print(f"{a} is the greatest number.")
    else:       # This means that a > b and c >= a.
        print(f"{c} is the greatest number.")
else:           # This means that b >= a.
    if b > c:  # This means that b >= a and b > c.
        print(f"{b} is the greatest number.")
    else:       # This means that b >= a and c >= b.
        print(f"{c} is the greatest number.")

# Some side information: The max and min function are very useful.
print(f"The maximum is {max(a, b, c)}.")  # max: get maximum of a
    ↪ sequence
print(f"The minimum is {min(a, b, c)}.")  # min: get minimum of a
    ↪ sequence
```

↓ python3 if_else_nested.py ↓

```
13 is the greatest number.
The maximum is 13.
The minimum is 7.
```

Beispiel

- Falls `a > c` wahr ist, dann drucken wir den f-String `f"{a} is the greatest number."` aus.
- Wenn es dagegen `False` ist, dann gilt entweder `a < c` oder `a == c`.
- So oder so, `f"{c} is the greatest number."` ist richtig.
- Nun müssen wir wieder zum äußeren `if` zurück und den alternativen Zweig bearbeiten, den `else`-Teil.

```python
"""An example of using the nested `if-else` statements."""

a: int = 13  # the first number
b: int = 7  # the second number
c: int = 9  # the third number

if a > b:       # This means that a > b
    if a > c:  # This means that a > b and a > c.
        print(f"{a} is the greatest number.")
    else:       # This means that a > b and c >= a.
        print(f"{c} is the greatest number.")
else:           # This means that b >= a.
    if b > c:  # This means that b >= a and b > c.
        print(f"{b} is the greatest number.")
    else:       # This means that b >= a and c >= b.
        print(f"{c} is the greatest number.")

# Some side information: The max and min function are very useful.
print(f"The maximum is {max(a, b, c)}.")  # max: get maximum of a
    ↪ sequence
print(f"The minimum is {min(a, b, c)}.")  # min: get minimum of a
    ↪ sequence
```

↓ python3 if_else_nested.py ↓

```
13 is the greatest number.
The maximum is 13.
The minimum is 7.
```

Beispiel

- Falls `a > c` wahr ist, dann drucken wir den f-String `f"{a} is the greatest number."` aus.
- Wenn es dagegen `False` ist, dann gilt entweder `a < c` oder `a == c`.
- So oder so, `f"{c} is the greatest number."` ist richtig.
- Nun müssen wir wieder zum äußeren `if` zurück und den alternativen Zweig bearbeiten, den `else`-Teil.
- Was ist, wenn `a > b` nicht `True` war?

```python
"""An example of using the nested `if-else` statements."""

a: int = 13  # the first number
b: int = 7  # the second number
c: int = 9  # the third number

if a > b:       # This means that a > b
    if a > c:  # This means that a > b and a > c.
        print(f"{a} is the greatest number.")
    else:       # This means that a > b and c >= a.
        print(f"{c} is the greatest number.")
else:           # This means that b >= a.
    if b > c:  # This means that b >= a and b > c.
        print(f"{b} is the greatest number.")
    else:       # This means that b >= a and c >= b.
        print(f"{c} is the greatest number.")

# Some side information: The max and min function are very useful.
print(f"The maximum is {max(a, b, c)}.")  # max: get maximum of a
    ↪ sequence
print(f"The minimum is {min(a, b, c)}.")  # min: get minimum of a
    ↪ sequence
```

↓ python3 if_else_nested.py ↓

```
13 is the greatest number.
The maximum is 13.
The minimum is 7.
```

Beispiel

- Wenn es dagegen `False` ist, dann gilt entweder `a < c` oder `a == c`.
- So oder so, `f"{c} is the greatest number."` ist richtig.
- Nun müssen wir wieder zum äußeren `if` zurück und den alternativen Zweig bearbeiten, den `else`-Teil.
- Was ist, wenn `a > b` nicht `True` war?
- Dann gilt entweder `b > a` oder `b == a`.

```
"""An example of using the nested `if-else` statements."""

a: int = 13  # the first number
b: int = 7  # the second number
c: int = 9  # the third number

if a > b:       # This means that a > b
    if a > c:  # This means that a > b and a > c.
        print(f"{a} is the greatest number.")
    else:       # This means that a > b and c >= a.
        print(f"{c} is the greatest number.")
else:           # This means that b >= a.
    if b > c:  # This means that b >= a and b > c.
        print(f"{b} is the greatest number.")
    else:       # This means that b >= a and c >= b.
        print(f"{c} is the greatest number.")

# Some side information: The max and min function are very useful.
print(f"The maximum is {max(a, b, c)}.")  # max: get maximum of a
    ↪ sequence
print(f"The minimum is {min(a, b, c)}.")  # min: get minimum of a
    ↪ sequence
```

↓ python3 if_else_nested.py ↓

```
13 is the greatest number.
The maximum is 13.
The minimum is 7.
```

Beispiel

- So oder so, `f"{c} is the greatest number."` ist richtig.
- Nun müssen wir wieder zum äußeren `if` zurück und den alternativen Zweig bearbeiten, den `else`-Teil.
- Was ist, wenn `a > b` nicht `True` war?
- Dann gilt entweder `b > a` oder `b == a`.
- Es reicht dann, `b` mit `c` zu vergleichen.

```python
"""An example of using the nested `if-else` statements."""

a: int = 13  # the first number
b: int = 7  # the second number
c: int = 9  # the third number

if a > b:       # This means that a > b
    if a > c:  # This means that a > b and a > c.
        print(f"{a} is the greatest number.")
    else:       # This means that a > b and c >= a.
        print(f"{c} is the greatest number.")
else:           # This means that b >= a.
    if b > c:  # This means that b >= a and b > c.
        print(f"{b} is the greatest number.")
    else:       # This means that b >= a and c >= b.
        print(f"{c} is the greatest number.")

# Some side information: The max and min function are very useful.
print(f"The maximum is {max(a, b, c)}.")  # max: get maximum of a
    ↪ sequence
print(f"The minimum is {min(a, b, c)}.")  # min: get minimum of a
    ↪ sequence
```

↓ python3 if_else_nested.py ↓

```
13 is the greatest number.
The maximum is 13.
The minimum is 7.
```

Beispiel

- Nun müssen wir wieder zum äußeren `if` zurück und den alternativen Zweig bearbeiten, den `else`-Teil.
- Was ist, wenn `a > b` nicht `True` war?
- Dann gilt entweder `b > a` oder `b == a`.
- Es reicht dann, `b` mit `c` zu vergleichen.
- Wir rücken also ein weiteres `if` ein, diesmal mit der Bedingung `b > c`.

```python
"""An example of using the nested `if-else` statements."""

a: int = 13  # the first number
b: int = 7  # the second number
c: int = 9  # the third number

if a > b:       # This means that a > b
    if a > c:  # This means that a > b and a > c.
        print(f"{a} is the greatest number.")
    else:       # This means that a > b and c >= a.
        print(f"{c} is the greatest number.")
else:           # This means that b >= a.
    if b > c:  # This means that b >= a and b > c.
        print(f"{b} is the greatest number.")
    else:       # This means that b >= a and c >= b.
        print(f"{c} is the greatest number.")

# Some side information: The max and min function are very useful.
print(f"The maximum is {max(a, b, c)}.")  # max: get maximum of a
    ↪ sequence
print(f"The minimum is {min(a, b, c)}.")  # min: get minimum of a
    ↪ sequence
```

↓ python3 if_else_nested.py ↓

```
13 is the greatest number.
The maximum is 13.
The minimum is 7.
```

Beispiel

- Was ist, wenn `a > b` nicht `True` war?
- Dann gilt entweder `b > a` oder `b == a`.
- Es reicht dann, `b` mit `c` zu vergleichen.
- Wir rücken also ein weiteres `if` ein, diesmal mit der Bedingung `b > c`.
- Wenn diese `True` ist, dann ist der größte Wert gleich dem in `b` gespeicherten Wert.

```python
"""An example of using the nested `if-else` statements."""

a: int = 13  # the first number
b: int = 7  # the second number
c: int = 9  # the third number

if a > b:       # This means that a > b
    if a > c:  # This means that a > b and a > c.
        print(f"{a} is the greatest number.")
    else:       # This means that a > b and c >= a.
        print(f"{c} is the greatest number.")
else:           # This means that b >= a.
    if b > c:  # This means that b >= a and b > c.
        print(f"{b} is the greatest number.")
    else:       # This means that b >= a and c >= b.
        print(f"{c} is the greatest number.")

# Some side information: The max and min function are very useful.
print(f"The maximum is {max(a, b, c)}.")  # max: get maximum of a
    ↪ sequence
print(f"The minimum is {min(a, b, c)}.")  # min: get minimum of a
    ↪ sequence
```

↓ python3 if_else_nested.py ↓

```
13 is the greatest number.
The maximum is 13.
The minimum is 7.
```

Beispiel

- Dann gilt entweder `b > a` oder `b == a`.
- Es reicht dann, `b` mit `c` zu vergleichen.
- Wir rücken also ein weiteres `if` ein, diesmal mit der Bedingung `b > c`.
- Wenn diese `True` ist, dann ist der größte Wert gleich dem in `b` gespeicherten Wert.
- Wir können also `f"{b} is the greatest number."` drucken.

```python
"""An example of using the nested `if-else` statements."""

a: int = 13  # the first number
b: int = 7  # the second number
c: int = 9  # the third number

if a > b:       # This means that a > b
    if a > c:  # This means that a > b and a > c.
        print(f"{a} is the greatest number.")
    else:       # This means that a > b and c >= a.
        print(f"{c} is the greatest number.")
else:           # This means that b >= a.
    if b > c:  # This means that b >= a and b > c.
        print(f"{b} is the greatest number.")
    else:       # This means that b >= a and c >= b.
        print(f"{c} is the greatest number.")

# Some side information: The max and min function are very useful.
print(f"The maximum is {max(a, b, c)}.")  # max: get maximum of a
    ↪ sequence
print(f"The minimum is {min(a, b, c)}.")  # min: get minimum of a
    ↪ sequence
```

↓ python3 if_else_nested.py ↓

```
13 is the greatest number.
The maximum is 13.
The minimum is 7.
```

Beispiel

- Es reicht dann, `b` mit `c` zu vergleichen.
- Wir rücken also ein weiteres `if` ein, diesmal mit der Bedingung `b > c`.
- Wenn diese `True` ist, dann ist der größte Wert gleich dem in `b` gespeicherten Wert.
- Wir können also `f"{b} is the greatest number."` drucken.
- Wenn die Bedingung dagegen `False` war, dann kommen wir in den `else`-Zweig.

```
"""An example of using the nested `if-else` statements."""

a: int = 13  # the first number
b: int = 7  # the second number
c: int = 9  # the third number

if a > b:       # This means that a > b
    if a > c:  # This means that a > b and a > c.
        print(f"{a} is the greatest number.")
    else:       # This means that a > b and c >= a.
        print(f"{c} is the greatest number.")
else:           # This means that b >= a.
    if b > c:  # This means that b >= a and b > c.
        print(f"{b} is the greatest number.")
    else:       # This means that b >= a and c >= b.
        print(f"{c} is the greatest number.")

# Some side information: The max and min function are very useful.
print(f"The maximum is {max(a, b, c)}.")  # max: get maximum of a
    ↪ sequence
print(f"The minimum is {min(a, b, c)}.")  # min: get minimum of a
    ↪ sequence
```

↓ python3 if_else_nested.py ↓

```
13 is the greatest number.
The maximum is 13.
The minimum is 7.
```

Beispiel

- Wir rücken also ein weiteres `if` ein, diesmal mit der Bedingung `b > c`.
- Wenn diese `True` ist, dann ist der größte Wert gleich dem in `b` gespeicherten Wert.
- Wir können also `f"{b} is the greatest number."` drucken.
- Wenn die Bedingung dagegen `False` war, dann kommen wir in den `else`-Zweig.
- Wir wissen, dass `b >= a` und das entweder `c > b` oder `c >= b` gilt.

```python
"""An example of using the nested `if-else` statements."""

a: int = 13  # the first number
b: int = 7  # the second number
c: int = 9  # the third number

if a > b:       # This means that a > b
    if a > c:  # This means that a > b and a > c.
        print(f"{a} is the greatest number.")
    else:       # This means that a > b and c >= a.
        print(f"{c} is the greatest number.")
else:           # This means that b >= a.
    if b > c:  # This means that b >= a and b > c.
        print(f"{b} is the greatest number.")
    else:       # This means that b >= a and c >= b.
        print(f"{c} is the greatest number.")

# Some side information: The max and min function are very useful.
print(f"The maximum is {max(a, b, c)}.")  # max: get maximum of a
    ↪ sequence
print(f"The minimum is {min(a, b, c)}.")  # min: get minimum of a
    ↪ sequence
```

↓ python3 if_else_nested.py ↓

```
13 is the greatest number.
The maximum is 13.
The minimum is 7.
```

Beispiel

- Wenn diese `True` ist, dann ist der größte Wert gleich dem in `b` gespeicherten Wert.
- Wir können also `f"{b} is the greatest number."` drucken.
- Wenn die Bedingung dagegen `False` war, dann kommen wir in den `else`-Zweig.
- Wir wissen, dass `b >= a` und das entweder `c > b` oder `c >= b` gilt.
- Dafür können wir nun wieder `f"{c} is the greatest number."` drucken.

```python
"""An example of using the nested `if-else` statements."""

a: int = 13  # the first number
b: int = 7  # the second number
c: int = 9  # the third number

if a > b:       # This means that a > b
    if a > c:  # This means that a > b and a > c.
        print(f"{a} is the greatest number.")
    else:       # This means that a > b and c >= a.
        print(f"{c} is the greatest number.")
else:           # This means that b >= a.
    if b > c:  # This means that b >= a and b > c.
        print(f"{b} is the greatest number.")
    else:       # This means that b >= a and c >= b.
        print(f"{c} is the greatest number.")

# Some side information: The max and min function are very useful.
print(f"The maximum is {max(a, b, c)}.")  # max: get maximum of a
    ↪ sequence
print(f"The minimum is {min(a, b, c)}.")  # min: get minimum of a
    ↪ sequence
```

↓ python3 if_else_nested.py ↓

```
13 is the greatest number.
The maximum is 13.
The minimum is 7.
```

Beispiel

- Wenn die Bedingung dagegen `False` war, dann kommen wir in den `else`-Zweig.
- Wir wissen, dass `b >= a` und das entweder `c > b` oder `c >= b` gilt.
- Dafür können wir nun wieder `f"{c} is the greatest number."` drucken.
- Anmerkung: Python hat die beiden Funktionen `min` und `max`, die jeweils eine Sequenz von Werten als Parameter bekommen und den kleinsten bzw. größten Wert davon zurückliefern.

```python
"""An example of using the nested `if-else` statements."""

a: int = 13  # the first number
b: int = 7  # the second number
c: int = 9  # the third number

if a > b:       # This means that a > b
    if a > c:  # This means that a > b and a > c.
        print(f"{a} is the greatest number.")
    else:       # This means that a > b and c >= a.
        print(f"{c} is the greatest number.")
else:           # This means that b >= a.
    if b > c:  # This means that b >= a and b > c.
        print(f"{b} is the greatest number.")
    else:       # This means that b >= a and c >= b.
        print(f"{c} is the greatest number.")

# Some side information: The max and min function are very useful.
print(f"The maximum is {max(a, b, c)}.")  # max: get maximum of a
    ↪ sequence
print(f"The minimum is {min(a, b, c)}.")  # min: get minimum of a
    ↪ sequence
```

↓ python3 if_else_nested.py ↓

```
13 is the greatest number.
The maximum is 13.
The minimum is 7.
```

Beispiel

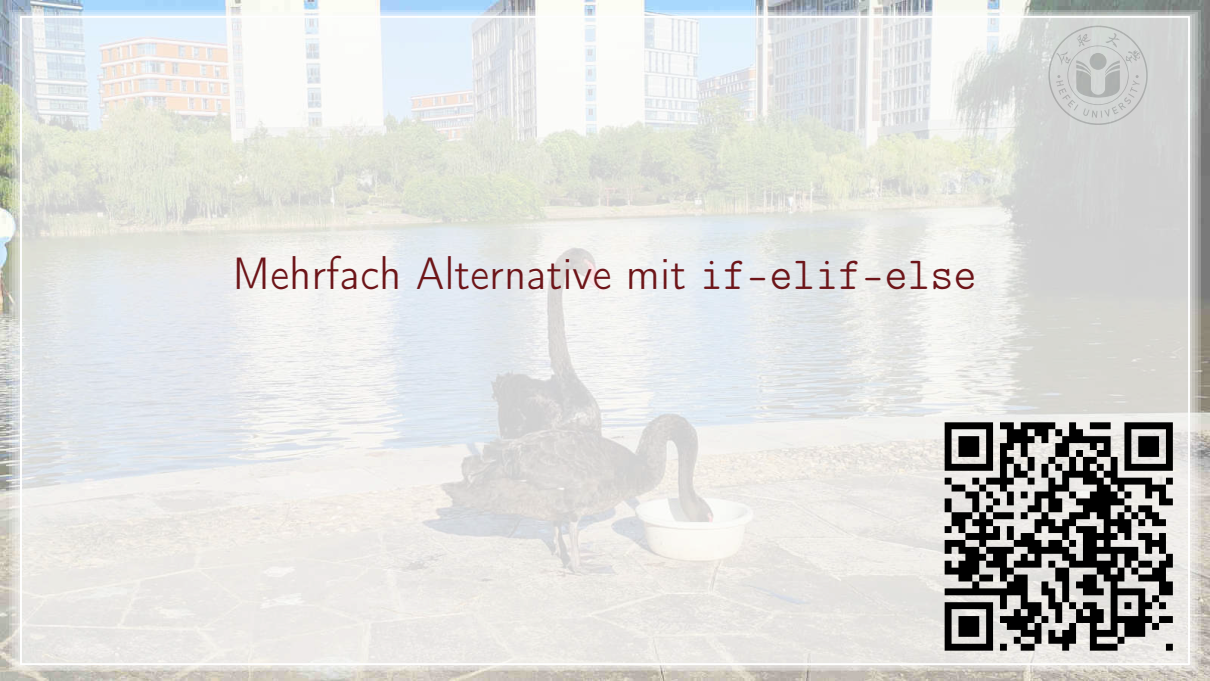
- Wir wissen, dass `b >= a` und das entweder `c > b` oder `c >= b` gilt.
- Dafür können wir nun wieder `f"{c} is the greatest number."` drucken.
- Anmerkung: Python hat die beiden Funktionen `min` und `max`, die jeweils eine Sequenz von Werten als Parameter bekommen und den kleinsten bzw. größten Wert davon zurückliefern.
- In den letzten beiden Zeilen unseres Programms wenden wir sie auf die Variablen `a`, `b`, und `c` an.

