



Programming with Python

43. Klassen: Vererbung

Thomas Weise (汤卫思)
tweise@hfuu.edu.cn

School of Artificial Intelligence and Big Data
Hefei University
Hefei, Anhui, China

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-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



1. Einleitung
2. Beispiel: Formen in der Euklidischen Ebene
3. Zusammenfassung





Einleitung



Einleitung

- Vererbung und Spezialisierung sind eines der wichtigsten Konzepte der Object-Orientierten Programmierung (OOP) (EN: *object-oriented programming*).





Einleitung

- Vererbung und Spezialisierung sind eines der wichtigsten Konzepte der Object-Orientierten Programmierung (OOP) (EN: *object-oriented programming*).
- Eine Subklasse kann von einer existierenden Klasse abgeleitet werden.



Einleitung

- Vererbung und Spezialisierung sind eines der wichtigsten Konzepte der Object-Orientierten Programmierung (OOP) (EN: *object-oriented programming*).
- Eine Subklasse kann von einer existierenden Klasse abgeleitet werden.
- Sie erbt dann Methoden und Attribute von der Basisklasse.



Einleitung

- Vererbung und Spezialisierung sind eines der wichtigsten Konzepte der Object-Orientierten Programmierung (OOP) (EN: *object-oriented programming*).
- Eine Subklasse kann von einer existierenden Klasse abgeleitet werden.
- Sie erbt dann Methoden und Attribute von der Basisklasse.
- Sie kann auch neue Methoden und Attribute hinzufügen.



Einleitung

- Vererbung und Spezialisierung sind eines der wichtigsten Konzepte der Object-Orientierten Programmierung (OOP) (EN: *object-oriented programming*).
- Eine Subklasse kann von einer existierenden Klasse abgeleitet werden.
- Sie erbt dann Methoden und Attribute von der Basisklasse.
- Sie kann auch neue Methoden und Attribute hinzufügen.
- Und sie kann Methoden überschreiben, also neu implementieren, also ihr Verhalten ändern.



Einleitung

- Vererbung und Spezialisierung sind eines der wichtigsten Konzepte der Object-Orientierten Programmierung (OOP) (EN: *object-oriented programming*).
- Eine Subklasse kann von einer existierenden Klasse abgeleitet werden.
- Sie erbt dann Methoden und Attribute von der Basisklasse.
- Sie kann auch neue Methoden und Attribute hinzufügen.
- Und sie kann Methoden überschreiben, also neu implementieren, also ihr Verhalten ändern.
- Dieser Mechanismus von Vererbung kann aus verschiedenen Blickwinkeln betrachtet werden.

Vererbung als mengentheoretisches Konzept

- Wir können Vererbung aus der Perspektive der Mengentheorie betrachten.



Vererbung als mengentheoretisches Konzept

- Wir können Vererbung aus der Perspektive der Mengentheorie betrachten.
- Wir können eine Klasse als die Menge aller Objekte, die zu ihr gehören, betrachten.



Vererbung als mengentheoretisches Konzept

- Wir können Vererbung aus der Perspektive der Mengentheorie betrachten.
- Wir können eine Klasse als die Menge aller Objekte, die zu ihr gehören, betrachten.
- Subklassen sind dann Teilmengen dieser Menge.



Vererbung als mengentheoretisches Konzept

- Wir können Vererbung aus der Perspektive der Mengentheorie betrachten.
- Wir können eine Klasse als die Menge aller Objekte, die zu ihr gehören, betrachten.
- Subklassen sind dann Teilmengen dieser Menge.
- Erinnern wir uns an die baumartige Struktur der Built-In Ausnahmeklassen in Python von Einheit 31.



Vererbung als mengentheoretisches Konzept

- Wir können Vererbung aus der Perspektive der Mengentheorie betrachten.
- Wir können eine Klasse als die Menge aller Objekte, die zu ihr gehören, betrachten.
- Subklassen sind dann Teilmengen dieser Menge.
- Erinnern wir uns an die baumartige Struktur der Built-In Ausnahmeklassen in Python von Einheit 31.
- Die „Wurzel“ wurde von `BaseException` geformt.



Vererbung als mengentheoretisches Konzept



- Wir können Vererbung aus der Perspektive der Mengentheorie betrachten.
- Wir können eine Klasse als die Menge aller Objekte, die zu ihr gehören, betrachten.
- Subklassen sind dann Teilmengen dieser Menge.
- Erinnern wir uns an die baumartige Struktur der Built-In Ausnahmeklassen in Python von Einheit 31.
- Die „Wurzel“ wurde von `BaseException` geformt.
- Alle Ausnahmeobjekte sind Instanzen dieser Klasse.

Vererbung als mengentheoretisches Konzept



- Wir können Vererbung aus der Perspektive der Mengentheorie betrachten.
- Wir können eine Klasse als die Menge aller Objekte, die zu ihr gehören, betrachten.
- Subklassen sind dann Teilmengen dieser Menge.
- Erinnern wir uns an die baumartige Struktur der Built-In Ausnahmeklassen in Python von Einheit 31.
- Die „Wurzel“ wurde von `BaseException` geformt.
- Alle Ausnahmeobjekte sind Instanzen dieser Klasse.
- Sie hat eine Subklasse `Exception`.



Vererbung als mengentheoretisches Konzept

- Wir können Vererbung aus der Perspektive der Mengentheorie betrachten.
- Wir können eine Klasse als die Menge aller Objekte, die zu ihr gehören, betrachten.
- Subklassen sind dann Teilmengen dieser Menge.
- Erinnern wir uns an die baumartige Struktur der Built-In Ausnahmeklassen in Python von Einheit 31.
- Die „Wurzel“ wurde von `BaseException` geformt.
- Alle Ausnahmeobjekte sind Instanzen dieser Klasse.
- Sie hat eine Subklasse `Exception`.
- Alle Instanzen von `Exception` sind `BaseExceptions`, aber nicht alle `BaseExceptions` sind `Exceptions`.



Vererbung als mengentheoretisches Konzept

- Wir können Vererbung aus der Perspektive der Mengentheorie betrachten.
- Wir können eine Klasse als die Menge aller Objekte, die zu ihr gehören, betrachten.
- Subklassen sind dann Teilmengen dieser Menge.
- Erinnern wir uns an die baumartige Struktur der Built-In Ausnahmeklassen in Python von Einheit 31.
- Die „Wurzel“ wurde von `BaseException` geformt.
- Alle Ausnahmeobjekte sind Instanzen dieser Klasse.
- Sie hat eine Subklasse `Exception`.
- Alle Instanzen von `Exception` sind `BaseExceptions`, aber nicht alle `BaseExceptions` sind `Exceptions`.
- `ArithmetricError` wiederum ist eine Subklasse von `Exception`, was heist, dass alle `ArithmetricErrors` Instanzen von `Exceptions` sind – aber nicht andersherum.



Vererbung als mengentheoretisches Konzept

- Wir können eine Klasse als die Menge aller Objekte, die zu ihr gehören, betrachten.
- Subklassen sind dann Teilmengen dieser Menge.
- Erinnern wir uns an die baumartige Struktur der Built-In Ausnahmeklassen in Python von Einheit 31.
- Die „Wurzel“ wurde von `BaseException` geformt.
- Alle Ausnahmeobjekte sind Instanzen dieser Klasse.
- Sie hat eine Subklasse `Exception`.
- Alle Instanzen von `Exception` sind `BaseExceptions`, aber nicht alle `BaseExceptions` sind `Exceptions`.
- `ArithmeticError` wiederum ist eine Subklasse von `Exception`, was heist, dass alle `Arithmetics` Instanzen von `Exceptions` sind – aber nicht andersherum.
- Wir könnten sagen, dass wenn eine `ClassA` eine Subklasse von einer Klasse `ClassB` ist, also wenn `issubclass(ClassA, ClassB)` gilt, dann ist `ClassA` sozusagen eine Teilmenge von `ClassB`.



Vererbung als mengentheoretisches Konzept

- Wir können eine Klasse als die Menge aller Objekte, die zu ihr gehören, betrachten.
- Subklassen sind dann Teilmengen dieser Menge.
- Erinnern wir uns an die baumartige Struktur der Built-In Ausnahmeklassen in Python von Einheit 31.
- Die „Wurzel“ wurde von `BaseException` geformt.
- Alle Ausnahmeobjekte sind Instanzen dieser Klasse.
- Sie hat eine Subklasse `Exception`.
- Alle Instanzen von `Exception` sind `BaseExceptions`, aber nicht alle `BaseExceptions` sind `Exceptions`.
- `ArithmetError` wiederum ist eine Subklasse von `Exception`, was heist, dass alle `ArithmetErrors` Instanzen von `Exceptions` sind – aber nicht andersherum.
- Wir könnten sagen, dass wenn eine `ClassA` eine Subklasse von einer Klasse `ClassB` ist, also wenn `issubclass(ClassA, ClassB)` gilt, dann ist `ClassA` sozusagen eine Teilmenge von `ClassB`.

$$\text{ClassA} \subseteq \text{ClassB} \Leftrightarrow \text{issubclass}(\text{ClassA}, \text{ClassB})$$

(1)

Veerbung als Spezialisierung

- Eine andere Perspektive ist das Konzept der Spezialisierung.





Veerbung als Spezialisierung

- Eine andere Perspektive ist das Konzept der Spezialisierung.
- In vernünftigen Designs sind Subklassen ja nicht einfach irgendwelche Teilmengen ihrer Basisklasse.



Veerbung als Spezialisierung

- Eine andere Perspektive ist das Konzept der Spezialisierung.
- In vernünftigen Designs sind Subklassen ja nicht einfach irgendwelche Teilmengen ihrer Basisklasse.
- Normalerweise definieren sie bestimmte Eigenschaften, die alle ihre Objektinstanzen haben aber andere Objekte eben nicht.



Veerbung als Spezialisierung

- Eine andere Perspektive ist das Konzept der Spezialisierung.
- In vernünftigen Designs sind Subklassen ja nicht einfach irgendwelche Teilmengen ihrer Basisklasse.
- Normalerweise definieren sie bestimmte Eigenschaften, die alle ihre Objektinstanzen haben aber andere Objekte eben nicht.
- Wenn `Tier` eine Klasse wäre, dann wäre `Vogel` eine Subklasse.



Veerbung als Spezialisierung

- Eine andere Perspektive ist das Konzept der Spezialisierung.
- In vernünftigen Designs sind Subklassen ja nicht einfach irgendwelche Teilmengen ihrer Basisklasse.
- Normalerweise definieren sie bestimmte Eigenschaften, die alle ihre Objektinstanzen haben aber andere Objekte eben nicht.
- Wenn `Tier` eine Klasse wäre, dann wäre `Vogel` eine Subklasse.
- Es wäre aber nicht irgendeine zufällige Gruppe von Tieren.



Veerbung als Spezialisierung

- Eine andere Perspektive ist das Konzept der Spezialisierung.
- In vernünftigen Designs sind Subklassen ja nicht einfach irgendwelche Teilmengen ihrer Basisklasse.
- Normalerweise definieren sie bestimmte Eigenschaften, die alle ihre Objektinstanzen haben aber andere Objekte eben nicht.
- Wenn `Tier` eine Klasse wäre, dann wäre `Vogel` eine Subklasse.
- Es wäre aber nicht irgendeine zufällige Gruppe von Tieren.
- Es würde die Tiere repräsentieren, die zwei Füße, Flügel, und Federn haben sowie die Eier legen.



Veerbung als Spezialisierung

- Eine andere Perspektive ist das Konzept der Spezialisierung.
- In vernünftigen Designs sind Subklassen ja nicht einfach irgendwelche Teilmengen ihrer Basisklasse.
- Normalerweise definieren sie bestimmte Eigenschaften, die alle ihre Objektinstanzen haben aber andere Objekte eben nicht.
- Wenn `Tier` eine Klasse wäre, dann wäre `Vogel` eine Subklasse.
- Es wäre aber nicht irgendeine zufällige Gruppe von Tieren.
- Es würde die Tiere repräsentieren, die zwei Füße, Flügel, und Federn haben sowie die Eier legen.
- Es könnte das Attribut `Gewicht` und die Methode `essen()` von seiner Basisklasse `Tier` erben.



Veerbung als Spezialisierung

- Eine andere Perspektive ist das Konzept der Spezialisierung.
- In vernünftigen Designs sind Subklassen ja nicht einfach irgendwelche Teilmengen ihrer Basisklasse.
- Normalerweise definieren sie bestimmte Eigenschaften, die alle ihre Objektinstanzen haben aber andere Objekte eben nicht.
- Wenn `Tier` eine Klasse wäre, dann wäre `Vogel` eine Subklasse.
- Es wäre aber nicht irgendeine zufällige Gruppe von Tieren.
- Es würde die Tiere repräsentieren, die zwei Füße, Flügel, und Federn haben sowie die Eier legen.
- Es könnte das Attribut `Gewicht` und die Methode `essen()` von seiner Basisklasse `Tier` erben.
- Es könnte die Methoden `laufen()` und `fliegen()` hinzufügen.



Veerbung als Spezialisierung

- Eine andere Perspektive ist das Konzept der Spezialisierung.
- In vernünftigen Designs sind Subklassen ja nicht einfach irgendwelche Teilmengen ihrer Basisklasse.
- Normalerweise definieren sie bestimmte Eigenschaften, die alle ihre Objektinstanzen haben aber andere Objekte eben nicht.
- Wenn `Tier` eine Klasse wäre, dann wäre `Vogel` eine Subklasse.
- Es wäre aber nicht irgendeine zufällige Gruppe von Tieren.
- Es würde die Tiere repräsentieren, die zwei Füße, Flügel, und Federn haben sowie die Eier legen.
- Es könnte das Attribut `Gewicht` und die Methode `essen()` von seiner Basisklasse `Tier` erben.
- Es könnte die Methoden `laufen()` und `fliegen()` hinzufügen.
- `Spatz` und `Strauß` wären dann Spezialfälle, also Subklassen, von `Vogel`, wobei jede wieder ihre eigenen Characteristiken mitbringt, die sie von anderen Vögeln unterscheidet.



Veerbung als Spezialisierung

- Eine andere Perspektive ist das Konzept der Spezialisierung.
- In vernünftigen Designs sind Subklassen ja nicht einfach irgendwelche Teilmengen ihrer Basisklasse.
- Normalerweise definieren sie bestimmte Eigenschaften, die alle ihre Objektinstanzen haben aber andere Objekte eben nicht.
- Wenn `Tier` eine Klasse wäre, dann wäre `Vogel` eine Subklasse.
- Es wäre aber nicht irgendeine zufällige Gruppe von Tieren.
- Es würde die Tiere repräsentieren, die zwei Füße, Flügel, und Federn haben sowie die Eier legen.
- Es könnte das Attribut `Gewicht` und die Methode `essen()` von seiner Basisklasse `Tier` erben.
- Es könnte die Methoden `laufen()` und `fliegen()` hinzufügen.
- `Spatz` und `Strauß` wären dann Spezialfälle, also Subklassen, von `Vogel`, wobei jede wieder ihre eigenen Characteristiken mitbringt, die sie von anderen Vögeln unterscheidet.
- So könnte `Strauß` z. B. die Methode `fliegen()` so überschreiben, dass sie einen `NotImplementedError` auslöst.

Vererbung als Grundlage für APIs

- Die Vererbung und das Überschreiben von Methoden macht Klassen besonders geeignet, APIs zu definieren und zu implementieren.



Vererbung als Grundlage für APIs



- Die Vererbung und das Überschreiben von Methoden macht Klassen besonders geeignet, APIs zu definieren und zu implementieren.
- Wir können eine Basisklasse entwerfen, die die Methoden für eine bestimmte Aufgabe als abstrakte Hüllen bereitstellt, ohne diese jedoch zu implementieren.

Vererbung als Grundlage für APIs



- Die Vererbung und das Überschreiben von Methoden macht Klassen besonders geeignet, APIs zu definieren und zu implementieren.
- Wir können eine Basisklasse entwerfen, die die Methoden für eine bestimmte Aufgabe als abstrakte Hülle bereitstellt, ohne diese jedoch zu implementieren.
- Sie könnten nichts machen oder einen `NotImplementedError` auslösen.



Vererbung als Grundlage für APIs

- Die Vererbung und das Überschreiben von Methoden macht Klassen besonders geeignet, APIs zu definieren und zu implementieren.
- Wir können eine Basisklasse entwerfen, die die Methoden für eine bestimmte Aufgabe als abstrakte Hülle bereitstellt, ohne diese jedoch zu implementieren.
- Sie könnten nichts machen oder einen `NotImplementedError` auslösen.
- Sie würden nur die API definieren.



Vererbung als Grundlage für APIs

- Die Vererbung und das Überschreiben von Methoden macht Klassen besonders geeignet, APIs zu definieren und zu implementieren.
- Wir können eine Basisklasse entwerfen, die die Methoden für eine bestimmte Aufgabe als abstrakte Hülle bereitstellt, ohne diese jedoch zu implementieren.
- Sie könnten nichts machen oder einen `NotImplementedError` auslösen.
- Sie würden nur die API definieren.
- In Subklassen können die Methoden dann mit vernünftigem Verhalten implementiert werden.



Vererbung als Grundlage für APIs

- Die Vererbung und das Überschreiben von Methoden macht Klassen besonders geeignet, APIs zu definieren und zu implementieren.
- Wir können eine Basisklasse entwerfen, die die Methoden für eine bestimmte Aufgabe als abstrakte Hülle bereitstellt, ohne diese jedoch zu implementieren.
- Sie könnten nichts machen oder einen `NotImplementedError` auslösen.
- Sie würden nur die API definieren.
- In Subklassen können die Methoden dann mit vernünftigem Verhalten implementiert werden.
- Wir können die API in verschiedenen Subklassen auf verschiedene Art implementieren.



Vererbung als Grundlage für APIs

- Die Vererbung und das Überschreiben von Methoden macht Klassen besonders geeignet, APIs zu definieren und zu implementieren.
- Wir können eine Basisklasse entwerfen, die die Methoden für eine bestimmte Aufgabe als abstrakte Hülle bereitstellt, ohne diese jedoch zu implementieren.
- Sie könnten nichts machen oder einen `NotImplementedError` auslösen.
- Sie würden nur die API definieren.
- In Subklassen können die Methoden dann mit vernünftigem Verhalten implementiert werden.
- Wir können die API in verschiedenen Subklassen auf verschiedene Art implementieren.
- Man könnte sich z. B. eine API zum rendern von Dokumenten vorstellen, die in einer Subklasse für PDF und in einer anderen Subklasse für SVG output implementiert wird.



Vererbung als Grundlage für APIs

- Die Vererbung und das Überschreiben von Methoden macht Klassen besonders geeignet, APIs zu definieren und zu implementieren.
- Wir können eine Basisklasse entwerfen, die die Methoden für eine bestimmte Aufgabe als abstrakte Hülle bereitstellt, ohne diese jedoch zu implementieren.
- Sie könnten nichts machen oder einen `NotImplementedError` auslösen.
- Sie würden nur die API definieren.
- In Subklassen können die Methoden dann mit vernünftigem Verhalten implementiert werden.
- Wir können die API in verschiedenen Subklassen auf verschiedene Art implementieren.
- Man könnte sich z. B. eine API zum rendern von Dokumenten vorstellen, die in einer Subklasse für PDF und in einer anderen Subklasse für SVG output implementiert wird.
- Ein Benutzer der API muss dann nur die Dokumentation und Methoden SignATUREN der Basisklasse verstehen.



Vererbung als Grundlage für APIs

- Die Vererbung und das Überschreiben von Methoden macht Klassen besonders geeignet, APIs zu definieren und zu implementieren.
- Wir können eine Basisklasse entwerfen, die die Methoden für eine bestimmte Aufgabe als abstrakte Hülle bereitstellt, ohne diese jedoch zu implementieren.
- Sie könnten nichts machen oder einen `NotImplementedError` auslösen.
- Sie würden nur die API definieren.
- In Subklassen können die Methoden dann mit vernünftigem Verhalten implementiert werden.
- Wir können die API in verschiedenen Subklassen auf verschiedene Art implementieren.
- Man könnte sich z. B. eine API zum rendern von Dokumenten vorstellen, die in einer Subklasse für PDF und in einer anderen Subklasse für SVG output implementiert wird.
- Ein Benutzer der API muss dann nur die Dokumentation und Methoden SignATUREN der Basisklasse verstehen.
- Er kann dann jede Subklasse benutzen, jenachdem, was für Output er will.



Beispiel: Formen in der Euklidischen Ebene



Beispiel: Formen in der Euklidischen Ebene



- Wir wollen alle geschlossenen Formen auf der zweidimensionalen Euklidischen Ebene repräsentieren können.

Beispiel: Formen in der Euklidischen Ebene



- Wir wollen alle geschlossenen Formen auf der zweidimensionalen Euklidischen Ebene repräsentieren können.
- Wir können eine Basisclass `Shape` zu diesem Zweck erstellen.

Shape

+area(): int | float
+perimeter(): int | float

Beispiel: Formen in der Euklidischen Ebene



- Wir wollen alle geschlossenen Formen auf der zweidimensionalen Euklidischen Ebene repräsentieren können.
- Wir können eine Basisclass `Shape` zu diesem Zweck erstellen.
- Jede geschlossene Form hat einen Flächeninhalt (EN: *area*) und einen Umfang (EN: *perimeter*).

`Shape`

+`area()`: int | float
+`perimeter()`: int | float

Beispiel: Formen in der Euklidischen Ebene



- Wir wollen alle geschlossenen Formen auf der zweidimensionalen Euklidischen Ebene repräsentieren können.
- Wir können eine Basisclass `Shape` zu diesem Zweck erstellen.
- Jede geschlossene Form hat einen Flächeninhalt (EN: `area`) und einen Umfang (EN: `perimeter`).
- Darum definiert `Shape` zwei Methoden, `area` und `perimeter`, die die Fläche in Flächeneinheiten und den Umfang in Längeneinheiten zurückliefern.

`Shape`

+`area()`: int | float
+`perimeter()`: int | float

Beispiel: Formen in der Euklidischen Ebene



- Wir können eine Basisclass `Shape` zu diesem Zweck erstellen.
- Jede geschlossene Form hat einen Flächeninhalt (EN: *area*) und einen Umfang (EN: *perimeter*).
- Darum definiert `Shape` zwei Methoden, `area` und `perimeter`, die die Fläche in Flächeneinheiten und den Umfang in Längeneinheiten zurückliefern.
- Es implementiert diese Methoden noch nicht, weil das Berechnen des Umfangs und der Fläche für verschiedene Arten von Formen anders funktioniert.

`Shape`

+`area()`: int | float
+`perimeter()`: int | float

Beispiel: Formen in der Euklidischen Ebene



- Jede geschlossene Form hat einen Flächeninhalt (EN: *area*) und einen Umfang (EN: *perimeter*).
- Darum definiert `Shape` zwei Methoden, `area` und `perimeter`, die die Fläche in Flächeneinheiten und den Umfang in Längeneinheiten zurückliefern.
- Es implementiert diese Methoden noch nicht, weil das Berechnen des Umfangs und der Fläche für verschiedene Arten von Formen anders funktioniert.
- In dem wir die Methoden aber definieren, machen wir klar, dass diese beiden Dinge für jede wirkliche Instanz von `Shape` berechnet werden kann.

`Shape`

+`area()`: int | float
+`perimeter()`: int | float

Beispiel: Die Klasse Shape

- In Datei `shape.py` erstellen wir diese neue Basisklasse `Shape`.

```
1 """
2 A base class for shapes defines a general interface for defining shapes.
3
4 It has no attributes, but two methods, `area` and `perimeter`, which
5 non-abstract subclasses must implement to return the area and perimeter
6 of the shapes they represent.
7
8 >>> from pytest import raises
9 >>> s = Shape()
10 >>> with raises(NotImplementedError):
11 ...     s.area()
12 >>> with raises(NotImplementedError):
13 ...     s.perimeter()
14 """
15
16
17 class Shape:
18     """A closed geometric shape has an area and a perimeter."""
19
20     def area(self) -> int | float:
21         """
22             Get the area of this shape.
23
24             :return: the area of this shape
25             :raises NotImplementedError: You must overwrite this method.
26         """
27         raise NotImplementedError # must be implemented by subclasses
28
29     def perimeter(self) -> int | float:
30         """
31             Get the perimeter of this shape.
32
33             :return: the perimeter of this shape
34             :raises NotImplementedError: You must overwrite this method.
35         """
36         raise NotImplementedError # must be implemented by subclasses
```

Beispiel: Die Klasse Shape

- In Datei `shape.py` erstellen wir diese neue Basisklasse `Shape`.
- Die Klasse ist nicht dafür da, um wirklich instantiiert zu werden.

```
1 """
2 A base class for shapes defines a general interface for defining shapes.
3
4 It has no attributes, but two methods, `area` and `perimeter`, which
5 non-abstract subclasses must implement to return the area and perimeter
6 of the shapes they represent.
7
8 >>> from pytest import raises
9 >>> s = Shape()
10 >>> with raises(NotImplementedError):
11 ...     s.area()
12 >>> with raises(NotImplementedError):
13 ...     s.perimeter()
14 """
15
16
17 class Shape:
18     """A closed geometric shape has an area and a perimeter."""
19
20     def area(self) -> int | float:
21         """
22             Get the area of this shape.
23
24             :return: the area of this shape
25             :raises NotImplementedError: You must overwrite this method.
26         """
27         raise NotImplementedError # must be implemented by subclasses
28
29     def perimeter(self) -> int | float:
30         """
31             Get the perimeter of this shape.
32
33             :return: the perimeter of this shape
34             :raises NotImplementedError: You must overwrite this method.
35         """
36         raise NotImplementedError # must be implemented by subclasses
```

Beispiel: Die Klasse Shape

- In Datei `shape.py` erstellen wir diese neue Basisklasse `Shape`.
- Die Klasse ist nicht dafür da, um wirklich instantiiert zu werden.
- Stattdessen wollen wir sie als Basisklasse für die Arten von 2D-Formen benutzen, die wir schon in der Grundschule gelernt haben.

```
1 """
2 A base class for shapes defines a general interface for defining shapes.
3
4 It has no attributes, but two methods, `area` and `perimeter`, which
5 non-abstract subclasses must implement to return the area and perimeter
6 of the shapes they represent.
7
8 >>> from pytest import raises
9 >>> s = Shape()
10 >>> with raises(NotImplementedError):
11 ...     s.area()
12 >>> with raises(NotImplementedError):
13 ...     s.perimeter()
14 """
15
16
17 class Shape:
18     """A closed geometric shape has an area and a perimeter."""
19
20     def area(self) -> int | float:
21         """
22             Get the area of this shape.
23
24             :return: the area of this shape
25             :raises NotImplementedError: You must overwrite this method.
26         """
27         raise NotImplementedError # must be implemented by subclasses
28
29     def perimeter(self) -> int | float:
30         """
31             Get the perimeter of this shape.
32
33             :return: the perimeter of this shape
34             :raises NotImplementedError: You must overwrite this method.
35         """
36         raise NotImplementedError # must be implemented by subclasses
```

Beispiel: Die Klasse Shape

- In Datei `shape.py` erstellen wir diese neue Basisklasse `Shape`.
- Die Klasse ist nicht dafür da, um wirklich instantiiert zu werden.
- Stattdessen wollen wir sie als Basisklasse für die Arten von 2D-Formen benutzen, die wir schon in der Grundschule gelernt haben.
- `Shape` hat keine Attribute.

```
1 """
2 A base class for shapes defines a general interface for defining shapes.
3
4 It has no attributes, but two methods, `area` and `perimeter`, which
5 non-abstract subclasses must implement to return the area and perimeter
6 of the shapes they represent.
7
8 >>> from pytest import raises
9 >>> s = Shape()
10 >>> with raises(NotImplementedError):
11 ...     s.area()
12 >>> with raises(NotImplementedError):
13 ...     s.perimeter()
14 """
15
16
17 class Shape:
18     """A closed geometric shape has an area and a perimeter."""
19
20     def area(self) -> int | float:
21         """
22             Get the area of this shape.
23
24             :return: the area of this shape
25             :raises NotImplementedError: You must overwrite this method.
26         """
27         raise NotImplementedError # must be implemented by subclasses
28
29     def perimeter(self) -> int | float:
30         """
31             Get the perimeter of this shape.
32
33             :return: the perimeter of this shape
34             :raises NotImplementedError: You must overwrite this method.
35         """
36         raise NotImplementedError # must be implemented by subclasses
```

Beispiel: Die Klasse Shape

- In Datei `shape.py` erstellen wir diese neue Basisklasse `Shape`.
- Die Klasse ist nicht dafür da, um wirklich instantiiert zu werden.
- Stattdessen wollen wir sie als Basisklasse für die Arten von 2D-Formen benutzen, die wir schon in der Grundschule gelernt haben.
- `Shape` hat keine Attribute.
- Aber, wie gesagt, hat diese Klasse die beiden Methoden `area` und `perimeter`.

```
1 """
2 A base class for shapes defines a general interface for defining shapes.
3
4 It has no attributes, but two methods, `area` and `perimeter`, which
5 non-abstract subclasses must implement to return the area and perimeter
6 of the shapes they represent.
7
8 >>> from pytest import raises
9 >>> s = Shape()
10 >>> with raises(NotImplementedError):
11 ...     s.area()
12 >>> with raises(NotImplementedError):
13 ...     s.perimeter()
14 """
15
16
17 class Shape:
18     """A closed geometric shape has an area and a perimeter."""
19
20     def area(self) -> int | float:
21         """
22             Get the area of this shape.
23
24             :return: the area of this shape
25             :raises NotImplementedError: You must overwrite this method.
26         """
27         raise NotImplementedError # must be implemented by subclasses
28
29     def perimeter(self) -> int | float:
30         """
31             Get the perimeter of this shape.
32
33             :return: the perimeter of this shape
34             :raises NotImplementedError: You must overwrite this method.
35         """
36         raise NotImplementedError # must be implemented by subclasses
```

Beispiel: Die Klasse Shape

- Die Klasse ist nicht dafür da, um wirklich instantiiert zu werden.
- Stattdessen wollen wir sie als Basisklasse für die Arten von 2D-Formen benutzen, die wir schon in der Grundschule gelernt haben.
- `Shape` hat keine Attribute.
- Aber, wie gesagt, hat diese Klasse die beiden Methoden `area` und `perimeter`.
- Natürlich können wir die Fläche und den Umfang von solchen abstrakten Formen nicht wirklich berechnen.

```
1 """
2 A base class for shapes defines a general interface for defining shapes.
3
4 It has no attributes, but two methods, `area` and `perimeter`, which
5 non-abstract subclasses must implement to return the area and perimeter
6 of the shapes they represent.
7
8 >>> from pytest import raises
9 >>> s = Shape()
10 >>> with raises(NotImplementedError):
11 ...     s.area()
12 >>> with raises(NotImplementedError):
13 ...     s.perimeter()
14 """
15
16
17 class Shape:
18     """A closed geometric shape has an area and a perimeter."""
19
20     def area(self) -> int | float:
21         """
22             Get the area of this shape.
23
24             :return: the area of this shape
25             :raises NotImplementedError: You must overwrite this method.
26         """
27         raise NotImplementedError # must be implemented by subclasses
28
29     def perimeter(self) -> int | float:
30         """
31             Get the perimeter of this shape.
32
33             :return: the perimeter of this shape
34             :raises NotImplementedError: You must overwrite this method.
35         """
36         raise NotImplementedError # must be implemented by subclasses
```

Beispiel: Die Klasse Shape

- Stattdessen wollen wir sie als Basisklasse für die Arten von 2D-Formen benutzen, die wir schon in der Grundschule gelernt haben.
- `Shape` hat keine Attribute.
- Aber, wie gesagt, hat diese Klasse die beiden Methoden `area` und `perimeter`.
- Natürlich können wir die Fläche und den Umfang von solchen abstrakten Formen nicht wirklich berechnen.
- Deshalb lösen beide Methoden einen `NotImplementedError` aus.

```
1 """
2 A base class for shapes defines a general interface for defining shapes.
3
4 It has no attributes, but two methods, `area` and `perimeter`, which
5 non-abstract subclasses must implement to return the area and perimeter
6 of the shapes they represent.
7
8 >>> from pytest import raises
9 >>> s = Shape()
10 >>> with raises(NotImplementedError):
11 ...     s.area()
12 >>> with raises(NotImplementedError):
13 ...     s.perimeter()
14 """
15
16
17 class Shape:
18     """A closed geometric shape has an area and a perimeter."""
19
20     def area(self) -> int | float:
21         """
22             Get the area of this shape.
23
24             :return: the area of this shape
25             :raises NotImplementedError: You must overwrite this method.
26         """
27         raise NotImplementedError # must be implemented by subclasses
28
29     def perimeter(self) -> int | float:
30         """
31             Get the perimeter of this shape.
32
33             :return: the perimeter of this shape
34             :raises NotImplementedError: You must overwrite this method.
35         """
36         raise NotImplementedError # must be implemented by subclasses
```

Beispiel: Die Klasse Shape

- Shape hat keine Attribute.
- Aber, wie gesagt, hat diese Klasse die beiden Methoden area und perimeter.
- Natürlich können wir die Fläche und den Umfang von solchen abstrakten Formen nicht wirklich berechnen.
- Deshalb lösen beide Methoden einen NotImplementedError aus.
- Wenn jemand unsere Klasse Shape tatsächlich instantiiieren würde, sagen wir, durch s = Shape() und dann versuchen würde, s.perimeter() zu machen, dann würde das fehlgeschlagen.

```
1 """
2 A base class for shapes defines a general interface for defining shapes.
3
4 It has no attributes, but two methods, `area` and `perimeter`, which
5 non-abstract subclasses must implement to return the area and perimeter
6 of the shapes they represent.
7
8 >>> from pytest import raises
9 >>> s = Shape()
10 >>> with raises(NotImplementedError):
11 ...     s.area()
12 >>> with raises(NotImplementedError):
13 ...     s.perimeter()
14 """
15
16
17 class Shape:
18     """A closed geometric shape has an area and a perimeter."""
19
20     def area(self) -> int | float:
21         """
22             Get the area of this shape.
23
24             :return: the area of this shape
25             :raises NotImplementedError: You must overwrite this method.
26         """
27         raise NotImplementedError # must be implemented by subclasses
28
29     def perimeter(self) -> int | float:
30         """
31             Get the perimeter of this shape.
32
33             :return: the perimeter of this shape
34             :raises NotImplementedError: You must overwrite this method.
35         """
36         raise NotImplementedError # must be implemented by subclasses
```

Beispiel: Die Klasse Shape

- Aber, wie gesagt, hat diese Klasse die beiden Methoden `area` und `perimeter`.
- Natürlich können wir die Fläche und den Umfang von solchen abstrakten Formen nicht wirklich berechnen.
- Deshalb lösen beide Methoden einen `NotImplementedError` aus.
- Wenn jemand unsere Klasse `Shape` tatsächlich instantiiieren würde, sagen wir, durch `s = Shape()` und dann versuchen würde, `s.perimeter()` zu machen, dann würde das fehlschlagen.
- Dieses Verhalten wird mit einem Doctest im Kopf des Moduls getestet.

```
1 """
2 A base class for shapes defines a general interface for defining shapes.
3
4 It has no attributes, but two methods, `area` and `perimeter`, which
5 non-abstract subclasses must implement to return the area and perimeter
6 of the shapes they represent.
7
8 >>> from pytest import raises
9 >>> s = Shape()
10 >>> with raises(NotImplementedError):
11 ...     s.area()
12 >>> with raises(NotImplementedError):
13 ...     s.perimeter()
14 """
15
16
17 class Shape:
18     """A closed geometric shape has an area and a perimeter."""
19
20     def area(self) -> int | float:
21         """
22             Get the area of this shape.
23
24             :return: the area of this shape
25             :raises NotImplementedError: You must overwrite this method.
26         """
27         raise NotImplementedError # must be implemented by subclasses
28
29     def perimeter(self) -> int | float:
30         """
31             Get the perimeter of this shape.
32
33             :return: the perimeter of this shape
34             :raises NotImplementedError: You must overwrite this method.
35         """
36         raise NotImplementedError # must be implemented by subclasses
```

Beispiel: Die Klasse Shape

- Natürlich können wir die Fläche und den Umfang von solchen abstrakten Formen nicht wirklich berechnen.
- Deshalb lösen beide Methoden einen `NotImplementedError` aus.
- Wenn jemand unsere Klasse `Shape` tatsächlich instantiiieren würde, sagen wir, durch `s = Shape()` und dann versuchen würde, `s.perimeter()` zu machen, dann würde das fehlschlagen.
- Dieses Verhalten wird mit einem Doctest im Kopf des Moduls getestet.
- Das Wichtige ist jedoch, dass die Methoden da sind.

```
1 """
2 A base class for shapes defines a general interface for defining shapes.
3
4 It has no attributes, but two methods, `area` and `perimeter`, which
5 non-abstract subclasses must implement to return the area and perimeter
6 of the shapes they represent.
7
8 >>> from pytest import raises
9 >>> s = Shape()
10 >>> with raises(NotImplementedError):
11 ...     s.area()
12 >>> with raises(NotImplementedError):
13 ...     s.perimeter()
14 """
15
16
17 class Shape:
18     """A closed geometric shape has an area and a perimeter."""
19
20     def area(self) -> int | float:
21         """
22             Get the area of this shape.
23
24             :return: the area of this shape
25             :raises NotImplementedError: You must overwrite this method.
26         """
27         raise NotImplementedError # must be implemented by subclasses
28
29     def perimeter(self) -> int | float:
30         """
31             Get the perimeter of this shape.
32
33             :return: the perimeter of this shape
34             :raises NotImplementedError: You must overwrite this method.
35         """
36         raise NotImplementedError # must be implemented by subclasses
```

Beispiel: Die Klasse Shape

- Deshalb lösen beide Methoden einen `NotImplementedError` aus.
- Wenn jemand unsere Klasse `Shape` tatsächlich instantiiieren würde, sagen wir, durch `s = Shape()` und dann versuchen würde, `s.perimeter()` zu machen, dann würde das fehlgeschlagen.
- Dieses Verhalten wird mit einem Doctest im Kopf des Moduls getestet.
- Das Wichtige ist jedoch, dass die Methoden da sind.
- Alle nicht-abstrakten abgeleitete Klassen müssen sie überschreiben und sinnvoll implementieren.

```
1 """
2 A base class for shapes defines a general interface for defining shapes.
3
4 It has no attributes, but two methods, `area` and `perimeter`, which
5 non-abstract subclasses must implement to return the area and perimeter
6 of the shapes they represent.
7
8 >>> from pytest import raises
9 >>> s = Shape()
10 >>> with raises(NotImplementedError):
11 ...     s.area()
12 >>> with raises(NotImplementedError):
13 ...     s.perimeter()
14 """
15
16
17 class Shape:
18     """A closed geometric shape has an area and a perimeter."""
19
20     def area(self) -> int | float:
21         """
22             Get the area of this shape.
23
24             :return: the area of this shape
25             :raises NotImplementedError: You must overwrite this method.
26         """
27         raise NotImplementedError # must be implemented by subclasses
28
29     def perimeter(self) -> int | float:
30         """
31             Get the perimeter of this shape.
32
33             :return: the perimeter of this shape
34             :raises NotImplementedError: You must overwrite this method.
35         """
36         raise NotImplementedError # must be implemented by subclasses
```

Beispiel: Die Klasse Shape

- Wenn jemand unsere Klasse `Shape` tatsächlich instantiiieren würde, sagen wir, durch `s = Shape()` und dann versuchen würde, `s.perimeter()` zu machen, dann würde das fehlschlagen.
- Dieses Verhalten wird mit einem Doctest im Kopf des Moduls getestet.
- Das Wichtige ist jedoch, dass die Methoden da sind.
- Alle nicht-abstrakten abgeleitete Klassen müssen sie überschreiben und sinnvoll implementieren.
- Der Benutzer kann also die Fläche und den Umfang von jeder echten Instanz von `Shape` auf genau die gleiche Art abfragen.

```
1 """
2 A base class for shapes defines a general interface for defining shapes.
3
4 It has no attributes, but two methods, `area` and `perimeter`, which
5 non-abstract subclasses must implement to return the area and perimeter
6 of the shapes they represent.
7
8 >>> from pytest import raises
9 >>> s = Shape()
10 >>> with raises(NotImplementedError):
11 ...     s.area()
12 >>> with raises(NotImplementedError):
13 ...     s.perimeter()
14 """
15
16
17 class Shape:
18     """A closed geometric shape has an area and a perimeter."""
19
20     def area(self) -> int | float:
21         """
22             Get the area of this shape.
23
24             :return: the area of this shape
25             :raises NotImplementedError: You must overwrite this method.
26         """
27         raise NotImplementedError # must be implemented by subclasses
28
29     def perimeter(self) -> int | float:
30         """
31             Get the perimeter of this shape.
32
33             :return: the perimeter of this shape
34             :raises NotImplementedError: You must overwrite this method.
35         """
36         raise NotImplementedError # must be implemented by subclasses
```

Beispiel: Kreise



- Ein Kreis ist eine besondere zwei-dimensionale Form.





