



合肥大學
HEFEI UNIVERSITY



Programming with Python

16. Gleichheit und Identität

Thomas Weise (汤卫思)
twaise@hfu.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

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-Code 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



1. Einleitung
2. Gleichheit und Identität
3. Zusammenfassung





Einleitung



Einleitung



- Wir benutzen Variablen, um Objekte im Speicher zu referenzieren.

Einleitung



- Wir benutzen Variablen, um Objekte im Speicher zu referenzieren.
- Wenn wir zwei Variablen vergleichen, dann vergleichen wir genau genommen die Objekte, die sie referenzieren.

Einleitung



- Wir benutzen Variablen, um Objekte im Speicher zu referenzieren.
- Wenn wir zwei Variablen vergleichen, dann vergleichen wir genau genommen die Objekte, die sie referenzieren.
- Es ist klar, dass zwei Variablen auf Objekte verweisen können, die *gleich* oder *ungleich* sein können.

Einleitung



- Wir benutzen Variablen, um Objekte im Speicher zu referenzieren.
- Wenn wir zwei Variablen vergleichen, dann vergleichen wir genau genommen die Objekte, die sie referenzieren.
- Es ist klar, dass zwei Variablen auf Objekte verweisen können, die *gleich* oder *ungleich* sein können.
- Sie können aber auch das *selbe Objekt* referenzieren.

Einleitung



- Wir benutzen Variablen, um Objekte im Speicher zu referenzieren.
- Wenn wir zwei Variablen vergleichen, dann vergleichen wir genau genommen die Objekte, die sie referenzieren.
- Es ist klar, dass zwei Variablen auf Objekte verweisen können, die *gleich* oder *ungleich* sein können.
- Sie können aber auch das *selbe Objekt* referenzieren.
- Wir müssen diese beiden Konzepte, *Gleichheit* und *Identität*, unterscheiden.



Gleichheit und Identität



Gleichheit und Identität



- Wir müssen diese beiden Konzepte, *Gleichheit* und *Identität*, unterscheiden.



Gleichheit und Identität



- Wir müssen diese beiden Konzepte, *Gleichheit* und *Identität*, unterscheiden.
- Und das ist einfach.

中国安徽德国中心

合肥德国应用科学学院

Gleichheit und Identität



- Wir müssen diese beiden Konzepte, *Gleichheit* und *Identität*, unterscheiden.
- Und das ist einfach.
- Stellen Sie sich vor, dass Sie eine grüne Jacke kaufen. Wir nennen sie *A*.

中国安徽德国中心

合肥德国应用科学学院

Gleichheit und Identität



- Wir müssen diese beiden Konzepte, *Gleichheit* und *Identität*, unterscheiden.
- Und das ist einfach.
- Stellen Sie sich vor, dass Sie eine grüne Jacke kaufen. Wir nennen sie *A*.
- Am nächsten Tag kaufen Sie eine gleiche grüne Jacke. Wie nennen sie *B*.

Gleichheit und Identität



- Wir müssen diese beiden Konzepte, *Gleichheit* und *Identität*, unterscheiden.
- Und das ist einfach.
- Stellen Sie sich vor, dass Sie eine grüne Jacke kaufen. Wir nennen sie A .
- Am nächsten Tag kaufen Sie eine gleiche grüne Jacke. Wie nennen sie B .
- Nun haben Sie zwei *gleiche* Jacken, A und B . Es gilt $A == B$.

Gleichheit und Identität



- Wir müssen diese beiden Konzepte, *Gleichheit* und *Identität*, unterscheiden.
- Und das ist einfach.
- Stellen Sie sich vor, dass Sie eine grüne Jacke kaufen. Wir nennen sie *A*.
- Am nächsten Tag kaufen Sie eine gleiche grüne Jacke. Wie nennen sie *B*.
- Nun haben Sie zwei *gleiche* Jacken, *A* und *B*. Es gilt $A == B$.
- *A* und *B* sind Namen für zwei separate Objekte mit verschiedener Identität. Es gilt $A \text{ is not } B$.