```python
"""An example of using the nested `if-else` statements."""

a: int = 13  # the first number
b: int = 7  # the second number
c: int = 9  # the third number

if a > b:       # This means that a > b
    if a > c:  # This means that a > b and a > c.
        print(f"{a} is the greatest number.")
    else:       # This means that a > b and c >= a.
        print(f"{c} is the greatest number.")
else:           # This means that b >= a.
    if b > c:  # This means that b >= a and b > c.
        print(f"{b} is the greatest number.")
    else:       # This means that b >= a and c >= b.
        print(f"{c} is the greatest number.")

# Some side information: The max and min function are very useful.
print(f"The maximum is {max(a, b, c)}.")  # max: get maximum of a
    ↪ sequence
print(f"The minimum is {min(a, b, c)}.")  # min: get minimum of a
    ↪ sequence
```

↓ `python3 if_else_nested.py` ↓

```
13 is the greatest number.
The maximum is 13.
The minimum is 7.
```

Mehrfach Alternative mit `if-elif-else`



Mehrfach Alternative mit `if-elif-else`

- In manchen Fällen müssen wir Alternativen so verschachteln, dass der `else`-Block des ersten `if`s ein weiteres `if` beinhaltet, und genau nur das `if`.

Mehrfach Alternative mit `if-elif-else`

- In manchen Fällen müssen wir Alternativen so verschachteln, dass der `else`-Block des ersten `if`s ein weiteres `if` beinhaltet, und genau nur das `if`.
  - Dessen `else`-Block beinhaltet dann vielleicht ein weiteres `if` beinhaltet, und genau nur dieses `if`.

Mehrfach Alternative mit `if-elif-else`

- In manchen Fällen müssen wir Alternativen so verschachteln, dass der `else`-Block des ersten `if`s ein weiteres `if` beinhaltet, und genau nur das `if`.
  - Dessen `else`-Block beinhaltet dann vielleicht ein weiteres `if` beinhaltet, und genau nur dieses `if`.
    - Dessen `else`-Block beinhaltet dann vielleicht ein weiteres `if` beinhaltet, und genau nur dieses `if`.

Mehrfach Alternative mit `if-elif-else`

- In manchen Fällen müssen wir Alternativen so verschachteln, dass der `else`-Block des ersten `if`s ein weiteres `if` beinhaltet, und genau nur das `if`.
  - Dessen `else`-Block beinhaltet dann vielleicht ein weiteres `if` beinhaltet, und genau nur dieses `if`.
    - Dessen `else`-Block beinhaltet dann vielleicht ein weiteres `if` beinhaltet, und genau nur dieses `if`. Und so weiter.

Mehrfach Alternative mit `if-elif-else`

- In manchen Fällen müssen wir Alternativen so verschachteln, dass der `else`-Block des ersten `if`s ein weiteres `if` beinhaltet, und genau nur das `if`.
  - Dessen `else`-Block beinhaltet dann vielleicht ein weiteres `if` beinhaltet, und genau nur dieses `if`.
    - Dessen `else`-Block beinhaltet dann vielleicht ein weiteres `if` beinhaltet, und genau nur dieses `if`. Und so weiter.
- Das sieht ziemlich häßlich aus, weil wir ja jede neue Alternative wieder mit vier Leerzeichen einrücken müssen.

Mehrfach Alternative mit `if-elif-else`

- In manchen Fällen müssen wir Alternativen so verschachteln, dass der `else`-Block des ersten `if`s ein weiteres `if` beinhaltet, und genau nur das `if`.
  - Dessen `else`-Block beinhaltet dann vielleicht ein weiteres `if` beinhaltet, und genau nur dieses `if`.
    - Dessen `else`-Block beinhaltet dann vielleicht ein weiteres `if` beinhaltet, und genau nur dieses `if`. Und so weiter.
- Das sieht ziemlich häßlich aus, weil wir ja jede neue Alternative wieder mit vier Leerzeichen einrücken müssen.
- Es füllt auch die Horizontale unseres Bildschirms schnell aus.

Mehrfach Alternative mit `if-elif-else`

- Darum wurde das `elif`-Statement entwickelt, das genau ein `else`, welches nur ein `if` beinhaltet, ersetzt (oder eher kombiniert)[17].

Mehrfach Alternative mit `if-elif-else`

- Darum wurde das `elif`-Statement entwickelt, das genau ein `else`, welches nur ein `if` beinhaltet, ersetzt (oder eher kombiniert)[17].

```
"""The syntax of an "if-elif-else"-statement in Python."""

if booleanExpression 1:
    conditional 1 statement 1  # Only executed if booleanExpression 1
    conditional 1 statement 2  # evaluates to True.
    #...
elif booleanExpression 2:  # such block can be placed arbitrarily often
    conditional 2 statement 1  # Only executed if booleanExpression 1
    conditional 2 statement 2  # evaluates to False and
    #...                        # booleanExpression 2 evaluates to True.
else:  # The else-paart is optional, can be left away.
    alternative statement 1  # Only executed if booleanExpression 1 and
    alternative statement 2  # 2 both evaluate to False.
    #...

normal statement 1  # Always executed, regardless of the result of
normal statement 2  # booleanExpression 1 and 2.
#...
```

Mehrfach Alternative mit `if-elif-else`

- Darum wurde das `elif`-Statement entwickelt, das genau ein `else`, welches nur ein `if` beinhaltet, ersetzt (oder eher kombiniert)[17].
- Anstelle dem `else` mit dem darin eingerückten `if` schreiben wir einfach `elif` mit der Bedingung, die das `if` gehabt hätte.

```
"""The syntax of an "if-elif-else"-statement in Python."""

if booleanExpression 1:
    conditional 1 statement 1  # Only executed if booleanExpression 1
    conditional 1 statement 2  # evaluates to True.
    #...
elif booleanExpression 2:  # such block can be placed arbitrarily often
    conditional 2 statement 1  # Only executed if booleanExpression 1
    conditional 2 statement 2  # evaluates to False and
    #...                        # booleanExpression 2 evaluates to True.
else:  # The else-paart is optional, can be left away.
    alternative statement 1  # Only executed if booleanExpression 1 and
    alternative statement 2  # 2 both evaluate to False.
    #...

normal statement 1  # Always executed, regardless of the result of
normal statement 2  # booleanExpression 1 and 2.
#...
```

Mehrfach Alternative mit `if-elif-else`

- Darum wurde das `elif`-Statement entwickelt, das genau ein `else`, welches nur ein `if` beinhaltet, ersetzt (oder eher kombiniert)[17].
- Anstelle dem `else` mit dem darin eingerückten `if` schreiben wir einfach `elif` mit der Bedingung, die das `if` gehabt hätte.
- Wir können optional am Ende noch einen `else`-Block haben, der nur dann ausgeführt wird, wenn *keine* der Bedingungen des `if` und der `elif`s wahr war.

```
"""The syntax of an "if-elif-else"-statement in Python."""

if booleanExpression 1:
    conditional 1 statement 1  # Only executed if booleanExpression 1
    conditional 1 statement 2  # evaluates to True.
    #...
elif booleanExpression 2:  # such block can be placed arbitrarily often
    conditional 2 statement 1  # Only executed if booleanExpression 1
    conditional 2 statement 2  # evaluates to False and
    #...                        # booleanExpression 2 evaluates to True.
else:  # The else-paart is optional, can be left away.
    alternative statement 1  # Only executed if booleanExpression 1 and
    alternative statement 2  # 2 both evaluate to False.
    #...

normal statement 1  # Always executed, regardless of the result of
normal statement 2  # booleanExpression 1 and 2.
#...
```

Beispiel (1)

- Wir benutzen diese Syntax in Programm `if_elif_example.py`, um die Lebensphase einer Person basierend auf ihrem Alter zu berechnen.

```python
"""An example of `elif` using human age groups."""

age: int = 42  # the age of the person
phase: str | None = None  # the life phase, to be computed

if age <= 3:     # If the age is no more than 3 years...
    phase = "infancy"  # then the person is in their infancy.
elif age <= 6:  # If (NOT age <= 3) and (age <= 6) ...
    phase = "early childhood"  # then they are in their early childhood.
elif age <= 8:  # If (NOT age <= 3) and (NOT age <= 6) and (age <= 8)
    phase = "middle childhood"
elif age <= 11:  # If ... (NOT age <= 8) and (age <= 11)
    phase = "late childhood"
elif age <= 20:  # If ... (NOT age <= 11) and (age <= 20)
    phase = "adolescence"
elif age <= 35:  # If ... (NOT age <= 20) and (age <= 35)
    phase = "early adulthood"
elif age <= 50:  # If ... (NOT age <= 35) and (age <= 50)
    phase = "midlife"
elif age < 80:   # If ... (NOT age <= 50) and (age < 80)
    phase = "mature adulthood"
else:  # otherwise, i.e., if age >= 8
    phase = "late adulthood"

print(f"A person of {age} years is in their {phase}.")
```

↓ `python3 if_elif_example.py` ↓

```
A person of 42 years is in their midlife.
```

Beispiel (1)

- Wir benutzen diese Syntax in Programm `if_elif_example.py`, um die Lebensphase einer Person basierend auf ihrem Alter zu berechnen.
- Wir definieren die Ganzzahl-Variable `age` für das Alter und setzen sie auf, sagen, wir 42.

```python
"""An example of `elif` using human age groups."""

age: int = 42  # the age of the person
phase: str | None = None  # the life phase, to be computed

if age <= 3:     # If the age is no more than 3 years...
    phase = "infancy"  # then the person is in their infancy.
elif age <= 6:  # If (NOT age <= 3) and (age <= 6) ...
    phase = "early childhood"  # then they are in their early childhood.
elif age <= 8:  # If (NOT age <= 3) and (NOT age <= 6) and (age <= 8)
    phase = "middle childhood"
elif age <= 11:  # If ... (NOT age <= 8) and (age <= 11)
    phase = "late childhood"
elif age <= 20:  # If ... (NOT age <= 11) and (age <= 20)
    phase = "adolescence"
elif age <= 35:  # If ... (NOT age <= 20) and (age <= 35)
    phase = "early adulthood"
elif age <= 50:  # If ... (NOT age <= 35) and (age <= 50)
    phase = "midlife"
elif age < 80:   # If ... (NOT age <= 50) and (age < 80)
    phase = "mature adulthood"
else:  # otherwise, i.e., if age >= 8
    phase = "late adulthood"

print(f"A person of {age} years is in their {phase}.")
```

↓ python3 if_elif_example.py ↓

```
A person of 42 years is in their midlife.
```

Beispiel (1)

- Wir benutzen diese Syntax in Programm `if_elif_example.py`, um die Lebensphase einer Person basierend auf ihrem Alter zu berechnen.
- Wir definieren die Ganzzahl-Variable `age` für das Alter und setzen sie auf, sagen, wir 42.
- Wir wollen einen String in die Variable `phase` schreiben, der die Lebensphase der Person beschreibt.

```python
"""An example of `elif` using human age groups."""

age: int = 42  # the age of the person
phase: str | None = None  # the life phase, to be computed

if age <= 3:     # If the age is no more than 3 years...
    phase = "infancy"  # then the person is in their infancy.
elif age <= 6:  # If (NOT age <= 3) and (age <= 6) ...
    phase = "early childhood"  # then they are in their early childhood.
elif age <= 8:  # If (NOT age <= 3) and (NOT age <= 6) and (age <= 8)
    phase = "middle childhood"
elif age <= 11:  # If ... (NOT age <= 8) and (age <= 11)
    phase = "late childhood"
elif age <= 20:  # If ... (NOT age <= 11) and (age <= 20)
    phase = "adolescence"
elif age <= 35:  # If ... (NOT age <= 20) and (age <= 35)
    phase = "early adulthood"
elif age <= 50:  # If ... (NOT age <= 35) and (age <= 50)
    phase = "midlife"
elif age < 80:   # If ... (NOT age <= 50) and (age < 80)
    phase = "mature adulthood"
else:  # otherwise, i.e., if age >= 8
    phase = "late adulthood"

print(f"A person of {age} years is in their {phase}.")
```

↓ python3 if_elif_example.py ↓

```
A person of 42 years is in their midlife.
```

Beispiel (1)

- Wir benutzen diese Syntax in Programm `if_elif_example.py`, um die Lebensphase einer Person basierend auf ihrem Alter zu berechnen.
- Wir definieren die Ganzzahl-Variable `age` für das Alter und setzen sie auf, sagen, wir 42.
- Wir wollen einen String in die Variable `phase` schreiben, der die Lebensphase der Person beschreibt.
- Am Anfang steht nichts in der Variable, also setzen wir `phase = None`.

```python
"""An example of `elif` using human age groups."""

age: int = 42  # the age of the person
phase: str | None = None  # the life phase, to be computed

if age <= 3:     # If the age is no more than 3 years...
    phase = "infancy"  # then the person is in their infancy.
elif age <= 6:  # If (NOT age <= 3) and (age <= 6) ...
    phase = "early childhood"  # then they are in their early childhood.
elif age <= 8:  # If (NOT age <= 3) and (NOT age <= 6) and (age <= 8)
    phase = "middle childhood"
elif age <= 11:  # If ... (NOT age <= 8) and (age <= 11)
    phase = "late childhood"
elif age <= 20:  # If ... (NOT age <= 11) and (age <= 20)
    phase = "adolescence"
elif age <= 35:  # If ... (NOT age <= 20) and (age <= 35)
    phase = "early adulthood"
elif age <= 50:  # If ... (NOT age <= 35) and (age <= 50)
    phase = "midlife"
elif age < 80:   # If ... (NOT age <= 50) and (age < 80)
    phase = "mature adulthood"
else:  # otherwise, i.e., if age >= 8
    phase = "late adulthood"

print(f"A person of {age} years is in their {phase}.")
```

↓ `python3 if_elif_example.py` ↓

```
A person of 42 years is in their midlife.
```

Beispiel (1)

- Wir benutzen diese Syntax, um die Lebensphase einer Person basierend auf ihrem Alter zu berechnen.
- Wir definieren die Ganzzahl-Variable `age` für das Alter und setzen sie auf, sagen, wir 42.
- Wir wollen einen String in die Variable `phase` schreiben, der die Lebensphase der Person beschreibt.
- Am Anfang steht nichts in der Variable, also setzen wir `phase = None`.
- Wir benutzen den Type Hint `str | None` um eine Variable zu spezifizieren, die entweder `None` oder eine String beinhalten kann[19].

```python
"""An example of `elif` using human age groups."""

age: int = 42  # the age of the person
phase: str | None = None  # the life phase, to be computed

if age <= 3:     # If the age is no more than 3 years...
    phase = "infancy"  # then the person is in their infancy.
elif age <= 6:  # If (NOT age <= 3) and (age <= 6) ...
    phase = "early childhood"  # then they are in their early childhood.
elif age <= 8:  # If (NOT age <= 3) and (NOT age <= 6) and (age <= 8)
    phase = "middle childhood"
elif age <= 11:  # If ... (NOT age <= 8) and (age <= 11)
    phase = "late childhood"
elif age <= 20:  # If ... (NOT age <= 11) and (age <= 20)
    phase = "adolescence"
elif age <= 35:  # If ... (NOT age <= 20) and (age <= 35)
    phase = "early adulthood"
elif age <= 50:  # If ... (NOT age <= 35) and (age <= 50)
    phase = "midlife"
elif age < 80:   # If ... (NOT age <= 50) and (age < 80)
    phase = "mature adulthood"
else:  # otherwise, i.e., if age >= 8
    phase = "late adulthood"

print(f"A person of {age} years is in their {phase}.")
```

↓ python3 if_elif_example.py ↓

```
A person of 42 years is in their midlife.
```

Beispiel (1)

- Wir definieren die Ganzzahl-Variable `age` für das Alter und setzen sie auf, sagen, wir 42.
- Wir wollen einen String in die Variable `phase` schreiben, der die Lebensphase der Person beschreibt.
- Am Anfang steht nichts in der Variable, also setzen wir `phase = None`.
- Wir benutzen den Type Hint `str | None` um eine Variable zu spezifizieren, die entweder `None` oder eine String beinhalten kann[19].
- Wir beginning mit der ersten Bedingung und schreiben `if age <= 3`.