Beispiel: Kreise

- Ein Kreis ist eine besondere zwei-dimensionale Form.
- Ein Kreis hat einen Mittelpunkt (EN: *center*) und einen Radius.



Shape
+area(): int float
+perimeter(): int float

Beispiel: Kreise



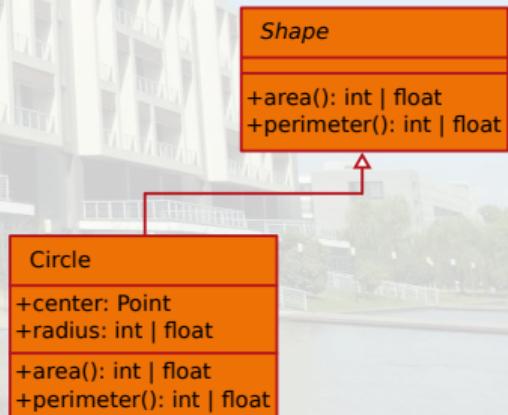
- Ein Kreis ist eine besondere zwei-dimensionale Form.
- Ein Kreis hat einen Mittelpunkt (EN: *center*) und einen Radius.
- Wenn wir diese Attribute kennen, dann könnten wir einen Kreis z. B. an seinem korrekten Ort malen oder seinen Flächeninhalt und Umfang berechnen.

Shape
+area(): int float
+perimeter(): int float



Beispiel: Kreise

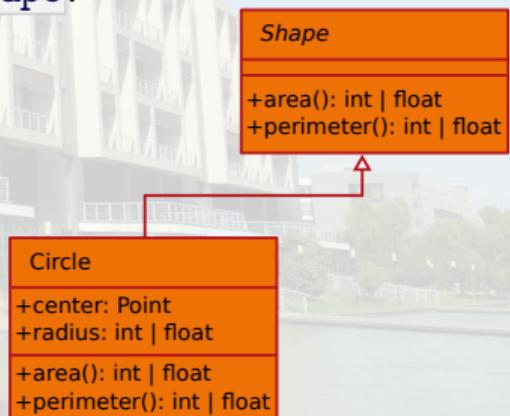
- Ein Kreis ist eine besondere zwei-dimensionale Form.
- Ein Kreis hat einen Mittelpunkt (EN: *center*) und einen Radius.
- Wenn wir diese Attribute kennen, dann könnten wir einen Kreis z. B. an seinem korrekten Ort malen oder seinen Flächeninhalt und Umfang berechnen.
- Wenn wir das als Klasse ausdrücken wollen, dann könnten die Klasse `Circle` als Subklasse der Klasse `Shape` definieren.





Beispiel: Kreise

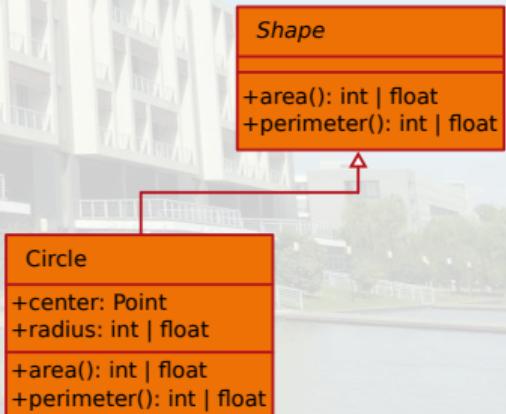
- Ein Kreis hat einen Mittelpunkt (EN: *center*) und einen Radius.
- Wenn wir diese Attribute kennen, dann könnten wir einen Kreis z. B. an seinem korrekten Ort malen oder seinen Flächeninhalt und Umfang berechnen.
- Wenn wir das als Klasse ausdrücken wollen, dann könnten die Klasse `Circle` als Subklasse der Klasse `Shape` definieren.
- Nicht alle `Shapes` sind `Circles`, aber jeder `Circle` ist eine `Shape`.





Beispiel: Kreise

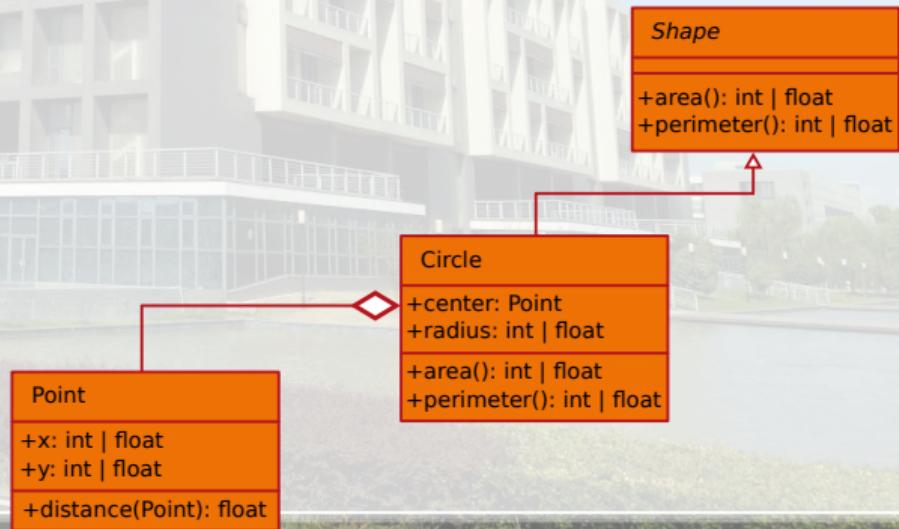
- Wenn wir diese Attribute kennen, dann könnten wir einen Kreis z. B. an seinem korrekten Ort malen oder seinen Flächeninhalt und Umfang berechnen.
- Wenn wir das als Klasse ausdrücken wollen, dann könnten die Klasse `Circle` als Subklasse der Klasse `Shape` definieren.
- Nicht alle `Shapes` sind `Circles`, aber jeder `Circle` ist eine `Shape`.
- Die Klasse `Circle` hätte zwei Attribute.





Beispiel: Kreise

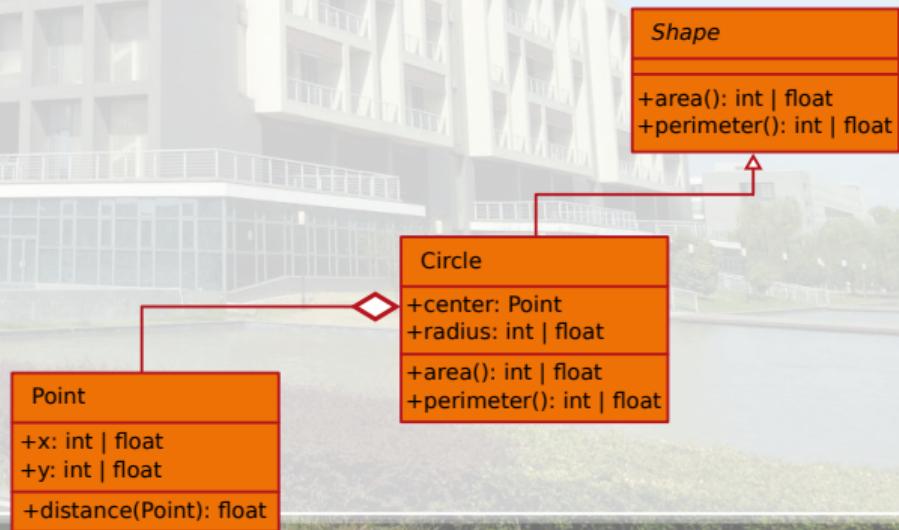
- Wenn wir das als Klasse ausdrücken wollen, dann könnten die Klasse `Circle` als Subklasse der Klasse `Shape` definieren.
- Nicht alle `Shapes` sind `Circles`, aber jeder `Circle` ist eine `Shape`.
- Die Klasse `Circle` hätte zwei Attribute.
- Das Attribut `center` könnte eine Instanz unserer allerersten Klasse `Point` sein.



Beispiel: Kreise



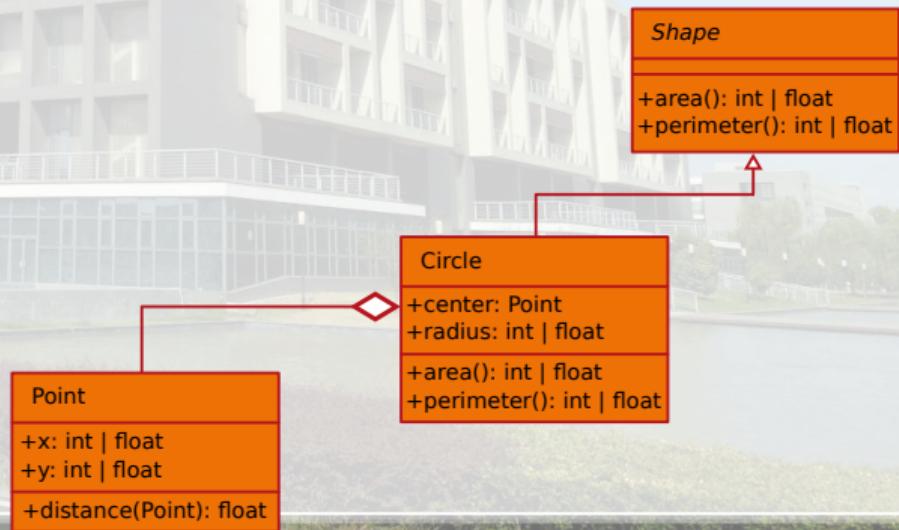
- Nicht alle `Shapes` sind `Circles`, aber jeder `Circle` ist eine `Shape`.
- Die Klasse `Circle` hätte zwei Attribute.
- Das Attribut `center` könnte eine Instanz unserer allerersten Klasse `Point` sein.
- Das Attribut `radius` könnte ein `int` oder `float` sein.





Beispiel: Kreise

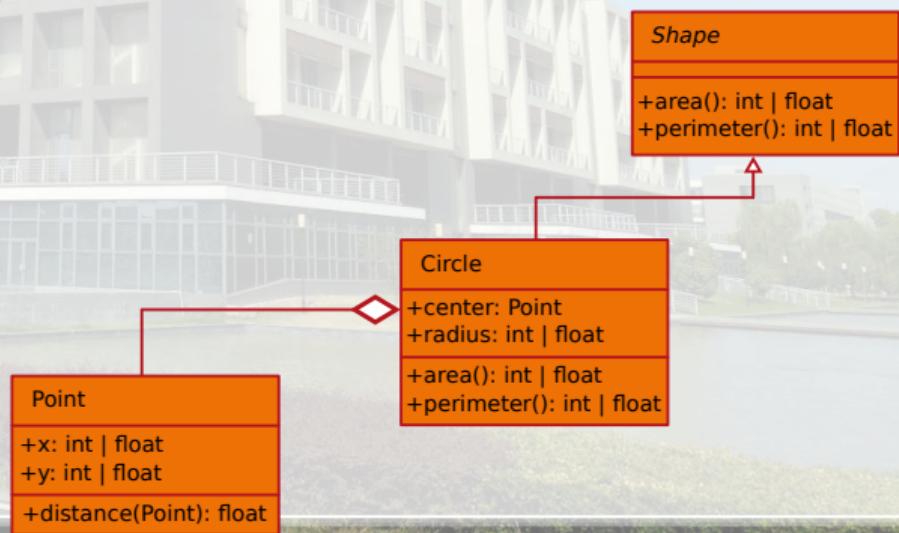
- Die Klasse `Circle` hätte zwei Attribute.
- Das Attribut `center` könnte eine Instanz unserer allerersten Klasse `Point` sein.
- Das Attribut `radius` könnte ein `int` oder `float` sein.
- Die Methoden `area` und `perimeter` können dann vernünftig implementiert werden.



Beispiel: Kreise



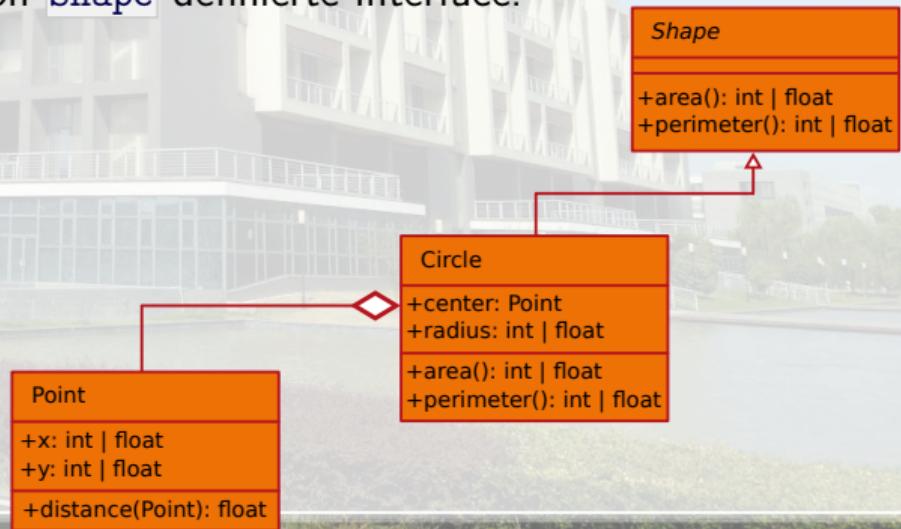
- Das Attribut `center` könnte eine Instanz unserer allerersten Klasse `Point` sein.
- Das Attribut `radius` könnte ein `int` oder `float` sein.
- Die Methoden `area` und `perimeter` können dann vernünftig implementiert werden.
- `Shape` selbst ist nutzlos, aber wir können verschiedene, spezialisierte Subklassen davon erzeugen, die die Methoden `area` und `perimeter` und implementieren.





Beispiel: Kreise

- Das Attribut `radius` könnte ein `int` oder `float` sein.
- Die Methoden `area` und `perimeter` können dann vernünftig implementiert werden.
- `Shape` selbst ist nutzlos, aber wir können verschiedene, spezialisierte Subklassen davon erzeugen, die die Methoden `area` und `perimeter` und implementieren.
- Der Nutzer dieser Klassen kann dann alle verschiedenen Subklassen genau gleich behandeln, denn alle unterstützen das von `Shape` definierte Interface.



Beispiel: Die Klasse Circle

- In Datei `circle.py` definieren wie die Klasse `Circle`.

```
1 """
2 A class for circles.
3
4 >>> c = Circle(Point(1, 2), 3)
5 >>> print(f"{c.center.x}, {c.center.y}, {c.radius}")
6 1, 2, 3
7 """
8
9 from math import isfinite, pi
10 from typing import Final
11
12 from point import Point
13 from shape import Shape
14
15
16 class Circle(Shape):
17     """A circle is a round shape with a center point and radius."""
18
19     def __init__(self, center: Point, radius: int | float) -> None:
20         """
21             Create the circle.
22
23             :param center: the center coordinate
24             :param radius: the radius
25
26         >>> try:
27             ...     Circle(Point(2, 3), -1)
28             ... except ValueError as ve:
29             ...     print(ve)
30             radius=-1 must be finite and >0.
31
32         if not (isfinite(radius) and (radius > 0)): # sanity check
33             raise ValueError(f"radius={radius} must be finite and >0.")
34         #: the center point of the circle
35         self.center: Final[Point] = center
36         #: the radius
37         self.radius: Final[int | float] = radius
38
39     def area(self) -> int | float:
40         """
41             Get the area of this cirle.
42
43             :return: the area of this cirle
44
45         >>> Circle(Point(3, 4), 10).area()
46         314.1592653589793
47
48         return pi * self.radius ** 2
49
50     def perimeter(self) -> int | float:
51         """
52             Get the perimeter of this cirle.
53
54             :return: the perimeter of this cirle
55
56         >>> Circle(Point(4, 1), 5).perimeter()
57         31.41592653589793
58
59         return 2 * pi * self.radius
```



Beispiel: Die Klasse Circle

- In Datei `circle.py` definieren wie die Klasse `Circle`.
- In dem wir schreiben `class Circle(Shape)`, deklarieren wir sie als Subklasse von `Shape`.

```
13 from shape import Shape
14
15
16 class Circle(Shape):
17     """A circle is a round shape with a center point and radius."""
18
19     def __init__(self, center: Point, radius: int | float) -> None:
20         """
21             Create the circle.
22
23             :param center: the center coordinate
24             :param radius: the radius
25
26             >>> try:
27                 ...     Circle(Point(2, 3), -1)
28                 ... except ValueError as ve:
29                     print(ve)
30             radius=-1 must be finite and >0.
31
32             if not (isfinite(radius) and (radius > 0)): # sanity check
33                 raise ValueError(f"radius={radius} must be finite and >0.")
34             #: the center point of the circle
35             self.center: Final[Point] = center
36             #: the radius
37             self.radius: Final[int | float] = radius
38
39     def area(self) -> int | float:
40         """
41             Get the area of this cirlce.
42
43             :return: the area of this cirlce
44
45             >>> Circle(Point(3, 4), 10).area()
46             314.1592653589793
47
48             """
49             return pi * self.radius ** 2
50
51     def perimeter(self) -> int | float:
52         """
53             Get the perimeter of this cirlce.
54
55             :return: the perimeter of this cirlce
56
57             >>> Circle(Point(4, 1), 5).perimeter()
58             31.41592653589793
59
60             """
61             return 2 * pi * self.radius
```

Beispiel: Die Klasse Circle

- In Datei `circle.py` definieren wie die Klasse `Circle`.
- In dem wir schreiben `class Circle(Shape)`, deklarieren wir sie als Subklasse von `Shape`.
- Durch ihren Initialisierer `__init__` liefern wir zwei Parameter.

```
13 from shape import Shape
14
15
16 class Circle(Shape):
17     """A circle is a round shape with a center point and radius."""
18
19     def __init__(self, center: Point, radius: int | float) -> None:
20         """
21             Create the circle.
22
23             :param center: the center coordinate
24             :param radius: the radius
25
26             >>> try:
27                 ...     Circle(Point(2, 3), -1)
28                 ... except ValueError as ve:
29                     print(ve)
30             radius=-1 must be finite and >0.
31
32             if not (isfinite(radius) and (radius > 0)): # sanity check
33                 raise ValueError(f"radius={radius} must be finite and >0.")
34             #: the center point of the circle
35             self.center: Final[Point] = center
36             #: the radius
37             self.radius: Final[int | float] = radius
38
39
40
41     def area(self) -> int | float:
42         """
43             Get the area of this cirlce.
44
45             :return: the area of this cirlce
46
47             >>> Circle(Point(3, 4), 10).area()
48             314.1592653589793
49
50             """
51             return pi * self.radius ** 2
52
53
54     def perimeter(self) -> int | float:
55         """
56             Get the perimeter of this cirlce.
57
58             :return: the perimeter of this cirlce
59
60             >>> Circle(Point(4, 1), 5).perimeter()
61             31.41592653589793
62
63             """
64             return 2 * pi * self.radius
```

Beispiel: Die Klasse Circle

- In Datei `circle.py` definieren wie die Klasse `Circle`.
- In dem wir schreiben `class Circle(Shape)`, deklarieren wir sie als Subklasse von `Shape`.
- Durch ihren Initialisierer `__init__` liefern wir zwei Parameter.
- Der Erste ist `center`, eine Instanz unserer Klasse `Point`.

```
13 from shape import Shape
14
15
16 class Circle(Shape):
17     """A circle is a round shape with a center point and radius."""
18
19     def __init__(self, center: Point, radius: int | float) -> None:
20         """
21             Create the circle.
22
23             :param center: the center coordinate
24             :param radius: the radius
25
26             >>> try:
27                 ...     Circle(Point(2, 3), -1)
28                 ... except ValueError as ve:
29                 ...     print(ve)
30                 radius=-1 must be finite and >0.
31             """
32             if not (isfinite(radius) and (radius > 0)): # sanity check
33                 raise ValueError(f"radius={radius} must be finite and >0.")
34             #: the center point of the circle
35             self.center: Final[Point] = center
36             #: the radius
37             self.radius: Final[int | float] = radius
38
39     def area(self) -> int | float:
40         """
41             Get the area of this cirlce.
42
43             :return: the area of this cirlce
44
45             >>> Circle(Point(3, 4), 10).area()
46             314.1592653589793
47             """
48             return pi * self.radius ** 2
49
50     def perimeter(self) -> int | float:
51         """
52             Get the perimeter of this cirlce.
53
54             :return: the perimeter of this cirlce
55
56             >>> Circle(Point(4, 1), 5).perimeter()
57             31.41592653589793
58             """
59             return 2 * pi * self.radius
```

Beispiel: Die Klasse Circle

- In Datei `circle.py` definieren wie die Klasse `Circle`.
- In dem wir schreiben `class Circle(Shape)`, deklarieren wir sie als Subklasse von `Shape`.
- Durch ihren Initialisierer `__init__` liefern wir zwei Parameter.
- Der Erste ist `center`, eine Instanz unserer Klasse `Point`.
- Der zweite ist `radius`, der entweder ein `int` oder ein `float` seien kann.

```
13 from shape import Shape
14
15
16 class Circle(Shape):
17     """A circle is a round shape with a center point and radius."""
18
19     def __init__(self, center: Point, radius: int | float) -> None:
20         """
21             Create the circle.
22
23             :param center: the center coordinate
24             :param radius: the radius
25
26             >>> try:
27                 ...     Circle(Point(2, 3), -1)
28                 ... except ValueError as ve:
29                 ...     print(ve)
30                 radius=-1 must be finite and >0.
31             """
32             if not (isfinite(radius) and (radius > 0)): # sanity check
33                 raise ValueError(f"radius={radius} must be finite and >0.")
34             #: the center point of the circle
35             self.center: Final[Point] = center
36             #: the radius
37             self.radius: Final[int | float] = radius
38
39     def area(self) -> int | float:
40         """
41             Get the area of this cirlce.
42
43             :return: the area of this cirlce
44
45             >>> Circle(Point(3, 4), 10).area()
46             314.1592653589793
47             """
48             return pi * self.radius ** 2
49
50     def perimeter(self) -> int | float:
51         """
52             Get the perimeter of this cirlce.
53
54             :return: the perimeter of this cirlce
55
56             >>> Circle(Point(4, 1), 5).perimeter()
57             31.41592653589793
58             """
59             return 2 * pi * self.radius
```

Beispiel: Die Klasse Circle

- In dem wir schreiben `class Circle(Shape)`, deklarieren wir sie als Subklasse von `Shape`.
- Durch ihren Initialisierer `__init__` liefern wir zwei Parameter.
- Der Erste ist `center`, eine Instanz unserer Klasse `Point`.
- Der zweite ist `radius`, der entweder ein `int` oder ein `float` seien kann.
- Der Initialisierer prüft erst, ob `radius` eine endliche positive Zahl ist und löst andernfalls einen `ValueError` aus.

```
13 from shape import Shape
14
15
16 class Circle(Shape):
17     """A circle is a round shape with a center point and radius."""
18
19     def __init__(self, center: Point, radius: int | float) -> None:
20         """
21             Create the circle.
22
23             :param center: the center coordinate
24             :param radius: the radius
25
26         >>> try:
27             ...     Circle(Point(2, 3), -1)
28             ... except ValueError as ve:
29             ...     print(ve)
30             radius=-1 must be finite and >0.
31         """
32
33         if not (isfinite(radius) and (radius > 0)): # sanity check
34             raise ValueError(f"radius={radius} must be finite and >0.")
35         #: the center point of the circle
36         self.center: Final[Point] = center
37         #: the radius
38         self.radius: Final[int | float] = radius
39
40     def area(self) -> int | float:
41         """
42             Get the area of this cirlce.
43
44             :return: the area of this cirlce
45
46         >>> Circle(Point(3, 4), 10).area()
47         314.1592653589793
48
49         return pi * self.radius ** 2
50
51     def perimeter(self) -> int | float:
52         """
53             Get the perimeter of this cirlce.
54
55             :return: the perimeter of this cirlce
56
57         >>> Circle(Point(4, 1), 5).perimeter()
58         31.41592653589793
59
60         return 2 * pi * self.radius
```

Beispiel: Die Klasse Circle

- Durch ihren Initialisierer `__init__` liefern wir zwei Parameter.
- Der Erste ist `center`, eine Instanz unserer Klasse `Point`.
- Der zweite ist `radius`, der entweder ein `int` oder ein `float` seien kann.
- Der Initialisierer prüft erst, ob `radius` eine endliche positive Zahl ist und löst andernfalls einen `ValueError` aus.
- Der Initialisierer der Klasse `Point` prüft das ja, das beide Koordinaten endliche Zahlen sind, also müssen wir das nicht mehr testen.

```
13 from shape import Shape
14
15
16 class Circle(Shape):
17     """A circle is a round shape with a center point and radius."""
18
19     def __init__(self, center: Point, radius: int | float) -> None:
20         """
21             Create the circle.
22
23             :param center: the center coordinate
24             :param radius: the radius
25
26             >>> try:
27                 ...     Circle(Point(2, 3), -1)
28                 ... except ValueError as ve:
29                     print(ve)
30             radius=-1 must be finite and >0.
31
32             if not (isfinite(radius) and (radius > 0)): # sanity check
33                 raise ValueError(f"radius={radius} must be finite and >0.")
34             #: the center point of the circle
35             self.center: Final[Point] = center
36             #: the radius
37             self.radius: Final[int | float] = radius
38
39
40
41     def area(self) -> int | float:
42         """
43             Get the area of this cirlce.
44
45             :return: the area of this cirlce
46
47             >>> Circle(Point(3, 4), 10).area()
48             314.1592653589793
49
50             """
51             return pi * self.radius ** 2
52
53
54     def perimeter(self) -> int | float:
55         """
56             Get the perimeter of this cirlce.
57
58             :return: the perimeter of this cirlce
59
60             >>> Circle(Point(4, 1), 5).perimeter()
61             31.41592653589793
62
63             """
64             return 2 * pi * self.radius
```

Beispiel: Die Klasse Circle

- Der Erste ist `center`, eine Instanz unserer Klasse `Point`.
- Der zweite ist `radius`, der entweder ein `int` oder ein `float` seien kann.
- Der Initialisierer prüft erst, ob `radius` eine endliche positive Zahl ist und löst andernfalls einen `ValueError` aus.
- Der Initialisierer der Klasse `Point` prüft das ja, das beide Koordinaten endliche Zahlen sind, also müssen wir das nicht mehr testen.
- Wir haben also sichergestellt, dass wir nur gültige Kreise erstellen.

```
13 from shape import Shape
14
15
16 class Circle(Shape):
17     """A circle is a round shape with a center point and radius."""
18
19     def __init__(self, center: Point, radius: int | float) -> None:
20         """
21             Create the circle.
22
23             :param center: the center coordinate
24             :param radius: the radius
25
26         >>> try:
27             ...     Circle(Point(2, 3), -1)
28             ... except ValueError as ve:
29             ...     print(ve)
30             radius=-1 must be finite and >0.
31         """
32         if not (isfinite(radius) and (radius > 0)): # sanity check
33             raise ValueError(f"radius={radius} must be finite and >0.")
34         #: the center point of the circle
35         self.center: Final[Point] = center
36         #: the radius
37         self.radius: Final[int | float] = radius
38
39     def area(self) -> int | float:
40         """
41             Get the area of this cirlce.
42
43             :return: the area of this cirlce
44
45         >>> Circle(Point(3, 4), 10).area()
46         314.1592653589793
47         """
48         return pi * self.radius ** 2
49
50     def perimeter(self) -> int | float:
51         """
52             Get the perimeter of this cirlce.
53
54             :return: the perimeter of this cirlce
55
56         >>> Circle(Point(4, 1), 5).perimeter()
57         31.41592653589793
58         """
59         return 2 * pi * self.radius
```

Beispiel: Die Klasse Circle

- Der zweite ist `radius`, der entweder ein `int` oder ein `float` seien kann.
- Der Initialisierer prüft erst, ob `radius` eine endliche positive Zahl ist und löst andernfalls einen `ValueError` aus.
- Der Initialisierer der Klasse `Point` prüft das ja, das beide Koordinaten endliche Zahlen sind, also müssen wir das nicht mehr testen.
- Wir haben also sichergestellt, dass wir nur gültige Kreise erstellen.
- Wir speichern `center` und `radius` in zwei Attributen mit den gleichen Namen.

```
13 from shape import Shape
14
15
16 class Circle(Shape):
17     """A circle is a round shape with a center point and radius."""
18
19     def __init__(self, center: Point, radius: int | float) -> None:
20         """
21             Create the circle.
22
23             :param center: the center coordinate
24             :param radius: the radius
25
26         >>> try:
27             ...     Circle(Point(2, 3), -1)
28             ... except ValueError as ve:
29             ...     print(ve)
30             radius=-1 must be finite and >0.
31         """
32
33         if not (isfinite(radius) and (radius > 0)): # sanity check
34             raise ValueError(f"radius={radius} must be finite and >0.")
35         #: the center point of the circle
36         self.center: Final[Point] = center
37         #: the radius
38         self.radius: Final[int | float] = radius
39
40
41     def area(self) -> int | float:
42         """
43             Get the area of this cirlce.
44
45             :return: the area of this cirlce
46
47         >>> Circle(Point(3, 4), 10).area()
48         314.1592653589793
49         """
50
51         return pi * self.radius ** 2
52
53     def perimeter(self) -> int | float:
54         """
55             Get the perimeter of this cirlce.
56
57             :return: the perimeter of this cirlce
58
59         >>> Circle(Point(4, 1), 5).perimeter()
60         31.41592653589793
61         """
62
63         return 2 * pi * self.radius
```

Beispiel: Die Klasse Circle

- Der Initialisierer prüft erst, ob `radius` eine endliche positive Zahl ist und löst andernfalls einen `ValueError` aus.
- Der Initialisierer der Klasse `Point` prüft das ja, das beide Koordinaten endliche Zahlen sind, also müssen wir das nicht mehr testen.
- Wir haben also sichergestellt, dass wir nur gültige Kreise erstellen.
- Wir speichern `center` und `radius` in zwei Attributen mit den gleichen Namen.
- Wir annotieren diese mit dem Type Hint `Final`, was bedeutet, dass sie nicht verändert werden sollen.

```
13 from shape import Shape
14
15
16 class Circle(Shape):
17     """A circle is a round shape with a center point and radius."""
18
19     def __init__(self, center: Point, radius: int | float) -> None:
20         """
21             Create the circle.
22
23             :param center: the center coordinate
24             :param radius: the radius
25
26         >>> try:
27             ...     Circle(Point(2, 3), -1)
28             ... except ValueError as ve:
29             ...     print(ve)
30             radius=-1 must be finite and >0.
31         """
32
33         if not (isfinite(radius) and (radius > 0)): # sanity check
34             raise ValueError(f"radius={radius} must be finite and >0.")
35         #: the center point of the circle
36         self.center: Final[Point] = center
37         #: the radius
38         self.radius: Final[int | float] = radius
39
40     def area(self) -> int | float:
41         """
42             Get the area of this cirlce.
43
44             :return: the area of this cirlce
45
46         >>> Circle(Point(3, 4), 10).area()
47         314.1592653589793
48
49         return pi * self.radius ** 2
50
51     def perimeter(self) -> int | float:
52         """
53             Get the perimeter of this cirlce.
54
55             :return: the perimeter of this cirlce
56
57         >>> Circle(Point(4, 1), 5).perimeter()
58         31.41592653589793
59
60         return 2 * pi * self.radius
```

Beispiel: Die Klasse Circle

- Der Initialisierer der Klasse `Point` prüft das ja, das beide Koordinaten endliche Zahlen sind, also müssen wir das nicht mehr testen.
- Wir haben also sichergestellt, dass wir nur gültige Kreise erstellen.
- Wir speichern `center` und `radius` in zwei Attributen mit den gleichen Namen.
- Wir annotieren diese mit dem Type Hint `Final`, was bedeutet, dass sie nicht verändert werden sollen.
- Wir erstellen unsere Klasse also nach dem Prinzip der Unveränderlichkeit.

```
13 from shape import Shape
14
15
16 class Circle(Shape):
17     """A circle is a round shape with a center point and radius."""
18
19     def __init__(self, center: Point, radius: int | float) -> None:
20         """
21             Create the circle.
22
23             :param center: the center coordinate
24             :param radius: the radius
25
26         >>> try:
27             ...     Circle(Point(2, 3), -1)
28             ... except ValueError as ve:
29             ...     print(ve)
30             radius=-1 must be finite and >0.
31         """
32
33         if not (isfinite(radius) and (radius > 0)): # sanity check
34             raise ValueError(f"radius={radius} must be finite and >0.")
35
36         #: the center point of the circle
37         self.center: Final[Point] = center
38
39         #: the radius
40         self.radius: Final[int | float] = radius
41
42
43     def area(self) -> int | float:
44         """
45             Get the area of this cirlce.
46
47             :return: the area of this cirlce
48
49         >>> Circle(Point(3, 4), 10).area()
50         314.1592653589793
51
52         """
53         return pi * self.radius ** 2
54
55     def perimeter(self) -> int | float:
56         """
57             Get the perimeter of this cirlce.
58
59             :return: the perimeter of this cirlce
60
61         >>> Circle(Point(4, 1), 5).perimeter()
62         31.41592653589793
63
64         """
65         return 2 * pi * self.radius
```

Beispiel: Die Klasse Circle

- Wir haben also sichergestellt, dass wir nur gültige Kreise erstellen.
- Wir speichern `center` und `radius` in zwei Attributen mit den gleichen Namen.
- Wir annotieren diese mit dem Type Hint `Final`, was bedeutet, dass sie nicht verändert werden sollen.
- Wir erstellen unsere Klasse also nach dem Prinzip der Unveränderlichkeit.
- Wir können nun die Methode `area` so implementieren, dass sie πradius^2 liefert.

```
13 from shape import Shape
14
15
16 class Circle(Shape):
17     """A circle is a round shape with a center point and radius."""
18
19     def __init__(self, center: Point, radius: int | float) -> None:
20         """
21             Create the circle.
22
23             :param center: the center coordinate
24             :param radius: the radius
25
26         >>> try:
27             ...     Circle(Point(2, 3), -1)
28             ... except ValueError as ve:
29             ...     print(ve)
30             radius=-1 must be finite and >0.
31         """
32
33         if not (isfinite(radius) and (radius > 0)): # sanity check
34             raise ValueError(f"radius={radius} must be finite and >0.")
35
36         #: the center point of the circle
37         self.center: Final[Point] = center
38
39         #: the radius
40         self.radius: Final[int | float] = radius
41
42     def area(self) -> int | float:
43         """
44             Get the area of this cirlce.
45
46             :return: the area of this cirlce
47
48         >>> Circle(Point(3, 4), 10).area()
49         314.1592653589793
50         """
51
52         return pi * self.radius ** 2
53
54     def perimeter(self) -> int | float:
55         """
56             Get the perimeter of this cirlce.
57
58             :return: the perimeter of this cirlce
59
60         >>> Circle(Point(4, 1), 5).perimeter()
61         31.41592653589793
62         """
63
64         return 2 * pi * self.radius
```

Beispiel: Die Klasse Circle

- Wir speichern `center` und `radius` in zwei Attributen mit den gleichen Namen.
- Wir annotieren diese mit dem Type Hint `Final`, was bedeutet, dass sie nicht verändert werden sollen.
- Wir erstellen unsere Klasse also nach dem Prinzip der Unveränderlichkeit.
- Wir können nun die Methode `area` so implementieren, dass sie πradius^2 liefert.
- Die Methode `perimeter` liefert $2\pi \text{radius}$.

```
13 from shape import Shape
14
15
16 class Circle(Shape):
17     """A circle is a round shape with a center point and radius."""
18
19     def __init__(self, center: Point, radius: int | float) -> None:
20         """
21             Create the circle.
22
23             :param center: the center coordinate
24             :param radius: the radius
25
26             >>> try:
27                 ...     Circle(Point(2, 3), -1)
28                 ... except ValueError as ve:
29                 ...     print(ve)
30                 radius=-1 must be finite and >0.
31             """
32             if not (isfinite(radius) and (radius > 0)): # sanity check
33                 raise ValueError(f"radius={radius} must be finite and >0.")
34             #: the center point of the circle
35             self.center: Final[Point] = center
36             #: the radius
37             self.radius: Final[int | float] = radius
38
39
40     def area(self) -> int | float:
41         """
42             Get the area of this cirlce.
43
44             :return: the area of this cirlce
45
46             >>> Circle(Point(3, 4), 10).area()
47             314.1592653589793
48             """
49             return pi * self.radius ** 2
50
51     def perimeter(self) -> int | float:
52         """
53             Get the perimeter of this cirlce.
54
55             :return: the perimeter of this cirlce
56
57             >>> Circle(Point(4, 1), 5).perimeter()
58             31.41592653589793
59             """
60             return 2 * pi * self.radius
```

Beispiel: Die Klasse Circle

- Wir annotieren diese mit dem Type Hint `Final`, was bedeutet, dass sie nicht verändert werden sollen.
- Wir erstellen unsere Klasse also nach dem Prinzip der Unveränderlichkeit.
- Wir können nun die Methode `area` so implementieren, dass sie πradius^2 liefert.
- Die Methode `perimeter` liefert $2\pi \text{radius}$.
- Damit haben wir das ganze, von der Basisklasse `Shape` definierte, Interface mit Bedeutung gefüllt.

```
13 from shape import Shape
14
15
16 class Circle(Shape):
17     """A circle is a round shape with a center point and radius."""
18
19     def __init__(self, center: Point, radius: int | float) -> None:
20         """
21             Create the circle.
22
23             :param center: the center coordinate
24             :param radius: the radius
25
26             >>> try:
27                 ...     Circle(Point(2, 3), -1)
28                 ... except ValueError as ve:
29                     print(ve)
30             radius=-1 must be finite and >0.
31
32             if not (isfinite(radius) and (radius > 0)): # sanity check
33                 raise ValueError(f"radius={radius} must be finite and >0.")
34             #: the center point of the circle
35             self.center: Final[Point] = center
36             #: the radius
37             self.radius: Final[int | float] = radius
38
39
40     def area(self) -> int | float:
41         """
42             Get the area of this cirlce.
43
44             :return: the area of this cirlce
45
46             >>> Circle(Point(3, 4), 10).area()
47             314.1592653589793
48
49             return pi * self.radius ** 2
50
51
52     def perimeter(self) -> int | float:
53         """
54             Get the perimeter of this cirlce.
55
56             :return: the perimeter of this cirlce
57
58             >>> Circle(Point(4, 1), 5).perimeter()
59             31.41592653589793
60
61             return 2 * pi * self.radius
```

Beispiel: Die Klasse Circle

- Wir erstellen unsere Klasse also nach dem Prinzip der Unveränderlichkeit.
- Wir können nun die Methode `area` so implementieren, dass sie πradius^2 liefert.
- Die Methode `perimeter` liefert $2\pi \text{radius}$.
- Damit haben wir das ganze, von der Basisklasse `Shape` definierte, Interface mit Bedeutung gefüllt.
- Wir fügen natürlich umfassende Doctests in unsere Datei ein, damit wir sofort testen können, dass alles so funktioniert, wie es soll.

```
13 from shape import Shape
14
15
16 class Circle(Shape):
17     """A circle is a round shape with a center point and radius."""
18
19     def __init__(self, center: Point, radius: int | float) -> None:
20         """
21             Create the circle.
22
23             :param center: the center coordinate
24             :param radius: the radius
25
26         >>> try:
27             ...     Circle(Point(2, 3), -1)
28             ... except ValueError as ve:
29             ...     print(ve)
30             radius=-1 must be finite and >0.
31         """
32
33         if not (isfinite(radius) and (radius > 0)): # sanity check
34             raise ValueError(f"radius={radius} must be finite and >0.")
35
36         #: the center point of the circle
37         self.center: Final[Point] = center
38
39         #: the radius
40         self.radius: Final[int | float] = radius
41
42
43     def area(self) -> int | float:
44         """
45             Get the area of this cirlce.
46
47             :return: the area of this cirlce
48
49         >>> Circle(Point(3, 4), 10).area()
50         314.1592653589793
51
52         """
53         return pi * self.radius ** 2
54
55     def perimeter(self) -> int | float:
56         """
57             Get the perimeter of this cirlce.
58
59             :return: the perimeter of this cirlce
60
61         >>> Circle(Point(4, 1), 5).perimeter()
62         31.41592653589793
63
64         """
65         return 2 * pi * self.radius
```

Beispiel: Die Klasse Circle benutzen

- Programm `circle_user.py` erforschen wir diese neue Klasse etwas.