Gleichheit und Identität



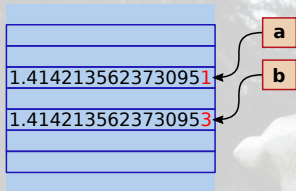
- Wir müssen diese beiden Konzepte, *Gleichheit* und *Identität*, unterscheiden.
- Und das ist einfach.
- Stellen Sie sich vor, dass Sie eine grüne Jacke kaufen. Wir nennen sie *A*.
- Am nächsten Tag kaufen Sie eine gleiche grüne Jacke. Wie nennen sie *B*.
- Nun haben Sie zwei *gleiche* Jacken, *A* und *B*. Es gilt $A == B$.
- *A* und *B* sind Namen für zwei separate Objekte mit verschiedener Identität. Es gilt $A \text{ is not } B$.
- Sie können Jacke *A* in den Kleiderschrank tun. Wenn Sie sie morgen herausholen, können Sie die Jacke *C* nennen.

Gleichheit und Identität



- Wir müssen diese beiden Konzepte, *Gleichheit* und *Identität*, unterscheiden.
- Und das ist einfach.
- Stellen Sie sich vor, dass Sie eine grüne Jacke kaufen. Wir nennen sie *A*.
- Am nächsten Tag kaufen Sie eine gleiche grüne Jacke. Wie nennen sie *B*.
- Nun haben Sie zwei *gleiche* Jacken, *A* und *B*. Es gilt $A == B$.
- *A* und *B* sind Namen für zwei separate Objekte mit verschiedener Identität. Es gilt $A \text{ is not } B$.
- Sie können Jacke *A* in den Kleiderschrank tun. Wenn Sie sie morgen herausholen, können Sie die Jacke *C* nennen.
- Dann sind *A* und *C* zwei Namen für ein und das selbe Objekt. Es gilt $A == C$ und $A \text{ is } C$.

Gleichheit und Identität Illustriert



`a == b: False`

:

:

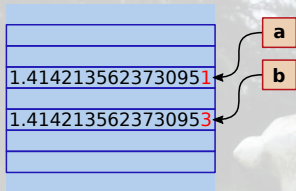
`a is b: False`

:

:

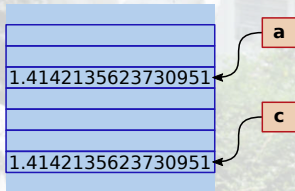
- Wenn wir zwei Variablen `a=1.4142135623730951` und `b=1.4142135623730953` haben, dann referenzieren sie verschiedene Objekte mit verschiedenen `float`-Werten, die an verschiedenen Speicherstellen stehen.

Gleichheit und Identität Illustriert



```
a == b: False
```

```
a is b: False
```

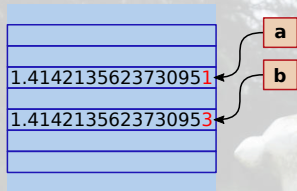


```
a == c: True
```

```
a is c: False
```

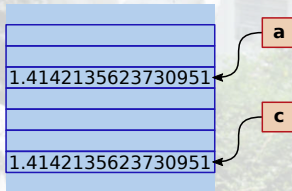
- Wenn wir zwei Variablen `a=1.4142135623730951` und `b=1.4142135623730953` haben, dann referenzieren sie verschiedene Objekte mit verschiedenen `float`-Werten, die an verschiedenen Speicherstellen stehen.
- Jetzt haben wir eine dritte Variable `c`, die ein `float`-Object referenziert, dass den gleichen Wert wie `a` speichert.