```python
"""An example of `elif` using human age groups."""

age: int = 42  # the age of the person
phase: str | None = None  # the life phase, to be computed

if age <= 3:     # If the age is no more than 3 years...
    phase = "infancy"  # then the person is in their infancy.
elif age <= 6:  # If (NOT age <= 3) and (age <= 6) ...
    phase = "early childhood"  # then they are in their early childhood.
elif age <= 8:  # If (NOT age <= 3) and (NOT age <= 6) and (age <= 8)
    phase = "middle childhood"
elif age <= 11:  # If ... (NOT age <= 8) and (age <= 11)
    phase = "late childhood"
elif age <= 20:  # If ... (NOT age <= 11) and (age <= 20)
    phase = "adolescence"
elif age <= 35:  # If ... (NOT age <= 20) and (age <= 35)
    phase = "early adulthood"
elif age <= 50:  # If ... (NOT age <= 35) and (age <= 50)
    phase = "midlife"
elif age < 80:   # If ... (NOT age <= 50) and (age < 80)
    phase = "mature adulthood"
else:  # otherwise, i.e., if age >= 8
    phase = "late adulthood"

print(f"A person of {age} years is in their {phase}.")
```

↓ `python3 if_elif_example.py` ↓

```
A person of 42 years is in their midlife.
```

Beispiel (1)

- Am Anfang steht nichts in der Variable, also setzen wir `phase = None`.
- Wir benutzen den Type Hint `str | None` um eine Variable zu spezifizieren, die entweder `None` oder eine String beinhalten kann[19].
- Wir beginning mit der ersten Bedingung und schreiben `if age <= 3`.
- Wenn das `True` ergibt, dann setzen wir `phase = "infancy"`, was bedeutet, dass sich die Person in der frühesten Kindheitsphase befindet.

```python
"""An example of `elif` using human age groups."""

age: int = 42  # the age of the person
phase: str | None = None  # the life phase, to be computed

if age <= 3:     # If the age is no more than 3 years...
    phase = "infancy"  # then the person is in their infancy.
elif age <= 6:  # If (NOT age <= 3) and (age <= 6) ...
    phase = "early childhood"  # then they are in their early childhood.
elif age <= 8:  # If (NOT age <= 3) and (NOT age <= 6) and (age <= 8)
    phase = "middle childhood"
elif age <= 11:  # If ... (NOT age <= 8) and (age <= 11)
    phase = "late childhood"
elif age <= 20:  # If ... (NOT age <= 11) and (age <= 20)
    phase = "adolescence"
elif age <= 35:  # If ... (NOT age <= 20) and (age <= 35)
    phase = "early adulthood"
elif age <= 50:  # If ... (NOT age <= 35) and (age <= 50)
    phase = "midlife"
elif age < 80:   # If ... (NOT age <= 50) and (age < 80)
    phase = "mature adulthood"
else:  # otherwise, i.e., if age >= 8
    phase = "late adulthood"

print(f"A person of {age} years is in their {phase}.")
```

↓ python3 if_elif_example.py ↓

```
A person of 42 years is in their midlife.
```

Beispiel (1)

- Wir benutzen den Type Hint `str | None` um eine Variable zu spezifizieren, die entweder `None` oder eine String beinhalten kann[19].
- Wir beginning mit der ersten Bedingung und schreiben `if age <= 3`.
- Wenn das `True` ergibt, dann setzen wir `phase = "infancy"`, was bedeutet, dass sich die Person in der frühesten Kindheitsphase befindet.
- In diesem Fall ended auch der ganze `if...elif..else` block und keine weiteren Bedingungen werden geprüft.

```python
"""An example of `elif` using human age groups."""

age: int = 42  # the age of the person
phase: str | None = None  # the life phase, to be computed

if age <= 3:     # If the age is no more than 3 years...
    phase = "infancy"  # then the person is in their infancy.
elif age <= 6:  # If (NOT age <= 3) and (age <= 6) ...
    phase = "early childhood"  # then they are in their early childhood.
elif age <= 8:  # If (NOT age <= 3) and (NOT age <= 6) and (age <= 8)
    phase = "middle childhood"
elif age <= 11:  # If ... (NOT age <= 8) and (age <= 11)
    phase = "late childhood"
elif age <= 20:  # If ... (NOT age <= 11) and (age <= 20)
    phase = "adolescence"
elif age <= 35:  # If ... (NOT age <= 20) and (age <= 35)
    phase = "early adulthood"
elif age <= 50:  # If ... (NOT age <= 35) and (age <= 50)
    phase = "midlife"
elif age < 80:   # If ... (NOT age <= 50) and (age < 80)
    phase = "mature adulthood"
else:  # otherwise, i.e., if age >= 8
    phase = "late adulthood"

print(f"A person of {age} years is in their {phase}.")
```

↓ python3 if_elif_example.py ↓

```
A person of 42 years is in their midlife.
```

Beispiel (1)

- Wir beginning mit der ersten Bedingung und schreiben `if age <= 3`.
- Wenn das `True` ergibt, dann setzen wir `phase = "infancy"`, was bedeutet, dass sich die Person in der frühesten Kindheitsphase befindet.
- In diesem Fall ended auch der ganze `if...elif..else` block und keine weiteren Bedingungen werden geprüft.
- Andernfalls, also wenn `age <= 3` `False` ergibt, also wenn `age > 3` gilt, wird die nächste Bedingung geprüft.

```python
"""An example of `elif` using human age groups."""

age: int = 42  # the age of the person
phase: str | None = None  # the life phase, to be computed

if age <= 3:     # If the age is no more than 3 years...
    phase = "infancy"  # then the person is in their infancy.
elif age <= 6:  # If (NOT age <= 3) and (age <= 6) ...
    phase = "early childhood"  # then they are in their early childhood.
elif age <= 8:  # If (NOT age <= 3) and (NOT age <= 6) and (age <= 8)
    phase = "middle childhood"
elif age <= 11:  # If ... (NOT age <= 8) and (age <= 11)
    phase = "late childhood"
elif age <= 20:  # If ... (NOT age <= 11) and (age <= 20)
    phase = "adolescence"
elif age <= 35:  # If ... (NOT age <= 20) and (age <= 35)
    phase = "early adulthood"
elif age <= 50:  # If ... (NOT age <= 35) and (age <= 50)
    phase = "midlife"
elif age < 80:   # If ... (NOT age <= 50) and (age < 80)
    phase = "mature adulthood"
else:  # otherwise, i.e., if age >= 8
    phase = "late adulthood"

print(f"A person of {age} years is in their {phase}.")
```

↓ `python3 if_elif_example.py` ↓

```
A person of 42 years is in their midlife.
```

Beispiel (1)

- Wenn das `True` ergibt, dann setzen wir `phase = "infancy"`, was bedeutet, dass sich die Person in der frühesten Kindheitsphase befindet.
- In diesem Fall ended auch der ganze `if...elif..else` block und keine weiteren Bedingungen werden geprüft.
- Andernfalls, also wenn `age <= 3` `False` ergibt, also wenn `age > 3` gilt, wird die nächste Bedingung geprüft.
- Anstelle ein `else` mit einem weiteren eingerückten `if` können wir einfach `elif age <= 6` ohne weitere Einrückung schreiben.

```python
"""An example of `elif` using human age groups."""

age: int = 42  # the age of the person
phase: str | None = None  # the life phase, to be computed

if age <= 3:     # If the age is no more than 3 years...
    phase = "infancy"  # then the person is in their infancy.
elif age <= 6:  # If (NOT age <= 3) and (age <= 6) ...
    phase = "early childhood"  # then they are in their early childhood.
elif age <= 8:  # If (NOT age <= 3) and (NOT age <= 6) and (age <= 8)
    phase = "middle childhood"
elif age <= 11:  # If ... (NOT age <= 8) and (age <= 11)
    phase = "late childhood"
elif age <= 20:  # If ... (NOT age <= 11) and (age <= 20)
    phase = "adolescence"
elif age <= 35:  # If ... (NOT age <= 20) and (age <= 35)
    phase = "early adulthood"
elif age <= 50:  # If ... (NOT age <= 35) and (age <= 50)
    phase = "midlife"
elif age < 80:   # If ... (NOT age <= 50) and (age < 80)
    phase = "mature adulthood"
else:  # otherwise, i.e., if age >= 8
    phase = "late adulthood"

print(f"A person of {age} years is in their {phase}.")
```

↓ python3 if_elif_example.py ↓

```
A person of 42 years is in their midlife.
```

Beispiel (1)

- In diesem Fall ended auch der ganze `if...elif..else` block und keine weiteren Bedingungen werden geprüft.
- Andernfalls, also wenn `age <= 3` `False` ergibt, also wenn `age > 3` gilt, wird die nächste Bedingung geprüft.
- Anstelle ein `else` mit einem weiteren eingerückten `if` können wir einfach `elif age <= 6` ohne weitere Einrückung schreiben.
- Das `elif` funktioniert wie ein `else` gefolgt von einem eingrückten `if`.

```python
"""An example of `elif` using human age groups."""

age: int = 42  # the age of the person
phase: str | None = None  # the life phase, to be computed

if age <= 3:     # If the age is no more than 3 years...
    phase = "infancy"  # then the person is in their infancy.
elif age <= 6:  # If (NOT age <= 3) and (age <= 6) ...
    phase = "early childhood"  # then they are in their early childhood.
elif age <= 8:  # If (NOT age <= 3) and (NOT age <= 6) and (age <= 8)
    phase = "middle childhood"
elif age <= 11:  # If ... (NOT age <= 8) and (age <= 11)
    phase = "late childhood"
elif age <= 20:  # If ... (NOT age <= 11) and (age <= 20)
    phase = "adolescence"
elif age <= 35:  # If ... (NOT age <= 20) and (age <= 35)
    phase = "early adulthood"
elif age <= 50:  # If ... (NOT age <= 35) and (age <= 50)
    phase = "midlife"
elif age < 80:   # If ... (NOT age <= 50) and (age < 80)
    phase = "mature adulthood"
else:  # otherwise, i.e., if age >= 8
    phase = "late adulthood"

print(f"A person of {age} years is in their {phase}.")
```

↓ python3 if_elif_example.py ↓

```
A person of 42 years is in their midlife.
```

Beispiel (1)

- In diesem Fall ended auch der ganze `if...elif..else` block und keine weiteren Bedingungen werden geprüft.
- Andernfalls, also wenn `age <= 3` `False` ergibt, also wenn `age > 3` gilt, wird die nächste Bedingung geprüft.
- Anstelle ein `else` mit einem weiteren eingerückten `if` können wir einfach `elif age <= 6` ohne weitere Einrückung schreiben.
- Das `elif` funktioniert wie ein `else` gefolgt von einem eingrückten `if`.
- In unserem Fall prüft das `elif` ob `age <= 6` gilt (nachdem wir an dieser Stelle schon wissen, dass `age > 3`).

```python
"""An example of `elif` using human age groups."""

age: int = 42  # the age of the person
phase: str | None = None  # the life phase, to be computed

if age <= 3:     # If the age is no more than 3 years...
    phase = "infancy"  # then the person is in their infancy.
elif age <= 6:  # If (NOT age <= 3) and (age <= 6) ...
    phase = "early childhood"  # then they are in their early childhood.
elif age <= 8:  # If (NOT age <= 3) and (NOT age <= 6) and (age <= 8)
    phase = "middle childhood"
elif age <= 11:  # If ... (NOT age <= 8) and (age <= 11)
    phase = "late childhood"
elif age <= 20:  # If ... (NOT age <= 11) and (age <= 20)
    phase = "adolescence"
elif age <= 35:  # If ... (NOT age <= 20) and (age <= 35)
    phase = "early adulthood"
elif age <= 50:  # If ... (NOT age <= 35) and (age <= 50)
    phase = "midlife"
elif age < 80:   # If ... (NOT age <= 50) and (age < 80)
    phase = "mature adulthood"
else:  # otherwise, i.e., if age >= 8
    phase = "late adulthood"

print(f"A person of {age} years is in their {phase}.")
```

↓ python3 if_elif_example.py ↓

```
A person of 42 years is in their midlife.
```

Beispiel (1)

- Andernfalls, also wenn `age <= 3` `False` ergibt, also wenn `age > 3` gilt, wird die nächste Bedingung geprüft.
- Anstelle ein `else` mit einem weiteren eingerückten `if` können wir einfach `elif age <= 6` ohne weitere Einrückung schreiben.
- Das `elif` funktioniert wie ein `else` gefolgt von einem eingrückten `if`.
- In unserem Fall prüft das `elif` ob `age <= 6` gilt (nachdem wir an dieser Stelle schon wissen, dass `age > 3`).
- Wenn ja, dann nennen wir diese Lebensphase `"early childhood"`.

```python
"""An example of `elif` using human age groups."""

age: int = 42  # the age of the person
phase: str | None = None  # the life phase, to be computed

if age <= 3:     # If the age is no more than 3 years...
    phase = "infancy"  # then the person is in their infancy.
elif age <= 6:  # If (NOT age <= 3) and (age <= 6) ...
    phase = "early childhood"  # then they are in their early childhood.
elif age <= 8:  # If (NOT age <= 3) and (NOT age <= 6) and (age <= 8)
    phase = "middle childhood"
elif age <= 11:  # If ... (NOT age <= 8) and (age <= 11)
    phase = "late childhood"
elif age <= 20:  # If ... (NOT age <= 11) and (age <= 20)
    phase = "adolescence"
elif age <= 35:  # If ... (NOT age <= 20) and (age <= 35)
    phase = "early adulthood"
elif age <= 50:  # If ... (NOT age <= 35) and (age <= 50)
    phase = "midlife"
elif age < 80:   # If ... (NOT age <= 50) and (age < 80)
    phase = "mature adulthood"
else:  # otherwise, i.e., if age >= 8
    phase = "late adulthood"

print(f"A person of {age} years is in their {phase}.")
```

↓ python3 if_elif_example.py ↓

```
A person of 42 years is in their midlife.
```

Beispiel (1)

- Anstelle ein `else` mit einem weiteren eingerückten `if` können wir einfach `elif age <= 6` ohne weitere Einrückung schreiben.
- Das `elif` funktioniert wie ein `else` gefolgt von einem eingrückten `if`.
- In unserem Fall prüft das `elif` ob `age <= 6` gilt (nachdem wir an dieser Stelle schon wissen, dass `age > 3`).
- Wenn ja, dann nennen wir diese Lebensphase `"early childhood"`.
- Wenn das zutrifft, dann ended der ganze `if...elif...else`-Block und keine weiteren Bedigungen werden geprüft.

```python
"""An example of `elif` using human age groups."""

age: int = 42  # the age of the person
phase: str | None = None  # the life phase, to be computed

if age <= 3:     # If the age is no more than 3 years...
    phase = "infancy"  # then the person is in their infancy.
elif age <= 6:  # If (NOT age <= 3) and (age <= 6) ...
    phase = "early childhood"  # then they are in their early childhood.
elif age <= 8:  # If (NOT age <= 3) and (NOT age <= 6) and (age <= 8)
    phase = "middle childhood"
elif age <= 11:  # If ... (NOT age <= 8) and (age <= 11)
    phase = "late childhood"
elif age <= 20:  # If ... (NOT age <= 11) and (age <= 20)
    phase = "adolescence"
elif age <= 35:  # If ... (NOT age <= 20) and (age <= 35)
    phase = "early adulthood"
elif age <= 50:  # If ... (NOT age <= 35) and (age <= 50)
    phase = "midlife"
elif age < 80:   # If ... (NOT age <= 50) and (age < 80)
    phase = "mature adulthood"
else:  # otherwise, i.e., if age >= 8
    phase = "late adulthood"

print(f"A person of {age} years is in their {phase}.")
```

↓ python3 if_elif_example.py ↓

```
A person of 42 years is in their midlife.
```

Beispiel (1)

- Das `elif` funktioniert wie ein `else` gefolgt von einem eingrückten `if`.
- In unserem Fall prüft das `elif` ob `age <= 6` gilt (nachdem wir an dieser Stelle schon wissen, dass `age > 3`).
- Wenn ja, dann nennen wir diese Lebensphase `"early childhood"`.
- Wenn das zutrifft, dann ended der ganze `if...elif...else`-Block und keine weiteren Bedigungen werden geprüft.
- Wenn nicht, also wenn `age <= 6` `False` ergibt, dann geht es mit dem nächsten `elif` weiter.

```python
"""An example of `elif` using human age groups."""

age: int = 42  # the age of the person
phase: str | None = None  # the life phase, to be computed

if age <= 3:     # If the age is no more than 3 years...
    phase = "infancy"  # then the person is in their infancy.
elif age <= 6:  # If (NOT age <= 3) and (age <= 6) ...
    phase = "early childhood"  # then they are in their early childhood.
elif age <= 8:  # If (NOT age <= 3) and (NOT age <= 6) and (age <= 8)
    phase = "middle childhood"
elif age <= 11:  # If ... (NOT age <= 8) and (age <= 11)
    phase = "late childhood"
elif age <= 20:  # If ... (NOT age <= 11) and (age <= 20)
    phase = "adolescence"
elif age <= 35:  # If ... (NOT age <= 20) and (age <= 35)
    phase = "early adulthood"
elif age <= 50:  # If ... (NOT age <= 35) and (age <= 50)
    phase = "midlife"
elif age < 80:   # If ... (NOT age <= 50) and (age < 80)
    phase = "mature adulthood"
else:  # otherwise, i.e., if age >= 8
    phase = "late adulthood"

print(f"A person of {age} years is in their {phase}.")
```

↓ python3 if_elif_example.py ↓

```
A person of 42 years is in their midlife.
```

Beispiel (1)

- In unserem Fall prüft das `elif` ob `age <= 6` gilt (nachdem wir an dieser Stelle schon wissen, dass `age > 3`).
- Wenn ja, dann nennen wir diese Lebensphase `"early childhood"`.
- Wenn das zutrifft, dann ended der ganze `if...elif...else`-Block und keine weiteren Bedigungen werden geprüft.
- Wenn nicht, also wenn `age <= 6` `False` ergibt, dann geht es mit dem nächsten `elif` weiter.
- Nur dann wird also `elif age <= 8` ausgefürt.