Beispiel: Die Klasse Circle benutzen



- Programm `circle_user.py` erforschen wir diese neue Klasse etwas.
- Wir erstellen die Instance `circ` of `Circle` und geben `point=Point(2, 3)` und `radius=4` an.

```
1 """Examples for using our class :class:`Circle`."""
2
3 from circle import Circle # Our new class `Circle`.
4 from point import Point # Our very first class ever: `Point`.
5 from shape import Shape # Our base class `Shape`.
6
7 circ: Circle = Circle(Point(2, 3), 5) # Create a new circle instance.
8 print(f" center: {circ.center.x}, {circ.center.y}")
9 print(f" radius: {circ.radius}")
10 print(f" perimeter: {circ.perimeter()}")
11 print(f" area: {circ.area()}")
12
13 # Do some instance tests.
14 print(f" isinstance(circ, Circle): {isinstance(circ, Circle)}")
15 print(f" isinstance(circ, Shape): {isinstance(circ, Shape)}")
16 print(f" isinstance(circ, object): {isinstance(circ, object)}")
17
18 # Explore the class hierarchy.
19 print(f" issubclass(Circle, Shape): {issubclass(Circle, Shape)}")
20 print(f" issubclass(Shape, Circle): {issubclass(Shape, Circle)}")
21 print(f" issubclass(Shape, object): {issubclass(Shape, object)}")
22 print(f" issubclass(Circle, object): {issubclass(Circle, object)})")
```

↓ python3 circle_user.py ↓

```
1 center: (2, 3)
2 radius: 5
3 perimeter: 31.41592653589793
4 area: 78.53981633974483
5 isinstance(circ, Circle): True
6 isinstance(circ, Shape): True
7 isinstance(circ, object): True
8 issubclass(Circle, Shape): True
9 issubclass(Shape, Circle): False
10 issubclass(Shape, object): True
11 issubclass(Circle, object): True
```

Beispiel: Die Klasse Circle benutzen



- Programm `circle_user.py` erforschen wir diese neue Klasse etwas.
- Wir erstellen die Instance `circ` of `Circle` und geben `point=Point(2, 3)` und `radius=4` an.
- Wir stellen fest, dass die Parameter ordentlich in den Attributen gespeichert werden.

```
1 """Examples for using our class :class:`Circle`."""
2
3 from circle import Circle # Our new class `Circle`.
4 from point import Point # Our very first class ever: `Point`.
5 from shape import Shape # Our base class `Shape`.
6
7 circ: Circle = Circle(Point(2, 3), 5) # Create a new circle instance.
8 print(f" center: {circ.center.x}, {circ.center.y}")
9 print(f" radius: {circ.radius}")
10 print(f" perimeter: {circ.perimeter()}")
11 print(f" area: {circ.area()}")
12
13 # Do some instance tests.
14 print(f" isinstance(circ, Circle): {isinstance(circ, Circle)}")
15 print(f" isinstance(circ, Shape): {isinstance(circ, Shape)}")
16 print(f" isinstance(circ, object): {isinstance(circ, object)}")
17
18 # Explore the class hierarchy.
19 print(f" issubclass(Circle, Shape): {issubclass(Circle, Shape)}")
20 print(f" issubclass(Shape, Circle): {issubclass(Shape, Circle)}")
21 print(f" issubclass(Shape, object): {issubclass(Shape, object)}")
22 print(f" issubclass(Circle, object): {issubclass(Circle, object)})")
```

↓ python3 circle_user.py ↓

```
1 center: (2, 3)
2 radius: 5
3 perimeter: 31.41592653589793
4 area: 78.53981633974483
5 isinstance(circ, Circle): True
6 isinstance(circ, Shape): True
7 isinstance(circ, object): True
8 issubclass(Circle, Shape): True
9 issubclass(Shape, Circle): False
10 issubclass(Shape, object): True
11 issubclass(Circle, object): True
```

Beispiel: Die Klasse Circle benutzen



- Programm `circle_user.py` erforschen wir diese neue Klasse etwas.
- Wir erstellen die Instance `circ` of `Circle` und geben `point=Point(2, 3)` und `radius=4` an.
- Wir stellen fest, dass die Parameter ordentlich in den Attributen gespeichert werden.
- Der Operator `isinstance(a, B)` prüft ob ein Object `a` eine Instanz der Klasse `B` ist.

```
1 """Examples for using our class :class:`Circle`."""
2
3 from circle import Circle # Our new class `Circle`.
4 from point import Point # Our very first class ever: `Point`.
5 from shape import Shape # Our base class `Shape`.
6
7 circ: Circle = Circle(Point(2, 3), 5) # Create a new circle instance.
8 print(f" center: {circ.center.x}, {circ.center.y}")
9 print(f" radius: {circ.radius}")
10 print(f" perimeter: {circ.perimeter()}")
11 print(f" area: {circ.area()}")
12
13 # Do some instance tests.
14 print(f" isinstance(circ, Circle): {isinstance(circ, Circle)}")
15 print(f" isinstance(circ, Shape): {isinstance(circ, Shape)}")
16 print(f" isinstance(circ, object): {isinstance(circ, object)}")
17
18 # Explore the class hierarchy.
19 print(f" issubclass(Circle, Shape): {issubclass(Circle, Shape)}")
20 print(f" issubclass(Shape, Circle): {issubclass(Shape, Circle)}")
21 print(f" issubclass(Shape, object): {issubclass(Shape, object)}")
22 print(f" issubclass(Circle, object): {issubclass(Circle, object)})")
```

↓ python3 circle_user.py ↓

```
1 center: (2, 3)
2 radius: 5
3 perimeter: 31.41592653589793
4 area: 78.53981633974483
5 isinstance(circ, Circle): True
6 isinstance(circ, Shape): True
7 isinstance(circ, object): True
8 issubclass(Circle, Shape): True
9 issubclass(Shape, Circle): False
10 issubclass(Shape, object): True
11 issubclass(Circle, object): True
```

Beispiel: Die Klasse Circle benutzen

- Programm `circle_user.py` erforschen wir diese neue Klasse etwas.
- Wir erstellen die Instance `circ` of `Circle` und geben `point=Point(2, 3)` und `radius=4` an.
- Wir stellen fest, dass die Parameter ordentlich in den Attributen gespeichert werden.
- Der Operator `isinstance(a, B)` prüft ob ein Object `a` eine Instanz der Klasse `B` ist.
- Wenn ja, dann liefert er `True` und sonst `False`.

```
1 """Examples for using our class :class:`Circle`."""
2
3 from circle import Circle # Our new class `Circle`.
4 from point import Point # Our very first class ever: `Point`.
5 from shape import Shape # Our base class `Shape`.
6
7 circ: Circle = Circle(Point(2, 3), 5) # Create a new circle instance.
8 print(f" center: {circ.center.x}, {circ.center.y}")
9 print(f" radius: {circ.radius}")
10 print(f" perimeter: {circ.perimeter()}")
11 print(f" area: {circ.area()}")
12
13 # Do some instance tests.
14 print(f" isinstance(circ, Circle): {isinstance(circ, Circle)}")
15 print(f" isinstance(circ, Shape): {isinstance(circ, Shape)}")
16 print(f" isinstance(circ, object): {isinstance(circ, object)}")
17
18 # Explore the class hierarchy.
19 print(f" issubclass(Circle, Shape): {issubclass(Circle, Shape)}")
20 print(f" issubclass(Shape, Circle): {issubclass(Shape, Circle)}")
21 print(f" issubclass(Shape, object): {issubclass(Shape, object)}")
22 print(f" issubclass(Circle, object): {issubclass(Circle, object)})")
```

↓ python3 circle_user.py ↓

```
1 center: (2, 3)
2 radius: 5
3 perimeter: 31.41592653589793
4 area: 78.53981633974483
5 isinstance(circ, Circle): True
6 isinstance(circ, Shape): True
7 isinstance(circ, object): True
8 issubclass(Circle, Shape): True
9 issubclass(Shape, Circle): False
10 issubclass(Shape, object): True
11 issubclass(Circle, object): True
```

Beispiel: Die Klasse Circle benutzen

- Wir erstellen die Instance `circ` of `Circle` und geben `point=Point(2, 3)` und `radius=4` an.
- Wir stellen fest, dass die Parameter ordentlich in den Attributen gespeichert werden.
- Der Operator `isinstance(a, B)` prüft ob ein Object `a` eine Instanz der Klasse `B` ist.
- Wenn ja, dann liefert er `True` und sonst `False`.
- Wir bestätigen dass `isinstance(circ, Circle)` `True` ist.

```
1 """Examples for using our class :class:`Circle`."""
2
3 from circle import Circle # Our new class `Circle`.
4 from point import Point # Our very first class ever: `Point`.
5 from shape import Shape # Our base class `Shape`.
6
7 circ: Circle = Circle(Point(2, 3), 5) # Create a new circle instance.
8 print(f"center: {circ.center.x}, {circ.center.y}")
9 print(f"radius: {circ.radius}")
10 print(f"perimeter: {circ.perimeter()}")
11 print(f"area: {circ.area()}")
12
13 # Do some instance tests.
14 print(f" isinstance(circ, Circle): {isinstance(circ, Circle)}")
15 print(f" isinstance(circ, Shape): {isinstance(circ, Shape)}")
16 print(f" isinstance(circ, object): {isinstance(circ, object)}")
17
18 # Explore the class hierarchy.
19 print(f" issubclass(Circle, Shape): {issubclass(Circle, Shape)}")
20 print(f" issubclass(Shape, Circle): {issubclass(Shape, Circle)}")
21 print(f" issubclass(Shape, object): {issubclass(Shape, object)}")
22 print(f"issubclass(Circle, object): {issubclass(Circle, object)})")
```

↓ python3 circle_user.py ↓

```
1 center: (2, 3)
2 radius: 5
3 perimeter: 31.41592653589793
4 area: 78.53981633974483
5 isinstance(circ, Circle): True
6 isinstance(circ, Shape): True
7 isinstance(circ, object): True
8 issubclass(Circle, Shape): True
9 issubclass(Shape, Circle): False
10 issubclass(Shape, object): True
11 issubclass(Circle, object): True
```

Beispiel: Die Klasse Circle benutzen

- Wir stellen fest, dass die Parameter ordentlich in den Attributen gespeichert werden.
- Der Operator `isinstance(a, B)` prüft ob ein Object `a` eine Instanz der Klasse `B` ist.
- Wenn ja, dann liefert er `True` und sonst `False`.
- Wir bestätigen dass `isinstance(cir, Circle)` `True` ist.
- Es gilt auch `isinstance(cir, Shape)`.

```
1 """Examples for using our class :class:`Circle`."""
2
3 from circle import Circle # Our new class `Circle`.
4 from point import Point # Our very first class ever: `Point`.
5 from shape import Shape # Our base class `Shape`.
6
7 circ: Circle = Circle(Point(2, 3), 5) # Create a new circle instance.
8 print(f" center: {circ.center.x}, {circ.center.y}")
9 print(f" radius: {circ.radius}")
10 print(f" perimeter: {circ.perimeter()}")
11 print(f" area: {circ.area()}")
12
13 # Do some instance tests.
14 print(f" isinstance(circ, Circle): {isinstance(circ, Circle)}")
15 print(f" isinstance(circ, Shape): {isinstance(circ, Shape)}")
16 print(f" isinstance(circ, object): {isinstance(circ, object)}")
17
18 # Explore the class hierarchy.
19 print(f" issubclass(Circle, Shape): {issubclass(Circle, Shape)}")
20 print(f" issubclass(Shape, Circle): {issubclass(Shape, Circle)}")
21 print(f" issubclass(Shape, object): {issubclass(Shape, object)}")
22 print(f" issubclass(Circle, object): {issubclass(Circle, object)})")
```

↓ python3 circle_user.py ↓

```
1 center: (2, 3)
2 radius: 5
3 perimeter: 31.41592653589793
4 area: 78.53981633974483
5 isinstance(circ, Circle): True
6 isinstance(circ, Shape): True
7 isinstance(circ, object): True
8 issubclass(Circle, Shape): True
9 issubclass(Shape, Circle): False
10 issubclass(Shape, object): True
11 issubclass(Circle, object): True
```

Beispiel: Die Klasse Circle benutzen

- Der Operator `isinstance(a, B)` prüft ob ein Object `a` eine Instanz der Klasse `B` ist.
- Wenn ja, dann liefert er `True` und sonst `False`.
- Wir bestätigen dass `isinstance(cir, Circle) True` ist.
- Es gilt auch `isinstance(cir, Shape)`.
- Jede Instanz von `Circle` ist auch eine Instanz von `Shape`.

```
1 """Examples for using our class :class:`Circle`."""
2
3 from circle import Circle # Our new class `Circle`.
4 from point import Point # Our very first class ever: `Point`.
5 from shape import Shape # Our base class `Shape`.
6
7 circ: Circle = Circle(Point(2, 3), 5) # Create a new circle instance.
8 print(f"center: {circ.center.x}, {circ.center.y}")
9 print(f"radius: {circ.radius}")
10 print(f"perimeter: {circ.perimeter()}")
11 print(f"area: {circ.area()}")
12
13 # Do some instance tests.
14 print(f" isinstance(circ, Circle): {isinstance(circ, Circle)}")
15 print(f" isinstance(circ, Shape): {isinstance(circ, Shape)}")
16 print(f" isinstance(circ, object): {isinstance(circ, object)}")
17
18 # Explore the class hierarchy.
19 print(f" issubclass(Circle, Shape): {issubclass(Circle, Shape)}")
20 print(f" issubclass(Shape, Circle): {issubclass(Shape, Circle)}")
21 print(f" issubclass(Shape, object): {issubclass(Shape, object)}")
22 print(f"issubclass(Circle, object): {issubclass(Circle, object)})")
```

↓ python3 circle_user.py ↓

```
1 center: (2, 3)
2 radius: 5
3 perimeter: 31.41592653589793
4 area: 78.53981633974483
5 isinstance(circ, Circle): True
6 isinstance(circ, Shape): True
7 isinstance(circ, object): True
8 issubclass(Circle, Shape): True
9 issubclass(Shape, Circle): False
10 issubclass(Shape, object): True
11 issubclass(Circle, object): True
```

Beispiel: Die Klasse Circle benutzen

- Wenn ja, dann liefert er `True` und sonst `False`.
- Wir bestätigen dass `isinstance(cir, Circle)` `True` ist.
- Es gilt auch `isinstance(cir, Shape)`.
- Jede Instanz von `Circle` ist auch eine Instanz von `Shape`.
- Weil `Circle` ein Spezialfall von `Shape` ist.

```
1 """Examples for using our class :class:`Circle`."""
2
3 from circle import Circle # Our new class `Circle`.
4 from point import Point # Our very first class ever: `Point`.
5 from shape import Shape # Our base class `Shape`.
6
7 circ: Circle = Circle(Point(2, 3), 5) # Create a new circle instance.
8 print(f"center: {circ.center.x}, {circ.center.y}")
9 print(f"radius: {circ.radius}")
10 print(f"perimeter: {circ.perimeter()}")
11 print(f"area: {circ.area()}")
12
13 # Do some instance tests.
14 print(f" isinstance(circ, Circle): {isinstance(circ, Circle)}")
15 print(f" isinstance(circ, Shape): {isinstance(circ, Shape)}")
16 print(f" isinstance(circ, object): {isinstance(circ, object)}")
17
18 # Explore the class hierarchy.
19 print(f" issubclass(Circle, Shape): {issubclass(Circle, Shape)}")
20 print(f" issubclass(Shape, Circle): {issubclass(Shape, Circle)}")
21 print(f" issubclass(Shape, object): {issubclass(Shape, object)}")
22 print(f"issubclass(Circle, object): {issubclass(Circle, object)})")
```

↓ python3 circle_user.py ↓

```
1 center: (2, 3)
2 radius: 5
3 perimeter: 31.41592653589793
4 area: 78.53981633974483
5 isinstance(circ, Circle): True
6 isinstance(circ, Shape): True
7 isinstance(circ, object): True
8 issubclass(Circle, Shape): True
9 issubclass(Shape, Circle): False
10 issubclass(Shape, object): True
11 issubclass(Circle, object): True
```

Beispiel: Die Klasse Circle benutzen



- Wir bestätigen dass
`isinstance(cir, Circle)` `True` ist.
- Es gilt auch
`isinstance(cir, Shape)`.
- Jede Instanz von `Circle` ist auch eine Instanz von `Shape`.
- Weil `Circle` ein Spezialfall von `Shape` ist.
- Es gilt auch
`isinstance(cir, object)`.

```
1 """Examples for using our class :class:`Circle`."""
2
3 from circle import Circle # Our new class `Circle`.
4 from point import Point # Our very first class ever: `Point`.
5 from shape import Shape # Our base class `Shape`.
6
7 circ: Circle = Circle(Point(2, 3), 5) # Create a new circle instance.
8 print(f" center: {circ.center.x}, {circ.center.y}")
9 print(f" radius: {circ.radius}")
10 print(f" perimeter: {circ.perimeter()}")
11 print(f" area: {circ.area()}")
12
13 # Do some instance tests.
14 print(f" isinstance(circ, Circle): {isinstance(circ, Circle)}")
15 print(f" isinstance(circ, Shape): {isinstance(circ, Shape)}")
16 print(f" isinstance(circ, object): {isinstance(circ, object)}")
17
18 # Explore the class hierarchy.
19 print(f" issubclass(Circle, Shape): {issubclass(Circle, Shape)}")
20 print(f" issubclass(Shape, Circle): {issubclass(Shape, Circle)}")
21 print(f" issubclass(Shape, object): {issubclass(Shape, object)}")
22 print(f" issubclass(Circle, object): {issubclass(Circle, object)})")
```

↓ python3 circle_user.py ↓

```
1 center: (2, 3)
2 radius: 5
3 perimeter: 31.41592653589793
4 area: 78.53981633974483
5 isinstance(circ, Circle): True
6 isinstance(circ, Shape): True
7 isinstance(circ, object): True
8 issubclass(Circle, Shape): True
9 issubclass(Shape, Circle): False
10 issubclass(Shape, object): True
11 issubclass(Circle, object): True
```

Beispiel: Die Klasse Circle benutzen



- Es gilt auch
`isinstance(cir, Shape)`.
- Jede Instanz von `Circle` ist auch eine Instanz von `Shape`.
- Weil `Circle` ein Spezialfall von `Shape` ist.
- Es gilt auch
`isinstance(cir, object)`.
- `object` ist die niedrigste Basisklasse aller Klassen in Python^{14,66}.

```
1 """Examples for using our class :class:`Circle`."""
2
3 from circle import Circle # Our new class `Circle`.
4 from point import Point # Our very first class ever: `Point`.
5 from shape import Shape # Our base class `Shape`.
6
7 circ: Circle = Circle(Point(2, 3), 5) # Create a new circle instance.
8 print(f" center: {circ.center.x}, {circ.center.y}")
9 print(f" radius: {circ.radius}")
10 print(f" perimeter: {circ.perimeter()}")
11 print(f" area: {circ.area()}")
12
13 # Do some instance tests.
14 print(f" isinstance(circ, Circle): {isinstance(circ, Circle)}")
15 print(f" isinstance(circ, Shape): {isinstance(circ, Shape)}")
16 print(f" isinstance(circ, object): {isinstance(circ, object)}")
17
18 # Explore the class hierarchy.
19 print(f" issubclass(Circle, Shape): {issubclass(Circle, Shape)}")
20 print(f" issubclass(Shape, Circle): {issubclass(Shape, Circle)}")
21 print(f" issubclass(Shape, object): {issubclass(Shape, object)}")
22 print(f" issubclass(Circle, object): {issubclass(Circle, object)})")
```

↓ python3 circle_user.py ↓

```
1 center: (2, 3)
2 radius: 5
3 perimeter: 31.41592653589793
4 area: 78.53981633974483
5 isinstance(circ, Circle): True
6 isinstance(circ, Shape): True
7 isinstance(circ, object): True
8 issubclass(Circle, Shape): True
9 issubclass(Shape, Circle): False
10 issubclass(Shape, object): True
11 issubclass(Circle, object): True
```

Beispiel: Die Klasse Circle benutzen

- Jede Instanz von `Circle` ist auch eine Instanz von `Shape`.
- Weil `Circle` ein Spezialfall von `Shape` ist.
- Es gilt auch `isinstance(cir, object)`.
- `object` ist die niedrigste Basisklasse aller Klassen in Python^{14,66}.
- Wenn wir nicht explizit eine Basisklasse bei der Klassendefinition angeben, dann wird `object` verwendet.

```
1 """Examples for using our class :class:`Circle`."""
2
3 from circle import Circle # Our new class `Circle`.
4 from point import Point # Our very first class ever: `Point`.
5 from shape import Shape # Our base class `Shape`.
6
7 circ: Circle = Circle(Point(2, 3), 5) # Create a new circle instance.
8 print(f"center: {circ.center.x}, {circ.center.y}")
9 print(f"radius: {circ.radius}")
10 print(f"perimeter: {circ.perimeter()}")
11 print(f"area: {circ.area()}")
12
13 # Do some instance tests.
14 print(f" isinstance(circ, Circle): {isinstance(circ, Circle)}")
15 print(f" isinstance(circ, Shape): {isinstance(circ, Shape)}")
16 print(f" isinstance(circ, object): {isinstance(circ, object)}")
17
18 # Explore the class hierarchy.
19 print(f" issubclass(Circle, Shape): {issubclass(Circle, Shape)}")
20 print(f" issubclass(Shape, Circle): {issubclass(Shape, Circle)}")
21 print(f" issubclass(Shape, object): {issubclass(Shape, object)}")
22 print(f"issubclass(Circle, object): {issubclass(Circle, object)})")
```

↓ python3 circle_user.py ↓

```
1 center: (2, 3)
2 radius: 5
3 perimeter: 31.41592653589793
4 area: 78.53981633974483
5 isinstance(circ, Circle): True
6 isinstance(circ, Shape): True
7 isinstance(circ, object): True
8 issubclass(Circle, Shape): True
9 issubclass(Shape, Circle): False
10 issubclass(Shape, object): True
11 issubclass(Circle, object): True
```

Beispiel: Die Klasse Circle benutzen

- Weil `Circle` ein Spezialfall von `Shape` ist.
- Es gilt auch `isinstance(cir, object)`.
- `object` ist die niedrigste Basisklasse aller Klassen in Python^{14,66}.
- Wenn wir nicht explizit eine Basisklasse bei der Klassendefinition angeben, dann wird `object` verwendet.
- Das ist der Fall für unsere Klasse `Shape` und auch für unsere älteren Klassen `Point` und `KahanSum`.

```
1 """Examples for using our class :class:`Circle`."""
2
3 from circle import Circle # Our new class `Circle`.
4 from point import Point # Our very first class ever: `Point`.
5 from shape import Shape # Our base class `Shape`.
6
7 circ: Circle = Circle(Point(2, 3), 5) # Create a new circle instance.
8 print(f"center: {circ.center.x}, {circ.center.y}")
9 print(f"radius: {circ.radius}")
10 print(f"perimeter: {circ.perimeter()}")
11 print(f"area: {circ.area()}")
12
13 # Do some instance tests.
14 print(f" isinstance(circ, Circle): {isinstance(circ, Circle)}")
15 print(f" isinstance(circ, Shape): {isinstance(circ, Shape)}")
16 print(f" isinstance(circ, object): {isinstance(circ, object)}")
17
18 # Explore the class hierarchy.
19 print(f" issubclass(Circle, Shape): {issubclass(Circle, Shape)}")
20 print(f" issubclass(Shape, Circle): {issubclass(Shape, Circle)}")
21 print(f" issubclass(Shape, object): {issubclass(Shape, object)}")
22 print(f"issubclass(Circle, object): {issubclass(Circle, object)})")
```

↓ python3 circle_user.py ↓

```
1 center: (2, 3)
2 radius: 5
3 perimeter: 31.41592653589793
4 area: 78.53981633974483
5 isinstance(circ, Circle): True
6 isinstance(circ, Shape): True
7 isinstance(circ, object): True
8 issubclass(Circle, Shape): True
9 issubclass(Shape, Circle): False
10 issubclass(Shape, object): True
11 issubclass(Circle, object): True
```

Beispiel: Die Klasse Circle benutzen



- Es gilt auch `isinstance(cir, object)`.
- `object` ist die niedrigste Basisklasse aller Klassen in Python^{14,66}.
- Wenn wir nicht explizit eine Basisklasse bei der Klassendefinition angeben, dann wird `object` verwendet.
- Das ist der Fall für unsere Klasse `Shape` und auch für unsere älteren Klassen `Point` und `KahanSum`.
- Jeder `Circle` ist also auch ein `object`.

```
1 """Examples for using our class :class:`Circle`."""
2
3 from circle import Circle # Our new class `Circle`.
4 from point import Point # Our very first class ever: `Point`.
5 from shape import Shape # Our base class `Shape`.
6
7 circ: Circle = Circle(Point(2, 3), 5) # Create a new circle instance.
8 print(f" center: {circ.center.x}, {circ.center.y}")
9 print(f" radius: {circ.radius}")
10 print(f" perimeter: {circ.perimeter()}")
11 print(f" area: {circ.area()}")
12
13 # Do some instance tests.
14 print(f" isinstance(circ, Circle): {isinstance(circ, Circle)}")
15 print(f" isinstance(circ, Shape): {isinstance(circ, Shape)}")
16 print(f" isinstance(circ, object): {isinstance(circ, object)}")
17
18 # Explore the class hierarchy.
19 print(f" issubclass(Circle, Shape): {issubclass(Circle, Shape)}")
20 print(f" issubclass(Shape, Circle): {issubclass(Shape, Circle)}")
21 print(f" issubclass(Shape, object): {issubclass(Shape, object)}")
22 print(f" issubclass(Circle, object): {issubclass(Circle, object)})")
```

↓ python3 circle_user.py ↓

```
1 center: (2, 3)
2 radius: 5
3 perimeter: 31.41592653589793
4 area: 78.53981633974483
5 isinstance(circ, Circle): True
6 isinstance(circ, Shape): True
7 isinstance(circ, object): True
8 issubclass(Circle, Shape): True
9 issubclass(Shape, Circle): False
10 issubclass(Shape, object): True
11 issubclass(Circle, object): True
```

Beispiel: Die Klasse Circle benutzen

- `object` ist die niedrigste Basisklasse aller Klassen in Python^{14,66}.
- Wenn wir nicht explizit eine Basisklasse bei der Klassendefinition angeben, dann wird `object` verwendet.
- Das ist der Fall für unsere Klasse `Shape` und auch für unsere älteren Klassen `Point` und `KahanSum`.
- Jeder `Circle` ist also auch ein `object`.
- Während der Operator `isinstance` Objekte mit Klassen in Verbindung bringt, macht der Operator `issubclass` das selbe mit zwei Klassen.

```
1 """Examples for using our class :class:`Circle`."""
2
3 from circle import Circle # Our new class `Circle`.
4 from point import Point # Our very first class ever: `Point`.
5 from shape import Shape # Our base class `Shape`.
6
7 circ: Circle = Circle(Point(2, 3), 5) # Create a new circle instance.
8 print(f"center: {circ.center.x}, {circ.center.y}")
9 print(f"radius: {circ.radius}")
10 print(f"perimeter: {circ.perimeter()}")
11 print(f"area: {circ.area()}")
12
13 # Do some instance tests.
14 print(f" isinstance(circ, Circle): {isinstance(circ, Circle)}")
15 print(f" isinstance(circ, Shape): {isinstance(circ, Shape)}")
16 print(f" isinstance(circ, object): {isinstance(circ, object)}")
17
18 # Explore the class hierarchy.
19 print(f" issubclass(Circle, Shape): {issubclass(Circle, Shape)}")
20 print(f" issubclass(Shape, Circle): {issubclass(Shape, Circle)}")
21 print(f" issubclass(Shape, object): {issubclass(Shape, object)}")
22 print(f" issubclass(Circle, object): {issubclass(Circle, object)}")
```

↓ python3 circle_user.py ↓

```
1 center: (2, 3)
2 radius: 5
3 perimeter: 31.41592653589793
4 area: 78.53981633974483
5 isinstance(circ, Circle): True
6 isinstance(circ, Shape): True
7 isinstance(circ, object): True
8 issubclass(Circle, Shape): True
9 issubclass(Shape, Circle): False
10 issubclass(Shape, object): True
11 issubclass(Circle, object): True
```

Beispiel: Die Klasse Circle benutzen

- Das ist der Fall für unsere Klasse `Shape` und auch für unsere älteren Klassen `Point` und `KahanSum`.
- Jeder `Circle` ist also auch ein `object`.
- Während der Operator `isinstance` Objekte mit Klassen in Verbindung bringt, macht der Operator `issubclass` das gleiche mit zwei Klassen.
- `issubclass(A, B)` ist `True` wenn Klasse `A` eine Subklasse von Klasse `B` ist, also von `B` erbt.

```
1 """Examples for using our class :class:`Circle`."""
2
3 from circle import Circle # Our new class `Circle`.
4 from point import Point # Our very first class ever: `Point`.
5 from shape import Shape # Our base class `Shape`.
6
7 circ: Circle = Circle(Point(2, 3), 5) # Create a new circle instance.
8 print(f"center: {circ.center.x}, {circ.center.y}")
9 print(f"radius: {circ.radius}")
10 print(f"perimeter: {circ.perimeter()}")
11 print(f"area: {circ.area()}")
12
13 # Do some instance tests.
14 print(f" isinstance(circ, Circle): {isinstance(circ, Circle)}")
15 print(f" isinstance(circ, Shape): {isinstance(circ, Shape)}")
16 print(f" isinstance(circ, object): {isinstance(circ, object)}")
17
18 # Explore the class hierarchy.
19 print(f" issubclass(Circle, Shape): {issubclass(Circle, Shape)}")
20 print(f" issubclass(Shape, Circle): {issubclass(Shape, Circle)}")
21 print(f" issubclass(Shape, object): {issubclass(Shape, object)}")
22 print(f" issubclass(Circle, object): {issubclass(Circle, object)}")
```

↓ python3 circle_user.py ↓

```
1 center: (2, 3)
2 radius: 5
3 perimeter: 31.41592653589793
4 area: 78.53981633974483
5 isinstance(circ, Circle): True
6 isinstance(circ, Shape): True
7 isinstance(circ, object): True
8 issubclass(Circle, Shape): True
9 issubclass(Shape, Circle): False
10 issubclass(Shape, object): True
11 issubclass(Circle, object): True
```

Beispiel: Die Klasse Circle benutzen

- Jeder `Circle` ist also auch ein `object`.
- Während der Operator `isinstance` Objekte mit Klassen in Verbindung bringt, macht der Operator `issubclass` das selbe mit zwei Klassen.
- `issubclass(A, B)` ist `True` wenn Klasse `A` eine Subklasse von Klasse `B` ist, also von `B` erbt.
- Darum ist `issubclass(Circle, Shape)` `True` während `issubclass(Shape, Circle)` `False` ergibt.

```
1 """Examples for using our class :class:`Circle`."""
2
3 from circle import Circle # Our new class `Circle`.
4 from point import Point # Our very first class ever: `Point`.
5 from shape import Shape # Our base class `Shape`.
6
7 circ: Circle = Circle(Point(2, 3), 5) # Create a new circle instance.
8 print(f" center: {circ.center.x}, {circ.center.y}")
9 print(f" radius: {circ.radius}")
10 print(f" perimeter: {circ.perimeter()}")
11 print(f" area: {circ.area()}")
12
13 # Do some instance tests.
14 print(f" isinstance(circ, Circle): {isinstance(circ, Circle)}")
15 print(f" isinstance(circ, Shape): {isinstance(circ, Shape)}")
16 print(f" isinstance(circ, object): {isinstance(circ, object)}")
17
18 # Explore the class hierarchy.
19 print(f" issubclass(Circle, Shape): {issubclass(Circle, Shape)}")
20 print(f" issubclass(Shape, Circle): {issubclass(Shape, Circle)}")
21 print(f" issubclass(Shape, object): {issubclass(Shape, object)}")
22 print(f" issubclass(Circle, object): {issubclass(Circle, object)})")
```

↓ python3 circle_user.py ↓

```
1 center: (2, 3)
2 radius: 5
3 perimeter: 31.41592653589793
4 area: 78.53981633974483
5 isinstance(circ, Circle): True
6 isinstance(circ, Shape): True
7 isinstance(circ, object): True
8 issubclass(Circle, Shape): True
9 issubclass(Shape, Circle): False
10 issubclass(Shape, object): True
11 issubclass(Circle, object): True
```

Beispiel: Die Klasse Circle benutzen

- Während der Operator `isinstance` Objekte mit Klassen in Verbindung bringt, macht der Operator `issubclass` das selbe mit zwei Klassen.
- `issubclass(A, B)` ist `True` wenn Klasse `A` eine Subklasse von Klasse `B` ist, also von `B` erbt.
- Darum ist `issubclass(Circle, Shape)` `True` während `issubclass(Shape, Circle)` `False` ergibt.
- `issubclass(Shape, object)` und `issubclass(Circle, object)` sind auch beide `True`.

```
1 """Examples for using our class :class:`Circle`."""
2
3 from circle import Circle # Our new class `Circle`.
4 from point import Point # Our very first class ever: `Point`.
5 from shape import Shape # Our base class `Shape`.
6
7 circ: Circle = Circle(Point(2, 3), 5) # Create a new circle instance.
8 print(f"center: {circ.center.x}, {circ.center.y}")
9 print(f"radius: {circ.radius}")
10 print(f"perimeter: {circ.perimeter()}")
11 print(f"area: {circ.area()}")
12
13 # Do some instance tests.
14 print(f" isinstance(circ, Circle): {isinstance(circ, Circle)}")
15 print(f" isinstance(circ, Shape): {isinstance(circ, Shape)}")
16 print(f" isinstance(circ, object): {isinstance(circ, object)}")
17
18 # Explore the class hierarchy.
19 print(f" issubclass(Circle, Shape): {issubclass(Circle, Shape)}")
20 print(f" issubclass(Shape, Circle): {issubclass(Shape, Circle)}")
21 print(f" issubclass(Shape, object): {issubclass(Shape, object)}")
22 print(f" issubclass(Circle, object): {issubclass(Circle, object)})")
```

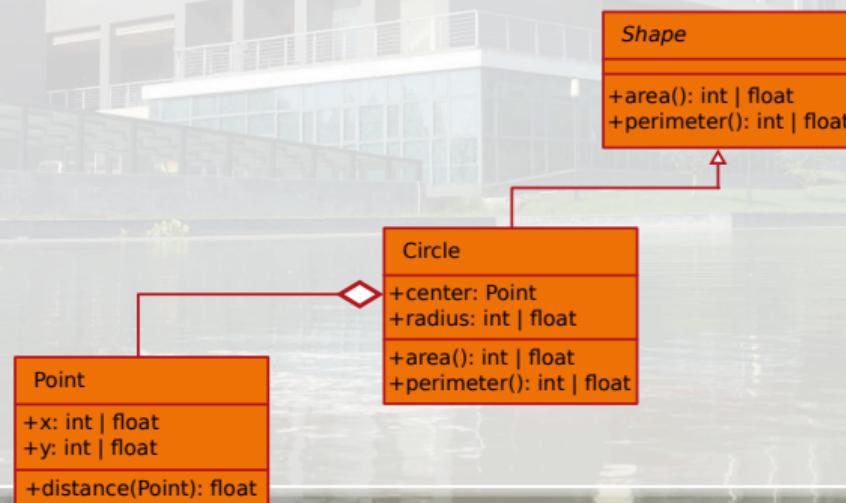
↓ python3 circle_user.py ↓

```
1 center: (2, 3)
2 radius: 5
3 perimeter: 31.41592653589793
4 area: 78.53981633974483
5
6 isinstance(circ, Circle): True
7 isinstance(circ, Shape): True
8 isinstance(circ, object): True
9 issubclass(Circle, Shape): True
10 issubclass(Shape, Circle): False
11 issubclass(Shape, object): True
12 issubclass(Circle, object): True
```



Beispiel: Polygone

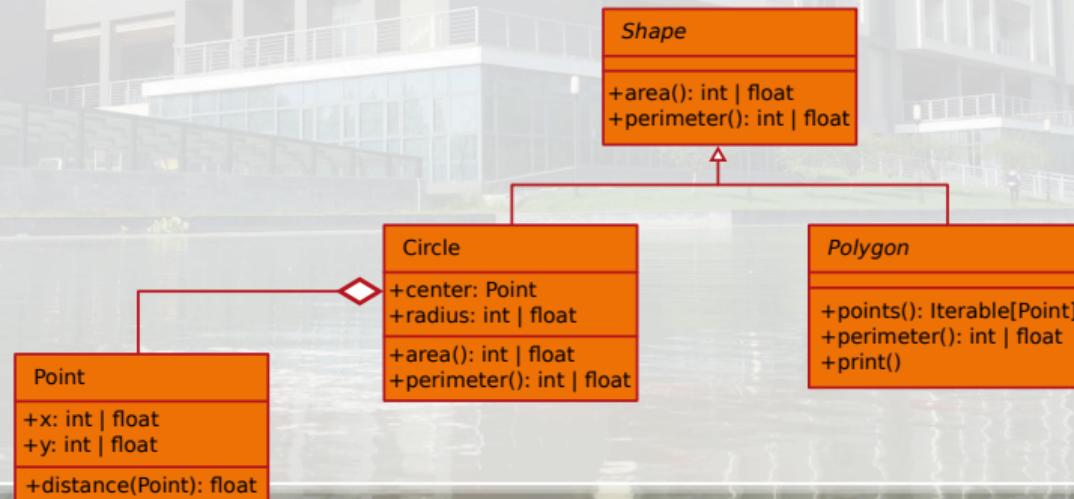
- Kreise sind nicht die einzigen Sonderfälle von Formen.