Gleichheit und Identität Illustriert



`a == b: False`

`a is b: False`

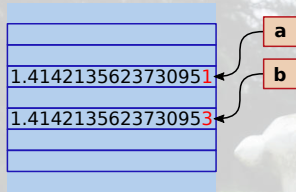


`a == c: True`

`a is c: False`

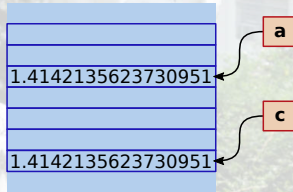
- Wenn wir zwei Variablen `a=1.4142135623730951` und `b=1.4142135623730953` haben, dann referenzieren sie verschiedene Objekte mit verschiedenen `float`-Werten, die an verschiedenen Speicherstellen stehen.
- Jetzt haben wir eine dritte Variable `c`, die ein `float`-Object referenziert, dass den gleichen Wert wie `a` speichert.
- Weil es ein anderes Objekt ist, steht es auch woanders im Speicher.

Gleichheit und Identität Illustriert



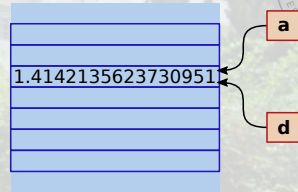
```
a == b: False
```

```
a is b: False
```



```
a == c: True
```

```
a is c: False
```



```
a == d: True
```

```
a is d: True
```

- Wenn wir zwei Variablen `a=1.4142135623730951` und `b=1.4142135623730953` haben, dann referenzieren sie verschiedene Objekte mit verschiedenen `float`-Werten, die an verschiedenen Speicherstellen stehen.
- Jetzt haben wir eine dritte Variable `c`, die ein `float`-Object referenziert, dass den gleichen Wert wie `a` speichert.
- Weil es ein anderes Objekt ist, steht es auch woanders im Speicher.
- Setzen wir `d = a`, dann zeigt `d` auf das selbe (identische) `float`-Object wie `a`, auf die selbe Speicherstelle.

Beispiel 1

```
1  # We define sqrt_2 to be a constant with the value of square root of 2.
2  sqrt_2: float = 1.4142135623730951  # We set the value of the variable.
3  print(f"sqrt_2 = {sqrt_2}")  # We print the value of the variable
4
5  # We can also compute the square root using the `sqrt` function.
6  from math import sqrt  # Import the root function from the math module.
7  sqrt_2_computed: float = sqrt(2.0)  # Compute the square root of 2.0.
8  print(f"{sqrt_2_computed = }")  # Print the value.
9
10 # Let's compare the computed and the constant value:
11 print(f"are they equal: {sqrt_2 == sqrt_2_computed}")
12 print(f"are they the same object: {sqrt_2 is sqrt_2_computed}")
```

↓ python3 identity_1.py ↓

```
1  sqrt_2 = 1.4142135623730951
2  sqrt_2_computed = 1.4142135623730951
3  are they equal: True
4  are they the same object: False
```

Beispiel 2

- String-Literale werden gecached und wiederverwendet, Verbindungen von String-Literalen in einem Ausdruck werden gleich als String-Literale interpretiert – `a`, `b`, und `c` sind das selbe Objekt.

```
1 # String and integer literals and identity.
2 a: str = "Hello World!"
3 b: str = "Hello World!"
4 print(f"Are 'a' and 'b' the same object: {a is b}")
5
6 c: str = "Hello " + "World!"
7 print(f"Are 'a' and 'c' the same object: {a is c}")
8
9 d: str = "Hello"
10 d = d + " World!"
11 print(f"Are 'a' and 'd' the same object: {a is d}")
12 print(f"Are 'a' and 'd' equal objects: {a == d}")
13
14 e: int = 10
15 mul: int = 5
16 f: int = (e * mul) // mul
17 print(f"Are 'e' and 'f' the same object: {e is f}")
18
19 g: int = 1_000_000_000_000_000_000
20 h: int = (g * mul) // mul
21 print(f"Are 'g' and 'h' the same object: {g is h}")
22 print(f"Are 'g' and 'h' equal objects: {g == h}")
```