```python
"""An example of `elif` using human age groups."""

age: int = 42  # the age of the person
phase: str | None = None  # the life phase, to be computed

if age <= 3:     # If the age is no more than 3 years...
    phase = "infancy"  # then the person is in their infancy.
elif age <= 6:  # If (NOT age <= 3) and (age <= 6) ...
    phase = "early childhood"  # then they are in their early childhood.
elif age <= 8:  # If (NOT age <= 3) and (NOT age <= 6) and (age <= 8)
    phase = "middle childhood"
elif age <= 11:  # If ... (NOT age <= 8) and (age <= 11)
    phase = "late childhood"
elif age <= 20:  # If ... (NOT age <= 11) and (age <= 20)
    phase = "adolescence"
elif age <= 35:  # If ... (NOT age <= 20) and (age <= 35)
    phase = "early adulthood"
elif age <= 50:  # If ... (NOT age <= 35) and (age <= 50)
    phase = "midlife"
elif age < 80:   # If ... (NOT age <= 50) and (age < 80)
    phase = "mature adulthood"
else:  # otherwise, i.e., if age >= 8
    phase = "late adulthood"

print(f"A person of {age} years is in their {phase}.")
```

↓ python3 if_elif_example.py ↓

```
A person of 42 years is in their midlife.
```

Beispiel (1)

- Wenn ja, dann nennen wir diese Lebensphase `"early childhood"`.
- Wenn das zutrifft, dann ended der ganze `if...elif...else`-Block und keine weiteren Bedigungen werden geprüft.
- Wenn nicht, also wenn `age <= 6` `False` ergibt, dann geht es mit dem nächsten `elif` weiter.
- Nur dann wird also `elif age <= 8` ausgefürt.
- Wenn diese Bedingung wahr ist, dann befindet sich die Person in der `"middle childhood"` und sind mit dem `if...elif...else` fertig.

```python
"""An example of `elif` using human age groups."""

age: int = 42  # the age of the person
phase: str | None = None  # the life phase, to be computed

if age <= 3:     # If the age is no more than 3 years...
    phase = "infancy"  # then the person is in their infancy.
elif age <= 6:  # If (NOT age <= 3) and (age <= 6) ...
    phase = "early childhood"  # then they are in their early childhood.
elif age <= 8:  # If (NOT age <= 3) and (NOT age <= 6) and (age <= 8)
    phase = "middle childhood"
elif age <= 11:  # If ... (NOT age <= 8) and (age <= 11)
    phase = "late childhood"
elif age <= 20:  # If ... (NOT age <= 11) and (age <= 20)
    phase = "adolescence"
elif age <= 35:  # If ... (NOT age <= 20) and (age <= 35)
    phase = "early adulthood"
elif age <= 50:  # If ... (NOT age <= 35) and (age <= 50)
    phase = "midlife"
elif age < 80:   # If ... (NOT age <= 50) and (age < 80)
    phase = "mature adulthood"
else:  # otherwise, i.e., if age >= 8
    phase = "late adulthood"

print(f"A person of {age} years is in their {phase}.")
```

↓ python3 if_elif_example.py ↓

```
A person of 42 years is in their midlife.
```

Beispiel (1)

- Wenn das zutrifft, dann ended der ganze `if...elif...else`-Block und keine weiteren Bedigungen werden geprüft.
- Wenn nicht, also wenn `age <= 6` `False` ergibt, dann geht es mit dem nächsten `elif` weiter.
- Nur dann wird also `elif age <= 8` ausgefürt.
- Wenn diese Bedingung wahr ist, dann befindet sich die Person in der `"middle childhood"` und sind mit dem `if...elif...else` fertig.
- Wenn `age <= 8` nicht gilt, dann geht es wieder mit dem nächsten `elif` weiter.

```python
"""An example of `elif` using human age groups."""

age: int = 42  # the age of the person
phase: str | None = None  # the life phase, to be computed

if age <= 3:     # If the age is no more than 3 years...
    phase = "infancy"  # then the person is in their infancy.
elif age <= 6:  # If (NOT age <= 3) and (age <= 6) ...
    phase = "early childhood"  # then they are in their early childhood.
elif age <= 8:  # If (NOT age <= 3) and (NOT age <= 6) and (age <= 8)
    phase = "middle childhood"
elif age <= 11:  # If ... (NOT age <= 8) and (age <= 11)
    phase = "late childhood"
elif age <= 20:  # If ... (NOT age <= 11) and (age <= 20)
    phase = "adolescence"
elif age <= 35:  # If ... (NOT age <= 20) and (age <= 35)
    phase = "early adulthood"
elif age <= 50:  # If ... (NOT age <= 35) and (age <= 50)
    phase = "midlife"
elif age < 80:   # If ... (NOT age <= 50) and (age < 80)
    phase = "mature adulthood"
else:  # otherwise, i.e., if age >= 8
    phase = "late adulthood"

print(f"A person of {age} years is in their {phase}.")
```

↓ python3 if_elif_example.py ↓

```
A person of 42 years is in their midlife.
```

Beispiel (1)

- Wenn nicht, also wenn `age <= 6` `False` ergibt, dann geht es mit dem nächsten `elif` weiter.
- Nur dann wird also `elif age <= 8` ausgefürt.
- Wenn diese Bedingung wahr ist, dann befindet sich die Person in der `"middle childhood"` und sind mit dem `if...elif...else` fertig.
- Wenn `age <= 8` nicht gilt, dann geht es wieder mit dem nächsten `elif` weiter.
- Und so weiter.

```python
"""An example of `elif` using human age groups."""

age: int = 42  # the age of the person
phase: str | None = None  # the life phase, to be computed

if age <= 3:     # If the age is no more than 3 years...
    phase = "infancy"  # then the person is in their infancy.
elif age <= 6:  # If (NOT age <= 3) and (age <= 6) ...
    phase = "early childhood"  # then they are in their early childhood.
elif age <= 8:  # If (NOT age <= 3) and (NOT age <= 6) and (age <= 8)
    phase = "middle childhood"
elif age <= 11:  # If ... (NOT age <= 8) and (age <= 11)
    phase = "late childhood"
elif age <= 20:  # If ... (NOT age <= 11) and (age <= 20)
    phase = "adolescence"
elif age <= 35:  # If ... (NOT age <= 20) and (age <= 35)
    phase = "early adulthood"
elif age <= 50:  # If ... (NOT age <= 35) and (age <= 50)
    phase = "midlife"
elif age < 80:   # If ... (NOT age <= 50) and (age < 80)
    phase = "mature adulthood"
else:  # otherwise, i.e., if age >= 8
    phase = "late adulthood"

print(f"A person of {age} years is in their {phase}.")
```

↓ `python3 if_elif_example.py` ↓

```
A person of 42 years is in their midlife.
```

Beispiel (1)

- Nur dann wird also `elif age <= 8` ausgefürt.
- Wenn diese Bedingung wahr ist, dann befindet sich die Person in der `"middle childhood"` und sind mit dem `if...elif...else` fertig.
- Wenn `age <= 8` nicht gilt, dann geht es wieder mit dem nächsten `elif` weiter.
- Und so weiter.
- Wenn selbst `age < 80`, die Bedingung des letzten `elif`, auch `False` ist, dann wird der `else`-Block ausgeführt.

```python
"""An example of `elif` using human age groups."""

age: int = 42  # the age of the person
phase: str | None = None  # the life phase, to be computed

if age <= 3:     # If the age is no more than 3 years...
    phase = "infancy"  # then the person is in their infancy.
elif age <= 6:  # If (NOT age <= 3) and (age <= 6) ...
    phase = "early childhood"  # then they are in their early childhood.
elif age <= 8:  # If (NOT age <= 3) and (NOT age <= 6) and (age <= 8)
    phase = "middle childhood"
elif age <= 11:  # If ... (NOT age <= 8) and (age <= 11)
    phase = "late childhood"
elif age <= 20:  # If ... (NOT age <= 11) and (age <= 20)
    phase = "adolescence"
elif age <= 35:  # If ... (NOT age <= 20) and (age <= 35)
    phase = "early adulthood"
elif age <= 50:  # If ... (NOT age <= 35) and (age <= 50)
    phase = "midlife"
elif age < 80:   # If ... (NOT age <= 50) and (age < 80)
    phase = "mature adulthood"
else:  # otherwise, i.e., if age >= 8
    phase = "late adulthood"

print(f"A person of {age} years is in their {phase}.")
```

↓ `python3 if_elif_example.py` ↓

```
A person of 42 years is in their midlife.
```

Beispiel (1)

- Wenn diese Bedingung wahr ist, dann befindet sich die Person in der `"middle childhood"` und sind mit dem `if...elif...else` fertig.
- Wenn `age <= 8` nicht gilt, dann geht es wieder mit dem nächsten `elif` weiter.
- Und so weiter.
- Wenn selbst `age < 80`, die Bedingung des letzten `elif`, auch `False` ist, dann wird der `else`-Block ausgeführt.
- Dann setzen wir `phase = "late adulthood"`.

```python
"""An example of `elif` using human age groups."""

age: int = 42  # the age of the person
phase: str | None = None  # the life phase, to be computed

if age <= 3:     # If the age is no more than 3 years...
    phase = "infancy"  # then the person is in their infancy.
elif age <= 6:  # If (NOT age <= 3) and (age <= 6) ...
    phase = "early childhood"  # then they are in their early childhood.
elif age <= 8:  # If (NOT age <= 3) and (NOT age <= 6) and (age <= 8)
    phase = "middle childhood"
elif age <= 11:  # If ... (NOT age <= 8) and (age <= 11)
    phase = "late childhood"
elif age <= 20:  # If ... (NOT age <= 11) and (age <= 20)
    phase = "adolescence"
elif age <= 35:  # If ... (NOT age <= 20) and (age <= 35)
    phase = "early adulthood"
elif age <= 50:  # If ... (NOT age <= 35) and (age <= 50)
    phase = "midlife"
elif age < 80:   # If ... (NOT age <= 50) and (age < 80)
    phase = "mature adulthood"
else:  # otherwise, i.e., if age >= 8
    phase = "late adulthood"

print(f"A person of {age} years is in their {phase}.")
```

↓ python3 if_elif_example.py ↓

```
A person of 42 years is in their midlife.
```

Beispiel (1)

- Wenn `age <= 8` nicht gilt, dann geht es wieder mit dem nächsten `elif` weiter.
- Und so weiter.
- Wenn selbst `age < 80`, die Bedingung des letzten `elif`, auch `False` ist, dann wird der `else`-Block ausgeführt.
- Dann setzen wir `phase = "late adulthood"`.
- Am Ende drucken wir die Lebensphase mit Hilfe von mit einem f-String aus.

```python
"""An example of `elif` using human age groups."""

age: int = 42  # the age of the person
phase: str | None = None  # the life phase, to be computed

if age <= 3:     # If the age is no more than 3 years...
    phase = "infancy"  # then the person is in their infancy.
elif age <= 6:  # If (NOT age <= 3) and (age <= 6) ...
    phase = "early childhood"  # then they are in their early childhood.
elif age <= 8:  # If (NOT age <= 3) and (NOT age <= 6) and (age <= 8)
    phase = "middle childhood"
elif age <= 11:  # If ... (NOT age <= 8) and (age <= 11)
    phase = "late childhood"
elif age <= 20:  # If ... (NOT age <= 11) and (age <= 20)
    phase = "adolescence"
elif age <= 35:  # If ... (NOT age <= 20) and (age <= 35)
    phase = "early adulthood"
elif age <= 50:  # If ... (NOT age <= 35) and (age <= 50)
    phase = "midlife"
elif age < 80:   # If ... (NOT age <= 50) and (age < 80)
    phase = "mature adulthood"
else:  # otherwise, i.e., if age >= 8
    phase = "late adulthood"

print(f"A person of {age} years is in their {phase}.")
```

↓ python3 if_elif_example.py ↓

```
A person of 42 years is in their midlife.
```

Beispiel (2)

- Schauen wir uns nochmal unser Programm `if_else_nested.py` an.

```python
"""An example of using the nested `if-else` statements."""

a: int = 13  # the first number
b: int = 7  # the second number
c: int = 9  # the third number

if a > b:       # This means that a > b
    if a > c:  # This means that a > b and a > c.
        print(f"{a} is the greatest number.")
    else:       # This means that a > b and c >= a.
        print(f"{c} is the greatest number.")
else:           # This means that b >= a.
    if b > c:  # This means that b >= a and b > c.
        print(f"{b} is the greatest number.")
    else:       # This means that b >= a and c >= b.
        print(f"{c} is the greatest number.")

# Some side information: The max and min function are very useful.
print(f"The maximum is {max(a, b, c)}.")  # max: get maximum of a
    ↪ sequence
print(f"The minimum is {min(a, b, c)}.")  # min: get minimum of a
    ↪ sequence
```

↓ `python3 if_else_nested.py` ↓

```
13 is the greatest number.
The maximum is 13.
The minimum is 7.
```

Beispiel (2)

- Schauen wir uns nochmal unser Programm `if_else_nested.py` an.
- Sieht eigentlich gut aus.

```python
"""An example of using the nested `if-else` statements."""

a: int = 13  # the first number
b: int = 7  # the second number
c: int = 9  # the third number

if a > b:       # This means that a > b
    if a > c:  # This means that a > b and a > c.
        print(f"{a} is the greatest number.")
    else:       # This means that a > b and c >= a.
        print(f"{c} is the greatest number.")
else:           # This means that b >= a.
    if b > c:  # This means that b >= a and b > c.
        print(f"{b} is the greatest number.")
    else:       # This means that b >= a and c >= b.
        print(f"{c} is the greatest number.")

# Some side information: The max and min function are very useful.
print(f"The maximum is {max(a, b, c)}.")  # max: get maximum of a
    ↪ sequence
print(f"The minimum is {min(a, b, c)}.")  # min: get minimum of a
    ↪ sequence
```

↓ `python3 if_else_nested.py` ↓

```
13 is the greatest number.
The maximum is 13.
The minimum is 7.
```

Beispiel (2)

- Schauen wir uns nochmal unser Programm `if_else_nested.py` an.
- Sieht eigentlich gut aus.
- Was sagt eigentlich Ruff dazu?

```python
"""An example of using the nested `if-else` statements."""

a: int = 13  # the first number
b: int = 7  # the second number
c: int = 9  # the third number

if a > b:       # This means that a > b
    if a > c:  # This means that a > b and a > c.
        print(f"{a} is the greatest number.")
    else:       # This means that a > b and c >= a.
        print(f"{c} is the greatest number.")
else:           # This means that b >= a.
    if b > c:  # This means that b >= a and b > c.
        print(f"{b} is the greatest number.")
    else:       # This means that b >= a and c >= b.
        print(f"{c} is the greatest number.")

# Some side information: The max and min function are very useful.
print(f"The maximum is {max(a, b, c)}.")  # max: get maximum of a
    ↪ sequence
print(f"The minimum is {min(a, b, c)}.")  # min: get minimum of a
    ↪ sequence
```

↓ `python3 if_else_nested.py` ↓

```
13 is the greatest number.
The maximum is 13.
The minimum is 7.
```

Beispiel (2)

- Schauen wir uns nochmal unser Programm `if_else_nested.py` an.
- Sieht eigentlich gut aus.
- Was sagt eigentlich Ruff dazu?

```
$ ruff check --target-version py312 --select=A,AIR,ANN,ASYNC,B,BLE,C,C4,
    ↪ COM,D,DJ,DTZ,E,ERA,EXE,F,FA,FIX,FLY,FURB,G,I,ICN,INP,ISC,INT,LOG,N
    ↪ ,NPY,PERF,PIE,PLC,PLE,PLR,PLW,PT,PYI,Q,RET,RSE,RUF,S,SIM,T,T10,TD,
    ↪ TID,TRY,UP,W,YTT --ignore=A005,ANN001,ANN002,ANN003,ANN204,ANN401,
    ↪ B008,B009,B010,C901,D203,D208,D212,D401,D407,D413,INP001,N801,
    ↪ PLC2801,PLR0904,PLR0911,PLR0912,PLR0913,PLR0914,PLR0915,PLR0916,
    ↪ PLR0917,PLR1702,PLR2004,PLR6301,PT011,PT012,PT013,PYI041,RUF100,S,
    ↪ T201,TRY003,UP035,W --line-length 79 if_else_nested.py
PLR5501 [*] Use `elif` instead of `else` then `if`, to reduce
    ↪ indentation
  --> if_else_nested.py:12:1
   |
10 |       else:      # This means that a > b and c >= a.
11 |           print(f"{c} is the greatest number.")
12 | / else:          # This means that b >= a.
13 | |     if b > c:  # This means that b >= a and b > c.
   | |____^
14 |           print(f"{b} is the greatest number.")
15 |       else:      # This means that b >= a and c >= b.
   |
help: Convert to `elif`

Found 1 error.
[*] 1 fixable with the `--fix` option.
# ruff 0.12.12 failed with exit code 1.
```

Beispiel (2)

- Schauen wir uns nochmal unser Programm `if_else_nested.py` an.
- Sieht eigentlich gut aus.
- Was sagt eigentlich Ruff dazu?
- Hm. Da hat Ruff eigentlich recht.