Beispiel: Polygone

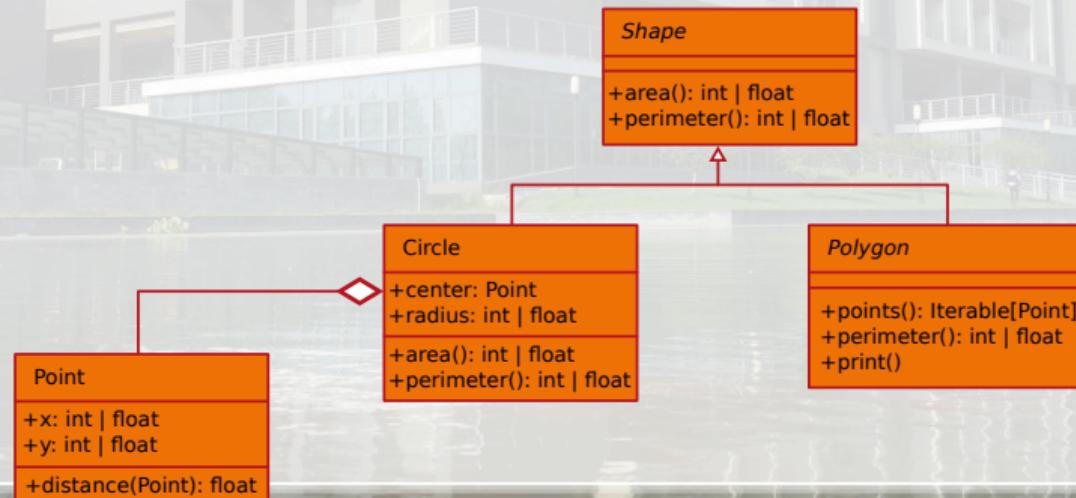
- Kreise sind nicht die einzigen Sonderfälle von Formen.
- Eine andere sehr allgemeine Klasse von Formen sind Polygone.





Beispiel: Polygone

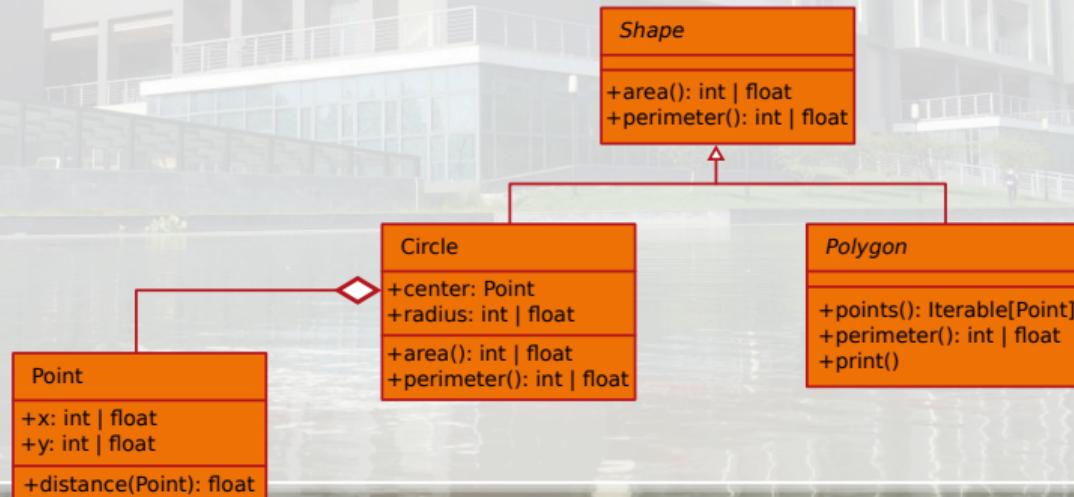
- Kreise sind nicht die einzigen Sonderfälle von Formen.
- Eine andere sehr allgemeine Klasse von Formen sind Polygone.
- Polygone sind Formen, die von geraden Linien begrenzt werden.





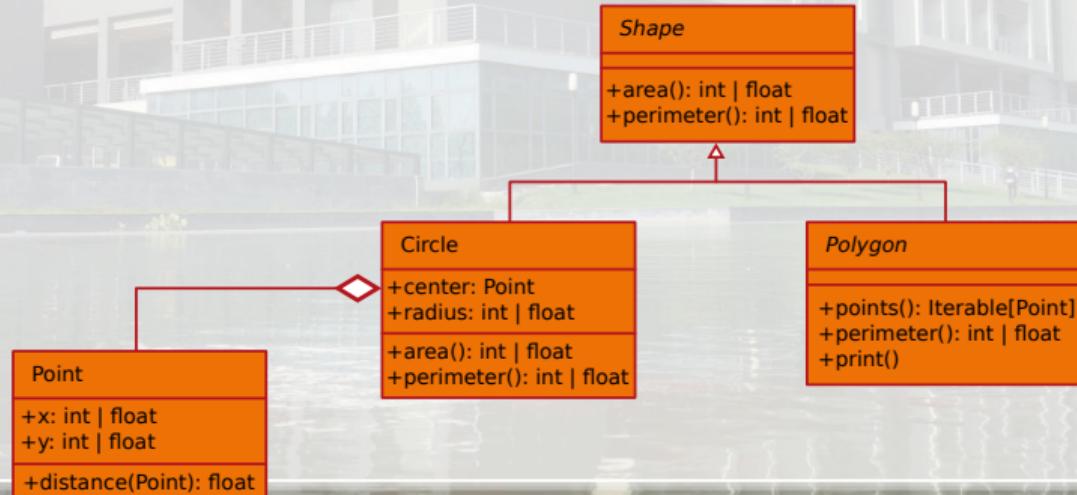
Beispiel: Polygone

- Kreise sind nicht die einzigen Sonderfälle von Formen.
- Eine andere sehr allgemeine Klasse von Formen sind Polygone.
- Polygone sind Formen, die von geraden Linien begrenzt werden.
- Deshalb kann jedes Polygon durch das Spezifizieren seiner Eckpunkte definiert werden.



Beispiel: Polygone

- Kreise sind nicht die einzigen Sonderfälle von Formen.
- Eine andere sehr allgemeine Klasse von Formen sind Polygone.
- Polygone sind Formen, die von geraden Linien begrenzt werden.
- Deshalb kann jedes Polygon durch das Spezifizieren seiner Eckpunkte definiert werden.
- Wir definieren also für jedes Polygon eine Methode `points()`, die eine Sequenz der Eckpunkte zurückliefern soll.



Beispiel: Die Klasse Circle

- Wir definieren die Klasse `Polygon` in Datei `polygon.py`.

```
1 """A polygon is a figure described by its corner points."""
2
3 from typing import Iterable
4
5 from kahan_sum import KahanSum
6 from point import Point
7 from shape import Shape
8
9
10 class Polygon(Shape):
11     """Polygons are shapes delimited by straight lines."""
12
13     def points(self) -> Iterable[Point]:
14         """
15             Get a :class:`Iterable` over the points describing this polygon.
16
17             :return: the points describing the polygon
18             :raises NotImplementedError: Must be implemented by subclasses.
19         """
20
21         raise NotImplementedError # must be implemented by subclasses
22
23     def perimeter(self) -> int | float:
24         """
25             Get the perimeter of this polygon.
26
27             :return: the perimeter of this polygon
28         """
29
30         previous: Point | None = None # the previous point
31         first: Point | None = None # the first point
32         total: KahanSum = KahanSum() # the total perimeter length sum
33         for current in self.points():
34             # Iterate over the points.
35             if previous is None: # We got the first point.
36                 previous = first = current # Remember it for last step.
37             else: # We now have previous != None, so we can add length.
38                 total.add(previous.distance(current)) # Add length.
39                 previous = current # Current point becomes previous.
40         total.add(previous.distance(first)) # distance back to start
41         return total.result() # Return the perimeter.
42
43     def print(self) -> None:
44         """Print the points of this polygon."""
45         print(", ".join(f"({p.x}, {p.y})" for p in self.points()))
```

Beispiel: Die Klasse Circle

- Wir definieren die Klasse `Polygon` in Datei `polygon.py`.
- Sie ist ein Spezialfall und erbt daher von `Shape`.

```
1 """A polygon is a figure described by its corner points."""
2
3 from typing import Iterable
4
5 from kahan_sum import KahanSum
6 from point import Point
7 from shape import Shape
8
9
10 class Polygon(Shape):
11     """Polygons are shapes delimited by straight lines."""
12
13     def points(self) -> Iterable[Point]:
14         """
15             Get a :class:`Iterable` over the points describing this polygon.
16
17             :return: the points describing the polygon
18             :raises NotImplementedError: Must be implemented by subclasses.
19         """
20
21         raise NotImplementedError # must be implemented by subclasses
22
23     def perimeter(self) -> int | float:
24         """
25             Get the perimeter of this polygon.
26
27             :return: the perimeter of this polygon
28         """
29
30         previous: Point | None = None # the previous point
31         first: Point | None = None # the first point
32         total: KahanSum = KahanSum() # the total perimeter length sum
33         for current in self.points():
34             # Iterate over the points.
35             if previous is None: # We got the first point.
36                 previous = first = current # Remember it for last step.
37             else: # We now have previous != None, so we can add length.
38                 total.add(previous.distance(current)) # Add length.
39                 previous = current # Current point becomes previous.
40         total.add(previous.distance(first)) # distance back to start
41         return total.result() # Return the perimeter.
42
43     def print(self) -> None:
44         """Print the points of this polygon."""
45         print(", ".join(f"({p.x}, {p.y})" for p in self.points()))
```

Beispiel: Die Klasse Circle

- Wir definieren die Klasse `Polygon` in Datei `polygon.py`.
- Sie ist ein Spezialfall und erbt daher von `Shape`.
- `Polygon` erweitert das Interface von `Shape`, in dem es die Methode `points` definiert.

```
1 """A polygon is a figure described by its corner points."""
2
3 from typing import Iterable
4
5 from kahan_sum import KahanSum
6 from point import Point
7 from shape import Shape
8
9
10 class Polygon(Shape):
11     """Polygons are shapes delimited by straight lines."""
12
13     def points(self) -> Iterable[Point]:
14         """
15             Get a :class:`Iterable` over the points describing this polygon.
16
17             :return: the points describing the polygon
18             :raises NotImplementedError: Must be implemented by subclasses.
19         """
20
21         raise NotImplementedError # must be implemented by subclasses
22
23     def perimeter(self) -> int | float:
24         """
25             Get the perimeter of this polygon.
26
27             :return: the perimeter of this polygon
28         """
29
30         previous: Point | None = None # the previous point
31         first: Point | None = None # the first point
32         total: KahanSum = KahanSum() # the total perimeter length sum
33         for current in self.points():
34             # Iterate over the points.
35             if previous is None: # We got the first point.
36                 previous = first = current # Remember it for last step.
37             else: # We now have previous != None, so we can add length.
38                 total.add(previous.distance(current)) # Add length.
39                 previous = current # Current point becomes previous.
40         total.add(previous.distance(first)) # distance back to start
41         return total.result() # Return the perimeter.
42
43     def print(self) -> None:
44         """Print the points of this polygon."""
45         print(", ".join(f"({p.x}, {p.y})" for p in self.points()))
```

Beispiel: Die Klasse Circle

- Wir definieren die Klasse `Polygon` in Datei `polygon.py`.
- Sie ist ein Spezialfall und erbt daher von `Shape`.
- `Polygon` erweitert das Interface von `Shape`, in dem es die Methode `points` definiert.
- Diese Methode liefert einen `Iterable` von Instanzen von `Point`, die die Eckpunkte des Polygons definieren.

```
1 """A polygon is a figure described by its corner points."""
2
3 from typing import Iterable
4
5 from kahan_sum import KahanSum
6 from point import Point
7 from shape import Shape
8
9
10 class Polygon(Shape):
11     """Polygons are shapes delimited by straight lines."""
12
13     def points(self) -> Iterable[Point]:
14         """
15             Get a :class:`Iterable` over the points describing this polygon.
16
17             :return: the points describing the polygon
18             :raises NotImplementedError: Must be implemented by subclasses.
19         """
20         raise NotImplementedError # must be implemented by subclasses
21
22     def perimeter(self) -> int | float:
23         """
24             Get the perimeter of this polygon.
25
26             :return: the perimeter of this polygon
27         """
28         previous: Point | None = None # the previous point
29         first: Point | None = None # the first point
30         total: KahanSum = KahanSum() # the total perimeter length sum
31         for current in self.points():
32             # Iterate over the points.
33             if previous is None: # We got the first point.
34                 previous = first = current # Remember it for last step.
35             else: # We now have previous != None, so we can add length.
36                 total.add(previous.distance(current)) # Add length.
37                 previous = current # Current point becomes previous.
38         total.add(previous.distance(first)) # distance back to start
39         return total.result() # Return the perimeter.
40
41     def print(self) -> None:
42         """Print the points of this polygon."""
43         print(", ".join(f"({p.x}, {p.y})" for p in self.points()))
```

Beispiel: Die Klasse Circle

- Wir definieren die Klasse `Polygon` in Datei `polygon.py`.
- Sie ist ein Spezialfall und erbt daher von `Shape`.
- `Polygon` erweitert das Interface von `Shape`, in dem es die Methode `points` definiert.
- Diese Methode liefert einen `Iterable` von Instanzen von `Point`, die die Eckpunkte des Polygons definieren.
- Unser Ziel ist es hier, eine Basisklasse für verschiedene Arten von Polygonen bereitzustellen.

```
1 """A polygon is a figure described by its corner points."""
2
3 from typing import Iterable
4
5 from kahan_sum import KahanSum
6 from point import Point
7 from shape import Shape
8
9
10 class Polygon(Shape):
11     """Polygons are shapes delimited by straight lines."""
12
13     def points(self) -> Iterable[Point]:
14         """
15             Get a :class:`Iterable` over the points describing this polygon.
16
17             :return: the points describing the polygon
18             :raises NotImplementedError: Must be implemented by subclasses.
19         """
20         raise NotImplementedError # must be implemented by subclasses
21
22     def perimeter(self) -> int | float:
23         """
24             Get the perimeter of this polygon.
25
26             :return: the perimeter of this polygon
27         """
28         previous: Point | None = None # the previous point
29         first: Point | None = None # the first point
30         total: KahanSum = KahanSum() # the total perimeter length sum
31         for current in self.points():
32             # Iterate over the points.
33             if previous is None: # We got the first point.
34                 previous = first = current # Remember it for last step.
35             else: # We now have previous != None, so we can add length.
36                 total.add(previous.distance(current)) # Add length.
37                 previous = current # Current point becomes previous.
38         total.add(previous.distance(first)) # distance back to start
39         return total.result() # Return the perimeter.
40
41     def print(self) -> None:
42         """Print the points of this polygon."""
43         print(", ".join(f"({p.x}, {p.y})" for p in self.points()))
```

Beispiel: Die Klasse Circle

- Sie ist ein Spezialfall und erbt daher von `Shape`.
- `Polygon` erweitert das Interface von `Shape`, in dem es die Methode `points` definiert.
- Diese Methode liefert einen `Iterable` von Instanzen von `Point`, die die Eckpunkte des Polygons definieren.
- Unser Ziel ist es hier, eine Basisklasse für verschiedene Arten von Polygonen bereitzustellen.
- Wir wollen nicht direkt und sofort eine Datenstruktur für beliebige Polygone implementieren.

```
1 """A polygon is a figure described by its corner points."""
2
3 from typing import Iterable
4
5 from kahan_sum import KahanSum
6 from point import Point
7 from shape import Shape
8
9
10 class Polygon(Shape):
11     """Polygons are shapes delimited by straight lines."""
12
13     def points(self) -> Iterable[Point]:
14         """
15             Get a :class:`Iterable` over the points describing this polygon.
16
17             :return: the points describing the polygon
18             :raises NotImplementedError: Must be implemented by subclasses.
19         """
20         raise NotImplementedError # must be implemented by subclasses
21
22     def perimeter(self) -> int | float:
23         """
24             Get the perimeter of this polygon.
25
26             :return: the perimeter of this polygon
27         """
28         previous: Point | None = None # the previous point
29         first: Point | None = None # the first point
30         total: KahanSum = KahanSum() # the total perimeter length sum
31         for current in self.points():
32             # Iterate over the points.
33             if previous is None: # We got the first point.
34                 previous = first = current # Remember it for last step.
35             else: # We now have previous != None, so we can add length.
36                 total.add(previous.distance(current)) # Add length.
37                 previous = current # Current point becomes previous.
38         total.add(previous.distance(first)) # distance back to start
39         return total.result() # Return the perimeter.
40
41     def print(self) -> None:
42         """Print the points of this polygon."""
43         print(", ".join(f"({p.x}, {p.y})" for p in self.points()))
```

Beispiel: Die Klasse Circle

- Diese Methode liefert einen **Iterable** von Instanzen von **Point**, die die Eckpunkte des Polygons definieren.
- Unser Ziel ist es hier, eine Basisklasse für verschiedene Arten von Polygonen bereitzustellen.
- Wir wollen nicht direkt und sofort eine Datenstruktur für beliebige Polygone implementieren.
- Daher löst die Methode **points** auch einen **NotImplementedError** und muss von Subklassen implementiert werden.

```
1 """A polygon is a figure described by its corner points."""
2
3 from typing import Iterable
4
5 from kahan_sum import KahanSum
6 from point import Point
7 from shape import Shape
8
9
10 class Polygon(Shape):
11     """Polygons are shapes delimited by straight lines."""
12
13     def points(self) -> Iterable[Point]:
14         """
15             Get a :class:`Iterable` over the points describing this polygon.
16
17             :return: the points describing the polygon
18             :raises NotImplementedError: Must be implemented by subclasses.
19         """
20
21         raise NotImplementedError # must be implemented by subclasses
22
23     def perimeter(self) -> int | float:
24         """
25             Get the perimeter of this polygon.
26
27             :return: the perimeter of this polygon
28         """
29
30         previous: Point | None = None # the previous point
31         first: Point | None = None # the first point
32         total: KahanSum = KahanSum() # the total perimeter length sum
33         for current in self.points():
34             # Iterate over the points.
35             if previous is None: # We got the first point.
36                 previous = first = current # Remember it for last step.
37             else: # We now have previous != None, so we can add length.
38                 total.add(previous.distance(current)) # Add length.
39                 previous = current # Current point becomes previous.
40         total.add(previous.distance(first)) # distance back to start
41         return total.result() # Return the perimeter.
42
43     def print(self) -> None:
44         """Print the points of this polygon."""
45         print(", ".join(f"({p.x}, {p.y})" for p in self.points()))
```

Beispiel: Die Klasse Circle

- Unser Ziel ist es hier, eine Basisklasse für verschiedene Arten von Polygonen bereitzustellen.
- Wir wollen nicht direkt und sofort eine Datenstruktur für beliebige Polygone implementieren.
- Daher löst die Methode `points` auch einen `NotImplementedError` und muss von Subklassen implementiert werden.
- Wir können also annehmen, dass alle nicht-abstrakten Subklassen von `Polygon` immer die Methode `points` vernünftig implementieren und diese die Sequenz von Eckpunkten des Polygons liefert.

```
1 """A polygon is a figure described by its corner points."""
2
3 from typing import Iterable
4
5 from kahan_sum import KahanSum
6 from point import Point
7 from shape import Shape
8
9
10 class Polygon(Shape):
11     """Polygons are shapes delimited by straight lines."""
12
13     def points(self) -> Iterable[Point]:
14         """
15             Get a :class:`Iterable` over the points describing this polygon.
16
17             :return: the points describing the polygon
18             :raises NotImplementedError: Must be implemented by subclasses.
19         """
20
21         raise NotImplementedError # must be implemented by subclasses
22
23     def perimeter(self) -> int | float:
24         """
25             Get the perimeter of this polygon.
26
27             :return: the perimeter of this polygon
28         """
29
30         previous: Point | None = None # the previous point
31         first: Point | None = None # the first point
32         total: KahanSum = KahanSum() # the total perimeter length sum
33         for current in self.points():
34             # Iterate over the points.
35             if previous is None: # We got the first point.
36                 previous = first = current # Remember it for last step.
37             else: # We now have previous != None, so we can add length.
38                 total.add(previous.distance(current)) # Add length.
39                 previous = current # Current point becomes previous.
40         total.add(previous.distance(first)) # distance back to start
41         return total.result() # Return the perimeter.
42
43     def print(self) -> None:
44         """Print the points of this polygon."""
45         print(", ".join(f"({p.x}, {p.y})" for p in self.points()))
```

Beispiel: Die Klasse Circle

- Daher löst die Methode `points` auch einen `NotImplementedError` und muss von Subklassen implementiert werden.
- Wir können also annehmen, dass alle nicht-abstrakten Subklassen von `Polygon` immer die Methode `points` vernünftig implementieren und diese die Sequenz von Eckpunkten des Polygons liefert.
- Wenn wir die Sequenz der Eckpunkte kennen und diese mit geraden Linien verbinden sind, dann können wir den Umfang leicht berechnen.

```
1 """A polygon is a figure described by its corner points."""
2
3 from typing import Iterable
4
5 from kahan_sum import KahanSum
6 from point import Point
7 from shape import Shape
8
9
10 class Polygon(Shape):
11     """Polygons are shapes delimited by straight lines."""
12
13     def points(self) -> Iterable[Point]:
14         """
15             Get a :class:`Iterable` over the points describing this polygon.
16
17             :return: the points describing the polygon
18             :raises NotImplementedError: Must be implemented by subclasses.
19         """
20         raise NotImplementedError # must be implemented by subclasses
21
22     def perimeter(self) -> int | float:
23         """
24             Get the perimeter of this polygon.
25
26             :return: the perimeter of this polygon
27         """
28         previous: Point | None = None # the previous point
29         first: Point | None = None # the first point
30         total: KahanSum = KahanSum() # the total perimeter length sum
31         for current in self.points():
32             # Iterate over the points.
33             if previous is None: # We got the first point.
34                 previous = first = current # Remember it for last step.
35             else: # We now have previous != None, so we can add length.
36                 total.add(previous.distance(current)) # Add length.
37                 previous = current # Current point becomes previous.
38         total.add(previous.distance(first)) # distance back to start
39         return total.result() # Return the perimeter.
40
41     def print(self) -> None:
42         """Print the points of this polygon."""
43         print(", ".join(f"({p.x}, {p.y})" for p in self.points()))
```

Beispiel: Die Klasse Circle

- Daher löst die Methode `points` auch einen `NotImplementedError` und muss von Subklassen implementiert werden.
- Wir können also annehmen, dass alle nicht-abstrakten Subklassen von `Polygon` immer die Methode `points` vernünftig implementieren und diese die Sequenz von Eckpunkten des Polygons liefert.
- Wenn wir die Sequenz der Eckpunkte kennen und diese mit geraden Linien verbinden sind, dann können wir den Umfang leicht berechnen.
- Wir können daher die Methode `perimeter` implementieren.

```
1 """A polygon is a figure described by its corner points."""
2
3 from typing import Iterable
4
5 from kahan_sum import KahanSum
6 from point import Point
7 from shape import Shape
8
9
10 class Polygon(Shape):
11     """Polygons are shapes delimited by straight lines."""
12
13     def points(self) -> Iterable[Point]:
14         """
15             Get a :class:`Iterable` over the points describing this polygon.
16
17             :return: the points describing the polygon
18             :raises NotImplementedError: Must be implemented by subclasses.
19         """
20         raise NotImplementedError # must be implemented by subclasses
21
22     def perimeter(self) -> int | float:
23         """
24             Get the perimeter of this polygon.
25
26             :return: the perimeter of this polygon
27         """
28         previous: Point | None = None # the previous point
29         first: Point | None = None # the first point
30         total: KahanSum = KahanSum() # the total perimeter length sum
31         for current in self.points():
32             # Iterate over the points.
33             if previous is None: # We got the first point.
34                 previous = first = current # Remember it for last step.
35             else: # We now have previous != None, so we can add length.
36                 total.add(previous.distance(current)) # Add length.
37                 previous = current # Current point becomes previous.
38         total.add(previous.distance(first)) # distance back to start
39         return total.result() # Return the perimeter.
40
41     def print(self) -> None:
42         """Print the points of this polygon."""
43         print(", ".join(f"({p.x}, {p.y})" for p in self.points()))
```

Beispiel: Die Klasse Circle

- Wir können also annehmen, dass alle nicht-abstrakten Subklassen von `Polygon` immer die Methode `points` vernünftig implementieren und diese die Sequenz von Eckpunkten des Polygons liefert.
- Wenn wir die Sequenz der Eckpunkte kennen und diese mit geraden Linien verbinden sind, dann können wir den Umfang leicht berechnen.
- Wir können daher die Methode `perimeter` implementieren.
- Wir iterieren über die Instanzen von `Point`, die `points()` uns liefert.

```
1 """A polygon is a figure described by its corner points."""
2
3 from typing import Iterable
4
5 from kahan_sum import KahanSum
6 from point import Point
7 from shape import Shape
8
9
10 class Polygon(Shape):
11     """Polygons are shapes delimited by straight lines."""
12
13     def points(self) -> Iterable[Point]:
14         """
15             Get a :class:`Iterable` over the points describing this polygon.
16
17             :return: the points describing the polygon
18             :raises NotImplementedError: Must be implemented by subclasses.
19         """
20         raise NotImplementedError # must be implemented by subclasses
21
22     def perimeter(self) -> int | float:
23         """
24             Get the perimeter of this polygon.
25
26             :return: the perimeter of this polygon
27         """
28         previous: Point | None = None # the previous point
29         first: Point | None = None # the first point
30         total: KahanSum = KahanSum() # the total perimeter length sum
31         for current in self.points():
32             # Iterate over the points.
33             if previous is None: # We got the first point.
34                 previous = first = current # Remember it for last step.
35             else: # We now have previous != None, so we can add length.
36                 total.add(previous.distance(current)) # Add length.
37                 previous = current # Current point becomes previous.
38         total.add(previous.distance(first)) # distance back to start
39         return total.result() # Return the perimeter.
40
41     def print(self) -> None:
42         """Print the points of this polygon."""
43         print(", ".join(f"({p.x}, {p.y})" for p in self.points()))
```

Beispiel: Die Klasse Circle

- Wenn wir die Sequenz der Eckpunkte kennen und diese mit geraden Linien verbinden sind, dann können wir den Umfang leicht berechnen.
- Wir können daher die Methode `perimeter` implementieren.
- Wir iterieren über die Instanzen von `Point`, die `points()` uns liefert.
- In einer Summenvariable addieren wie Entfernung von jedem Punkt zu seinem Nachfolger in der Sequenz.

```
1 """A polygon is a figure described by its corner points."""
2
3 from typing import Iterable
4
5 from kahan_sum import KahanSum
6 from point import Point
7 from shape import Shape
8
9
10 class Polygon(Shape):
11     """Polygons are shapes delimited by straight lines."""
12
13     def points(self) -> Iterable[Point]:
14         """
15             Get a :class:`Iterable` over the points describing this polygon.
16
17             :return: the points describing the polygon
18             :raises NotImplementedError: Must be implemented by subclasses.
19         """
20
21         raise NotImplementedError # must be implemented by subclasses
22
23     def perimeter(self) -> int | float:
24         """
25             Get the perimeter of this polygon.
26
27             :return: the perimeter of this polygon
28         """
29
30         previous: Point | None = None # the previous point
31         first: Point | None = None # the first point
32         total: KahanSum = KahanSum() # the total perimeter length sum
33         for current in self.points():
34             # Iterate over the points.
35             if previous is None: # We got the first point.
36                 previous = first = current # Remember it for last step.
37             else: # We now have previous != None, so we can add length.
38                 total.add(previous.distance(current)) # Add length.
39                 previous = current # Current point becomes previous.
40         total.add(previous.distance(first)) # distance back to start
41         return total.result() # Return the perimeter.
42
43     def print(self) -> None:
44         """Print the points of this polygon."""
45         print(", ".join(f"({p.x}, {p.y})" for p in self.points()))
```

Beispiel: Die Klasse Circle

- Wenn wir die Sequenz der Eckpunkte kennen und diese mit geraden Linien verbinden sind, dann können wir den Umfang leicht berechnen.
- Wir können daher die Methode `perimeter` implementieren.
- Wir iterieren über die Instanzen von `Point`, die `points()` uns liefert.
- In einer Summenvariable addieren wie Entfernung von jedem Punkt zu seinem Nachfolger in der Sequenz.
- Am Ende addieren wir noch die Entfernung vom letzten Punkt zum ersten Punkt.

```
1 """A polygon is a figure described by its corner points."""
2
3 from typing import Iterable
4
5 from kahan_sum import KahanSum
6 from point import Point
7 from shape import Shape
8
9
10 class Polygon(Shape):
11     """Polygons are shapes delimited by straight lines."""
12
13     def points(self) -> Iterable[Point]:
14         """
15             Get a :class:`Iterable` over the points describing this polygon.
16
17             :return: the points describing the polygon
18             :raises NotImplementedError: Must be implemented by subclasses.
19         """
20
21         raise NotImplementedError # must be implemented by subclasses
22
23     def perimeter(self) -> int | float:
24         """
25             Get the perimeter of this polygon.
26
27             :return: the perimeter of this polygon
28         """
29
30         previous: Point | None = None # the previous point
31         first: Point | None = None # the first point
32         total: KahanSum = KahanSum() # the total perimeter length sum
33         for current in self.points():
34             # Iterate over the points.
35             if previous is None: # We got the first point.
36                 previous = first = current # Remember it for last step.
37             else: # We now have previous != None, so we can add length.
38                 total.add(previous.distance(current)) # Add length.
39                 previous = current # Current point becomes previous.
40         total.add(previous.distance(first)) # distance back to start
41         return total.result() # Return the perimeter.
42
43     def print(self) -> None:
44         """Print the points of this polygon."""
45         print(", ".join(f"({p.x}, {p.y})" for p in self.points()))
```

Beispiel: Die Klasse Circle

- Wir können daher die Methode `perimeter` implementieren.
- Wir iterieren über die Instanzen von `Point`, die `points()` uns liefert.
- In einer Summenvariable addieren wie Entfernung von jedem Punkt zu seinem Nachfolger in der Sequenz.
- Am Ende addieren wir noch die Entfernung vom letzten Punkt zum ersten Punkt.
- Die Entfernungen können mit der Methode `distance` der `Point`-Klasse berechnet werden.

```
1 """A polygon is a figure described by its corner points."""
2
3 from typing import Iterable
4
5 from kahan_sum import KahanSum
6 from point import Point
7 from shape import Shape
8
9
10 class Polygon(Shape):
11     """Polygons are shapes delimited by straight lines."""
12
13     def points(self) -> Iterable[Point]:
14         """
15             Get a :class:`Iterable` over the points describing this polygon.
16
17             :return: the points describing the polygon
18             :raises NotImplementedError: Must be implemented by subclasses.
19         """
20         raise NotImplementedError # must be implemented by subclasses
21
22     def perimeter(self) -> int | float:
23         """
24             Get the perimeter of this polygon.
25
26             :return: the perimeter of this polygon
27         """
28         previous: Point | None = None # the previous point
29         first: Point | None = None # the first point
30         total: KahanSum = KahanSum() # the total perimeter length sum
31         for current in self.points():
32             # Iterate over the points.
33             if previous is None: # We got the first point.
34                 previous = first = current # Remember it for last step.
35             else: # We now have previous != None, so we can add length.
36                 total.add(previous.distance(current)) # Add length.
37                 previous = current # Current point becomes previous.
38         total.add(previous.distance(first)) # distance back to start
39         return total.result() # Return the perimeter.
40
41     def print(self) -> None:
42         """Print the points of this polygon."""
43         print(", ".join(f"({p.x}, {p.y})" for p in self.points()))
```

Beispiel: Die Klasse Circle

- Wir iterieren über die Instanzen von `Point`, die `points()` uns liefert.
- In einer Summenvariable addieren wie Entfernung von jedem Punkt zu seinem Nachfolger in der Sequenz.
- Am Ende addieren wir noch die Entfernung vom letzten Punkt zum ersten Punkt.
- Die Entfernungen können mit der Methode `distance` der `Point`-Klasse berechnet werden.
- Nun implementiert `Polygon` selbst nicht die Methode `points`.

```
1 """A polygon is a figure described by its corner points."""
2
3 from typing import Iterable
4
5 from kahan_sum import KahanSum
6 from point import Point
7 from shape import Shape
8
9
10 class Polygon(Shape):
11     """Polygons are shapes delimited by straight lines."""
12
13     def points(self) -> Iterable[Point]:
14         """
15             Get a :class:`Iterable` over the points describing this polygon.
16
17             :return: the points describing the polygon
18             :raises NotImplementedError: Must be implemented by subclasses.
19         """
20         raise NotImplementedError # must be implemented by subclasses
21
22     def perimeter(self) -> int | float:
23         """
24             Get the perimeter of this polygon.
25
26             :return: the perimeter of this polygon
27         """
28         previous: Point | None = None # the previous point
29         first: Point | None = None # the first point
30         total: KahanSum = KahanSum() # the total perimeter length sum
31         for current in self.points():
32             # Iterate over the points.
33             if previous is None: # We got the first point.
34                 previous = first = current # Remember it for last step.
35             else: # We now have previous != None, so we can add length.
36                 total.add(previous.distance(current)) # Add length.
37                 previous = current # Current point becomes previous.
38         total.add(previous.distance(first)) # distance back to start
39         return total.result() # Return the perimeter.
40
41     def print(self) -> None:
42         """Print the points of this polygon."""
43         print(", ".join(f"({p.x}, {p.y})" for p in self.points()))
```

Beispiel: Die Klasse Circle

- In einer Summenvariable addieren wie Entfernung von jedem Punkt zu seinem Nachfolger in der Sequenz.
- Am Ende addieren wir noch die Entfernung vom letzten Punkt zum ersten Punkt.
- Die Entfernungen können mit der Methode `distance` der `Point`-Klasse berechnet werden.
- Nun implementiert `Polygon` selbst nicht die Methode `points`.
- Aber ihre Subklassen werden das tun.

```
1 """A polygon is a figure described by its corner points."""
2
3 from typing import Iterable
4
5 from kahan_sum import KahanSum
6 from point import Point
7 from shape import Shape
8
9
10 class Polygon(Shape):
11     """Polygons are shapes delimited by straight lines."""
12
13     def points(self) -> Iterable[Point]:
14         """
15             Get a :class:`Iterable` over the points describing this polygon.
16
17             :return: the points describing the polygon
18             :raises NotImplementedError: Must be implemented by subclasses.
19         """
20
21         raise NotImplementedError # must be implemented by subclasses
22
23     def perimeter(self) -> int | float:
24         """
25             Get the perimeter of this polygon.
26
27             :return: the perimeter of this polygon
28         """
29
30         previous: Point | None = None # the previous point
31         first: Point | None = None # the first point
32         total: KahanSum = KahanSum() # the total perimeter length sum
33         for current in self.points():
34             # Iterate over the points.
35             if previous is None: # We got the first point.
36                 previous = first = current # Remember it for last step.
37             else: # We now have previous != None, so we can add length.
38                 total.add(previous.distance(current)) # Add length.
39                 previous = current # Current point becomes previous.
40         total.add(previous.distance(first)) # distance back to start
41         return total.result() # Return the perimeter.
42
43     def print(self) -> None:
44         """Print the points of this polygon."""
45         print(", ".join(f"({p.x}, {p.y})" for p in self.points()))
```

Beispiel: Die Klasse Circle

- Am Ende addieren wir noch die Entfernung vom letzten Punkt zum ersten Punkt.
- Die Entfernungen können mit der Methode `distance` der `Point`-Klasse berechnet werden.
- Nun implementiert `Polygon` selbst nicht die Methode `points`.
- Aber ihre Subklassen werden das tun.
- Wenn wir `a.perimeter()` aufrufen, dann wird *automatisch* die richtige Implementierung von `points()` der Subklasse, zu der `a` gehört verwendet.

```
1 """A polygon is a figure described by its corner points."""
2
3 from typing import Iterable
4
5 from kahan_sum import KahanSum
6 from point import Point
7 from shape import Shape
8
9
10 class Polygon(Shape):
11     """Polygons are shapes delimited by straight lines."""
12
13     def points(self) -> Iterable[Point]:
14         """
15             Get a :class:`Iterable` over the points describing this polygon.
16
17             :return: the points describing the polygon
18             :raises NotImplementedError: Must be implemented by subclasses.
19         """
20
21         raise NotImplementedError # must be implemented by subclasses
22
23     def perimeter(self) -> int | float:
24         """
25             Get the perimeter of this polygon.
26
27             :return: the perimeter of this polygon
28         """
29
30         previous: Point | None = None # the previous point
31         first: Point | None = None # the first point
32         total: KahanSum = KahanSum() # the total perimeter length sum
33         for current in self.points():
34             # Iterate over the points.
35             if previous is None: # We got the first point.
36                 previous = first = current # Remember it for last step.
37             else: # We now have previous != None, so we can add length.
38                 total.add(previous.distance(current)) # Add length.
39                 previous = current # Current point becomes previous.
40         total.add(previous.distance(first)) # distance back to start
41         return total.result() # Return the perimeter.
42
43     def print(self) -> None:
44         """Print the points of this polygon."""
45         print(", ".join(f"({p.x}, {p.y})" for p in self.points()))
```