↓ python3 identity_2.py ↓

```
1 Are 'a' and 'b' the same object: True
2 Are 'a' and 'c' the same object: True
3 Are 'a' and 'd' the same object: False
4 Are 'a' and 'd' equal objects: True
5 Are 'e' and 'f' the same object: True
6 Are 'g' and 'h' the same object: False
7 Are 'g' and 'h' equal objects: True
```


Beispiel 2

- String-Literale werden gecached und wiederverwendet, Verbindungen von String-Literalen in einem Ausdruck werden gleich als String-Literale interpretiert – `a`, `b`, und `c` sind das selbe Objekt.
- `d` wird in zwei Schritten berechnet, kommt nicht aus dem Cache, und ist daher ein anderes Objekt.

```
1 # String and integer literals and identity.
2 a: str = "Hello World!"
3 b: str = "Hello World!"
4 print(f"Are 'a' and 'b' the same object: {a is b}")
5
6 c: str = "Hello " + "World!"
7 print(f"Are 'a' and 'c' the same object: {a is c}")
8
9 d: str = "Hello"
10 d = d + " World!"
11 print(f"Are 'a' and 'd' the same object: {a is d}")
12 print(f"Are 'a' and 'd' equal objects: {a == d}")
13
14 e: int = 10
15 mul: int = 5
16 f: int = (e * mul) // mul
17 print(f"Are 'e' and 'f' the same object: {e is f}")
18
19 g: int = 1_000_000_000_000_000_000
20 h: int = (g * mul) // mul
21 print(f"Are 'g' and 'h' the same object: {g is h}")
22 print(f"Are 'g' and 'h' equal objects: {g == h}")
```

↓ python3 identity_2.py ↓

```
1 Are 'a' and 'b' the same object: True
2 Are 'a' and 'c' the same object: True
3 Are 'a' and 'd' the same object: False
4 Are 'a' and 'd' equal objects: True
5 Are 'e' and 'f' the same object: True
6 Are 'g' and 'h' the same object: False
7 Are 'g' and 'h' equal objects: True
```

Beispiel 2

- String-Literale werden gecached und wiederverwendet, Verbindungen von String-Literalen in einem Ausdruck werden gleich als String-Literale interpretiert – `a`, `b`, und `c` sind das selbe Objekt.
- `d` wird in zwei Schritten berechnet, kommt nicht aus dem Cache, und ist daher ein anderes Objekt.
- Implementationsspezifisch cached und wiederverwendet der Python-Interpreter kleine Integer-Werte (`e` und `f`).

```
1 # String and integer literals and identity.
2 a: str = "Hello World!"
3 b: str = "Hello World!"
4 print(f"Are 'a' and 'b' the same object: {a is b}")
5
6 c: str = "Hello " + "World!"
7 print(f"Are 'a' and 'c' the same object: {a is c}")
8
9 d: str = "Hello"
10 d = d + " World!"
11 print(f"Are 'a' and 'd' the same object: {a is d}")
12 print(f"Are 'a' and 'd' equal objects: {a == d}")
13
14 e: int = 10
15 mul: int = 5
16 f: int = (e * mul) // mul
17 print(f"Are 'e' and 'f' the same object: {e is f}")
18
19 g: int = 1_000_000_000_000_000_000
20 h: int = (g * mul) // mul
21 print(f"Are 'g' and 'h' the same object: {g is h}")
22 print(f"Are 'g' and 'h' equal objects: {g == h}")
```

↓ python3 identity_2.py ↓

```
1 Are 'a' and 'b' the same object: True
2 Are 'a' and 'c' the same object: True
3 Are 'a' and 'd' the same object: False
4 Are 'a' and 'd' equal objects: True
5 Are 'e' and 'f' the same object: True
6 Are 'g' and 'h' the same object: False
7 Are 'g' and 'h' equal objects: True
```