```
$ ruff check --target-version py312 --select=A,AIR,ANN,ASYNC,B,BLE,C,C4,
    ↪ COM,D,DJ,DTZ,E,ERA,EXE,F,FA,FIX,FLY,FURB,G,I,ICN,INP,ISC,INT,LOG,N
    ↪ ,NPY,PERF,PIE,PLC,PLE,PLR,PLW,PT,PYI,Q,RET,RSE,RUF,S,SIM,T,T10,TD,
    ↪ TID,TRY,UP,W,YTT --ignore=A005,ANN001,ANN002,ANN003,ANN204,ANN401,
    ↪ B008,B009,B010,C901,D203,D208,D212,D401,D407,D413,INP001,N801,
    ↪ PLC2801,PLR0904,PLR0911,PLR0912,PLR0913,PLR0914,PLR0915,PLR0916,
    ↪ PLR0917,PLR1702,PLR2004,PLR6301,PT011,PT012,PT013,PYI041,RUF100,S,
    ↪ T201,TRY003,UP035,W --line-length 79 if_else_nested.py
PLR5501 [*] Use `elif` instead of `else` then `if`, to reduce
    ↪ indentation
  --> if_else_nested.py:12:1
   |
10 |       else:      # This means that a > b and c >= a.
11 |           print(f"{c} is the greatest number.")
12 | / else:          # This means that b >= a.
13 | |     if b > c:  # This means that b >= a and b > c.
   | |____^
14 |           print(f"{b} is the greatest number.")
15 |       else:      # This means that b >= a and c >= b.
   |
help: Convert to `elif`

Found 1 error.
[*] 1 fixable with the `--fix` option.
# ruff 0.12.12 failed with exit code 1.
```

Beispiel (2)

- Schauen wir uns nochmal unser Programm `if_else_nested.py` an.
- Sieht eigentlich gut aus.
- Was sagt eigentlich Ruff dazu?
- Hm. Da hat Ruff eigentlich recht.
- Machen wir das mal.

```
$ ruff check --target-version py312 --select=A,AIR,ANN,ASYNC,B,BLE,C,C4,
    ↪ COM,D,DJ,DTZ,E,ERA,EXE,F,FA,FIX,FLY,FURB,G,I,ICN,INP,ISC,INT,LOG,N
    ↪ ,NPY,PERF,PIE,PLC,PLE,PLR,PLW,PT,PYI,Q,RET,RSE,RUF,S,SIM,T,T10,TD,
    ↪ TID,TRY,UP,W,YTT --ignore=A005,ANN001,ANN002,ANN003,ANN204,ANN401,
    ↪ B008,B009,B010,C901,D203,D208,D212,D401,D407,D413,INP001,N801,
    ↪ PLC2801,PLR0904,PLR0911,PLR0912,PLR0913,PLR0914,PLR0915,PLR0916,
    ↪ PLR0917,PLR1702,PLR2004,PLR6301,PT011,PT012,PT013,PYI041,RUF100,S,
    ↪ T201,TRY003,UP035,W --line-length 79 if_else_nested.py
PLR5501 [*] Use `elif` instead of `else` then `if`, to reduce
    ↪ indentation
  --> if_else_nested.py:12:1
   |
10 |         else:       # This means that a > b and c >= a.
11 |             print(f"{c} is the greatest number.")
12 | / else:           # This means that b >= a.
13 | |     if b > c:  # This means that b >= a and b > c.
   | |____^
14 |             print(f"{b} is the greatest number.")
15 |         else:       # This means that b >= a and c >= b.
   |
help: Convert to `elif`

Found 1 error.
[*] 1 fixable with the `--fix` option.
# ruff 0.12.12 failed with exit code 1.
```

Beispiel (2)

- Schauen wir uns nochmal unser Programm `if_else_nested.py` an.
- Sieht eigentlich gut aus.
- Was sagt eigentlich Ruff dazu?
- Hm. Da hat Ruff eigentlich recht.
- Machen wir das mal.

```python
"""An example of using the nested `if-else` statements."""

a: int = 13  # the first number
b: int = 7  # the second number
c: int = 9  # the third number

if a > b:       # This means that a > b
    if a > c:  # This means that a > b and a > c.
        print(f"{a} is the greatest number.")
    else:       # This means that a > b and c >= a.
        print(f"{c} is the greatest number.")
else:           # This means that b >= a.
    if b > c:  # This means that b >= a and b > c.
        print(f"{b} is the greatest number.")
    else:       # This means that b >= a and c >= b.
        print(f"{c} is the greatest number.")

# Some side information: The max and min function are very useful.
print(f"The maximum is {max(a, b, c)}.")  # max: get maximum of a
    ↪ sequence
print(f"The minimum is {min(a, b, c)}.")  # min: get minimum of a
    ↪ sequence
```

↓ `python3 if_else_nested.py` ↓

```
13 is the greatest number.
The maximum is 13.
The minimum is 7.
```

Beispiel (2)

- Schauen wir uns nochmal unser Programm `if_else_nested.py` an.
- Sieht eigentlich gut aus.
- Was sagt eigentlich Ruff dazu?
- Hm. Da hat Ruff eigentlich recht.
- Machen wir das mal.
- Das sieht wirklich besser aus.

```python
"""An example of using the nested `if-else` statements and `elif`."""

a: int = 13  # the first number
b: int = 7  # the second number
c: int = 9  # the third number

if a > b:       # This means that a > b
    if a > c:  # This means that a > b and a > c.
        print(f"{a} is the greatest number.")
    else:       # This means that a > b and c >= a.
        print(f"{c} is the greatest number.")
elif b > c:     # This means that b >= a and b > c.
    print(f"{b} is the greatest number.")
else:           # This means that b >= a and c >= b.
    print(f"{c} is the greatest number.")
```

↓ `python3 if_elif_nested.py` ↓

```
13 is the greatest number.
```

Beispiel (2)

- Schauen wir uns nochmal unser Programm `if_else_nested.py` an.
- Sieht eigentlich gut aus.
- Was sagt eigentlich Ruff dazu?
- Hm. Da hat Ruff eigentlich recht.
- Machen wir das mal.
- Das sieht wirklich besser aus.
- Sehen Sie: Manchmal können wir was von einem Linter lernen.

```python
"""An example of using the nested `if-else` statements and `elif`."""

a: int = 13  # the first number
b: int = 7  # the second number
c: int = 9  # the third number

if a > b:       # This means that a > b
    if a > c:  # This means that a > b and a > c.
        print(f"{a} is the greatest number.")
    else:       # This means that a > b and c >= a.
        print(f"{c} is the greatest number.")
elif b > c:     # This means that b >= a and b > c.
    print(f"{b} is the greatest number.")
else:           # This means that b >= a and c >= b.
    print(f"{c} is the greatest number.")
```

↓ `python3 if_elif_nested.py` ↓

```
13 is the greatest number.
```

Beispiel (2)

- Schauen wir uns nochmal unser Programm `if_else_nested.py` an.
- Sieht eigentlich gut aus.
- Was sagt eigentlich Ruff dazu?
- Hm. Da hat Ruff eigentlich recht.
- Machen wir das mal.
- Das sieht wirklich besser aus.
- Sehen Sie: Manchmal können wir was von einem Linter lernen.
- Diese Werkzeuge können uns schon echt helfen, die Kode-Qualität zu vergessen.

```python
"""An example of using the nested `if-else` statements and `elif`."""

a: int = 13  # the first number
b: int = 7  # the second number
c: int = 9  # the third number

if a > b:       # This means that a > b
    if a > c:  # This means that a > b and a > c.
        print(f"{a} is the greatest number.")
    else:       # This means that a > b and c >= a.
        print(f"{c} is the greatest number.")
elif b > c:     # This means that b >= a and b > c.
    print(f"{b} is the greatest number.")
else:           # This means that b >= a and c >= b.
    print(f"{c} is the greatest number.")
```

↓ `python3 if_elif_nested.py` ↓

```
13 is the greatest number.
```

Beispiel (2)

- Schauen wir uns nochmal unser Programm `if_else_nested.py` an.
- Sieht eigentlich gut aus.
- Was sagt eigentlich Ruff dazu?
- Hm. Da hat Ruff eigentlich recht.
- Machen wir das mal.
- Das sieht wirklich besser aus.
- Sehen Sie: Manchmal können wir was von einem Linter lernen.
- Diese Werkzeuge können uns schon echt helfen, die Kode-Qualität zu vergessen.
- Wir sparen eine Zeile Kode.

```python
"""An example of using the nested `if-else` statements and `elif`."""

a: int = 13  # the first number
b: int = 7  # the second number
c: int = 9  # the third number

if a > b:       # This means that a > b
    if a > c:  # This means that a > b and a > c.
        print(f"{a} is the greatest number.")
    else:       # This means that a > b and c >= a.
        print(f"{c} is the greatest number.")
elif b > c:     # This means that b >= a and b > c.
    print(f"{b} is the greatest number.")
else:           # This means that b >= a and c >= b.
    print(f"{c} is the greatest number.")
```

↓ `python3 if_elif_nested.py` ↓

```
13 is the greatest number.
```

Beispiel (2)

- Schauen wir uns nochmal unser Programm `if_else_nested.py` an.
- Sehen Sie: Manchmal können wir was von einem Linter lernen.
- Diese Werkzeuge können uns schon echt helfen, die Kode-Qualität zu vergessen.
- Wir sparen eine Zeile Kode.

Gute Praxis

Bevorzugen Sie `elif` über geschachtelte `else ... if` Konstrukte[15].

Inline bzw. Ternäres `if-else`

Use Case

- Ein häufiger Use-Case für `if...else` ist die konditionelle Zuweisung von Werten auf Variablen.

Use Case

- Ein häufiger Use-Case für `if...else` ist die konditionelle Zuweisung von Werten auf Variablen.
- Hier haben wir so ein Beispiel.

```python
"""An example of if-elif-else expressions that could be inlined."""

number: int = 100  # the number

# Let's say: Numbers with an absolute value less than ten are small.
# If their absolute value is >= ten, they are large.
size: str
if abs(number) < 10:  # Just a random threshold for this example...
    size = "small"    # If |number| < 10, we say the number is "small".
else:
    size = "large"    # If |number| >= 10, we say the number is "large".

# Numbers can be positive, negative, or unsigned (0 is unsigned).
sign: str
if number < 0:    # If the number is < 0, the sign is "negative".
    sign = "negative"
elif number > 0:  # Otherwise, if it is > 0, the sign is "positive".
    sign = "positive"
else:             # "unsigned" means neither "positive" nor "negative".
    sign = "unsigned"

print(f"The number {number} is {size} and {sign}.")  # Print the result.
```

↓ `python3 if_else_could_be_inline.py` ↓

```
The number 100 is large and positive.
```

Use Case

- Ein häufiger Use-Case für `if...else` ist die konditionelle Zuweisung von Werten auf Variablen.
- Hier haben wir so ein Beispiel.
- Wir wollen ein Programm schreiben, dass uns mitteilt, ob eine Zahl groß oder klein bzw. positiv oder negativ ist.

```python
"""An example of if-elif-else expressions that could be inlined."""

number: int = 100  # the number

# Let's say: Numbers with an absolute value less than ten are small.
# If their absolute value is >= ten, they are large.
size: str
if abs(number) < 10:  # Just a random threshold for this example...
    size = "small"    # If |number| < 10, we say the number is "small".
else:
    size = "large"    # If |number| >= 10, we say the number is "large".

# Numbers can be positive, negative, or unsigned (0 is unsigned).
sign: str
if number < 0:    # If the number is < 0, the sign is "negative".
    sign = "negative"
elif number > 0:  # Otherwise, if it is > 0, the sign is "positive".
    sign = "positive"
else:             # "unsigned" means neither "positive" nor "negative".
    sign = "unsigned"

print(f"The number {number} is {size} and {sign}.")  # Print the result.
```

↓ `python3 if_else_could_be_inline.py` ↓

```
The number 100 is large and positive.
```

Use Case

- Ein häufiger Use-Case für `if...else` ist die konditionelle Zuweisung von Werten auf Variablen.
- Hier haben wir so ein Beispiel.
- Wir wollen ein Programm schreiben, dass uns mitteilt, ob eine Zahl groß oder klein bzw. positiv oder negativ ist.
- Nehmen wir an, dass die Zahl in der Ganzzahl-Variablen `number` gespeichert ist.

```python
"""An example of if-elif-else expressions that could be inlined."""

number: int = 100  # the number

# Let's say: Numbers with an absolute value less than ten are small.
# If their absolute value is >= ten, they are large.
size: str
if abs(number) < 10:  # Just a random threshold for this example...
    size = "small"    # If |number| < 10, we say the number is "small".
else:
    size = "large"    # If |number| >= 10, we say the number is "large".

# Numbers can be positive, negative, or unsigned (0 is unsigned).
sign: str
if number < 0:    # If the number is < 0, the sign is "negative".
    sign = "negative"
elif number > 0:  # Otherwise, if it is > 0, the sign is "positive".
    sign = "positive"
else:             # "unsigned" means neither "positive" nor "negative".
    sign = "unsigned"

print(f"The number {number} is {size} and {sign}.")  # Print the result.
```

↓ `python3 if_else_could_be_inline.py` ↓

```
The number 100 is large and positive.
```

Use Case

- Ein häufiger Use-Case für `if...else` ist die konditionelle Zuweisung von Werten auf Variablen.
- Hier haben wir so ein Beispiel.
- Wir wollen ein Programm schreiben, dass uns mitteilt, ob eine Zahl groß oder klein bzw. positiv oder negativ ist.
- Nehmen wir an, dass die Zahl in der Ganzzahl-Variablen `number` gespeichert ist.
- Als Beispiel nehmen wir den Wert `100`.

```python
"""An example of if-elif-else expressions that could be inlined."""

number: int = 100  # the number

# Let's say: Numbers with an absolute value less than ten are small.
# If their absolute value is >= ten, they are large.
size: str
if abs(number) < 10:  # Just a random threshold for this example...
    size = "small"    # If |number| < 10, we say the number is "small".
else:
    size = "large"    # If |number| >= 10, we say the number is "large".

# Numbers can be positive, negative, or unsigned (0 is unsigned).
sign: str
if number < 0:    # If the number is < 0, the sign is "negative".
    sign = "negative"
elif number > 0:  # Otherwise, if it is > 0, the sign is "positive".
    sign = "positive"
else:             # "unsigned" means neither "positive" nor "negative".
    sign = "unsigned"

print(f"The number {number} is {size} and {sign}.")  # Print the result.
```

↓ `python3 if_else_could_be_inline.py` ↓

```
The number 100 is large and positive.
```

Use Case

- Hier haben wir so ein Beispiel.
- Wir wollen ein Programm schreiben, dass uns mitteilt, ob eine Zahl groß oder klein bzw. positiv oder negativ ist.
- Nehmen wir an, dass die Zahl in der Ganzzahl-Variablen `number` gespeichert ist.
- Als Beispiel nehmen wir den Wert `100`.
- Im Beispiel nehmen wir auch an, dass eine Zahl `number` deren Betrag `|number|` weniger als Zehn ist, als klein angesehen wird.

```python
"""An example of if-elif-else expressions that could be inlined."""

number: int = 100  # the number

# Let's say: Numbers with an absolute value less than ten are small.
# If their absolute value is >= ten, they are large.
size: str
if abs(number) < 10:  # Just a random threshold for this example...
    size = "small"    # If |number| < 10, we say the number is "small".
else:
    size = "large"    # If |number| >= 10, we say the number is "large".

# Numbers can be positive, negative, or unsigned (0 is unsigned).
sign: str
if number < 0:    # If the number is < 0, the sign is "negative".
    sign = "negative"
elif number > 0:  # Otherwise, if it is > 0, the sign is "positive".
    sign = "positive"
else:             # "unsigned" means neither "positive" nor "negative".
    sign = "unsigned"

print(f"The number {number} is {size} and {sign}.")  # Print the result.
```

↓ `python3 if_else_could_be_inline.py` ↓

```
The number 100 is large and positive.
```

Use Case

- Wir wollen ein Programm schreiben, dass uns mitteilt, ob eine Zahl groß oder klein bzw. positiv oder negativ ist.
- Nehmen wir an, dass die Zahl in der Ganzzahl-Variablen `number` gespeichert ist.
- Als Beispiel nehmen wir den Wert `100`.
- Im Beispiel nehmen wir auch an, dass eine Zahl `number` deren Betrag `|number|` weniger als Zehn ist, als klein angesehen wird.
- Den Betrag einer Zahl können wir in Python mit der Funktion `abs` berechnen.

```python
"""An example of if-elif-else expressions that could be inlined."""

number: int = 100  # the number

# Let's say: Numbers with an absolute value less than ten are small.
# If their absolute value is >= ten, they are large.
size: str
if abs(number) < 10:  # Just a random threshold for this example...
    size = "small"    # If |number| < 10, we say the number is "small".
else:
    size = "large"    # If |number| >= 10, we say the number is "large".