Beispiel: Die Klasse Circle

- Die Entfernungen können mit der Methode `distance` der `Point`-Klasse berechnet werden.
- Nun implementiert `Polygon` selbst nicht die Methode `points`.
- Aber ihre Subklassen werden das tun.
- Wenn wir `a.perimeter()` aufrufen, dann wird *automatisch* die richtige Implementierung von `points()` der Subklasse, zu der `a` gehört verwendet.
- Wir können hier also `perimeter` implementieren, selbst wenn `points` noch nicht implementiert ist.

```
1 """A polygon is a figure described by its corner points."""
2
3 from typing import Iterable
4
5 from kahan_sum import KahanSum
6 from point import Point
7 from shape import Shape
8
9
10 class Polygon(Shape):
11     """Polygons are shapes delimited by straight lines."""
12
13     def points(self) -> Iterable[Point]:
14         """
15             Get a :class:`Iterable` over the points describing this polygon.
16
17             :return: the points describing the polygon
18             :raises NotImplementedError: Must be implemented by subclasses.
19         """
20         raise NotImplementedError # must be implemented by subclasses
21
22     def perimeter(self) -> int | float:
23         """
24             Get the perimeter of this polygon.
25
26             :return: the perimeter of this polygon
27         """
28         previous: Point | None = None # the previous point
29         first: Point | None = None # the first point
30         total: KahanSum = KahanSum() # the total perimeter length sum
31         for current in self.points():
32             # Iterate over the points.
33             if previous is None: # We got the first point.
34                 previous = first = current # Remember it for last step.
35             else: # We now have previous != None, so we can add length.
36                 total.add(previous.distance(current)) # Add length.
37                 previous = current # Current point becomes previous.
38         total.add(previous.distance(first)) # distance back to start
39         return total.result() # Return the perimeter.
40
41     def print(self) -> None:
42         """Print the points of this polygon."""
43         print(", ".join(f"({p.x}, {p.y})" for p in self.points()))
```

Beispiel: Die Klasse Circle

- Nun implementiert `Polygon` selbst nicht die Methode `points`.
- Aber ihre Subklassen werden das tun.
- Wenn wir `a.perimeter()` aufrufen, dann wird *automatisch* die richtige Implementierung von `points()` der Subklasse, zu der `a` gehört verwendet.
- Wir können hier also `perimeter` implementieren, selbst wenn `points` noch nicht implementiert ist.
- Wenn wir eine Subklasse erstellen, die `points` implementiert, und dann `perimeter` von einem Objekt der Subklasse aufrufen, dann wird das funktionieren.

```
1 """A polygon is a figure described by its corner points."""
2
3 from typing import Iterable
4
5 from kahan_sum import KahanSum
6 from point import Point
7 from shape import Shape
8
9
10 class Polygon(Shape):
11     """Polygons are shapes delimited by straight lines."""
12
13     def points(self) -> Iterable[Point]:
14         """
15             Get a :class:`Iterable` over the points describing this polygon.
16
17             :return: the points describing the polygon
18             :raises NotImplementedError: Must be implemented by subclasses.
19         """
20         raise NotImplementedError # must be implemented by subclasses
21
22     def perimeter(self) -> int | float:
23         """
24             Get the perimeter of this polygon.
25
26             :return: the perimeter of this polygon
27         """
28         previous: Point | None = None # the previous point
29         first: Point | None = None # the first point
30         total: KahanSum = KahanSum() # the total perimeter length sum
31         for current in self.points():
32             # Iterate over the points.
33             if previous is None: # We got the first point.
34                 previous = first = current # Remember it for last step.
35             else: # We now have previous != None, so we can add length.
36                 total.add(previous.distance(current)) # Add length.
37                 previous = current # Current point becomes previous.
38         total.add(previous.distance(first)) # distance back to start
39         return total.result() # Return the perimeter.
40
41     def print(self) -> None:
42         """Print the points of this polygon."""
43         print(", ".join(f"({p.x}, {p.y})" for p in self.points()))
```

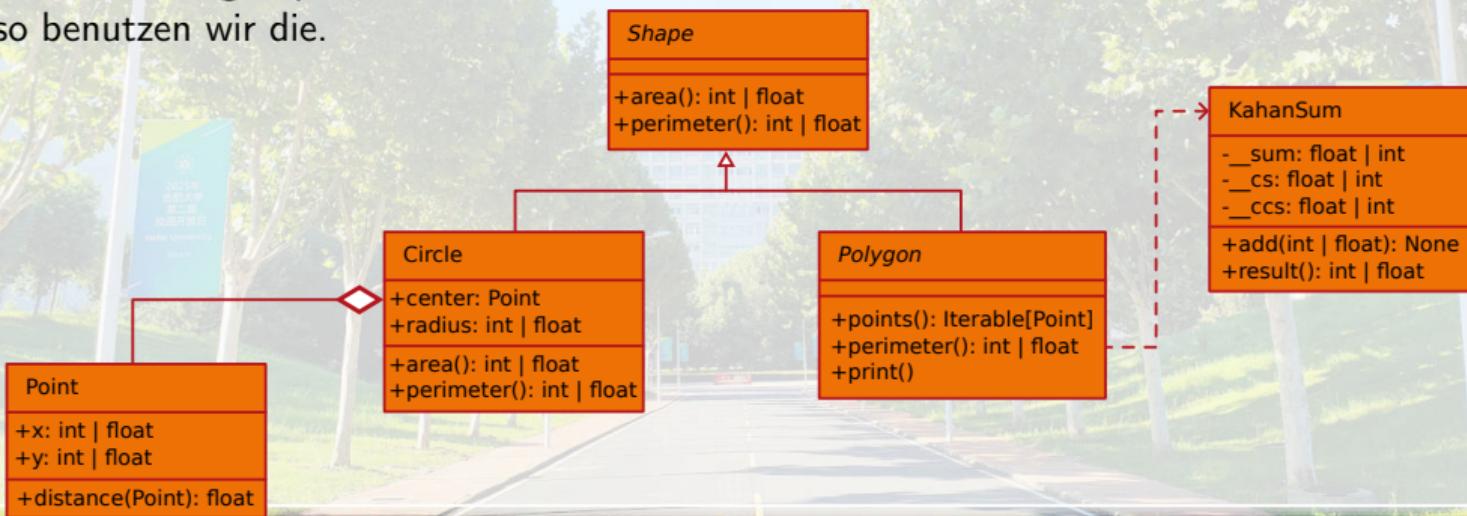
Beispiel: Die Klasse Circle

- Wenn wir `a.perimeter()` aufrufen, dann wird *automatisch* die richtige Implementierung von `points()` der Subklasse, zu der `a` gehört verwendet.
- Wir können hier also `perimeter` implementieren, selbst wenn `points` noch nicht implementiert ist.
- Wenn wir eine Subklasse erstellen, die `points` implementiert, und dann `perimeter` von einem Objekt der Subklasse aufrufen, dann wird das funktionieren.
- Das Aufsummieren der Entferungen könnte leicht mit einer normalen Variable vom Typ `float` gemacht werden.

```
1 """A polygon is a figure described by its corner points."""
2
3 from typing import Iterable
4
5 from kahan_sum import KahanSum
6 from point import Point
7 from shape import Shape
8
9
10 class Polygon(Shape):
11     """Polygons are shapes delimited by straight lines."""
12
13     def points(self) -> Iterable[Point]:
14         """
15             Get a :class:`Iterable` over the points describing this polygon.
16
17             :return: the points describing the polygon
18             :raises NotImplementedError: Must be implemented by subclasses.
19         """
20         raise NotImplementedError # must be implemented by subclasses
21
22     def perimeter(self) -> int | float:
23         """
24             Get the perimeter of this polygon.
25
26             :return: the perimeter of this polygon
27         """
28         previous: Point | None = None # the previous point
29         first: Point | None = None # the first point
30         total: KahanSum = KahanSum() # the total perimeter length sum
31         for current in self.points():
32             # Iterate over the points.
33             if previous is None: # We got the first point.
34                 previous = first = current # Remember it for last step.
35             else: # We now have previous != None, so we can add length.
36                 total.add(previous.distance(current)) # Add length.
37                 previous = current # Current point becomes previous.
38         total.add(previous.distance(first)) # distance back to start
39         return total.result() # Return the perimeter.
40
41     def print(self) -> None:
42         """Print the points of this polygon."""
43         print(", ".join(f"({p.x}, {p.y})" for p in self.points()))
```

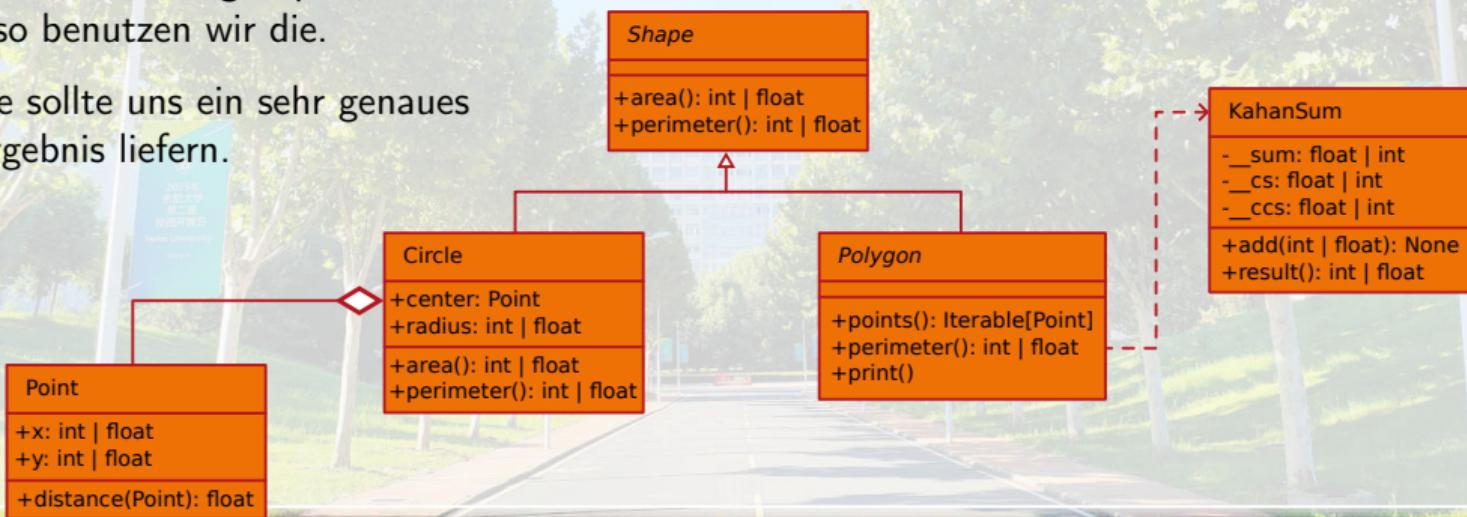
Beispiel: Die Klasse Circle

- Das Aufsummieren der Entfernungen könnte leicht mit einer normalen Variable vom Typ `float` gemacht werden.
- Wir haben aber schon die Kahan-Babuška-Neumaier-Summe zweiter Ordnung implementiert ... also benutzen wir die.



Beispiel: Die Klasse Circle

- Das Aufsummieren der Entfernungen könnte leicht mit einer normalen Variable vom Typ `float` gemacht werden.
- Wir haben aber schon die Kahan-Babuška-Neumaier-Summe zweiter Ordnung implementiert ... also benutzen wir die.
- Sie sollte uns ein sehr genaues Ergebnis liefern.



Beispiel: Die Klasse Circle

- Das Aufsummieren der Entfernungen könnte leicht mit einer normalen Variable vom Typ `float` gemacht werden.
- Wir haben aber schon die Kahan-Babuška-Neumaier-Summe zweiter Ordnung implementiert ... also benutzen wir die.
- Sie sollte uns ein sehr genaues Ergebnis liefern.
- Beachten Sie, dass wir hier `fsum` nicht verwenden können, ohne erst alle Entfernungen in einer Liste zu speichern.

```
1 """A polygon is a figure described by its corner points."""
2
3 from typing import Iterable
4
5 from kahan_sum import KahanSum
6 from point import Point
7 from shape import Shape
8
9
10 class Polygon(Shape):
11     """Polygons are shapes delimited by straight lines."""
12
13     def points(self) -> Iterable[Point]:
14         """
15             Get a :class:`Iterable` over the points describing this polygon.
16
17             :return: the points describing the polygon
18             :raises NotImplementedError: Must be implemented by subclasses.
19         """
20         raise NotImplementedError # must be implemented by subclasses
21
22     def perimeter(self) -> int | float:
23         """
24             Get the perimeter of this polygon.
25
26             :return: the perimeter of this polygon
27         """
28         previous: Point | None = None # the previous point
29         first: Point | None = None # the first point
30         total: KahanSum = KahanSum() # the total perimeter length sum
31         for current in self.points():
32             # Iterate over the points.
33             if previous is None: # We got the first point.
34                 previous = first = current # Remember it for last step.
35             else: # We now have previous != None, so we can add length.
36                 total.add(previous.distance(current)) # Add length.
37                 previous = current # Current point becomes previous.
38         total.add(previous.distance(first)) # distance back to start
39         return total.result() # Return the perimeter.
40
41     def print(self) -> None:
42         """Print the points of this polygon."""
43         print(", ".join(f"({p.x}, {p.y})" for p in self.points()))
```

Beispiel: Die Klasse Circle

- Wir haben aber schon die Kahan-Babuška-Neumaier-Summe zweiter Ordnung implementiert ... also benutzen wir die.
- Sie sollte uns ein sehr genaues Ergebnis liefern.
- Beachten Sie, dass wir hier `fsum` nicht verwenden können, ohne erst alle Entferungen in einer Liste zu speichern.
- Der Grund ist, dass wir über diese Entfernungen in der Sequenz von Punkten summieren müssen, aber auch die Entfernung vom ersten zum letzten Punkt brauchen.

```
1 """A polygon is a figure described by its corner points."""
2
3 from typing import Iterable
4
5 from kahan_sum import KahanSum
6 from point import Point
7 from shape import Shape
8
9
10 class Polygon(Shape):
11     """Polygons are shapes delimited by straight lines."""
12
13     def points(self) -> Iterable[Point]:
14         """
15             Get a :class:`Iterable` over the points describing this polygon.
16
17             :return: the points describing the polygon
18             :raises NotImplementedError: Must be implemented by subclasses.
19         """
20
21         raise NotImplementedError # must be implemented by subclasses
22
23     def perimeter(self) -> int | float:
24         """
25             Get the perimeter of this polygon.
26
27             :return: the perimeter of this polygon
28         """
29
30         previous: Point | None = None # the previous point
31         first: Point | None = None # the first point
32         total: KahanSum = KahanSum() # the total perimeter length sum
33         for current in self.points():
34             # Iterate over the points.
35             if previous is None: # We got the first point.
36                 previous = first = current # Remember it for last step.
37             else: # We now have previous != None, so we can add length.
38                 total.add(previous.distance(current)) # Add length.
39                 previous = current # Current point becomes previous.
40         total.add(previous.distance(first)) # distance back to start
41         return total.result() # Return the perimeter.
42
43     def print(self) -> None:
44         """Print the points of this polygon."""
45         print(", ".join(f"({p.x}, {p.y})" for p in self.points()))
```

Beispiel: Die Klasse Circle

- Sie sollte uns ein sehr genaues Ergebnis liefern.
- Beachten Sie, dass wir hier `fsum` nicht verwenden können, ohne erst alle Entfernungen in einer Liste zu speichern.
- Der Gund ist, dass wir über diese Entfernungen in der Sequenz von Punkten summieren müssen, aber auch die Entfernung vom ersten zum letzten Punkt brauchen.
- Unsere `KahanSum` hat hier also doch einen Vorteil.

```
1 """A polygon is a figure described by its corner points."""
2
3 from typing import Iterable
4
5 from kahan_sum import KahanSum
6 from point import Point
7 from shape import Shape
8
9
10 class Polygon(Shape):
11     """Polygons are shapes delimited by straight lines."""
12
13     def points(self) -> Iterable[Point]:
14         """
15             Get a :class:`Iterable` over the points describing this polygon.
16
17             :return: the points describing the polygon
18             :raises NotImplementedError: Must be implemented by subclasses.
19         """
20
21         raise NotImplementedError # must be implemented by subclasses
22
23     def perimeter(self) -> int | float:
24         """
25             Get the perimeter of this polygon.
26
27             :return: the perimeter of this polygon
28         """
29
30         previous: Point | None = None # the previous point
31         first: Point | None = None # the first point
32         total: KahanSum = KahanSum() # the total perimeter length sum
33         for current in self.points():
34             # Iterate over the points.
35             if previous is None: # We got the first point.
36                 previous = first = current # Remember it for last step.
37             else: # We now have previous != None, so we can add length.
38                 total.add(previous.distance(current)) # Add length.
39                 previous = current # Current point becomes previous.
40         total.add(previous.distance(first)) # distance back to start
41         return total.result() # Return the perimeter.
42
43     def print(self) -> None:
44         """Print the points of this polygon."""
45         print(", ".join(f"({p.x}, {p.y})" for p in self.points()))
```

Beispiel: Die Klasse Circle

- Beachten Sie, dass wir hier `fsum` nicht verwenden können, ohne erst alle Entfernungen in einer Liste zu speichern.
- Der Grund ist, dass wir über diese Entfernungen in der Sequenz von Punkten summieren müssen, aber auch die Entfernung vom ersten zum letzten Punkt brauchen.
- Unsere `KahanSum` hat hier also doch einen Vorteil.
- Aus Bequemlichkeit fügen wir noch eine Methode `print` mit dazu, die einfach die Eckpunkte des Poligons ausgibt.

```
1 """A polygon is a figure described by its corner points."""
2
3 from typing import Iterable
4
5 from kahan_sum import KahanSum
6 from point import Point
7 from shape import Shape
8
9
10 class Polygon(Shape):
11     """Polygons are shapes delimited by straight lines."""
12
13     def points(self) -> Iterable[Point]:
14         """
15             Get a :class:`Iterable` over the points describing this polygon.
16
17             :return: the points describing the polygon
18             :raises NotImplementedError: Must be implemented by subclasses.
19         """
20         raise NotImplementedError # must be implemented by subclasses
21
22     def perimeter(self) -> int | float:
23         """
24             Get the perimeter of this polygon.
25
26             :return: the perimeter of this polygon
27         """
28         previous: Point | None = None # the previous point
29         first: Point | None = None # the first point
30         total: KahanSum = KahanSum() # the total perimeter length sum
31         for current in self.points():
32             # Iterate over the points.
33             if previous is None: # We got the first point.
34                 previous = first = current # Remember it for last step.
35             else: # We now have previous != None, so we can add length.
36                 total.add(previous.distance(current)) # Add length.
37                 previous = current # Current point becomes previous.
38         total.add(previous.distance(first)) # distance back to start
39         return total.result() # Return the perimeter.
40
41     def print(self) -> None:
42         """Print the points of this polygon."""
43         print(", ".join(f"({p.x}, {p.y})" for p in self.points()))
```

Beispiel: Die Klasse Circle

- Der Grund ist, dass wir über diese Entfernungen in der Sequenz von Punkten summieren müssen, aber auch die Entfernung vom ersten zum letzten Punkt brauchen.
- Unsere `KahanSum` hat hier also doch einen Vorteil.
- Aus Bequemlichkeit fügen wir noch eine Methode `print` mit dazu, die einfach die Eckpunkte des Poligons ausgibt.
- Wir haben gelernt, dass Subklassen neue Methoden und neues Verhalten einführen können.

```
1 """A polygon is a figure described by its corner points."""
2
3 from typing import Iterable
4
5 from kahan_sum import KahanSum
6 from point import Point
7 from shape import Shape
8
9
10 class Polygon(Shape):
11     """Polygons are shapes delimited by straight lines."""
12
13     def points(self) -> Iterable[Point]:
14         """
15             Get a :class:`Iterable` over the points describing this polygon.
16
17             :return: the points describing the polygon
18             :raises NotImplementedError: Must be implemented by subclasses.
19         """
20
21         raise NotImplementedError # must be implemented by subclasses
22
23     def perimeter(self) -> int | float:
24         """
25             Get the perimeter of this polygon.
26
27             :return: the perimeter of this polygon
28         """
29
30         previous: Point | None = None # the previous point
31         first: Point | None = None # the first point
32         total: KahanSum = KahanSum() # the total perimeter length sum
33         for current in self.points():
34             # Iterate over the points.
35             if previous is None: # We got the first point.
36                 previous = first = current # Remember it for last step.
37             else: # We now have previous != None, so we can add length.
38                 total.add(previous.distance(current)) # Add length.
39                 previous = current # Current point becomes previous.
40         total.add(previous.distance(first)) # distance back to start
41         return total.result() # Return the perimeter.
42
43     def print(self) -> None:
44         """Print the points of this polygon."""
45         print(", ".join(f"({p.x}, {p.y})" for p in self.points()))
```

Beispiel: Die Klasse Circle

- Unsere KahanSum hat hier also doch einen Vorteil.
- Aus Bequemlichkeit fügen wir noch eine Methode `print` mit dazu, die einfach die Eckpunkte des Poligons ausgibt.
- Wir haben gelernt, dass Subklassen neue Methoden und neues Verhalten einführen können.
- Wir haben auch gelernt, dass Subklassen selbst auch als abstrakte Basisklassen entworfen werden können.

```
1 """A polygon is a figure described by its corner points."""
2
3 from typing import Iterable
4
5 from kahan_sum import KahanSum
6 from point import Point
7 from shape import Shape
8
9
10 class Polygon(Shape):
11     """Polygons are shapes delimited by straight lines."""
12
13     def points(self) -> Iterable[Point]:
14         """
15             Get a :class:`Iterable` over the points describing this polygon.
16
17             :return: the points describing the polygon
18             :raises NotImplementedError: Must be implemented by subclasses.
19         """
20
21         raise NotImplementedError # must be implemented by subclasses
22
23     def perimeter(self) -> int | float:
24         """
25             Get the perimeter of this polygon.
26
27             :return: the perimeter of this polygon
28         """
29
30         previous: Point | None = None # the previous point
31         first: Point | None = None # the first point
32         total: KahanSum = KahanSum() # the total perimeter length sum
33         for current in self.points():
34             # Iterate over the points.
35             if previous is None: # We got the first point.
36                 previous = first = current # Remember it for last step.
37             else: # We now have previous != None, so we can add length.
38                 total.add(previous.distance(current)) # Add length.
39                 previous = current # Current point becomes previous.
40         total.add(previous.distance(first)) # distance back to start
41         return total.result() # Return the perimeter.
42
43     def print(self) -> None:
44         """Print the points of this polygon."""
45         print(", ".join(f"({p.x}, {p.y})" for p in self.points()))
```

Beispiel: Die Klasse Circle

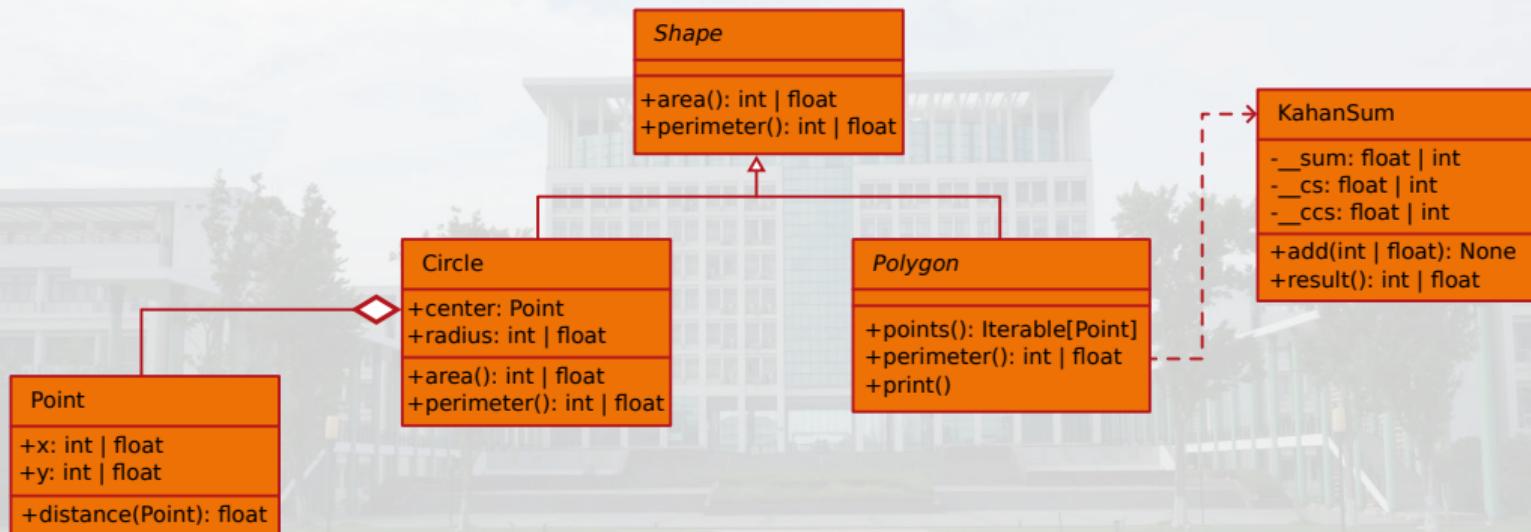
- Wir haben gelernt, dass Subklassen neue Methoden und neues Verhalten einführen können.
- Wir haben auch gelernt, dass Subklassen selbst auch als abstrakte Basisklassen entworfen werden können.
- Und wir haben gelernt, dass wenn wir `a.method()` aufrufen (oder `self.method()` in einer Methode), dann wird das immer die „letzte“ Implementierung der Methode `method` in der Klassenhierarchie, zu der Object `a` gehört, aufrufen.

```
1 """A polygon is a figure described by its corner points."""
2
3 from typing import Iterable
4
5 from kahan_sum import KahanSum
6 from point import Point
7 from shape import Shape
8
9
10 class Polygon(Shape):
11     """Polygons are shapes delimited by straight lines."""
12
13     def points(self) -> Iterable[Point]:
14         """
15             Get a :class:`Iterable` over the points describing this polygon.
16
17             :return: the points describing the polygon
18             :raises NotImplementedError: Must be implemented by subclasses.
19         """
20
21         raise NotImplementedError # must be implemented by subclasses
22
23     def perimeter(self) -> int | float:
24         """
25             Get the perimeter of this polygon.
26
27             :return: the perimeter of this polygon
28         """
29
30         previous: Point | None = None # the previous point
31         first: Point | None = None # the first point
32         total: KahanSum = KahanSum() # the total perimeter length sum
33         for current in self.points():
34             # Iterate over the points.
35             if previous is None: # We got the first point.
36                 previous = first = current # Remember it for last step.
37             else: # We now have previous != None, so we can add length.
38                 total.add(previous.distance(current)) # Add length.
39                 previous = current # Current point becomes previous.
40         total.add(previous.distance(first)) # distance back to start
41         return total.result() # Return the perimeter.
42
43     def print(self) -> None:
44         """Print the points of this polygon."""
45         print(", ".join(f"({p.x}, {p.y})" for p in self.points()))
```



Beispiel: Rechtecke und Dreiecke

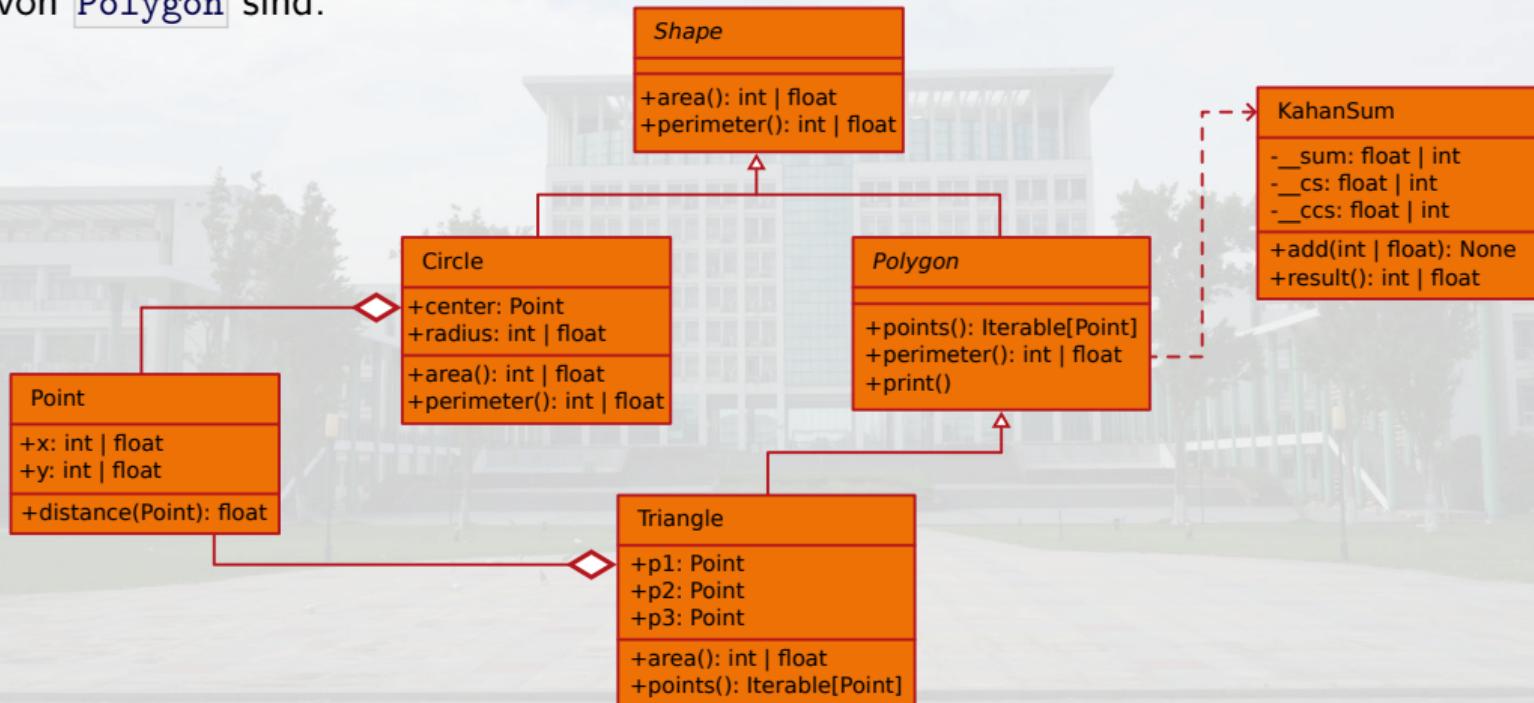
- Rechtecke und Dreiecke sind Sonderfälle von Polygonen.





Beispiel: Rechtecke und Dreiecke

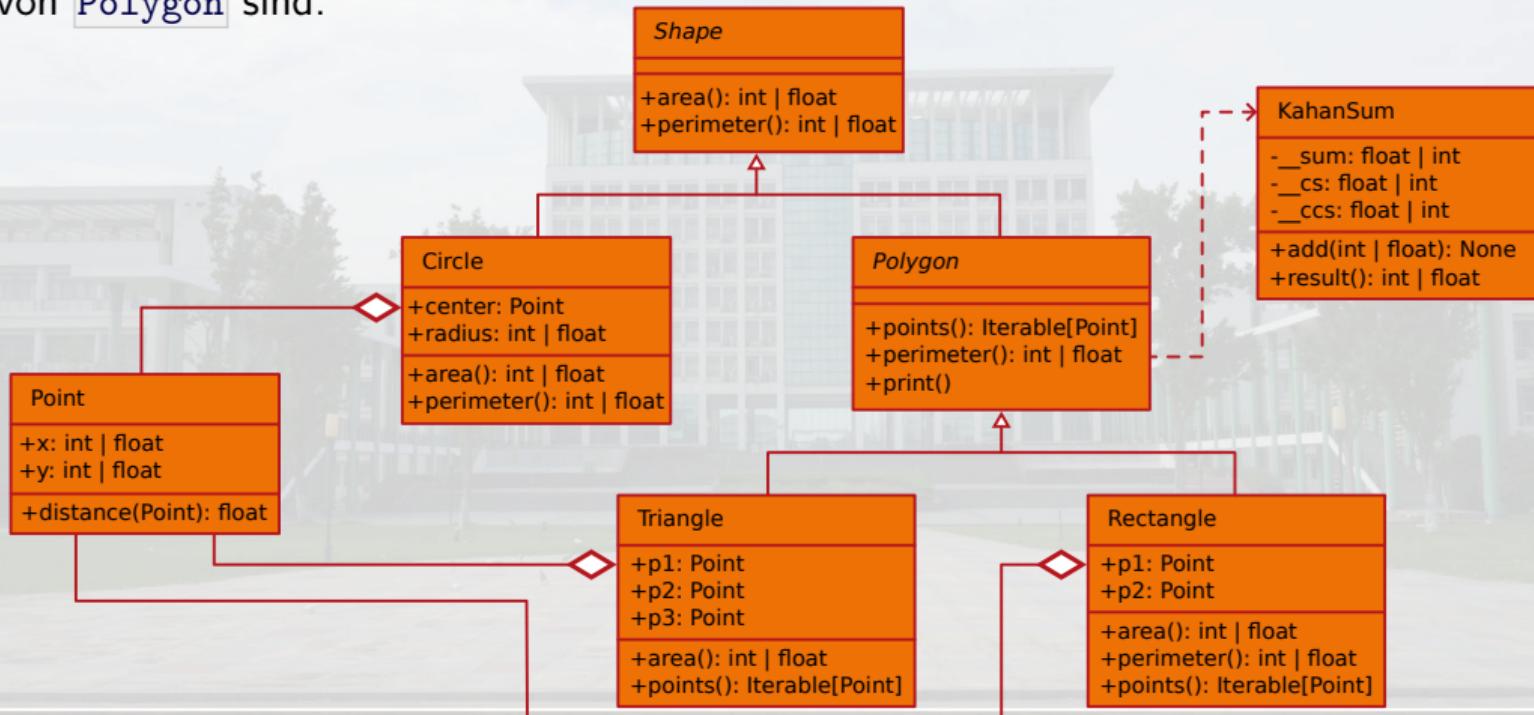
- Rechtecke und Dreiecke sind Sonderfälle von Polygonen.
- Wir implementieren sie als Klassen `Triangle` und `Rectangle`, welche beide Subklassen von `Polygon` sind.





Beispiel: Rechtecke und Dreiecke

- Rechtecke und Dreiecke sind Sonderfälle von Polygonen.
- Wir implementieren sie als Klassen `Triangle` und `Rectangle`, welche beide Subklassen von `Polygon` sind.