Beispiel 2

- `d` wird in zwei Schritten berechnet, kommt nicht aus dem Cache, und ist daher ein anderes Objekt.
- Implementationsspezifisch cached und wiederverwendet der Python-Interpreter kleine Integer-Werte (`e` und `f`).
- Große aber nicht (`g` und `h`).

```
1 # String and integer literals and identity.
2 a: str = "Hello World!"
3 b: str = "Hello World!"
4 print(f"Are 'a' and 'b' the same object: {a is b}")
5
6 c: str = "Hello " + "World!"
7 print(f"Are 'a' and 'c' the same object: {a is c}")
8
9 d: str = "Hello"
10 d = d + " World!"
11 print(f"Are 'a' and 'd' the same object: {a is d}")
12 print(f"Are 'a' and 'd' equal objects: {a == d}")
13
14 e: int = 10
15 mul: int = 5
16 f: int = (e * mul) // mul
17 print(f"Are 'e' and 'f' the same object: {e is f}")
18
19 g: int = 1_000_000_000_000_000_000
20 h: int = (g * mul) // mul
21 print(f"Are 'g' and 'h' the same object: {g is h}")
22 print(f"Are 'g' and 'h' equal objects: {g == h}")
```

↓ python3 identity_2.py ↓

```
1 Are 'a' and 'b' the same object: True
2 Are 'a' and 'c' the same object: True
3 Are 'a' and 'd' the same object: False
4 Are 'a' and 'd' equal objects: True
5 Are 'e' and 'f' the same object: True
6 Are 'g' and 'h' the same object: False
7 Are 'g' and 'h' equal objects: True
```



Zusammenfassung



Zusammenfassung

- Wir unterscheiden *Gleichheit* und *Identität*.



Zusammenfassung



- Wir unterscheiden *Gleichheit* und *Identität*.
- Gleichheit wird mit dem Operator `==` und Ungleichheit mit `!=` geprüft.

Zusammenfassung



- Wir unterscheiden *Gleichheit* und *Identität*.
- Gleichheit wird mit dem Operator `==` und Ungleichheit mit `!=` geprüft.
- Identität wird mit dem Operator `is` und nicht-Identität mit `is not` geprüft.

Zusammenfassung



- Wir unterscheiden *Gleichheit* und *Identität*.
- Gleichheit wird mit dem Operator `==` und Ungleichheit mit `!=` geprüft.
- Identität wird mit dem Operator `is` und nicht-Identität mit `is not` geprüft.
- Zwei separate Objekte können gleich oder ungleich sein, aber niemals identisch.

Zusammenfassung



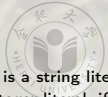
- Wir unterscheiden *Gleichheit* und *Identität*.
- Gleichheit wird mit dem Operator `==` und Ungleichheit mit `!=` geprüft.
- Identität wird mit dem Operator `is` und nicht-Identität mit `is not` geprüft.
- Zwei separate Objekte können gleich oder ungleich sein, aber niemals identisch.
- Zwei identische Objekte sind immer gleich, also `A is B` \Rightarrow `A == B`.



谢谢你们！
Thank you!
Vielen Dank!



Glossary (in English) I



literal A literal is a specific concrete value, something that is written down as-is^{4,7}. In Python, for example, `"abc"` is a string literal, `5` is an integer literal, and `23.3` is a `float` literal. In contrast, `sin(3)` is not a literal. Also, while `5` is an integer literal, if we create a variable `a = 5` then `a` is not a literal either (it is a variable). Hence, literals are values that the Python interpreter reads directly from the source code and creates as objects in memory. They are not something that is the result from a computation or the result of a variable lookup. Python supports some type hints for literals, including the type `LiteralString` for string literals and the type `Literal[xyz]` for arbitrary literals `xyz`.

Mypy is a static type checking tool for Python⁵ that makes use of type hints. Learn more at <https://github.com/python/mypy> and in⁹.

Python The Python programming language^{1,3,6,9}, i.e., what you will learn about in our book⁹. Learn more at <https://python.org>.

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^{2,8}. 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.