# Numbers can be positive, negative, or unsigned (0 is unsigned).
sign: str
if number < 0:    # If the number is < 0, the sign is "negative".
    sign = "negative"
elif number > 0:  # Otherwise, if it is > 0, the sign is "positive".
    sign = "positive"
else:             # "unsigned" means neither "positive" nor "negative".
    sign = "unsigned"

print(f"The number {number} is {size} and {sign}.")  # Print the result.
```

↓ `python3 if_else_could_be_inline.py` ↓

```
The number 100 is large and positive.
```

Use Case

- Nehmen wir an, dass die Zahl in der Ganzzahl-Variablen `number` gespeichert ist.
- Als Beispiel nehmen wir den Wert `100`.
- Im Beispiel nehmen wir auch an, dass eine Zahl `number` deren Betrag |`number`| weniger als Zehn ist, als klein angesehen wird.
- Den Betrag einer Zahl können wir in Python mit der Funktion `abs` berechnen.
- Wir können also eine Alternative mit der Bedingung `if abs(number)< 10:` bauen.

```python
"""An example of if-elif-else expressions that could be inlined."""

number: int = 100  # the number

# Let's say: Numbers with an absolute value less than ten are small.
# If their absolute value is >= ten, they are large.
size: str
if abs(number) < 10:  # Just a random threshold for this example...
    size = "small"    # If |number| < 10, we say the number is "small".
else:
    size = "large"    # If |number| >= 10, we say the number is "large".

# Numbers can be positive, negative, or unsigned (0 is unsigned).
sign: str
if number < 0:    # If the number is < 0, the sign is "negative".
    sign = "negative"
elif number > 0:  # Otherwise, if it is > 0, the sign is "positive".
    sign = "positive"
else:             # "unsigned" means neither "positive" nor "negative".
    sign = "unsigned"

print(f"The number {number} is {size} and {sign}.")  # Print the result.
```

↓ `python3 if_else_could_be_inline.py` ↓

```
The number 100 is large and positive.
```

Use Case

- Als Beispiel nehmen wir den Wert `100`.
- Im Beispiel nehmen wir auch an, dass eine Zahl `number` deren Betrag |`number`| weniger als Zehn ist, als klein angesehen wird.
- Den Betrag einer Zahl können wir in Python mit der Funktion `abs` berechnen.
- Wir können also eine Alternative mit der Bedingung `if abs(number)< 10:` bauen.
- Wenn die Bedigung zurtifft, dann speichern wir den String `"small"` in der Variable `size`.

```python
"""An example of if-elif-else expressions that could be inlined."""

number: int = 100  # the number

# Let's say: Numbers with an absolute value less than ten are small.
# If their absolute value is >= ten, they are large.
size: str
if abs(number) < 10:  # Just a random threshold for this example...
    size = "small"    # If |number| < 10, we say the number is "small".
else:
    size = "large"    # If |number| >= 10, we say the number is "large".

# Numbers can be positive, negative, or unsigned (0 is unsigned).
sign: str
if number < 0:    # If the number is < 0, the sign is "negative".
    sign = "negative"
elif number > 0:  # Otherwise, if it is > 0, the sign is "positive".
    sign = "positive"
else:             # "unsigned" means neither "positive" nor "negative".
    sign = "unsigned"

print(f"The number {number} is {size} and {sign}.")  # Print the result.
```

↓ `python3 if_else_could_be_inline.py` ↓

```
The number 100 is large and positive.
```

Use Case

- Im Beispiel nehmen wir auch an, dass eine Zahl `number` deren Betrag `|number|` weniger als Zehn ist, als klein angesehen wird.
- Den Betrag einer Zahl können wir in Python mit der Funktion `abs` berechnen.
- Wir können also eine Alternative mit der Bedingung `if abs(number)< 10:` bauen.
- Wenn die Bedigung zurtifft, dann speichern wir den String `"small"` in der Variable `size`.
- Trifft die Bedingung nicht zu, dann speichern wir stattdessen `"large"` in `size`.

```python
"""An example of if-elif-else expressions that could be inlined."""

number: int = 100  # the number

# Let's say: Numbers with an absolute value less than ten are small.
# If their absolute value is >= ten, they are large.
size: str
if abs(number) < 10:  # Just a random threshold for this example...
    size = "small"    # If |number| < 10, we say the number is "small".
else:
    size = "large"    # If |number| >= 10, we say the number is "large".

# Numbers can be positive, negative, or unsigned (0 is unsigned).
sign: str
if number < 0:    # If the number is < 0, the sign is "negative".
    sign = "negative"
elif number > 0:  # Otherwise, if it is > 0, the sign is "positive".
    sign = "positive"
else:             # "unsigned" means neither "positive" nor "negative".
    sign = "unsigned"

print(f"The number {number} is {size} and {sign}.")  # Print the result.
```

↓ `python3 if_else_could_be_inline.py` ↓

```
The number 100 is large and positive.
```

Use Case

- Wir können also eine Alternative mit der Bedingung `if abs(number)< 10:` bauen.
- Wenn die Bedigung zurtifft, dann speichern wir den String `"small"` in der Variable `size`.
- Trifft die Bedingung nicht zu, dann speichern wir stattdessen `"large"` in `size`.
- Mit einer ähnlichen Alternative belegen wir die Variable `sign`.

```python
"""An example of if-elif-else expressions that could be inlined."""

number: int = 100  # the number

# Let's say: Numbers with an absolute value less than ten are small.
# If their absolute value is >= ten, they are large.
size: str
if abs(number) < 10:  # Just a random threshold for this example...
    size = "small"    # If |number| < 10, we say the number is "small".
else:
    size = "large"    # If |number| >= 10, we say the number is "large".

# Numbers can be positive, negative, or unsigned (0 is unsigned).
sign: str
if number < 0:    # If the number is < 0, the sign is "negative".
    sign = "negative"
elif number > 0:  # Otherwise, if it is > 0, the sign is "positive".
    sign = "positive"
else:             # "unsigned" means neither "positive" nor "negative".
    sign = "unsigned"

print(f"The number {number} is {size} and {sign}.")  # Print the result.
```

↓ `python3 if_else_could_be_inline.py` ↓

```
The number 100 is large and positive.
```

Use Case

- Wenn die Bedigung zurtifft, dann speichern wir den String `"small"` in der Variable `size`.
- Trifft die Bedingung nicht zu, dann speichern wir stattdessen `"large"` in `size`.
- Mit einer ähnlichen Alternative belegen wir die Variable `sign`.
- Wir speichern `"negative"` in `sign` wenn `number < 0` zutrifft, `"positive"` wenn `number > 0`, und andernfalls, also wenn `number == 0`, speichern wir `"unsigned"`.

```python
"""An example of if-elif-else expressions that could be inlined."""

number: int = 100  # the number

# Let's say: Numbers with an absolute value less than ten are small.
# If their absolute value is >= ten, they are large.
size: str
if abs(number) < 10:  # Just a random threshold for this example...
    size = "small"    # If |number| < 10, we say the number is "small".
else:
    size = "large"    # If |number| >= 10, we say the number is "large".

# Numbers can be positive, negative, or unsigned (0 is unsigned).
sign: str
if number < 0:    # If the number is < 0, the sign is "negative".
    sign = "negative"
elif number > 0:  # Otherwise, if it is > 0, the sign is "positive".
    sign = "positive"
else:             # "unsigned" means neither "positive" nor "negative".
    sign = "unsigned"

print(f"The number {number} is {size} and {sign}.")  # Print the result.
```

↓ `python3 if_else_could_be_inline.py` ↓

```
The number 100 is large and positive.
```

Use Case

- Trifft die Bedingung nicht zu, dann speichern wir stattdessen `"large"` in `size`.
- Mit einer ähnlichen Alternative belegen wir die Variable `sign`.
- Wir speichern `"negative"` in `sign` wenn `number < 0` zutrifft, `"positive"` wenn `number > 0`, und andernfalls, also wenn `number == 0`, speichern wir `"unsigned"`.
- Am Ende drucken wir das Ergebnis wieder unter Verwendung eines f-Strings.

```python
"""An example of if-elif-else expressions that could be inlined."""

number: int = 100  # the number

# Let's say: Numbers with an absolute value less than ten are small.
# If their absolute value is >= ten, they are large.
size: str
if abs(number) < 10:  # Just a random threshold for this example...
    size = "small"    # If |number| < 10, we say the number is "small".
else:
    size = "large"    # If |number| >= 10, we say the number is "large".

# Numbers can be positive, negative, or unsigned (0 is unsigned).
sign: str
if number < 0:    # If the number is < 0, the sign is "negative".
    sign = "negative"
elif number > 0:  # Otherwise, if it is > 0, the sign is "positive".
    sign = "positive"
else:             # "unsigned" means neither "positive" nor "negative".
    sign = "unsigned"

print(f"The number {number} is {size} and {sign}.")  # Print the result.
```

↓ `python3 if_else_could_be_inline.py` ↓

```
The number 100 is large and positive.
```

Use Case

- Mit einer ähnlichen Alternative belegen wir die Variable `sign`.
- Wir speichern `"negative"` in `sign` wenn `number < 0` zutrifft, `"positive"` wenn `number > 0`, und andernfalls, also wenn `number == 0`, speichern wir `"unsigned"`.
- Am Ende drucken wir das Ergebnis wieder unter Verwendung eines f-Strings.
- Fragen wir mal Ruff, was es von unserem Kode hält.

```python
"""An example of if-elif-else expressions that could be inlined."""

number: int = 100  # the number

# Let's say: Numbers with an absolute value less than ten are small.
# If their absolute value is >= ten, they are large.
size: str
if abs(number) < 10:  # Just a random threshold for this example...
    size = "small"    # If |number| < 10, we say the number is "small".
else:
    size = "large"    # If |number| >= 10, we say the number is "large".

# Numbers can be positive, negative, or unsigned (0 is unsigned).
sign: str
if number < 0:    # If the number is < 0, the sign is "negative".
    sign = "negative"
elif number > 0:  # Otherwise, if it is > 0, the sign is "positive".
    sign = "positive"
else:             # "unsigned" means neither "positive" nor "negative".
    sign = "unsigned"

print(f"The number {number} is {size} and {sign}.")  # Print the result.
```

↓ `python3 if_else_could_be_inline.py` ↓

```
The number 100 is large and positive.
```

Use Case

- Wir speichern `"negative"` in `sign` wenn `number < 0` zutrifft, `"positive"` wenn `number > 0`, und andernfalls, also wenn `number == 0`, speichern wir `"unsigned"`.
- Am Ende drucken wir das Ergebnis wieder unter Verwendung eines f-Strings.
- Fragen wir mal Ruff, was es von unserem Kode hält.

```
$ ruff check --target-version py312 --select=A,AIR,ANN,ASYNC,B,BLE,C,C4,
    ↪ COM,D,DJ,DTZ,E,ERA,EXE,F,FA,FIX,FLY,FURB,G,I,ICN,INP,ISC,INT,LOG,N
    ↪ ,NPY,PERF,PIE,PLC,PLE,PLR,PLW,PT,PYI,Q,RET,RSE,RUF,S,SIM,T,T10,TD,
    ↪ TID,TRY,UP,W,YTT --ignore=A005,ANN001,ANN002,ANN003,ANN204,ANN401,
    ↪ B008,B009,B010,C901,D203,D208,D212,D401,D407,D413,INP001,N801,
    ↪ PLC2801,PLR0904,PLR0911,PLR0912,PLR0913,PLR0914,PLR0915,PLR0916,
    ↪ PLR0917,PLR1702,PLR2004,PLR6301,PT011,PT012,PT013,PYI041,RUF100,S,
    ↪ T201,TRY003,UP035,W --line-length 79 if_else_could_be_inline.py
SIM108 Use ternary operator `size = "small" if abs(number) < 10 else "
    ↪ large"` instead of `if`-`else`-block
  --> if_else_could_be_inline.py:8:1
   |
 6 |   # If their absolute value is >= ten, they are large.
 7 |   size: str
 8 | / if abs(number) < 10:  # Just a random threshold for this example
    ↪ ...
 9 | |     size = "small"    # If |number| < 10, we say the number is "
    ↪ small".
10 | | else:
11 | |     size = "large"    # If |number| >= 10, we say the number is "
    ↪ large".
   | |___________________^
12 |
13 |   # Numbers can be positive, negative, or unsigned (0 is unsigned).
   |
help: Replace `if`-`else`-block with `size = "small" if abs(number) < 10
    ↪  else "large"`

Found 1 error.
# ruff 0.12.12 failed with exit code 1.
```

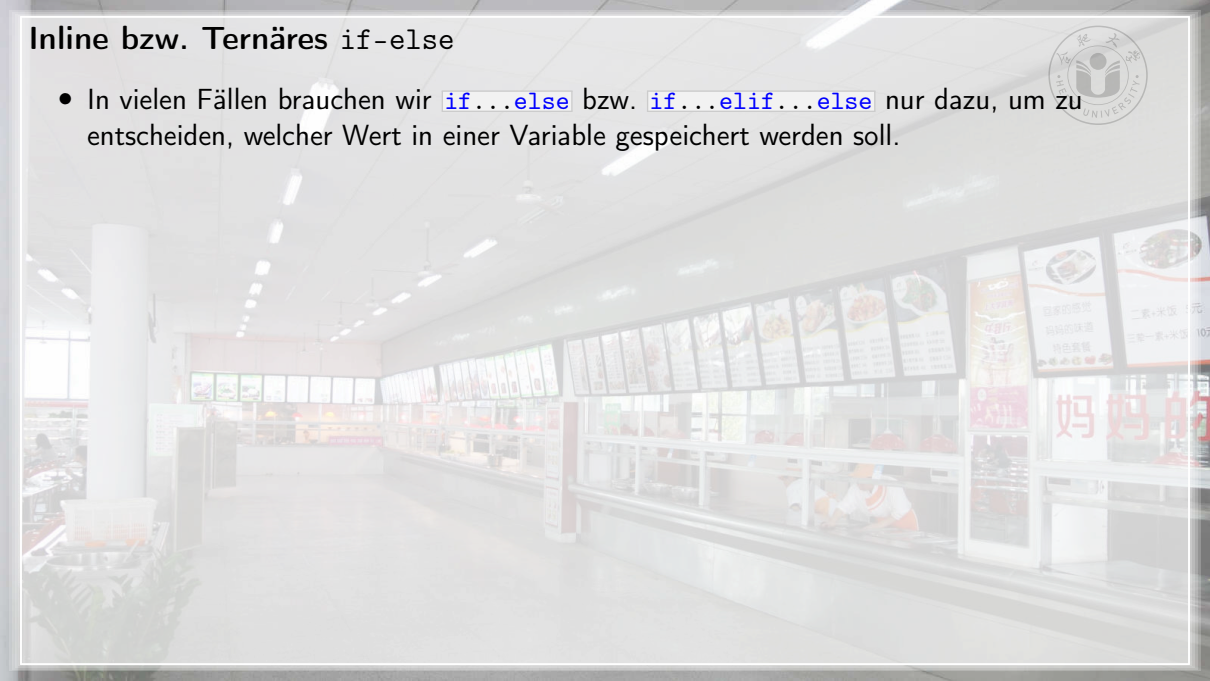
Use Case

- Wir speichern `"negative"` in `sign` wenn `number < 0` zutrifft, `"positive"` wenn `number > 0`, und andernfalls, also wenn `number == 0`, speichern wir `"unsigned"`.
- Am Ende drucken wir das Ergebnis wieder unter Verwendung eines f-Strings.
- Fragen wir mal Ruff, was es von unserem Kode hält.
- Hm. Interessant.