Beispiel: Die Klasse Rectangle

- Die Klasse `Rectangle` für Rechtecke wird in Datei `rectangle.py` erstellt.

```
1 """
2 A class for rectangles.
3
4 >>> Rectangle(Point(22, 1), Point(4, 12)).print()
5 (4, 1), (4, 12), (22, 12), (22, 1)
6 """
7
8 from typing import Final
9
10 from point import Point
11 from polygon import Polygon
12
13
14 class Rectangle(Polygon):
15     """A rectangle defined by its bottom-left and top-right corners."""
16
17     def __init__(self, p1: Point, p2: Point) -> None:
18         """
19             Create a rectangle.
20
21             :param p1: the first point spanning the rectangle
22             :param p2: the second point spanning the rectangle
23         """
24         if (p1.x == p2.x) or (p1.y == p2.y): # check for non-emptiness
25             raise ValueError(f"{p1}, {p1.y}, {p2.x}, {p2.y} is empty.")
26         #: the bottom-left point spanning the rectangle
27         self.p1: Final[Point] = Point(min(p1.x, p2.x), min(p1.y, p2.y))
28         #: the top-right point spanning the rectangle
29         self.p2: Final[Point] = Point(max(p1.x, p2.x), max(p1.y, p2.y))
30
31     def area(self) -> int | float:
32         """
33             Get the area of this rectangle.
34
35             :return: the area of this rectangle
36         """
37         >>> Rectangle(Point(7, 3), Point(12, 6)).area()
38         15
39         """
40         return (self.p2.x - self.p1.x) * (self.p2.y - self.p1.y)
41
42     def perimeter(self) -> int | float:
43         """
44             Get the perimeter of this rectangle.
45
46             :return: the perimeter of this rectangle
47         """
48         >>> Rectangle(Point(10, 5), Point(4, 9)).perimeter()
49         20
50         """
51         return 2 * ((self.p2.x - self.p1.x) + (self.p2.y - self.p1.y))
52
53     def points(self) -> tuple[Point, Point, Point, Point]:
54         """
55             Get the four corner points of this rectangle.
56
57             :return: a tuple with the four corners of this rectangle
58         """
59         return (self.p1, Point(self.p1.x, self.p2.y), self.p2,
60                 Point(self.p2.x, self.p1.y))
```



Beispiel: Die Klasse Rectangle

- Die Klasse `Rectangle` für Rechtecke wird in Datei `rectangle.py` erstellt.
- Ein Rechteck kann durch seine linke untere und rechte obere Ecke definiert werden.

```
12
13
14 class Rectangle(Polygon):
15     """A rectangle defined by its bottom-left and top-right corners."""
16
17     def __init__(self, p1: Point, p2: Point) -> None:
18         """
19             Create a rectangle.
20
21             :param p1: the first point spanning the rectangle
22             :param p2: the second point spanning the rectangle
23             """
24
25             if (p1.x == p2.x) or (p1.y == p2.y): # check for non-emptiness
26                 raise ValueError(f'{p1.x},{p1.y},{p2.x},{p2.y} is empty.')
27             #: the bottom-left point spanning the rectangle
28             self.p1: Final[Point] = Point(min(p1.x, p2.x), min(p1.y, p2.y))
29             #: the top-right point spanning the rectangle
30             self.p2: Final[Point] = Point(max(p1.x, p2.x), max(p1.y, p2.y))
31
32     def area(self) -> int | float:
33         """
34             Get the area of this rectangle.
35
36             :return: the area of this rectangle
37
38             >>> Rectangle(Point(7, 3), Point(12, 6)).area()
39             15
40             """
41
42             return (self.p2.x - self.p1.x) * (self.p2.y - self.p1.y)
43
44     def perimeter(self) -> int | float:
45         """
46             Get the perimeter of this rectangle.
47
48             :return: the perimeter of this rectangle
49
50             >>> Rectangle(Point(10, 5), Point(4, 9)).perimeter()
51             20
52             """
53
54             return 2 * ((self.p2.x - self.p1.x) + (self.p2.y - self.p1.y))
55
56     def points(self) -> tuple[Point, Point, Point, Point]:
57         """
58             Get the four corner points of this rectangle.
59
60             :return: a tuple with the four corners of this rectangle
61
62             return (self.p1, Point(self.p1.x, self.p2.y), self.p2,
63                     Point(self.p2.x, self.p1.y))
```

Beispiel: Die Klasse Rectangle

- Die Klasse `Rectangle` für Rechtecke wird in Datei `rectangle.py` erstellt.
- Ein Rechteck kann durch seine linke untere und rechte obere Ecke definiert werden.
- Der Initialisierer `__init__` der Klasse `Rectangle` akzeptiert daher zwei Punkte `p1` und `p2` als Parameter.

```
12
13
14 class Rectangle(Polygon):
15     """A rectangle defined by its bottom-left and top-right corners."""
16
17     def __init__(self, p1: Point, p2: Point) -> None:
18         """
19             Create a rectangle.
20
21             :param p1: the first point spanning the rectangle
22             :param p2: the second point spanning the rectangle
23             """
24
25             if (p1.x == p2.x) or (p1.y == p2.y): # check for non-emptiness
26                 raise ValueError(f'{p1.x},{p1.y},{p2.x},{p2.y} is empty.')
27             #: the bottom-left point spanning the rectangle
28             self.p1: Final[Point] = Point(min(p1.x, p2.x), min(p1.y, p2.y))
29             #: the top-right point spanning the rectangle
30             self.p2: Final[Point] = Point(max(p1.x, p2.x), max(p1.y, p2.y))
31
32     def area(self) -> int | float:
33         """
34             Get the area of this rectangle.
35
36             :return: the area of this rectangle
37
38             >>> Rectangle(Point(7, 3), Point(12, 6)).area()
39             15
40             """
41
42             return (self.p2.x - self.p1.x) * (self.p2.y - self.p1.y)
43
44     def perimeter(self) -> int | float:
45         """
46             Get the perimeter of this rectangle.
47
48             :return: the perimeter of this rectangle
49
50             >>> Rectangle(Point(10, 5), Point(4, 9)).perimeter()
51             20
52             """
53
54             return 2 * ((self.p2.x - self.p1.x) + (self.p2.y - self.p1.y))
55
56     def points(self) -> tuple[Point, Point, Point, Point]:
57         """
58             Get the four corner points of this rectangle.
59
60             :return: a tuple with the four corners of this rectangle
61
62             return (self.p1, Point(self.p1.x, self.p2.y), self.p2,
63                     Point(self.p2.x, self.p1.y))
```

Beispiel: Die Klasse Rectangle

- Die Klasse `Rectangle` für Rechtecke wird in Datei `rectangle.py` erstellt.
- Ein Rechteck kann durch seine linke untere und rechte obere Ecke definiert werden.
- Der Initialisierer `__init__` der Klasse `Rectangle` akzeptiert daher zwei Punkte `p1` und `p2` als Parameter.
- Wir lösen einen `ValueError` aus, wenn diese gleich sind und kein Rechteck ergeben.

```
12
13
14 class Rectangle(Polygon):
15     """A rectangle defined by its bottom-left and top-right corners."""
16
17     def __init__(self, p1: Point, p2: Point) -> None:
18         """
19             Create a rectangle.
20
21             :param p1: the first point spanning the rectangle
22             :param p2: the second point spanning the rectangle
23             """
24
25             if (p1.x == p2.x) or (p1.y == p2.y): # check for non-emptiness
26                 raise ValueError(f'{p1.x},{p1.y},{p2.x},{p2.y} is empty.')
27             #: the bottom-left point spanning the rectangle
28             self.p1: Final[Point] = Point(min(p1.x, p2.x), min(p1.y, p2.y))
29             #: the top-right point spanning the rectangle
30             self.p2: Final[Point] = Point(max(p1.x, p2.x), max(p1.y, p2.y))
31
32     def area(self) -> int | float:
33         """
34             Get the area of this rectangle.
35
36             :return: the area of this rectangle
37
38             >>> Rectangle(Point(7, 3), Point(12, 6)).area()
39             15
40             """
41
42             return (self.p2.x - self.p1.x) * (self.p2.y - self.p1.y)
43
44     def perimeter(self) -> int | float:
45         """
46             Get the perimeter of this rectangle.
47
48             :return: the perimeter of this rectangle
49
50             >>> Rectangle(Point(10, 5), Point(4, 9)).perimeter()
51             20
52             """
53
54             return 2 * ((self.p2.x - self.p1.x) + (self.p2.y - self.p1.y))
55
56     def points(self) -> tuple[Point, Point, Point, Point]:
57         """
58             Get the four corner points of this rectangle.
59
60             :return: a tuple with the four corners of this rectangle
61
62             return (self.p1, Point(self.p1.x, self.p2.y), self.p2,
63                     Point(self.p2.x, self.p1.y))
```

Beispiel: Die Klasse Rectangle

- Die Klasse `Rectangle` für Rechtecke wird in Datei `rectangle.py` erstellt.
- Ein Rechteck kann durch seine linke untere und rechte obere Ecke definiert werden.
- Der Initialisierer `__init__` der Klasse `Rectangle` akzeptiert daher zwei Punkte `p1` und `p2` als Parameter.
- Wir lösen einen `ValueError` aus, wenn diese gleich sind und kein Rechteck ergeben.
- Andernfalls speichern wir die minimalen x- und y-Koordinaten im „links-unten“ Punkt-Attribut `p1`.

```
12
13
14 class Rectangle(Polygon):
15     """A rectangle defined by its bottom-left and top-right corners."""
16
17     def __init__(self, p1: Point, p2: Point) -> None:
18         """
19             Create a rectangle.
20
21             :param p1: the first point spanning the rectangle
22             :param p2: the second point spanning the rectangle
23             """
24
25             if (p1.x == p2.x) or (p1.y == p2.y): # check for non-emptiness
26                 raise ValueError(f'{p1.x},{p1.y},{p2.x},{p2.y} is empty.')
27             #: the bottom-left point spanning the rectangle
28             self.p1: Final[Point] = Point(min(p1.x, p2.x), min(p1.y, p2.y))
29             #: the top-right point spanning the rectangle
30             self.p2: Final[Point] = Point(max(p1.x, p2.x), max(p1.y, p2.y))
31
32     def area(self) -> int | float:
33         """
34             Get the area of this rectangle.
35
36             :return: the area of this rectangle
37
38             >>> Rectangle(Point(7, 3), Point(12, 6)).area()
39             15
40             """
41
42             return (self.p2.x - self.p1.x) * (self.p2.y - self.p1.y)
43
44     def perimeter(self) -> int | float:
45         """
46             Get the perimeter of this rectangle.
47
48             :return: the perimeter of this rectangle
49
50             >>> Rectangle(Point(10, 5), Point(4, 9)).perimeter()
51             20
52             """
53
54             return 2 * ((self.p2.x - self.p1.x) + (self.p2.y - self.p1.y))
55
56     def points(self) -> tuple[Point, Point, Point, Point]:
57         """
58             Get the four corner points of this rectangle.
59
60             :return: a tuple with the four corners of this rectangle
61
62             return (self.p1, Point(self.p1.x, self.p2.y), self.p2,
63                     Point(self.p2.x, self.p1.y))
```

Beispiel: Die Klasse Rectangle

- Ein Rechteck kann durch seine linke untere und rechte obere Ecke definiert werden.
- Der Initialisierer `__init__` der Klasse `Rectangle` akzeptiert daher zwei Punkte `p1` und `p2` als Parameter.
- Wir lösen einen `ValueError` aus, wenn diese gleich sind und kein Rechteck ergeben.
- Andernfalls speichern wir die minimalen x- und y-Koordinaten im „links-unten“ Punkt-Attribut `p1`.
- Wir speichern die maximale x- und y-Koordinaten im Attribut `p2`, was den Punkt oben rechts markiert.

```
12
13
14 class Rectangle(Polygon):
15     """A rectangle defined by its bottom-left and top-right corners."""
16
17     def __init__(self, p1: Point, p2: Point) -> None:
18         """
19             Create a rectangle.
20
21             :param p1: the first point spanning the rectangle
22             :param p2: the second point spanning the rectangle
23             """
24
25             if (p1.x == p2.x) or (p1.y == p2.y): # check for non-emptiness
26                 raise ValueError(f'{p1.x},{p1.y},{p2.x},{p2.y} is empty.')
27             #: the bottom-left point spanning the rectangle
28             self.p1: Final[Point] = Point(min(p1.x, p2.x), min(p1.y, p2.y))
29             #: the top-right point spanning the rectangle
30             self.p2: Final[Point] = Point(max(p1.x, p2.x), max(p1.y, p2.y))
31
32     def area(self) -> int | float:
33         """
34             Get the area of this rectangle.
35
36             :return: the area of this rectangle
37
38             >>> Rectangle(Point(7, 3), Point(12, 6)).area()
39             15
40             """
41
42             return (self.p2.x - self.p1.x) * (self.p2.y - self.p1.y)
43
44     def perimeter(self) -> int | float:
45         """
46             Get the perimeter of this rectangle.
47
48             :return: the perimeter of this rectangle
49
50             >>> Rectangle(Point(10, 5), Point(4, 9)).perimeter()
51             20
52             """
53
54             return 2 * ((self.p2.x - self.p1.x) + (self.p2.y - self.p1.y))
55
56     def points(self) -> tuple[Point, Point, Point, Point]:
57         """
58             Get the four corner points of this rectangle.
59
60             :return: a tuple with the four corners of this rectangle
61
62             return (self.p1, Point(self.p1.x, self.p2.y), self.p2,
63                     Point(self.p2.x, self.p1.y))
```

Beispiel: Die Klasse Rectangle

- Der Initialisierer `__init__` der Klasse `Rectangle` akzeptiert daher zwei Punkte `p1` und `p2` als Parameter.
- Wir lösen einen `ValueError` aus, wenn diese gleich sind und kein Rechteck ergeben.
- Andernfalls speichern wir die minimalen x- und y-Koordinaten im „links-unten“ Punkt-Attribut `p1`.
- Wir speichern die maximale x- und y-Koordinaten im Attribut `p2`, was den Punkt oben rechts markiert.
- Wir können nun die Methode `points` so implementieren, dass sie alle vier Eckpunkte des Rechtecks zurückliefert.

```
12
13
14 class Rectangle(Polygon):
15     """A rectangle defined by its bottom-left and top-right corners."""
16
17     def __init__(self, p1: Point, p2: Point) -> None:
18         """
19             Create a rectangle.
20
21             :param p1: the first point spanning the rectangle
22             :param p2: the second point spanning the rectangle
23             """
24
25             if (p1.x == p2.x) or (p1.y == p2.y): # check for non-emptiness
26                 raise ValueError(f'{p1.x},{p1.y},{p2.x},{p2.y} is empty.')
27             #: the bottom-left point spanning the rectangle
28             self.p1: Final[Point] = Point(min(p1.x, p2.x), min(p1.y, p2.y))
29             #: the top-right point spanning the rectangle
30             self.p2: Final[Point] = Point(max(p1.x, p2.x), max(p1.y, p2.y))
31
32     def area(self) -> int | float:
33         """
34             Get the area of this rectangle.
35
36             :return: the area of this rectangle
37
38             >>> Rectangle(Point(7, 3), Point(12, 6)).area()
39             15
40             """
41
42             return (self.p2.x - self.p1.x) * (self.p2.y - self.p1.y)
43
44     def perimeter(self) -> int | float:
45         """
46             Get the perimeter of this rectangle.
47
48             :return: the perimeter of this rectangle
49
50             >>> Rectangle(Point(10, 5), Point(4, 9)).perimeter()
51             20
52             """
53
54             return 2 * ((self.p2.x - self.p1.x) + (self.p2.y - self.p1.y))
55
56     def points(self) -> tuple[Point, Point, Point, Point]:
57         """
58             Get the four corner points of this rectangle
59
60             :return: a tuple with the four corners of this rectangle
61
62             return (self.p1, Point(self.p1.x, self.p2.y), self.p2,
63                     Point(self.p2.x, self.p1.y))
```

Beispiel: Die Klasse Rectangle

- Andernfalls speichern wir die minimalen x- und y-Koordinaten im „links-unten“ Punkt-Attribut `p1`.
- Wir speichern die maximale x- und y-Koordinaten im Attribut `p2`, was den Punkt oben rechts markiert.
- Wir können nun die Methode `points` so implementieren, dass sie alle vier Eckpunkte des Rechtecks zurückliefert.
- Wir brauchen zwar nur zwei Punkte speichern, aber hier müssen wir alle vier zurückliefern, um der Definition der Methode zu entsprechen.

```
12
13
14 class Rectangle(Polygon):
15     """A rectangle defined by its bottom-left and top-right corners."""
16
17     def __init__(self, p1: Point, p2: Point) -> None:
18         """
19             Create a rectangle.
20
21             :param p1: the first point spanning the rectangle
22             :param p2: the second point spanning the rectangle
23             """
24
25             if (p1.x == p2.x) or (p1.y == p2.y): # check for non-emptiness
26                 raise ValueError(f'{p1.x},{p1.y},{p2.x},{p2.y} is empty.')
27             #: the bottom-left point spanning the rectangle
28             self.p1: Final[Point] = Point(min(p1.x, p2.x), min(p1.y, p2.y))
29             #: the top-right point spanning the rectangle
30             self.p2: Final[Point] = Point(max(p1.x, p2.x), max(p1.y, p2.y))
31
32     def area(self) -> int | float:
33         """
34             Get the area of this rectangle.
35
36             :return: the area of this rectangle
37
38             >>> Rectangle(Point(7, 3), Point(12, 6)).area()
39             15
40             """
41
42             return (self.p2.x - self.p1.x) * (self.p2.y - self.p1.y)
43
44     def perimeter(self) -> int | float:
45         """
46             Get the perimeter of this rectangle.
47
48             :return: the perimeter of this rectangle
49
50             >>> Rectangle(Point(10, 5), Point(4, 9)).perimeter()
51             20
52             """
53
54             return 2 * ((self.p2.x - self.p1.x) + (self.p2.y - self.p1.y))
55
56     def points(self) -> tuple[Point, Point, Point, Point]:
57         """
58             Get the four corner points of this rectangle.
59
60             :return: a tuple with the four corners of this rectangle
61
62             return (self.p1, Point(self.p1.x, self.p2.y), self.p2,
63                     Point(self.p2.x, self.p1.y))
```

Beispiel: Die Klasse Rectangle

- Wir speichern die maximale x- und y-Koordinaten im Attribut `p2`, was den Punkt oben rechts markiert.
- Wir können nun die Methode `points` so implementieren, dass sie alle vier Eckpunkte des Rechtecks zurückliefert.
- Wir brauchen zwar nur zwei Punkte speichern, aber hier müssen wir alle vier zurückliefern, um der Definition der Methode zu entsprechen.
- Wir können die Punkte problemlos berechnen und in einem Tupel zurückgeben.

```
12
13
14 class Rectangle(Polygon):
15     """A rectangle defined by its bottom-left and top-right corners."""
16
17     def __init__(self, p1: Point, p2: Point) -> None:
18         """
19             Create a rectangle.
20
21             :param p1: the first point spanning the rectangle
22             :param p2: the second point spanning the rectangle
23             """
24
25             if (p1.x == p2.x) or (p1.y == p2.y): # check for non-emptiness
26                 raise ValueError(f'{p1.x},{p1.y},{p2.x},{p2.y} is empty.')
27             #: the bottom-left point spanning the rectangle
28             self.p1: Final[Point] = Point(min(p1.x, p2.x), min(p1.y, p2.y))
29             #: the top-right point spanning the rectangle
30             self.p2: Final[Point] = Point(max(p1.x, p2.x), max(p1.y, p2.y))
31
32     def area(self) -> int | float:
33         """
34             Get the area of this rectangle.
35
36             :return: the area of this rectangle
37
38             >>> Rectangle(Point(7, 3), Point(12, 6)).area()
39             15
40             """
41
42             return (self.p2.x - self.p1.x) * (self.p2.y - self.p1.y)
43
44     def perimeter(self) -> int | float:
45         """
46             Get the perimeter of this rectangle.
47
48             :return: the perimeter of this rectangle
49
50             >>> Rectangle(Point(10, 5), Point(4, 9)).perimeter()
51             20
52             """
53
54             return 2 * ((self.p2.x - self.p1.x) + (self.p2.y - self.p1.y))
55
56     def points(self) -> tuple[Point, Point, Point, Point]:
57         """
58             Get the four corner points of this rectangle.
59
60             :return: a tuple with the four corners of this rectangle
61
62             return (self.p1, Point(self.p1.x, self.p2.y), self.p2,
63                     Point(self.p2.x, self.p1.y))
```

Beispiel: Die Klasse Rectangle

- Wir speichern die maximale x- und y-Koordinaten im Attribut `p2`, was den Punkt oben rechts markiert.
- Wir können nun die Methode `points` so implementieren, dass sie alle vier Eckpunkte des Rechtecks zurückliefert.
- Wir brauchen zwar nur zwei Punkte speichern, aber hier müssen wir alle vier zurückliefern, um der Definition der Methode zu entsprechen.
- Wir können die Punkte problemlos berechnen und in einem Tupel zurückgeben.
- Die Fläche des Rechtecks ist leicht zu berechnen.

```
12
13
14 class Rectangle(Polygon):
15     """A rectangle defined by its bottom-left and top-right corners."""
16
17     def __init__(self, p1: Point, p2: Point) -> None:
18         """
19             Create a rectangle.
20
21             :param p1: the first point spanning the rectangle
22             :param p2: the second point spanning the rectangle
23             """
24
25             if (p1.x == p2.x) or (p1.y == p2.y): # check for non-emptiness
26                 raise ValueError(f'{p1.x},{p1.y},{p2.x},{p2.y} is empty.')
27             #: the bottom-left point spanning the rectangle
28             self.p1: Final[Point] = Point(min(p1.x, p2.x), min(p1.y, p2.y))
29             #: the top-right point spanning the rectangle
30             self.p2: Final[Point] = Point(max(p1.x, p2.x), max(p1.y, p2.y))
31
32     def area(self) -> int | float:
33         """
34             Get the area of this rectangle.
35
36             :return: the area of this rectangle
37
38             >>> Rectangle(Point(7, 3), Point(12, 6)).area()
39             15
40             """
41
42             return (self.p2.x - self.p1.x) * (self.p2.y - self.p1.y)
43
44     def perimeter(self) -> int | float:
45         """
46             Get the perimeter of this rectangle.
47
48             :return: the perimeter of this rectangle
49
50             >>> Rectangle(Point(10, 5), Point(4, 9)).perimeter()
51             20
52             """
53
54             return 2 * ((self.p2.x - self.p1.x) + (self.p2.y - self.p1.y))
55
56     def points(self) -> tuple[Point, Point, Point, Point]:
57         """
58             Get the four corner points of this rectangle.
59
60             :return: a tuple with the four corners of this rectangle
61
62             return (self.p1, Point(self.p1.x, self.p2.y), self.p2,
63                     Point(self.p2.x, self.p1.y))
```

Beispiel: Die Klasse Rectangle

- Wir brauchen zwar nur zwei Punkte speichern, aber hier müssen wir alle vier zurückliefern, um der Definition der Methode zu entsprechen.
- Wir können die Punkte problemlos berechnen und in einem Tupel zurückgeben.
- Die Fläche des Rechtecks ist leicht zu berechnen.
- Die Methode `area` muss dafür nur die Breite und Höhe des Rechtecks berechnen und miteinander multiplizieren.

```
12
13
14 class Rectangle(Polygon):
15     """A rectangle defined by its bottom-left and top-right corners."""
16
17     def __init__(self, p1: Point, p2: Point) -> None:
18         """
19             Create a rectangle.
20
21             :param p1: the first point spanning the rectangle
22             :param p2: the second point spanning the rectangle
23             """
24
25         if (p1.x == p2.x) or (p1.y == p2.y): # check for non-emptiness
26             raise ValueError(f'{p1.x},{p1.y},{p2.x},{p2.y} is empty.')
27
28         #: the bottom-left point spanning the rectangle
29         self.p1: Final[Point] = Point(min(p1.x, p2.x), min(p1.y, p2.y))
30
31         #: the top-right point spanning the rectangle
32         self.p2: Final[Point] = Point(max(p1.x, p2.x), max(p1.y, p2.y))
33
34     def area(self) -> int | float:
35         """
36             Get the area of this rectangle.
37
38             :return: the area of this rectangle
39
40             >>> Rectangle(Point(7, 3), Point(12, 6)).area()
41             15
42             """
43
44         return (self.p2.x - self.p1.x) * (self.p2.y - self.p1.y)
45
46     def perimeter(self) -> int | float:
47         """
48             Get the perimeter of this rectangle.
49
50             :return: the perimeter of this rectangle
51
52             >>> Rectangle(Point(10, 5), Point(4, 9)).perimeter()
53             20
54             """
55
56         return 2 * ((self.p2.x - self.p1.x) + (self.p2.y - self.p1.y))
57
58     def points(self) -> tuple[Point, Point, Point, Point]:
59         """
60             Get the four corner points of this rectangle.
61
62             :return: a tuple with the four corners of this rectangle
63
64             return (self.p1, Point(self.p1.x, self.p2.y), self.p2,
65                     Point(self.p2.x, self.p1.y))
```

Beispiel: Die Klasse Rectangle

- Wir brauchen zwar nur zwei Punkte speichern, aber hier müssen wir alle vier zurückliefern, um der Definition der Methode zu entsprechen.
- Wir können die Punkte problemlos berechnen und in einem Tupel zurückgeben.
- Die Fläche des Rechtecks ist leicht zu berechnen.
- Die Methode `area` muss dafür nur die Breite und Höhe des Rechtecks berechnen und miteinander multiplizieren.
- Die Breite ist der Unterschied zwischen den x-Koordinaten der beiden Eckpunkte.

```
12
13
14 class Rectangle(Polygon):
15     """A rectangle defined by its bottom-left and top-right corners."""
16
17     def __init__(self, p1: Point, p2: Point) -> None:
18         """
19             Create a rectangle.
20
21             :param p1: the first point spanning the rectangle
22             :param p2: the second point spanning the rectangle
23             """
24
25             if (p1.x == p2.x) or (p1.y == p2.y): # check for non-emptiness
26                 raise ValueError(f'{p1.x},{p1.y},{p2.x},{p2.y} is empty.')
27             #: the bottom-left point spanning the rectangle
28             self.p1: Final[Point] = Point(min(p1.x, p2.x), min(p1.y, p2.y))
29             #: the top-right point spanning the rectangle
30             self.p2: Final[Point] = Point(max(p1.x, p2.x), max(p1.y, p2.y))
31
32     def area(self) -> int | float:
33         """
34             Get the area of this rectangle.
35
36             :return: the area of this rectangle
37
38             >>> Rectangle(Point(7, 3), Point(12, 6)).area()
39             15
40             """
41
42             return (self.p2.x - self.p1.x) * (self.p2.y - self.p1.y)
43
44     def perimeter(self) -> int | float:
45         """
46             Get the perimeter of this rectangle.
47
48             :return: the perimeter of this rectangle
49
50             >>> Rectangle(Point(10, 5), Point(4, 9)).perimeter()
51             20
52             """
53
54             return 2 * ((self.p2.x - self.p1.x) + (self.p2.y - self.p1.y))
55
56     def points(self) -> tuple[Point, Point, Point, Point]:
57         """
58             Get the four corner points of this rectangle.
59
60             :return: a tuple with the four corners of this rectangle
61
62             return (self.p1, Point(self.p1.x, self.p2.y), self.p2,
63                     Point(self.p2.x, self.p1.y))
```

Beispiel: Die Klasse Rectangle

- Wir können die Punkte problemlos berechnen und in einem Tupel zurückgeben.
- Die Fläche des Rechtecks ist leicht zu berechnen.
- Die Methode `area` muss dafür nur die Breite und Höhe des Rechtecks berechnen und miteinander multiplizieren.
- Die Breite ist der Unterschied zwischen den x-Koordinaten der beiden Eckpunkte.
- Die Höhe ist der Unterschied zwischen den y-Koordinaten der beiden Eckpunkte.

```
12
13
14 class Rectangle(Polygon):
15     """A rectangle defined by its bottom-left and top-right corners."""
16
17     def __init__(self, p1: Point, p2: Point) -> None:
18         """
19             Create a rectangle.
20
21             :param p1: the first point spanning the rectangle
22             :param p2: the second point spanning the rectangle
23             """
24
25             if (p1.x == p2.x) or (p1.y == p2.y): # check for non-emptiness
26                 raise ValueError(f'{p1.x},{p1.y},{p2.x},{p2.y} is empty.')
27             #: the bottom-left point spanning the rectangle
28             self.p1: Final[Point] = Point(min(p1.x, p2.x), min(p1.y, p2.y))
29             #: the top-right point spanning the rectangle
30             self.p2: Final[Point] = Point(max(p1.x, p2.x), max(p1.y, p2.y))
31
32
33     def area(self) -> int | float:
34         """
35             Get the area of this rectangle.
36
37             :return: the area of this rectangle
38
39             >>> Rectangle(Point(7, 3), Point(12, 6)).area()
40             15
41             """
42
43             return (self.p2.x - self.p1.x) * (self.p2.y - self.p1.y)
44
45
46     def perimeter(self) -> int | float:
47         """
48             Get the perimeter of this rectangle.
49
50             :return: the perimeter of this rectangle
51
52             >>> Rectangle(Point(10, 5), Point(4, 9)).perimeter()
53             20
54             """
55
56             return 2 * ((self.p2.x - self.p1.x) + (self.p2.y - self.p1.y))
57
58
59     def points(self) -> tuple[Point, Point, Point, Point]:
60         """
61             Get the four corner points of this rectangle.
62
63             :return: a tuple with the four corners of this rectangle
64
65             return (self.p1, Point(self.p1.x, self.p2.y), self.p2,
66                     Point(self.p2.x, self.p1.y))
```

Beispiel: Die Klasse Rectangle

- Die Methode `area` muss dafür nur die Breite und Höhe des Rechtecks berechnen und miteinander multiplizieren.
- Die Breite ist der Unterschied zwischen den x-Koordinaten der beiden Eckpunkte.
- Die Höhe ist der Unterschied zwischen den y-Koordinaten der beiden Eckpunkte.
- Eigentlich kann die Methode `perimeter`, die den Umfang des Rechtecks berechnet, eigentlich schon mit unserer Methode `points` funktionieren.

```
12
13
14 class Rectangle(Polygon):
15     """A rectangle defined by its bottom-left and top-right corners."""
16
17     def __init__(self, p1: Point, p2: Point) -> None:
18         """
19             Create a rectangle.
20
21             :param p1: the first point spanning the rectangle
22             :param p2: the second point spanning the rectangle
23             """
24
25             if (p1.x == p2.x) or (p1.y == p2.y): # check for non-emptiness
26                 raise ValueError(f'{p1.x},{p1.y},{p2.x},{p2.y} is empty.')
27             #: the bottom-left point spanning the rectangle
28             self.p1: Final[Point] = Point(min(p1.x, p2.x), min(p1.y, p2.y))
29             #: the top-right point spanning the rectangle
30             self.p2: Final[Point] = Point(max(p1.x, p2.x), max(p1.y, p2.y))
31
32
33     def area(self) -> int | float:
34         """
35             Get the area of this rectangle.
36
37             :return: the area of this rectangle
38
39             >>> Rectangle(Point(7, 3), Point(12, 6)).area()
40             15
41             """
42
43             return (self.p2.x - self.p1.x) * (self.p2.y - self.p1.y)
44
45
46     def perimeter(self) -> int | float:
47         """
48             Get the perimeter of this rectangle.
49
50             :return: the perimeter of this rectangle
51
52             >>> Rectangle(Point(10, 5), Point(4, 9)).perimeter()
53             20
54             """
55
56             return 2 * ((self.p2.x - self.p1.x) + (self.p2.y - self.p1.y))
57
58
59     def points(self) -> tuple[Point, Point, Point, Point]:
60         """
61             Get the four corner points of this rectangle.
62
63             :return: a tuple with the four corners of this rectangle
64
65             return (self.p1, Point(self.p1.x, self.p2.y), self.p2,
66                     Point(self.p2.x, self.p1.y))
```

Beispiel: Die Klasse Rectangle

- Die Methode `area` muss dafür nur die Breite und Höhe des Rechtecks berechnen und miteinander multiplizieren.
- Die Breite ist der Unterschied zwischen den x-Koordinaten der beiden Eckpunkte.
- Die Höhe ist der Unterschied zwischen den y-Koordinaten der beiden Eckpunkte.
- Eigentlich kann die Methode `perimeter`, die den Umfang des Rechtecks berechnet, eigentlich schon mit unserer Methode `points` funktionieren.
- Wir überschreiben sie aber trotzdem.

```
12
13
14 class Rectangle(Polygon):
15     """A rectangle defined by its bottom-left and top-right corners."""
16
17     def __init__(self, p1: Point, p2: Point) -> None:
18         """
19             Create a rectangle.
20
21             :param p1: the first point spanning the rectangle
22             :param p2: the second point spanning the rectangle
23             """
24
25             if (p1.x == p2.x) or (p1.y == p2.y): # check for non-emptiness
26                 raise ValueError(f'{p1.x},{p1.y},{p2.x},{p2.y} is empty.')
27             #: the bottom-left point spanning the rectangle
28             self.p1: Final[Point] = Point(min(p1.x, p2.x), min(p1.y, p2.y))
29             #: the top-right point spanning the rectangle
30             self.p2: Final[Point] = Point(max(p1.x, p2.x), max(p1.y, p2.y))
31
32
33     def area(self) -> int | float:
34         """
35             Get the area of this rectangle.
36
37             :return: the area of this rectangle
38
39             >>> Rectangle(Point(7, 3), Point(12, 6)).area()
40             15
41             """
42
43             return (self.p2.x - self.p1.x) * (self.p2.y - self.p1.y)
44
45
46     def perimeter(self) -> int | float:
47         """
48             Get the perimeter of this rectangle.
49
50             :return: the perimeter of this rectangle
51
52             >>> Rectangle(Point(10, 5), Point(4, 9)).perimeter()
53             20
54             """
55
56             return 2 * ((self.p2.x - self.p1.x) + (self.p2.y - self.p1.y))
57
58
59     def points(self) -> tuple[Point, Point, Point, Point]:
60         """
61             Get the four corner points of this rectangle.
62
63             :return: a tuple with the four corners of this rectangle
64
65             return (self.p1, Point(self.p1.x, self.p2.y), self.p2,
66                     Point(self.p2.x, self.p1.y))
```

Beispiel: Die Klasse Rectangle

- Die Höhe ist der Unterschied zwischen den y-Koordinaten der beiden Eckpunkte.
- Eigentlich kann die Methode `perimeter`, die den Umfang des Rechtecks berechnet, eigentlich schon mit unserer Methode `points` funktionieren.
- Wir überschreiben sie aber trotzdem.
- Das ist sinnvoll, weil wir den Umfang schneller und genauer berechnen können, in dem wir einfach die doppelte Summe von Breite und Höhe des Rechtecks zurückliefern.

```
12
13
14 class Rectangle(Polygon):
15     """A rectangle defined by its bottom-left and top-right corners."""
16
17     def __init__(self, p1: Point, p2: Point) -> None:
18         """
19             Create a rectangle.
20
21             :param p1: the first point spanning the rectangle
22             :param p2: the second point spanning the rectangle
23             """
24
25             if (p1.x == p2.x) or (p1.y == p2.y): # check for non-emptiness
26                 raise ValueError(f'{p1.x},{p1.y},{p2.x},{p2.y} is empty.')
27             #: the bottom-left point spanning the rectangle
28             self.p1: Final[Point] = Point(min(p1.x, p2.x), min(p1.y, p2.y))
29             #: the top-right point spanning the rectangle
30             self.p2: Final[Point] = Point(max(p1.x, p2.x), max(p1.y, p2.y))
31
32     def area(self) -> int | float:
33         """
34             Get the area of this rectangle.
35
36             :return: the area of this rectangle
37
38             >>> Rectangle(Point(7, 3), Point(12, 6)).area()
39             15
40             """
41
42             return (self.p2.x - self.p1.x) * (self.p2.y - self.p1.y)
43
44     def perimeter(self) -> int | float:
45         """
46             Get the perimeter of this rectangle.
47
48             :return: the perimeter of this rectangle
49
50             >>> Rectangle(Point(10, 5), Point(4, 9)).perimeter()
51             20
52             """
53
54             return 2 * ((self.p2.x - self.p1.x) + (self.p2.y - self.p1.y))
55
56     def points(self) -> tuple[Point, Point, Point, Point]:
57         """
58             Get the four corner points of this rectangle.
59
60             :return: a tuple with the four corners of this rectangle
61
62             return (self.p1, Point(self.p1.x, self.p2.y), self.p2,
63                     Point(self.p2.x, self.p1.y))
```

Beispiel: Die Klasse Rectangle

- Eigentlich kann die Methode **perimeter**, die den Umfang des Rechtecks berechnet, eigentlich schon mit unserer Methode **points** funktionieren.
- Wir überschreiben sie aber trotzdem.
- Das ist sinnvoll, weil wir den Umfang schneller und genauer berechnen können, in dem wir einfach die doppelte Summe von Breite und Höhe des Rechtecks zurückliefern.
- Die geerbte **perimeter** methode würde stattdessen über die vier Euklidischen Punktabstände iterieren, wobei jeweils eine Quadratwurzel berechnet werden muss.

```
12
13
14 class Rectangle(Polygon):
15     """A rectangle defined by its bottom-left and top-right corners."""
16
17     def __init__(self, p1: Point, p2: Point) -> None:
18         """
19             Create a rectangle.
20
21             :param p1: the first point spanning the rectangle
22             :param p2: the second point spanning the rectangle
23             """
24
25             if (p1.x == p2.x) or (p1.y == p2.y): # check for non-emptiness
26                 raise ValueError(f'{p1.x},{p1.y},{p2.x},{p2.y} is empty.')
27             #: the bottom-left point spanning the rectangle
28             self.p1: Final[Point] = Point(min(p1.x, p2.x), min(p1.y, p2.y))
29             #: the top-right point spanning the rectangle
30             self.p2: Final[Point] = Point(max(p1.x, p2.x), max(p1.y, p2.y))
31
32     def area(self) -> int | float:
33         """
34             Get the area of this rectangle.
35
36             :return: the area of this rectangle
37
38             >>> Rectangle(Point(7, 3), Point(12, 6)).area()
39             15
40             """
41
42             return (self.p2.x - self.p1.x) * (self.p2.y - self.p1.y)
43
44     def perimeter(self) -> int | float:
45         """
46             Get the perimeter of this rectangle.
47
48             :return: the perimeter of this rectangle
49
50             >>> Rectangle(Point(10, 5), Point(4, 9)).perimeter()
51             20
52             """
53
54             return 2 * ((self.p2.x - self.p1.x) + (self.p2.y - self.p1.y))
55
56     def points(self) -> tuple[Point, Point, Point, Point]:
57         """
58             Get the four corner points of this rectangle.
59
60             :return: a tuple with the four corners of this rectangle
61
62             return (self.p1, Point(self.p1.x, self.p2.y), self.p2,
63                     Point(self.p2.x, self.p1.y))
```

Beispiel: Die Klasse Rectangle

- Wir überschreiben sie aber trotzdem.
- Das ist sinnvoll, weil wir den Umfang schneller und genauer berechnen können, in dem wir einfach die doppelte Summe von Breite und Höhe des Rechtecks zurückliefern.
- Die geerbte `perimeter` methode würde stattdessen über die vier Euklidischen Punktabstände iterieren, wobei jeweils eine Quadratwurzel berechnet werden muss.
- Das ist langsamer und ungenauer, besonders wenn die Koordinaten ganzzahlig sind.

```
12
13
14 class Rectangle(Polygon):
15     """A rectangle defined by its bottom-left and top-right corners."""
16
17     def __init__(self, p1: Point, p2: Point) -> None:
18         """
19             Create a rectangle.
20
21             :param p1: the first point spanning the rectangle
22             :param p2: the second point spanning the rectangle
23             """
24
25         if (p1.x == p2.x) or (p1.y == p2.y): # check for non-emptiness
26             raise ValueError(f'{p1.x},{p1.y},{p2.x},{p2.y} is empty.')
27         #: the bottom-left point spanning the rectangle
28         self.p1: Final[Point] = Point(min(p1.x, p2.x), min(p1.y, p2.y))
29         #: the top-right point spanning the rectangle
30         self.p2: Final[Point] = Point(max(p1.x, p2.x), max(p1.y, p2.y))
31
32     def area(self) -> int | float:
33         """
34             Get the area of this rectangle.
35
36             :return: the area of this rectangle
37
38             >>> Rectangle(Point(7, 3), Point(12, 6)).area()
39             15
40             """
41
42         return (self.p2.x - self.p1.x) * (self.p2.y - self.p1.y)
43
44     def perimeter(self) -> int | float:
45         """
46             Get the perimeter of this rectangle.
47
48             :return: the perimeter of this rectangle
49
50             >>> Rectangle(Point(10, 5), Point(4, 9)).perimeter()
51             20
52             """
53
54         return 2 * ((self.p2.x - self.p1.x) + (self.p2.y - self.p1.y))
55
56     def points(self) -> tuple[Point, Point, Point, Point]:
57         """
58             Get the four corner points of this rectangle.
59
60             :return: a tuple with the four corners of this rectangle
61
62             return (self.p1, Point(self.p1.x, self.p2.y), self.p2,
63                     Point(self.p2.x, self.p1.y))
```

Beispiel: Die Klasse Triangle

- Die Klasse `Triangle` für Dreiecke wird in der Datei `triangle.py` implementiert.

```
1 """
2 A class for triangles.
3
4 >>> Triangle(Point(22, 1), Point(4, 12), Point(6, 3)).print()
5 (22, 1), (4, 12), (6, 3)
6 >>> Triangle(Point(0, 0), Point(0, 3), Point(4, 0)).perimeter()
7 12.0
8 """
9
10 from typing import Final
11
12 from point import Point
13 from polygon import Polygon
14
15
16 class Triangle(Polygon):
17     """The class for triangles."""
18
19     def __init__(self, p1: Point, p2: Point, p3: Point) -> None:
20         """
21             Create a triangle.
22
23             :param p1: the first point spanning the triangle
24             :param p2: the second point spanning the triangle
25             :param p3: the third point spanning the triangle
26         """
27         if (p1.distance(p2) <= 0) or (p2.distance(p3) <= 0) or (
28             p3.distance(p1) <= 0): # check for non-emptiness
29             raise ValueError("empty triangle")
30         self.p1: Final[Point] = p1
31         self.p2: Final[Point] = p2
32         self.p3: Final[Point] = p3
33
34     def area(self) -> int | float:
35         """
36             Get the area of this triangle.
37
38             :return: the area of this triangle
39
40         >>> Triangle(Point(-1, 2), Point(2, 3), Point(4, -3)).area()
41         10.0
42         """
43         return 0.5 * abs(self.p1.x * (self.p2.y - self.p3.y)
44                         + self.p2.x * (self.p3.y - self.p1.y)
45                         + self.p3.x * (self.p1.y - self.p2.y))
46
47     def points(self) -> tuple[Point, Point, Point]:
48         """
49             Get the three points describing this triangle.
50
51             :return: a tuple with the three corners of this triangle
52         """
53         return self.p1, self.p2, self.p3
```