```
$ ruff check --target-version py312 --select=A,AIR,ANN,ASYNC,B,BLE,C,C4,
    ↪ COM,D,DJ,DTZ,E,ERA,EXE,F,FA,FIX,FLY,FURB,G,I,ICN,INP,ISC,INT,LOG,N
    ↪ ,NPY,PERF,PIE,PLC,PLE,PLR,PLW,PT,PYI,Q,RET,RSE,RUF,S,SIM,T,T10,TD,
    ↪ TID,TRY,UP,W,YTT --ignore=A005,ANN001,ANN002,ANN003,ANN204,ANN401,
    ↪ B008,B009,B010,C901,D203,D208,D212,D401,D407,D413,INP001,N801,
    ↪ PLC2801,PLR0904,PLR0911,PLR0912,PLR0913,PLR0914,PLR0915,PLR0916,
    ↪ PLR0917,PLR1702,PLR2004,PLR6301,PT011,PT012,PT013,PYI041,RUF100,S,
    ↪ T201,TRY003,UP035,W --line-length 79 if_else_could_be_inline.py
SIM108 Use ternary operator `size = "small" if abs(number) < 10 else "
    ↪ large"` instead of `if`-`else`-block
  --> if_else_could_be_inline.py:8:1
   |
 6 |   # If their absolute value is >= ten, they are large.
 7 |   size: str
 8 | / if abs(number) < 10:  # Just a random threshold for this example
    ↪ ...
 9 | |     size = "small"    # If |number| < 10, we say the number is "
    ↪ small".
10 | | else:
11 | |     size = "large"    # If |number| >= 10, we say the number is "
    ↪ large".
   | |___________________^
12 |
13 |   # Numbers can be positive, negative, or unsigned (0 is unsigned).
   |
help: Replace `if`-`else`-block with `size = "small" if abs(number) < 10
    ↪  else "large"`

Found 1 error.
# ruff 0.12.12 failed with exit code 1.
```

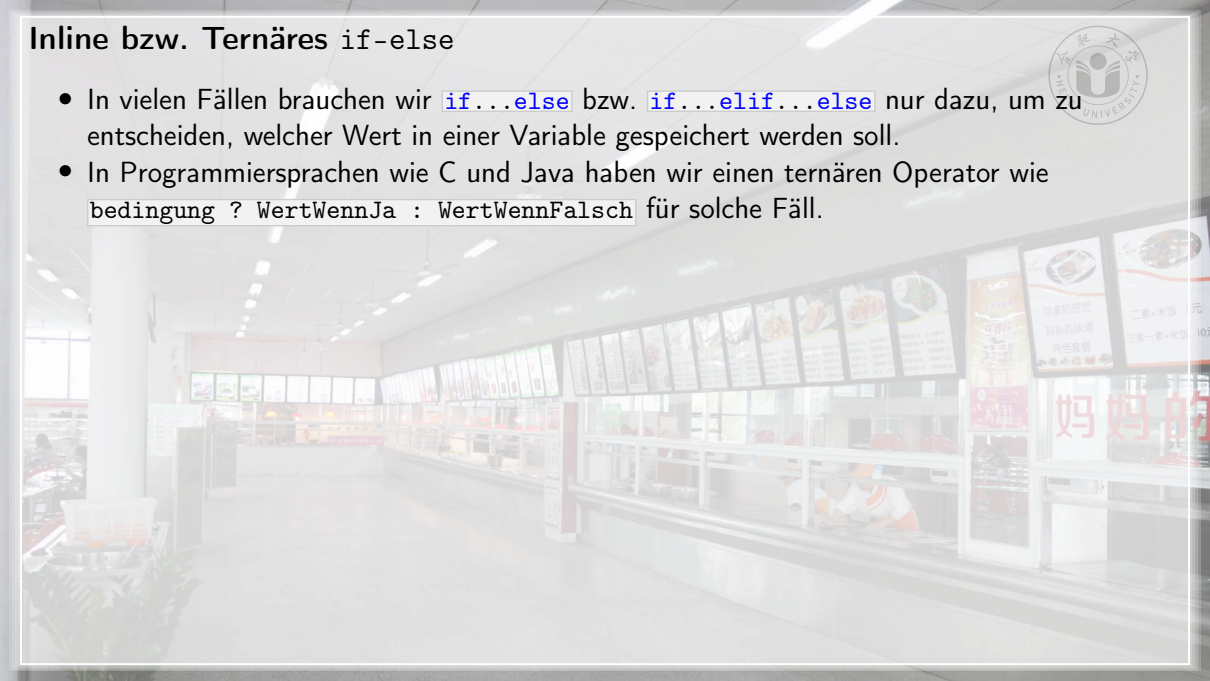
Inline bzw. Ternäres `if-else`

- In vielen Fällen brauchen wir `if...else` bzw. `if...elif...else` nur dazu, um zu entscheiden, welcher Wert in einer Variable gespeichert werden soll.



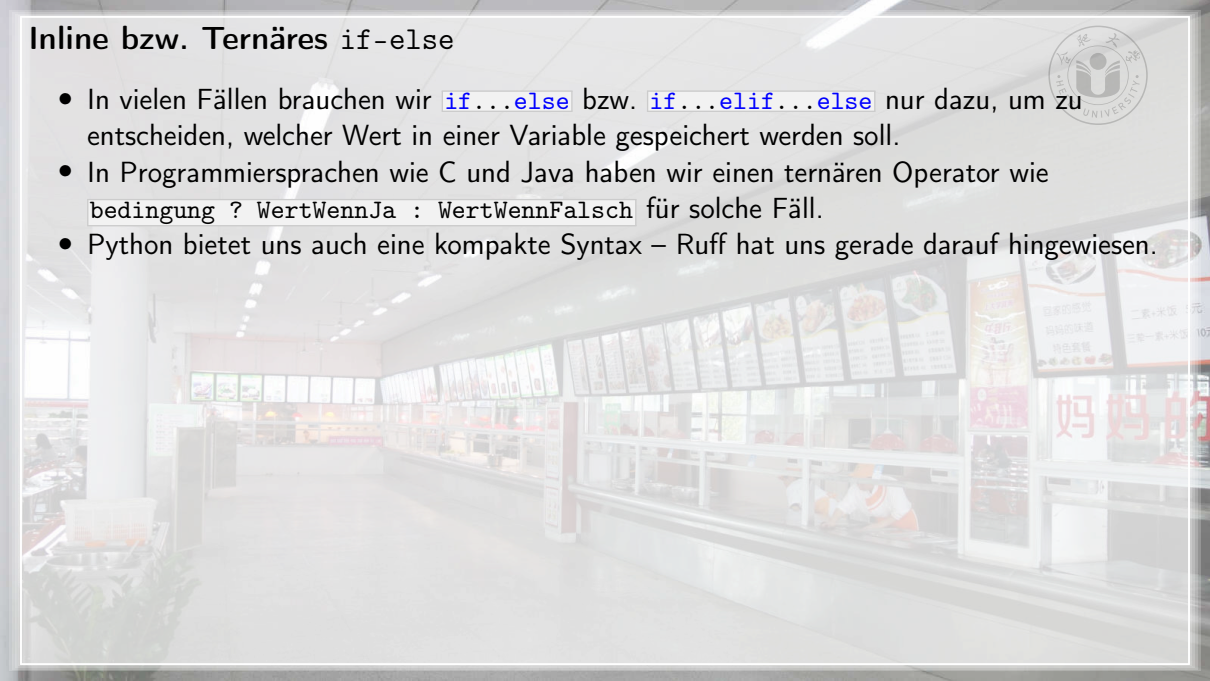
Inline bzw. Ternäres `if-else`

- In vielen Fällen brauchen wir `if...else` bzw. `if...elif...else` nur dazu, um zu entscheiden, welcher Wert in einer Variable gespeichert werden soll.
- In Programmiersprachen wie C und Java haben wir einen ternären Operator wie `bedingung ? WertWennJa : WertWennFalsch` für solche Fäll.



Inline bzw. Ternäres `if-else`

- In vielen Fällen brauchen wir `if...else` bzw. `if...elif...else` nur dazu, um zu entscheiden, welcher Wert in einer Variable gespeichert werden soll.
- In Programmiersprachen wie C und Java haben wir einen ternären Operator wie `bedingung ? WertWennJa : WertWennFalsch` für solche Fäll.
- Python bietet uns auch eine kompakte Syntax – Ruff hat uns gerade darauf hingewiesen.



Inline bzw. Ternäres `if-else`

- In vielen Fällen brauchen wir `if...else` bzw. `if...elif...else` nur dazu, um zu entscheiden, welcher Wert in einer Variable gespeichert werden soll.
- In Programmiersprachen wie C und Java haben wir einen ternären Operator wie `bedingung ? WertWennJa : WertWennFalsch` für solche Fäll.
- Python bietet uns auch eine kompakte Syntax – Ruff hat uns gerade darauf hingewiesen.
- Das inline / ternäre `if...else` Statement sieht wie folgt aus[29]:

Inline bzw. Ternäres `if-else`

- Das inline / ternäre `if...else` Statement sieht wie folgt aus[29]:

```
"""The syntax of a inline "if-else"-statements in Python."""

# If conditionForUsingValueA evaluates to True, then valueA will be
# assigned to variable. Otherwise, valueB will be assigned to variable.
variable = valueA if conditionForUsingValueA else valueB

# If conditionForUsingValueA evaluates to True, then valueA will be
# assigned to variable. Otherwise, conditionForUsingValueB is evaluated.
# If conditionForUsingValueB did evaluate to True, then valueB is
# assigned to variable (but of course, only if conditionForUsingValueA
# was False). If conditionForUsingValueB did evaluate to False (and
# conditionForUsingValueA was also False), then valueC will be assigned
# to variable.
variable = valueA if conditionForUsingValueA else (valueB if
    ↪ conditionForUsingValueB else valueC)
```

Inline bzw. Ternäres `if-else`

- Das inline / ternäre `if...else` Statement sieht wie folgt aus[29]:
- In der ersten Variante wird `valueA` der Variable `variable` zugewiesen when Bedingung `conditionForUsingValueA` `True` ergibt.

```
"""The syntax of a inline "if-else"-statements in Python."""

# If conditionForUsingValueA evaluates to True, then valueA will be
# assigned to variable. Otherwise, valueB will be assigned to variable.
variable = valueA if conditionForUsingValueA else valueB

# If conditionForUsingValueA evaluates to True, then valueA will be
# assigned to variable. Otherwise, conditionForUsingValueB is evaluated.
# If conditionForUsingValueB did evaluate to True, then valueB is
# assigned to variable (but of course, only if conditionForUsingValueA
# was False). If conditionForUsingValueB did evaluate to False (and
# conditionForUsingValueA was also False), then valueC will be assigned
# to variable.
variable = valueA if conditionForUsingValueA else (valueB if
    ↪ conditionForUsingValueB else valueC)
```

Inline bzw. Ternäres `if-else`

- In der ersten Variante wird `valueA` der Variable `variable` zugewiesen when Bedingung `conditionForUsingValueA` `True` ergibt.
- Sonst wird `valueB` zugewiesen.

```
"""The syntax of a inline "if-else"-statements in Python."""

# If conditionForUsingValueA evaluates to True, then valueA will be
# assigned to variable. Otherwise, valueB will be assigned to variable.
variable = valueA if conditionForUsingValueA else valueB

# If conditionForUsingValueA evaluates to True, then valueA will be
# assigned to variable. Otherwise, conditionForUsingValueB is evaluated.
# If conditionForUsingValueB did evaluate to True, then valueB is
# assigned to variable (but of course, only if conditionForUsingValueA
# was False). If conditionForUsingValueB did evaluate to False (and
# conditionForUsingValueA was also False), then valueC will be assigned
# to variable.
variable = valueA if conditionForUsingValueA else (valueB if
    ↪ conditionForUsingValueB else valueC)
```

Inline bzw. Ternäres `if-else`

- Sonst wird `valueB` zugewiesen.
- Solche Statements können beliebig geschachtelt werden, wie die zweite Variante zeigt.

```
"""The syntax of a inline "if-else"-statements in Python."""

# If conditionForUsingValueA evaluates to True, then valueA will be
# assigned to variable. Otherwise, valueB will be assigned to variable.
variable = valueA if conditionForUsingValueA else valueB

# If conditionForUsingValueA evaluates to True, then valueA will be
# assigned to variable. Otherwise, conditionForUsingValueB is evaluated.
# If conditionForUsingValueB did evaluate to True, then valueB is
# assigned to variable (but of course, only if conditionForUsingValueA
# was False). If conditionForUsingValueB did evaluate to False (and
# conditionForUsingValueA was also False), then valueC will be assigned
# to variable.
variable = valueA if conditionForUsingValueA else (valueB if
    ↪ conditionForUsingValueB else valueC)
```

Inline bzw. Ternäres `if-else`

- Solche Statements können beliebig geschachtelt werden, wie die zweite Variante zeigt.
- Hier wird wieder `valueA` in `variable` gespeichert wenn `conditionForUsingValueA` wahr ist.

```
"""The syntax of a inline "if-else"-statements in Python."""

# If conditionForUsingValueA evaluates to True, then valueA will be
# assigned to variable. Otherwise, valueB will be assigned to variable.
variable = valueA if conditionForUsingValueA else valueB

# If conditionForUsingValueA evaluates to True, then valueA will be
# assigned to variable. Otherwise, conditionForUsingValueB is evaluated.
# If conditionForUsingValueB did evaluate to True, then valueB is
# assigned to variable (but of course, only if conditionForUsingValueA
# was False). If conditionForUsingValueB did evaluate to False (and
# conditionForUsingValueA was also False), then valueC will be assigned
# to variable.
variable = valueA if conditionForUsingValueA else (valueB if
    ↪ conditionForUsingValueB else valueC)
```

Inline bzw. Ternäres `if-else`

- Hier wird wieder `valueA` in `variable` gespeichert wenn `conditionForUsingValueA` wahr ist.
- Andernfalls wird `valueB` benutzt, wenn `conditionForUsingValueB` wahr ist und wenn nicht dann eben `valueC`.

```
"""The syntax of a inline "if-else"-statements in Python."""

# If conditionForUsingValueA evaluates to True, then valueA will be
# assigned to variable. Otherwise, valueB will be assigned to variable.
variable = valueA if conditionForUsingValueA else valueB

# If conditionForUsingValueA evaluates to True, then valueA will be
# assigned to variable. Otherwise, conditionForUsingValueB is evaluated.
# If conditionForUsingValueB did evaluate to True, then valueB is
# assigned to variable (but of course, only if conditionForUsingValueA
# was False). If conditionForUsingValueB did evaluate to False (and
# conditionForUsingValueA was also False), then valueC will be assigned
# to variable.
variable = valueA if conditionForUsingValueA else (valueB if
    ↪ conditionForUsingValueB else valueC)
```

Inline bzw. Ternäres `if-else`

- Das ternäre inline `if...else` ist sehr nützlich.

Gute Praxis

Wenn ein `if...else`-Statement nur dazu verwendet wird, zu entscheiden welcher Wert in einer Variable gespeichert wird, dann nehmen Sie stattdessen die ternäre inline Variante, da diese kompakter ist[15].

Beispiel

- Schauen wir uns nochmal das vorige Beispiel an.

```python
"""An example of if-elif-else expressions that could be inlined."""

number: int = 100  # the number

# Let's say: Numbers with an absolute value less than ten are small.
# If their absolute value is >= ten, they are large.
size: str
if abs(number) < 10:  # Just a random threshold for this example...
    size = "small"    # If |number| < 10, we say the number is "small".
else:
    size = "large"    # If |number| >= 10, we say the number is "large".

# Numbers can be positive, negative, or unsigned (0 is unsigned).
sign: str
if number < 0:    # If the number is < 0, the sign is "negative".
    sign = "negative"
elif number > 0:  # Otherwise, if it is > 0, the sign is "positive".
    sign = "positive"
else:             # "unsigned" means neither "positive" nor "negative".
    sign = "unsigned"

print(f"The number {number} is {size} and {sign}.")  # Print the result.
```

↓ python3 if_else_could_be_inline.py ↓

```
The number 100 is large and positive.
```

Beispiel

- Schauen wir uns nochmal das vorige Beispiel an.
- Bauen wir es jetzt mit der ternären inline Variante vom `if...else` nach.

```python
"""An example of if-elif-else expressions that could be inlined."""

number: int = 100  # the number

# Let's say: Numbers with an absolute value less than ten are small.
# If their absolute value is >= ten, they are large.
size: str
if abs(number) < 10:  # Just a random threshold for this example...
    size = "small"    # If |number| < 10, we say the number is "small".
else:
    size = "large"    # If |number| >= 10, we say the number is "large".

# Numbers can be positive, negative, or unsigned (0 is unsigned).
sign: str
if number < 0:    # If the number is < 0, the sign is "negative".
    sign = "negative"
elif number > 0:  # Otherwise, if it is > 0, the sign is "positive".
    sign = "positive"
else:             # "unsigned" means neither "positive" nor "negative".
    sign = "unsigned"

print(f"The number {number} is {size} and {sign}.")  # Print the result.
```

↓ python3 if_else_could_be_inline.py ↓

```
The number 100 is large and positive.
```

Beispiel

- Schauen wir uns nochmal das vorige Beispiel an.
- Bauen wir es jetzt mit der ternären inline Variante vom `if...else` nach.

```python
"""An example of using the inline if-else expression."""

number: int = 100  # the number

# Let's say: Numbers with an absolute value less than ten are small.
# If their absolute value is >= ten, they are large.
size: str = "small" if abs(number) < 10 else "large"

# Numbers can be positive, negative, or unsigned (0 is unsigned).
sign: str = "negative" if number < 0 else (
    "positive" if number > 0 else "unsigned")

print(f"The number {number} is {size} and {sign}.")  # Print the result.
```

↓ python3 inline_if_else.py ↓

```
The number 100 is large and positive.
```

Beispiel

- Schauen wir uns nochmal das vorige Beispiel an.
- Bauen wir es jetzt mit der ternären inline Variante vom `if...else` nach.
- Wow. Wir brauchen jetzt nur noch 13 anstatt von 22 Zeilen Kode.

```python
"""An example of using the inline if-else expression."""

number: int = 100  # the number

# Let's say: Numbers with an absolute value less than ten are small.
# If their absolute value is >= ten, they are large.
size: str = "small" if abs(number) < 10 else "large"

# Numbers can be positive, negative, or unsigned (0 is unsigned).
sign: str = "negative" if number < 0 else (
    "positive" if number > 0 else "unsigned")

print(f"The number {number} is {size} and {sign}.")  # Print the result.
```

↓ python3 inline_if_else.py ↓

```
The number 100 is large and positive.
```

Beispiel

- Schauen wir uns nochmal das vorige Beispiel an.
- Bauen wir es jetzt mit der ternären inline Variante vom `if...else` nach.
- Wow. Wir brauchen jetzt nur noch 13 anstatt von 22 Zeilen Kode.
- Wieder ein Beispiel dafür, wie nützlich Linter seien können.

```python
"""An example of using the inline if-else expression."""

number: int = 100  # the number

# Let's say: Numbers with an absolute value less than ten are small.
# If their absolute value is >= ten, they are large.
size: str = "small" if abs(number) < 10 else "large"

# Numbers can be positive, negative, or unsigned (0 is unsigned).
sign: str = "negative" if number < 0 else (
    "positive" if number > 0 else "unsigned")

print(f"The number {number} is {size} and {sign}.")  # Print the result.
```

↓ python3 inline_if_else.py ↓

```
The number 100 is large and positive.
```

Zusammenfassung

Zusammenfassung

- Wir sind nun in der Lage, Programme zu bauen, die Entscheidungen basierend auf Daten treffen können.

Zusammenfassung

- Wir sind nun in der Lage, Programme zu bauen, die Entscheidungen basierend auf Daten treffen können.
- Vorher konnten wir nur einfache Berechnungen durchführen und die Ergebnisse einfacher Funktionen berechnen.

Zusammenfassung

- Wir sind nun in der Lage, Programme zu bauen, die Entscheidungen basierend auf Daten treffen können.
- Vorher konnten wir nur einfache Berechnungen durchführen und die Ergebnisse einfacher Funktionen berechnen.
- Nun können unsere Variablen das Ergebnis einer Funktion $\mathcal{A}$ empfangen, wenn unsere Eingabedaten die Bedingung B erfüllen und andernfalls das Ergebnis einer Funktion $\mathcal{C}$.

Zusammenfassung

- Wir sind nun in der Lage, Programme zu bauen, die Entscheidungen basierend auf Daten treffen können.
- Vorher konnten wir nur einfache Berechnungen durchführen und die Ergebnisse einfacher Funktionen berechnen.
- Nun können unsere Variablen das Ergebnis einer Funktion $\mathcal{A}$ empfangen, wenn unsere Eingabedaten die Bedingung B erfüllen und andernfalls das Ergebnis einer Funktion $\mathcal{C}$.
- Das ist schon ziemlich cool.

Zusammenfassung

- Wir sind nun in der Lage, Programme zu bauen, die Entscheidungen basierend auf Daten treffen können.
- Vorher konnten wir nur einfache Berechnungen durchführen und die Ergebnisse einfacher Funktionen berechnen.
- Nun können unsere Variablen das Ergebnis einer Funktion $\mathcal{A}$ empfangen, wenn unsere Eingabedaten die Bedingung B erfüllen und andernfalls das Ergebnis einer Funktion $\mathcal{C}$.
- Das ist schon ziemlich cool.
- Wir könnten jetzt schon hard-kodierte Entscheidungsbäume implementieren[21,24] und im Grunde war ja `if_elif_example.py` ein Beispiel genau dafür.

Zusammenfassung

- Wir sind nun in der Lage, Programme zu bauen, die Entscheidungen basierend auf Daten treffen können.
- Vorher konnten wir nur einfache Berechnungen durchführen und die Ergebnisse einfacher Funktionen berechnen.
- Nun können unsere Variablen das Ergebnis einer Funktion $\mathcal{A}$ empfangen, wenn unsere Eingabedaten die Bedingung B erfüllen und andernfalls das Ergebnis einer Funktion $\mathcal{C}$.
- Das ist schon ziemlich cool.
- Wir könnten jetzt schon hard-kodierte Entscheidungsbäume implementieren[21,24] und im Grunde war ja `if_elif_example.py` ein Beispiel genau dafür.
- Mit Alternativen können wir im Grunde "über Instruktionen springen".
- Mit der Ausnahme, dass wir Befehle jetzt auch übergehen können, werden unsere Programm allerdings immer noch in der Reihenfolge abgearbeitet, in der wir sie aufgeschrieben haben.

Zusammenfassung

- Wir sind nun in der Lage, Programme zu bauen, die Entscheidungen basierend auf Daten treffen können.
- Vorher konnten wir nur einfache Berechnungen durchführen und die Ergebnisse einfacher Funktionen berechnen.
- Nun können unsere Variablen das Ergebnis einer Funktion $\mathcal{A}$ empfangen, wenn unsere Eingabedaten die Bedingung B erfüllen und andernfalls das Ergebnis einer Funktion $\mathcal{C}$.
- Das ist schon ziemlich cool.
- Wir könnten jetzt schon hard-kodierte Entscheidungsbäume implementieren[21,24] und im Grunde war ja `if_elif_example.py` ein Beispiel genau dafür.
- Mit Alternativen können wir im Grunde "über Instruktionen springen".
- Mit der Ausnahme, dass wir Befehle jetzt auch übergehen können, werden unsere Programm allerdings immer noch in der Reihenfolge abgearbeitet, in der wir sie aufgeschrieben haben.
- Wir können Instruktionen noch nicht mehrfach ausführen bzw. im Kode "zurück springen".

Zusammenfassung

- Wir sind nun in der Lage, Programme zu bauen, die Entscheidungen basierend auf Daten treffen können.
- Vorher konnten wir nur einfache Berechnungen durchführen und die Ergebnisse einfacher Funktionen berechnen.
- Nun können unsere Variablen das Ergebnis einer Funktion $\mathcal{A}$ empfangen, wenn unsere Eingabedaten die Bedingung B erfüllen und andernfalls das Ergebnis einer Funktion $\mathcal{C}$.
- Das ist schon ziemlich cool.
- Wir könnten jetzt schon hard-kodierte Entscheidungsbäume implementieren[21,24] und im Grunde war ja `if_elif_example.py` ein Beispiel genau dafür.
- Mit Alternativen können wir im Grunde "über Instruktionen springen".
- Mit der Ausnahme, dass wir Befehle jetzt auch übergehen können, werden unsere Programm allerdings immer noch in der Reihenfolge abgearbeitet, in der wir sie aufgeschrieben haben.
- Wir können Instruktionen noch nicht mehrfach ausführen bzw. im Kode "zurück springen".
- Warten Sie mal auf die nächsten Einheiten...

谢谢您们！

Thank you!

Vielen Dank!

References I

[1] Joshua Bloch. *Effective Java*. Reading, MA, USA: Addison-Wesley Professional, Mai 2008. ISBN: 978-0-321-35668-0 (siehe S. 191).

[2] Florian Bruhin. *Python f-Strings*. Winterthur, Switzerland: Bruhin Software, 31. Mai 2023. URL: https://fstring.help (besucht am 2024-07-25) (siehe S. 191).

[3] Slobodan Dmitrović. *Modern C for Absolute Beginners: A Friendly Introduction to the C Programming Language*. New York, NY, USA: Apress Media, LLC, März 2024. ISBN: 979-8-8688-0224-9 (siehe S. 191).

[4] "Formatted String Literals". In: *Python 3 Documentation. The Python Tutorial*. Beaverton, OR, USA: Python Software Foundation (PSF), 2001–2025. Kap. 7.1.1. URL: https://docs.python.org/3/tutorial/inputoutput.html#formatted-string-literals (besucht am 2024-07-25) (siehe S. 191).

[5] Bhavesh Gawade. "Mastering F-Strings in Python: Efficient String Handling in Python Using Smart F-Strings". In: *C O D E B*. Mumbai, Maharashtra, India: Code B Solutions Pvt Ltd, 25. Apr.–3. Juni 2025. URL: https://code-b.dev/blog/f-strings-in-python (besucht am 2025-08-04) (siehe S. 191).

[6] Olaf Górski. "Why f-strings are awesome: Performance of different string concatenation methods in Python". In: *DEV Community*. Sacramento, CA, USA: DEV Community Inc., 8. Nov. 2022. URL: https://dev.to/grski/performance-of-different-string-concatenation-methods-in-python-why-f-strings-are-awesome-2e97 (besucht am 2025-08-04) (siehe S. 191).

[7] John Hunt. *A Beginners Guide to Python 3 Programming*. 2. Aufl. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2023. ISBN: 978-3-031-35121-1. doi:10.1007/978-3-031-35122-8 (siehe S. 191).

[8] Stephen Curtis Johnson. *Lint, a C Program Checker*. Computing Science Technical Report 78–1273. New York, NY, USA: Bell Telephone Laboratories, Incorporated, 25. Okt. 1978. URL: https://wolfram.schneider.org/bsd/7thEdManVol2/lint/lint.pdf (besucht am 2024-08-23) (siehe S. 191).

[9] Łukasz Langa. *Literature Overview for Type Hints*. Python Enhancement Proposal (PEP) 482. Beaverton, OR, USA: Python Software Foundation (PSF), 8. Jan. 2015. URL: https://peps.python.org/pep-0482 (besucht am 2024-10-09) (siehe S. 192).

[10] Kent D. Lee und Steve Hubbard. *Data Structures and Algorithms with Python*. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2015. ISBN: 978-3-319-13071-2. doi:10.1007/978-3-319-13072-9 (siehe S. 191).

References II

[11] Jukka Lehtosalo, Ivan Levkivskyi, Jared Hance, Ethan Smith, Guido van Rossum, Jelle "JelleZijlstra" Zijlstra, Michael J. Sullivan, Shantanu Jain, Xuanda Yang, Jingchen Ye, Nikita Sobolev und Mypy Contributors. *Mypy – Static Typing for Python*. San Francisco, CA, USA: GitHub Inc, 2024. URL: https://github.com/python/mypy (besucht am 2024-08-17) (siehe S. 191).

[12] Marc Loy, Patrick Niemeyer und Daniel Leuck. *Learning Java*. 5. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2020. ISBN: 978-1-4920-5627-0 (siehe S. 191).

[13] Mark Lutz. *Learning Python*. 6. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2025. ISBN: 978-1-0981-7130-8 (siehe S. 191).

[14] Charlie Marsh. "Ruff". In: URL: https://pypi.org/project/ruff (besucht am 2025-08-29) (siehe S. 191).

[15] Charlie Marsh. *ruff: An Extremely Fast Python Linter and Code Formatter, Written in Rust*. New York, NY, USA: Astral Software Inc., 28. Aug. 2022. URL: https://docs.astral.sh/ruff (besucht am 2024-08-23) (siehe S. 132–143, 161–171, 191).

[16] Aaron Maxwell. *What are f-strings in Python and how can I use them?* Oakville, ON, Canada: Infinite Skills Inc, Juni 2017. ISBN: 978-1-4919-9486-3 (siehe S. 191).

[17] "More Control Flow Tools". In: *Python 3 Documentation. The Python Tutorial*. Beaverton, OR, USA: Python Software Foundation (PSF), 2001–2025. Kap. 4. URL: https://docs.python.org/3/tutorial/controlflow.html (besucht am 2025-09-03) (siehe S. 18–20, 54–62, 100–109).

[18] Yasset Pérez-Riverol, Laurent Gatto, Rui Wang, Timo Sachsenberg, Julian Uszkoreit, Felipe da Veiga Leprevost, Christian Fufezan, Tobias Ternent, Stephen J. Eglen, Daniel S. Katz, Tom J. Pollard, Alexander Konovalov, Robert M. Flight, Kai Blin und Juan Antonio Vizcaíno. "Ten Simple Rules for Taking Advantage of Git and GitHub". *PLOS Computational Biology* 12(7), 14. Juli 2016. San Francisco, CA, USA: Public Library of Science (PLOS). ISSN: 1553-7358. doi:10.1371/JOURNAL.PCBI.1004947 (siehe S. 191).

[19] Philippe PRADOS und Maggie Moss. *Allow Writing Union Types as* `X | Y`. Python Enhancement Proposal (PEP) 604. Beaverton, OR, USA: Python Software Foundation (PSF), 28. Aug. 2019. URL: https://peps.python.org/pep-0604 (besucht am 2024-10-09) (siehe S. 110–117).

References III

[20] *Programming Languages – C, Working Document of SC22/WG14*. International Standard ISO/IEC9899:2017 C17 Ballot N2176. Geneva, Switzerland: International Organization for Standardization (ISO) und International Electrotechnical Commission (IEC), Nov. 2017. URL: https://files.lhmouse.com/standards/ISO%20C%20N2176.pdf (besucht am 2024-06-29) (siehe S. 191).

[21] Stuart J. Russell und Peter Norvig. *Artificial Intelligence: A Modern Approach (AIMA)*. 4. Aufl. Hoboken, NJ, USA: Pearson Education, Inc. ISBN: 978-1-292-40113-3. URL: https://aima.cs.berkeley.edu (besucht am 2024-06-27) (siehe S. 178–185).

[22] Yeonhee Ryou, Sangwoo Joh, Joonmo Yang, Sujin Kim und Youil Kim. "Code Understanding Linter to Detect Variable Misuse". In: *37th IEEE/ACM International Conference on Automated Software Engineering (ASE'2022)*. 10.–14. Okt. 2022, Rochester, MI, USA. New York, NY, USA: Association for Computing Machinery (ACM), 2022, 133:1–133:5. ISBN: 978-1-4503-9475-8. doi:10.1145/3551349.3559497 (siehe S. 191).

[23] Veenu Setia, Emily Rodriguez, Aakanksha Gaur, Meg Matthias und Gloria Lotha. "Leap Year". In: *Encyclopaedia Britannica*. Hrsg. von The Editors of Encyclopaedia Britannica. Chicago, IL, USA: Encyclopædia Britannica, Inc., 20. Aug. 2024. URL: https://www.britannica.com/science/leap-year-calendar (besucht am 2024-08-29) (siehe S. 27–33).

[24] Shai Shalev-Shwartz und Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge, England, UK: Cambridge University Press (CUP), Juli 2014. ISBN: 978-1-107-05713-5. URL: http://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning (besucht am 2024-06-27) (siehe S. 178–185).

[25] Anna Skoulikari. *Learning Git*. Sebastopol, CA, USA: O'Reilly Media, Inc., Mai 2023. ISBN: 978-1-0981-3391-7 (siehe S. 191).

[26] Eric V. "ericvsmith" Smith. *Literal String Interpolation*. Python Enhancement Proposal (PEP) 498. Beaverton, OR, USA: Python Software Foundation (PSF), 6. Nov. 2016–9. Sep. 2023. URL: https://peps.python.org/pep-0498 (besucht am 2024-07-25) (siehe S. 191).

[27] *Python 3 Documentation. The Python Tutorial*. Beaverton, OR, USA: Python Software Foundation (PSF), 2001–2025. URL: https://docs.python.org/3/tutorial (besucht am 2025-04-26).

[28] Mariot Tsitoara. *Beginning Git and GitHub: Version Control, Project Management and Teamwork for the New Developer*. New York, NY, USA: Apress Media, LLC, März 2024. ISBN: 979-8-8688-0215-7 (siehe S. 191, 192).

References IV

[29] Guido van Rossum und Raymond Hettinger. *Conditional Expressions*. Python Enhancement Proposal (PEP) 308. Beaverton, OR, USA: Python Software Foundation (PSF), 7.–11. Feb. 2003. URL: https://peps.python.org/pep-0308 (besucht am 2025-08-31) (siehe S. 161–166).

[30] Guido van Rossum und Łukasz Langa. *Type Hints*. Python Enhancement Proposal (PEP) 484. Beaverton, OR, USA: Python Software Foundation (PSF), 29. Sep. 2014. URL: https://peps.python.org/pep-0484 (besucht am 2024-08-22) (siehe S. 192).

[31] Guido van Rossum, Barry Warsaw und Alyssa Coghlan. *Style Guide for Python Code*. Python Enhancement Proposal (PEP) 8. Beaverton, OR, USA: Python Software Foundation (PSF), 5. Juli 2001. URL: https://peps.python.org/pep-0008 (besucht am 2024-07-27) (siehe S. 18–26).

[32] Thomas Weise (汤卫思). *Programming with Python*. Hefei, Anhui, China (中国安徽省合肥市): Hefei University (合肥大学), School of Artificial Intelligence and Big Data (人工智能与大数据学院), Institute of Applied Optimization (应用优化研究所, IAO), 2024–2025. URL: https://thomasweise.github.io/programmingWithPython (besucht am 2025-01-05) (siehe S. 191).

Glossary (in English) I

C is a programming language, which is very successful in system programming situations[3,20].

f-string let you include the results of expressions in strings[2,4–6,16,26]. They can contain expressions (in curly braces) like `f"a{6-1}b"` that are then transformed to text via (string) interpolation, which turns the string to `"a5b"`. F-strings are delimited by `f"..."`.

Git is a distributed Version Control Systems (VCS) which allows multiple users to work on the same code while preserving the history of the code changes[25,28]. Learn more at `https://git-scm.com`.

GitHub is a website where software projects can be hosted and managed via the Git VCS[18,28]. Learn more at `https://github.com`.

Java is another very successful programming language, with roots in the C family of languages[1,12].

linter A linter is a tool for analyzing program code to identify bugs, problems, vulnerabilities, and inconsistent code styles[8,22]. Ruff is an example for a linter used in the Python world.

modulo division is, in Python, done by the operator `%` that computes the remainder of a division. `15 % 6` gives us `3`.

Mypy is a static type checking tool for Python[11] that makes use of type hints. Learn more at `https://github.com/python/mypy` and in[32].

Python The Python programming language[7,10,13,32], i.e., what you will learn about in our book[32]. Learn more at `https://python.org`.

Ruff is a linter and code formatting tool for Python[14,15]. Learn more at `https://docs.astral.sh/ruff` or in[32].

Glossary (in English) II

(string) interpolation In Python, string interpolation is the process where all the expressions in an f-string are evaluated and the final string is constructed. An example for string interpolation is turning `f"Rounded {1.234:.2f}"` to `"Rounded 1.23"`.

type hint are annotations that help programmers and static code analysis tools such as Mypy to better understand what type a variable or function parameter is supposed to be[9,30]. Python is a dynamically typed programming language where you do not need to specify the type of, e.g., a variable. This creates problems for code analysis, both automated as well as manual: For example, it may not always be clear whether a variable or function parameter should be an integer or floating point number. The annotations allow us to explicitly state which type is expected. They are *ignored* during the program execution. They are a basically a piece of documentation.

VCS A *Version Control System* is a software which allows you to manage and preserve the historical development of your program code[28]. A distributed VCS allows multiple users to work on the same code and upload their changes to the server, which then preserves the change history. The most popular distributed VCS is Git.