Beispiel: Die Klasse Triangle

- Die Klasse `Triangle` für Dreiecke wird in der Datei `triangle.py` implementiert.
- Dieses mal müssen wir alle drei Eckpunkte speichern.

```
14
15
16 class Triangle(Polygon):
17     """The class for triangles."""
18
19     def __init__(self, p1: Point, p2: Point, p3: Point) -> None:
20         """
21             Create a triangle.
22
23             :param p1: the first point spanning the triangle
24             :param p2: the second point spanning the triangle
25             :param p3: the third point spanning the triangle
26         """
27         if (p1.distance(p2) <= 0) or (p2.distance(p3) <= 0) or (
28             p3.distance(p1) <= 0): # check for non-emptiness
29             raise ValueError("empty triangle")
30         #: the first point spanning the triangle
31         self.p1: Final[Point] = p1
32         #: the second point spanning the triangle
33         self.p2: Final[Point] = p2
34         #: the third point spanning the triangle
35         self.p3: Final[Point] = p3
36
37     def area(self) -> int | float:
38         """
39             Get the area of this triangle.
40
41             :return: the area of this triangle
42
43             >>> Triangle(Point(-1, 2), Point(2, 3), Point(4, -3)).area()
44             10.0
45         """
46         return 0.5 * abs(self.p1.x * (self.p2.y - self.p3.y)
47                         + self.p2.x * (self.p3.y - self.p1.y)
48                         + self.p3.x * (self.p1.y - self.p2.y))
49
50     def points(self) -> tuple[Point, Point, Point]:
51         """
52             Get the three points describing this triangle.
53
54             :return: a tuple with the three corners of this triangle
55
56             return self.p1, self.p2, self.p3
```

Beispiel: Die Klasse Triangle

- Die Klasse `Triangle` für Dreiecke wird in der Datei `triangle.py` implementiert.
- Dieses mal müssen wir alle drei Eckpunkte speichern.
- Im Initialisierer `__init__` prüfen wir aber erst, ob eine Seitenlänge 0 ist, denn dann würden die Punkte kein Dreieck beschreiben.

```
14
15
16 class Triangle(Polygon):
17     """The class for triangles."""
18
19     def __init__(self, p1: Point, p2: Point, p3: Point) -> None:
20         """
21             Create a triangle.
22
23             :param p1: the first point spanning the triangle
24             :param p2: the second point spanning the triangle
25             :param p3: the third point spanning the triangle
26         """
27         if (p1.distance(p2) <= 0) or (p2.distance(p3) <= 0) or (
28             p3.distance(p1) <= 0): # check for non-emptiness
29             raise ValueError("empty triangle")
30         #: the first point spanning the triangle
31         self.p1: Final[Point] = p1
32         #: the second point spanning the triangle
33         self.p2: Final[Point] = p2
34         #: the third point spanning the triangle
35         self.p3: Final[Point] = p3
36
37     def area(self) -> int | float:
38         """
39             Get the area of this triangle.
40
41             :return: the area of this triangle
42
43             >>> Triangle(Point(-1, 2), Point(2, 3), Point(4, -3)).area()
44             10.0
45         """
46         return 0.5 * abs(self.p1.x * (self.p2.y - self.p3.y)
47                         + self.p2.x * (self.p3.y - self.p1.y)
48                         + self.p3.x * (self.p1.y - self.p2.y))
49
50     def points(self) -> tuple[Point, Point, Point]:
51         """
52             Get the three points describing this triangle.
53
54             :return: a tuple with the three corners of this triangle
55
56             return self.p1, self.p2, self.p3
```

Beispiel: Die Klasse Triangle

- Die Klasse `Triangle` für Dreiecke wird in der Datei `triangle.py` implementiert.
- Dieses mal müssen wir alle drei Eckpunkte speichern.
- Im Initialisierer `__init__` prüfen wir aber erst, ob eine Seitenlänge 0 ist, denn dann würden die Punkte kein Dreieck beschreiben.
- In diesem Fall lösen wir einen `ValueError` aus.

```
14
15
16 class Triangle(Polygon):
17     """The class for triangles."""
18
19     def __init__(self, p1: Point, p2: Point, p3: Point) -> None:
20         """
21             Create a triangle.
22
23             :param p1: the first point spanning the triangle
24             :param p2: the second point spanning the triangle
25             :param p3: the third point spanning the triangle
26         """
27         if (p1.distance(p2) <= 0) or (p2.distance(p3) <= 0) or (
28             p3.distance(p1) <= 0): # check for non-emptiness
29             raise ValueError("empty triangle")
30         #: the first point spanning the triangle
31         self.p1: Final[Point] = p1
32         #: the second point spanning the triangle
33         self.p2: Final[Point] = p2
34         #: the third point spanning the triangle
35         self.p3: Final[Point] = p3
36
37     def area(self) -> int | float:
38         """
39             Get the area of this triangle.
40
41             :return: the area of this triangle
42
43             >>> Triangle(Point(-1, 2), Point(2, 3), Point(4, -3)).area()
44             10.0
45         """
46         return 0.5 * abs(self.p1.x * (self.p2.y - self.p3.y)
47                         + self.p2.x * (self.p3.y - self.p1.y)
48                         + self.p3.x * (self.p1.y - self.p2.y))
49
50     def points(self) -> tuple[Point, Point, Point]:
51         """
52             Get the three points describing this triangle.
53
54             :return: a tuple with the three corners of this triangle
55
56             return self.p1, self.p2, self.p3
```

Beispiel: Die Klasse Triangle

- Die Klasse `Triangle` für Dreiecke wird in der Datei `triangle.py` implementiert.
- Dieses mal müssen wir alle drei Eckpunkte speichern.
- Im Initialisierer `__init__` prüfen wir aber erst, ob eine Seitenlänge 0 ist, denn dann würden die Punkte kein Dreieck beschreiben.
- In diesem Fall lösen wir einen `ValueError` aus.
- In der Implementierung der Methode `points` liefern wir alle drei Eckpunkte zurück.

```
14
15
16 class Triangle(Polygon):
17     """The class for triangles."""
18
19     def __init__(self, p1: Point, p2: Point, p3: Point) -> None:
20         """
21             Create a triangle.
22
23             :param p1: the first point spanning the triangle
24             :param p2: the second point spanning the triangle
25             :param p3: the third point spanning the triangle
26         """
27         if (p1.distance(p2) <= 0) or (p2.distance(p3) <= 0) or (
28             p3.distance(p1) <= 0): # check for non-emptiness
29             raise ValueError("empty triangle")
30         #: the first point spanning the triangle
31         self.p1: Final[Point] = p1
32         #: the second point spanning the triangle
33         self.p2: Final[Point] = p2
34         #: the third point spanning the triangle
35         self.p3: Final[Point] = p3
36
37     def area(self) -> int | float:
38         """
39             Get the area of this triangle.
40
41             :return: the area of this triangle
42
43             >>> Triangle(Point(-1, 2), Point(2, 3), Point(4, -3)).area()
44             10.0
45         """
46         return 0.5 * abs(self.p1.x * (self.p2.y - self.p3.y)
47                         + self.p2.x * (self.p3.y - self.p1.y)
48                         + self.p3.x * (self.p1.y - self.p2.y))
49
50     def points(self) -> tuple[Point, Point, Point]:
51         """
52             Get the three points describing this triangle.
53
54             :return: a tuple with the three corners of this triangle
55
56             return self.p1, self.p2, self.p3
```

Beispiel: Die Klasse Triangle

- Dieses mal müssen wir alle drei Eckpunkte speichern.
- Im Initialisierer `__init__` prüfen wir aber erst, ob eine Seitenlänge 0 ist, denn dann würden die Punkte kein Dreieck beschreiben.
- In diesem Fall lösen wir einen `ValueError` aus.
- In der Implementierung der Methode `points` liefern wir alle drei Eckpunkte zurück.
- Diesmal gibt es keine besser Möglichkeit, den Umfang des Dreiecks zu berechnen, als das was die `perimeter` aus der `Polygon` sowieso schon macht.

```
14
15
16 class Triangle(Polygon):
17     """The class for triangles."""
18
19     def __init__(self, p1: Point, p2: Point, p3: Point) -> None:
20         """
21             Create a triangle.
22
23             :param p1: the first point spanning the triangle
24             :param p2: the second point spanning the triangle
25             :param p3: the third point spanning the triangle
26         """
27         if (p1.distance(p2) <= 0) or (p2.distance(p3) <= 0) or (
28             p3.distance(p1) <= 0): # check for non-emptiness
29             raise ValueError("empty triangle")
30         #: the first point spanning the triangle
31         self.p1: Final[Point] = p1
32         #: the second point spanning the triangle
33         self.p2: Final[Point] = p2
34         #: the third point spanning the triangle
35         self.p3: Final[Point] = p3
36
37     def area(self) -> int | float:
38         """
39             Get the area of this triangle.
40
41             :return: the area of this triangle
42
43             >>> Triangle(Point(-1, 2), Point(2, 3), Point(4, -3)).area()
44             10.0
45         """
46         return 0.5 * abs(self.p1.x * (self.p2.y - self.p3.y)
47                         + self.p2.x * (self.p3.y - self.p1.y)
48                         + self.p3.x * (self.p1.y - self.p2.y))
49
50     def points(self) -> tuple[Point, Point, Point]:
51         """
52             Get the three points describing this triangle.
53
54             :return: a tuple with the three corners of this triangle
55
56             return self.p1, self.p2, self.p3
```

Beispiel: Die Klasse Triangle

- Im Initialisierer `__init__` prüfen wir aber erst, ob eine Seitenlänge 0 ist, denn dann würden die Punkte kein Dreieck beschreiben.
- In diesem Fall lösen wir einen `ValueError` aus.
- In der Implementierung der Methode `points` liefern wir alle drei Eckpunkte zurück.
- Diesmal gibt es keine besser Möglichkeit, den Umfang des Dreiecks zu berechnen, als das was die `perimeter` aus der `Polygon` sowieso schon macht.
- Deshalb überschreiben wir sie diesmal nicht.

```
14
15
16 class Triangle(Polygon):
17     """The class for triangles."""
18
19     def __init__(self, p1: Point, p2: Point, p3: Point) -> None:
20         """
21             Create a triangle.
22
23             :param p1: the first point spanning the triangle
24             :param p2: the second point spanning the triangle
25             :param p3: the third point spanning the triangle
26         """
27         if (p1.distance(p2) <= 0) or (p2.distance(p3) <= 0) or (
28             p3.distance(p1) <= 0): # check for non-emptiness
29             raise ValueError("empty triangle")
30         #: the first point spanning the triangle
31         self.p1: Final[Point] = p1
32         #: the second point spanning the triangle
33         self.p2: Final[Point] = p2
34         #: the third point spanning the triangle
35         self.p3: Final[Point] = p3
36
37     def area(self) -> int | float:
38         """
39             Get the area of this triangle.
40
41             :return: the area of this triangle
42
43             >>> Triangle(Point(-1, 2), Point(2, 3), Point(4, -3)).area()
44             10.0
45         """
46         return 0.5 * abs(self.p1.x * (self.p2.y - self.p3.y)
47                         + self.p2.x * (self.p3.y - self.p1.y)
48                         + self.p3.x * (self.p1.y - self.p2.y))
49
50     def points(self) -> tuple[Point, Point, Point]:
51         """
52             Get the three points describing this triangle.
53
54             :return: a tuple with the three corners of this triangle
55
56             return self.p1, self.p2, self.p3
```

Beispiel: Die Klasse Triangle

- In diesem Fall lösen wir einen `ValueError` aus.
- In der Implementierung der Methode `points` liefern wir alle drei Eckpunkte zurück.
- Diesmal gibt es keine besser Möglichkeit, den Umfang des Dreiecks zu berechnen, als das was die `perimeter` aus der `Polygon` sowieso schon macht.
- Deshalb überschreiben wir sie diesmal nicht.
- Die Fläche des Dreiecks kann über die Formel $A = x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)$ berechnet werden.

```
14
15
16 class Triangle(Polygon):
17     """The class for triangles."""
18
19     def __init__(self, p1: Point, p2: Point, p3: Point) -> None:
20         """
21             Create a triangle.
22
23             :param p1: the first point spanning the triangle
24             :param p2: the second point spanning the triangle
25             :param p3: the third point spanning the triangle
26         """
27         if (p1.distance(p2) <= 0) or (p2.distance(p3) <= 0) or (
28             p3.distance(p1) <= 0): # check for non-emptiness
29             raise ValueError("empty triangle")
30         #: the first point spanning the triangle
31         self.p1: Final[Point] = p1
32         #: the second point spanning the triangle
33         self.p2: Final[Point] = p2
34         #: the third point spanning the triangle
35         self.p3: Final[Point] = p3
36
37     def area(self) -> int | float:
38         """
39             Get the area of this triangle.
40
41             :return: the area of this triangle
42
43             >>> Triangle(Point(-1, 2), Point(2, 3), Point(4, -3)).area()
44             10.0
45         """
46         return 0.5 * abs(self.p1.x * (self.p2.y - self.p3.y)
47                         + self.p2.x * (self.p3.y - self.p1.y)
48                         + self.p3.x * (self.p1.y - self.p2.y))
49
50     def points(self) -> tuple[Point, Point, Point]:
51         """
52             Get the three points describing this triangle.
53
54             :return: a tuple with the three corners of this triangle
55
56             return self.p1, self.p2, self.p3
```

Beispiel: Die Klasse Triangle

- Diesmal gibt es keine besser Möglichkeit, den Umfang des Dreiecks zu berechnen, als das was die `perimeter` aus der `Polygon` sowieso schon macht.
- Deshalb überschreiben wir sie diesmal nicht.
- Die Fläche des Dreiecks kann über die Formel $A = x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)$ berechnet werden.
- Beide Subklassen von `Polygon` benutzen Doctests in ihren Docstrings.

```
14
15
16 class Triangle(Polygon):
17     """The class for triangles."""
18
19     def __init__(self, p1: Point, p2: Point, p3: Point) -> None:
20         """
21             Create a triangle.
22
23             :param p1: the first point spanning the triangle
24             :param p2: the second point spanning the triangle
25             :param p3: the third point spanning the triangle
26         """
27         if (p1.distance(p2) <= 0) or (p2.distance(p3) <= 0) or (
28             p3.distance(p1) <= 0): # check for non-emptiness
29             raise ValueError("empty triangle")
30         #: the first point spanning the triangle
31         self.p1: Final[Point] = p1
32         #: the second point spanning the triangle
33         self.p2: Final[Point] = p2
34         #: the third point spanning the triangle
35         self.p3: Final[Point] = p3
36
37     def area(self) -> int | float:
38         """
39             Get the area of this triangle.
40
41             :return: the area of this triangle
42
43             >>> Triangle(Point(-1, 2), Point(2, 3), Point(4, -3)).area()
44             10.0
45         """
46         return 0.5 * abs(self.p1.x * (self.p2.y - self.p3.y)
47                         + self.p2.x * (self.p3.y - self.p1.y)
48                         + self.p3.x * (self.p1.y - self.p2.y))
49
50     def points(self) -> tuple[Point, Point, Point]:
51         """
52             Get the three points describing this triangle.
53
54             :return: a tuple with the three corners of this triangle
55
56             return self.p1, self.p2, self.p3
```

Beispiel: Die Klasse Triangle

- Deshalb überschreiben wir sie diesmal nicht.
- Die Fläche des Dreiecks kann über die Formel $A = x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)$ berechnet werden.
- Beide Subklassen von `Polygon` benutzen Doctests in ihren Docstrings.
- Wir mussten diese Tests kurz halten, damit die Beispiele noch auf eine Slide passen.

```
14
15
16 class Triangle(Polygon):
17     """The class for triangles."""
18
19     def __init__(self, p1: Point, p2: Point, p3: Point) -> None:
20         """
21             Create a triangle.
22
23             :param p1: the first point spanning the triangle
24             :param p2: the second point spanning the triangle
25             :param p3: the third point spanning the triangle
26         """
27         if (p1.distance(p2) <= 0) or (p2.distance(p3) <= 0) or (
28             p3.distance(p1) <= 0): # check for non-emptiness
29             raise ValueError("empty triangle")
30         #: the first point spanning the triangle
31         self.p1: Final[Point] = p1
32         #: the second point spanning the triangle
33         self.p2: Final[Point] = p2
34         #: the third point spanning the triangle
35         self.p3: Final[Point] = p3
36
37     def area(self) -> int | float:
38         """
39             Get the area of this triangle.
40
41             :return: the area of this triangle
42
43             >>> Triangle(Point(-1, 2), Point(2, 3), Point(4, -3)).area()
44             10.0
45         """
46         return 0.5 * abs(self.p1.x * (self.p2.y - self.p3.y)
47                         + self.p2.x * (self.p3.y - self.p1.y)
48                         + self.p3.x * (self.p1.y - self.p2.y))
49
50     def points(self) -> tuple[Point, Point, Point]:
51         """
52             Get the three points describing this triangle.
53
54             :return: a tuple with the three corners of this triangle
55
56             return self.p1, self.p2, self.p3
```

Beispiel: Die Klasse Triangle

- Die Fläche des Dreiecks kann über die Formel $A = x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)$ berechnet werden.
- Beide Subklassen von `Polygon` benutzen Doctests in ihren Docstrings.
- Wir mussten diese Tests kurz halten, damit die Beispiele noch auf eine Slide passen.
- Trotzdem sind sie aufschlussreich für den Benutzer.

```
14
15
16 class Triangle(Polygon):
17     """The class for triangles."""
18
19     def __init__(self, p1: Point, p2: Point, p3: Point) -> None:
20         """
21             Create a triangle.
22
23             :param p1: the first point spanning the triangle
24             :param p2: the second point spanning the triangle
25             :param p3: the third point spanning the triangle
26         """
27         if (p1.distance(p2) <= 0) or (p2.distance(p3) <= 0) or (
28             p3.distance(p1) <= 0): # check for non-emptiness
29             raise ValueError("empty triangle")
30         #: the first point spanning the triangle
31         self.p1: Final[Point] = p1
32         #: the second point spanning the triangle
33         self.p2: Final[Point] = p2
34         #: the third point spanning the triangle
35         self.p3: Final[Point] = p3
36
37     def area(self) -> int | float:
38         """
39             Get the area of this triangle.
40
41             :return: the area of this triangle
42
43             >>> Triangle(Point(-1, 2), Point(2, 3), Point(4, -3)).area()
44             10.0
45         """
46         return 0.5 * abs(self.p1.x * (self.p2.y - self.p3.y)
47                         + self.p2.x * (self.p3.y - self.p1.y)
48                         + self.p3.x * (self.p1.y - self.p2.y))
49
50     def points(self) -> tuple[Point, Point, Point]:
51         """
52             Get the three points describing this triangle.
53
54             :return: a tuple with the three corners of this triangle
55
56             return self.p1, self.p2, self.p3
```

Beispiel: Die Subklassen von Shape benutzen

- Im Programm `shape_user.py` benutzen wir nun alle `Shape` abgeleiteten Klassen.



Beispiel: Die Subklassen von Shape benutzen



- Im Programm `shape_user.py` benutzen wir nun alle `Shape` abgeleiteten Klassen.
- Alle von ihnen unterstützen die Methoden `area` und `perimeter`.

```
1 """Examples for using the different :class:`Shape` classes."""
2
3 from circle import Circle      # Our new class `Circle`.
4 from point import Point        # Our very first class ever: `Point`.
5 from rectangle import Rectangle # Our new class `Rectangle`.
6 from triangle import Triangle  # Our new class `Triangle`.
7 from shape import Shape         # Our base class `Shape`.
8
9
10 shapes: list[Shape] = [    # We create list of shapes.
11     Circle(Point(2, 3), 5),
12     Rectangle(Point(2, 3), Point(3, 5)),
13     Triangle(Point(2, 3), Point(3, 5), Point(7, 4)),
14 ]
15
16 for s in shapes:  # Print shape classes, areas, and perimeters.
17     print(f"{type(s)} instance with A={s.area()} and P={s.perimeter()}"
```

↓ python3 shape_user.py ↓

```
1 <class 'circle.Circle'> instance with A=78.53981633974483 and P
2   ↪ =31.41592653589793
3 <class 'rectangle.Rectangle'> instance with A=2 and P=6
4 <class 'triangle.Triangle'> instance with A=4.5 and P=11.458193116710234
```



Beispiel: Die Subklassen von Shape benutzen

- Im Programm `shape_user.py` benutzen wir nun alle `Shape` abgeleiteten Klassen.
- Alle von ihnen unterstützen die Methoden `area` und `perimeter`.
- Wir deklarieren eine Variable `shapes`, die eine Liste mit je einer Instanz von jeder der Subklassen speichert.

```
1 """Examples for using the different :class:`Shape` classes."""
2
3 from circle import Circle      # Our new class `Circle`.
4 from point import Point        # Our very first class ever: `Point`.
5 from rectangle import Rectangle # Our new class `Rectangle`.
6 from triangle import Triangle  # Our new class `Triangle`.
7 from shape import Shape        # Our base class `Shape`.
8
9
10 shapes: list[Shape] = [    # We create list of shapes.
11     Circle(Point(2, 3), 5),
12     Rectangle(Point(2, 3), Point(3, 5)),
13     Triangle(Point(2, 3), Point(3, 5), Point(7, 4)),
14 ]
15
16 for s in shapes:  # Print shape classes, areas, and perimeters.
17     print(f"{type(s)} instance with A={s.area()} and P={s.perimeter()}"
```

↓ python3 shape_user.py ↓

```
1 <class 'circle.Circle'> instance with A=78.53981633974483 and P
2   ↪ =31.41592653589793
3 <class 'rectangle.Rectangle'> instance with A=2 and P=6
4 <class 'triangle.Triangle'> instance with A=4.5 and P=11.458193116710234
```



Beispiel: Die Subklassen von Shape benutzen

- Im Programm `shape_user.py` benutzen wir nun alle `Shape` abgeleiteten Klassen.
- Alle von ihnen unterstützen die Methoden `area` und `perimeter`.
- Wir deklarieren eine Variable `shapes`, die eine Liste mit je einer Instanz von jeder der Subklassen speichert.
- Wir können die Liste mit dem Type Hint `list[Shape]` annotieren.

```
1  """Examples for using the different :class:`Shape` classes."""
2
3  from circle import Circle      # Our new class `Circle`.
4  from point import Point        # Our very first class ever: `Point`.
5  from rectangle import Rectangle # Our new class `Rectangle`.
6  from triangle import Triangle  # Our new class `Triangle`.
7  from shape import Shape        # Our base class `Shape`.
8
9
10 shapes: list[Shape] = [    # We create list of shapes.
11     Circle(Point(2, 3), 5),
12     Rectangle(Point(2, 3), Point(3, 5)),
13     Triangle(Point(2, 3), Point(3, 5), Point(7, 4)),
14 ]
15
16 for s in shapes:  # Print shape classes, areas, and perimeters.
17     print(f"{type(s)} instance with A={s.area()} and P={s.perimeter()}")
↓ python3 shape_user.py ↓
1 <class 'circle.Circle'> instance with A=78.53981633974483 and P
2   ↪ =31.41592653589793
3 <class 'rectangle.Rectangle'> instance with A=2 and P=6
4 <class 'triangle.Triangle'> instance with A=4.5 and P=11.458193116710234
```



Beispiel: Die Subklassen von Shape benutzen

- Im Programm `shape_user.py` benutzen wir nun alle `Shape` abgeleiteten Klassen.
- Alle von ihnen unterstützen die Methoden `area` und `perimeter`.
- Wir deklarieren eine Variable `shapes`, die eine Liste mit je einer Instanz von jeder der Subklassen speichert.
- Wir können die Liste mit dem Type Hint `list[Shape]` annotieren.
- Der Type Hint sagt aus, dass die Liste nur Instanzen von `Shape` beinhalten kann.

```
1 """Examples for using the different :class:`Shape` classes."""
2
3 from circle import Circle      # Our new class `Circle`.
4 from point import Point        # Our very first class ever: `Point`.
5 from rectangle import Rectangle # Our new class `Rectangle`.
6 from triangle import Triangle  # Our new class `Triangle`.
7 from shape import Shape        # Our base class `Shape`.
8
9
10 shapes: list[Shape] = [    # We create list of shapes.
11     Circle(Point(2, 3), 5),
12     Rectangle(Point(2, 3), Point(3, 5)),
13     Triangle(Point(2, 3), Point(3, 5), Point(7, 4)),
14 ]
15
16 for s in shapes:  # Print shape classes, areas, and perimeters.
17     print(f"{type(s)} instance with A={s.area()} and P={s.perimeter()}")
↓ python3 shape_user.py ↓
1 <class 'circle.Circle'> instance with A=78.53981633974483 and P
2   ↪ =31.41592653589793
3 <class 'rectangle.Rectangle'> instance with A=2 and P=6
4 <class 'triangle.Triangle'> instance with A=4.5 and P=11.458193116710234
```

Beispiel: Die Subklassen von Shape benutzen



- Alle von ihnen unterstützen die Methoden `area` und `perimeter`.
- Wir deklarieren eine Variable `shapes`, die eine Liste mit je einer Instanz von jeder der Subklassen speichert.
- Wir können die Liste mit dem Type Hint `list[Shape]` annotieren.
- Der Type Hint sagt aus, dass die Liste nur Instanzen von `Shape` beinhalten kann.
- Da alle Instanzen von `Circle`, `Rectangle`, und `Triangle` auch Instanzen von `Shape` sind, funktioniert das gut.

```
1 """Examples for using the different :class:`Shape` classes."""
2
3 from circle import Circle      # Our new class `Circle`.
4 from point import Point        # Our very first class ever: `Point`.
5 from rectangle import Rectangle # Our new class `Rectangle`.
6 from triangle import Triangle  # Our new class `Triangle`.
7 from shape import Shape        # Our base class `Shape`.
8
9
10 shapes: list[Shape] = [    # We create list of shapes.
11     Circle(Point(2, 3), 5),
12     Rectangle(Point(2, 3), Point(3, 5)),
13     Triangle(Point(2, 3), Point(3, 5), Point(7, 4)),
14 ]
15
16 for s in shapes:  # Print shape classes, areas, and perimeters.
17     print(f"{type(s)} instance with A={s.area()} and P={s.perimeter()}")
↓ python3 shape_user.py ↓
1 <class 'circle.Circle'> instance with A=78.53981633974483 and P
2   ↪ =31.41592653589793
3 <class 'rectangle.Rectangle'> instance with A=2 and P=6
4 <class 'triangle.Triangle'> instance with A=4.5 and P=11.458193116710234
```



Beispiel: Die Subklassen von Shape benutzen

- Wir deklarieren eine Variable `shapes`, die eine Liste mit je einer Instanz von jeder der Subklassen speichert.
- Wir können die Liste mit dem Type Hint `list[Shape]` annotieren.
- Der Type Hint sagt aus, dass die Liste nur Instanzen von `Shape` beinhalten kann.
- Da alle Instanzen von `Circle`, `Rectangle`, und `Triangle` auch Instanzen von `Shape` sind, funktioniert das gut.
- Wir iterieren dann über die Liste und geben die Fläche und den Umfang jeder Form aus.

```
1 """Examples for using the different :class:`Shape` classes."""
2
3 from circle import Circle      # Our new class `Circle`.
4 from point import Point        # Our very first class ever: `Point`.
5 from rectangle import Rectangle # Our new class `Rectangle`.
6 from triangle import Triangle  # Our new class `Triangle`.
7 from shape import Shape        # Our base class `Shape`.
8
9
10 shapes: list[Shape] = [    # We create list of shapes.
11     Circle(Point(2, 3), 5),
12     Rectangle(Point(2, 3), Point(3, 5)),
13     Triangle(Point(2, 3), Point(3, 5), Point(7, 4)),
14 ]
15
16 for s in shapes:  # Print shape classes, areas, and perimeters.
17     print(f"{type(s)} instance with A={s.area()} and P={s.perimeter()}")
↓ python3 shape_user.py ↓
1 <class 'circle.Circle'> instance with A=78.53981633974483 and P
2   ↪ =31.41592653589793
3 <class 'rectangle.Rectangle'> instance with A=2 and P=6
4 <class 'triangle.Triangle'> instance with A=4.5 and P=11.458193116710234
```



Zusammenfassung



Zusammenfassung

- Wir haben nun diskutiert wie Vererbung mit Klassen in Python funktioniert.



Zusammenfassung

- Wir haben nun diskutiert wie Vererbung mit Klassen in Python funktioniert.
- Wir benutzten dieses Wissen, um eine Hierarchie von geometrischen Objekten in der zweidimensionalen Euklidischen Ebene zu konstruieren.



Zusammenfassung



- Wir haben nun diskutiert wie Vererbung mit Klassen in Python funktioniert.
- Wir benutzten dieses Wissen, um eine Hierarchie von geometrischen Objekten in der zweidimensionalen Euklidischen Ebene zu konstruieren.
- Während wir das gemacht haben, haben wir gesehen, wie Methoden auf eine abstrakte Art in einer Basisklasse definiert und dann in Subklassen implementiert werden können.

Zusammenfassung



- Wir haben nun diskutiert wie Vererbung mit Klassen in Python funktioniert.
- Wir benutzten dieses Wissen, um eine Hierarchie von geometrischen Objekten in der zweidimensionalen Euklidischen Ebene zu konstruieren.
- Während wir das gemacht haben, haben wir gesehen, wie Methoden auf eine abstrakte Art in einer Basisklasse definiert und dann in Subklassen implementiert werden können.
- Was wir hier in einer sehr abgekürzten Art gesehen haben ist, wie komplexe APIs definiert und implementiert werden können.

Zusammenfassung



- Wir haben nun diskutiert wie Vererbung mit Klassen in Python funktioniert.
- Wir benutzten dieses Wissen, um eine Hierarchie von geometrischen Objekten in der zweidimensionalen Euklidischen Ebene zu konstruieren.
- Während wir das gemacht haben, haben wir gesehen, wie Methoden auf eine abstrakte Art in einer Basisklasse definiert und dann in Subklassen implementiert werden können.
- Was wir hier in einer sehr abgekürzten Art gesehen haben ist, wie komplexe APIs definiert und implementiert werden können.
- Auf eine gewisse Art definiert die Klasse `Shape` ja eine API für geometrische Objekte.

Zusammenfassung



- Wir haben nun diskutiert wie Vererbung mit Klassen in Python funktioniert.
- Wir benutzten dieses Wissen, um eine Hierarchie von geometrischen Objekten in der zweidimensionalen Euklidischen Ebene zu konstruieren.
- Während wir das gemacht haben, haben wir gesehen, wie Methoden auf eine abstrakte Art in einer Basisklasse definiert und dann in Subklassen implementiert werden können.
- Was wir hier in einer sehr abgekürzten Art gesehen haben ist, wie komplexe APIs definiert und implementiert werden können.
- Auf eine gewisse Art definiert die Klasse `Shape` ja eine API für geometrische Objekte.
- Sie definiert dass jedes Objekt die beiden Operationen `area()` und `perimeter()` unterstützen muss.

Zusammenfassung



- Wir haben nun diskutiert wie Vererbung mit Klassen in Python funktioniert.
- Wir benutzten dieses Wissen, um eine Hierarchie von geometrischen Objekten in der zweidimensionalen Euklidischen Ebene zu konstruieren.
- Während wir das gemacht haben, haben wir gesehen, wie Methoden auf eine abstrakte Art in einer Basisklasse definiert und dann in Subklassen implementiert werden können.
- Was wir hier in einer sehr abgekürzten Art gesehen haben ist, wie komplexe APIs definiert und implementiert werden können.
- Auf eine gewisse Art definiert die Klasse `Shape` ja eine API für geometrische Objekte.
- Sie definiert dass jedes Objekt die beiden Operationen `area()` und `perimeter()` unterstützen muss.
- Die Klassen `Circle`, `Triangle`, und `Rectangle` haben diese API dann implementiert.

Zusammenfassung



- Wir haben nun diskutiert wie Vererbung mit Klassen in Python funktioniert.
- Wir benutzten dieses Wissen, um eine Hierarchie von geometrischen Objekten in der zweidimensionalen Euklidischen Ebene zu konstruieren.
- Während wir das gemacht haben, haben wir gesehen, wie Methoden auf eine abstrakte Art in einer Basisklasse definiert und dann in Subklassen implementiert werden können.
- Was wir hier in einer sehr abgekürzten Art gesehen haben ist, wie komplexe APIs definiert und implementiert werden können.
- Auf eine gewisse Art definiert die Klasse `Shape` ja eine API für geometrische Objekte.
- Sie definiert dass jedes Objekt die beiden Operationen `area()` und `perimeter()` unterstützen muss.
- Die Klassen `Circle`, `Triangle`, und `Rectangle` haben diese API dann implementiert.
- OK, das klingt etwas weit hergeholt.

Zusammenfassung



- Wir haben nun diskutiert wie Vererbung mit Klassen in Python funktioniert.
- Wir benutzten dieses Wissen, um eine Hierarchie von geometrischen Objekten in der zweidimensionalen Euklidischen Ebene zu konstruieren.
- Während wir das gemacht haben, haben wir gesehen, wie Methoden auf eine abstrakte Art in einer Basisklasse definiert und dann in Subklassen implementiert werden können.
- Was wir hier in einer sehr abgekürzten Art gesehen haben ist, wie komplexe APIs definiert und implementiert werden können.
- Auf eine gewisse Art definiert die Klasse `Shape` ja eine API für geometrische Objekte.
- Sie definiert dass jedes Objekt die beiden Operationen `area()` und `perimeter()` unterstützen muss.
- Die Klassen `Circle`, `Triangle`, und `Rectangle` haben diese API dann implementiert.
- OK, das klingt etwas weit hergeholt.
- Aber in der Realität könnte es schon so in etwa funktionieren.

Zusammenfassung



- Wir haben nun diskutiert wie Vererbung mit Klassen in Python funktioniert.
- Wir benutzten dieses Wissen, um eine Hierarchie von geometrischen Objekten in der zweidimensionalen Euklidischen Ebene zu konstruieren.
- Während wir das gemacht haben, haben wir gesehen, wie Methoden auf eine abstrakte Art in einer Basisklasse definiert und dann in Subklassen implementiert werden können.
- Was wir hier in einer sehr abgekürzten Art gesehen haben ist, wie komplexe APIs definiert und implementiert werden können.
- Auf eine gewisse Art definiert die Klasse `Shape` ja eine API für geometrische Objekte.
- Sie definiert dass jedes Objekt die beiden Operationen `area()` und `perimeter()` unterstützen muss.
- Die Klassen `Circle`, `Triangle`, und `Rectangle` haben diese API dann implementiert.
- OK, das klingt etwas weit hergeholt.
- Aber in der Realität könnte es schon so in etwa funktionieren.
- Stellen wir uns vor das wir eine API zum Erstellen von Grafiken durch Programme erstellen sollen.



Zusammenfassung

- Wir benutzten dieses Wissen, um eine Hierarchie von geometrischen Objekten in der zweidimensionalen Euklidischen Ebene zu konstruieren.
- Während wir das gemacht haben, haben wir gesehen, wir Methoden auf eine abstrakte Art in einer Basisklasse definiert und dann in Subklassen implementiert werden können.
- Was wir hier in einer sehr abgekürzten Art gesehen haben ist, wie komplexe APIs definiert und implementiert werden können.
- Auf eine gewisse Art definiert die Klasse `Shape` ja eine API für geometrische Objekte.
- Sie definiert dass jedes Objekt die beiden Operationen `area()` und `perimeter()` unterstützen muss.
- Die Klassen `Circle`, `Triangle`, und `Rectangle` haben diese API dann implementiert.
- OK, das klingt etwas weit hergeholt.
- Aber in der Realität könnte es schon so in etwa funktionieren.
- Stellen wir uns vor das wir eine API zum Erstellen von Grafiken durch Programme erstellen sollen.
- Ihre Objekte würden sich dann wie eine blanke Leinwand verhalten.

Zusammenfassung



- Was wir hier in einer sehr abgekürzten Art gesehen haben ist, wie komplexe APIs definiert und implementiert werden können.
- Auf eine gewisse Art definiert die Klasse `Shape` ja eine API für geometrische Objekte.
- Sie definiert dass jedes Objekt die beiden Operationen `area()` und `perimeter()` unterstützen muss.
- Die Klassen `Circle`, `Triangle`, und `Rectangle` haben diese API dann implementiert.
- OK, das klingt etwas weit hergeholt.
- Aber in der Realität könnte es schon so in etwa funktionieren.
- Stellen wir uns vor das wir eine API zum Erstellen von Grafiken durch Programme erstellen sollen.
- Ihre Objekte würden sich dann wie eine blanke Leinwand verhalten.
- Sie hätten wahrscheinlich eine Methode, um ein Rechteck mit bestimmten Koordinaten und einer bestimmten Farbe zu malen.



Zusammenfassung

- Auf eine gewisse Art definiert die Klasse `Shape` ja eine API für geometrische Objekte.
- Sie definiert dass jedes Objekt die beiden Operationen `area()` und `perimeter()` unterstützen muss.
- Die Klassen `Circle`, `Triangle`, und `Rectangle` haben diese API dann implementiert.
- OK, das klingt etwas weit hergeholt.
- Aber in der Realität könnte es schon so in etwa funktionieren.
- Stellen wir uns vor das wir eine API zum Erstellen von Grafiken durch Programme erstellen sollen.
- Ihre Objekte würden sich dann wie eine blanke Leinwand verhalten.
- Sie hätten wahrscheinlich eine Methode, um ein Rechteck mit bestimmten Koordinaten und einer bestimmten Farbe zu malen.
- Sie hätten wahrscheinlich eine Methode, um eine Linie mit bestimmten Koordinaten und einer bestimmten Farbe zu malen.



Zusammenfassung

- Auf eine gewisse Art definiert die Klasse `Shape` ja eine API für geometrische Objekte.
- Sie definiert dass jedes Objekt die beiden Operationen `area()` und `perimeter()` unterstützen muss.
- Die Klassen `Circle`, `Triangle`, und `Rectangle` haben diese API dann implementiert.
- OK, das klingt etwas weit hergeholt.
- Aber in der Realität könnte es schon so in etwa funktionieren.
- Stellen wir uns vor das wir eine API zum Erstellen von Grafiken durch Programme erstellen sollen.
- Ihre Objekte würden sich dann wie eine blanke Leinwand verhalten.
- Sie hätten wahrscheinlich eine Methode, um ein Rechteck mit bestimmten Koordinaten und einer bestimmten Farbe zu malen.
- Sie hätten wahrscheinlich eine Methode, um eine Linie mit bestimmten Koordinaten und einer bestimmten Farbe zu malen.
- Sie könnten das als abstrakte Basisklasse mit mindestens zwei Methoden, `draw_line` und `draw_rectangle`, definieren.

Zusammenfassung



- Sie definiert dass jedes Objekt die beiden Operationen `area()` und `perimeter()` unterstützen muss.
- Die Klassen `Circle`, `Triangle`, und `Rectangle` haben diese API dann implementiert.
- OK, das klingt etwas weit hergeholt.
- Aber in der Realität könnte es schon so in etwa funktionieren.
- Stellen wir uns vor das wir eine API zum Erstellen von Grafiken durch Programme erstellen sollen.
- Ihre Objekte würden sich dann wie eine blanke Leinwand verhalten.
- Sie hätten wahrscheinlich eine Methode, um ein Rechteck mit bestimmten Koordinaten und einer bestimmten Farbe zu malen.
- Sie hätten wahrscheinlich eine Methode, um eine Linie mit bestimmten Koordinaten und einer bestimmten Farbe zu malen.
- Sie könnten das als abstrakte Basisklasse mit mindestens zwei Methoden, `draw_line` und `draw_rectangle`, definieren.
- Sie könnten diese Methoden dann in einer Subklasse so implementieren, dass sie eine SVG-Grafik²³ im Speicher erstellen.



Zusammenfassung

- Die Klassen `Circle`, `Triangle`, und `Rectangle` haben diese API dann implementiert.
- OK, das klingt etwas weit hergeholt.
- Aber in der Realität könnte es schon so in etwa funktionieren.
- Stellen wir uns vor das wir eine API zum Erstellen von Grafiken durch Programme erstellen sollen.
- Ihre Objekte würden sich dann wie eine blanke Leinwand verhalten.
- Sie hätten wahrscheinlich eine Methode, um ein Rechteck mit bestimmten Koordinaten und einer bestimmten Farbe zu malen.
- Sie hätten wahrscheinlich eine Methode, um eine Linie mit bestimmten Koordinaten und einer bestimmten Farbe zu malen.
- Sie könnten das als abstrakte Basisklasse mit mindestens zwei Methoden, `draw_line` und `draw_rectangle`, definieren.
- Sie könnten diese Methoden dann in einer Subklasse so implementieren, dass sie eine SVG-Grafik²³ im Speicher erstellen.
- Eine andere Subklasse könnte stattdessen eine Grafik im Adobe PDF-Format^{37,100} erstellen.

Zusammenfassung



- Ihre Objekte würden sich dann wie eine blanke Leinwand verhalten.
- Sie hätten wahrscheinlich eine Methode, um ein Rechteck mit bestimmten Koordinaten und einer bestimmten Farbe zu malen.
- Sie hätten wahrscheinlich eine Methode, um eine Linie mit bestimmten Koordinaten und einer bestimmten Farbe zu malen.
- Sie könnten das als abstrakte Basisklasse mit mindestens zwei Methoden, `draw_line` und `draw_rectangle`, definieren.
- Sie könnten diese Methoden dann in einer Subklasse so implementieren, dass sie eine SVG-Grafik²³ im Speicher erstellen.
- Eine andere Subklasse könnte stattdessen eine Grafik im Adobe PDF-Format^{37,100} erstellen.
- So etwas zu machen ist viel komplizierter braucht viel mehr Platz als wie hier für ein vernünftiges Beispiel investieren können.



Zusammenfassung

- Sie hätten wahrscheinlich eine Methode, um ein Rechteck mit bestimmten Koordinaten und einer bestimmten Farbe zu malen.
- Sie hätten wahrscheinlich eine Methode, um eine Linie mit bestimmten Koordinaten und einer bestimmten Farbe zu malen.
- Sie könnten das als abstrakte Basisklasse mit mindestens zwei Methoden, `draw_line` und `draw_rectangle`, definieren.
- Sie könnten diese Methoden dann in einer Subklasse so implementieren, dass sie eine SVG-Grafik²³ im Speicher erstellen.
- Eine andere Subklasse könnte stattdessen eine Grafik im Adobe PDF-Format^{37,100} erstellen.
- So etwas zu machen ist viel komplizierter braucht viel mehr Platz als wie hier für ein vernünftiges Beispiel investieren können.
- Der prinzipielle Ansatz wäre aber wahrscheinlich nicht sehr anders, als was wir hier gelernt haben.



谢谢您们！
Thank you!
Vielen Dank!



References I



- [1] Adam Aspin und Karine Aspin. *Query Answers with MariaDB – Volume I: Introduction to SQL Queries*. Tetras Publishing, Okt. 2018. ISBN: 978-1-9996172-4-0. See also² (siehe S. 197, 210).
- [2] Adam Aspin und Karine Aspin. *Query Answers with MariaDB – Volume II: In-Depth Querying*. Tetras Publishing, Okt. 2018. ISBN: 978-1-9996172-5-7. See also¹ (siehe S. 197, 210).
- [3] Ivo Babuška. "Numerical Stability in Mathematical Analysis". In: *World Congress on Information Processing (IFIP'1968)*. Bd. 1: Mathematics, Software. 5. Aug. 1966–10. Aug. 1968, Edinburgh, Scotland, UK. Hrsg. von A.J.H. Morrell. Laxenburg, Austria: International Federation for Information Processing (IFIP). Amsterdam, The Netherlands: North-Holland Publishing Co., 1969, S. 11–23. ISBN: 978-0-7204-2032-6 (siehe S. 109–132).
- [4] Daniel J. Barrett. *Efficient Linux at the Command Line*. Sebastopol, CA, USA: O'Reilly Media, Inc., Feb. 2022. ISBN: 978-1-0981-1340-7 (siehe S. 210, 212).
- [5] Daniel Bartholomew. *Learning the MariaDB Ecosystem: Enterprise-level Features for Scalability and Availability*. New York, NY, USA: Apress Media, LLC, Okt. 2019. ISBN: 978-1-4842-5514-8 (siehe S. 210).
- [6] Kent L. Beck. *JUnit Pocket Guide*. Sebastopol, CA, USA: O'Reilly Media, Inc., Sep. 2004. ISBN: 978-0-596-00743-0 (siehe S. 213).
- [7] Tim Berners-Lee. *Re: Qualifiers on Hypertext links...* Geneva, Switzerland: World Wide Web project, European Organization for Nuclear Research (CERN) und Newsgroups: alt.hypertext, 6. Aug. 1991. URL: <https://www.w3.org/People/Berners-Lee/1991/08/art-6484.txt> (besucht am 2025-02-05) (siehe S. 213).
- [8] Alex Berson. *Client/Server Architecture*. 2. Aufl. Computer Communications Series. New York, NY, USA: McGraw-Hill, 29. März 1996. ISBN: 978-0-07-005664-0 (siehe S. 209).
- [9] Silvia Botros und Jeremy Tinley. *High Performance MySQL*. 4. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., Nov. 2021. ISBN: 978-1-4920-8051-0 (siehe S. 211).
- [10] Ed Bott. *Windows 11 Inside Out*. Hoboken, NJ, USA: Microsoft Press, Pearson Education, Inc., Feb. 2023. ISBN: 978-0-13-769132-6 (siehe S. 210).

References II



- [11] Ron Brash und Ganesh Naik. *Bash Cookbook*. Birmingham, England, UK: Packt Publishing Ltd, Juli 2018. ISBN: 978-1-78862-936-2 (siehe S. 209).
- [12] Tim Bray. *The JavaScript Object Notation (JSON) Data Interchange Format*. Request for Comments (RFC) 8259. Wilmington, DE, USA: Internet Engineering Task Force (IETF), Dez. 2017. URL: <https://www.ietf.org/rfc/rfc8259.txt> (besucht am 2025-02-05) (siehe S. 210).
- [13] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen und Eve Maler, Hrsg. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. W3C Recommendation. Wakefield, MA, USA: World Wide Web Consortium (W3C), 26. Nov. 2008–7. Feb. 2013. URL: <http://www.w3.org/TR/2008/REC-xml-20081126> (besucht am 2024-12-15) (siehe S. 213).
- [14] "Built-in Functions: class object". In: *Python 3 Documentation. The Python Standard Library*. Beaverton, OR, USA: Python Software Foundation (PSF), 2001–2025. URL: <https://docs.python.org/3/library/functions.html#object> (besucht am 2025-09-25) (siehe S. 86–100).
- [15] Brett Cannon, Jiwon Seo, Yury Selivanov und Larry Hastings. *Function Signature Object*. Python Enhancement Proposal (PEP) 362. Beaverton, OR, USA: Python Software Foundation (PSF), 21. Aug. 2006–4. Juni 2012. URL: <https://peps.python.org/pep-0362> (besucht am 2024-12-12) (siehe S. 212).
- [16] Jason Cannon. *High Availability for the LAMP Stack*. Shelter Island, NY, USA: Manning Publications, Juni 2022 (siehe S. 210, 211).
- [17] Donald D. Chamberlin. "50 Years of Queries". *Communications of the ACM (CACM)* 67(8):110–121, Aug. 2024. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/3649887. URL: <https://cacm.acm.org/research/50-years-of-queries> (besucht am 2025-01-09) (siehe S. 212).
- [18] David Clinton und Christopher Negus. *Ubuntu Linux Bible*. 10. Aufl. Bible Series. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd., 10. Nov. 2020. ISBN: 978-1-119-72233-5 (siehe S. 212).
- [19] Edgar Frank „Ted“ Codd. "A Relational Model of Data for Large Shared Data Banks". *Communications of the ACM (CACM)* 13(6):377–387, Juni 1970. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/362384.362685. URL: <https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf> (besucht am 2025-01-05) (siehe S. 211).



References III

- [20] Coding Gears und Train Your Brain. *YAML Fundamentals for DevOps, Cloud and IaC Engineers*. Birmingham, England, UK: Packt Publishing Ltd, März 2022. ISBN: 978-1-80324-243-9 (siehe S. 213).
- [21] Timothy W. Cole und Myung-Ja K. Han. *XML for Catalogers and Metadata Librarians (Third Millennium Cataloging)*. 1. Aufl. Dublin, OH, USA: Libraries Unlimited, 23. Mai 2013. ISBN: 978-1-59884-519-8 (siehe S. 213).
- [22] "CSV – CSV File Reading and Writing". In: *Python 3 Documentation. The Python Standard Library*. Beaverton, OR, USA: Python Software Foundation (PSF), 2001–2025. URL: <https://docs.python.org/3/library/csv.html> (besucht am 2024-11-14) (siehe S. 209).
- [23] Erik Dahlström, Patrick Dengler, Anthony Grasso, Chris Lilley, Cameron McCormack, Doug Schepers, Jonathan Watt, Jon Ferraiolo, Jun Fujisawa und Dean Jackson, Hrsg. *Scalable Vector Graphics (SVG) 1.1 (Second Edition)*. W3C Recommendation. Wakefield, MA, USA: World Wide Web Consortium (W3C), 16. Aug. 2011. URL: <http://www.w3.org/TR/2011/REC-SVG11-20110816> (besucht am 2024-12-17) (siehe S. 178–195, 212).
- [24] *Database Language SQL*. Techn. Ber. ANSI X3.135-1986. Washington, D.C., USA: American National Standards Institute (ANSI), 1986 (siehe S. 212).
- [25] Matt David und Blake Barnhill. *How to Teach People SQL*. San Francisco, CA, USA: The Data School, Chart.io, Inc., 10. Dez. 2019–10. Apr. 2023. URL: <https://dataschool.com/how-to-teach-people-sql> (besucht am 2025-02-27) (siehe S. 212).
- [26] *Database Language SQL*. International Standard ISO 9075-1987. Geneva, Switzerland: International Organization for Standardization (ISO), 1987 (siehe S. 212).
- [27] Paul Deitel, Harvey Deitel und Abbey Deitel. *Internet & World Wide Web: How to Program*. 5. Aufl. Hoboken, NJ, USA: Pearson Education, Inc., Nov. 2011. ISBN: 978-0-13-299045-5 (siehe S. 213).
- [28] Alfredo Deza und Noah Gift. *Testing In Python*. San Francisco, CA, USA: Pragmatic AI Labs, Feb. 2020. ISBN: 979-8-6169-6064-1 (siehe S. 211).
- [29] "Doctest – Test Interactive Python Examples". In: *Python 3 Documentation. The Python Standard Library*. Beaverton, OR, USA: Python Software Foundation (PSF), 2001–2025. URL: <https://docs.python.org/3/library/doctest.html> (besucht am 2024-11-07) (siehe S. 210).

References IV



- [30] Ingy döt Net, Tina Müller, Pantelis Antoniou, Eemeli Aro, Thomas Smith, Oren Ben-Kiki und Clark C. Evans. *YAML Ain't Markup Language (YAML™) version 1.2*. Revision 1.2.2. Seattle, WA, USA: YAML Language Development Team, 1. Okt. 2021. URL: <https://yaml.org/spec/1.2.2> (besucht am 2025-01-05) (siehe S. 213).
- [31] Russell J.T. Dyer. *Learning MySQL and MariaDB*. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2015. ISBN: 978-1-4493-6290-4 (siehe S. 210, 211).
- [32] *ECMAScript Language Specification*. Standard ECMA-262, 3rd Edition. Geneva, Switzerland: Ecma International, Dez. 1999. URL: https://ecma-international.org/wp-content/uploads/ECMA-262_3rd_edition_december_1999.pdf (besucht am 2024-12-15) (siehe S. 210).
- [33] Leonhard Euler. "An Essay on Continued Fractions". Übers. von Myra F. Wyman und Bostwick F. Wyman. *Mathematical Systems Theory* 18(1):295–328, Dez. 1985. New York, NY, USA: Springer Science+Business Media, LLC. ISSN: 1432-4350.
doi:10.1007/BF01699475. URL: <https://www.researchgate.net/publication/301720080> (besucht am 2024-09-24). Translation of³⁴. (Siehe S. 200).
- [34] Leonhard Euler. "De Fractionibus Continuis Dissertation". *Commentarii Academiae Scientiarum Petropolitanae* 9:98–137, 1737–1744. Petropolis (St. Petersburg), Russia: Typis Academiae. URL: <https://scholarlycommons.pacific.edu/cgi/viewcontent.cgi?article=1070> (besucht am 2024-09-24). See³³ for a translation. (Siehe S. 200, 214).
- [35] Luca Ferrari und Enrico Pirozzi. *Learn PostgreSQL*. 2. Aufl. Birmingham, England, UK: Packt Publishing Ltd, Okt. 2023. ISBN: 978-1-83763-564-1 (siehe S. 211).
- [36] Michael Filaseta. "The Transcendence of e and π ". In: *Math 785: Transcendental Number Theory*. Columbia, SC, USA: University of South Carolina, Frühling 2011. Kap. 6. URL: <https://people.math.sc.edu/filaseta/gradcourses/Math785/Math785Notes6.pdf> (besucht am 2024-07-05) (siehe S. 214).
- [37] *PDF 32000-1:2008 – Document Management – Portable Document Format – Part 1: PDF 1.7*. 1. Aufl. San Jose, CA, USA: Adobe Systems Incorporated, 1. Juli 2008. URL: https://pdf-lib.js.org/assets/with_large_page_count.pdf (besucht am 2024-12-12) (siehe S. 178–195, 211).

References V



- [38] David Goodger und Guido van Rossum. *Docstring Conventions*. Python Enhancement Proposal (PEP) 257. Beaverton, OR, USA: Python Software Foundation (PSF), 29. Mai–13. Juni 2001. URL: <https://peps.python.org/pep-0257> (besucht am 2024-07-27) (siehe S. 209).
- [39] Michael Goodwin. *What is an API?* Armonk, NY, USA: International Business Machines Corporation (IBM), 9. Apr. 2024. URL: <https://www.ibm.com/topics/api> (besucht am 2024-12-12) (siehe S. 209).
- [40] Terry Halpin und Tony Morgan. *Information Modeling and Relational Databases*. 3. Aufl. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, Juli 2024. ISBN: [978-0-443-23791-1](#) (siehe S. 211).
- [41] Jan L. Harrington. *Relational Database Design and Implementation*. 4. Aufl. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, Apr. 2016. ISBN: [978-0-12-849902-3](#) (siehe S. 211).
- [42] Michael Hausenblas. *Learning Modern Linux*. Sebastopol, CA, USA: O'Reilly Media, Inc., Apr. 2022. ISBN: [978-1-0981-0894-6](#) (siehe S. 210).
- [43] Christian Heimes. "`defusedxml` 0.7.1: XML Bomb Protection for Python stdlib Modules". In: 8. März 2021. URL: <https://pypi.org/project/defusedxml> (besucht am 2024-12-15) (siehe S. 213).
- [44] Matthew Helmke. *Ubuntu Linux Unleashed 2021 Edition*. 14. Aufl. Reading, MA, USA: Addison-Wesley Professional, Aug. 2020. ISBN: [978-0-13-668539-5](#) (siehe S. 210, 212).
- [45] 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](https://doi.org/10.1007/978-3-031-35122-8) (siehe S. 211).
- [46] *Information Technology – Database Languages – SQL – Part 1: Framework (SQL/Framework), Part 1*. International Standard ISO/IEC 9075-1:2023(E), Sixth Edition, (ANSI X3.135). Geneva, Switzerland: International Organization for Standardization (ISO) und International Electrotechnical Commission (IEC), Juni 2023. URL: [https://standards.iso.org/ittf/PubliclyAvailableStandards/ISO_IEC_9075-1_2023_ed_6_-_id_76583_Publication_PDF_\(en\).zip](https://standards.iso.org/ittf/PubliclyAvailableStandards/ISO_IEC_9075-1_2023_ed_6_-_id_76583_Publication_PDF_(en).zip) (besucht am 2025-01-08). Consists of several parts, see <https://modern-sql.com/standard> for information where to obtain them. (Siehe S. 212).

References VI



- [47] Arthur Jones, Kenneth R. Pearson und Sidney A. Morris. "Transcendence of e and π ". In: *Abstract Algebra and Famous Impossibilities*. Universitext (UTX). New York, NY, USA: Springer New York, 1991. Kap. 9, S. 115–161. ISSN: 0172-5939. ISBN: 978-1-4419-8552-1. doi:10.1007/978-1-4419-8552-1_8 (siehe S. 214).
- [48] William Kahan. "Pracniques: Further Remarks on Reducing Truncation Errors". *Communications of the ACM (CACM)* 8(1):40, Jan. 1965. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/363707.363723. URL: <https://www.convexoptimization.com/TOOLS/Kahan.pdf> (besucht am 2024-11-18) (siehe S. 109–132).
- [49] Katie Kodes. *Intro to XML, JSON, & YAML*. London, England, UK: Payhip, 2019–4. Sep. 2020 (siehe S. 213).
- [50] Holger Krekel und pytest-Dev Team. "How to Run Doctests". In: *pytest Documentation*. Release 8.4. Freiburg, Baden-Württemberg, Germany: merlinux GmbH. Kap. 2.8, S. 65–69. URL: <https://docs.pytest.org/en/stable/how-to/doctest.html> (besucht am 2024-11-07) (siehe S. 210).
- [51] Holger Krekel und pytest-Dev Team. *pytest Documentation*. Release 8.4. Freiburg, Baden-Württemberg, Germany: merlinux GmbH. URL: <https://readthedocs.org/projects/pytest/downloads/pdf/latest> (besucht am 2024-11-07) (siehe S. 211).
- [52] Jay LaCroix. *Mastering Ubuntu Server*. 4. Aufl. Birmingham, England, UK: Packt Publishing Ltd, Sep. 2022. ISBN: 978-1-80323-424-3 (siehe S. 211).
- [53] 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. 212).
- [54] 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. 211).
- [55] 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. 211).

References VII



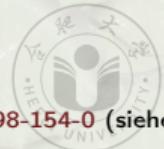
- [56] Gloria Lotha, Aakanksha Gaur, Erik Gregersen, Swati Chopra und William L. Hosch. "Client-Server Architecture". In: *Encyclopaedia Britannica*. Hrsg. von The Editors of Encyclopaedia Britannica. Chicago, IL, USA: Encyclopædia Britannica, Inc., 3. Jan. 2025. URL: <https://www.britannica.com/technology/client-server-architecture> (besucht am 2025-01-20) (siehe S. 209).
- [57] Mark Lutz. *Learning Python*. 6. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2025. ISBN: 978-1-0981-7130-8 (siehe S. 211).
- [58] *MariaDB Server Documentation*. Milpitas, CA, USA: MariaDB, 2025. URL: <https://mariadb.com/kb/en/documentation> (besucht am 2025-04-24) (siehe S. 210).
- [59] "Mathematical Functions and Operators". In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. 9.3. URL: <https://www.postgresql.org/docs/17/functions-math.html> (besucht am 2025-02-27) (siehe S. 214).
- [60] MDN Contributors. *Signature (Functions)*. San Francisco, CA, USA: Mozilla Corporation, 8. Juni 2023. URL: <https://developer.mozilla.org/en-US/docs/Glossary/Signature/Function> (besucht am 2024-12-12) (siehe S. 212).
- [61] Jim Melton und Alan R. Simon. *SQL: 1999 – Understanding Relational Language Components*. The Morgan Kaufmann Series in Data Management Systems. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, Juni 2001. ISBN: 978-1-55860-456-8 (siehe S. 212).
- [62] Arnold Neumaier. "Rundungsfehleranalyse einiger Verfahren zur Summation endlicher Summen". *ZAMM – Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik* 54(1):39–51, 1974. Weinheim, Baden-Württemberg, Germany: Wiley-VCH GmbH. ISSN: 0044-2267. doi:10.1002/zamm.19740540106. URL: <https://arnold-neumaier.at/scan/01.pdf> (besucht am 2024-11-18) (siehe S. 109–132).
- [63] Cameron Newham und Bill Rosenblatt. *Learning the Bash Shell – Unix Shell Programming: Covers Bash 3.0*. 3. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., 2005. ISBN: 978-0-596-00965-6 (siehe S. 209).
- [64] Ivan Niven. "The Transcendence of π ". *The American Mathematical Monthly* 46(8):469–471, Okt. 1939. London, England, UK: Taylor and Francis Ltd. ISSN: 1930-0972. doi:10.2307/2302515 (siehe S. 214).
- [65] Regina O. Obe und Leo S. Hsu. *PostgreSQL: Up and Running*. 3. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., Okt. 2017. ISBN: 978-1-4919-6336-4 (siehe S. 211).

References VIII



- [66] “`object`”. In: *Python 3 Documentation. Glossary*. Beaverton, OR, USA: Python Software Foundation (PSF), 2001–2025. URL: <https://docs.python.org/3/glossary.html#term-object> (besucht am 2025-09-25) (siehe S. 86–100).
- [67] A. Jefferson Offutt. “Unit Testing Versus Integration Testing”. In: *Test: Faster, Better, Sooner – IEEE International Test Conference (ITC'1991)*. 26.–30. Okt. 1991, Nashville, TN, USA. Los Alamitos, CA, USA: IEEE Computer Society, 1991. Kap. Paper P2.3, S. 1108–1109. ISSN: 1089-3539. ISBN: 978-0-8186-9156-0. doi:10.1109/TEST.1991.519784 (siehe S. 213).
- [68] Brian Okken. *Python Testing with pytest*. Flower Mound, TX, USA: Pragmatic Bookshelf by The Pragmatic Programmers, L.L.C., Feb. 2022. ISBN: 978-1-68050-860-4 (siehe S. 211).
- [69] Michael Olan. “Unit Testing: Test Early, Test Often”. *Journal of Computing Sciences in Colleges (JCSC)* 19(2):319–328, Dez. 2003. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 1937-4771. doi:10.5555/948785.948830. URL: <https://www.researchgate.net/publication/255673967> (besucht am 2025-09-05) (siehe S. 213).
- [70] Robert Orfali, Dan Harkey und Jeri Edwards. *Client/Server Survival Guide*. 3. Aufl. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd., 25. Jan. 1999. ISBN: 978-0-471-31615-2 (siehe S. 209).
- [71] Ashwin Pajankar. *Python Unit Test Automation: Automate, Organize, and Execute Unit Tests in Python*. New York, NY, USA: Apress Media, LLC, Dez. 2021. ISBN: 978-1-4842-7854-3 (siehe S. 211, 213).
- [72] 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. 210).
- [73] *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), Feb. 2025. URL: <https://www.postgresql.org/docs/17/index.html> (besucht am 2025-02-25).
- [74] *PostgreSQL Essentials: Leveling Up Your Data Work*. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2024 (siehe S. 211).
- [75] Abhishek Ratan, Eric Chou, Pradeeban Kathiravelu und Dr. M.O. Faruque Sarker. *Python Network Programming*. Birmingham, England, UK: Packt Publishing Ltd, Jan. 2019. ISBN: 978-1-78883-546-6 (siehe S. 209).

References IX



- [76] Federico Razzoli. *Mastering MariaDB*. Birmingham, England, UK: Packt Publishing Ltd, Sep. 2014. ISBN: 978-1-78398-154-0 (siehe S. 210).
- [77] Mike Reichardt, Michael Gundall und Hans D. Schotten. "Benchmarking the Operation Times of NoSQL and MySQL Databases for Python Clients". In: *47th Annual Conference of the IEEE Industrial Electronics Society (IECON'2021)*. 13.–15. Okt. 2021, Toronto, ON, Canada. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE), 2021, S. 1–8. ISSN: 2577-1647. ISBN: 978-1-6654-3554-3. doi:10.1109/IECON48115.2021.9589382 (siehe S. 211).
- [78] Mark Richards und Neal Ford. *Fundamentals of Software Architecture: An Engineering Approach*. Sebastopol, CA, USA: O'Reilly Media, Inc., Jan. 2020. ISBN: 978-1-4920-4345-4 (siehe S. 209).
- [79] Per Runeson. "A Survey of Unit Testing Practices". *IEEE Software* 23(4):22–29, Juli–Aug. 2006. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE). ISSN: 0740-7459. doi:10.1109/MS.2006.91 (siehe S. 213).
- [80] Stephen R. Schach. *Object-Oriented Software Engineering*. New York, NY, USA: McGraw-Hill, Sep. 2007. ISBN: 978-0-07-352333-0 (siehe S. 211).
- [81] Yakov Shafranovich. *Common Format and MIME Type for Comma-Separated Values (CSV) Files*. Request for Comments (RFC) 4180. Wilmington, DE, USA: Internet Engineering Task Force (IETF), Okt. 2005. URL: <https://www.ietf.org/rfc/rfc4180.txt> (besucht am 2025-02-05) (siehe S. 209).
- [82] Ellen Siever, Stephen Figgins, Robert Love und Arnold Robbins. *Linux in a Nutshell*. 6. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., Sep. 2009. ISBN: 978-0-596-15448-6 (siehe S. 210).
- [83] Anna Skoulikari. *Learning Git*. Sebastopol, CA, USA: O'Reilly Media, Inc., Mai 2023. ISBN: 978-1-0981-3391-7 (siehe S. 210).
- [84] John Miles Smith und Philip Yen-Tang Chang. "Optimizing the Performance of a Relational Algebra Database Interface". *Communications of the ACM (CACM)* 18(10):568–579, Okt. 1975. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/361020.361025 (siehe S. 211).
- [85] "SQL Commands". In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. Part VI. Reference. URL: <https://www.postgresql.org/docs/17/sql-commands.html> (besucht am 2025-02-25) (siehe S. 212).



References X

- [86] Ryan K. Stephens und Ronald R. Plew. *Sams Teach Yourself SQL in 21 Days*. 4. Aufl. Sams Tech Yourself. Indianapolis, IN, USA: SAMS Technical Publishing und Hoboken, NJ, USA: Pearson Education, Inc., Okt. 2002. ISBN: 978-0-672-32451-2 (siehe S. 206, 212).
- [87] Ryan K. Stephens, Ronald R. Plew, Bryan Morgan und Jeff Perkins. *SQL in 21 Tagen. Die Datenbank-Abfragesprache SQL vollständig erklärt (in 14/21 Tagen)*. 6. Aufl. Burghann, Bayern, Germany: Markt+Technik Verlag GmbH, Feb. 1998. ISBN: 978-3-8272-2020-2. Translation of⁸⁶ (siehe S. 212).
- [88] Allen Taylor. *Introducing SQL and Relational Databases*. New York, NY, USA: Apress Media, LLC, Sep. 2018. ISBN: 978-1-4842-3841-7 (siehe S. 211, 212).
- [89] Alkin Tezuysal und Ibrar Ahmed. *Database Design and Modeling with PostgreSQL and MySQL*. Birmingham, England, UK: Packt Publishing Ltd, Juli 2024. ISBN: 978-1-80323-347-5 (siehe S. 211).
- [90] *The JSON Data Interchange Syntax*. Standard ECMA-404, 2nd Edition. Geneva, Switzerland: Ecma International, Dez. 2017. URL: <https://ecma-international.org/publications-and-standards/standards/ecma-404> (besucht am 2024-12-15) (siehe S. 210).
- [91] *Python 3 Documentation. The Python Standard Library*. Beaverton, OR, USA: Python Software Foundation (PSF), 2001–2025. URL: <https://docs.python.org/3/library> (besucht am 2025-04-27).
- [92] George K. Thiruvathukal, Konstantin Läufer und Benjamin Gonzalez. "Unit Testing Considered Useful". *Computing in Science & Engineering* 8(6):76–87, Nov.–Dez. 2006. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE). ISSN: 1521-9615. doi:10.1109/MCSE.2006.124. URL: <https://www.researchgate.net/publication/220094077> (besucht am 2024-10-01) (siehe S. 213).
- [93] Linus Torvalds. "The Linux Edge". *Communications of the ACM (CACM)* 42(4):38–39, Apr. 1999. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/299157.299165 (siehe S. 210).
- [94] 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. 210, 213).



References XI

- [95] 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. 212).
- [96] 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. 209).
- [97] Sander van Vugt. *Linux Fundamentals*. 2. Aufl. Hoboken, NJ, USA: Pearson IT Certification, Juni 2022. ISBN: 978-0-13-792931-3 (siehe S. 210).
- [98] Thomas Weise (汤卫思). *Databases*. 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), 2025. URL: <https://thomasweise.github.io/databases> (besucht am 2025-01-05) (siehe S. 209, 211).
- [99] 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), 2024–2025. URL: <https://thomasweise.github.io/programmingWithPython> (besucht am 2025-01-05) (siehe S. 211).
- [100] *What does PDF mean?* San Jose, CA, USA: Adobe Systems Incorporated, 2024. URL: <https://www.adobe.com/acrobat/about-adobe-pdf.html> (besucht am 2024-12-12) (siehe S. 178–195, 211).
- [101] *What is a Relational Database?* Armonk, NY, USA: International Business Machines Corporation (IBM), 20. Okt. 2021–12. Dez. 2024. URL: <https://www.ibm.com/think/topics/relational-databases> (besucht am 2025-01-05) (siehe S. 211).
- [102] Ulf Michael „Monty“ Widenius, David Axmark und Uppsala, Sweden: MySQL AB. *MySQL Reference Manual – Documentation from the Source*. Sebastopol, CA, USA: O'Reilly Media, Inc., 9. Juli 2002. ISBN: 978-0-596-00265-7 (siehe S. 211).
- [103] Kevin Wilson. *Python Made Easy*. Birmingham, England, UK: Packt Publishing Ltd, Aug. 2024. ISBN: 978-1-83664-615-0 (siehe S. 211).

References XII



- [104] Kinza Yasar und Craig S. Mullins. *Definition: Database Management System (DBMS)*. Newton, MA, USA: TechTarget, Inc., Juni 2024. URL: <https://www.techtarget.com/searchdatamanagement/definition/database-management-system> (besucht am 2025-01-11) (siehe S. 209).
- [105] Giorgio Zarrelli. *Mastering Bash*. Birmingham, England, UK: Packt Publishing Ltd, Juni 2017. ISBN: 978-1-78439-687-9 (siehe S. 209).



Glossary (in English) I

API An *Application Programming Interface* is a set of rules or protocols that enables one software application or component to use or communicate with another³⁹.

Bash is the shell used under Ubuntu Linux, i.e., the program that „runs“ in the terminal and interprets your commands, allowing you to start and interact with other programs^{11,63,105}. Learn more at <https://www.gnu.org/software/bash>.

client In a client-server architecture, the client is a device or process that requests a service from the server. It initiates the communication with the server, sends a request, and receives the response with the result of the request. Typical examples for clients are web browsers in the internet as well as clients for database management systems (DBMSes), such as psql.

client-server architecture is a system design where a central server receives requests from one or multiple clients^{8,56,70,75,78}. These requests and responses are usually sent over network connections. A typical example for such a system is the World Wide Web (WWW), where web servers host websites and make them available to web browsers, the clients. Another typical example is the structure of database (DB) software, where a central server, the DBMS, offers access to the DB to the different clients. Here, the client can be some terminal software shipping with the DBMS, such as psql, or the different applications that access the DBs.

CSV *Comma-Separated Values* is a very common and simple text format for exchanging tabular or matrix data⁸¹. Each row in the text file represents one row in the table or matrix. The elements in the row are separated by a fixed delimiter, usually a comma („,“), sometimes a semicolon („;“). Python offers some out-of-the-box CSV support in the `csv` module²².

DB A *database* is an organized collection of structured information or data, typically stored electronically in a computer system. Databases are discussed in our book *Databases*⁹⁸.

DBMS A *database management system* is the software layer located between the user or application and the DB. The DBMS allows the user/application to create, read, write, update, delete, and otherwise manipulate the data in the DB¹⁰⁴.

docstring Docstrings are special string constants in Python that contain documentation for modules or functions³⁸. They must be delimited by `"""..."""`^{38,96}.

Glossary (in English) II



doctest *doctests* are unit tests in the form of small pieces of code in the docstrings that look like interactive Python sessions. The first line of a statement in such a Python snippet is indented with Python»> and the following lines by These snippets can be executed by modules like `doctest`²⁹ or tools such as `pytest`⁵⁰. Their output is compared to the text following the snippet in the docstring. If the output matches this text, the test succeeds. Otherwise it fails.

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^{83,94}. Learn more at <https://git-scm.com>.

GitHub is a website where software projects can be hosted and managed via the Git VCS^{72,94}. Learn more at <https://github.com>.

IT information technology

JavaScript JavaScript is the predominant programming language used in websites to develop interactive contents for display in browsers³².

JSON *JavaScript Object Notation* is a data interchange format^{12,90} based on JavaScript³² syntax.

LAMP Stack A system setup for web applications: Linux, Apache (a web server), MySQL, and the server-side scripting language PHP^{16,44}.

Linux is the leading open source operating system, i.e., a free alternative for Microsoft Windows^{4,42,82,93,97}. We recommend using it for this course, for software development, and for research. Learn more at <https://www.linux.org>. Its variant Ubuntu is particularly easy to use and install.

MariaDB An open source relational database management system that has forked off from MySQL^{1,2,5,31,58,76}. See <https://mariadb.org> for more information.

Microsoft Windows is a commercial proprietary operating system¹⁰. It is widely spread, but we recommend using a Linux variant such as Ubuntu for software development and for our course. Learn more at <https://www.microsoft.com/windows>.



Glossary (in English) III

Mypy is a static type checking tool for Python⁵⁵ that makes use of type hints. Learn more at <https://github.com/python/mypy> and in⁹⁹.

MySQL An open source relational database management system^{9,31,77,89,102}. MySQL is famous for its use in the LAMP Stack. See <https://www.mysql.com> for more information.

OOP Object-Oriented Programming⁸⁰

PDF The Portable Document Format^{37,100} is the format in which provide this book. It is the standard format for the exchange of documents in the internet.

PostgreSQL An open source object-relational DBMS^{35,65,74,89}. See <https://postgresql.org> for more information.

psql is the client program used to access the PostgreSQL DBMS server.

pytest is a framework for writing and executing unit tests in Python^{28,51,68,71,103}. Learn more at <https://pytest.org>.

Python The Python programming language^{45,54,57,99}, i.e., what you will learn about in our book⁹⁹. Learn more at <https://python.org>.

relational database A relational DB is a database that organizes data into rows (tuples, records) and columns (attributes), which collectively form tables (relations) where the data points are related to each other^{19,40,41,84,88,98,101}.

server In a client-server architecture, the server is a process that fulfills the requests of the clients. It usually waits for incoming communication carrying the requests from the clients. For each request, it takes the necessary actions, performs the required computations, and then sends a response with the result of the request. Typical examples for servers are web servers¹⁶ in the internet as well as DBMSes. It is also common to refer to the computer running the server processes as server as well, i.e., to call it the „server computer“⁵².

Glossary (in English) IV



- signature** The signature of a function refers to the parameters and their types, the return type, and the exceptions that the function can raise⁶⁰. In Python, the function `signature` of the module `inspect` provides some information about the signature of a function¹⁵.
- SQL** The *Structured Query Language* is basically a programming language for querying and manipulating relational databases^{17,24–26,46,61,85–88}. It is understood by many DBMSes. You find the Structured Query Language (SQL) commands supported by PostgreSQL in the reference⁸⁵.
- SVG** The Scalable Vector Graphics (SVG) format is an Extensible Markup Language (XML)-based format for vector graphics²³. Vector graphics are composed of geometric shapes like lines, rectangles, circles, and text. As opposed to raster / pixel graphics, they can be scaled seamlessly and without artifacts. They are stored losslessly.
- terminal** A terminal is a text-based window where you can enter commands and execute them^{4,18}. Knowing what a terminal is and how to use it is very essential in any programming- or system administration-related task. If you want to open a terminal under Microsoft Windows, you can Druck auf + , dann Schreiben von `cmd`, dann Druck auf . Under Ubuntu Linux, + + opens a terminal, which then runs a Bash shell inside.
- 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^{53,95}. 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.
- Ubuntu** is a variant of the open source operating system Linux^{18,44}. We recommend that you use this operating system to follow this class, for software development, and for research. Learn more at <https://ubuntu.com>. If you are in China, you can download it from <https://mirrors.ustc.edu.cn/ubuntu-releases>.



Glossary (in English) V

- unit test** Software development is centered around creating the program code of an application, library, or otherwise useful system. A *unit test* is an *additional* code fragment that is not part of that productive code. It exists to execute (a part of) the productive code in a certain scenario (e.g., with specific parameters), to observe the behavior of that code, and to compare whether this behavior meets the specification^{6,67,69,71,79,92}. If not, the unit test fails. The use of unit tests is at least threefold: First, they help us to detect errors in the code. Second, program code is usually not developed only once and, from then on, used without change indefinitely. Instead, programs are often updated, improved, extended, and maintained over a long time. Unit tests can help us to detect whether such changes in the program code, maybe after years, violate the specification or, maybe, cause another, depending, module of the program to violate its specification. Third, they are part of the documentation or even specification of a program.
- VCS** A *Version Control System* is a software which allows you to manage and preserve the historical development of your program code⁹⁴. 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.
- WWW** World Wide Web^{7,27}
- XML** The *Extensible Markup Language* is a text-based language for storing and transporting of data^{13,21,49}. It allows you to define elements in the form `<myElement myAttr="x">...text..</myElement>`. Different from comma-separated values (CSV), elements in XML can be hierarchically nested, like `<a><c>test</c>bla`, and thus easily represent tree structures. XML is one of most-used data interchange formats. To process XML in Python, use the `defusedxml` library⁴³, as it protects against several security issues.
- YAML** *YAML Ain't Markup Language™* is a human-friendly data serialization language for all programming languages^{20,30,49}. It is widely used for configuration files in the DevOps environment. See <https://yaml.org> for more information.



Glossary (in English) VI

- π is the ratio of the circumference U of a circle and its diameter d , i.e., $\pi = U/d$. $\pi \in \mathbb{R}$ is an irrational and transcendental number^{36,47,64}, which is approximately $\pi \approx 3.141\,592\,653\,589\,793\,238\,462\,643$. In Python, it is provided by the `math` module as constant `pi` with value `3.141592653589793`. In PostgreSQL, it is provided by the SQL function `pi()` with value `3.141592653589793`⁵⁹.
- e is Euler's number³⁴, the base of the natural logarithm. $e \in \mathbb{R}$ is an irrational and transcendental number^{36,47}, which is approximately $e \approx 2.718\,281\,828\,459\,045\,235\,360$. In Python, it is provided by the `math` module as constant `e` with value `2.718281828459045`. In PostgreSQL, you can obtain it via the SQL function `exp(1)` as value `2.718281828459045`⁵⁹.
- \mathbb{R} the set of the real numbers.